A Recursive Enumeration of the Inequivalent Binary $(2k, k)$ Self-Dual Codes, for $2k \leq 32$

by

Richard T. Bilous

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba, Canada, 1998

©Richard T. Bilous 1998

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-32056-1

Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

A RECURSIVE ENUMERATION OF THE INEQUIVALENT BINARY ($2\underline{k}$, $\underline{k}$) SELF-DUAL CODES, FOR $2\underline{k} \leq 32$

BY

RICHARD T. BILOUS

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

RICHARD T. BILOUS ©1998

Permission has been granted to the Library of The University of Manitoba to lend or sell copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and to Dissertations Abstracts International to publish an abstract of this thesis/practicum.

The author reserves other publication rights, and neither this thesis/practicum nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

Abstract

A binary self-dual code of length $2k$ is a $(2k, k)$ binary linear code C with the property that every pair of codewords in C are orthogonal. Two binary self-dual codes of equal length, C_1 and C_2, are said to be equivalent if and only if there exists a permutation of the coordinates of C_1 that takes C_1 into C_2. If C_1 and C_2 are not equivalent then C_1 and C_2 are said to be inequivalent. The automorphism group of a binary linear code C is the set of all permutations of the coordinates of C that takes C into itself.

The main topic in this thesis is the enumeration of lists of inequivalent binary self-dual codes. We have developed algorithms that have allowed us to enumerate lists of inequivalent binary self-dual codes of lengths up to and including 32. This is the first time a list of inequivalent binary-self dual codes of length 32 has been enumerated. Our algorithms also find the size of the automorphism group of each code.

Contents

1 Introduction **1**

1.1 Definitions and Properties 1

1.1.1 Linear Codes 2

1.1.2 Binary Self-Dual Codes 8

1.2 The Problem 11

1.3 History 13

1.4 Approach 15

1.5 What's Ahead 16

2 Generator Matrices of the form $G = [I_k|A]$ **19**

2.1 Binary Self-Dual Codes Generated By $G = [I_k|A]$ 20

2.2 Permutation Conventions and Notation 24

2.3 The Three Operations on A 27

2.3.1 Swapping Two Columns of I_k 28

2.3.2 Swapping Two Columns of A 29

2.3.3 Swapping a Column of I_k with a Column of A 30

2.4 Equivalent Codes Generated by Matrices of the form $G = [I_k|A]$ 33

2.5 Weight d Words in $G = [I_k|A]$ 39

2.6 Summary 41

3 Ordering The Matrix A **43**

3.1 Ordering the Rows and Columns of A 44

3.2 Minimizing The Row Weights of A 73

3.3 Summary 78

4 A Recursive Enumeration of the $(2k, k, 4)$ Self-Dual Codes **80**

4.1 Enumerating the $(2k, k, 4)$ Self-Dual Codes 81

4.2 Weight 4 Blocks and Weight 4 Sets 94

4.2.1 The Weight 4 Blocks of $G = [I_k|A]$ 99

4.2.2 Generators For Weight 4 Sets of C 102

4.2.3 A List of All the Generator Blocks for Weight 4 Sets that may occur in $G = [I_k|A]$ 121

4.3 Classifying the $(2k, k, 4)$ Self-Dual Codes 131

4.4 Revised Enumeration of the $(2k, k, 4)$ Self-Dual Codes 133

4.4.1 Enumeration of the e_{2i}-codes, $i \geq 2$ 134

4.4.2 Enumeration of the d_i-codes, $i \geq 1$ 137

4.4.3 Enumeration of the e_3-codes 143

4.5 Further Improvements 149

4.5.1 Equivalent Extensions of Generator Blocks 155

4.5.2 Minimal Blocks for Extensions of Generator Blocks 158

4.6 Results of Enumeration 166

5 A Recursive Enumeration of the $(2k, k, d \geq 6)$ Self-Dual Codes **170**

5.1 The Three Codes Under a Weight 4 Word In A Self-Dual Code 171

5.2 An Extension of the $(2k, k, d \geq 6)$ Self-Dual Codes 186

5.3 Enumeration of the $(2k, k, d \geq 6)$ Self-Dual Codes 197

5.3.1 Equivalent Extensions of The d_1-Codes 198

5.3.2 Extensions of the d_1-Codes 203

5.3.3 Finding The Two Related Codes 210

5.3.4 Enumeration of the $(2k, k, d \geq 6)$ self-dual codes 212

5.4 Further Improvements 214

5.5 Results of Enumeration 241

6 Finding A Unique Representative For Each Equivalence Class of $(2k, k)$ Self-Dual Codes **243**

6.1 Definitions, Concepts, and Notation 245

6.2 Finding a Unique Representative for any Equivalence Class of Self-Dual Codes 249

6.3 Enumerating the Subset of Generator Matrices with the form $G = [I_k|A]$ in $\mathcal{G}(C)$ 253

6.3.1 A Two Stage Method for Performing the Permutations in Π_{2k} . . . 254

6.3.2 Enumerating the Set $\mathcal{G}_{[I|A]}(C)$ 257

6.3.3 Using the $k \times k$ Matrix A to Enumerate $\mathcal{G}_{[I|A]}(C)$ 266

6.3.4 Concluding Remarks 270

6.4 A General Unique Representative Algorithm 274

6.4.1 The Combination Algorithm 274

6.4.2 The Permutation Algorithm 281

6.4.3 The Unique Representative Algorithm 287

6.5 The Decomposable, Distance 2, and e_{2i}-Codes 288

6.5.1 The Decomposable and Distance 2 Self-Dual Codes 289

6.5.2 The e_{2i}-Codes 290

6.6 Generator Blocks with Weight 4 Rows 295

6.6.1 Generator Blocks for Weight 4 Sets 295

6.6.2 Generator Blocks with Weight 4 Rows in the Combination Algorithm 300

6.7 Generator Blocks with Weight 4 Rows In the Combination Algorithm . . 320

6.7.1 Maximal Combinations of Generator Blocks With Weight 4 Rows . 322

6.7.2 The Combination Algorithm for the e_3 and d_i-Codes 328

6.8 Weight 4 Blocks In the Permutation Algorithm 338

6.8.1 The Weight 3 Blocks in A 339

6.8.2 The Permutation Algorithm for the e_3 and d_i-Codes 341

6.9 A Unique Representative Algorithm for the Distance ≥ 6 Codes 351

6.9.1 The $f_{d-1,i}$-Blocks 352

6.9.2 $f_{d-1,i}$-Blocks and the Unique Representative Algorithm 359

6.9.3 The Combination Algorithm for the $(2k, k, d \geq 6)$ Self-Dual Codes 373

6.9.4 The Permutation Algorithm for the $(2k, k, d \geq 6)$ Self-Dual Codes 393

6.9.5 Combining our Unique Representative Algorithms 399

6.10 Using the Sizes of Automorphism Groups to Check Our Results 407

6.11 Results 410

7 Final Results **415**

7.1 The Number of Inequivalent $(2k, k)$ Self-Dual Codes, $2k \leq 32$ 415

7.2 The Weight Distribution and Size of the Automorphism Group of each $(2k, k, d \geq 4)$ Self-Dual Code, $2k \leq 32$ 418

Chapter 1

Introduction

In this chapter, we will discuss the basic problem we will consider in this thesis: the enumeration of inequivalent $(2k, k)$ self-dual codes.

In Section 1.1, we will give some of the basic coding theory definitions and some of the basic properties of self-dual codes that we will use throughout this thesis. In Section 1.2, we will define the main problem that we will consider in this thesis: the enumeration of a list of inequivalent $(2k, k)$ self-dual codes. In Section 1.3, we will discuss the work that has already been done on this problem. In Section 1.4, we will discuss our approach to this problem. Finally, in Section 1.5, we will give a brief description of each the remaining chapters in the thesis.

1.1 Definitions and Properties

In this section we will give some of the basic coding theory definitions and some of the basic properties of both linear codes and the subclass of linear codes we will consider in this thesis: the binary self-dual codes. For a more detailed discussion on linear codes see [1].

1.1.1 Linear Codes

We will begin by giving some of the basic definitions and properties of linear codes.

Let $GF(m)$ denote the field of m elements. Let $V_n(m)$ denote the vector space of all n–vectors over the field $GF(m)$. An (n,k) *linear code* over $GF(m)$ is a k dimensional subspace C of $V_n(m)$.

Example 1.1 Consider the following set of 6–vectors over $GF(2)$:

$$C = \left\{ \begin{array}{llll} (0,0,0,0,0,0), & (0,0,0,0,1,1), & (0,0,1,1,0,0), & (1,1,0,0,0,0), \\ (0,0,1,1,1,1), & (1,1,0,0,1,1), & (1,1,1,1,0,0), & (1,1,1,1,1,1) \end{array} \right\}$$

Notice that the zero vector is in C. Furthermore, if $\vec{v_1}$ and $\vec{v_2}$ are any pair of vectors in C then $\vec{v_1} + \vec{v_2}$ is also in C. Therefore, C is a subspace of $V_6(2)$. The vectors $(0,0,0,0,1,1)$, $(0,0,1,1,0,0)$, and $(1,1,0,0,0,0)$ form a basis for C. Therefore, C has dimension 3. Thus, C is a $(6,3)$ linear code over $GF(2)$.

In this thesis we will only consider the (n,k) linear codes over $GF(2)$. Unless it is explicitly stated that a linear code is over $GF(m)$, whenever we use the term *linear code* in this thesis we are referring to the linear codes over $GF(2)$ only. Note that the addition and multiplication operators for this field are addition modulo 2 and multiplication modulo 2.

Let C be an (n,k) linear code over $GF(2)$. The integer n is referred to as the *length* of C. The integer k is referred to as the *dimension* of C. The vectors in C are called *codewords*. We will denote the codewords of C as binary strings of length n. We will use the phrase "the coordinates of C" to refer to the coordinates of the codewords in C. The *weight* of a codeword $\vec{c}$ in C is the number of non-zero components in $\vec{c}$. We will denote the weight of a codeword $\vec{c}$ by $w(\vec{c})$. The *weight distribution* of C is a count of the number of weight w words in C, for $w = 0, 1, \ldots, n$. We will denote the weight distribution of C by the sequence $(A_0, A_1, \ldots, A_n)$, where A_i is the number of weight i

words in C, $0 \leq i \leq n$. The *distance* of C is the smallest non-zero weight of any codeword in C. An (n, k, d) *linear code* is an (n, k) linear code with distance d.

Example 1.2 Let C be the (6,3) linear code of example 1.1. C consists of exactly 8 codewords:

$$\begin{array}{cccccccc} \vec{c_1} & = & 000000 & \vec{c_2} & = & 000011 & \vec{c_3} & = & 001100 & \vec{c_4} & = & 110000 \\ \vec{c_5} & = & 001111 & \vec{c_6} & = & 110011 & \vec{c_7} & = & 111100 & \vec{c_8} & = & 111111 \end{array}$$

The weight of $\vec{c_1}$ is 0. Each of $\vec{c_2}$, $\vec{c_3}$, and $\vec{c_4}$ have weight 2. Each of $\vec{c_5}$, $\vec{c_6}$, and $\vec{c_7}$ have weight 4. The weight of $\vec{c_8}$ is 6. The weight distribution of C is $(1, 0, 3, 0, 3, 0, 1)$. The distance of C is 2.

Let C be an (n, k) linear code over $GF(2)$. A *generator matrix* for C is a $k \times n$ matrix G whose rows form a basis for C. If G is a generator matrix for C then G is said to *generate* C. We will generally represent linear codes with generator matrices.

Example 1.3 Consider the $(6, 3)$ linear code C over $GF(2)$ of example 1.1. The vectors 000011, 001100, and 110000 form a basis for C. Therefore,

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

is a generator matrix for C.

Let G be a generator matrix for the (n, k) linear code C. Then any $k \times n$ matrix G' whose rows are a permutation of the rows of G is also a generator matrix for C. Furthermore, since there may exist more than one basis for C, there may exist generator matrices G'' for C in which the rows of G'' are not simply permutations of the rows of G. In other words, there may exist many $k \times n$ matrices that generate the (n, k) linear code C.

Example 1.4 Consider the $(6,3)$ linear code C over $GF(2)$ of Example 1.1. Since the vectors 000011, 001100, and 110000 form a basis for C, the following two matrices are both generator matrices for C:

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

Since 001111, 111100, and 111111 also form a basis for C, the matrix

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

is also a generator matrix for C.

Let G and G' be any two generator matrices for the (n,k) linear code C. Then G' can be produced from G by applying an appropriate sequence of the three *elementary row-operations* to G. The three elementary row operations are: swap two rows, multiply a row by a non-zero scalar, and replace a row by the sum of itself with the scalar multiple of another row. If we perform any sequence of the three elementary row-operations on G then the matrix that results is also a generator matrix for C.

If G is a $k \times n$ matrix over $GF(2)$, the elementary row-operation "multiply a row by a non-zero scalar" has no effect on G. Therefore, if G is a $k \times n$ matrix over $GF(2)$, then there are in fact only two elementary row-operations defined on G: swap two rows, and replace a row by the sum of itself with another row.

Now, given any generator matrix G' for the linear code C, by applying an appropriate sequence of elementary row-operations on G', we can produce a unique generator matrix G for C, in which G is in *row-reduced echelon form.* A $k \times n$ matrix G over $GF(2)$ is in row-reduced echelon form if and only if for each row i in G, where $1 \leq i \leq k$, if j is the first non-zero column in row i then:

- all other rows of G have a value of 0 in column j, and
- the first $j-1$ columns of rows $i+1, i+2, \ldots, k$ all have a value of 0.

Given any $k \times n$ matrix G' over $GF(2)$ there always exists a sequence of elementary row-operations on G' that results in a matrix G that is in row-reduced echelon form. The matrix G is called the *row-reduced echelon form of* G'. The row-reduced echelon of G' is unique. Furthermore, if G' and G'' are any two generator matrices for a linear code C, then the row-reduced echelon forms of G' and G'' are equal. In other words, for a given (n,k) linear code C there exists one and only one $k \times n$ matrix G such that G is a generator matrix for C, and, G is in row-reduced echelon form. For a given linear code C, we will generally represent C with the unique generator matrix G for C that is in row-reduced echelon form.

Example 1.5 The matrix G' of Example 1.4 is the row-reduced echelon of G. We will use this matrix consistently to represent the code C of this example.

Let C and C' be any two (n,k) linear codes over $GF(2)$. Let G and G' be the unique generator matrices for C and C', respectively, that are in row-reduced echelon form. C and C' are said to be *equivalent* if and only if there exists a permutation of coordinates that sends one code into the other. In terms of the generator matrices G and G', it is easy to show that the linear codes C and C' are equivalent if and only if there exists a permutation π of the columns of G such that the row-reduced echelon form of $\pi(G)$ is equal to G'. If C and C' are not equivalent then C and C' are said to be *inequivalent.*

Example 1.6 Let C, C', and C'' be the $(6,3)$ linear codes over $GF(2)$ generated by G, G', and G'', respectively, where:

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

G, G', and G'' are all in row-reduced echelon form. Let $\pi = (2\ 4)(3\ 5)$ be a permutation of the columns of G'. The row-reduced echelon form of $\pi(G')$ is G. Therefore, C and C' are equivalent. However, there does not exist a permutation π of the columns of G'' in which the row-reduced echelon form of $\pi(G'')$ is equal to G. Therefore, C and C'' are inequivalent.

Let C, C', and C'' be any three (n, k) linear codes. It is easy to show that equivalence of codes is an equivalent relation. That is:

- C is equivalent C,
- If C is equivalent to C' then C' is equivalent to C, and
- If C is equivalent to C' and C' is equivalent to C'', then C is equivalent to C''.

Thererfore, the set S of all (n, k) linear codes over $GF(2)$ can be uniquely partitioned into a set of disjoint sets $S_1, S_2, \ldots, S_x$ (where x is some positive integer) such that for any pair of linear codes $C, C' \in S$, C and C' are elements of the same set S_i if and only if C and C' are equivalent. Each set S_i is referred to as an *equivalence class* of (n, k) linear codes over $GF(2)$. Given any $C \in S$, the equivalence class that C belongs to is referred to as the *equivalence class of the* (n, k) *linear code* C.

Let C be an (n, k) linear code. The *automorphism group* of C is the set of all permutations π of the coordinates of C that take C into itself. We can also define the automorphism group of C in terms of the unique generator matrix G of C that is in row-reduced echelon form. In this case, the automorphism group is the set of all permutations π of the columns G such that the row-reduced echelon form of $\pi(G)$ is equal to G.

Let C be any (n, k) linear code over $GF(2)$. Let G be the unique generator matrix for C that is in row-reduced echelon form. Since G is in row-reduced echelon form, and since the k rows of G form a basis for the k dimensional subspace C, G must contain the

following set of k columns:

$$\left\{ \begin{array}{ccccc} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{array} \right\}$$

Therefore, there exists a permutation of the columns of G that results in a matrix G' (that generates a code equivalent to C) in which G' has the form $[I_k|A']$, where I_k is a $k \times k$ identity matrix and A' is a $k \times (n-k)$ matrix over $GF(2)$. In other words, for any (n,k) linear code C over $GF(2)$, there exists at least one (n,k) linear code C' over $GF(2)$ in which C' is equivalent to C, and, C' is generated by a matrix of the form $G' = [I_k|A']$, where I_k is the $k \times k$ identity matrix.

Example 1.7 Let C be the $(8,4)$ linear code over $GF(2)$ that is generated by:

$$G = \left[\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array} \right]$$

G is in row-reduced echelon form. Let $\pi = (4\ 5)$ be a permutation of the columns of G. Then the matrix $G' = \pi(G)$, which happens to be in row-reduced echelon form, is:

$$G' = \left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array} \right]$$

The matrix G' has the form $[I_4|A']$ and generates an $(8,4)$ linear code C' that is equivalent to C.

Let C_1 be an (n_1, k_1) linear code over $GF(2)$. Let C_2 be an (n_2, k_2) linear code over $GF(2)$. The *direct sum* of C_1 and C_2, written as $C_1 \oplus C_2$, is the $(n_1 + n_2, k_1 + k_2)$ linear

code over $GF(2)$ that consists of the set of all codewords $\vec{c_1}\vec{c_2}$, where $\vec{c_1}$ is a codeword of C_1, $\vec{c_2}$ is a codeword of C_2, and $\vec{c_1}\vec{c_2}$ is the $(n_1 + n_2)$–vector that is the concatenation of $\vec{c_1}$ and $\vec{c_2}$.

Let G_1 be a generator matrix for an (n_1, k_1) linear code C_1 and let G_2 be a generator matrix for an (n_2, k_2) linear code C_2. The *composition* of G_1 and G_2, written $G_1 \oplus G_2$, is the $(k_1 + k_2) \times (n_1 + n_2)$ matrix:

$$G_1 \oplus G_2 = \begin{bmatrix} G_1 & 0 \\ 0 & G_2 \end{bmatrix}$$

$G_1 \oplus G_2$ is a generator matrix for the linear code $C_1 \oplus C_2$.

Let C be an (n, k) linear code over $GF(2)$. C is *decomposable* if and only if C is equivalent to an (n, k) linear code C' that is the direct sum of two codes C_1 and C_2, where C_1 is an (n_1, k_1) linear code, C_2 is an (n_2, k_2) linear code, $n = n_1 + n_2$, and $k = k_1 + k_2$. If C is not decomposable then C is said to be *indecomposable.*

1.1.2 Binary Self-Dual Codes

In this thesis we will only consider a subclass of the linear codes over $GF(2)$ known as the *self-dual codes over* $GF(2)$, or, more simply put, the *binary self-dual codes.* Therefore, we will conclude this section by defining what a binary self-dual code is and giving some of the basic properties of binary self-dual codes.

Let C be an (n, k) linear code over $GF(2)$. Let $\vec{x} = x_1 x_2 \ldots x_n$ and $\vec{y} = y_1 y_2 \ldots y_n$ be any two codewords in C. The codewords $\vec{x}$ and $\vec{y}$ are *orthogonal* if and only if $\vec{x} \bullet \vec{y} = \sum_{i=1}^{n} x_i y_i = 0$. In other words (since C is a subspace of $V_n(2)$) $\vec{x}$ and $\vec{y}$ are orthogonal if and only if they *intersect* in an even number of positions.

Let C be an (n, k) linear code over $GF(2)$. The *orthogonal complement* of C, written $C^{\perp}$, is the set of all n–vectors $\vec{x}$ in $V_n(2)$ with the property that for any codeword $\vec{y}$ in C, $\vec{x}$ and $\vec{y}$ are orthogonal. The orthogonal complement $C^{\perp}$ is an $(n, n - k)$ linear code

over $GF(2)$. Furthermore, if C is generated by a matrix of the form $G = [I_k|A]$ then $C^{\perp}$ is generated by $[A^T|I_{n-k}]$, where A_T is the transpose of the matrix A.

Let C be an (n,k) linear code over $GF(2)$ and let $C^{\perp}$ be the orthogonal complement of C. C is a *self-orthogonal code* if and only if $C \subseteq C^{\perp}$. C is a *self-dual code* if and only if $C = C^{\perp}$. Since $C^{\perp}$ is an $(n, n-k)$ code, if C is a self-dual code then we must have $k = n - k$, which implies $n = 2k$. Also note, if C is a self-dual code then since every codeword in C must be orthogonal to itself, every codeword in C has even weight.

Example 1.8 Let C be the $(8,4)$ linear code over $GF(2)$ that is generated by the matrix:

$$G = [I_4|A] = \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

The orthogonal complement $C^{\perp}$ of C is generated by the matrix:

$$G' = [A^T|I_4] = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

The row-reduced echelon form of G' is G. Therefore, G and G' generate the same code. Thus, $C = C^{\perp}$ and C is a self-dual code.

Theorem 1.9 Let C be a $(2k,k)$ linear code over $GF(2)$. Then C is a self-dual code if and only if for all pairs of codewords $\vec{x}, \vec{y} \in C$ (including $\vec{x} = \vec{y}$), we have $\vec{x} \bullet \vec{y} = 0$.

Proof Suppose C is a self-dual code. Then $C = C^{\perp}$, and by the definition of $C^{\perp}$, we have $\vec{x} \bullet \vec{y} = 0$, for all $\vec{x}, \vec{y} \in C$.

Suppose for all $\vec{x}, \vec{y} \in C$, we have $\vec{x} \bullet \vec{y} = 0$. Then, by the definition of $C^{\perp}$, we have $C \subseteq C^{\perp}$. Furthermore, since C has dimension k and $C^{\perp}$ has dimension $2k - k = k$, we have $C = C^{\perp}$. □

Example 1.10 Consider the $(8, 4)$ linear code C of example 1.8. The codewords in C are:

00000000	10001110	01001101	11000011
00101011	10100101	01100110	11101000
00010111	10011001	01011010	00111100
11010100	10110010	01110001	11111111

For any pair of codewords $\vec{x}, \vec{y} \in C$ (including $\vec{x} = \vec{y}$), we have $\vec{x} \bullet \vec{y} = 0$. Therefore, C is a self-dual code.

Notice that the self-dual code in Example 1.10 contains the all–ones vector. This is in fact true for all self-dual codes as the following theorem demonstrates.

Theorem 1.11 Let C be a $(2k, k)$ self-dual code over $GF(2)$. Then the all–ones vector is an element of C.

Proof Since C is a self-dual code, every codeword in C has even weight. Since C has length $2k$, the all–ones vector $\vec{1}$ has even weight. Therefore, for any $\vec{c} \in C$, we have $\vec{c} \bullet \vec{1} = 0$. Thus, by the definition of $C^{\perp}$, we have $\vec{1} \in C^{\perp}$, which implies $\vec{1} \in C$ □

Note, by Theorem 1.11, if C is a $(2k, k)$ self-dual code then the number of weight i words in C is equal to the number of weight $2k - i$ words in C. The reason for this is that for any weight i word $\vec{c}$ in C, the weight $2k - i$ word $\vec{c} + \vec{1}$ is also in C (and vice-versa), where $\vec{1}$ is the all–ones vector.

An important subclass of the binary self-dual codes that we will occasionally mention in this thesis is the *doubly-even* self-dual codes. Let C be a $(2k, k)$ self-dual code. Then C is *doubly-even* if and only if every codeword in C has weight that is divisible by 4.

We will conclude this section by counting the total number of $(2k, k)$ self-dual codes over $GF(2)$.

Theorem 1.12 The number of binary self-dual codes of length $n = 2k$ is $\prod_{i=1}^{k-1}(2^i + 1)$.

Proof Let us first count the number of ways we can select, vector by vector, a basis for such a subspace of n–space. Since the all–ones vector $\vec{1}$ is always in a self-dual code, we can fix the first vector in our basis to $\vec{1}$. Now, there are 2^{n-1} n–vectors that are orthogonal to the all–ones vector $\vec{1}$. Therefore, we have $2^{n-1} - 2$ choices for our second basis vector (the 2^{n-1} vectors orthogonal to the all–ones vector minus the all–ones vector and the all–zeroes vector). In general, there are 2^{n-i} vectors orthogonal to the first i basis vectors we select. Among these 2^{n-i} vectors, 2^i of them will be in the span of the i basis vectors already selected. This gives us a total of $2^{n-i} - 2^i$ choices for the $(i+1)^{th}$ basis vector. Therefore, starting with the all–ones vector, the number of ways we can select, vector by vector, a basis for such a subspace is $\prod_{i=1}^{k-1} 2^{2k-i} - 2^i$. Of course, this is an overcount (on the total number of binary self-dual codes of length $n = 2k$) since each subspace can be picked many times.

The number of ways a particular subspace is selected is easy to count. Starting with the all–ones vector, the number of ways we can select an n–vector (from our k dimensional subspace) for our second basis vector is $2^k - 2$ and, in general, the number of ways we can select an n–vector (from our subspace) for our $(i+1)^{th}$ basis vector is $2^k - 2^i$. Therefore, our vector by vector method for finding bases will find a basis for a particular subspace $\prod_{i=1}^{k-1} 2^k - 2^i$ times. Dividing and cancelling gives us our result. □

For a more detailed proof see [1].

1.2 The Problem

The problem we will discuss in this thesis is the enumeration of lists of inequivalent $(2k, k)$ self-dual codes, for $k = 1, 2, \ldots, 16$. That is, for each k, $1 \leq k \leq 16$, we will look at how

we can produce a list L of $(2k, k)$ self-dual codes with the property that for any $(2k, k)$ self-dual code C', there exists one and only one code $C \in L$ such that C is equivalent to C'. For each code C in the list L, we will also compute the size of the automorphism group of C.

There are a number of reasons for wanting to produce such a list L of $(2k, k)$ self-dual codes. One reason is that we can learn much about the $(2k, k)$ self-dual codes from such a list. For example, for a given k, it is generally not known what the largest distance d a $(2k, k)$ self-dual code may have. It is also not generally known what weight distributions and automorphism group sizes a $(2k, k)$ self-dual code may have. Since, for any two equivalent $(2k, k)$ self-dual codes, the distance, weight distribution, and size of the automorphism group is the same for both codes, by enumerating a list of inequivalent $(2k, k)$ self-dual codes, we can determine the largest distance, the different weight distributions, and the different automorphism group sizes that exist for the $(2k, k)$ self-dual codes, for $k = 1, 2, \ldots, 16$.

The reason for enumerating a list L containing only inequivalent $(2k, k)$ self-dual codes, instead of all the $(2k, k)$ self-dual codes, is that the number of inequivalent $(2k, k)$ self-dual codes is substantially smaller. For example, the number of inequivalent $(26, 13)$ self-dual codes is 103 while the total number of $(26, 13)$ self-dual codes (which we can compute using Theorem 1.12) is a 24–digit number. Another reason for enumerating only inequivalent $(2k, k)$ self-dual codes is that if C and C' are equivalent $(2k, k)$ self-dual codes then C' can be found by performing some permutation of the coordinates of C. In other words, the *structures* of C and C' are, essentially, the same.

The reason for computing the size of the automorphism group of each $(2k, k)$ self-dual code we produce is that, as we shall see later in the thesis, we can use the automorphism group sizes together with Theorem 1.12 to help check that we have not made any mistakes in our enumeration.

1.3 History

Complete lists of inequivalent $(2k, k)$ self-dual codes, for $k = 1, 2, \ldots, 10$, were first enumerated by Vera Pless in 1971 [2] along with complete lists of $(2k + 1, k)$ self-orthogonal codes, for $k = 1, 2, \ldots, 9$. For each code produced, the weight distribution, the automorphism group, and the size of the automorphism group were all given. The sizes of the automorphism groups and Theorem 1.12 (i.e. the total number of $(2k, k)$ self-dual codes) were used to check that the lists of $(2k, k)$ self-dual codes produced were complete. Similarly, the sizes of the automorphism groups and a count of the total number of $(2k + 1, k)$ self-orthogonal codes were used to check that the lists of $(2k + 1, k)$ self-orthogonal codes were complete.

Complete lists of inequivalent $(22, 11)$ self-dual codes and $(24, 12)$ self-dual codes were first enumerated by Pless and Sloane in 1973 [3]. As with the lists of $(2k, k)$ self-dual codes previously produced, for $k = 1, 2, \ldots, 10$, for each of the $(22, 11)$ and $(24, 12)$ self-dual codes produced, the weight distribution, the automorphism group, and the size of the automorphism group were all given. Similarly, the sizes of the automorphism groups and Theorem 1.12 were used to check that the lists of $(22, 11)$ self-dual codes and $(24, 12)$ self-dual codes produced were complete.

In 1978 [4], Conway and Pless produced complete lists of inequivalent *doubly-even* $(32, 16)$ self-dual codes. Again, for each code produced, the weight distribution, the automorphism group, and the size of the automorphism group were all given. The sizes of the automorphism groups and a count of the total number of doubly-even $(32, 16)$ self-dual codes were used to check that the list of doubly-even $(32, 16)$ self-dual codes produced were complete. The complete list of inequivalent doubly-even $(32, 16)$ self-dual codes produced by Conway and Pless can be used to find complete lists of inequivalent $(2k, k)$ self-dual codes, for $k \leq 15$. In [4], Conway and Pless used the list to count the number of inequivalent $(26, 13)$ self-dual codes and the number of inequivalent $(28, 14)$ self-dual codes.

The number of inequivalent $(30, 15)$ self-dual codes were counted by Conway, Pless, and Sloane in 1990 [5]. Also given were the weight distributions for the $(2k, k, d \geq 6)$ self-dual codes, for $k = 13, 14, 15$.

These four papers provided us with most of the coding theory ideas we used from the literature. Among these ideas were the importance of the weight 4 codewords in characterizing the $(2k, k, 4)$ self-dual codes, and, the use of the size of the automorphism group of a code in ensuring that our results were correct. Many other papers have been done on self-dual codes of which we list a few that pertain to the research we have done.

In [6], Conway and Sloane show there are several self-dual codes with length 34 whose automorphism group has size 1. They also state the following conjecture:

Conjecture 1.13 The smallest possible length for a self-dual code whose automorphism group has size 1 is 34.

We will settle this conjecture in Chapter 7.

The list of inequivalent doubly-even $(32, 16)$ self-dual codes produced by Conway and Pless [4] contains exactly five $(32, 16, 8)$ self-dual codes. In [7], Koch verifies this result by giving a short proof (that does not require knowledge of the size of the automorphism group) that there are exactly 5 doubly-even self-dual codes of length 32 with distance 8. We also found that Oral's paper [8] could be used to check our results. In it, Oral proves that a binary self-orthogonal code with distance 4 can not have an automorphism group with size 1.

On the other hand, most of the combinatorial algorithm ideas in this thesis were not taken from the literature but were developed from general, sound computing practices. If one wishes to read a slightly different approach to these ideas on algorithms and computing practices for combinatorial objects, one can read Lam and Thiel [9].

1.4 Approach

Vera Pless's method for finding a list of inequivalent $(2k, k)$ self-dual codes was to find each inequivalent code, one by one. For each code C found, the size of the automorphism group was computed, which was then used to find the number of codes equivalent to C. The enumeration stopped once the sum of the total number of codes equivalent to the codes produced added up to the appropriate number (i.e. either the total number of $(2k, k)$ self-dual codes for $k = 1, 2, \ldots, 12$, or, the total number of doubly-even $(32, 16)$ self-dual codes). As pointed out by Koch [7], one of the drawbacks of Pless's approach is that the inequivalent codes were found by several processes "including divination." Another drawback is that in order to find a complete list of inequivalent $(2k, k)$ self-dual codes, for $k = 16, 17, 18, 19$ (using the same method that was used to find the $(2k, k)$ self-dual codes, for $k = 13, 14, 15$) a complete list of inequivalent doubly-even $(40, 20)$ self-dual codes must first be found, which would require too much computing time since, as pointed out by Pless in [4], there are at least 17000 inequivalent doubly-even $(40, 20)$ self-dual codes.

In this thesis, we will present a systematic method for producing a complete list of inequivalent $(2k, k)$ self-dual codes. With our method, we can systematically produce a complete list of inequivalent $(2k, k)$ self-dual codes for any k, provided we have complete lists of inequivalent $(2j, j)$ self-dual codes, for $j = 1, 2, \ldots, k-1$, and provided we have enough time and memory. In practice, we were able to use our method to produce complete lists of inequivalent $(2k, k)$ self-dual codes for k up to and including $k = 16$. It also seems very promising that we will be able to soon use our method to produce a complete list of inequivalent $(34, 17)$ self-dual codes.

Our method for producing a complete list of inequivalent $(2k, k)$ self-dual codes works in two stages:

1. In the first stage, we produce a list L of generator matrices with the property that

for any $(2k, k)$ self-dual code C', there exists at least one G in L in which G generates a code C that is equivalent to C'.

2. For each matrix G in the list L, our second stage permutes the columns of G and row-reduces, producing a unique generator matrix G_0 for a unique code in the equivalence class of C, where C is the $(2k, k)$ self-dual code generated by G. In other words, for any pair of matrices $G_1, G_2 \in L$, where G_1 and G_2 generate equivalent codes C_1 and C_2, respectively, when either one of G_1 or G_2 is input into the second stage, a unique matrix G_0 is produced that generates a code equivalent to both C_1 and C_2. For each $G \in L$, the second stage will also count the size of the automorphism group of the code C generated by G.

The distinct generator matrices produced in the second stage of our algorithm give us our complete list of inequivalent $(2k, k)$ self-dual codes.

Note that provided we have complete lists of inequivalent $(2j, j)$ self-dual codes, for $j = 1, 2, \ldots, k-1$, we can produce a complete list of decomposable $(2k, k)$ self-dual codes in a straightforward manner. The sizes of the automorphism groups of the decomposable $(2k, k)$ self-dual codes can also be computed in a straightforward manner. Therefore, we only need to consider the indecomposable $(2k, k)$ self-dual codes in our enumeration. Furthermore, with the exception of the trivial $(2, 1, 2)$ self-dual code, generated by $G =$ [1 1], every $(2k, k, 2)$ self-dual code is decomposable. Therefore, we only need to consider the $(2k, k, d \geq 4)$ self-dual codes in our enumeration.

1.5 What's Ahead

For any $(2k, k)$ self-dual code over $GF(2)$ there exists an equivalent code C that is generated by a matrix of the form $G = [I_k|A]$, where I_k is the $k \times k$ identity matrix and A is a $k \times k$ identity matrix. Therefore, to produce a complete list of inequivalent $(2k, k)$ self-dual codes we can (and will) limit the generator matrices we produce to just those

that have the form $[I_k|A]$. In Chapter 2, we will investigate some of the properties that matrices of the form $G = [I_k|A]$ posses, where $G = [I_k|A]$ generates a $(2k, k)$ self-dual code.

The first stage of our enumeration will produce a list L of generator matrices $G = [I_k|A]$, with the property that for any $(2k, k, d \geq 4)$ self-dual code C there exists at least one matrix $G = [I_k|A]$ in L that generates a code equivalent to C. Our method for producing the list L may result in a large number of generator matrices $G_1 = [I_k|A_1], G_2 = [I_k|A_2], \ldots, G_m = [I_k|A_m]$ being produced in which $G_1, G_2, \ldots, G_m$ all generate equivalent codes, and $A_1 \neq A_2 \neq \cdots \neq A_m$. If every one of these matrices are recorded in the list L then the second stage of our enumeration will perform a lot of unnecessary and time consuming work. Therefore, we would like the list L produced in the first stage of our enumeration to contain as few pairs, $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$, as possible, where $G_1 \neq G_2$, and G_1 and G_2 generate equivalent codes. One way we can do this is to restrict the generator matrices we produce for each equivalence class of $(2k, k, d \geq 4)$ self-dual codes. This is the topic of Chapter 3.

In Chapter 4, we will develop an algorithm for finding a list L_1 of generator matrices $G = [I_k|A]$, with the property that for any $(2k, k, 4)$ self-dual code C there exists at least one matrix $G = [I_k|A]$ in L_1 that generates a code equivalent to C. The algorithm is recursive and assumes complete lists of inequivalent $(2j, j, d)$ self-dual codes are available, where $j \leq k - 2$ and $d \geq 4$. The algorithm is based on the result that for any $(2k, k, 4)$ self-dual code C, there exists a generator matrix $G = [I_k|A]$, where G generates a code equivalent to C, such that G can be *constructed* from a generator matrix for a $(2k-4, k-2)$ self-dual code.

In Chapter 5, we will develop an algorithm for finding a list L_2 of generator matrices $G = [I_k|A]$, with the property that for any $(2k, k, d \geq 6)$ self-dual code C there exists at least one matrix $G = [I_k|A]$ in L_2 that generates a code equivalent to C. The algorithm is recursive and assumes complete lists of inequivalent $(2k, k, 4)$ self-dual codes C_0 are

available, where C_0 does not contain any intersecting weight 4 words. The algorithm is based on the result that for any $(2k, k, d \geq 6)$ self-dual code C, there exists a generator matrix $G = [I_k|A]$, where G generates a code equivalent to C, such that G can be *constructed* from a generator matrix for a $(2k, k, 4)$ self-dual code that does not contain any intersecting weight 4 words.

In Chapter 6, we will develop an algorithm that, given any generator matrix $G = [I_k|A]$, for a $(2k, k)$ self-dual code C, produces a unique matrix $G_0 = [I_k|A_0]$ that generates a code equivalent to C. That is, we will develop an algorithm that, given any pair of matrices $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$, where G_1 and G_2 generate equivalent codes, produces the same matrix $G_0 = [I_k|A_0]$ when run on either one of G_1 or G_2. The algorithm will also count the size of the automorphism group of C. The algorithm developed in this chapter can be used to find a list L of generator matrices $G = [I_k|A]$, with the property that for any $(2k, k, d \geq 4)$ self-dual code C there exists one and only one $G = [I_k|A]$ in L such that G generates a code equivalent to C, by running the algorithm on the lists L_1 and L_2 (produced by the algorithms developed in Chapters 4 and 5) and recording every distinct matrix produced in L. As we shall see in Chapter 6, once we have produced our list L, we can then use the sizes of the automorphism groups of the codes we produced to check the results of our enumeration.

Finally, in Chapter 7, we will give the results of our enumeration. Various counts of $(2k, k)$ self-dual codes are given, for $k = 1, 2, \ldots, 16$, including the total number of inequivalent $(2k, k)$ self-dual codes, the number of indecomposable inequivalent $(2k, k)$ self-dual codes, and the number of inequivalent $(2k, k, d)$ self-dual codes. Also given are tables listing all combinations of weight distributions and automorphism group sizes for the $(2k, k, d \geq 4)$ self-dual codes, for $k = 1, 2, \ldots, 16$.

Chapter 2

Generator Matrices of the form $G = [I_k|A]$

Any (n, k) linear code is equivalent to a code generated by a matrix of the form $G = [I_k|A]$, where I_k is the $k \times k$ identity matrix. Therefore, we can restrict our enumeration of inequivalent self-dual codes to ones that are generated by matrices of the form $G = [I_k|A]$. Furthermore, since all our generator matrices have the identity matrix on the left, we need only store the matrix A. Since we will store only A in our computer programs, we will also develop operations on A, each of which will produce a matrix A' such that $G' = [I_k|A']$ generates a code C' that is equivalent to the code C generated by $G = [I_k|A]$. In this way, we need only store and operate on A, rather than $G = [I_k|A]$. This will make the programs developed later in the thesis more efficient.

In this chapter, we will look at some of the properties of self-dual codes (and linear codes) generated by matrices of the form $G = [I_k|A]$. Section 2.1 looks at some of the general properties that any A must have, if $G = [I_k|A]$ generates a self-dual code. Section 2.2 discusses the conventions and notation we will use in this chapter, and throughout the remainder of the thesis, regarding permutations of the columns of G and row operations on G. Section 2.3 defines 3 operations on the matrix A. Each of these operations are used

to produce a matrix A' such that $G = [I_k|A]$ and $G' = [I_k|A']$ generate equivalent codes. Section 2.4 shows that, given any A, these 3 operations are sufficient to produce any A', where $G' = [I_k|A']$ generates a code equivalent to the code generated by $G = [I_k|A]$. Finally, Section 2.5 gives a useful theorem on weight d words (where d is the distance of the code) in generator matrices of the form $G = [I_k|A]$.

The theorems in this chapter are all deduced from well known facts in coding theory and are generally known. We are not claiming they are new or original.

2.1 Binary Self-Dual Codes Generated By $G = [I_k|A]$

A $(2k, k)$ linear code C, over $GF(2)$, is a self-dual code if for any codewords $\vec{c_i}, \vec{c_j} \in C$ (including $i = j$), $\vec{c_i} \bullet \vec{c_j} = 0$. Since, $\vec{c_i} \bullet \vec{c_i} = 0$, the codewords of C must have even weight. Since, $\vec{c_i} \bullet \vec{c_j} = 0$, any pair of codewords must intersect evenly. Therefore, if G generates a self-dual code C, the rows of G (which are codewords of C) have even weight and intersect evenly. In this section, we will use these two properties to find some of the properties that any $G = [I_k|A]$ that generates a self-dual code must possess.

The first two theorems look at the parity of the rows and columns of A.

Theorem 2.1 Let $G = [I_k|A]$ be a generator matrix for a $(2k, k)$ self-dual code C. Then the rows and columns of A must have odd weight. [1]

Proof Since $G = [I_k|A]$ generates a self-dual code, the rows of G must have even weight, and therefore, the rows of A must have odd weight (since I_k contributes exactly one 1 to the weight). Since C is a self-dual code, $G^{\perp} = [A^T|I_k]$ also generates C. Therefore, the rows of $G^{\perp}$ must have even weight, and therefore, the rows of A^T have odd weight, which implies that the columns of A have odd weight. □

[1] Here the word weight is referring to the number of ones in a row or a column of A.

Theorem 2.2 Let $G = [I_k|A]$ be a generator matrix for a $(2k, k)$ self-dual code C. Then the rows of A intersect evenly and the columns of A intersect evenly.

Proof Since C is a self-dual code, the rows of G must intersect evenly, and since the rows of I_k intersect evenly (i.e. in 0 positions), the rows of A must intersect evenly. Since $G^{\perp} = [A^T|I_k]$ also generates C, the columns of A must intersect evenly. □

We will now show that for any A satisfying the conditions in Theorems 2.1 and 2.2, $G = [I_k|A]$ generates a self-dual code.

Theorem 2.3 Let A be any $k \times k$ binary matrix such that:

1. The rows (and columns) of A have odd weight, and,
2. The rows (and columns) of A intersect evenly.

Then $G = [I_k|A]$ generates a $(2k, k)$ self-dual code.

Proof Since G contains the identity matrix I_k, G generates a $(2k, k)$ linear code C. The codewords of C are any linear combination of the rows of G. Let $\vec{r_i}$ denote the rows of G, $1 \leq i \leq k$. Then, any codeword of C has the form $\vec{c} = \sum_{i \in S} \vec{r_i}$, where S is a subset of the set of integers $\{1, 2, \ldots, k\}$. What we need to show is that for any codewords $\vec{c_a}, \vec{c_b} \in C$, $\vec{c_a} \bullet \vec{c_b} = 0$.

Since the rows of A have odd weight, the rows of G have even weight, and therefore, $\vec{r_i} \bullet \vec{r_i} = 0$. Since the rows of A intersect evenly, the rows of G intersect evenly, and therefore, $\vec{r_i} \bullet \vec{r_j} = 0$, $i \neq j$. Let $\vec{c_a}, \vec{c_b} \in C$ (including $a = b$). Then $\vec{c_a} = \sum_{i \in S_a} \vec{r_i}$ and $\vec{c_b} = \sum_{j \in S_b} \vec{r_j}$, where S_a and S_b are subsets of the set of integers $\{1, 2, \ldots, k\}$. By repeated use of the distributive law $\vec{a} \bullet (\vec{b} + \vec{c}) = \vec{a} \bullet \vec{b} + \vec{a} \bullet \vec{c}$ we see that:

$$\vec{c_a} \bullet \vec{c_b} = (\sum_{i \in S_a} \vec{r_i}) \bullet (\sum_{j \in S_b} \vec{r_j})$$

$$
\begin{aligned}
&= \sum_{i \in S_a} (\vec{r_i} \bullet \sum_{j \in S_b} \vec{r_j}) \\
&= \sum_{i \in S_a} (\sum_{j \in S_b} (\vec{r_i} \bullet \vec{r_j})) \\
&= \sum_{i \in S_a} (\sum_{j \in S_b} 0) \\
&= 0
\end{aligned}
$$

Therefore, C is a self-dual code. □

Note that in our proof of Theorem 2.3, we did not use the fact that the columns of A have odd weight and intersect evenly. Therefore, if the rows of A have odd weight and intersect evenly, then since $G = [I_k|A]$ generates a self-dual code, the columns of A *must* have odd weight and intersect evenly.

In conclusion, our first three theorems show us that $G = [I_k|A]$ generates a self-dual code if and only if the rows (and columns) of A have odd weight and the rows (and columns) of A intersect evenly.

Let us now consider the weights of the rows (and columns) of A with respect to the distance of the code.

Theorem 2.4 Let C be a $(2k, k, d)$ self-dual code generated by $G = [I_k|A]$. Then the rows (and columns) of A have weight $\geq d - 1$.

Proof Since the rows of G are (non-zero) codewords with weight $\geq d$, and since the identity I_k contributes exactly one 1 to the weight, the rows of A have weight $\geq d - 1$. Since $G^{\perp} = [A^T|I_k]$ also generates C, the columns of A have weight $\geq d - 1$. □

We will now conclude this section by showing that any codeword with weight w is the sum of at most $w - 1$ rows of $G = [I_k|A]$. A few lemmas must first be proved in order to show this.

Lemma 2.5 Let M be an $i \times j$ matrix over $GF(2)$ such that:

1. each column of M has even weight, and

2. every pair of rows of M intersect in an even number of positions.

Then, each row of M has even weight.

Proof Let $\vec{r}$ be any row of M, and, let w be the weight of $\vec{r}$. Let S be the $(i-1) \times w$ submatrix consisting of:

- all rows of M, excluding $\vec{r}$, and
- all columns of M where $\vec{r}$ contains a 1.

To show $\vec{r}$ has *even* weight w, the total number of 1's in S will be counted in two ways. Let r_x be the number of 1's in row x of S. Let c_y be the number of 1's in column y of S. Let n be the total number of 1's in S.
By condition 2, $r_x \equiv 0 \bmod 2$, since $\vec{r}$ intersects all rows in an even number of positions. Thus, $n = \sum_{\text{all } x \in S} r_x \equiv (i-1) \cdot 0 \equiv 0 \bmod 2$.
By condition 1, $c_y \equiv 1 \bmod 2$, since all columns of S are missing one 1 from M and all columns of M have even weight. Thus, $n = \sum_{\text{all } y \in S} c_y \equiv w \cdot 1 \equiv w \bmod 2$.
Thus, $w \equiv \sum_{\text{all } y \in S} c_y \equiv \sum_{\text{all } x \in S} r_x \equiv 0 \bmod 2$.
Therefore, any row of M has even weight. □

Lemma 2.6 Let M be an $i \times j$ matrix over $GF(2)$ such that:

1. each row of M has odd weight, and

2. every pair of rows of M intersect in an even number of positions.

Then M contains at least one column with odd weight.

Proof Suppose M does not contain a column with odd weight, i.e. suppose all columns of M have even weight. Then M is a matrix that satisfies the conditions of Lemma 2.5 which implies that all rows of M have even weight, which is a contradiction. □

Lemma 2.7 Let $G = [I_k|A]$ be a generator matrix for a self-dual code C. Then any non-empty subset of the rows of A contains at least one column with odd weight.

Proof Let M be any non-empty subset of the rows of A. Since $G = [I_k|A]$ generates a self-dual code:

1. Each row of M has odd weight, and
2. Every pair of rows of M intersect in an even number of positions.

Thus, by Lemma 2.6, M contains at least one column with odd weight. □

Theorem 2.8 Let $\vec{c}$ be any non-zero codeword of a $(2k, k)$ self-dual code C, generated by $G = [I_k|A]$. Let w be the weight of $\vec{c}$. Then $\vec{c}$ is the sum of at most $w - 1$ rows of G.

Proof Any non-zero $\vec{c}$ is the sum of r rows of $G = [I_k|A]$, $1 \leq r \leq k$. The columns of I_k will contribute r 1's to the weight of $\vec{c}$, and by Lemma 2.7, the columns of A contribute at least one 1 to the weight. Thus, $w \geq r + 1$ which implies $r \leq w - 1$. □

2.2 Permutation Conventions and Notation

In this section, some of the conventions and notation we will use in this chapter, and throughout the rest of the thesis, for permuting the columns of generator matrices are discussed.

Let G be a generator matrix for an (n, k) linear code C. The columns of G will be referenced by the sequence of integers $1, 2, 3, \ldots, n$. The way we will carry out permutations of the columns of G is as follows: Let

$$\pi = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ b_1 & b_2 & b_3 & \cdots & b_n \end{pmatrix}$$

be a permutation of the columns of G and let $\pi(G)$ denote the matrix that results from applying π to G. Then column i of $\pi(G)$ will be column b_i of G, $1 \leq i \leq n$. Products of permutations will be carried out from right to left. That is $\pi_1\pi_2(G)$ means $\pi_1(\pi_2(G))$.

When we are working with a generator matrix of the form $G = [I_k|A]$, we will sometimes describe permutations of the columns of G in terms of the columns of I_k and/or the columns of A. When this is the case, the columns of I_k will be referenced by the sequence of integers $1, 2, \ldots, k$ and the columns of A will be referenced by the sequence of integers $1, 2, \ldots, n-k$.

Let $G = [I_k|A]$ be a generator matrix for an (n, k) linear code and let π be permutation of the columns of G such that the resulting $\pi(G)$ row-reduces to a matrix of the form $G' = [I_k|A']$. Many of our examples will apply such a π to G and row-reduce the resulting $\pi(G)$ to $G' = [I_k|A']$. In many cases, we will be interested in what happens to the columns of I_k in G, the columns of A in G, and/or the rows of G. Whenever we are interested in keeping track of the columns of I_k in G, the columns of I_k will be labeled by the sequence $a_1, a_2, \ldots, a_k$. Whenever we are interested in keeping track of the columns of A in G, the columns of A will be labeled by the sequence $c_1, c_2, \ldots, c_{n-k}$. Whenever we are interested in keeping track of the rows of G, the rows will be labeled by the sequence $r_1, r_2 \ldots, r_k$.

Example 2.9 Let C be the $(8,4)$ self-dual code generated by:

$$G = [I_4|A] = \begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{matrix}$$

The columns $1,2,3,4$ of I_4 are labeled with a_1, a_2, a_3, a_4, the columns $1,2,3,4$ of A (i.e. columns $5,6,7,8$ of G) are labeled with c_1, c_2, c_3, c_4, and the rows $1,2,3,4$ of G are labeled with r_1, r_2, r_3, r_4.

If $\pi = (1\ 2\ 8)(3)(4\ 5\ 7\ 6)$ is a permutation of the columns of G then

$$\pi(G) = \begin{array}{c} \begin{matrix} a_2 & c_4 & a_3 & c_1 & c_3 & a_4 & c_2 & a_1 \end{matrix} \\ \left[\begin{array}{cccc|cccc} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{matrix}$$

The row-reduced echelon form of $\pi(G)$ is:

$$G' = [I_4|A'] = \begin{array}{c} \begin{matrix} a_2 & c_4 & a_3 & c_1 & c_3 & a_4 & c_2 & a_1 \end{matrix} \\ \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \oplus r_2 \oplus r_3 \\ r_4 \\ r_1 \oplus r_3 \oplus r_4 \\ r_1 \end{matrix}$$

When we only work with the matrix A, we may also be interested in what happens to the rows and columns of the matrix $G = [I_k|A]$. Whenever we are interested in keeping track of the rows of $G = [I_k|A]$, the corresponding rows of A will be labeled with $r_1, r_2, \ldots, r_k$. Whenever we are interested in keeping track of the columns of I_k in

$G = [I_k|A]$, the rows of A will be labeled with $(a_1), (a_2), \ldots, (a_k)$. How labeling the rows of A allows us to keep track of the columns of I_k in $G = [I_k|A]$ will be made clear in the next section.

Example 2.10 For the $(8,4)$ self-dual code C generated by $G = [I_4|A]$ in example 2.9, the rows and columns of A will be labeled as follows:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{ll} r_1 & (a_1) \\ r_2 & (a_2) \\ r_3 & (a_3) \\ r_4 & (a_4) \end{array}$$

2.3 The Three Operations on A

Let C_0 and C_1 be any two (n,k) linear codes over GF(2) generated by $G_0 = [I_k|A_0]$ and $G_1 = [I_k|A_1]$, respectively. Recall that C_0 and C_1 are *equivalent* if and only if there exists a permutation π of the columns of G_0 such that $\pi(G_0)$ and G_1 generate the same code. Furthermore, any permutation π can be written as a product of transpositions. Therefore, for any (n,k) linear code code C generated by $G = [I_k|A]$, any transposition of the columns of G results in a matrix that generates a code equivalent to C. In this section, we will look at the subset of transpositions of $G = [I_k|A]$ in which the resulting matrix row-reduces to a matrix of the form $G' = [I_k|A']$. The end result will be the definition of 3 operations on A such that each operation results in a matrix A' such that $G' = [I_k|A']$ generates a code equivalent to C.

Let C be any (n,k) linear code generated by $G = [I_k|A]$. Let T be the set of the transpositions of the columns of G such that for any $\pi \in T$, the row-reduced echelon form of $M = \pi(G)$ has the form $G' = [I_k|A']$. Since we are only storing the matrix A, we would like to find operations on A that result in the matrix A', when any transposition $\pi \in T$ is applied to G. To do this, we will separate the transpositions of T into 3 classes:

1. transpositions between two columns of I_k,

2. transpositions between two columns of A, and

3. transpositions between a column of I_k and a column of A.

For each, we will give the row-operations on M necessary to produce G', and use this to define our operations on A.

2.3.1 Swapping Two Columns of I_k

Let C be an (n,k) linear code code generated by $G = [I_k|A]$. Let i_0 and i_1 be any two columns of I_k (i.e. two distinct integers between 1 and k). If we swap these two columns in G, we produce the matrix $M = [N|A]$ where:

- row i_0 of N contains a 1 in column i_1, and 0's in its other columns,
- row i_1 of N contains a 1 in column i_0, and 0's in its other columns, and
- any other row i_2 in N contains a 1 in column i_2, and 0's in its other columns.

Therefore, to row-reduce M to $G' = [I_k|A']$, all we need to do is swap rows i_0 and i_1 in M. Thus, A' is A with rows i_0 and i_1 swapped.

Example 2.11 Let C be the $(8,4)$ self-dual code generated by:

$$G = [I_4|A] = \begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 \end{matrix} \\ \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{matrix}$$

If we swap columns 1 and 3 of I_4, we produce the matrix:

$$M = [N|A] = \begin{array}{c} \begin{matrix} a_3 & a_2 & a_1 & a_4 \end{matrix} \\ \left[\begin{array}{cccc|cccc} 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{matrix}$$

By swapping rows 1 and 3 of M, it is row-reduced to:

$$G' = [I_4|A'] = \begin{array}{c} \begin{matrix} a_3 & a_2 & a_1 & a_4 \end{matrix} \\ \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_3 \\ r_2 \\ r_1 \\ r_4 \end{matrix}$$

Thus, A' is just A with rows 1 and 3 swapped:

$$A = \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right] \begin{matrix} r_1 & (a_1) \\ r_2 & (a_2) \\ r_3 & (a_3) \\ r_4 & (a_4) \end{matrix} \qquad A' = \left[\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{array}\right] \begin{matrix} r_3 & (a_3) \\ r_2 & (a_2) \\ r_1 & (a_1) \\ r_4 & (a_4) \end{matrix}$$

Therefore, for any (n, k) linear code C generated by $G = [I_k|A]$, if we swap any two columns, i_0 and i_1, of I_k, we get an equivalent code C' generated by the matrix $G' = [I_k|A']$, where A' is A with rows i_0 and i_1 swapped. We will use the notation $\text{SWAP}_r(i_0, i_1)$ to represent the swapping of rows i_0 and i_1 of A. The resulting matrix A' will be denoted by $\text{SWAP}_r(i_0, i_1)(A)$.

2.3.2 Swapping Two Columns of A

Let C be an (n, k) linear code generated by $G = [I_k|A]$. Let j_0 and j_1 be any two columns of A. If we swap these two columns in G, we produce the matrix $G' = [I_k|A']$, where A' is A with columns j_0 and j_1 swapped. No row-reductions are necessary.

Therefore, for any (n,k) linear code C generated by $G = [I_k|A]$, if we swap any two columns j_0 and j_1 of A, we get an equivalent code C' generated by the matrix $G' = [I_k|A']$, where A' is A with columns j_0 and j_1 swapped. We will use the notation $\text{SWAP}_c(j_0, j_1)$ to represent the swapping of columns j_0 and j_1 of A. The resulting matrix A' will be denoted by $\text{SWAP}_c(j_0, j_1)(A)$.

2.3.3 Swapping a Column of I_k with a Column of A

Let C be an (n,k) linear code generated by $G = [I_k|A]$. Let i be a column of I_k and let j be a column of A (i.e. i is an integer between 1 and k and j is an integer between 1 and $n-k$). If we swap these two columns in G, producing the matrix $M = [M_0|M_1]$, row i of M_0 will contain the value of entry (i,j) of A in column i, and the value of 0 in all other columns. Therefore, if we want this transposition to produce an M that can be row-reduced to $G' = [I_k|A']$, we require that entry (i,j) of A be 1 (otherwise, M_0 would contain a row of all 0's). To row-reduce $M = [M_0|M_1]$ to $G' = [I_k|A']$, we must add row i of M to all other rows of M that contain a 1 in column j of M_0.

Example 2.12 Let C be the $(12,6)$ self-dual code generated by:

$$G = [I_6|A] = \begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{matrix} \\ \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{matrix}$$

If we swap column 5 of I_6 with column 1 of A, we produce the matrix:

$$M = [M_0|M_1] = \begin{array}{c} \begin{array}{cccccc|cccccc} a_1 & a_2 & a_3 & a_4 & c_1 & a_6 & a_5 & c_2 & c_3 & c_4 & c_5 & c_6 \end{array} \\ \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{array}$$

M is row-reduced to $G' = [I_6|A']$ by adding row 5 of M to all other rows of M that contain a 1 in column 5 of M_0. That is, row 5 is added to rows $1, 2, 3, 4$ producing the matrix:

$$G' = [I_6|A'] = \begin{array}{c} \begin{array}{cccccc|cccccc} a_1 & a_2 & a_3 & a_4 & c_1 & a_6 & a_5 & c_2 & c_3 & c_4 & c_5 & c_6 \end{array} \\ \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ r_1 \oplus r_5 \\ r_2 \oplus r_5 \\ r_3 \oplus r_5 \\ r_4 \oplus r_5 \\ r_5 \\ r_6 \end{array}$$

Let us now consider what effect such a transposition, and the row-reductions, have on A. Since column i of M_0 contains column j of A, row i of M will be added to a row i_2 of M, $i_2 \neq i$, if and only if row i_2 of G contains a 1 in column j of A. Furthermore, since column j of M_1 contains a 1 in row i and 0 in all other rows, column j of A' and column j of A are equal. Thus, we can find A' from A as follows:

1. Let $\vec{v}$ be row i of A with a 0 in place of the 1 in column j.
2. Add $\vec{v}$ to all other rows of A that contain a 1 column j, producing A'.

We shall call this operation *add row i at column j* of A.

Example 2.13 Let C be the $(12,6)$ self-dual code of example 2.12 generated by $G = [I_6|A]$, where:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{ll} r_1 & (a_1) \\ r_2 & (a_2) \\ r_3 & (a_3) \\ r_4 & (a_4) \\ r_5 & (a_5) \\ r_6 & (a_6) \end{array}$$

If we add row 5 of at column 1 of A, we produce the matrix:

$$A' = \begin{array}{c} \begin{matrix} a_5 & c_2 & c_3 & c_4 & c_5 & c_6 \end{matrix} \\ \left[\begin{array}{cccccc} \mathbf{1} & 1 & 0 & 0 & 0 & 1 \\ \mathbf{1} & 1 & 0 & 0 & 1 & 0 \\ \mathbf{1} & 1 & 0 & 1 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{ll} r_1 \oplus r_5 & (a_1) \\ r_2 \oplus r_5 & (a_2) \\ r_3 \oplus r_5 & (a_3) \\ r_4 \oplus r_5 & (a_4) \\ r_5 & (c_1) \\ r_6 & (a_6) \end{array}$$

A' is produced by adding $\vec{v} = \mathbf{0}\ 0\ 1\ 1\ 1\ 1$ to rows $1, 2, 3, 4$ of A.

Therefore, for any (n,k) linear code C generated by $G = [I_k|A]$, if we swap column i of I_k with column j of A, where entry (i,j) of A is 1, we get an equivalent code C' generated by a matrix $G' = [I_k|A']$, where A' is produced by applying the operation *add row i at column j* to A. We will use the notation $\mathrm{ADD}_{r,c}(i,j)$ to represent this operation on A. The resulting matrix A' will be denoted by $\mathrm{ADD}_{r,c}(i,j)(A)$.

Summarizing, let C be any (n,k) linear code generated by $G = [I_k|A]$. We have defined 3 operations on A:

1. $\text{SWAP}_r(i_0, i_1)$, which swaps rows i_0 and i_1 of A,

2. $\text{SWAP}_c(j_0, j_1)$, which swaps columns j_0 and j_1 of A, and,

3. $\text{ADD}_{r,c}(i, j)$: which *adds row i at column j* of A, where entry (i, j) of A is 1.

Each of these operations simulates a transposition of the columns of G, followed by the row operations that produce the matrix $G' = [I_k|A']$, which generates a code equivalent to C. $\text{SWAP}_r(i_0, i_1)$ swaps any 2 columns of I_k in G, and row-reduces to G', $\text{SWAP}_c(j_0, j_1)$ swaps any 2 columns of A in G, producing G', and $\text{ADD}_{r,c}(i, j)$ swaps a column of I_k with a column of A in G, and row-reduces to G'.

These three operations are used throughout the rest of the thesis to find generator matrices of equivalent codes that can be generated by matrices of the form $G = [I_k|A]$.

2.4 Equivalent Codes Generated by Matrices of the form $G = [I_k|A]$

Given two matrices, $G_0 = [I_k|A_0]$ and $G_1 = [I_k|A_1]$, where $A_0 \neq A_1$, G_0 and G_1 may or may not generate equivalent codes. If they do generate equivalent codes then there exists a permutation π of the columns of G_0 such that the row-reduced echelon form of $\pi(G_0)$ is G_1. Now, any permutation π can be written as a product of transpositions, and we have three operations on the matrix A that perform a transposition on the columns of $G = [I_k|A]$. They are the $\text{SWAP}_r(i_0, i_1)$, $\text{SWAP}_c(j_0, j_1)$, and $\text{ADD}_{r,c}(i, j)$ operations. A SWAP_r can be used to swap any two columns of I_k and a SWAP_c can be used to swap any two columns of A. However, an $\text{ADD}_{r,c}$ can only perform a subset of the transpositions between columns of I_k and A. Therefore, it is not clear if these three operations are sufficient to permute a given $G_0 = [I_k|A_0]$ to any $G_1 = [I_k|A_1]$ that generates an equivalent code. In this section we prove that these three operations are indeed sufficient to carry out any such permutation.

We will demonstrate the method of our proof with an example.

Example 2.14 Let C_0 be the $(11, 4)$ code generated by:

$$G_0 = [I_4|A_0] = \begin{array}{c} \begin{array}{ccccccccccc} a_1 & a_2 & a_3 & a_4 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{array} \\ \left[\begin{array}{cccc|ccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{l} r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}$$

Let us first find a matrix $G' = [I_4|A']$ that generates a code C_1 that is equivalent to C_0.

If we perform the permutation $(1\ 2\ 6)(3)(4\ 9\ 7)(5\ 8)(10)(11)$ on the columns of G, we produce the matrix:

$$M = [M_0|M_1] = \begin{array}{c} \begin{array}{ccccccccccc} a_2 & c_2 & a_3 & c_5 & c_4 & a_1 & a_4 & c_1 & c_3 & c_6 & c_7 \end{array} \\ \left[\begin{array}{cccc|ccccccc} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{array}{l} r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}$$

which row-reduces to the matrix:

$$G_1 = [I_4|A_1] = \begin{array}{c} \begin{array}{ccccccccccc} a_2 & c_2 & a_3 & c_5 & c_4 & a_1 & a_4 & c_1 & c_3 & c_6 & c_7 \end{array} \\ \left[\begin{array}{cccc|ccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{array}{l} r_1 \oplus r_2 \\ r_1 \\ r_2 \oplus r_3 \oplus r_4 \\ r_4 \end{array}$$

which generates a code C_1 that is equivalent to C_0.

We now have two equivalent codes C_0 and C_1, generated by $G_0 = [I_4|A_0]$ and $G_1 = [I_4|A_1]$, respectively. Let us now see how we can find A_1 by applying a sequence of the three operations $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c to A_0.

First, let us write A_0 and A_1 with the labels for the columns of G_0 and G_1:

$$A_0 = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} (a_1) \\ (a_2) \\ (a_3) \\ (a_4) \end{matrix}$$

$$A_1 = \begin{array}{c} \begin{matrix} c_4 & a_1 & a_4 & c_1 & c_3 & c_6 & c_7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} (a_2) \\ (c_2) \\ (a_3) \\ (c_5) \end{matrix}$$

We will produce A_1 from A_0 in 2 stages, as follows:

1. Our first stage is to ensure that I_4 and A_0 contain the correct columns in some sequence. That is, our first stage produces a matrix A', from A_0, in which the rows and columns of A' are permutations of the rows and columns of A_1. In terms of the matrix $G_0 = [I_4|A_0]$, the first stage will swap the set of columns $\{1,4\}$ of I_4 (i.e. the columns labeled with a_1 and a_4) with the set of columns $\{2,5\}$ of A_0 (i.e. the columns labeled with c_2 and c_5). This results in a matrix $M = [M_0|M_1]$, such that the first 4 columns of M are a permutation of the set of columns of G_0 labeled with a_2, a_3, c_2, c_5 and the last 7 columns of M are a permutation of the set of columns of G_0 labeled with $a_1, a_4, c_1, c_3, c_4, c_6, c_7$. The row-reduced echelon form of M is a matrix $G' = [I_k|A']$. As we shall see, A' can be found by applying a sequence of $\text{ADD}_{r,c}$ operations to A_0.

2. The second stage ensures that the sequence of rows and columns in I_4 and A' are correct. The second stage does this by performing the necessary SWAP_r and SWAP_c operations on A' until A_1 is produced.

Let us now produce A_1 from A_0 using this 2 stage method.

Stage 1: Swap the set of columns $\{1,4\}$ of I_4 with the set of columns $\{2,5\}$ of A.

- Entry $(1,2)$ of A_0 contains a 1 so we can perform an $\text{ADD}_{r,c}(1,2)$ on A_0, producing

the matrix:

$$A_0' = \begin{array}{c} \begin{matrix} c_1 & a_1 & c_3 & c_4 & c_5 & c_6 & c_7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} (c_2) \\ (a_2) \\ (a_3) \\ (a_4) \end{matrix}$$

- Entry $(4,5)$ of A_0' contains a 1 so we can perform an $\mathrm{ADD}_{r,c}(4,5)$ on A_0', producing the matrix:

$$A' = \begin{array}{c} \begin{matrix} c_1 & a_1 & c_3 & c_4 & a_4 & c_6 & c_7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} (c_2) \\ (a_2) \\ (a_3) \\ (c_5) \end{matrix}$$

Stage 2: Perform SWAP_r and SWAP_c operations on A' until A_1 is produced.

- A $\mathrm{SWAP}_r(1,2)$ operation take care of the rows, and the 2 operations $\mathrm{SWAP}_c(1,4)$ and $\mathrm{SWAP}_c(3,5)$ takes care of the columns. This gives us the matrix:

$$A_1 = \begin{array}{c} \begin{matrix} c_4 & a_1 & a_4 & c_1 & c_3 & c_6 & c_7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} (a_2) \\ (c_2) \\ (a_3) \\ (c_5) \end{matrix}$$

Thus, we have found A_1 by applying a sequence of the 3 operations $\mathrm{ADD}_{r,c}$, SWAP_r, and SWAP_c to A_0.

We are now ready to prove that for any two matrices $G_0 = [I_k|A_0]$ and $G_1 = [I_k|A_1]$, where G_0 and G_1 generate equivalent codes, that A_1 can be found by applying a sequence of the three operations $\mathrm{ADD}_{r,c}$, SWAP_r, and SWAP_c to A_0. We will begin by proving

there exists a sequence of $\mathrm{ADD}_{r,c}$ operations that accomplishes the first stage of our example.

Lemma 2.15 Let $G_0 = [I_k|A_0]$ be a generator matrix for an (n,k) linear code C_0. Let T_I denote any set of m columns of I_k and let T_A denote any set of m columns of A. $1 \leq m \leq \min(k, n-k)$, such that any matrix M that results from swapping the set of columns T_I with the set of columns T_A, has a row-reduced echelon form of $[I_k|A']$. Let T_R denote the set of m rows of A that corresponds to T_I. Then there exists a sequence of m $\mathrm{ADD}_{r,c}$ operations on A that includes every row of T_R and every column of T_A. That is, there exists a sequence $\mathrm{ADD}_{r,c}(i_1,j_1), \mathrm{ADD}_{r,c}(i_2,j_2), \ldots, \mathrm{ADD}_{r,c}(i_m,j_m)$, where $T_R = \{i_1, i_2, \ldots, i_m\}$ and $T_A = \{j_1, j_2, \ldots, j_m\}$, such that for $x = 1, 2, \ldots, m$, entry (i_x, j_x) of the matrix that results from applying the sequence $\mathrm{ADD}_{r,c}(i_1,j_1), \ldots, \mathrm{ADD}_{r,c}(i_{x-1},j_{x-1})$ to A has a value of 1.

Proof by induction on m.

Base Case: Both T_I and T_A consist of a single element (i.e. column index). Let i_1 be the column index in T_I and let j_1 be the column index in T_A. Then entry (i_1, j_1) of A must have a value of 1, for if it did not then if we swapped column i_1 of I_k with column j_1 of A, producing the matrix $M = [M_0|M_1]$, row i_1 of M_0 would be a row of zeroes, which implies M does not row-reduce to a matrix with the form $[I_k|A']$, which is a contradiction. Therefore, $\mathrm{ADD}_{r,c}(i_1,j_1)$ is defined.

Induction Step: Assume for any matrix $G' = [I_k|A']$ and sets T'_I, T'_A, T'_R that meet the criteria of our theorem, where $|T'_I| = m-1$, $1 \leq m-1 < \min(k, n-k)$, that there exists a sequence of $m-1$ $\mathrm{ADD}_{r,c}$ operations on A' that includes every element of T'_R and every element of T'_A.

Let $G = [I_k|A]$ be any matrix and let T_I, T_A, T_R be any sets that meet the criteria of our theorem, where $|T_I| = m$. Consider the $m \times m$ submatrix S of A determined by T_R and T_A. S cannot be the zero matrix, for if it was then if we swapped, in G, the set of

columns T_I with the set of columns T_A, producing the matrix $M = [M_0|M_1]$, M_0 would contain m rows of zeroes, which implies M does not row-reduce to a matrix with the form $[I_k|A']$, which is a contradiction. Therefore, there exists an entry (i_1, j_1) in A with a value of 1, where $i_1 \in T_R$ and $j_1 \in T_A$.

Apply $\text{ADD}_{r,c}(i_1, j_1)$ to A, producing the matrix A', and let $T'_I = T_I - i_1$, $T'_A = T_A - j_1$, and $T'_R = T_R - i_1$. If we swap the set of columns T'_I with the set of columns T'_A in $G' = [I_k|A']$, then the resulting matrix M must have a row-reduced echelon form of $[I_k|A'']$ (since swapping T_I and T_A in G results in such an M). Therefore, the matrix $G' = [I_k|A']$ and the sets T'_I, T'_A, T'_R meet the criteria of our theorem. Since $|T'_I| = m - 1$, by our induction hypothesis, there must exist a sequence $\text{ADD}_{r,c}(i_2, j_2), \ldots, \text{ADD}_{r,c}(i_m, j_m)$ of $m - 1$ $\text{ADD}_{r,c}$ operations on A', where, $T'_I = \{i_2, \ldots, i_m\}$ and $T'_A = \{j_2, \ldots, j_m\}$. Therefore, $\text{ADD}_{r,c}(i_1, j_1), \text{ADD}_{r,c}(i_2, j_2), \ldots, \text{ADD}_{r,c}(i_m, j_m)$ is a sequence of m $\text{ADD}_{r,c}$ operations on A that includes every element of T_R and every element of T_A. Thus, there exists a sequence of m $\text{ADD}_{r,c}$ operations on A that includes every elment of T_R and every element of T_A.

Thus, our result follows by the principle of mathematical induction. □

Therefore, given any two matrices $G_0 = [I_k|A_0]$ and $G_1 = [I_k|A_1]$, where G_0 and G_1 generate equivalent (n, k) linear codes, there exists a sequence of $\text{ADD}_{r,c}$ operations on A that results in a matrix A' such that the rows and columns of A' are a permutation of the rows and columns of A_1. Since we can perform any permutation of rows with SWAP_r operations and we can perform any permutation of columns with SWAP_c operations, we have the following:

Theorem 2.16 Let $G_0 = [I_k|A_0]$ be a generator matrix for an (n, k) linear code C_0. Let $G_1 = [I_k|A_1]$ be any matrix that generates an (n, k) linear code C_1, equivalent to C_0. Then A_1 can be produced by applying a sequence of the three operations $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c to A_0.

2.5 Weight d Words in $G = [I_k|A]$

In this section, we will prove the very useful result that, for any (n, k, d) linear code C and weight d word $\vec{w} \in C$, there exists an equivalent code C', generated by $G' = [I_k|A']$, such that $\vec{w}'$, the weight d word in C' corresponding to $\vec{w}$, is a row of G'

Since for any linear code there exists an equivalent code generated by a matrix of the form $[I_k|A]$, we can prove our result by considering the weight d words of a (n, k, d) linear code C that is generated by a matrix of the form $G = [I_k|A]$. We will prove our result by showing that for any weight d word in such a C, there exists a sequence of $\text{ADD}_{r,c}$ operations on A, that results in a matrix A', such that the weight d word in the equivalent code C', generated by $G' = [I_k|A']$, that corresponds to $\vec{w}$, is a row of G'.

We will begin by examining the effect that an $\text{ADD}_{r,c}$ operation on A has on the rows of G that sum to a specific codeword.

Let $\vec{w}$ be any codeword in C and let w denote the weight of $\vec{w}$. Let $G_{\vec{w}}$ denote the set of m rows of G that sum to $\vec{w}$ (recall, by Theorem 2.8, $1 \leq m \leq w - 1$). Let $A_{\vec{w}}$ denote the set of m rows of A corresponding to $G_{\vec{w}}$. Note that since I_k contributes exactly m ones to the weight of $\vec{w}$, $A_{\vec{w}}$ must contain exactly $w - m$ columns with odd weight.

Suppose $A_{\vec{w}}$ contains a column j with even weight ≥ 2. Then there exists a non-zero entry (i, j) in A, where i is a row of $A_{\vec{w}}$. Let A' be the matrix that results from applying $\text{ADD}_{r,c}(i, j)$ to A. Then $G' = [I_k|A']$ generates a code C', equivalent to C. The effect the $\text{ADD}_{r,c}$ operation has on the matrix $G = [I_k|A]$ is to swap column i of I_k with column j of A, and then row-reduce to $G' = [I_k|A']$. Since column i of I_k contributes a one to $\vec{w}$ and column j of A contributes a 0 to $\vec{w}$, the identity I_k in $G' = [I_k|A']$ will contribute exactly $m - 1$ ones to the weight of the codeword $\vec{w}' \in C'$ that corresponds to the weight w word $\vec{w}$ in C. Therefore, $\vec{w}'$ is the sum of $m - 1$ rows of G'. (The $m - 1$ rows of G' corresponding to $G_{\vec{w}}$, excluding row i).

Example 2.17 Let C be the $(14, 7, 4)$ self-dual code generated by:

$$G = [I_7|A] = \left[\begin{array}{ccccccc|ccccccc|l} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & r_1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & r_2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & r_4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & r_5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & r_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & r_7 \end{array}\right]$$

Consider the weight 4 word $\vec{w} = r_4 \oplus r_5 \oplus r_6$. The columns of G that contribute the 4 ones to $\vec{w}$ are columns $4, 5, 6$ of I_7 and column 5 of A. $A_{\vec{w}}$ = rows $4, 5, 6$ of A, and $(i, j) = (6, 7)$ is a non-zero entry in A such that row 6 of A is a row of $A_{\vec{w}}$ and column 7 in $A_{\vec{w}}$ has even weight ≥ 2. Let A' be the matrix that results from applying $\mathrm{ADD}_{r,c}(6, 7)$ to A. Then,

$$G' = [I_7|A'] = \left[\begin{array}{ccccccc|ccccccc|l} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & r_1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & r_2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & r_4 \oplus r_6 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & r_5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & r_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & r_6 \oplus r_7 \end{array}\right]$$

which generates an equivalent code C'. The weight 4 word in C' that corresponds to $\vec{w}$ is the codeword $\vec{w}' = (r_4 \oplus r_6) \oplus r_5$ (i.e. the sum of rows 4 and 5 of G'). The columns of G' that contribute the 4 ones to $\vec{w}'$ are columns $4, 5$ of I_7 and columns $5, 7$ of A.

What we have shown is, for any codeword $\vec{w}$ of a (n, k, d) linear code C generated by $G = [I_k|A]$, such that $\vec{w}$ is the sum of the m rows $G_{\vec{w}}$ of G, where $G_{\vec{w}}$ contains at least one column with even weight ≥ 2, there exists an equivalent code C', generated by

$G' = [I_k|A']$, such that $\vec{w}'$, the codeword of C' corresponding to $\vec{w}$, is the sum of $m-1$ rows of G'. This result leads us to the following theorem:

Theorem 2.18 Let C be a (n,k,d) linear code generated by $G = [I_k|A]$. Let $\vec{w}$ be any weight d word of C. Then there exists a code C', equivalent to C, generated by $G' = [I_k|A']$, such that $\vec{w}'$, the codeword of C' corresponding to $\vec{w}$, is a row of G'.

Proof To prove this, all we need to show now is that for any weight d word $\vec{w}$ that is the sum of $m \geq 2$ rows, $G_{\vec{w}}$ of G, there exists a column j in A such that $G_{\vec{w}}$ has even weight ≥ 2 in column j. If this is the case, then, as we've just seen, we can perform a sequence of $m-1$ $\text{ADD}_{r,c}$ operations on A, until an A' is produced such that for the code C', generated by $G' = [I_k|A']$, the weight d word $\vec{w}'$ in C' that corresponds to $\vec{w}$ is a row of G'.

Suppose such a column j does not exist. That is, suppose $\vec{w}$ is a weight d word that is the sum of m rows, $G_{\vec{w}}$ of G, $m \geq 2$, such that $G_{\vec{w}}$ does not contain a column with even weight ≥ 2. Let $A_{\vec{w}}$ be the columns of A corresponding to $G_{\vec{w}}$. Since $G_{\vec{w}}$ does not contain any columns with even weight ≥ 2, $A_{\vec{w}}$ consists of $d-m$ columns with odd weight and $(n-k)-(d-m)$ columns of zeros. This implies that any row in $G_{\vec{w}}$ has weight $\leq 1+d-m \leq 1+d-2 = d-1 < d$, which is a contradiction. Therefore, there cannot exist a weight d word $\vec{w}$ that is the sum of ≥ 2 rows of G that contains no columns with even weight ≥ 2.

Thus, for any weight d word of C, there exists an equivalent code C', generated by $G' = [I_k|A']$, such that the codeword of C' corresponding to $\vec{w}$ is a row of G'. □

2.6 Summary

In this chapter we had a look at some properties of (n,k) linear codes that are generated by matrices of the form $G = [I_k|A]$.

We found that $G = [I_k|A]$ generates a self-dual code C if and only if the rows (and columns) of A have odd weight and intersect evenly. It was also shown that if $G = [I_k|A]$ generates a $(2k, k, d)$ self-dual code then the rows (and columns) of A must have weight $\geq d - 1$. Furthermore, any weight w word of C is the sum of no more than $w - 1$ rows of $G = [I_k|A]$.

For a linear code C generated by $G = [I_k|A]$, we defined three operations on A:

- $\text{SWAP}_r(i_0, i_1)$, which swaps rows i_0 and i_1 of A,
- $\text{SWAP}_c(j_0, j_1)$, which swaps columns j_0 and j_1 of A, and,
- for any entry (i, j) of A with a value of 1, $\text{ADD}_{r,c}(i, j)$, which adds row i (excluding column j) to all other rows of A that have a 1 in column j.

Each of these operations results in an A' such that $G' = [I_k|A']$ generates a code equivalent to C. Furthermore, these three operations are sufficient (i.e. we can apply a sequence of them to A) to produce any A' such that $G' = [I_k|A']$ generates a code C', equivalent to C.

We concluded, in the previous section, with a theorem that will prove very useful throughout the thesis. That is, for any (n, k, d) linear code C and weight d word $\vec{w} \in C$, there exists an equivalent code C', generated by $G' = [I_k|A']$, such that $\vec{w}'$, the weight d word in C' corresponding to $\vec{w}$, is a row of G'.

Chapter 3

Ordering The Matrix A

Our approach to enumerating a list of inequivalent $(2k, k)$ self-dual codes, for $2k \leq 32$, consists of two main steps:

1. Find a list L of $k \times k$ matrices such that for any $(2k, k)$ self-dual code C there exists at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C.

2. For each matrix $A \in L$, permute the columns of $G = [I_k|A]$ (and row-reduce) in such a way that a unique matrix $G_0 = [I_k|A_0]$ is produced. That is, for any pair of matrices $A_1, A_2 \in L$, where $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes C_1 and C_2, respectively, when either one of G_1 or G_2 is input into the second stage, a unique matrix G_0 is produced that generates a code equivalent to both C_1 and C_2.

Our algorithms for finding the list L in the first step of our enumeration may result in a large number of matrices $A_1, A_2, \ldots, A_m$ being produced in which the generator matrices $[I_k|A_1], [I_k|A_2], \ldots, [I_k|A_m]$ all generate equivalent codes, but $A_1 \neq A_2 \neq \cdots \neq A_m$. The second step of our enumeration will have to be applied to each of these matrices and can be quite time consuming. Therefore, we would like the list L found in the first step to

contain as few pairs A_1, A_2 as possible, in which $[I_k|A_1]$ and $[I_k|A_2]$ generate equivalent codes and $A_1 \neq A_2$. One way we can do this is to limit the number of distinct matrices that generate equivalent codes which can be produced in the first step of our enumeration. This is the topic of this chapter.

Let $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ be any two matrices that generate equivalent $(2k, k)$ self-dual codes, where $A_1 \neq A_2$. As we saw in Chapter 2, we can find A_2 by applying some sequence of $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c operations to A_1. If A_2 can be found by applying a sequence of only SWAP_r and SWAP_c operations to A_1 then the rows and columns of A_1 and A_2 are permutations of one another. We can reduce the number of matrices A_1 and A_2 produced in the first step of our enumeration in which the rows and columns of A_1 and A_2 are simply permutations of one another, and $A_1 \neq A_2$, by *ordering* the rows and columns of any $k \times k$ matrix A we produce. This is the topic of Section 3.1. If $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes and A_2 *cannot* be found by applying a sequence of only SWAP_r and SWAP_c operations to A_1 then the rows and columns of A_1 and A_2 will not be permutations of one another. In this case, A_2 can be found by applying some sequence of $\text{ADD}_{r,c}$ operations to A_1 followed by some sequence of SWAP_r and SWAP_c operations. In Section 3.2, one simple idea is presented for reducing the number of matrices A_1 and A_2 that are produced in the first step of our enumeration in which the rows and columns of A_1 and A_2 are *not* permutations of one another. Further methods for reducing the number of different matrices produced in the first step of our enumeration are given in later chapters once our methods for finding the list L have been introduced.

3.1 Ordering the Rows and Columns of A

Let $G = [I_k|A]$ be a generator matrix for a $(2k, k)$ self-dual code C. As we saw in the last chapter, any permutation of the rows and/or columns of A results in a matrix A' such that $G' = [I_k|A']$ generates a code equivalent to C. Since there are $(k!)^2$ permutations

of the rows and columns of A, there may exist a large number of matrices $A' \neq A$ in which $G' = [I_k|A']$ generates a code equivalent to C and the rows and columns of A' are permutations of the rows and columns of A. Therefore, if we *order* the rows and columns of the matrices produced in the first step of our enumeration, the number of matrices A_1 and A_2 produced, in which the rows and columns of A_1 and A_2 are permutations of one another, may be reduced.

Let A be any $k \times k$ matrix such that $G = [I_k|A]$ generates a $(2k, k)$ self-dual code. In this section, two algorithms are given that order the rows and columns of A, producing a matrix A'. The first algorithm, called the *QuickOrder* algorithm, orders the rows and columns of any A in *descending* order. The algorithm runs in polynomial time with respect to the number of rows in A but does not guarantee that the A' produced is unique. That is, given any pair of matrices whose rows and columns are permutations of one another, the matrices that result from applying the *QuickOrder* algorithm to each are not necessarily equal. The second algorithm, called the *UniqueOrder* algorithm, also orders the rows and columns of A in descending order, but it guarantees that the A' produced is unique.

In later chapters, algorithms are developed (based on the structures in the input matrix) that permute a subset of $i-1$ rows of the input matrix, $1 \leq i \leq k+1$, to the top of the input matrix and then orders them.[1] These algorithms will result in a matrix in which the first $i-1$ rows and the first $i-1$ components of the columns of the matrix are in descending order. The remaining $k-(i-1)$ rows and $k-(i-1)$ components of the columns are then ordered by either the *QuickOrder* or *UniqueOrder* algorithms. Therefore, both the *QuickOrder* and *UniqueOrder* algorithms are designed to allow for some preprocessing of the input matrix in which the the first $i-1$ rows of the input matrix are predetermined, $1 \leq i \leq k+1$. In this case, the *QuickOrder* and *UniqueOrder* algorithms will only consider permutations of the last $k-(i-1)$ rows of

[1] These algorithms are used to help find a unique ordering of the rows and columns of a given A without having to perform all the permutations that the *UniqueOrder* algorithm, on its own, would perform.

the input matrix. Both algorithms will still order the columns of the input matrix in descending order. However, since the first $i-1$ rows of the input matrix will already be in descending order, the algorithms will only order the last $k-(i-1)$ rows of the input matrix in descending order. The first row considered by the *QuickOrder* and *UniqueOrder* algorithms (i.e. row i) will be referred to as the *starting row* of the algorithms. Note that if our starting row is $i = k+1$ then the algorithms will do nothing.

Let us now define precisely the properties of the matrices that are output by each of the *QuickOrder* and *UniqueOrder* algorithms. We will begin by defining precisely what is meant by the rows and columns of a matrix A being in descending order. For this, we require the following definition:

Definition 3.1 Let $G = [I_k|A]$ be a generator matrix for a $(2k, k)$ self-dual code C. Let i_0 and i_1 be any 2 rows in A. Then row i_0 is said to be *larger than* row i_1 if, from left to right, in the first column in which they differ row i_0 has a value of 1. Let j_0 and j_1 be any 2 columns in A. Then column j_0 is said to be *larger than* column j_1 if, from top to bottom, in the first row in which they differ column j_0 has a value of 1.

Example 3.2 Let C be the $(8,4)$ self-dual code generated by $G = [I_4|A]$, where:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Row 1 of A is larger than rows 2, 3, and 4, row 2 of A is larger than rows 3 and 4, and row 3 of A is larger than row 4. Similarly, column 1 of A is larger than columns 2, 3, and 4, column 2 of A is larger than columns 3 and 4, and column 3 of A is larger than column 4.

The rows of A are said to be in *descending order* if for any pair of rows indexed by i_0 and i_1, if $i_0 < i_1$ then row i_0 is larger than row i_1. Similarly, the columns of A are

said to be in *descending order* if for any pair of columns indexed by j_0 and j_1, if $j_0 < j_1$ then column j_0 is larger than column j_1. For example, the rows and columns of the matrix A in Example 3.2 are both in descending order. Given any A and a starting row i, $1 \leq i \leq k+1$, our *QuickOrder* algorithm will produce an A' in which both the last $k-(i-1)$ rows and the columns of A' are in descending order. Provided the first $i-1$ components of the columns of the input matrix A are in descending order, the first $i-1$ rows of the matrix A' output by the *QuickOrder* algorithm will be equal to the first $i-1$ rows of A. Note that if $i = 1$ then the *QuickOrder* algorithm will produce a matrix A' in which the rows and the columns of A' are both in descending order.

For a given A and starting row i there may exist more than one matrix A' in which both the last $k-(i-1)$ rows and the columns of A' are in descending order. That is, there may exist matrices A_1 and A_2, whose rows and columns are permutations of the rows and columns of A, such that $A_1 \neq A_2$ and both the last $k-(i-1)$ rows and the columns of both A_1 and A_2 are in descending order. Therefore, in order to define the unique A' our *UniqueOrder* algorithm will find, we require the following definition:

Definition 3.3 Let $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ be any two matrices that generate equivalent $(2k,k)$ self-dual codes. Then A_1 is said to be *larger than* A_2 if, from top to bottom, the first row in which they differ is larger in A_1.

Example 3.4 Let C_1 and C_2 be the equivalent $(8,4)$ self-dual codes, generated by $G_1 = [I_4|A_1]$ and $G_2 = [I_4|A_2]$ respectively, where:

$$A_1 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Row 3 is the first row in which they differ, and since row 3 of A_1 is larger than row 3 of A_2, A_1 is larger than A_2.

Given any A and a starting row i, $1 \leq i \leq k+1$, our *UniqueOrder* algorithm will produce the unique A' with both the last $k-(i-1)$ rows and the columns in descending order, such that A' is larger than or equal to any other A'' (whose last $k-(i-1)$ rows and columns are permutations of the last $k-(i-1)$ rows and the columns of A) in which both the last $k-(i-1)$ rows and the columns of A'' are in descending order. Provided the first $i-1$ components of the columns of A are in descending order, the first $i-1$ rows of the matrix A' output by the *UniqueOrder* algorithm will be equal to the first $i-1$ rows of A. Note that if $i=1$ then the *UniqueOrder* algorithm will produce the largest matrix A' in which all the rows and the columns of A' are both in descending order.

Let us now turn our attention to how these algorithms work. We will begin by giving an algorithm that performs just the permutations of the last $k-(i-1)$ rows and the columns of A that results in a matrix A' whose columns are in descending order. The algorithm is designed in such a way that we can easily modify it to avoid performing all permutations that result in an A' whose last $k-(i-1)$ rows are not in descending order, and avoid some of the permutations that result in an A' that cannot be the largest.

Our algorithm for performing all permutations of the last $k-(i-1)$ rows and the columns of A that results in an A' whose columns are in descending order is based on the following: Let A be any $k \times k$ matrix such that $G=[I_k|A]$ generates a $(2k,k)$ self-dual code. Since G generates a self-dual code, the columns of A have odd weight and intersect in an even number of positions (Theorems 2.1 and 2.2 of Chapter 2). Therefore, the columns of A must be distinct (if A had two equal columns then they would intersect in an odd number of positions, which is a contradiction). What this tells us is that for any matrix A'' whose rows are a permutation of the rows of A, there exists one and only one permutation of the columns of A'' that results in a matrix A' whose columns are in descending order (i.e. the j^{th} largest column of A'' is the j^{th} column of A').

Example 3.5 Consider the $(12,6)$ self-dual code generated by $G = [I_6|A]$ where:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{matrix} \\ \left[\begin{array}{cccccc} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{matrix}$$

Let A'' be the matrix that results from performing the permutation $\pi_r = (1\ 3\ 4)(2\ 6\ 5)$ on the rows of A. Then $\pi_c = (1\ 2\ 6\ 3\ 4)$ is the only permutation of the columns of A'' that results in a matrix A' whose columns are in descending order. That is, π_c is the permutation of the columns of A'' that permutes the largest column of A'' to the first column of A', the second largest column of A'' to the second column of A', and so on.

$$A'' = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{matrix} \\ \left[\begin{array}{cccccc} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_3 \\ r_6 \\ r_4 \\ r_1 \\ r_2 \\ r_5 \end{matrix} \qquad A' = \begin{array}{c} \begin{matrix} c_2 & c_6 & c_4 & c_1 & c_5 & c_3 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_3 \\ r_6 \\ r_4 \\ r_1 \\ r_2 \\ r_5 \end{matrix}$$

Note that while in this case the rows of the resulting matrix A' are also in descending order, this is not generally the case. (However, as our algorithms will demonstrate, for a given A and starting row i, there exists at least one A' in which both the last $k-(i-1)$ rows and the columns of A' are in descending order).

Since for any permutation of the rows of A there exists one and only one permutation of the columns that results in an A' whose columns are in descending order, finding all

permutations of the last $k-(i-1)$ rows and the columns of A, that result in an A' whose columns are in descending order, is simply a matter of performing each of the $(k-(i-1))!$ permutations of the last $k-(i-1)$ rows of A, and for each permutation of the rows, ordering the columns in descending order.

Our algorithm for performing all permutations of the last $k-(i-1)$ rows and ordering the columns of A into descending order will perform the $(k-(i-1))!$ permutations of the last $k-(i-1)$ rows of A *row by row*. That is, we will perform each permutation π_r of the last $k-(i-1)$ rows of A as a sequence of $k-(i-1)-1$ transpositions $(k-1\ r_{k-1})\cdots(i+1\ r_{i+1})(i\ r_i)$ where $m \leq r_m \leq k$. (Recall our convention that products of permutations are applied from right to left). For each π_r, we will also order the columns of the matrix $\pi_r(A)$ in descending order *row by row*. That is, after each transposition $(m\ r_m)$ is performed, the first m components of the columns will be ordered in descending order.

Example 3.6 Let C be the $(8,4)$ self-dual code generated by $G = [I_4|A]$ where:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{matrix}$$

If $i = 1$ is the starting row, then the algorithm will perform the last $(k-(i-1))! = 4!$ permutations of the 4 rows of A row by row, and for each permutation of the rows of A, the columns will be ordered in descending order, row by row. That is, for each permutation $\pi_r = (3\ r_3)(2\ r_2)(1\ r_1)$ of the rows of A, the algorithm will first swap rows 1 and r_1 of A, producing the matrix A'_0, and then the first component of the columns of A'_0 will be ordered in descending order, producing A_1. The algorithm will then swap rows 2 and r_2 of A_1, producing the matrix A'_1, and then the first 2 components of the columns of A'_1 will be ordered in descending order, producing A_2, and so on.

For example, for the permutation $\pi_r = (3\ 3)(2\ 3)(1\ 4)$ of the rows of A, the algorithm will apply π_r to A, row by row, and order the columns of $\pi_r(A)$ in descending order, row by row, as follows: The algorithm first swaps rows 1 and 4 of A, producing the matrix A'_0, and then orders the first component of the columns of A'_0 in descending order, producing the matrix A_1:

$$A'_0 = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} r_4 \\ r_2 \\ r_3 \\ r_1 \end{matrix} \qquad A_1 = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_4 & c_3 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} r_4 \\ r_2 \\ r_3 \\ r_1 \end{matrix}$$

The algorithm then swaps rows 2 and 3 of A_1, producing the matrix A'_1, and then orders the first 2 components of the columns of A'_1 in descending order, producing the matrix A_2:

$$A'_1 = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_4 & c_3 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} r_4 \\ r_3 \\ r_2 \\ r_1 \end{matrix} \qquad A_2 = \begin{array}{c} \begin{matrix} c_1 & c_4 & c_2 & c_3 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_4 \\ r_3 \\ r_2 \\ r_1 \end{matrix}$$

Since the next transposition is (3 3), the algorithm simply sets A'_2 equal to A_2 and then orders the first 3 components of the columns of A'_2 in descending order, producing the matrix A_3:

$$A'_2 = \begin{array}{c} \begin{matrix} c_1 & c_4 & c_2 & c_3 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_4 \\ r_3 \\ r_2 \\ r_1 \end{matrix} \qquad A_3 = \begin{array}{c} \begin{matrix} c_4 & c_1 & c_2 & c_3 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_4 \\ r_3 \\ r_2 \\ r_1 \end{matrix}$$

The resulting matrix A_3 is equal to the matrix $\pi_r(A)$ with its columns ordered in descending order. Notice that ordering the first 3 components of the columns of A'_2 was

sufficient to order all the columns of A'_2 in descending order.

Notice also that since the first $m-1$ components of the columns of each matrix A_{m-1} we produce are ordered in descending order, the first $m-1$ components of the columns of the matrix A'_{m-1} produced by swapping rows m and r_m of A_{m-1}, where $m \leq r_m \leq k$, are also in descending order. Therefore, when we order the first m components of the columns of A'_{m-1}, producing A_m, the first $m-1$ rows (i.e. the first $m-1$ components of the columns) of A_{m-1} and A_m will be equal. Therefore, what we are really doing when we order the first m components of the columns of A'_{m-1} in descending order is, essentially, *ordering the columns* of row m in A'_{m-1}.

So, before giving a formal description of our algorithm for finding all permutations of the last $k-(i-1)$ rows and ordering the columns of A into descending order, we will consider the problem of *ordering the columns* of row m of the matrix A'_{m-1}, producing the matrix A_m. The first thing to notice is that since the first $m-1$ components of A'_{m-1} and A_m will be equal, the relative position of any two columns in A'_{m-1} that differ in their first $m-1$ components will not change. Therefore, the first m components of the columns of A'_{m-1} can be ordered by a set of disjoint permutations where each permutation only permutes columns of A'_{m-1} that are equal in their first $m-1$ components. Furthermore, since the first $m-1$ components of A'_{m-1} are ordered in descending order, any subset of columns that are equal in their first $m-1$ components occur in consecutive columns of A'_{m-1}. Such a subset of columns shall be referred to as a *column block* in the first $m-1$ rows of A'_{m-1}. More formally, we have the following definition:

Definition 3.7 A *column block* in the first $m-1$ rows of A'_{m-1} is a set of consecutive columns $j, j+1, \ldots, j+c$ such that:

- The first $m-1$ components of each of the columns $j, j+1, \ldots, j+c$ are equal,
- if $j > 1$ then the first $m-1$ components of j and $j-1$ are not equal, and

- if $j + c < k$ then the first $m - 1$ components of $j + c$ and $j + c + 1$ are not equal.

The set of all column blocks in the first $m - 1$ rows of A'_{m-1} will be denoted by $B_{m-1,1}$, $B_{m-1,2}, \ldots, B_{m-1,y}$ with the convention that the columns of $B_{m-1,x}$ occur before the columns of $B_{m-1,x+1}$ in A'_{m-1}, $1 \leq x < y$.

If $m = 1$ then $B_{0,1}$ is defined as the set of all k columns of A'_0.

Example 3.8 Consider the following 7×7 matrix:

$$A'_2 = \left[\begin{array}{ccccc|c|c} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

the column blocks in the first 2 rows of A'_2 are:

$$\begin{aligned} B_{2,1} &= \{1, 2, 3, 4\} \\ B_{2,2} &= \{5\} \\ B_{2,3} &= \{6\} \\ B_{2,4} &= \{7\} \end{aligned}$$

Since the relative position of any two columns in different column blocks of A'_{m-1} will not change, ordering the columns of row m of A'_{m-1} is simply a matter of ordering the columns of each column block in the first $m - 1$ rows of A'_{m-1}. For each column block $B_{m-1,x}$ in A'_{m-1}, all we need to do to order the columns of $B_{m-1,x}$ is to permute the subset of columns in $B_{m-1,x}$ that contain a value of 1 in row m to the left of the columns in $B_{m-1,x}$ that contain a value of 0 in row m.

Example 3.9 Consider the following 7×7 matrix:

$$A'_2 = \left[\begin{array}{ccccc|c|c} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Notice that the first 2 components of the columns of A'_2 are in descending order. If for each column block $B_{2,x}$ in the first 2 rows of A'_2, we permute the columns of $B_{2,x}$ that contain a 1 in row 3 to the left of the columns of $B_{2,x}$ that contain a 0 in row 3 then the first 3 components of the columns of the resulting matrix A_3 will be in descending order. In this case, swapping columns 3 and 4 of A'_2 accomplishes this:

$$A_3 = \left[\begin{array}{cccc|c|c|c} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

The first 3 components of the columns of A_3 are in descending order.

This leads us to the following algorithm for ordering row m of each matrix A'_{m-1} we produce:

Algorithm 3.10 *OrderMatrixRow*(A'_{m-1}, m, A_m):

- Input: a $k \times k$ matrix A'_{m-1} and an integer m, $1 \leq m \leq k$. It is assumed that the first $m-1$ components of the columns of A'_{m-1} are in descending order.

- Output: a matrix A_m which is A'_{m-1} with the first m components of its columns ordered in descending order.

begin

Let $A_m = A'_{m-1}$.

Let $B_{m-1,1}, B_{m-1,2}, \ldots, B_{m-1,y}$ be the column blocks in the first $m-1$ rows of A'_{m-1}.

for $x = 1$ to y **do**

Permute the columns in $B_{m-1,x}$ that have a value of 1 in row m to the left of the columns if $B_{m-1,x}$ that have a value of 0 in row m.

end for

end

We are now ready to give a formal description of our row by row method for performing all of the permutations of the last $k-(i-1)$ rows and ordering the columns of A into descending order. The algorithm is recursive and is begun with the call *AllPermutations*(A, i, L) where $1 \leq i \leq k+1$ and $L = \emptyset$. It assumes the first $i-1$ components of the columns of A have already been ordered in descending order.

Algorithm 3.11 *AllPermutations*(A_{m-1}, m, L):

- Input: a $k \times k$ matrix A_{m-1} and an integer m, $1 \leq m \leq k+1$. It is assumed that the first $m-1$ components of the columns of A_{m-1} are in descending order.
- Output: a list L of all matrices A', whose rows and columns are permutations of the last $k-(m-1)$ rows and the columns of A_{m-1}, such that the columns of A' are in descending order.

begin

if $m \geq k$ **then begin**

Let $A' = A_{m-1}$.

Insert A' into L.

end if

else begin

for $r = m$ to k **do begin**

$A'_{m-1} = \text{SWAP}_r(m, r)(A_{m-1})$. { i.e. transpose rows m and r of A_{m-1}. }

OrderMatrixRow(A'_{m-1}, m, A_m).

AllPermutations$(A_m, m+1, L)$.

end for

end else

end

The algorithm does not have to order the columns of the last row of each A' inserted into L since the first $k-1$ components of the columns of A' must be distinct (if two columns in A' were equal in the first $k-1$ components then, since they both have odd weight, they must also be equal in the k^{th} component, which contradicts the columns of A' being distinct).

The advantage in using the row by row approach of Algorithm 3.11 is that if an A_m is produced in which rows $i, i+1, \ldots m$ of A_m are not in descending order, then we can stop the recursion because the algorithm will never produce, from A_m, the required matrix whose last $k-(i-1)$ rows and columns are in descending order.

The *UniqueOrder* and *QuickOrder* algorithms also perform permutations of the last $k-(i-1)$ rows and the columns of A, row by row. The *UniqueOrder* algorithm operates in the same fashion as Algorithm 3.11 with the difference that for each input matrix A_{m-1} only a subset of the transpositions $\text{SWAP}_r(m, r)$ are tried, and for each matrix A_m produced the recursion may or may not continue. A transposition is *not* tried only if it can be determined that *none* of the permutations of the last $k-m$ rows (and the columns)

of the resulting matrix will lead us to the largest matrix A' whose last $k-(i-1)$ rows and columns are in descending order. For each A_m produced, the recursion continues only if rows $i, i+1, \ldots, m$ of A_m are larger than or equal to rows $i, i+1, \ldots, m$ of the largest matrix A' found thus far. Another difference between the *UniqueOrder* algorithm and Algorithm 3.11 is that instead of recording every matrix A' produced in a list L, the *UniqueOrder* algorithm maintains a single matrix A_{max} which, at any point during the execution of the algorithm, is equal to the largest matrix A' found thus far in which both the last $k-(i-1)$ rows and the columns of A' are in descending order. Once the algorithm completes, A_{max} will be equal to the largest A' in which both the last $k-(i-1)$ rows and the columns of A' are in descending order. The *QuickOrder* algorithm simply performs the first permutation of the last $k-(i-1)$ rows of A that the *UniqueOrder* algorithm executes.

Another way to state the difference between the *UniqueOrder* and *QuickOrder* algorithms is that the *QuickOrder* algorithm is a greedy algorithm; i.e. the algorithm chooses the first available option, whereas the *UniqueOrder* algorithm chooses every option. It is well-known that greedy algorithms are not guaranteed to find optimal answers except in special circumstances, but they are very fast. Programs that try every option to find optimal answers but are very slow.

Both the *UniqueOrder* and the *QuickOrder* algorithms are based on the following observation: Suppose that A_{m-1} is a matrix with the following three properties:

1. The first $m-1$ components of the columns of A_{m-1} are in descending order.
2. Rows $i, i+1, \ldots, m-1$ are in descending order.
3. Row m is larger than each of the last $k-(m-1)$ rows of A_{m-1}.

For $r = m, m+1, \ldots, k$, let $A'_{m-1}(r)$ denote the matrix that results from swapping rows m and r of A_{m-1}, and let $A_m(r)$ denote the matrix that results from ordering row m of $A'_{m-1}(r)$. Let S_m denote the set of rows r, $m \leq r \leq k$, such that for any row

r_1, $m \leq r_1 \leq k$, row m of $A_m(r)$ is larger than or equal to row m of $A_m(r_1)$. (Note that S_m is not the empty set). Then for any $r \in S_m$, the matrix $A_m(r)$ has the same three properties as A_{m-1}, but in its first m rows. That is, the first m components of the columns of $A_m(r)$ are in descending order, rows $i, i+1, \ldots, m$ of $A_m(r)$ are in descending order, and row m of $A_m(r)$ is larger than each of its last $k-m$ rows.

Example 3.12 Consider the following 7×7 matrix:

$$A_3 = \left[\begin{array}{ccc|c|ccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Notice that the first 3 components of the columns of A_3 are in descending order, rows 1, 2, and 3 of A_3 are in descending order, and row 3 of A_3 is larger than each of the last 4 rows of A_3. For $r = 4, 5, 6, 7$, if we swap rows 4 and r of A_3, producing the matrix $A_3'(r)$, and then order row 4 of $A_3'(r)$, the matrices $A_4(r)$ we produce are:

$$A_4(4) = \left[\begin{array}{ccc|c|ccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \qquad A_4(5) = \left[\begin{array}{ccc|c|ccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$A_4(6) = \left[\begin{array}{ccc|c|c|c|c} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \qquad A_4(7) = \left[\begin{array}{ccc|c|c|c|c} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Row 4 in each of $A_4(4)$, $A_4(6)$, and $A_4(7)$ are equal and each is larger than row 4 of $A_4(5)$. Therefore, the set $S_4 = \{4, 6, 7\}$. Note that the first 4 components of the columns of each of $A_4(4)$, $A_4(6)$, and $A_4(7)$ are in descending order, the first 4 rows of each are in descending order, and row 4 of each is larger than each of their last 3 rows.

Since for any A_{m-1} with the three stated properties in its first $m-1$ rows, for any $r \in S_m$ the resulting matrix $A_m(r)$ has the same three properties but in its first m rows, there must exist a permutation of the last $k-m$ rows (and columns) of $A_m(r)$ that results in a matrix A' that has the three stated properties in all of its rows. In other words, there exists a permutation of the last $k-m$ rows (and the columns) of $A_m(r)$ that results in a matrix A' such that both the last $k-(i-1)$ rows and the columns of A' are in descending order. Furthermore, since for any $r_1 \notin S_m$, row m of $A_m(r)$ is larger than row m of $A_m(r_1)$, no permutations of the last $k-m$ rows (and columns) of $A_m(r_1)$ will result in a matrix A'' (with columns in descending order) such that A'' is larger than or equal to A'. These observations form the basis of the *UniqueOrder* and *QuickOrder* algorithms.

For each input matrix A_{m-1} to a recursive call of the *UniqueOrder* algorithm, the subset of transpositions $\text{SWAP}_r(m, r)$, $m \leq r \leq k$, that the *UniqueOrder* algorithm will try is the subset of transpositions $\text{SWAP}_r(m, r)$ in which $r \in S_m$. The *QuickOrder* algorithm works by simply selecting one $r \in S_m$ at each step. Thus, both the *QuickOrder* and *UniqueOrder* algorithms are dependent on finding the set of rows S_m. So, before giving a formal description of both the *QuickOrder* and *UniqueOrder* algorithms, we will

consider the problem of finding the subset of rows S_m in the last $k - m$ rows of A_{m-1}. For this, we require the following definition:

Definition 3.13 Let A_{m-1} be any matrix in which the first $m-1$ components of the columns of A_{m-1} are in descending order, $1 \leq m \leq k$. Let $B_{m-1,1}, B_{m-1,2}, \ldots, B_{m-1,y}$ denote the column blocks in the first $m-1$ rows of A_{m-1}. Then the *column block sequence* of row r, $m \leq r \leq k$, with respect to the first $m-1$ rows of A_{m-1}, is the sequence of integers $(c_{m-1,r,1}, c_{m-1,r,2}, \ldots, c_{m-1,r,y})$, where $c_{m-1,r,x}$ is the number of columns in $B_{m-1,x}$ in which row r of A_{m-1} has a value of 1. The column block sequence of row r, with respect to the first $m-1$ rows of A_{m-1}, will be denoted by $b_{m-1,r}$.

If $m = 1$, then the column block sequence $b_{0,r} = (w)$, where w is the weight of row r in A_{m-1}.

Example 3.14 For the $(14, 7)$ self-dual code generated by $G = [I_7 | A_3]$, where:

$$A_3 = \left[\begin{array}{ccc|c|ccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

The column block sequences, with respect to the first 3 rows of A_3, for each of the last 4 rows of A_3 are:

$$\begin{aligned} b_{3,4} &= (2,0,0,0,1) \\ b_{3,5} &= (0,1,1,1,0) \\ b_{3,6} &= (2,0,0,0,1) \\ b_{3,7} &= (2,0,0,0,1) \end{aligned}$$

For any 2 column block sequences b_{m-1,r_0} and b_{m-1,r_1} ($b_{m-1,r_0} \neq b_{m-1,r_1}$), we will say that b_{m-1,r_0} is *larger than* b_{m-1,r_1} if for the first x in which $c_{m-1,r_0,x} \neq c_{m-1,r_1,x}$ we have $c_{m-1,r_0,x} > c_{m-1,r_1,x}$. For example, the column block sequence $b_{3,4}$ in Example 3.14 is larger than the column block sequence $b_{3,5}$.

What the column block sequence $b_{m-1,r} = (c_{m-1,r,1}, c_{m-1,r,2}, \ldots, c_{m-1,r,y})$ of a row r in A_{m-1} tells us is that if we swap rows m and r of A_{m-1}, producing $A'_{m-1}(r)$, and then order row m of $A'_{m-1}(r)$, producing $A_m(r)$, then row m of $A_m(r)$ will be:

$$1^{a_1}0^{d_1}\ 1^{a_2}0^{d_2} \ldots 1^{a_y}0^{d_y},$$

where $a_x = c_{m-1,r,x}$ and $d_x = |B_{m-1,x}| - c_{m-1,r,x}$. Therefore, if for any two rows r_0 and r_1 with column blocks sequences b_{m-1,r_0} and b_{m-1,r_1}, respectively, if b_{m-1,r_0} is larger than b_{m-1,r_1} then row m of $A_m(r_0)$ will be larger than row m of $A_m(r_1)$. Thus, the subset S_m of the last $k - m$ rows of A_{m-1} is equal to the set of rows r, $m \leq r \leq k$, with the largest column block sequences. This leads us to the following algorithm for finding the set of rows S_m:

Algorithm 3.15 *FindLargestRemainingRows*(A_{m-1}, m, S_m):

- Input: a $k \times k$ matrix A_{m-1} and an integer m, $1 \leq m \leq k$. It is assumed that the first $m - 1$ components of the columns of A_{m-1} are in descending order.
- Output: the set S_m of row indices r, $m \leq r \leq k$, with the largest column block sequences with respect to the first $m - 1$ rows of A_{m-1}.

begin

for $r = m$ to k **do**

Compute the column block sequence $b_{m-1,r}$ for row r.

Let $S_m = \{m\}$.
for $r = m + 1$ to k **do begin**
Let r_0 be any row index in S.
if $b_{m-1,r} > b_{m-1,r_0}$ **then**
$S_m = \{r\}$.
else if $b_{m-1,r} = b_{m-1,r_0}$ **then**
$S_m = S_m \cup r$.
end for
end

We are now ready to give a formal description of both the *QuickOrder* and *Unique-Order* algorithms.

Given any A and starting row i, $1 \leq i \leq k + 1$, the *QuickOrder* algorithm permutes the last $k - (i - 1)$ rows and the columns of A, row by row, producing a matrix A' in which both the last $k - (i - 1)$ rows and the columns of A' are in descending order. The algorithm assumes that the first $i - 1$ components of the columns of A have already been ordered in descending order.

Algorithm 3.16 *QuickOrderMatrix*(A, i, A'):

- Input: a $k \times k$ matrix A and an integer i, $1 \leq i \leq k + 1$. It is assumed the first $i - 1$ components of the columns of A are in descending order.
- Output: a $k \times k$ matrix A', whose rows and columns are permutations of the rows and columns of A, such that the first $i - 1$ rows of A' are equal to the first $i - 1$ rows of A, and both the last $k - (i - 1)$ rows and the columns of A' are in descending order.

begin

Let $A' = A$.

for $m = i$ to k **do begin**

FindLargestRemainingRows(A', m, S)

Let r be the smallest row index in S.

$A' = \text{SWAP}_r(m, r)(A')$.

OrderMatrixRow(A', m, A').

end for

end

Given any A and starting row i, $1 \leq i \leq k+1$, the *UniqueOrder* algorithm finds the largest matrix A' (which we denote by A_{max}), whose rows and columns are a permutation of the rows and the columns of A, such that the last $k-(i-1)$ rows and the columns of A_{max} are in descending order. The algorithm is recursive and is invoked with the call *UniqueOrder*$(A, i, A_{max}, true)$, where A_{max} is initially undefined. The algorithm assumes that the first $i-1$ components of the columns of A have already been ordered in descending order. The boolean input variable *larger* is used to aid us in determining whether a matrix A_m has been produced that is smaller than the largest A_{max} found thus far. The variable *larger* has a value of true whenever the first $m-1$ rows of the input matrix A_{m-1} are larger than the first $m-1$ rows of the largest A_{max} seen thus far, otherwise *larger* has a value of false. What is meant by "the first $m-1$ rows of A_{m-1} are larger than the first $m-1$ rows of A_{max}" is that if row r is the first row in which they differ, $i \leq r \leq m-1$, then row r of A_{m-1} is larger than row r of A_{max}.

Algorithm 3.17 *UniqueOrderMatrix*$(A_{m-1}, m, A_{max}, larger)$:

- Input: the $k \times k$ matrix A_{m-1}, the integer m, $i \leq m \leq k+1$, and the boolean *larger*.
- Output: the $k \times k$ matrix A_{max}. A_{max} is the largest matrix whose rows and columns are permutations of the rows and columns of A_{m-1}, such that the first $m-1$ rows

of A_{max} and A_{m-1} are equal, and, both the last $k-(m-1)$ rows and the columns of A_{max} are in descending order.

begin
  if $m \geq k$ **then**
    if (A_{max} is undefined) **or** ($A_{m-1} > A_{max}$) **then**
      $A_{max} = A_{m-1}$.
  else begin
    FindLargestRemainingRows(A_{m-1}, m, S_m).
    for each $r \in S_m$ **do begin**
      $A'_{m-1} = \mathrm{SWAP}_r(m, r)(A_{m-1})$.
      OrderMatrixRow(A'_{m-1}, m, A_m).
      if *larger* **or** (row m of A_m) > (row m of A_{max}) **then**
        UniqueOrderMatrix$(A_m, m+1, A_{max}, true)$.
      else if (row m of A_m) = (row m of A_{max}) **then**
        UniqueOrderMatrix$(A_m, m+1, A_{max}, false)$.
    end for
  end else
end

We will conclude this section with examples that demonstrate how both the *QuickOrder* and *UniqueOrder* algorithms work.

Example 3.18 For this example, we will apply the *QuickOrder* algorithm (with starting row 1) to the following 11×11 matrix A_0. Note that $G_0 = [I_{11}|A_0]$ generates a $(22, 11)$

self-dual code.

$$A_0 = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} \end{matrix} \\ \left[\begin{array}{ccccccccccc} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \\ r_{11} \end{matrix} \end{array}$$

Iteration 1: The algorithm first calls *FindLargestRemainingRows*$(A_0, 1, S)$. Since every row in A_0 has weight 5, the set returned is $S = \{1,2,3,4,5,6,7,8,9,10,11\}$. The algorithm then selects row 1 from S, swaps it with row 1, and then orders the columns of row

1, producing the matrix:

c_3	c_4	c_5	c_8	c_{11}	c_6	c_7	c_2	c_9	c_{10}	c_1	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	0	0	0	1	1	0	0	0	1	1	r_2
0	1	0	1	0	1	0	1	0	1	0	r_3
0	0	1	0	1	0	0	1	1	1	0	r_4
0	0	1	1	0	1	0	0	1	0	1	r_5
0	1	0	0	1	1	1	0	1	0	0	r_6
0	0	0	1	1	0	1	1	0	0	1	r_7
1	0	1	0	0	1	1	1	0	0	0	r_8
1	0	0	1	0	0	1	0	1	1	0	r_9
1	1	0	0	0	0	0	1	1	0	1	r_{10}
0	1	1	0	0	0	1	0	0	1	1	r_{11}

Iteration 2: $S = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$. The algorithm selects row 2 from S, swaps it with row 2, and then orders the columns of row 2, producing the matrix:

c_3	c_{11}	c_5	c_8	c_4	c_6	c_{10}	c_1	c_9	c_7	c_2	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
0	0	0	1	1	1	1	0	0	0	1	r_3
0	1	1	0	0	0	1	0	1	0	1	r_4
0	0	1	1	0	1	0	1	1	0	0	r_5
0	1	0	0	1	1	0	0	1	1	0	r_6
0	1	0	1	0	0	0	1	0	1	1	r_7
1	0	1	0	0	1	0	0	0	1	1	r_8
1	0	0	1	0	0	1	0	1	1	0	r_9
1	0	0	0	1	0	0	1	1	0	1	r_{10}
0	0	1	0	1	0	1	1	0	1	0	r_{11}

Iteration 3: $S = \{4, 6, 7, 8, 9, 10\}$. The algorithm selects row 4 from S, swaps it with row 3, and then orders the columns of row 3, producing the matrix:

c_{11}	c_3	c_5	c_8	c_4	c_{10}	c_6	c_1	c_9	c_2	c_7	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
1	0	1	0	0	1	0	0	1	1	0	r_4
0	0	0	1	1	1	1	0	0	1	0	r_3
0	0	1	1	0	0	1	1	1	0	0	r_5
1	0	0	0	1	0	1	0	1	0	1	r_6
1	0	0	1	0	0	0	1	0	1	1	r_7
0	1	1	0	0	0	1	0	0	1	1	r_8
0	1	0	1	0	1	0	0	1	0	1	r_9
0	1	0	0	1	0	0	1	1	1	0	r_{10}
0	0	1	0	1	1	0	1	0	0	1	r_{11}

Iteration 4: $S = \{6, 7\}$. The algorithm selects row 6 from S, swaps it with row 4, and then orders the columns of row 4, producing the matrix:

c_{11}	c_3	c_5	c_4	c_8	c_{10}	c_6	c_1	c_9	c_2	c_7	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
1	0	1	0	0	1	0	0	1	1	0	r_4
1	0	0	1	0	0	1	0	1	0	1	r_6
0	0	1	0	1	0	1	1	1	0	0	r_5
0	0	0	1	1	1	1	0	0	1	0	r_3
1	0	0	0	1	0	0	1	0	1	1	r_7
0	1	1	0	0	0	1	0	0	1	1	r_8
0	1	0	0	1	1	0	0	1	0	1	r_9
0	1	0	1	0	0	0	1	1	1	0	r_{10}
0	0	1	1	0	1	0	1	0	0	1	r_{11}

Notice that the column blocks now all consist of a single column. Therefore, the order of the columns of each of the remaining 7 rows will not change and thus the algorithm will, essentially, just order the remaining 7 rows in descending order:

c_{11}	c_3	c_5	c_4	c_8	c_{10}	c_6	c_1	c_9	c_2	c_7	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
1	0	1	0	0	1	0	0	1	1	0	r_4
1	0	0	1	0	0	1	0	1	0	1	r_6
1	0	0	0	1	0	0	1	0	1	1	r_7
0	0	0	1	1	1	1	0	0	1	0	r_3
0	0	1	0	1	0	1	1	1	0	0	r_5
0	1	1	0	0	0	1	0	0	1	1	r_8
0	1	0	0	1	1	0	0	1	0	1	r_9
0	1	0	1	0	0	0	1	1	1	0	r_{10}
0	0	1	1	0	1	0	1	0	0	1	r_{11}

c_{11}	c_3	c_5	c_4	c_8	c_{10}	c_6	c_1	c_9	c_2	c_7	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
1	0	1	0	0	1	0	0	1	1	0	r_4
1	0	0	1	0	0	1	0	1	0	1	r_6
1	0	0	0	1	0	0	1	0	1	1	r_7
0	1	1	0	0	0	1	0	0	1	1	r_8
0	0	1	0	1	0	1	1	1	0	0	r_5
0	0	0	1	1	1	1	0	0	1	0	r_3
0	1	0	0	1	1	0	0	1	0	1	r_9
0	1	0	1	0	0	0	1	1	1	0	r_{10}
0	0	1	1	0	1	0	1	0	0	1	r_{11}

c_{11}	c_3	c_5	c_4	c_8	c_{10}	c_6	c_1	c_9	c_2	c_7	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
1	0	1	0	0	1	0	0	1	1	0	r_4
1	0	0	1	0	0	1	0	1	0	1	r_6
1	0	0	0	1	0	0	1	0	1	1	r_7
0	1	1	0	0	0	1	0	0	1	1	r_8
0	1	0	1	0	0	0	1	1	1	0	r_{10}
0	0	0	1	1	1	1	0	0	1	0	r_3
0	1	0	0	1	1	0	0	1	0	1	r_9
0	0	1	0	1	0	1	1	1	0	0	r_5
0	0	1	1	0	1	0	1	0	0	1	r_{11}

c_{11}	c_3	c_5	c_4	c_8	c_{10}	c_6	c_1	c_9	c_2	c_7	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
1	0	1	0	0	1	0	0	1	1	0	r_4
1	0	0	1	0	0	1	0	1	0	1	r_6
1	0	0	0	1	0	0	1	0	1	1	r_7
0	1	1	0	0	0	1	0	0	1	1	r_8
0	1	0	1	0	0	0	1	1	1	0	r_{10}
0	1	0	0	1	1	0	0	1	0	1	r_9
0	0	0	1	1	1	1	0	0	1	0	r_3
0	0	1	0	1	0	1	1	1	0	0	r_5
0	0	1	1	0	1	0	1	0	0	1	r_{11}

c_{11}	c_3	c_5	c_4	c_8	c_{10}	c_6	c_1	c_9	c_2	c_7	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
1	0	1	0	0	1	0	0	1	1	0	r_4
1	0	0	1	0	0	1	0	1	0	1	r_6
1	0	0	0	1	0	0	1	0	1	1	r_7
0	1	1	0	0	0	1	0	0	1	1	r_8
0	1	0	1	0	0	0	1	1	1	0	r_{10}
0	1	0	0	1	1	0	0	1	0	1	r_9
0	0	1	1	0	1	0	1	0	0	1	r_{11}
0	0	1	0	1	0	1	1	1	0	0	r_5
0	0	0	1	1	1	1	0	0	1	0	r_3

c_{11}	c_3	c_5	c_4	c_8	c_{10}	c_6	c_1	c_9	c_2	c_7	
1	1	1	1	1	0	0	0	0	0	0	r_1
1	1	0	0	0	1	1	1	0	0	0	r_2
1	0	1	0	0	1	0	0	1	1	0	r_4
1	0	0	1	0	0	1	0	1	0	1	r_6
1	0	0	0	1	0	0	1	0	1	1	r_7
0	1	1	0	0	0	1	0	0	1	1	r_8
0	1	0	1	0	0	0	1	1	1	0	r_{10}
0	1	0	0	1	1	0	0	1	0	1	r_9
0	0	1	1	0	1	0	1	0	0	1	r_{11}
0	0	1	0	1	0	1	1	1	0	0	r_5
0	0	0	1	1	1	1	0	0	1	0	r_3

The matrix output by the *QuickOrder* algorithm is the following 11×11 matrix A_0'

$$A_0' = \begin{array}{c} \begin{matrix} c_{11} & c_3 & c_5 & c_4 & c_8 & c_{10} & c_6 & c_1 & c_9 & c_2 & c_7 \end{matrix} \\ \left[\begin{array}{ccccccccccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{array}{l} r_1 \\ r_2 \\ r_4 \\ r_6 \\ r_7 \\ r_8 \\ r_{10} \\ r_9 \\ r_{11} \\ r_5 \\ r_3 \end{array}$$

The rows and columns of A_0' are permutations of the rows and columns of A_0. The rows and columns of A_0' are in descending order.

As this next example demonstrates, not all matrices whose rows and columns are permutations of the rows and columns of the matrix A_0 in Example 3.18 will have their rows and columns ordered to the same matrix by the *QuickOrder* algorithm.

Example 3.19 Suppose we swap rows 6 and 7 of the matrix A_0 in Example 3.18, which produces the following 11 x 11 matrix:

$$A_1 = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} \end{matrix} \\ \left[\begin{array}{ccccccccccc} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_7 \\ r_6 \\ r_8 \\ r_9 \\ r_{10} \\ r_{11} \end{matrix} \end{array}$$

If we apply the *QuickOrder* algorithm to this matrix, then the matrix that results is:

$$A_1' = \begin{array}{c} \begin{matrix} c_{11} & c_3 & c_5 & c_8 & c_4 & c_{10} & c_1 & c_6 & c_2 & c_9 & c_7 \end{matrix} \\ \left[\begin{array}{ccccccccccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{array}\right] \begin{matrix} r_1 \\ r_2 \\ r_4 \\ r_7 \\ r_6 \\ r_8 \\ r_9 \\ r_{10} \\ r_5 \\ r_{11} \\ r_3 \end{matrix} \end{array}$$

Notice that the last 6 rows of A_1' are not the same as the last 6 rows of the matrix A_0' produced by the *QuickOrder* algorithm in Example 3.18.

However, for any matrix whose rows and columns are permutations of the rows and columns of the matrix A_0 in Example 3.18, the *UniqueOrder* algorithm will produce a unique matrix A_{max}.

Example 3.20 For any matrix whose rows and columns are a permutation of the rows and columns of the matrix A_0 in Example 3.18 there are exactly 2 different possibilities

for the matrix output by the *QuickOrder* algorithm. They are:

$$A'_0 = \begin{array}{c} \begin{matrix} c_{11} & c_3 & c_5 & c_4 & c_8 & c_{10} & c_6 & c_1 & c_9 & c_2 & c_7 \end{matrix} \\ \left[\begin{matrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_4 \\ r_6 \\ r_7 \\ r_8 \\ r_{10} \\ r_9 \\ r_{11} \\ r_5 \\ r_3 \end{matrix}$$

$$A'_1 = \begin{array}{c} \begin{matrix} c_{11} & c_3 & c_5 & c_8 & c_4 & c_{10} & c_1 & c_6 & c_2 & c_9 & c_7 \end{matrix} \\ \left[\begin{matrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_4 \\ r_7 \\ r_6 \\ r_8 \\ r_9 \\ r_{10} \\ r_5 \\ r_{11} \\ r_3 \end{matrix}$$

Since A'_0 is larger than A'_1, the matrix A_{max} returned by the *UniqueOrder* algorithm is A'_0. During any point of the *UniqueOrder* algorithm, the matrix A_{max} will either be undefined

or will be equal to either A'_0 or A'_1. Once A_{max} has been set to A'_0, it will not change. While the *QuickOrder* algorithm only performs one permutation of the rows of its input matrix, the *UniqueOrder* algorithm will perform all permutations of the rows of its input matrix that lead to A'_0, and anywhere from 0 to all permutations of the rows of its input matrix that lead to A'_1 (depending on when it first finds A'_0). For example, if the matrix A_0 in Example 3.18 is the input matrix then the *UniqueOrder* algorithm will perform all 660 permutations of the rows of A_0 that results in A'_0 and 0 of the permutations of the rows of A_0 that results in A'_1. However, if the matrix A_1 in Example 3.19 is the input matrix then the *UniqueOrder* algorithm will perform all 660 permutations of the rows of A_1 that results in A'_0 and 1 of the permutations of the rows of A_1 that results in A'_1.

Two algorithms have been presented in this section for ordering the rows and columns of any $k \times k$ matrix A, where $G = [I_k|A]$ generates a $(2k, k)$ self-dual code. The *QuickOrder* algorithm orders the rows and columns of A in polynomial time, with respect to the number of rows in A, but does not guarantee that the A' produced is unique. However, if we apply the *QuickOrder* algorithm to every matrix we produce in the first step of our enumeration, we may reduce the number of matrices A_1 and A_2 that are produced in which $A_1 \neq A_2$ and the rows and columns of A_1 are permutations of the rows and columns of A_2. The *UniqueOrder* algorithm guarantees that the matrix it produces is unique but does not have a polynomial running time. However, with sufficient preprocessing of the input matrix (i.e. a sufficient number of the top rows fixed) it too may be efficiently used to reduce the number of different matrices produced in the first step of our enumeration.

3.2 Minimizing The Row Weights of A

As was pointed out in the introduction to this chapter, for any two matrices $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ that generate equivalent $(2k, k)$ self-dual codes, the rows and columns of A_1 are not necessarily permutations of the rows and columns of A_2. In this case, we

can find A_2 by first applying some sequence of $\mathrm{ADD}_{r,c}$ operations to A_1 followed by some sequence of SWAP_r and SWAP_c operations. Therefore, we can reduce the number of matrices A_1 and A_2 produced in the first step of our enumeration, in which the rows and columns of A_1 are *not* permutations of the rows and columns of A_2, by considering properties of any $k \times k$ matrix A that may change when an $\mathrm{ADD}_{r,c}$ operation is applied to A (but does not change when either a SWAP_r or SWAP_c operation is appled to A) and perform $\mathrm{ADD}_{r,c}$ operations on A until a matrix A' is produced with the desired properties. The rows and the columns of the matrix A' produced may then be ordered by either the *QuickOrder* or *UniqueOrder* algorithm.

In this section, an algorithm is given that will allow us to restrict the number of different matrices A_1 and A_2 that are produced in the first step of our enumeration, in which the rows and columns of A_1 are not simply permutations of the rows and columns of A_2. The algorithm is based on the property that for any $k \times k$ matrix A, where $G = [I_k|A]$ generates a $(2k, k, 4)$ self-dual code, the weights of the rows of A may change when an $\mathrm{ADD}_{r,c}$ operation is applied to A, but do not change when either a SWAP_r or SWAP_c operation is applied to A.

Example 3.21 Consider the following 6×6 matrix:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{ll} r_1 & (a_1) \\ r_2 & (a_2) \\ r_3 & (a_3) \\ r_4 & (a_4) \\ r_5 & (a_5) \\ r_6 & (a_6) \end{array}$$

A has four rows with weight 3 and two rows with weight 5. If we apply any SWAP_r or SWAP_c operation to A then the resulting matrix will still consist of four rows with weight 3 and two rows with weight 5. However, if we apply an $\mathrm{ADD}_{r,c}$ operation to A,

the weights of the rows of the resulting matrix may change. For example, consider the matrix A' that results from applying an $\mathrm{ADD}_{r,c}(5,1)$ to A:

$$A' = \begin{array}{c} \begin{matrix} a_5 & c_2 & c_3 & c_4 & c_5 & c_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{ll} r_1 \oplus r_5 & (a_1) \\ r_2 \oplus r_5 & (a_2) \\ r_3 \oplus r_5 & (a_3) \\ r_4 \oplus r_5 & (a_4) \\ r_5 & (c_1) \\ r_6 & (a_6) \end{array}$$

A' consists of six rows, each with weight 5.

Let $G = [I_k|A]$ be a generator matrix for a $(2k,k)$ self-dual code. Let $w_A(x)$ denote the number of rows in A with weight x. The algorithm given in this section for restricting the number of matrices produced in the first step of our enumeration works by applying $\mathrm{ADD}_{r,c}$ operations to A until a matrix A' is found with the property that the row weights of A' are *less than or equal to* the row weights of any matrix A'' that is produced by applying a single $\mathrm{ADD}_{r,c}$ operation to A'. What is meant by the row weights of A' are *less than or equal to* the row weights of A'' is that if x is the smallest weight in which $w_{A'}(x) \neq w_{A''}(x)$ (if such an x exists) then $w_{A'}(x) > w_{A''}(x)$. This process of applying $\mathrm{ADD}_{r,c}$ operations to A until an such an A' is produced shall be referred to as *minimizing the row weights* of A.

Example 3.22 Let us minimize the row weights of the following 6×6 matrix:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

If we apply an $\mathrm{ADD}_{r,c}(1,1)$ to A, then the row weights of the resulting matrix A' are less than the row weights of A:

$$A' = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

If we now apply any single $\mathrm{ADD}_{r,c}$ operation to A', the row weights of the resulting matrix will be greater than or equal to the row weights of A'. Therefore, the process stops and A' is the result of minimizing A.

Clearly this algorithm is a greedy algorithm. As such, the algorithm does not necessarily find the matrix A' with the *minimum* values for the row weights, where A' is obtained from the input matrix A by some sequence of $\mathrm{ADD}_{r,c}$ operations. Our algorithm produces only a matrix A' with row weights that are smaller than or equal to the row weights of any matrix that can be obtained by applying a single $\mathrm{ADD}_{r,c}$ operation to A'. However, there may exist sequences of two or more $\mathrm{ADD}_{r,c}$ operations on A' that result in a matrix that has smaller row weights than A'.

We are now ready to give a formal description of our algorithm for minimizing the row weights of any $k \times k$ matrix A:

Algorithm 3.23 *MinimizeRowWeights*(A, A'):

- Input: a $k \times k$ matrix A such that $G = [I_k|A]$ generates a $(2k, k)$ self-dual code C.
- Output: a matrix A' (where $G' = [I_k|A']$ generates a code equivalent to C) such that the row weights of A' are less than or equal to the row weights of any A'' that results from applying a single $\mathrm{ADD}_{r,c}$ operation to A'.

```
begin
  Set done = false.
  Set A' = A.
  while not done do begin
    Set found = false.
    for i = 1 to k do
      for j = 1 to k do begin
        if entry (i, j) of A' is 1 then begin
          Let A'' = ADD_{r,c}(i, j)(A').
          if the row weights of A'' are less than the row weights of A' then
            Set found = true.
          end if
        end if
        if found then exit for loops.
      end for
    if found then set A' = A''
    else done = true.
  end while
end
```

By applying the *MinimizeRowWeights* algorithm we may reduce the number of matrices A_1 and A_2 that are produced in which the rows and columns of A_1 are not permutations of the rows and columns of A_2, where $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes.

3.3 Summary

Our approach to enumerating a list of inequivalent $(2k, k)$ self-dual codes consists of two main steps. The first step is to produce a list L of $k \times k$ matrices such that for any $(2k, k)$ self-dual code C there exists at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C. For each $A \in L$, the second step is to use the three operations $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c to *order* the matrix A, producing a unique matrix A'. That is, for any pair of matrices $A_1, A_2 \in L$, where $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes C_1 and C_2, respectively, both A_1 and A_2 will be *ordered* to a unique matrix A', where $G' = [I_k|A']$ generates a code that is equivalent to both C_1 and C_2. Since the second step of the enumeration can be quite time consuming, it is desirable that the list L produced in the first step of our enumeration contains as few matrices as possible.

In this chapter, we presented several algorithms that can be used to reduce the number of matrices A_1 and A_2 that may be produced in the first step of the enumeration, where $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes, but $A_1 \neq A_2$.

In Section 3.1 two algorithms were presented for ordering the rows and columns of any $k \times k$ matrix A, where $G = [I_k|A]$ generates a $(2k, k)$ self-dual code. They were the *QuickOrder* and the *UniqueOrder* algorithms. By applying the *QuickOrder* algorithm to every matrix we produce in the first step of our enumeration, we are guaranteed that the rows and columns of every matrix we produce will be in descending order. The *UniqueOrder* algorithm may be used to further reduce the number of different matrices we produce, since it orders the rows and columns of any matrix uniquely.

In Section 3.2 the *MinimizeRowWeights* algorithm was presented for reducing the number of different matrices A_1 and A_2 produced, where $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes and the rows and columns of A_1 and A_2 are not simply permutations of one another. By applying the *MinimizeRowWeights* algorithm to every matrix we produce in the first step of our enumeration, we are guaranteed that every matrix A we produce has the property that no single $\text{ADD}_{r,c}$ operation on A will result in a matrix

whose row weights are less than the row weights of A.

Further methods for reducing the number of matrices produced in the first step of our enumeration are given in later chapters, once our methods for producing the list L have been introduced.

Chapter 4

A Recursive Enumeration of the $(2k, k, 4)$ Self-Dual Codes

In this chapter we will develop an algorithm that finds a list of $(2k, k, 4)$ self-dual codes such that for any $(2k, k, 4)$ self-dual code C there exists at least one code in the list that is equivalent to C. The algorithm is recursive and assumes complete lists of inequivalent $(2j, j, d)$ self-dual codes are available, where $j \leq k-2$ and $d \geq 4$.

In Section 4.1, we will begin by showing that any weight 4 word in a $(2k, k, 4)$ self-dual code generated by a matrix of the form $G = [I_k|A]$ is generated by the rows of G in one of three ways. We will make use of the three different ways a weight 4 word may be generated by $G = [I_k|A]$ throughout the remainder of the thesis. In Section 4.1, we will use the three different ways a weight 4 word may be generated by $G = [I_k|A]$ to develop a recursive algorithm for enumerating the $(2k, k, 4)$ self-dual codes from any list of inequivalent $(2k-4, k-2)$ self-dual codes.

In Sections 4.2 to 4.5, we will work towards improving the algorithm for enumerating the $(2k, k, 4)$ self-dual codes developed in Section 4.1. In Section 4.2, we will have a look at the different ways the weight 4 words of a $(2k, k, 4)$ self-dual code may intersect. In

Section 4.3, we will classify the $(2k, k, 4)$ self-dual codes based on the results of Section 4.2. In Section 4.4, we will use this classification of the $(2k, k, 4)$ self-dual codes to improve the algorithm for enumerating the $(2k, k, 4)$ self-dual codes developed in Section 4.1. These improvements will result in recursive algorithms for enumerating the $(2k, k, 4)$ self-dual codes from any complete list of inequivalent $(2j, j, d)$ self-dual codes, where $j \leq k-2$ and $d \geq 4$. In Section 4.5, we will look at ways in which we can further improve the algorithms developed in Section 4.4.

Based on the algorithms developed in this chapter, programs for enumerating the $(2k, k, 4)$ self-dual codes have been implemented and executed to enumerate lists of $(2k, k, 4)$ self-dual codes for $k \leq 16$. In Section 4.6, we will conclude this chapter by listing the results of the enumeration.

4.1 Enumerating the $(2k, k, 4)$ Self-Dual Codes

In this section, we will first find all the different ways that a weight 4 word may be generated by a matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code. We will then use this result to develop a recursive algorithm that accepts as input a list L_0 of $(k-2) \times (k-2)$ matrices A_0 and that produces as output a list L of $k \times k$ matrices A. The input to the algorithm is any list of L_0 of $(k-2) \times (k-2)$ matrices with the property that, for any $(2k-4, k-2)$ self-dual code C_0, there exists one and only one matrix A_0 in L_0 such that $G_0 = [I_{k-2}|A_0]$ generates a code equivalent to C_0. The output produced by the algorithm is a list L of $k \times k$ matrices with the property that, for any $(2k, k, 4)$ self-dual code C, there exists at least one matrix A in L such that $G = [I_k|A]$ generates a code equivalent to C.

Let us begin by finding all the ways that a weight 4 word may be generated by a matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code. Recall that if $G = [I_k|A]$ generates a $(2k, k)$ self-dual code C then the rows of A have odd weight and intersect in an even number of positions (Theorems 2.1 and 2.2 of Chapter 2).

Furthermore, any weight w word in C is the sum of at most $w-1$ rows of $G=[I_k|A]$ (Theorem 2.8 of Chapter 2). We will use these facts to find all the different ways a weight 4 word may be generated by a matrix $G=[I_k|A]$, where G generates a self-dual code with distance 4.

Our method for finding all the different ways a weight 4 word may be generated by a matrix of the form $G=[I_k|A]$, where G generates a $(2k,k,4)$ self-dual code C, works as follows: Let $\vec{w}$ be any weight 4 word in C. Then $\vec{w}$ is the sum of a set R of i rows of $G=[I_k|A]$, where $1 \leq i \leq 3$. Furthermore, since I_k contributes exactly i ones to the weight of $\vec{w}$, A contributes exactly $4-i$ ones to the weight of $\vec{w}$. In other words, there are exactly $4-i$ columns in A that have odd weight in the set of rows R. Let S denote the rows of A that correspond to the set of rows R. Now there are exactly 2^i-1 different non-zero columns that may occur in S. If we count the number of times each of these columns occurs in S then the following must be true:

- For any row r in S, the number of columns in S that contain a one in row r must be an odd number that is greater than or equal to 3 (the rows of A, and therefore S, have odd weight and C has distance 4).
- For any pair of rows r_1 and r_2 in S, the number of columns in S that contain a one in both r_1 and r_2 must be even (the rows of A intersect in an even number of positions).
- There are exactly $4-i$ columns in S that have odd weight.

These properties give us a number of equations, where the unknowns in the equations are the number of times each of the 2^i-1 different non-zero columns that may occur in S, do occur in S. Finding all solutions to these equations gives us all the different ways in which a weight 4 word may be generated by G. Using this method, we have the following theorem:

Theorem 4.1 Let C be a $(2k, k, 4)$ self-dual code generated by a matrix of the form $G = [I_k|A]$. Let $\vec{w}$ be any weight 4 word in C. Then $\vec{w}$ is either a row of G (as given below), or the sum of two rows of G (as given below), or the sum of three rows of G (as given below):

$$\left.\begin{array}{c|ccc} 1 & 1 & 1 & 1 \end{array}\right. \qquad \left.\begin{array}{cc|ccccc} 1 & 0 & 1 & 0 & 1 & 1 & a \\ 0 & 1 & 0 & 1 & 1 & 1 & a \end{array}\right. \qquad \left.\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 1 & 1 & 0 & a \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & b \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & a \oplus b \end{array}\right.$$

Notes on notation: To the left of the vertical bar are the columns of I_k that are non-zero in each of the rows given, and, to the right of the vertical bar are the columns of A that are non-zero in each of the rows given. For the case in which $\vec{w}$ is the sum of two rows, the letter a is used to indicate that both rows are the same in the remaining columns of A. For the case in which $\vec{w}$ is the sum of three rows, the notation a, b, and $a \oplus b$ is used to represent all the different values that the remaining columns may take. The notation $a \oplus b$ means add rows a and b componentwise (modulo 2).

Proof We know that $\vec{w}$ must either be a row of G, the sum of two rows of G, or the sum of three rows of G.

If $\vec{w}$ occurs in one row, it must have the form $\left.\begin{array}{c|ccc} 1 & 1 & 1 & 1 \end{array}\right.$.

Suppose $\vec{w}$ is the sum of two rows of $G = [I_k|A]$.

Let x_m, $1 \leq m \leq 3$, denote the number of occurrences of each of the 3 different non-zero columns that may occur in two rows of A:

$$\begin{array}{ccc} 1 & 1 & 0 \\ 1 & 0 & 1 \\ x_3 & x_2 & x_1 \end{array}$$

Then, using the facts that any row of A has odd weight, any two rows of A intersect evenly, and the number of odd weight columns in these two rows must be $4 - 2 = 2$, we

get the following equations:

$$x_3 + x_2 \equiv 1 \bmod 2 \tag{4.1}$$

$$x_3 + x_1 \equiv 1 \bmod 2 \tag{4.2}$$

$$x_3 \equiv 0 \bmod 2 \tag{4.3}$$

$$x_2 + x_1 = 2 \tag{4.4}$$

Equations 4.1, 4.2, and 4.3 imply that $x_2 \equiv x_1 \equiv 1 \bmod 2$, which together with Equation 4.4 implies $x_2 = 1$ and $x_1 = 1$. Furthermore, since the distance of our code is 4, $x_3 \geq 2$. Therefore, if $\vec{w}$ is the sum of two rows of G, then the two rows of G that sum to $\vec{w}$ must be:

$$\begin{array}{cc|ccccc} 1 & 0 & 1 & 0 & 1 & 1 & a \\ 0 & 1 & 0 & 1 & 1 & 1 & a \end{array}$$

Suppose $\vec{w}$ is the sum of three rows of $G = [I_k|A]$.

Let $x_m, 1 \leq m \leq 7$, denote the number of occurrences of each of the 7 different non-zero columns that may occur in three rows of A:

$$\begin{array}{ccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 \end{array}$$

Then, using the facts that any row of A has odd weight, any two rows of A intersect evenly, and the number of odd weight columns in these three rows must be $4 - 3 = 1$, we get the following equations:

$$x_7 + x_6 + x_5 + x_4 \equiv 1 \bmod 2 \tag{4.5}$$

$$x_7 + x_6 + x_3 + x_2 \equiv 1 \bmod 2 \tag{4.6}$$

$$x_7 + x_5 + x_3 + x_1 \equiv 1 \bmod 2 \tag{4.7}$$

$$x_7 + x_6 \equiv 0 \bmod 2 \tag{4.8}$$

$$x_7 + x_5 \equiv 0 \bmod 2 \tag{4.9}$$

$$x_7 + x_3 \equiv 0 \bmod 2 \tag{4.10}$$

$$x_7 + x_4 + x_2 + x_1 = 1 \tag{4.11}$$

Equations 4.8, 4.9, 4.10 imply:

$$x_7 \equiv x_6 \equiv x_5 \equiv x_3 \bmod 2 \tag{4.12}$$

Equations 4.5, 4.6, 4.7, 4.12 imply:

$$x_4 \equiv x_2 \equiv x_1 \equiv x_7 + 1 \bmod 2 \tag{4.13}$$

Equations 4.11, 4.13 imply:

$$x_4 = x_2 = x_1 = 0 \tag{4.14}$$

$$x_7 = 1 \tag{4.15}$$

And finally, Equations 4.12, 4.15 imply:

$$x_6 \equiv x_5 \equiv x_3 \equiv 1 \bmod 2 \tag{4.16}$$

Therefore, by Equations 4.14, 4.15, and 4.16, if $\vec{w}$ is the sum of three rows of G, then the three rows of G that sum to $\vec{w}$ must be:

$$\begin{array}{ccc|cccc|c} 1 & 0 & 0 & 1 & 1 & 1 & 0 & a \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & b \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & a \oplus b \end{array}$$

We have now found all the different ways that a weight 4 word may be generated by a matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code. □

Knowledge of the three different ways that a weight 4 word may be generated by a generator matrix $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code, will prove to be very useful throughout the remainder of the thesis. For this section, they will be used

to find a canonical $G = [I_k|A]$ that exists for any equivalence class of $(2k, k, 4)$ self-dual codes. This canonical form will then be used to develop a recursive algorithm for finding a list L of $k \times k$ matrices with the property that for any $(2k, k, 4)$ self-dual code C, there exists at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C.

Let C be a $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$. Recall the three operations that can be applied to the $k \times k$ matrix A that result in a matrix A' such that $G' = [I_k|A']$ generates a code equivalent to C. They were SWAP_r (swap two rows), SWAP_c (swap two columns), and $\text{ADD}_{r,c}$ (*add row i at column j*). We will use these operations to prove Theorem 4.2 which will lead us to our canonical form for $G = [I_k|A]$.

Theorem 4.2 Let C be any $(2k, k, 4)$ self-dual code and let $\vec{w}$ be any weight 4 word in C. Then there exists a matrix $G' = [I_k|A']$ that generates a code C', that is equivalent to C, such that G' contains the two rows:

$$\begin{array}{cc|ccccc} 1 & 0 & 1 & 0 & 1 & 1 & a \\ 0 & 1 & 0 & 1 & 1 & 1 & a \end{array} \qquad w(a) \equiv 0 \bmod 2$$

that sum to the weight 4 word $\vec{w'}$ of C' that corresponds to the weight 4 word $\vec{w}$ of C. Note that $w(a)$ denotes the number of ones in a.

Proof For any $(2k, k, 4)$ self-dual code there exists an equivalent code C generated by a matrix of the form $G = [I_k|A]$. By Theorem 4.1, any weight 4 word in C is either a row of G, generated by two rows of G, or is generated by three rows of G, as follows:

$$\begin{array}{c|ccc} 1 & 1 & 1 & 1 \end{array} \qquad \begin{array}{cc|ccccc} 1 & 0 & 1 & 0 & 1 & 1 & a \\ 0 & 1 & 0 & 1 & 1 & 1 & a \end{array} \qquad \begin{array}{ccc|ccccc} 1 & 0 & 0 & 1 & 1 & 1 & 0 & a \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & b \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & a \oplus b \end{array}$$

What we would like to show is that for any weight 4 word $\vec{w}$ in C, there always exists an equivalent code C', generated by $G' = [I_k|A']$, such that the weight 4 word $\vec{w'}$ of C', that corresponds to $\vec{w}$ of C, is the sum of two rows of G'. We will show this by showing that

for any weight 4 word $\vec{w}$ that is either a row of G or the sum of three rows of G, there exists an $\text{ADD}_{r,c}$ operation that can be applied to A that results in a matrix A' such that $\vec{w'}$ is the sum of two rows of $G' = [I_k|A']$.

Suppose the weight 4 word $\vec{w}$ is a row of $G = [I_k|A]$. Let r denote the corresponding weight 3 row in A. Since the columns of A have weight ≥ 3 (C is a self-dual code with distance 4), there must exist a row r_1 in A that intersects row r. Furthermore, since the rows of A have odd weight ≥ 3 and any pair of rows of A intersect evenly (C is a self-dual code with distance 4), row r_1 has odd weight ≥ 3 and intersects row r in exactly two positions. Therefore, G must contain the following two rows:

$$\begin{array}{c|cccc|ccccc|l} & & & & & & & & & & \\ r & 1 & 0 & 0^* & & 1 & 1 & 1 & 0 & 0^* & \\ r_1 & 0 & 1 & 0^* & & \mathbf{1} & 1 & 0 & 1 & a & w(a) \equiv 0 \bmod 2 \end{array}$$

Note that 0^* represents a string of zero or more zeroes. If we perform an $\text{ADD}_{r,c}$ operation on the boldface entry in A, producing the matrix A', then the resulting matrix $G' = [I_k|A']$ will contain the following two rows (which sum to the weight 4 word $\vec{w'}$ that corresponds to $\vec{w}$):

$$\begin{array}{c|cccc|ccccc|l} & & & & & & & & & & \\ r \oplus r_1 & 1 & 0 & 0^* & & 1 & 0 & 1 & 1 & a & \\ r_1 & 0 & 1 & 0^* & & 1 & 1 & 0 & 1 & a & w(a) \equiv 0 \bmod 2 \end{array}$$

A few columns swaps in A' give us the two rows we're looking for. This completes the case for a weight 4 word that is a row of G.

Suppose our weight 4 word $\vec{w}$ is the sum of 3 rows of $G = [I_k|A]$:

$$\begin{array}{c|cccc|ccccc|l} & & & & & & & & & & \\ r_1 & 1 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & a & w(a) \equiv 0 \bmod 2 \\ r_2 & 0 & 1 & 0 & 0^* & 1 & 1 & 0 & 1 & b & w(b) \equiv 0 \bmod 2 \\ r_3 & 0 & 0 & 1 & 0^* & 1 & 0 & 1 & 1 & a \oplus b & w(a \oplus b) \equiv 0 \bmod 2 \end{array}$$

If we perform an $\text{ADD}_{r,c}$ operation on the boldface entry in A, producing the matrix A', then then resulting matrix $G' = [I_k|A']$ will contain the following three rows, of which

the first two sum to the weight 4 word $\vec{w}'$ that corresponds to $\vec{w}$:

$$\begin{array}{rcccc|ccccc} r_1 & 1 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & a \\ r_2 \oplus r_3 & 0 & 1 & 0 & 0^* & 0 & 1 & 1 & 1 & a \\ r_3 & 0 & 0 & 1 & 0^* & 1 & 0 & 1 & 1 & a \oplus b \end{array} \qquad w(a) \equiv 0 \bmod 2$$

A few column swaps in A' give the two rows (i.e. the first two rows) we're looking for. This completes the case for a weight 4 word that is the sum of three rows of G.

This completes our proof. □

Thus, by Theorem 4.2, for any $(2k, k, 4)$ self-dual code, there exists an equivalent code C, generated by a matrix of the form:

$$G = [I_k|A] = \left[\begin{array}{cc|ccc||cc|c} 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \vec{v} \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 1 & \vec{v} \\ \hline 0 & 0 & & & & & & & \\ 0 & 0 & & & & & & & \\ \vdots & \vdots & & & I_{k-2} & & & F & A_0 \\ & & & & & & & & \\ 0 & 0 & & & & & & & \end{array}\right]$$

where the first 2 rows of G sum to a weight 4 word. This is our canonical $G = [I_k|A]$ for the $(2k, k, 4)$ self-dual codes. The remaining columns in A of our weight 4 word are now represented by the $(k-2)$–vector $\vec{v}$. Since C is self-dual and has distance 4, $w(\vec{v}) \equiv 0 \bmod 2$ and $w(\vec{v}) \geq 2$. The remaining rows of A are represented by the $(k-2) \times 2$ matrix F and the $(k-2) \times (k-2)$ matrix A_0.

Let us now have a look at the structures of F and A_0 in A. Since the rows of A intersect in an even number of positions and since the first two rows of A differ only in the first two columns, the first two columns in each of the last $k-2$ rows of A must be the same (for if A contained a row r, among its last $k-2$ rows, such that the first two columns of r were either 0 1 or 1 0, then r would intersect one of the first two rows of A in an even number of positions and the other in an odd number of positions, which is a contradiction). Therefore, each row in F contains either 0 0 or 1 1. Since the rows of F are

either 0 0 or 1 1 and since the rows of A have odd weight and intersect evenly, the rows of A_0 have odd weight and intersect evenly. Therefore, by Theorem 2.3 of Chapter 2, since the rows of the $(k-2) \times (k-2)$ matrix A_0 have these two properties, $G_0 = [I_{k-2}|A_0]$ generates a $(2k-4, k-2)$ self-dual code. Furthermore, since the rows of A intersect evenly, if a row of A_0 intersects $\vec{v}$ in an even number of positions, the corresponding row in F must contain 0 0, and if a row of A_0 intersects $\vec{v}$ in an odd number of positions, the corresponding row in F must contain 1 1. Therefore, the rows of F are determined by A_0 and $\vec{v}$.

Looking at this another way, for any $(2k, k, 4)$ self-dual code there exists an equivalent code C, generated by $G = [I_k|A]$, such that A can be found using the $(k-2)$-vector $\vec{v}$ and the $(k-2) \times (k-2)$ matrix A_0, where $G_0 = [I_{k-2}|A_0]$ generates a $(2k-4, k-2)$ self-dual code C_0. The last $k-2$ columns of the first two rows of A are both equal to $\vec{v}$. The $(k-2) \times (k-2)$ submatrix in the last $k-2$ rows and columns of A is A_0. The 2×2 submatrix in the first two rows and first two columns of A is the identity matrix I_2. The $(k-2) \times 2$ matrix F in the last $k-2$ rows of the first two columns of A is determined by A_0 and $\vec{v}$. We will use the notation $\mathcal{A}(A_0, \vec{v})$ to represent the matrix A that is determined by A_0 and $\vec{v}$.

We shall call this process of producing the generator matrix $G = [I_k|A]$ from the generator matrix $G_0 = [I_{k-2}|A_0]$ and the vector $\vec{v}$ *extending*[1] the $(2k-4, k-2)$ self-dual code C_0 to the $(2k, k, 4)$ self-dual code C, using the vector $\vec{v}$. The vector $\vec{v}$ will be referred to as an *extension vector*. $\mathcal{A}(A_0, \vec{v})$ will be referred to as an *extension matrix*. As our next theorem will demonstrate, this extension process will lead us to our recursive algorithm.

Theorem 4.3 Let C be the $(2k, k, 4)$ self-dual code generated by $G = [I_k|\mathcal{A}(A_0, \vec{v})]$, where $G_0 = [I_{k-2}|A_0]$ generates a $(2k-4, k-2)$ self-dual code C_0. Then for any $(2k-4, k-2)$ self-dual code C_0', equivalent to C_0, where C_0' is generated by $G_0' = [I_{k-2}|A_0']$, there exists a vector $\vec{v'}$ such that $G' = [I_k|\mathcal{A}(A_0', \vec{v'})]$ generates a code equivalent to C.

[1] Our use of the term extension is different from the standard meaning in coding theory.

Proof C is generated by the matrix:

$$G = [I_k|\mathcal{A}(A_0, \vec{v})] = \left[\begin{array}{cc|cc|ccc|cc|c} 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \vec{v} \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 1 & \vec{v} \\ \hline 0 & 0 & & & & & & & \\ 0 & 0 & & & & & & & \\ \vdots & \vdots & & & I_{k-2} & & & F & A_0 \\ & & & & & & & & \\ 0 & 0 & & & & & & & \end{array}\right]$$

where $\vec{v}$ is a $(k-2)$–vector and A_0 is a $(k-2) \times (k-2)$ matrix such that $G_0 = [I|A_0]$ generates a $(2k-4, k-2)$ self-dual code C_0. F is a $(k-2) \times 2$ matrix determined by $\vec{v}$ and A_0.

What we would like to show is that for any $(2k-4, k-2)$ self-dual code C_0' generated by a matrix of the form $G_0' = [I_{k-2}|A_0']$, where C_0' is equivalent to C_0, there exists an extension vector $\vec{v'}$ such that $\vec{v'}$ can be used to extend C_0' to a $(2k, k, 4)$ self-dual code C', equivalent to C, where C' is generated by the matrix $G' = [I_k|\mathcal{A}(A_0', \vec{v'})]$.

Recall that Theorem 2.16 of Chapter 2 states that for any self-dual code generated by $G_0 = [I_k|A_0]$, any equivalent code generated by $G_0' = [I_k|A_0']$ can be obtained by performing some sequence of the three operations SWAP_r, SWAP_c, and $\text{ADD}_{r,c}$ on A_0, producing the matrix A_0'. If we perform such a sequence of operations on the matrix A_0 in $G = [I_k|\mathcal{A}(A_0, \vec{v})]$, the vector $\vec{v}$ may change, as may F, but the end result will still be a matrix of the form:

$$G' = [I_k|\mathcal{A}(A_0', \vec{v'})] = \left[\begin{array}{cc|cc|ccc|cc|c} 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \vec{v'} \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 1 & \vec{v'} \\ \hline 0 & 0 & & & & & & & \\ 0 & 0 & & & & & & & \\ \vdots & \vdots & & & I_{k-2} & & & F' & A_0' \\ & & & & & & & & \\ 0 & 0 & & & & & & & \end{array}\right]$$

Therefore, given any $(2k-4, k-2)$ self-dual code C_0', generated by $G_0' = [I_{k-2}|A_0']$, there exists an extension vector $\vec{v'}$ such that $\vec{v'}$ can be used to extend C_0' to a $(2k, k, 4)$ self-dual code equivalent to C. □

Corollary 4.4 Let C be any $(2k, k, 4)$ self-dual code. Let L_0 be any list of $(k-2)\times(k-2)$ matrices A_0, where $G_0 = [I_{k-2}|A_0]$ generates a $(2k-4, k-2)$ self-dual code, such that for any $(2k-4, k-2)$ self-dual code C_0, there exists one and only one $A_0 \in L_0$ such that $G_0 = [I_{k-2}|A_0]$ generates a code equivalent to C. Then there exists a matrix $A_0 \in L_0$ and a $(k-2)$-vector $\vec{v}$ such that $G = [I_k|\mathcal{A}(A_0, \vec{v})]$ generates a $(2k, k, 4)$ self-dual code that is equivalent to C. □

Therefore, given any such list L_0, if for each matrix $A_0 \in L_0$ and for each of the $< 2^{k-2}$ possible extension vectors $\vec{v}$, where $w(\vec{v}) \equiv 0 \bmod 2$ and $w(\vec{v}) \geq 2$, we produce the matrix $A = \mathcal{A}(A_0, \vec{v})$, then the resulting list L of $k \times k$ matrices will have the property that for any $(2k, k, 4)$ self-dual code C there will exist at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C.

One last thing that needs to be considered before we give a formal description of our algorithm is the distance of the $(2k-4, k-2)$ self-dual codes we use. Suppose $G_0 = [I_{k-2}|A_0]$ generates a $(2k-4, k-2, 2)$ self-dual code C_0 and let $\vec{v}$ be a $(k-2)$-vector such that $w(\vec{v}) \equiv 0 \bmod 2$ and $w(\vec{v}) \geq 2$. Then the matrix $G = [I_k|\mathcal{A}(A_0, \vec{v})]$ generates a $(2k, k, d)$ self-dual code C, where either $d = 2$ or $d = 4$. However, we are only interested in self-dual codes with distance 4. As we shall see, if C_0 has distance 2, then the extension vector $\vec{v}$ determines the distance of C.

Let C be any $(2k, k, 2)$ self-dual code generated by $G = [I_k|A]$. Then any weight 2 word $\vec{w}$ of C must be a row of G (Theorem 2.8 of Chapter 2: weight d words are the sum of at most $d-1$ rows of $G = [I_k|A]$). This implies that A contains a row r with weight 1. Furthermore, the column of A in which row r is 1 contains 0's in all other rows of A (since rows of A intersect evenly and row r of A contains only one 1). Conversely, for any column of A that contains only one 1, say in row r, row r contains 0's in every other column of A (since columns of A intersect evenly), and therefore, row r corresponds to a weight 2 word. Therefore, any $(2k, k)$ self-dual code C, generated by $G = [I_k|A]$, has distance 2 if and only if A contains a column with weight 1. Furthermore, A contains a

column with weight 1 if and only if A contains a row with weight 1.

Let C_0 be any $(2k-4, k-2, 2)$ self-dual code generated by $G_0 = [I_{k-2}|A_0]$ and let $\vec{v}$ be any extension vector. Consider the distance of the code C, generated by $G = [I_k|A]$ where $A = \mathcal{A}(A_0, \vec{v})$, which will either be 2 or 4. Since $w(\vec{v}) \geq 2$ and since the first two columns of A must intersect each of the last $k-2$ columns of A in an even number of positions, the weight of the first two columns in A is ≥ 3 (i.e. since in the last $k-2$ columns of A there exists at least one column c that contains two 1's in its first two rows and since the first two columns of A only contain one 1 in their first 2 rows, there must exist a row among the last $k-2$ rows of A in which they intersect. This implies that the first two columns of A have weight ≥ 2 and since columns of A have odd weight, the first two columns of A have weight ≥ 3). Therefore, C will have distance 2 if and only if there exists a column among the last $k-2$ columns of $A = \mathcal{A}(A_0, \vec{v})$ that has weight 1. Thus, C will have distance 4 if and only if the last $k-2$ columns of A all have weight ≥ 3. Suppose the $(2k-4, k-2)$ self-dual code C_0 has distance ≥ 4. Then the columns of the matrix A_0 all have weight ≥ 3 which implies that the last $k-2$ columns of A all have weight ≥ 3. Suppose C_0 has distance 2. Then the last $k-2$ columns of A will all have weight ≥ 3 if and only if for any column in A_0 with weight 1, the corresponding column in the extension vector is 1. Therefore, the $(2k, k)$ self-dual code C generated by $G = [I_k|\mathcal{A}(A_0, \vec{v})]$ has distance 4 if and only if C_0 has distance ≥ 4 or if in every column of A_0 that has weight 1, the corresponding column in $\vec{v}$ is also 1.

We are now ready to give a formal description of our algorithm for enumerating the $(2k, k, 4)$ self-dual codes. The two procedures *QuickOrder* and *MinimizeRowWeights* used in the algorithm are defined in Chapter 3 and are used to reduce the number of pairs of matrices A_1 and A_2 inserted into L in which $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes and $A_1 \neq A_2$.

Algorithm 4.5 :

- Input: a list L_0 of $(k-2) \times (k-2)$ matrices A_0, where $G_0 = [I_{k-2}|A_0]$ generates a $(2k-4, k-2)$ self-dual code, such that for any $(2k-4, k-2)$ self-dual code C_0 there exists one and only one matrix A_0 in L_0 such that $G_0 = [I_k|A_0]$ generates a code equivalent to C_0.
- Output: a list L of $k \times k$ matrices A, where $G = [I_k|A]$ generates a $(2k, k, 4)$ self-dual code, such that for any $(2k, k, 4)$ self-dual code C there exists at least one matrix A in L such that $G = [I_k|A]$ generates a code equivalent to C.

begin
for each $A_0 \in L_0$ **do**
for each $(k-2)$–vector $\vec{v}$ such that $w(\vec{v}) \equiv 0 \bmod 2$, $w(\vec{v}) \geq 2$, and $\vec{v}$ contains a 1 in each column corresponding to a column in A_0 with weight 1 **do**
Let A=$\mathcal{A}(A_0, \vec{v})$.
QuickOrderMatrix$(A, 0, A')$.
MinimizeRowWeights(A', A'').
if $A'' \notin L$ **then**
Insert A'' into L.
end for
end for
end

In the remainder of this chapter, we will look at ways of improving this algorithm. These improvements will include reducing both the dimension of the input codes and the number of input codes used, and reducing both the dimension of the extension vector and the number of extension vectors used. These improvements will result in both a reduction on the running time of the algorithm and a reduction in the number of equivalent codes produced by the algorithm.

4.2 Weight 4 Blocks and Weight 4 Sets

Let C be a $(2k, k, 4)$ self-dual code generated by a matrix of the form $G = [I_k|A]$. In this section, we will look at the different ways in which weight 4 words may *interact* in such codes. We will look at:

- The ways in which weight 4 rows of G interact. For this, we will define a structure called a *weight 4 block* of G.
- The ways in which weight 4 words of C interact. For this, we will define a structure called a *weight 4 set* of C.
- The ways in which rows of G may *generate* a weight 4 set. For this, we will define a structure called a *generator block* for a weight 4 set.

We will use the results of this section to make improvements on our algorithm for enumerating the $(2k, k, 4)$ self-dual codes.

Let us begin by defining precisely what we mean by a weight 4 block of G:

Definition 4.6 Let G be a generator matrix for a $(2k, k, 4)$ self-dual code C. A *weight 4 block* of G is a set T of weight 4 rows in G such that:

1. If $|T| > 1$ then T cannot be partitioned into two disjoint sets T_1 and T_2 in which none of the rows in T_1 intersects any of the rows in T_2, and
2. there does not exist a weight 4 row $r \in G - T$ such that r intersects a row in T.

Notice that the first condition implies that if $|T| > 1$ then for any row $r_1 \in T$ there exists at least one row $r_2 \in T$ such that r_1 and r_2 intersect.

Example 4.7 Let C be the $(14, 7, 4)$ self-dual code generated by:

$$G = [I_7|A] = \left[\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right] \begin{array}{l} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \end{array}$$

The weight 4 blocks of G are $T_1 = \{r_1, r_2, r_3\}$ and $T_2 = \{r_4\}$.

In this section we will find all the different weight 4 blocks that may occur in a matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code. We will use the different weight 4 blocks to help us find all the different *generator blocks* for *weight 4 sets* that may occur in such a G.

A weight 4 block directly relates to an indecomposable self-orthogonal code, with distance 4, that is generated by a matrix whose rows all have weight four. All of these were first found by V.Pless and N. J. A. Sloane [3]. In their paper, they used them to aid them in their enumeration of the $(22, 11, 4)$ and $(24, 12, 4)$ self-dual codes. As we shall see, each of the weight 4 blocks we find directly relates to one of the self-orthogonal codes they found.

Let us now define precisely what we mean by a weight 4 set of C and what we mean by a generator block of G for a weight 4 set of C:

Definition 4.8 Let C be a $(2k, k, 4)$ self-dual code. A *weight 4 set* of C is a set B of weight 4 words in C such that:

1. If $|B| > 1$ then B cannot be partitioned into two disjoint sets B_1 and B_2 in which none of the weight 4 words in B_1 intersects any of the weight 4 words in B_2, and

2. there does not exist a weight 4 word $\vec{w} \in C - B$ such that $\vec{w}$ intersects a codeword in B.

Notice that the first condition implies that if $|B| > 1$ then for any weight 4 word $\vec{w_1} \in B$ there exists at least one weight 4 word $\vec{w_2} \in B$ such that $\vec{w_1}$ and $\vec{w_2}$ intersect.

Definition 4.9 A *generator block* of G for a weight 4 set B of C is a set of rows G_B in G such that:

1. the set of codewords that G_B generates contains B, and
2. for any row $r \in G_B$ the set of codewords that $G_B - \{r\}$ generates does not contain B.

We shall say the generator block G_B *generates* the weight 4 set B.

Example 4.10 Let C be the $(14, 7, 4)$ self-dual code generated by:

$$G = [I_7|A] = \begin{array}{c} \begin{array}{ccccccc|ccccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{array} \\ \left[\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right] \begin{array}{c} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \end{array} \end{array}$$

The weight 4 words of C are:

$$r_1, r_2, r_3, r_1 \oplus r_2, r_1 \oplus r_3, r_2 \oplus r_3, r_1 \oplus r_2 \oplus r_3,$$
$$r_4, r_5 \oplus r_6, r_5 \oplus r_7, r_6 \oplus r_7, r_4 \oplus r_5 \oplus r_6, r_4 \oplus r_5 \oplus r_7, r_4 \oplus r_6 \oplus r_7$$

The weight 4 sets of C are:

$$B_1 = \{r_1, r_2, r_3, r_1 \oplus r_2, r_1 \oplus r_3, r_2 \oplus r_3, r_1 \oplus r_2 \oplus r_3\}$$

$$B_2 = \{r_4, r_5 \oplus r_6, r_5 \oplus r_7, r_6 \oplus r_7, r_4 \oplus r_5 \oplus r_6, r_4 \oplus r_5 \oplus r_7, r_4 \oplus r_6 \oplus r_7\}$$

The set of rows $G_{B_1} = \{r_1, r_2, r_3\}$ generates B_1 and the set of rows $G_{B_2} = \{r_4, r_5, r_6, r_7\}$ generates B_2.

Let C be a $(2k, k, 4)$ self-dual code, generated by G, and let B be any weight 4 set in C, generated by G_B. Let c be any column in G for which there exists at least one weight 4 word $\vec{w}$ in B such that the rows of G_B that sum to $\vec{w}$ sum to one (modulo 2) in column c. Then, by our definition of a weight 4 set, if $\vec{w'}$ is any weight 4 word in C such that the rows of G that sum to $\vec{w}$ sum to one (modulo 2) in column c, $\vec{w'}$ must be in B. Therefore, in this sense, the column c is *related* to the weight 4 set B. We will use this observation to define the *columns of a weight 4 set*:

Definition 4.11 Let C be a $(2k, k, 4)$ self-dual code generated by G. Let B be a weight 4 set of C and let G_B be the generator block in G that generates B. We define the *columns* of the weight 4 set B (with respect to G) as the set of columns S_B in G such that a column c in G is an element of S_B if and only if there exists at least one weight 4 word $\vec{w}$ in B such that the set of rows of G_B that sum to $\vec{w}$ sum to one (modulo 2) in column c.

For example, the columns of the weight 4 set B_1 in Example 4.10 are $S_{B_1} = \{a_1, a_2, a_3, c_1, c_2, c_3, c_4\}$ and the columns of the weight 4 set B_2 are $S_{B_2} = \{a_4, a_5, a_6, a_7, c_5, c_6, c_7\}$. We may sometimes refer to the columns of the weight 4 set B as the columns of the generator block G_B.

Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, 4)$ self-dual code. Suppose G contains two generator blocks G_{B_1} and G_{B_2} for weight 4 sets B_1 and B_2, respectively. Let S_{B_1} and S_{B_2} be the columns of B_1 and B_2, respectively. Then, since none of the codewords

in B_1 intersects a codeword in B_2 (by definition of a weight 4 set), $S_{B_1} \cap S_{B_2} = \emptyset$. Now, suppose $G_{B_1} \cap G_{B_2} \neq \emptyset$. That is, suppose there exists a row r in $G = [I_k|A]$ such that $r \in G_{B_1}$ and $r \in G_{B_2}$. Then since the set of codewords generated by the set of rows $G_{B_1} - \{r\}$ does not contain every weight 4 word in B_1 (by definition of a generator block for a weight 4 set) there must exist a weight 4 word $\vec{w_1} \in B_1$ such that the set of rows in G_{B_1} that sum to $\vec{w_1}$ includes row r. Similarly, there must exist a weight 4 word $\vec{w_2} \in B_2$ such that the set of rows in G_{B_2} that sum to $\vec{w_2}$ includes row r. However, G is of the form $G = [I_k|A]$ which implies that the column of I_k that contains a one in row r is a column of both S_{B_1} and S_{B_2} (i.e. if c is the column of I_k that contains a one in row r, then the set of rows of G_{B_1} that sum to $\vec{w_1}$ sum to a one in column c and the set of rows of G_{B_2} that sum to $\vec{w_2}$ sum to a one in column c). However, as we have already seen, $S_{B_1} \cap S_{B_2} = \emptyset$. In other words, we have just shown that if G_{B_1} and G_{B_2} are generator blocks for weight 4 sets in a generator matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code, then $G_{B_1} \cap G_{B_2} = \emptyset$.

Therefore, for any generator matrix $G = [I_k|A]$ for a $(2k, k, 4)$ self-dual code C, if $B_1, B_2, \ldots, B_m$ are the weight 4 set in C, then the rows of G can be partitioned into the $m + 1$ disjoint sets $G_{B_1}, G_{B_2}, \ldots, G_{B_m}, G_R$ where G_{B_i} is the generator block for the weight 4 set B_i and G_R is the set of remaining rows of G. Furthermore, the columns of G may be partitioned into the $m + 1$ disjoint sets $S_{B_1}, S_{B_2}, \ldots, S_{B_m}, S_R$ where S_{B_i} are the columns of the weight 4 set B_i and S_R is the set of remaining columns of G.

In this section, we will find a list of generator blocks G_B for weight 4 sets B such that for any $(2k, k, 4)$ self-dual code there exists an equivalent code, generated by a matrix of the form $G = [I_k|A]$, such that G contains one of the generator blocks G_B. We will then use this list of generator blocks for weight 4 sets to find all the different generator blocks that may occur in a generator matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code.

4.2.1 The Weight 4 Blocks of $G = [I_k|A]$

We will begin by first finding all the different weight 4 blocks that may occur in a generator matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code. By *different*, we mean weight 4 blocks whose rows and columns are not simply permutations of one another. Since we are only looking for weight 4 blocks that may occur in generator matrices of the form $G = [I_k|A]$, and since any two rows in such a G only intersect in the columns of A, we will look only for the different weight 4 blocks in the columns of A. That is, we will look for the different ways that rows of A with weight 3 may intersect. Therefore, what we will be looking for is all the different sets T of weight 3 rows that may occur in such an A such that if $|T| > 1$ then T cannot be partitioned into two disjoint set T_1 and T_2 such that none of the rows in T_1 intersects a row in T_2, and there does not exist a weight 3 row in A that intersects a row in T. We shall call such a set T a *weight 3 block* of A. The number of rows in T will be called the *size* of T.

The following lemma provides us with a recursive method for finding all the different weight 3 blocks T that may occur in a matrix A, where $G = [I_k|A]$ generates a $(2k, k, 4)$ self-dual code. Note that since we are not looking for the weight 3 blocks in a specific A, the lemma considers a set T of weight 3 rows to be a weight 3 block if and only if T cannot be partitioned into the two disjoint sets T_1 and T_2.

Lemma 4.12 For any weight 3 block T of size $i \geq 2$, if we remove any row from T, we get a weight 3 block of size $i - 1$.

Proof This is certainly true for $i = 2$. For $i > 2$, what we need to show is that if we remove any row r from the weight 3 block T, then the resulting set of rows $T' = T - \{r\}$ still has the property that T' cannot be partitioned into two disjoint sets T_1' and T_2' such that none of the rows in T_1' intersects a row in T_2'. Assume this is not the case. That is, assume T' can be partitioned into two sets T_1' and T_2' such that none of the rows in T_1' intersects a row in T_2'. Consider the weight 3 row r that was removed from T. Since T is

a weight 3 set, $T = T' \cup \{r\}$ cannot be partitioned into two disjoint sets T_1 and T_2 such that none of the rows in T_1 intersects a row in T_2. Therefore, row r must intersect rows in both T_1' and T_2'. However, since any pair of rows in a weight 3 block that intersect must intersect in exactly two positions, row r must intersect some row in T_1' in two positions and r must intersect some row in T_2' in two positions. This implies that our weight 3 row r has weight 4, which is a contradiction. Therefore, the set $T' = T - \{r\}$ is a weight 3 block. □

Therefore, we can find all weight 3 blocks of size i by first finding all weight 3 blocks of size $i-1$ and then looking for all the different ways a weight 3 row can be adjoined to them.

Let us now use this recursive method to find all the different weight 3 blocks that may occur in the matrix A, where $G = [I_k|A]$ generates a $(2k, k, 4)$ self-dual code. The weight 3 block of size 1 is:

$$D_1 = \begin{array}{cccc} 1 & 1 & 1 & 0^* \end{array}$$

There is only one way in which D_1 can be extended to a weight 3 block of size 2. Let D_2 be our weight 3 block of size 2:

$$D_2 = \begin{array}{ccccc} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \end{array}$$

There are 2 different ways in which D_2 can be extended to a weight 4 block of size 3. Let D_3 and E_3 be our weight 3 blocks of size 3:

$$D_3 = \begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 1 & 0^* \end{array} \qquad E_3 = \begin{array}{cccccc} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \end{array}$$

The only way that any weight 3 row can be adjoined to D_3 is if it intersects D_3 in only the first 2 columns. Furthermore, we can adjoin any number of weight 3 rows, all intersecting

in the first 2 columns, to D_3. This gives us the following weight 3 blocks:

$$D_i = \begin{array}{c} \overbrace{}^{i} \\ \left.\begin{array}{ccccccccc} 1 & 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0 & \cdots & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 1 & \cdots & 0 & 0 & 0^* \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 1 & 1 & 0 & 0 & 0 & \cdots & 1 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 0^* \end{array}\right\}_i \end{array} \quad i \geq 4$$

There is only one way that E_3 can be extended to a weight 4 block of size 4. Let E_4 be the resulting weight 3 block of size 4:

$$E_4 = \begin{array}{ccccc} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \\ 0 & 1 & 1 & 1 & 0^* \end{array}$$

There is no way to adjoin a weight 3 row to E_4.

We have now found all the different weight 3 blocks that may occur in A. From this, we have our list of all the different weight 4 blocks that may occur in a generator matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code C. With the same names as their corresponding weight 3 blocks, they are:

$$E_3 = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & 0^* \\ 0 & 1 & 0 & 0 & 0^* & 1 & 1 & 0 & 1 & 0^* \\ 0 & 0 & 1 & 0 & 0^* & 1 & 0 & 1 & 1 & 0^* \end{array}\right]$$

$$E_4 = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & 0^* \\ 0 & 1 & 0 & 0 & 0^* & 1 & 1 & 0 & 1 & 0^* \\ 0 & 0 & 1 & 0 & 0^* & 1 & 0 & 1 & 1 & 0^* \\ 0 & 0 & 0 & 1 & 0^* & 0 & 1 & 1 & 1 & 0^* \end{array}\right]$$

$$D_i = \begin{array}{c|c} \overbrace{1\ 0\ \cdots\ 0}^{i}\ 0^* & 1\ 1\ \overbrace{1\ 0\ \cdots\ 0}^{i}\ 0^* \\ 0\ 1\ \cdots\ 0\ 0^* & 1\ 1\ 0\ 1\ \cdots\ 0\ 0^* \\ \vdots\ \vdots\ \quad\ \vdots\ \vdots & \vdots\ \vdots\ \vdots\ \vdots\ \quad\ \vdots\ \vdots \\ 0\ 0\ \cdots\ 1\ 0^* & 1\ 1\ 0\ 0\ \cdots\ 1\ 0^* \end{array} \Bigg\}_i \quad i \geq 1$$

We will refer to these weight 4 blocks simply as E_3-blocks, E_4-blocks, and D_i-blocks. The correspondence between our list of weight 4 blocks (with the columns of zeroes removed) and the indecomposable self-orthogonal codes with distance 4 generated by matrices with weight 4 rows given by V. Pless and N. J. A. Sloane [3]: our E_3-block generates a $(7,3,4)$ self-orthogonal code equivalent to their e_7-code, our E_4-block generates a $(8,4,4)$ self-orthogonal code equivalent to their e_8-code, and our d_i-block, $i \geq 1$, generates a $(2i+2, i, 4)$ self-orthogonal code equivalent to their d_{2i+2}-code.

Knowing all the different weight 4 blocks that may occur in a generator matrix of the form $G = [I_k|A]$, where G generates a $(2k,k,4)$ self-dual code, will prove useful throughout the remainder of this thesis. In the next subsection, we will use them to help us find all the different generator blocks for weight 4 sets that may occur in such a G.

4.2.2 Generators For Weight 4 Sets of C

Let C be a $(2k,k,4)$ self-dual code generated by $G = [I_k|A]$. In this subsection, we will show that for any weight 4 set B of C there exists an equivalent code C', generated by $G' = [I_k|A']$, such that B', the weight 4 set of C' that corresponds to the weight 4 set B of C, is generated by the generator block $G_{B'}$ of G', where $G_{B'}$ is one (and only one) of the following sets of rows:

$$e_3 = \begin{array}{c|c} 1\ 0\ 0\ 0^* & 1\ 1\ 1\ 0\ 0^* \\ 0\ 1\ 0\ 0^* & 1\ 1\ 0\ 1\ 0^* \\ 0\ 0\ 1\ 0^* & 1\ 0\ 1\ 1\ 0^* \end{array}$$

$$d_i = \left[\begin{array}{cccccc|ccccccc} \overbrace{1}^{i} & 0 & \cdots & 0 & 0^* & & 1 & 1 & \overbrace{1}^{i} & 0 & \cdots & 0 & 0^* \\ 0 & 1 & \cdots & 0 & 0^* & & 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0^* & & 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \end{array}\right] \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \} i \end{matrix} \quad i \geq 1$$

$$e_{2i} = \left[\begin{array}{ccccccc|ccccccc} \overbrace{1}^{2i-2} & 0 & \cdots & 0 & 0 & 0 & 0^* & 1 & 1 & \overbrace{1}^{2i-2} & 0 & \cdots & 0 & 0^* \\ 0 & 1 & \cdots & 0 & 0 & 0 & 0^* & 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & 0^* & 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \\ 0 & 0 & \cdots & 0 & 1 & 0 & 0^* & 1 & 0 & 1 & 1 & \cdots & 1 & 0^* \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0^* & 0 & 1 & 1 & 1 & \cdots & 1 & 0^* \end{array}\right] \begin{matrix} \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \} 2i-2 \end{matrix} \quad i \geq 2$$

Note that we are using small letters to distinguish the generator blocks for weight 4 sets from the weight 4 blocks. The reason we want to distinguish the generator blocks for weight 4 sets from the weight 4 blocks is that a weight 4 block, on its own, may or may not be a generator block for a 4 weight set. For example, the weight 4 block D_4 may either be a d_4-block or a part of an e_6-block.

Let C be a $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$ and let B be a weight 4 set in C. In order to show there exists an equivalent code C', generated by $G' = [I_k|A']$, such that the weight 4 set B' in C', that corresponds to B, is generated by either an e_3-block, a d_i-block, or an e_{2i}-block, we will first find a canonical form for the matrix A.

Let G_B be the set of rows in G that generates B. Let R_0 denote the set of weight 4 rows in G_B, and let R_1 denote the set of remaining rows in G_B (i.e. $R_1 = G_B - R_0$). Suppose there exists a weight 4 word $\vec{w}$ in B with the following properties:

- $\vec{w}$ is the sum of rows from R_1 only, and
- there exists a sequence S of $\mathrm{ADD}_{r,c}$ operations on A, only involving entries (i, j) of

A in which row i corresponds to a row in R_1 and none of the rows in R_0 contain a one in column j, such that the matrix A', that results from applying S to A, is such that $G' = [I_k|A']$ contains a weight 4 row that corresponds to the weight 4 word $\vec{w}$.

If such a weight 4 word exists, then we can apply S to A, producing A', which results in a generator matrix $G' = [I_k|A']$ such that B', the weight 4 set obtained by permuting B, is generated by a generator block $G_{B'}$ that contains at least one more weight 4 row that G_B. We can continue this process until no more weight 4 words with the above two properties exists. This is our canonical form of A for a given weight 4 set B of C.

Therefore, for any weight 4 set B of a $(2k, k, 4)$ self-dual code C, there exists an equivalent code C', generated by a matrix of the form $G' = [I_k|A']$, such that B', the weight 4 set of C' obtained by permuting B, is generated by the set of rows $G_{B'}$ of G', such that G' can be partitioned into the two sets of rows R_0 and $R_1 = G_{B'} - R_0$ where:

- R_0 is the set of weight 4 rows in $G_{B'}$, and
- R_1, the remaining rows of $G_{B'}$, has the property that for any weight 4 word $\vec{w}$ in B' that is the sum of rows from R_1 only (if such a word exists), there does *not* exist a sequence S of $\mathrm{ADD}_{r,c}$ operations on A' (only involving entries (i, j) where i corresponds to a row in R_1 and no rows of R_0 contain a one in column j) such that $\vec{w'}$, the weight 4 word obtained by permuting $\vec{w}$, is a row of the resulting generator matrix.

Note that R_1 may be the empty set.

Example 4.13 Let C' be a $(2k, k, 4)$ self-dual code generated by $G' = [I_k|A']$. Suppose C' contains the weight 4 set B' generated by the set of rows $G_{B'} = \{r_1, r_2, r_3, r_4, r_5, r_6\}$,

where the columns in A' of the rows in $G_{B'}$ are:

c_1	c_2	c_3	c_4	c_5	c_6		
1	1	1	0	0	0	0*	r_1
1	1	0	1	0	0	0*	r_2
1	1	0	0	1	0	0*	r_3
1	1	0	0	0	1	0*	r_4
1	0	1	1	1	1	0*	r_5
0	1	1	1	1	1	0*	r_6

Then $R_0 = \{r_1, r_2, r_3, r_3\}$ is the set of weight 4 rows of $G_{B'}$ and $R_1 = G_{B'} - R_0 = \{r_5, r_6\}$. The only weight 4 word in B' that is the sum of rows of R_1 exclusively is the codeword $\vec{w} = r_5 \oplus r_6$. There does not exist a sequence of $\mathrm{ADD}_{r,c}$ operations on A' that does not involve rows r_1, r_2, r_3, r_4 and columns $c_1, c_2, \ldots, c_6$, such that $\vec{w'}$, the weight 4 word obtained by permuting $\vec{w}$, is a row of the resulting generator matrix. Therefore, A' is in canonical form.

Example 4.14 Let C be a $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$. Suppose C contains the weight 4 set B generated by the set of rows $G_B = \{r_1, r_2, r_3, r_4, r_5, r_6\}$, where the columns in A of the rows in G_B are:

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8		
1	1	1	0	0	0	0	0	0*	r_1
1	1	0	1	0	0	0	0	0*	r_2
1	1	0	0	1	0	0	0	0*	r_3
1	1	0	0	0	1	0	0	0*	r_4
1	0	1	1	1	1	1	1	0*	r_5
0	1	1	1	1	1	1	1	0*	r_6

Then $R_0 = \{r_1, r_2, r_3, r_4\}$ is the set of weight 4 rows of G_B and $R_1 = G_B - R_0 = \{r_5, r_6\}$. The two sets R_0 and R_1 do not have the desired properties. To see this, consider the

weight 4 word $\vec{w} = r_5 \oplus r_6$ of B. If we perform an $\text{ADD}_{r,c}$ operation on the boldface entry (which does not involve any of the weight 4 words nor any of the columns in which at least one of the weight 4 rows contains a one), producing the matrix A', then $\vec{w}$ will be permuted to $\vec{w'}$, where $\vec{w'}$ is a row in the generator matrix $G' = [I_k|A']$. We now have a generator matrix $G' = [I_k|A']$ for a $(2k, k, 4)$ self-dual code C', equivalent to C, where B', the weight 4 set obtained by permuting B, is generated by the set of rows $G_{B'} = \{r_1, r_2, r_3, r_4, r_5 \oplus r_6\}$, where the columns in A' of the rows in $G_{B'}$ are:

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8		
1	1	1	0	0	0	0	0	0*	r_1
1	1	0	1	0	0	0	0	0*	r_2
1	1	0	0	1	0	0	0	0*	r_3
1	1	0	0	0	1	0	0	0*	r_4
1	1	0	0	0	0	1	0	0*	$r_5 \oplus r_6$
0	1	1	1	1	1	1	1	0*	r_6

Then $R_0 = \{r_1, r_2, r_3, r_4, r_5 \oplus r_6\}$ is the set of weight 4 rows in $G_{B'}$ and $R_1 = G_{B'} - R_0 = \emptyset$. The two sets R_0 and R_1 have the desired properties. Therefore, A' is in canonical form.

In this subsection we will find all the different generator blocks $G_{B'}$, for weight 4 sets B', that can be partitioned into the two sets R_0 and R_1. In order to do this, we will first prove that the set of weight 4 rows R_0 must be a weight 4 block (and not the composition of two or more weight 4 blocks), and we will prove that if $R_1 \neq \emptyset$ then there must exist a weight 4 word $\vec{w'} \in B'$, that is the sum of two rows in R_1, such that $\vec{w'}$ intersects at least one row in R_0. Once we have proved these results, we will use them to find all the different generator blocks $G_{B'}$, for weight 4 sets B', that can be partitioned into the two sets R_0 and R_1.

We will begin by first proving a theorem on the different ways the rows of $G_{B'}$ may generate the subset of weight 4 words $\vec{w'} \in B'$, where $\vec{w'}$ is not the sum of rows from R_0, exclusively. In order to prove the theorem, we first require the following lemma:

Lemma 4.15 Let $G_{B'}$ be a generator block for a weight 4 set B', such that $G_{B'}$ can be partitioned into the sets R_0 and R_1. Let W denote the set of weight 4 words contained in the set of codewords generated by R_0. Suppose R_1 is not the empty set. Then if $\vec{w}$ is a weight 4 word in $B' - W$ such that $\vec{w}$ intersects a weight 4 word in W, then $\vec{w}$ is either the sum of two rows $r_1, r_2 \in R_1$, as follows:

$$\begin{array}{l|llll|llll|l} r_1 & 1 & 0 & 0^* & 1 & 0 & a & & \\ r_2 & 0 & 1 & 0^* & 0 & 1 & a & & w(a) \geq 4 \end{array}$$

or $\vec{w}$ is the sum of one row r_0 from R_0 and two rows r_1, r_2 from R_1, as follows:

$$\begin{array}{l|llll|llll|l} r_0 & 1 & 0 & 0 & 0^* & 1 & 1 & 1 & 0^* & \\ r_1 & 0 & 1 & 0 & 0^* & 1 & 1 & 0 & a & \\ r_2 & 0 & 0 & 1 & 0^* & 1 & 0 & 1 & a & w(a) \geq 3 \end{array}$$

Proof Let $\vec{w}$ be any weight 4 word in $B' - W$ such that $\vec{w}$ intersects a weight 4 word in W. Let us consider all the ways in which $\vec{w}$ may be generated by rows of $G_{B'}$. Recall, if C' is a $(2k, k, 4)$ self-dual code generated by $G' = [I_k | A']$, then any weight 4 word in C' is generated by G' in one of the following 3 ways:

$$\begin{array}{l|lll} 1 & 1 & 1 & 1 \end{array} \qquad \begin{array}{ll|llll|l} 1 & 0 & 1 & 0 & 1 & 1 & a \\ 0 & 1 & 0 & 1 & 1 & 1 & a \end{array} \qquad \begin{array}{lll|llll|l} 1 & 0 & 0 & 1 & 1 & 1 & 0 & a \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & b \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & a \oplus b \end{array}$$

For each of these three cases, by considering all the different combinations for their rows occurring in R_0 and R_1, we can determine all the different ways that the rows of $G_{B'}$ can generate $\vec{w}$.

- Suppose $\vec{w}$ is a row of $G_{B'}$. Then since $\vec{w} \in B' - W$, $\vec{w}$ is not a row of R_0, which implies $\vec{w}$ is a row of R_1. But the rows of R_1 have weight ≥ 6 and $\vec{w}$ has weight 4, which is a contradiction. Therefore, $\vec{w}$ cannot be a row of $G_{B'}$.

- Suppose $\vec{w}$ is the sum of two rows in $G_{B'}$. Then since $\vec{w} \in B' - W$, $\vec{w}$ by definition cannot be the sum of two rows of R_0. Therefore, $\vec{w}$ is either the sum of two rows from R_1, or the sum of one row from R_0 and one row from R_1.

  Suppose $\vec{w}$ is the sum of one row $r_0 \in R_0$ with one row $r_1 \in R_1$:

$$\begin{array}{l|lllll} r_0 & 1 & 0 & 0^* & 1 & 0 & 1 & 1 & 0^* \\ r_1 & 0 & 1 & 0^* & 0 & 1 & 1 & 1 & a \end{array}$$

  Then since $\vec{w}$ is a weight 4 word, we must have $a = 0^*$. But this implies row r_1 has weight 4, which implies that $r_1 \in R_0$, which is a contradiction.

  Therefore, if $\vec{w}$ is the sum of two rows of $G_{B'}$, then $\vec{w}$ is the sum of two rows $r_1, r_2 \in R_1$:

$$\begin{array}{l|lll|ll} r_1 & 1 & 0 & 0^* & 1 & 0 & a \\ r_2 & 0 & 1 & 0^* & 0 & 1 & a & w(a) \geq 4 \end{array}$$

- Suppose $\vec{w}$ is the sum of three rows of $G_{B'}$. Then since $\vec{w} \in B' - W$, $\vec{w}$ is not the sum of three rows from R_0. Therefore, $\vec{w}$ is either the sum of two rows from R_0 and one row from R_1, the sum of one row from R_0 and two rows from R_1, or the sum of three rows from R_1.

  Suppose $\vec{w}$ is the sum of two rows $r_0, r_1 \in R_0$ and one row $r_2 \in R_1$:

$$\begin{array}{l|llll|ll} r_0 & 1 & 0 & 0 & 0^* & 1 & a & w(a) = 2 \\ r_1 & 0 & 1 & 0 & 0^* & 1 & b & w(b) = 2 \\ r_2 & 0 & 0 & 1 & 0^* & 1 & a \oplus b & w(a \oplus b) \geq 4 \end{array}$$

  Then since rows r_0 and r_1 must intersect in an even number of positions we must have $a = 1100^*$ and $b = 1010^*$. But this implies $a \oplus b = 0110^*$, which implies $w(a \oplus b) = 2$, which is a contradiction. Therefore $\vec{w}$ cannot be the sum of two rows from R_0 and one row from R_1.

  Suppose $\vec{w}$ is the sum of three rows $r_0, r_1, r_2 \in R_1$. For $i = 1, 2, 3$ let c_i denote the column in the identity I_k in which row r_i is one. Then, since $r_0, r_1, r_2 \notin R_0$, all of the

weight 4 words in W must contain zeroes in columns c_0, c_1, c_2. Furthermore, since $\vec{w}$ is the sum of rows r_0, r_1, and r_2, $\vec{w}$ contains a one in each of the columns c_0, c_1, c_2, and since codewords in a self-dual code intersect in an even number of positions, this implies $\vec{w}$ cannot intersect any of the weight 4 words in W, which contradicts the condition that $\vec{w}$ intersects at least one weight 4 word in W. Therefore, $\vec{w}$ cannot be the sum of three rows from R_1.

Therefore, if $\vec{w}$ is the sum of three rows of $G_{B'}$, then $\vec{w}$ must be the sum of one row $r_0 \in R_0$ with two rows $r_1, r_2 \in R_1$:

r_0	1	0	0	0^*	1	1	1	0^*	
r_1	0	1	0	0^*	1	1	0	a	
r_2	0	0	1	0^*	1	0	1	a	$w(a) \geq 3$

This completes our proof. □

We will now use Lemma 4.15 to prove the following theorem, which, together with Lemma 4.15, proves that any weight 4 word $\vec{w'} \in B'$ that is not the sum of rows of R_0 exclusively, is either the sum of two rows from R_1 (as given in Lemma 4.15) or is the sum of one row from R_0 together with two rows from R_1 (as given in Lemma 4.15).

Theorem 4.16 Let $G_{B'}$ be a generator block for a weight 4 set B', such that $G_{B'}$ can be partitioned into the sets R_0 and R_1. Let W be the set of weight 4 words contained in the set of codewords generated by R_0. Suppose R_1 is not the empty set. Then for any weight 4 word $\vec{w'} \in B' - W$, there exists at least one weight 4 word $\vec{w} \in W$ such that $\vec{w'}$ intersects $\vec{w}$ in exactly two positions.

Proof Let $\vec{w'}$ be any weight 4 word in $B' - W$ and let $\vec{w}$ be any weight 4 word in W. Then, since $\vec{w'} \neq \vec{w}$, either $\vec{w'}$ intersects $\vec{w}$ in 2 positions, or $\vec{w}$ and $\vec{w'}$ do not intersect. Therefore, the weight 4 words in $B' - W$ can be partitioned into two disjoint sets:

- W_0 = the set of weight 4 words $\vec{w'} \in B' - W$ such that $\vec{w'}$ does not intersect any of the weight 4 words in W.

- W_2 = the set of weight 4 words $\vec{w'} \in B' - W$ for which there exists at least one $\vec{w} \in W$ such that $\vec{w}$ and $\vec{w'}$ intersect in 2 positions.

We want to show that $W_0 = \emptyset$.

Suppose $W_0 \neq \emptyset$. Then, since B' is a weight 4 set, there must exist weight 4 words $\vec{w_0} \in W_0$ and $\vec{w_2} \in W_2$ such that $\vec{w_0}$ and $\vec{w_2}$ intersect. (If this were not the case, then we could partition the weight 4 words of B' into two disjoint sets, $W \cup W_2$ and W_0, such that none of the weight 4 words in $W \cup W_2$ intersects any of the weight 4 words in W_0, which contradicts B' being a weight 4 set). Consider the rows of the generator block $G_{B'}$ that sum to the weight 4 word $\vec{w_2}$. By Lemma 4.15, $\vec{w_2}$ is either the sum of two rows from R_1, or $\vec{w_2}$ is the sum of one row from R_0 and two rows from R_1. Therefore, we have two cases to consider:

- Suppose $\vec{w_2}$ is the sum of two rows $r_1, r_2 \in R_1$:

	c_1	c_2		c_3	c_4		
r_1	1	0	0^*	1	0	a	
r_2	0	1	0^*	0	1	a	$w(a) \geq 4$

  Let c_1, c_2, c_3, c_4 be the four columns in which rows r_1 and r_2 sum to one. Then, since $r_1, r_2 \in R_1$, none of the weight 4 words in W contain ones in columns c_1 and c_2. Therefore, $\vec{w_2}$ must intersect a weight 4 word in W in columns c_3 and c_4. Since $\vec{w_0}$ does not intersect any of the weight 4 words in W, this implies $\vec{w_2}$ and $\vec{w_0}$ intersect in columns c_1 and c_2. Since $\vec{w_0} \neq \vec{w_2}$, this implies that the weight 4 word $\vec{w_0}$ must be the sum of the two rows r_1 and r_2 together with some third row r_3:

	c_1	c_2	c_x		c_3	c_4	c_y	
r_1	1	0	0	0^*	1	0	1	a
r_2	0	1	0	0^*	0	1	1	a
r_3	0	0	1	0^*	1	1	1	b

Since these three rows sum to a weight 4 word, we must have $b = a \oplus a$, which implies $b = 0^*$. This implies that row r_3 has weight 4, and hence row r_3 must be a row in R_0. But this implies that $\vec{w_0}$ intersects r_3, a weight 4 word in W (in columns c_x and c_y), which implies $\vec{w_0} \notin W_0$, which is a contradiction. Therefore $\vec{w_2}$ cannot be the sum of two rows from R_1.

- Suppose $\vec{w_2}$ is the sum of one row $r_0 \in R_0$ and two rows $r_1, r_2 \in R_1$:

	c_0	c_1	c_2		c_3				
r_0	1	0	0	0^*	1	1	1	0^*	
r_1	0	1	0	0^*	1	1	0	a	
r_2	0	0	1	0^*	1	0	1	a	$w(a) \geq 3$

  Let c_0, c_1, c_2, c_3 be the four columns in which rows r_0, r_1, and r_2 sum to one. Then, since $r_0 \in R_0$, $\vec{w_2}$ intersects a weight 4 word in W in columns c_0 and c_3. This implies that $\vec{w_2}$ and $\vec{w_0}$ intersect in columns c_1 and c_2, which implies that $\vec{w_0}$ is either the sum of the two rows r_1 and r_2, or $\vec{w_0}$ is the sum of three rows from $G_{B'}$, where two of these rows are rows r_1 and r_2. Therefore, we have two cases to consider:

  - Suppose $\vec{w_0}$ is the sum of rows r_1 and r_2. Then $\vec{w_0}$ intersects the weight 4 row r_0. Therefore $\vec{w_0}$ intersects a weight 4 word in W, which implies $\vec{w_0} \notin W_0$, which is a contradiction.

  - Suppose $\vec{w_0}$ is the sum of rows r_1 and r_2, plus some third row $r_3 \in G_{B'}$:

	c_1	c_2	c_x				c_y	
r_1	1	0	0	0^*	1	0	1	a
r_2	0	1	0	0^*	0	1	1	a
r_3	0	0	1	0^*	1	1	1	b

    Since these three rows sum to a weight 4 word, we must have $b = a \oplus a$, which implies $b = 0^*$. This implies that row r_3 has weight 4. Furthermore, row r_3 intersects the weight 4 word $r_1 \oplus r_2$, which intersects the weight 4 word $\vec{w_2}$

(in columns c_1 and c_2) which implies $r_1 \oplus r_2 \in B'$. Therefore, r_3 intersects a weight 4 word $r_1 \oplus r_2$ that is in B', which implies $r_3 \in B'$, and therefore must be a row in R_0. But this implies that $\vec{w_0}$ intersects a weight 4 word in W (in columns c_x and c_y), which implies $\vec{w_0} \notin W_0$, which is a contradiction.

Therefore $\vec{w_2}$ cannot be the sum of one row from R_0 and two rows from R_1.

Therefore, there do not exist two weight 4 words $\vec{w_0} \in W_0$ and $\vec{w_2} \in W_2$ such that $\vec{w_0}$ and $\vec{w_2}$ intersect. Thus $W_0 = \emptyset$. This completes our proof. □

Corollary 4.17 Let $G_{B'}$ be a generator block for a weight 4 set B', such that $G_{B'}$ can be partitioned into the sets R_0 and R_1. Let W denote the set of weight 4 words contained in the set of codewords generated by R_0. Suppose R_1 is not the empty set. Then if $\vec{w'}$ is a weight 4 word in $B' - W$ then $\vec{w'}$ is either the sum of two rows $r_1, r_2 \in R_1$, as follows:

r_1	1	0	0^*	1	0	a	
r_2	0	1	0^*	0	1	a	$w(a) \geq 4$

or $\vec{w'}$ is the sum of one row r_0 from R_0 and two rows r_1, r_2 from R_1, as follows:

r_0	1	0	0	0^*	1	1	1	0^*	
r_1	0	1	0	0^*	1	1	0	a	
r_2	0	0	1	0^*	1	0	1	a	$w(a) \geq 3$

Proof By Theorem 4.16, any $\vec{w'} \in B' - W$ intersects some weight 4 word in W in two positions, and by Lemma 4.15, any such $\vec{w'}$ is either the sum of two rows from R_1 (as given above) or is the sum of one row from R_0 and two rows from R_1 (as given above).

□

We are now ready to prove the two results that we will use to find all the different generator blocks $G_{B'}$, for weight 4 sets B', that can be partitioned into the two sets R_0

and R_1. That is, we will prove that the set of weight 4 rows R_0 must be a weight 4 block (and not the composition of two or more weight 4 blocks), and we will prove that if $R_1 \neq \emptyset$ then there must exist a weight 4 word $\vec{w'} \in B'$ that is the sum of two rows in R_1, such that $\vec{w'}$ intersects a row in R_0.

We will begin our proof of the first result, that the set of rows R_0 must be a weight 4 block, by proving a small lemma that immediately follows from the corollary to Theorem 4.16 and the fact that we are working with generator matrices of the form $G' = [I_k|A']$.

Lemma 4.18 Let $G_{B'}$ be a generator block for a weight 4 set B', such that $G_{B'}$ can be partitioned into the sets R_0 and R_1. Let W denote the set of weight 4 words contained in the set of codewords generated by R_0. Suppose R_1 is not the empty set. Let $\vec{w'}$ be any weight 4 word in $B' - W$ and let W' denote the subset of weight 4 words in W that $\vec{w'}$ intersects. Then $\vec{w'}$ intersects every weight 4 word in W' in the same two columns.

Proof By Corollary 4.17, $\vec{w'}$ is either the sum two rows from R_1 or $\vec{w'}$ is the sum of one row from R_0 and two rows from R_1. In particular, the subset of rows of $G_{B'}$ that sum to $\vec{w'}$ contains two rows from R_1. Let c_1 and c_2 denote the two columns in the identity I_k that contain ones in rows r_1 and r_2, respectively. Then, since the rows of $G_{B'}$ that sum to $\vec{w'}$ sum to one in columns c_1 and c_2, and since for every $\vec{w} \in W$ the set of rows in R_0 that sum to $\vec{w}$ sum to zero in columns c_1 and c_2 (since $r_1, r_2 \in R_1$), $\vec{w'}$ does *not* intersect any of the weight 4 words in W in columns c_1 and c_2. Therefore, since $\vec{w'}$ has weight 4, $\vec{w'}$ must intersect every weight 4 word in W' in the same two columns (i.e. the two columns, other than c_1 and c_2, in which the rows of $G_{B'}$ that sum to $\vec{w'}$ sum to 1 1). □

Theorem 4.19 Let $G_{B'}$ be a generator block for a weight 4 set B', such that $G_{B'}$ can be partitioned into the sets R_0 and R_1. Then R_0 is a weight 4 block.

Proof If R_1 is the empty set, then our result immediately follows, since R_0 generates a weight 4 set, and therefore must be a weight 4 block. Suppose R_1 is not the empty set. We want to show that R_0 must still be a weight 4 block.

Suppose R_0 is not a weight 4 block. That is, suppose R_0 can be partitioned into two disjoint sets of rows, R_0' and R_0'', such that none of the rows in R_0' intersects any of the rows in R_0''. Consider the following two disjoint sets of weight 4 words:

- W' = the set of weight 4 words contained in the set of codewords generated by R_0'.
- W'' = the set of weight 4 words contained in the set of codewords generated by R_0''.

Since none of the rows in R_0' intersects any of the rows in R_0'', none of the weight 4 words in W' intersects any of the weight 4 words in W''. Furthermore, since none of the rows in R_0' intersects any of the rows in R_0'' and since the rows in both R_0' and R_0'' have weight 4, $W' \cup W'' = W$ = the set of weight 4 words contained in the set of codewords generated by the set of rows $R_0 = R_0' \cup R_0''$.

Therefore, by Lemma 4.19 and since B' is a weight 4 set, there must exist two weight 4 words $\vec{w'}, \vec{w''} \in B' - W$ such that $\vec{w'}$ intersects a weight 4 word in W', $\vec{w''}$ intersects a weight 4 word in W'', and $\vec{w'}$ and $\vec{w''}$ intersect each other. If this were not the case, then B' could be partitioned into the two disjoint sets W_0' and W_0'', where W_0' is W' plus the weight 4 words in $B' - W$ that intersects at least one weight 4 word in W', and W_0'' is W'' plus the weight 4 words in $B' - W$ that intersect at least one weight 4 word in W''. But this implies that B' is not a weight 4 set, which is a contradiction.

Now, let $\vec{w'}$ and $\vec{w''}$ be two such weight 4 words. Then, by Corollary 4.17, since $\vec{w'}, \vec{w''} \in B' - W$, the subset of rows of $G_{B'}$ that sum to $\vec{w'}$ contains exactly two rows $r_1', r_2' \in R_1$, and the subset of rows of $G_{B'}$ that sum to $\vec{w''}$ contains exactly two rows $r_1'', r_2'' \in R_1$. Let c_1', c_2' be the two columns in I_k that contain ones in rows r_1', r_2', respectively, and let c_1'', c_2'' be the two columns in I_k that contain ones in rows r_1'', r_2'', respectively. Since $\vec{w'}$ intersects weight 4 words in W', $\vec{w''}$ does not intersect any weight 4 words in W'.

and since for every $\vec{w} \in W'$ the set of rows in R'_0 that sum to $\vec{w}$ sum to zero in columns c'_1, c'_2, $\vec{w'}$ must intersect $\vec{w''}$ in columns c'_1, c'_2. Similarly, $\vec{w''}$ must intersect $\vec{w'}$ in columns c''_1, c''_2. In other words, $c'_1 = c''_1$ and $c'_2 = c''_2$, which implies that $r'_1 = r''_1$ and $r'_2 = r''_2$. Therefore, the set of rows that sum to $\vec{w'}$ and the set of rows that sum to $\vec{w''}$ must both contain the same two rows $r_1, r_2 \in R_1$. Thus, since $\vec{w'} \neq \vec{w''}$, one of them, say $\vec{w'}$, is the sum of rows r_1 and r_2, and the other one, say $\vec{w''}$, is the sum of rows r_1 and r_2 together with a third row $r_0 \in R_0$:

	c_0				c_3	c_4	c_5		
r_0	1	0	0	0*	1	1	1	0*	
r_1	0	1	0	0*	1	1	0	a	
r_2	0	0	1	0*	1	0	1	a	$w(a) \geq 3$

Since r_1 and r_2 sum to one in columns c_4 and c_5, and since $\vec{w'} = r_1 \oplus r_2$ only intersects weight 4 words in W', $r_0 \in W'$, and thus $r_0 \in R'_0$. Since r_0, r_1, and r_2 sum to one in columns c_0 and c_3, and since $\vec{w''} = r_0 \oplus r_1 \oplus r_2$ only intersects weight 4 words in W'', $r_0 \in W''$, and thus $r_0 \in R''_0$. However, $R'_0 \cap R''_0 = \emptyset$, and we have a contradiction.

Therefore, R_0 cannot be partitioned into two disjoint sets of rows, R'_0 and R''_0, such that none of the rows in R'_0 intersects any of the rows in R''_0. Thus, R_0 must be a weight 4 block. □

We will now show that if $R_1 \neq \emptyset$ then there must exist a weight 4 word $\vec{w'} \in B'$ that is the sum of two rows in R_1, such that $\vec{w'}$ intersects a row in R_0.

Theorem 4.20 Let $G_{B'}$ be a generator block for a weight 4 set B', such that $G_{B'}$ can be partitioned into the sets R_0 and R_1. Let W denote the set of weight 4 words contained in the set of codewords generated by R_0. Suppose R_1 is not the empty set. Then there exists a weight 4 word $\vec{w'} \in B' - W$ such that $\vec{w'}$ is the sum of two rows $r_1, r_2 \in R_1$:

				c_1	c_2		
r_1	1	0	0*	1	0	a	
r_2	0	1	0*	0	1	a	$w(a) \geq 4$

where $\vec{w'}$ intersects some row $r_0 \in R_0$ in columns c_1 and c_2.

Proof Let $\vec{w'} \in B' - W$. Then, by Theorem 4.16, there exists some $\vec{w} \in W$ such that $\vec{w}$ and $\vec{w'}$ intersect in two positions. Furthermore, by Corollary 4.17, we know that $\vec{w'}$ is either the sum of two rows from R_1 or $\vec{w'}$ is the sum of one row from R_0 and two rows from R_1. Therefore, we have two cases to consider:

- Suppose $\vec{w'}$ is the sum of two rows $r_1, r_2 \in R_1$:

				c_1	c_2		
r_1	1	0	0*	1	0	a	
r_2	0	1	0*	0	1	a	$w(a) \geq 4$

  Let $\vec{w}$ be a weight 4 word in W that $\vec{w'}$ intersects. Then, since $\vec{w}$ is the sum of some subset of rows in R_0, for each column c in which $\vec{w}$ is one there must exist at least one row in R_0 that has a one in column c. Therefore, there must exist at least one row $r_0 \in R_0$ such that $\vec{w'}$ (which intersects $\vec{w}$) intersects row r_0. Furthermore, since $\vec{w'} = r_1 \oplus r_2$, where $r_1, r_2 \in R_1$, $\vec{w'}$ must intersect row r_0 in columns c_1 and c_2. Therefore, $\vec{w'}$ is a weight 4 word that meets the criteria of our theorem.

- Suppose $\vec{w'}$ is the sum of one row $r_0 \in R_0$ with two rows $r_1, r_2 \in R_1$:

					c_1	c_2			
r_0	1	0	0	0*	1	1	1	0*	
r_1	0	1	0	0*	1	1	0	a	
r_2	0	0	1	0*	1	0	1	a	$w(a) \geq 3$

  Let $\vec{w''}$ be the sum of rows r_1 and r_2. Then $\vec{w''}$ intersects row r_0 in columns c_1 and c_2. Therefore, $\vec{w''}$ is a weight 4 word that meets the criteria of our theorem.

This completes our proof. □

We are now ready to find all the different generator blocks $G_{B'}$, for weight 4 sets B', that can be partitioned into the sets R_0 and R_1. The possibilities for the set R_0 have

already been determined. By Theorem 4.19, they are the weight 4 blocks E_3, E_4, and D_i, $i \geq 1$. What we need to find is all the different sets R_1 that may occur with R_0. If $R_1 \neq \emptyset$, then we can make use of Theorem 4.20 to determine the possibilities for R_1. That is, we can make use of the fact that if $R_1 \neq \emptyset$, then there must exist a weight 4 word $\vec{w} \in B$ such that $\vec{w}$ is the sum of two rows in R_1 and $\vec{w}$ intersects a least one row in R_0 (in two columns that are columns of A' in $G' = [I_k|A']$).

Let us begin by having a look at the weight 4 block E_3 and finding the different sets R_1 that may occur with it. Let $G' = [I_k|A']$ be a generator matrix for a $(2k, k, 4)$ self-dual code C' such that G' contains a generator block $G_{B'}$, for a weight 4 set B', such that $G_{B'}$ can be partitioned into the two disjoint sets R_0 and R_1, where R_0 is a E_3-block. Then, with just the columns of A' given, we have:

$$\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \\ \hline 0 & 1 & 1 & 1 & X \\ 0 & 0 & 0 & 0 & Y \end{array}$$

The rows below the horizontal line represent all the ways the remaining rows of A' may occur with an E_3-block. They are determined using the fact that each of the remaining rows of A' must intersect each of the rows of the E_3-block in an even number of positions.

Since all the remaining rows of G' that intersect at least one of the rows of the E_3-block are identical in the columns of A' that contain a one in a row of the E_3-block, there does not exist a weight 4 word $\vec{w'}$ in C' that is the sum of two of the remaining rows in G' such that $\vec{w'}$ intersects a row of the E_3-block. Therefore, if R_0 is an E_3-block, R_1 must be the empty set. This gives us an e_3-block:

$$e_3 = \left[\begin{array}{cccc|ccccc} 1 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & 0^* \\ 0 & 1 & 0 & 0^* & 1 & 1 & 0 & 1 & 0^* \\ 0 & 0 & 1 & 0^* & 1 & 0 & 1 & 1 & 0^* \end{array}\right]$$

Let us now look at the different sets R_1 that may occur with the weight 4 block E_4. Let $G' = [I_k|A']$ be a generator matrix for a $(2k, k, 4)$ self-dual code C' such that G' contains a generator block $G_{B'}$, for a weight 4 set B', such that $G_{B'}$ can be partitioned into the two disjoint sets R_0 and R_1, where R_0 is a E_4-block. Then, with just the columns of A' given, we have:

$$\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \\ 0 & 1 & 1 & 1 & 0^* \\ \hline 0 & 0 & 0 & 0 & X \end{array}$$

The row below the horizontal line represents all the ways the remaining rows of A' may occur with an E_4-block.

Since none of the remaining rows in G' intersects any of the rows of the E_4-block, there does not exist a weight 4 word $\vec{w'}$ in C' that is the sum of two of the remaining rows in G' such that $\vec{w'}$ intersects a row of the E_4-block. Therefore, if R_0 is an E_4-block, R_1 must be the empty set. This gives us an e_4-block:

$$e_4 = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & 0^* \\ 0 & 1 & 0 & 0 & 0^* & 1 & 1 & 0 & 1 & 0^* \\ 0 & 0 & 1 & 0 & 0^* & 1 & 0 & 1 & 1 & 0^* \\ 0 & 0 & 0 & 1 & 0^* & 0 & 1 & 1 & 1 & 0^* \end{array}\right]$$

Let us now have a look at the different sets R_1 that may occur with the weight 4 block D_i, $i \geq 1$. Let $G' = [I_k|A']$ be a generator matrix for a $(2k, k, 4)$ self-dual code C' such that G' contains a generator block $G_{B'}$, for a weight 4 set B', such that $G_{B'}$ can be partitioned into the two disjoint sets R_0 and R_1, where R_0 is a D_i-block. Then, with just

the columns of A' given, we have:

$$\begin{array}{cccccccc} & & \overbrace{\quad\quad\quad} & & & & & \\ & & i & & & & & \\ 1 & 1 & 1 & 0 & \cdots & 0 & 0^* \\ 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \\ \hline 1 & 1 & 0 & 0 & \cdots & 0 & X \\ 1 & 0 & 1 & 1 & \cdots & 1 & Y \\ 0 & 1 & 1 & 1 & \cdots & 1 & Z \\ 0 & 0 & 0 & 0 & \cdots & 0 & W \end{array}$$

The rows below the horizontal line represent all the ways the remaining rows of A' may occur with a D_i-block. They are determined using the fact that each of the remaining rows of A' must intersect each of the rows of the D_i-block in an even number of positions.

The remaining rows of G' can sum to a weight 4 word $\vec{w'}$ such that $\vec{w'}$ intersects at least one of the rows of the D_i-block only if (with just the columns in A' given) we have two rows:

$$\begin{array}{cccccccc} 1 & 0 & 1 & 1 & \cdots & 1 & y & w(y) \equiv i \bmod 2 \\ 0 & 1 & 1 & 1 & \cdots & 1 & z & w(z) \equiv i \bmod 2 \end{array}$$

where $y = z$. If $w(y) > 0$ then an $\mathrm{ADD}_{r,c}$ involving any non-zero entry in y would result in a matrix A'', such that the generator block $G_{B''}$ in $G'' = [I_k|A'']$ that corresponds to the generator block $G_{B'} \in G'$ contains a D_{i+1}-block, which contradicts A' being in canonical form. Therefore, for two such rows to be rows in the set R_1 we must have $w(y) = 0$, which can only occur if $i \equiv 0 \bmod 2$. Furthermore, since the rows of R_1 all have weight ≥ 6, we require that $i \geq 4$. Therefore, the remaining rows of G' can sum to a weight 4 word $\vec{w'}$ such that $\vec{w'}$ intersects at least one of the rows of the D_i-block only if G' contains (with just the columns in A' given) the following two rows r_1 and r_2:

$$\begin{array}{cccccccc} r_1 & 1 & 0 & 1 & 1 & \cdots & 1 & 0^* \\ r_2 & 0 & 1 & 1 & 1 & \cdots & 1 & 0^* \end{array}$$

Together, $R_0 = D_i$ and $R_1 = \{r_1, r_2\}$, gives us an e_{2i}-block:

$$e_{2i} = \left[\begin{array}{ccccccc|ccccccc} \overbrace{1}^{} & 0 & \cdots & 0 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & \cdots & 0 & 0^* \\ 0 & 1 & \cdots & 0 & 0 & 0 & 0^* & 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & 0^* & 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \\ 0 & 0 & \cdots & 0 & 1 & 0 & 0^* & 1 & 0 & 1 & 1 & \cdots & 1 & 0^* \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0^* & 0 & 1 & 1 & 1 & \cdots & 1 & 0^* \end{array}\right] \begin{array}{l} \overbrace{}^{2i-2} \\ \overbrace{}^{2i-2} \\ \left.\begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}\right\}{\scriptstyle 2i-2} \end{array} \qquad i \geq 3$$

If G' contains an e_{2i}-block then, as is the case with an e_4-block, none of the remaining rows of G' can intersect any of the rows of the e_{2i}-block. Therefore, if R_0 is a D_i-block, the only other possibility for the set R_1 is the empty set, which gives us a d_i-block:

$$d_i = \left[\begin{array}{ccccc|ccccccc} \overbrace{1}^{} & 0 & \cdots & 0 & 0^* & 1 & 1 & 1 & 0 & \cdots & 0 & 0^* \\ 0 & 1 & \cdots & 0 & 0^* & 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0^* & 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \end{array}\right] \begin{array}{l} \overbrace{}^{i} \\ \overbrace{}^{i} \\ \left.\begin{array}{c} \\ \\ \\ \\ \end{array}\right\}{\scriptstyle i} \end{array} \qquad i \geq 1$$

This completes the case when R_0 is D_i-block, and we are done.

Thus, we have just shown that for any weight 4 set B of a $(2k, k, 4)$ self-dual code C, generated by $G = [I_K|A]$, there exists an equivalent code C', generated by $G' = [I_k|A']$, such that B', the weight 4 set in C' that corresponds to the weight 4 set B of C, is generated by a generator block $G_{B'}$, where $G_{B'}$ is one (and only one) of the following sets of rows:

$$e_3 = \left[\begin{array}{cccc|ccccc} 1 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & 0^* \\ 0 & 1 & 0 & 0^* & 1 & 1 & 0 & 1 & 0^* \\ 0 & 0 & 1 & 0^* & 1 & 0 & 1 & 1 & 0^* \end{array}\right]$$

$$e_{2i} = \begin{array}{c|c} \overbrace{1\ 0\ \cdots\ 0}^{2i-2}\ 0\ 0\ 0^* & 1\ 1\ \overbrace{1\ 0\ \cdots\ 0}^{2i-2}\ 0^* \\ 0\ 1\ \cdots\ 0\ 0\ 0\ 0^* & 1\ 1\ 0\ 1\ \cdots\ 0\ 0^* \\ \vdots\ \vdots\ \quad\ \vdots\ \vdots\ \vdots\ \vdots & \vdots\ \vdots\ \vdots\ \vdots\ \quad\ \vdots\ \vdots \\ 0\ 0\ \cdots\ 1\ 0\ 0\ 0^* & 1\ 1\ 0\ 0\ \cdots\ 1\ 0^* \\ 0\ 0\ \cdots\ 0\ 1\ 0\ 0^* & 1\ 0\ 1\ 1\ \cdots\ 1\ 0^* \\ 0\ 0\ \cdots\ 0\ 0\ 1\ 0^* & 0\ 1\ 1\ 1\ \cdots\ 1\ 0^* \end{array} \quad i \geq 2$$

$$d_i = \begin{array}{c|c} \overbrace{1\ 0\ \cdots\ 0}^{i}\ 0^* & 1\ 1\ \overbrace{1\ 0\ \cdots\ 0}^{i}\ 0^* \\ 0\ 1\ \cdots\ 0\ 0^* & 1\ 1\ 0\ 1\ \cdots\ 0\ 0^* \\ \vdots\ \vdots\ \quad\ \vdots\ \vdots & \vdots\ \vdots\ \vdots\ \vdots\ \quad\ \vdots\ \vdots \\ 0\ 0\ \cdots\ 1\ 0^* & 1\ 1\ 0\ 0\ \cdots\ 1\ 0^* \end{array} \quad i \geq 1$$

4.2.3 A List of All the Generator Blocks for Weight 4 Sets that may occur in $G = [I_k|A]$

We will conclude this section by finding all the different generator blocks G_B for weight 4 sets B that may occur in a generator matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code. The reason we will do this is as follows: Let C be any $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$. Then the weight 4 words of C form one or more weight 4 sets $B_1, B_2, \ldots, B_m$ which are generated by the disjoint sets of rows of G: $G_{B_1}, G_{B_2}, \ldots, G_{B_m}$. As we have just seen, for any particular weight 4 set B_i, we can find an equivalent code C', generated by $G' = [I_k|A']$, such that B_i', the weight 4 set produced by permuting B_i, is generated by either an e_3-block, an e_{2i}-block, or a d_i-block. However, it is not always possible to find a C', generated by $G' = [I_k|A']$, such that all the weight 4 sets $B_1, B_2, \ldots, B_m$ in C are permuted to weight 4 sets $B_1', B_2', \ldots, B_m'$ in which *every* weight 4 set B_i' is generated by either an e_3-block, an e_{2i}-block, or a d_i-block.

Example 4.21 Let C be the $(20, 10, 4)$ self-dual code generated by $G = [I_{10}|A]$ where:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{matrix}$$

C contains 2 weight 4 sets B_1 and B_2. B_1 is generated by the set of rows $\{r_1, r_2, r_3, r_4, r_5\}$ which form a d_5-block. B_2 is generated by the set of rows $\{r_6, r_7, r_8, r_9, r_{10}\}$. By applying an appropriate sequence of $\mathrm{ADD}_{r,c}$ operations on A, we can produce a matrix A' such that in the equivalent code C', generated by $G' = [I_k|A']$, the weight 4 set B_2' (the set that is produced by permuting B_2) is generated by a d_3-block. For example an $\mathrm{ADD}_{r,c}(9, 1)$ followed by an $\mathrm{ADD}_{r,c}(10, 2)$ is a sequence of $\mathrm{ADD}_{r,c}$ operations that results in a B_2' that is generated by a d_3-block. However, any such sequence of $\mathrm{ADD}_{r,c}$ operations will result in B_1 being permuted to a weight 4 set B_1' that is not generated by a d_5-block.

Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, 4)$ self-dual code C and let G_B be a generator block in G for a weight 4 set B of C. Let C' be any code equivalent to C that is generated by a matrix of the form $G' = [I_k|A']$ and let $G_{B'}$ be the generator block in G' that generates the weight 4 set B' in C' that is produced by permuting B. Then A', and therefore $G_{B'}$, can be found by applying some sequence of $\mathrm{ADD}_{r,c}$ operations to A, followed by some sequence of SWAP_r and SWAP_c operations. We shall say that any generator block $G_{B'}$ that can be found this way is *equivalent* to G_B. For a given G_B, by considering all the different rows that may occur with G_B in $G = [I_k|A]$, we can find

all the *different* generator blocks $G_{B'}$ that are equivalent to G_B by considering all the different $\mathrm{ADD}_{r,c}$ operations that affect the rows and columns of G_B in A. What is meant by different generator blocks is generator blocks $G_{B'}$ such that the rows and the columns of $G_{B'}$ are not simply permutations of one another. (Recall that the columns of $G_{B'}$ form the set of columns c of G' for which there exists a weight 4 word $\vec{w} \in B'$ such that the set of rows in $G_{B'}$ that sum to $\vec{w}$, sum to one in column c). Since any generator block $G_{B'}$ is equivalent to either an e_3-block, an e_{2i}-block, or a d_i-block, we can find a list of all the different generator blocks that may occur in a generator matrix of the form $G = [I_k|A]$ by finding all the different generator blocks that are equivalent to each of the e_3, e_{2i}, and d_i-blocks.

Example 4.22 Finding the different generator blocks that are equivalent to a d_1-block.

A d_1-block is simply a weight 4 word and we already know that any weight 4 word may occur in $G = [I_k|A]$ in one of three ways. Therefore, if C is a $(2k, k, 4)$ self-dual code, generated by $G = [I_k|A]$, such that C contains a weight 4 set that is generated by a generator block G_B that is equivalent to a d_1-block, then G_B generates a weight 4 set that contains only one weight 4 word and G_B consists of either one weight 4 row, two rows that sum to a weight 4 word, or three rows that sum to a weight 4 word. We shall refer to these generator blocks as $d_{1,0}$, $d_{1,3}$, and $d_{1,4}$-blocks. With just their columns in A given, they are:

$$\begin{array}{c|l} 1\ 1\ 1 & 0^* \end{array} \qquad \begin{array}{cc|l} 1 & 0 & x_3 \\ 0 & 1 & x_3 \end{array} \qquad \begin{array}{c|l} 1 & x_4 \\ 1 & y_4 \\ 1 & x_4 \oplus y_4 \end{array}$$

$d_{1,0}$ $\qquad$ $d_{1,3}$ $\qquad$ $d_{1,4}$

Remarks on notation: To the left of the vertical line are the columns of the generator block G_B. To the right of the vertical line are the ways in which the rows of G_B must occur together in the remaining columns of A.

Note that since the generator block $d_{1,3}$ generates a weight 4 set with only one weight 4 word, the rows of a $d_{1,3}$-block must have weight ≥ 6 which implies $w(x_3) \geq 4$. Since the generator block $d_{1,4}$ generates a weight 4 set with only one weight 4 word, the rows of a $d_{1,4}$-block must have weight ≥ 6 and any two rows of a $d_{1,4}$-block must sum to a codeword with weight ≥ 6 which implies $w(x_4), w(y_4), w(x_4 \oplus y_4) \geq 4$.

Example 4.23 Finding the different generator blocks that are equivalent to a d_2-block.

Let C be any $(2k, k, 4)$ self-dual code, generated by $G = [I_k|A]$, such that C contains a weight 4 set that is generated by a d_2-block. Then the rows and columns of A are a permutation of:

$$A_0 = \left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ \hline 1 & 1 & 0 & 0 & X_0 \\ 1 & 0 & 1 & 1 & Y_0 \\ 0 & 1 & 1 & 1 & Z_0 \\ 0 & 0 & 0 & 0 & W_0 \end{array}\right] \begin{array}{l} \\ \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{array}$$

Remarks on notation: Above the horizontal line are the rows of the d_2-block. To the left of the vertical line are the columns of the d_2-block. The rows denoted by R_3, R_4, R_5, and R_6 represent sets of rows. For example, the row 1 1 0 0 X_0 represents the set of all rows whose first four columns are 1 1 0 0. R_3, R_4, R_5, and R_6 represent all the different ways a row of A_0 may intersect a d_2-block.

Let us have a look at what happens to the d_2-block when an $\text{ADD}_{r,c}$ operation is applied to A_0:

- Since the columns in the rows of the d_2-block to the right of the vertical line consist of zeroes, any $\text{ADD}_{r,c}$ operation on any non-zero entry that occurs to the right of the vertical line in A_0 will not have any affect on the d_2-block. Therefore, we need only consider $\text{ADD}_{r,c}$ operations that involve the first four columns of A_0.

- An $\mathrm{ADD}_{r,c}(1,1)$ followed by a $\mathrm{SWAP}_c(2,3)$ reproduces A. With appropriate SWAP_c operations the same is true for $\mathrm{ADD}_{r,c}(1,2)$, $\mathrm{ADD}_{r,c}(1,3)$, $\mathrm{ADD}_{r,c}(2,1)$, $\mathrm{ADD}_{r,c}(2,2)$, and $\mathrm{ADD}_{r,c}(2,4)$.

- Suppose $|R_4| > 0$ and/or $|R_5| > 0$. That is, suppose A_0 contains a row $r_4 \in R_4$ and/or a row $r_5 \in R_5$. Consider the four $\mathrm{ADD}_{r,c}$ operations $\mathrm{ADD}_{r,c}(r_4,3)$, $\mathrm{ADD}_{r,c}(r_4,4)$, $\mathrm{ADD}_{r,c}(r_5,3)$, and $\mathrm{ADD}_{r,c}(r_5,4)$. While these four operations may result in different matrices, both have the same affect on the d_2-block. Together with the appropriate SWAP_r and/or SWAP_c operations, and some simplifications, both result in a matrix that looks like:

$$A_1 = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0^* \\ 1 & 0 & 1 & x_1 \\ 0 & 1 & 1 & x_1 \\ \hline 1 & 1 & 0 & Y_1 \\ 0 & 0 & 0 & Z_1 \end{array}\right]$$

  The rows above the horizontal line form a generator block that is equivalent to a d_2-block.

- Suppose $|R_4| > 0$ and/or $|R_5| > 0$. That is, suppose A_0 contains a row $r_4 \in R_4$ and/or a row $r_5 \in R_5$. Consider the two $\mathrm{ADD}_{r,c}$ operations $\mathrm{ADD}_{r,c}(r_4,1)$ and $\mathrm{ADD}_{r,c}(r_5,2)$. While these two operations may result in different matrices, both have the same affect on the d_2-block. Together with the appropriate SWAP_r and/or SWAP_c operations, and some simplifications, both result in a matrix that looks like:

$$A_2 = \left[\begin{array}{ccc|c} 1 & 1 & 0 & x_2 \\ 1 & 0 & 1 & x_2 \\ 0 & 1 & 1 & x_2 \\ \hline 1 & 1 & 1 & Y_2 \\ 0 & 0 & 0 & Z_2 \end{array}\right]$$

The rows above the horizontal line form a generator block that is equivalent to a d_2-block.

- Suppose $|R_3| > 0$. That is, suppose A_0 contains a row $r_3 \in R_3$. Consider the two $\mathrm{ADD}_{r,c}$ operations $\mathrm{ADD}_{r,c}(r_3, 1)$ and $\mathrm{ADD}_{r,c}(r_3, 2)$. While these two operations may result in different matrices, both have the same affect on the d_2-block. Together with the appropriate SWAP_r and/or SWAP_c operations, and some simplifications, both result in a matrix that looks like:

$$A_3 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & x_3 \\ 0 & 1 & 0 & x_3 \\ 0 & 0 & 1 & x_3 \\ \hline 1 & 1 & 1 & Y_3 \\ 0 & 0 & 0 & Z_3 \end{array}\right]$$

The rows above the horizontal line form a generator block that is equivalent to a d_2-block.

This covers all the $\mathrm{ADD}_{r,c}$ operations that we can apply to A_0. Thus, we have found that three different generator blocks equivalent to a d_2-block result from applying one $\mathrm{ADD}_{r,c}$ operation to the entries of A_0.

By trying all the different $\mathrm{ADD}_{r,c}$ operations on the entries of each of A_1, A_2, and A_3, we will either produce matrices with the same structure as A_0, A_1, A_2, or A_3 or we will produce a matrix that looks like:

$$A_4 = \left[\begin{array}{cc|c} 1 & 1 & x_4 \\ 1 & 1 & y_4 \\ 1 & 0 & x_4 \oplus y_4 \\ 0 & 1 & x_4 \oplus y_4 \\ \hline 0 & 0 & Z_4 \end{array}\right]$$

The rows above the horizontal line form a generator block that is equivalent to a d_2-block. By trying all the different $\mathrm{ADD}_{r,c}$ operations on the entries of A_4, we will find that we will produce matrices with the same structure as either A_1, A_2, A_3, or A_4.

Thus, including the d_2-block, we have found that there are five different generator blocks, that are equivalent to a d_2-block that may occur in a generator matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code. We shall refer to these blocks as $d_{2,0}$, $d_{2,1}$, $d_{2,2}$, $d_{2,3}$, and $d_{2,4}$-blocks. With just their columns in A given, they are:

$$\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \end{array} \qquad \begin{array}{ccc|c} 1 & 1 & 1 & 0^* \\ 1 & 0 & 1 & x_1 \\ 0 & 1 & 1 & x_1 \end{array} \qquad \begin{array}{ccc|c} 1 & 1 & 0 & x_2 \\ 1 & 0 & 1 & x_2 \\ 0 & 1 & 1 & x_2 \end{array} \qquad \begin{array}{ccc|c} 1 & 0 & 0 & x_3 \\ 0 & 1 & 0 & x_3 \\ 0 & 0 & 1 & x_3 \end{array} \qquad \begin{array}{cc|c} 1 & 1 & x_4 \\ 1 & 1 & y_4 \\ 1 & 0 & x_4 \oplus y_4 \\ 0 & 1 & x_4 \oplus y_4 \end{array}$$

$$d_{2,0} \qquad d_{2,1} \qquad d_{2,2} \qquad d_{2,3} \qquad d_{2,4}$$

A d_2-block generates a weight 4 set with three weight 4 words. With the rows of each generator block denoted by r_1, r_2, r_3, r_4, the sum of rows that sum to each of these words in each of the generator blocks that are equivalent to a d_2-block are:

- r_1, r_2, and $r_1 \oplus r_2$ in a $d_{2,0}$-block.
- r_1, $r_2 \oplus r_3$, and $r_1 \oplus r_2 \oplus r_3$ in a $d_{2,1}$-block.
- $r_1 \oplus r_2$, $r_1 \oplus r_3$, and $r_2 \oplus r_3$ in both the $d_{2,2}$ and $d_{2,3}$-blocks.
- $r_3 \oplus r_4$, $r_1 \oplus r_2 \oplus r_3$, and $r_1 \oplus r_2 \oplus r_4$ in a $d_{4,0}$-block.

Since any other combination of rows in each of these blocks must sum to a codeword with weight ≥ 6, we have the following bounds:

- in a $d_{2,1}$-block, $w(x_1) \geq 3$.
- in a $d_{2,2}$-block, $w(x_2) \geq 3$.

- in a $d_{2,3}$-block, $w(x_3) \geq 4$.
- in a $d_{2,4}$-block, $w(x_4), w(y_4) \geq 3$ and $w(x_4 \oplus y_4) \geq 4$.

Using the same method, we can find all the different generator blocks equivalent to each of the e_3, e_{2i}, $i \geq 2$, and d_i-blocks, $i \geq 3$, that may occur in a generator matrix of the form $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code.

A complete list of all the different generator blocks G_B that may occur in $G = [I_k|A]$ is now given. Just the columns from A are given. Equivalent generator blocks are listed together. Vertical lines are used to separate the columns of G_B from the remaining columns of A. Also given, below the horizontal lines, are all the different ways the remaining rows of A may intersect the rows of each generator block. Note, though not given, since the G_B-block generates a weight 4 set (and therefore, the rows of G_B that sum to a weight 4 word are the only rows of G that can both sum to a weight 4 word and sum to one in any of the columns of the G_B-block), there are restrictions on the weights of the remaining rows of A and how they can interact.

Generator Blocks Equivalent To e_3

Including e_3, there are two generator blocks equivalent to an e_3-block:

$e_{3,0}$

1	1	1	0	0*
1	1	0	1	0*
1	0	1	1	0*
0	1	1	1	X_0
0	0	0	0	Y_0

$e_{3,1}$

1	1	1	0*
1	1	0	x_1
1	0	1	x_1
0	1	1	x_1
0	0	0	Y_1

$$w(x_1) \equiv 1 \bmod 2$$
$$w(x_1) \geq 3$$

We shall refer to these generator blocks as $e_{3,0}$-blocks and $e_{3,1}$-blocks. When no distinction between them is necessary, we shall refer to them simply as e_3-blocks. Note that any $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$, where G contains either one of these two blocks, is equivalent to a code generated by $G' = [I_k|A']$, where G' contains an $e_{3,0}$-block.

Generator Blocks Equivalent To e_{2i}

Including e_{2i}, $i \geq 2$, there is one generator block equivalent to an e_4-block and two generator blocks equivalent to an e_{2i}-block, $i \geq 3$:

$e_{2i,0}$, $i \geq 2$:

$\overbrace{\quad\quad}^{2i-2}$						
1	1	1	0	$\cdots$	0	0^*
1	1	0	1	$\cdots$	0	0^*
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
1	1	0	0	$\cdots$	1	0^*
1	0	1	1	$\cdots$	1	0^*
0	1	1	1	$\cdots$	1	0^*
0	0	0	0	$\cdots$	0	X_0

$\}\ 2i-2$

$e_{2i,1}$, $i \geq 3$:

$\overbrace{\quad\quad\quad}^{2i}$						
1	1	$\cdots$	1	1	0	0^*
1	1	$\cdots$	1	0	1	0^*
1	1	$\cdots$	0	1	1	0^*
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	0	$\cdots$	1	1	1	0^*
0	1	$\cdots$	1	1	1	0^*
0	0	$\cdots$	0	0	0	X_1

$\}\ 2i$

We shall refer to these generator blocks as $e_{2i,0}$-blocks and $e_{2i,1}$-blocks. When no distinction between them is necessary, we shall refer to them simply as e_{2i}-blocks. Note that any $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$, where G contains either one of these two blocks (one block if $i = 2$), is equivalent to a code generated by $G' = [I_k|A']$, where G' contains an $e_{2i,0}$-block.

Generator Blocks Equivalent To d_i

Including d_i, $i \geq 1$, there are three generator blocks equivalent to a d_1-block, and five generator blocks equivalent to a d_i-block, $i \geq 5$:

$d_{i,0}$
$i \geq 1$

		$\overbrace{\quad\quad\quad}^{i}$				
1	1	1	0	$\cdots$	0	0^*
1	1	0	1	$\cdots$	0	0^*
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
1	1	0	0	$\cdots$	1	0^*
1	1	0	0	$\cdots$	0	X_0
1	0	1	1	$\cdots$	1	Y_0
0	1	1	1	$\cdots$	1	Z_0
0	0	0	0	$\cdots$	0	W_0

$d_{i,1}$
$i \geq 2$

		$\overbrace{\quad\quad}^{i-1}$				
1	1	1	0	$\cdots$	0	0^*
1	1	0	1	$\cdots$	0	0^*
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
1	1	0	0	$\cdots$	1	0^*
1	0	1	1	$\cdots$	1	x_1
0	1	1	1	$\cdots$	1	x_1
1	1	0	0	$\cdots$	0	Y_1
0	0	0	0	$\cdots$	0	W_1

$w(x_1) \equiv i+1 \bmod 2$

if $i = 2$ then $w(x_1) \geq 3$

if $i = 3$ then $w(x_1) \geq 2$

$d_{i,2}$
$i \geq 2$

		$\overbrace{\quad\quad\quad}^{i+1}$				
1	1	$\cdots$	1	1	0	x_2
1	1	$\cdots$	1	0	1	x_2
1	1	$\cdots$	0	1	1	x_2
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	0	$\cdots$	1	1	1	x_2
0	1	$\cdots$	1	1	1	x_2
1	1	$\cdots$	1	1	1	Y_2
0	0	$\cdots$	0	0	0	W_2

$w(x_2) \equiv i+1 \bmod 2$

if $i = 2$ then $w(x_2) \geq 3$

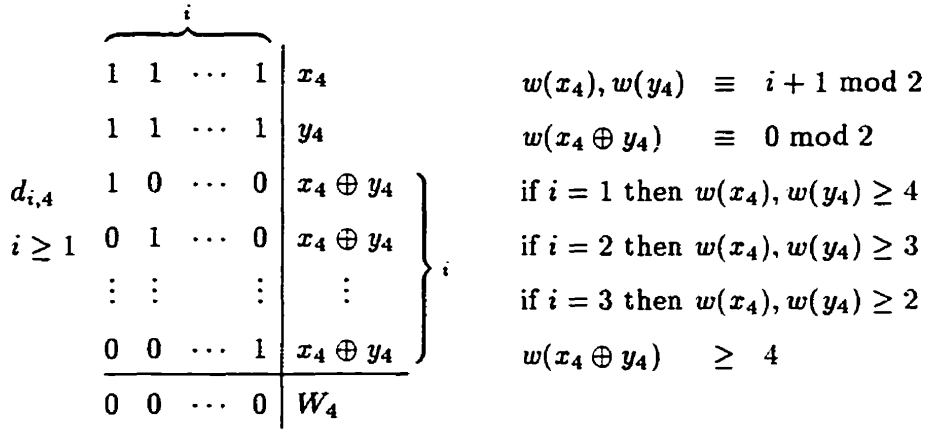
if $i = 3$ then $w(x_2) \geq 2$

$d_{i,3}$
$i \geq 1$

		$\overbrace{\quad\quad\quad}^{i+1}$				
1	0	0	$\cdots$	0	0	x_3
0	1	0	$\cdots$	0	0	x_3
0	0	1	$\cdots$	0	0	x_3
$\vdots$	$\vdots$			$\vdots$	$\vdots$	$\vdots$
0	0	0	$\cdots$	1	0	x_3
0	0	0	$\cdots$	0	1	x_3
1	1	$\cdots$	1	1	1	Y_3
0	0	$\cdots$	0	0	0	W_3

$w(x_3) \equiv 0 \bmod 2$

$w(x_3) \geq 4$



Note, for the generator blocks $d_{i,1}$, $d_{i,2}$, and $d_{i,3}$, if $i = 1$ then we get the same generator blocks. Therefore, we will only allow $i = 1$ for the generator block $d_{i,2}$.

We shall refer to these generator blocks as $d_{i,0}$-blocks, $d_{i,1}$-blocks, $d_{i,2}$-blocks, $d_{i,3}$-blocks, and $d_{i,4}$-blocks. When no distinction between them is necessary, we shall refer to them simply as d_i-blocks. Note that any $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$, where G contains either one of these five blocks (three blocks if $i = 1$), is equivalent to a code generated by $G' = [I_k|A']$, where G' contains an $d_{i,0}$-block.

4.3 Classifying the $(2k, k, 4)$ Self-Dual Codes

In this section, we will give a classification of the $(2k, k, 4)$ self-dual codes based on the weight 4 sets they contain. Let C be a $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$ and let G_B be a generator block in G for a weight 4 set B of C. Then we know that G_B is either an e_3-block, an e_{2i}-block, or a d_i-block. Since a $(2k, k, 4)$ self-dual code may contain more than one weight 4 set, we will give an order of precedence to the e_3-blocks, e_{2i}-blocks, $i \geq 2$, and the d_i-blocks, $i \geq 1$. In descending order, the precedence is:

$$\ldots, e_8, e_6, e_4, e_3, \ldots, d_i, d_{i-1}, \ldots, d_4, d_3, d_2, d_1.$$

We will classify our $(2k, k, 4)$ self-dual codes by the highest precedence weight 4 set generator block G_B they contain. We will refer to the code as a *G_B-code.*

Example 4.24 Let C be the $(12, 6, 4)$ self-dual code generated by $G = [I_6|A]$ where:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

C contains one weight 4 set, which is generated by an $e_{6,0}$-block. Thus, C will be classified as an e_6-code.

Example 4.25 Let C be the $(20, 10, 4)$ self-dual code generated by $G = [I_{10}|A]$ where:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{matrix}$$

C contains two weight 4 sets. One is generated by the $d_{5,1}$-block in rows r_1, r_2, r_3, r_4, r_9, and r_{10}. The other is generated by the $d_{3,1}$-block in rows r_5, r_6, r_7, and r_8. Since a d_5-block has higher precedence than a d_3-block, C will be classified as a d_5-code.

In the next section, we will use this classification to find algorithms for enumerating each class of $(2k, k, 4)$ self-dual codes.

4.4 Revised Enumeration of the $(2k, k, 4)$ Self-Dual Codes

Recall, from Section 4.1, that for any $(2k, k, 4)$ self-dual code there exists an equivalent code C, generated by $G = [I_k|A]$, where:

$$A = \mathcal{A}(A_0, \vec{v}) = \left[\begin{array}{cc|c} 1 & 0 & \vec{v} \\ 0 & 1 & \vec{v} \\ \hline & & \\ F & & A_0 \\ & & \\ & & \end{array}\right]$$

where $G_0 = [I_{k-2}|A_0]$ generates a $(2k-4, k-2)$ self-dual code C_0 and $\vec{v}$ is a $(k-2)$–vector with even weight ≥ 2. The rows of the $(k-2) \times 2$ matrix F are determined by A_0 and $\vec{v}$. Further recall that for any $(2k-4, k-2)$ self-dual code C_0' that is equivalent to C_0, where C_0' is generated by a matrix of the form $G_0' = [I_{k-2}|A_0']$, there exists a vector $\vec{v'}$ such that $G' = [I_k|\mathcal{A}(A_0', \vec{v'})]$ generates a $(2k, k, 4)$ self-dual code C' that is equivalent to C. We used this fact to develop a recursive algorithm that accepts as input a list L_0 of $(k-2) \times (k-2)$ matrices A_0 and that produces as output a list L of $k \times k$ matrices A. The input to the algorithm is any list L_0 of $(k-2) \times (k-2)$ matrices with the property that, for any $(2k-4, k-2)$ self-dual code C_0, there exists one and only one matrix A_0 in L_0 such that $G_0 = [I_{k-2}|A_0]$ generates a code equivalent to C_0. The output produced by the algorithm is a list L of $k \times k$ matrices with the property that, for any $(2k, k, 4)$ self-dual code C, there exists at least one matrix A in L such that $G = [I_k|A]$ generates a code equivalent to C. The algorithm works by finding all matrices $A = \mathcal{A}(A_0, \vec{v})$ for each matrix A_0 in L_0 and each $(k-2)$–vector $\vec{v}$ with even weight ≥ 2.

In this section, we will improve on this algorithm by finding a canonical $G = [I_k|A]$ for each class of $(2k, k, 4)$ self-dual codes: the e_3-codes, the e_{2i}-codes, and the d_i-codes. For the e_{2i}-codes, we will find that any e_{2i}-code is either simply a single e_{2i}-block or the composition of a $(2(k-2i), k-2i, d \geq 4)$ self-dual code with the code generated by an $e_{2i,0}$-block. Therefore, we can enumerate the e_{2i}-codes in a straightforward manner. For both the d_i-codes and the e_3-codes, we will find a canonical $G = [I_k|\mathcal{A}(A_1, \vec{v_1})]$ where $\mathcal{A}(A_1, \vec{v_1})$ is an extension matrix. For the d_i-codes, the matrix A_1 will turn out to be the composition of two matrices, $I_{i-1} \oplus A_0$, where I_{i-1} is a $(i-1) \times (i-1)$ identity matrix and $G_0 = [I_{k-i-1}|A_0]$ generates a $(2(k-i-1), k-i-1, d \geq 4)$ self-dual code, and the extension vector $\vec{v_1}$ will turn out to be the concatenation of two vectors, $1^{i-1}\vec{v}$, where $w(\vec{v}) \equiv i+1 \bmod 2$ and $w(\vec{v}) \geq 1$. For the e_3-codes, the matrix A_1 will turn out to be the composition of two matrices, $I_1 \oplus A_0$, where I_1 is a 1×1 identity matrix and $G_0 = [I_{k-3}|A_0]$ generates a $(2k-6, k-3, d \geq 4)$ self-dual code, and the extension vector $\vec{v_1}$ will turn out to be the concatenation of two vectors, $1\vec{v}$, where either $w(\vec{v}) = 1$ or $\vec{v}$ is a row of A_0. From this we will find that the d_i-codes can be found from the $(2(k-i-1), k-i-1, d \geq 4)$ self-dual codes and the e_3-codes can be found from the $(2k-6, k-3, d \geq 4)$ self-dual codes.

4.4.1 Enumeration of the e_{2i}-codes, $i \geq 2$

Recall, from Section 4.2, that any $(2k, k, 4)$ e_{2i}-code, $2 \leq i \leq k/2$, is equivalent to a code C, generated by $G = [I_k|A]$, where G contains an $e_{2i,0}$-block. Furthermore, since none of

the remaining rows of G can intersect any of the rows of the $e_{2i,0}$-block, we must have:

$$A = \left[\begin{array}{ccccccc|c} & & & \overbrace{}^{2i-2} & & & & \\ 1 & 1 & 1 & 0 & \ldots & 0 & & \\ 1 & 1 & 0 & 1 & \ldots & 0 & & \\ \vdots & \vdots & \vdots & \vdots & & \vdots & & 0 \\ 1 & 1 & 0 & 0 & \cdots & 1 & & \\ 1 & 0 & 1 & 1 & \cdots & 1 & & \\ 0 & 1 & 1 & 1 & \cdots & 1 & & \\ \hline & & & 0 & & & & A_0 \end{array}\right]$$

If $2i = k$ then $G = [I_k|A]$ is simply an e_{2i}-block. Otherwise, $G_0 = [I_{k-2i}|A_0]$ generates a $(2(k-2i), k-2i)$ self-dual code C_0. Since C has distance 4, C_0 must have distance ≥ 4. Furthermore, since C is an e_{2i}-code, G_0 cannot contain a generator block with higher precedence than an e_{2i}-block (i.e. G_0 cannot contain an $e_{2(i+j)}$-block, $j \geq 1$). Also, since the smallest self-dual code with distance ≥ 4 is the $(8,4,4)$ self-dual code generated by an $e_{4,0}$-block, we must have $k - 2i \geq 4$ which implies that $i \leq (k-4)/2$.

Therefore, any $(2k, k, 4)$ e_{2i}-code C is either equivalent to the $(2k, k, 4)$ self-dual code generated by an $e_{k,0}$-block, where $k \equiv 0 \bmod 2$ and $k \geq 4$, or C is equivalent to the composition of the code generated by an $e_{2i,0}$-block and a $(2(k-2i), k-2i, d \geq 4)$ self-dual code C_0, generated by $G_0 = [I_{k-2i}|A_0]$, where G_0 does not contain a generator block with higher precedence than an e_{2i}-block. Thus, given any list L_{2k-i}, $2 \leq i \leq \lfloor (k-4)/2 \rfloor$, of $(k-2i) \times (k-2i)$ matrices A_0, such that for any $(2(k-2i), k-2i, d \geq 4)$ self-dual code C_0 there exists one and only one A_0 in L_{k-2i} such that $G_0 = [I_{k-2i}|A_0]$ generates a code equivalent to C_0, if for each A_0 in L_{k-2i} (such that $G_0 = [I_k|A_0]$ does not contain an e_{2i+j}-block, $j \geq 1$) we produce the list L_{2i} of $k \times k$ matrices $A = e_{2i,0} \oplus A_0$, then for any $(2k, k, 4)$ e_{2i}-code C, there will exist an A in L_{2i} such that $G = [I_k|A]$ generates a code equivalent to C. Furthermore, since for any pair $A_0, A_0' \in L_{k-2i}$, $G_0 = [I_{k-2i}|A_0]$ and $G_0' = [I_{k-2i}|A_0']$ generate inequivalent codes, for any pair $A, A' \in L_{2i}$, $G = [I_k|A]$ and

$G' = [I_k|A']$ generate inequivalent codes. This gives us our algorithm for enumerating the e_{2i}-codes.

Algorithm 4.26 :

- Input: a list L_{k-2i} of $(k-2i) \times (k-2i)$ matrices A_0, $2 \leq i \leq \lfloor (k-4)/2 \rfloor$, where $G_0 = [I_{k-2i}|A_0]$ generates a $(2(k-2i), k-2i, d \geq 4)$ self-dual code, such that for any $(2(k-2i), k-2i, d \geq 4)$ self-dual code C_0 there exists one and only one matrix A_0 in L_{k-2i} such that $G_0 = [I_{k-2i}|A_0]$ generates a code equivalent to C_0.
- Output: a list L of $k \times k$ matrices A, where $G = [I_k|A]$ generates a $(2k, k, 4)$ e_{2i}-code, $1 \leq i \leq k/2$, such that for any $(2k, k, 4)$ e_{2i}-code C there exists at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C.

begin
  Let $L = \emptyset$.
  if $k \geq 4$ and $k \equiv 0 \bmod 2$ **then**
    Insert $e_{k,0}$ into L.
  for $i = 2$ to $\lfloor (k-4)/2 \rfloor$ **do**
    for each $A_0 \in L_{k-2i}$ **do**
      if A_0 does not contain a block with higher precedence than an e_{2i}-block **then**
        Let $A = e_{2i} \oplus A_0$.
        Insert A into L.
      end if
end

We now have our algorithm for enumerating the $(2k, k, 4)$ e_{2i}-codes. Next, we will develop an algorithm for enumerating the $(2k, k, 4)$ d_i-codes.

4.4.2 Enumeration of the d_i-codes, $i \geq 1$

In this subsection, we will find a canonical form for generator matrices of d_i-codes which shows that, for any $(2k, k, 4)$ d_i-code, $i \geq 1$, there exists an equivalent code generated by $G = [I_k | A(A_1, \vec{v_1})]$ where $A_1 = I_{i-1} \oplus A_0$ and $\vec{v_1} = 1^{i-1}\vec{v}$. We will then use this canonical G to develop an algorithm for enumerating the $(2k, k, 4)$ d_i-codes.

Recall, from Section 4.2, that any $(2k, k, 4)$ d_i-code is equivalent to a code C, generated by $G = [I_k | A]$, where the rows of A are as follows:

$$
\begin{array}{c|cccccc|c}
 & \overbrace{}^{i} & & & & & & \\
 & 1 & 1 & 1 & 0 & \cdots & 0 & 0^* \\
 & 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\
 & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\
 & 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \\
\hline
a & 1 & 1 & 0 & 0 & \cdots & 0 & X_0 \\
b & 1 & 0 & 1 & 1 & \cdots & 1 & Y_0 \\
c & 0 & 1 & 1 & 1 & \cdots & 1 & Z_0 \\
 & 0 & 0 & 0 & 0 & \cdots & 0 & W_0
\end{array}
\qquad \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right\} i
$$

Above the horizontal line are the rows of a $d_{i,0}$-block. Below the horizontal line are all the different possibilities for the remaining rows of A. They represent all the different ways in which a row of A may intersect a $d_{i,0}$-block.

Let a, b, and c represent the number of times each of the rows that intersect the $d_{i,0}$-block in two or more positions occur in A. Then the first column of A has weight $i + a + b$ and the second column of A has weight $i + a + c$. Furthermore, the first two columns of A intersect in $i + a$ positions. Since $G = [I_k | A]$ generates a self-dual code, the columns of A have odd weight and intersect in an even number of positions. Therefore:

$$
\begin{aligned}
i + a + b &\equiv 1 \bmod 2 \\
i + a + c &\equiv 1 \bmod 2
\end{aligned}
$$

$$i + a \equiv 0 \bmod 2$$

Solving these equations for b and c we get $b \equiv c \equiv 1 \bmod 2$. Therefore, $c \geq 1$, and thus A must contain a row:

$$0 \quad 1 \quad 1 \quad 1 \quad \cdots \quad 1 \quad z$$

Replacing z with $\vec{v'}$ and the sets of rows X_0, Y_0, Z_0 (minus z), and W_0 with the matrix M', we have the result that for any $(2k, k, 4)$ d_i-code there exists an equivalent code C', generated by $G' = [I_k | A']$, where:

$$A' = \left[\begin{array}{cc|\overbrace{ccccc}^{i}|c} 1 & 1 & 1 & 0 & \cdots & 0 & 0 & \\ 1 & 1 & 0 & 1 & \cdots & 0 & 0 & \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 1 & 0 & \\ 1 & 1 & 0 & 0 & \cdots & 0 & 1 & \\ \hline 0 & 1 & 1 & 1 & \cdots & 1 & 1 & \vec{v'} \\ \hline 1 & 1 & 0 & 0 & \cdots & 0 & 0 & \\ 1 & 0 & 1 & 1 & \cdots & 1 & 1 & M' \\ 0 & 1 & 1 & 1 & \cdots & 1 & 1 & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \end{array}\right]$$

M' is a $(k - i - 1) \times (k - i - 2)$ matrix and $\vec{v'}$ is a $(k - i - 2)$–vector. If we perform an

$\mathrm{ADD}_{r,c}$ operation on the boldface entry in A', we produce the matrix:

$$A'' = \left[\begin{array}{cc|c|c|c} 1 & 1 & \overbrace{1\ 0\ \ldots\ 0}^{i-1} & 0 & \\ 1 & 1 & 0\ 1\ \ldots\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots\ \vdots\ \quad\ \vdots & \vdots & \\ 1 & 1 & 0\ 0\ \cdots\ 1 & 0 & \\ \hline 1 & 0 & 1\ 1\ \cdots\ 1 & 1 & \vec{v'} \\ 0 & 1 & 1\ 1\ \cdots\ 1 & 1 & \vec{v'} \\ \hline 1 & 1 & 0\ 0\ \cdots\ 0 & 0 & \\ 1 & 1 & 0\ 0\ \cdots\ 0 & 1 & M'' \\ 0 & 0 & 0\ 0\ \cdots\ 0 & 1 & \\ 0 & 0 & 0\ 0\ \cdots\ 0 & 0 & \end{array}\right]$$

M'' is a $(k-i-1)\times(k-i-2)$ matrix and $\vec{v'}$ is a $(k-i-2)$–vector. $G'' = [I_k|A'']$ generates a $(2k,k,4)$ self-dual code code equivalent to C'. If we permute the i^{th} and $(i+1)^{th}$ rows of A'' above the first $i-1$ rows of A'' and then perform various simplifications, we get the result that for any $(2k,k,4)$ d_i-code there exists an equivalent code C, generated by $G = [I_k|A]$, where:

$$A = \left[\begin{array}{cc\|c|c} 1 & 0 & 1\ 1\ \ldots\ 1 & \vec{v} \\ 0 & 1 & 1\ 1\ \ldots\ 1 & \vec{v} \\ \hline\hline 1 & 1 & & \\ 1 & 1 & I_{i-1} & 0 \\ \vdots & \vdots & & \\ 1 & 1 & & \\ \hline & & & \\ F & & 0 & A_0 \\ & & & \end{array}\right]$$

where A_0 is a $(k-i-1)\times(k-i-1)$ matrix, $\vec{v}$ is a $(k-i-1)$–vector, and F is a $(k-i-1)\times 2$ matrix. This is our canonical $G = [I_k|A]$ for the $(2k,k,4)$ d_i-codes. Note that if $i \geq 2$ then the first $i+1$ rows of G form a $d_{i,1}$-block, and if $i = 1$ then the first 2 rows of G form a $d_{1,3}$-block.

Notice that A is the extension matrix $\mathcal{A}(I_{i-1} \oplus A_0, 1^{i-1}\vec{v})$. Therefore, $[I_{k-2}|I_{i-1} \oplus A_0]$

generates a $(2k-4, k-2)$ self-dual code which implies that $G_0 = [I_{k-i-1}|A_0]$ generates a $(2(k-i-1), k-i-1)$ self-dual code C_0. Furthermore, since any sequence of $\mathrm{ADD}_{r,c}$, SWAP_r, and SWAP_c operations on the entries of A_0 in A have no effect on the identity matrix I_{i-1} nor the vector 1^{i-1}, for any $(2(k-i-1), k-i-1)$ self-dual code C_0' that is equivalent to C_0, where C_0' is generated by a matrix of the form $G_0' = [I_{k-i-1}|A_0']$, there exists a vector $\vec{v'}$ such that $G' = [I_k|\mathcal{A}(I_{i-1} \oplus A_0', 1_{i-1}\vec{v'})]$ generates a $(2k, k, 4)$ self-dual code C' that is equivalent to the $(2k, k, 4)$ d_i-code generated by $G = [I_k|A]$. Thus, given any list L_0 of $(k-i-1)\times(k-i-1)$ matrices A_0, such that for any $(2(k-i-1), k-i-1)$ self-dual code C_0 there exists one and only one A_0 in L_0 such that $G_0 = [I_{k-i-1}|A_0]$ generates a code equivalent to C_0, if for each A_0 in L_0 and each $\vec{v}$ such that $w(\vec{v}) \equiv (i-1) \bmod 2$, we produce the matrix $A = \mathcal{A}(I_{i-1} \oplus A_0, 1^{i-1}\vec{v})$, then the resulting list L of $k \times k$ matrices A will have the property that for any $(2k, k, 4)$ d_i-code C there will exist at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C.

The matrix $\mathcal{A}(I_{i-1} \oplus A_0, 1_{i-1}\vec{v})$ will be referred to as an *extension matrix* for the $(2k, k, 4)$ d_i-codes. We will use the notation $\mathcal{A}_i(A_0, \vec{v})$ to denote $\mathcal{A}(I_{i-1} \oplus A_0, 1_{i-1}\vec{v})$. The $(k-i-1)$–vector $\vec{v}$ will be referred to as an *extension vector* for the d_i-codes. We shall call the process of producing then generator matrix $G = [I_k|\mathcal{A}_i(A_0, \vec{v})]$ *extending* the $(2(k-i-1), k-i-1)$ self-dual code C_0, generated by $G_0 = [I_{k-i-1}|A_0]$, to the $(2k, k, 4)$ d_i-code C, generated by G.

While it is true that for any $(2k, k, 4)$ d_i-code there exists an equivalent code C generated by a matrix of the form $G = [I_k|\mathcal{A}(A_0, \vec{v})]$, where $G_0 = [I_k|A_0]$ generates a $(2(k-i-1), k-i-1)$ self-dual code and $w(\vec{v}) \equiv (i-1) \bmod 2$, it is not true that any matrix of the form $G = [I_k|\mathcal{A}(A_0, \vec{v})]$ generates a $(2k, k, 4)$ d_i-code. Therefore, when we use the extension process to produce the d_i-codes, we must check whether or not the highest precedence generator block in the matrices we produce is a $d_{i,0}$.

Before giving a formal description of our algorithm for enumerating the $(2k, k, 4)$ d_i-codes, we will have a closer look at the weight of the extension vector $\vec{v}$ and the distance

of the $(2(k-i-1), k-i-1)$ self-dual code C_0, generated by $G_0 = [I_{k-i-1}|A_0]$. Let us first have a look at the weight of the extension vector $\vec{v}$. More specifically, let us find lower bounds on the weight of $\vec{v}$. We can find such bounds by using the facts that $w(\vec{v}) \equiv (i-1) \bmod 2$ and that if $i \geq 2$ then the first $i+1$ rows of $A = \mathcal{A}_i(A_0, \vec{v})$ form a $d_{i,1}$-block, and if $i = 1$ then the first 2 rows of A form a $d_{1,3}$-block. Since $w(\vec{v}) \equiv (i-1) \bmod 2$, if $i \equiv 0 \bmod 2$, $w(\vec{v}) \geq 1$. If $i \equiv 1 \bmod 2$ and $i \geq 3$ then since the first $i+1$ rows of A form a $d_{i,1}$-block, $w(\vec{v}) \geq 2$ (otherwise the first $i+1$ rows of A would form an $e_{i+1,0}$-block). If $i = 2$ then $w(\vec{v}) \geq 3$ (otherwise the first three rows of A would form an $e_{3,0}$-block). If $i = 1$ then $w(\vec{v}) \geq 4$ (if $w(\vec{v})$ were zero we would have a row in A with weight 1 and if $w(\vec{v})$ were two, the first two rows of A would form a $d_{2,0}$-block and not a $d_{1,3}$-block). Therefore, if $i \geq 3$ then $w(\vec{v}) \geq 1$, if $i = 2$ then $w(\vec{v}) \geq 3$, and if $i = 1$ then $w(\vec{v}) \geq 4$.

Let us now consider the distance of the $(2(k-i-1), k-i-1)$ self-dual code C_0 generated by $G_0 = [I_{k-i-1}|A_0]$. Suppose C_0 has distance 2. That is, suppose A_0 contains a row r with weight 1. Consider the corresponding row in A. Since the rows of A have weight ≥ 3, this row must contain 1 1 in F. But this implies that the first $i+1$ rows of G together with the row corresponding to row r form a $d_{i+1,1}$-block, which contradicts the fact that the first $i+1$ rows of G form a $d_{i,1}$-block if $i \geq 2$ and a $d_{1,3}$-block if $i = 1$. Therefore, C_0 cannot have distance 2. Thus, C_0 has distance ≥ 4.

Therefore, given any list L_0 of $(k-i-1) \times (k-i-1)$ matrices A_0, such that for any $(2(k-i-1), k-i-1, d \geq 4)$ self-dual code C_0 there exists one and only one A_0 in L_0 such that $G_0 = [I_{k-i-1}|A_0]$ generates a code equivalent to C_0, we can find a list L of $k \times k$ matrices A such that for any $(2k, k, 4)$ d_i-code C there exists at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C, by finding all matrices $A = \mathcal{A}_i(A_0, \vec{v})$, where A_0 in L_0, $w(\vec{v}) \equiv (i-1) \bmod 2$, and $w(\vec{v}) \geq 1$ if $i \geq 3$, $w(\vec{v}) \geq 3$ if $i = 2$, and $w(\vec{v}) \geq 4$ if $i = 1$. This gives us our algorithm for enumerating the $(2k, k, 4)$ d_i-codes.

Algorithm 4.27 :

- Input: a list L_0 of $(k-i-1) \times (k-i-1)$ matrices A_0, where $G_0 = [I_{k-i-1}|A_0]$ generates a $(2(k-i-1), k-i-1, d \geq 4)$ self-dual code, such that for any $(2(k-i-1), k-i-1, d \geq 4)$ self-dual code C_0 there exists one and only one matrix A_0 in L_0 such that $G_0 = [I_{k-i-1}|A_0]$ generates a code equivalent to C_0.
- Output: a list L of $k \times k$ matrices A, where $G = [I_k|A]$ generates a $(2k, k, 4)$ d_i-code, such that for any $(2k, k, 4)$ d_i-code C there exists at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C.

begin
 for each $A_0 \in L_0$ **do**
  for each $(k-i-1)$–vector $\vec{v}$ such that $w(\vec{v}) \equiv i+1 \bmod 2$ and $w(\vec{v}) \geq 1$ **do**
   (Note that if $i = 1$ then $w(\vec{v}) \geq 4$ and if $i = 2$ then $w(\vec{v}) \geq 3$)
    Let A=$\mathcal{A}_i(A_0, \vec{v})$.
    if A does not contain a block with higher precedence than a d_i-block **then**
     QuickOrderMatrix$(A, 0, A')$.
     if $A' \notin L$ **then**
      Insert A' into L.
    end if
  end for
 end for
end

We now have our algorithm for enumerating the $(2k, k, 4)$ d_i-codes. Next, we will develop an algorithm for enumerating the $(2k, k, 4)$ e_3-codes.

4.4.3 Enumeration of the e_3-codes

In this subsection, we fill find a canonical form for generator matrices of e_3-codes that shows us that for any $(2k, k, 4)$ e_3-code there exists an equivalent code generated by $G = [I_k|\mathcal{A}(A_1, \vec{v_1})]$ where $A_1 = I_1 \oplus A_0$ and $\vec{v_1} = 1\vec{v}$. We will then use this canonical G to develop an algorithm for enumerating the $(2k, k, 4)$ e_3-codes. Notice that the extension matrix $\mathcal{A}(I_1 \oplus A_0, 1\vec{v})$ for an e_3-code is equal to $\mathcal{A}_2(A_0, \vec{v})$, which is the extension matrix for a d_2-code. The difference, as we shall see, is in the vector $\vec{v}$.

Let us first find our canonical form for generator matrices of e_3-codes. Recall, from Section 4.2, that any $(2k, k, 4)$ e_3-code is equivalent to a code C, generated by $G = [I_k|A]$, where the rows of A are as follows:

	1	1	1	0	0^*
	1	1	0	1	0^*
	1	0	1	1	0^*
a	0	1	1	1	X_0
	0	0	0	0	Y_0

Above the horizontal line are the rows of an $e_{3,0}$-block. Below the horizontal line are all the different possibilities for the remaining rows of A. They represent all the different ways in which a row of A may intersect an $e_{3,0}$-block.

Let a represent the number of times a row with the form $0111x$ occurs in A. Then the second, third and fourth columns of A all have weight $a + 2$. Since $G = [I_k|A]$ generates a self-dual code, the columns of A must have odd weight. Therefore, $a + 2 \equiv 1 \bmod 2$ which implies that $a \geq 1$. Thus, A must contain a row $0111x$. Replacing x with $\vec{v'}$ and the set of rows X_0 (minus x) and Y_0 with the matrix M', we have the result that for any

$(2k, k, 4)$ e_3-code, there exists an equivalent code C', generated by $G' = [I_k|A']$, where:

$$A' = \left[\begin{array}{c|ccc|c} 1 & 1 & 1 & 0 & \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & \\ \hline 0 & 1 & 1 & 1 & \vec{v_1} \\ \hline 0 & 1 & 1 & 1 & \\ \vdots & \vdots & \vdots & \vdots & M_1 \\ 0 & 0 & 0 & 0 & \\ \vdots & \vdots & \vdots & \vdots & \end{array}\right]$$

M_1 is a $(k-4) \times (k-4)$ matrix and $\vec{v_1}$ is a $(k-4)$–vector. Note that $w(\vec{v_1}) \geq 2$, for if $w(\vec{v_1})$ were zero then the first four rows of A' would form an $e_{4,0}$-block. By performing various SWAP_r and SWAP_c operations on A, we get the result that for any $(2k, k, 4)$ e_3-code, there exists an equivalent code C'', generated by $G'' = [I_k|A'']$, where:

$$A'' = \left[\begin{array}{cc\|c|c|c} 1 & 0 & 1 & 1 & 0^* \\ 0 & 1 & 1 & 1 & 0^* \\ \hline\hline 1 & 1 & 1 & 0 & 0^* \\ \hline 1 & 1 & 0 & 1 & \vec{v_1} \\ \hline 1 & 1 & 0 & 1 & \\ \vdots & \vdots & \vdots & \vdots & M_1 \\ 0 & 0 & 0 & 0 & \\ \vdots & \vdots & \vdots & \vdots & \end{array}\right]$$

By replacing the $(k-3) \times (k-3)$ submatrix in the the last $k-3$ rows and last $k-3$ columns of A'' (i.e. the matrix M_1, the row above M_1, and the column to the left of M_1) with A_0 and replacing the vector 10^* in the last $k-3$ columns of the first 2 rows of A'' with the vector $\vec{v}$, we get the result that for any $(2k, k, 4)$ e_3-code there exists an equivalent code C, generated by $G = [I_k|A]$, where:

$$A = \left[\begin{array}{cc\|c|c} 1 & 0 & 1 & \vec{v} \\ 0 & 1 & 1 & \vec{v} \\ \hline\hline 1 & 1 & 1 & 0^* \\ \hline 1 & 1 & 0 & \\ \vdots & \vdots & \vdots & A_0 \\ 0 & 0 & 0 & \\ \vdots & \vdots & \vdots & \end{array}\right]$$

and A_0 is a $(k-3)\times(k-3)$ matrix, $\vec{v}$ is a $(k-3)$–vector, and $w(\vec{v})=1$.

Notice that A is the extension matrix $\mathcal{A}(I_1 \oplus A, 1\vec{v}) = \mathcal{A}_2(A,\vec{v})$. Therefore, $G_0 = [I_{k-3}|A_0]$ generates a $(2k-6,k-3)$ self-dual code C_0. However, the input list of $(k-3)\times(k-3)$ matrices we will use in our algorithm for enumerating the e_3-codes may not contain A_0, but instead a matrix A_0' where $G_0' = [I_k|A_0']$ generates a $(2k-6,k-3)$ self-dual code C_0' that is equivalent to C_0. Since we can find any such A_0' by applying sequences of $\mathrm{ADD}_{r,c}$ operations (together with appropriate SWAP_r and SWAP_c operations) to A_0, we would like to know what effect any sequence of $\mathrm{ADD}_{r,c}$ operations on the entires of A_0 in A has on A. More specifically, we would like to determine the effect that any sequence of $\mathrm{ADD}_{r,c}$ operations on the entries of A_0 in A has on the extension vector $\vec{v}$. From this, we will be able to determine the properties that any vector $\vec{v'}$ must posses for the matrix $G' = [I_k|\mathcal{A}_2(A_0',\vec{v'})]$ to be the generator matrix for a $(2k,k,4)$ e_3-code.

Let us determine the effects that any sequence of $\mathrm{ADD}_{r,c}$ operations on the entries of A_0 in A have on $\vec{v}$. Recall that A is a generalization of the following $k\times k$ matrix:

$$A'' = \left[\begin{array}{cc|cc|c} 1 & 0 & 1 & 1 & 0^* \\ 0 & 1 & 1 & 1 & 0^* \\ \hline 1 & 1 & 1 & 0 & 0^* \\ \hline 1 & 1 & 0 & 1 & \vec{v_1} \\ \hline 1 & 1 & 0 & 1 & \\ \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & M_1 \\ \vdots & \vdots & \vdots & \vdots & \end{array}\right]$$

A_0 is the $(k-3)\times(k-3)$ submatrix in the last $k-3$ rows and last $k-3$ columns of A'' and $\vec{v}$ is equal to vector 10^* in the last $k-3$ columns of the first row of A''. Now since the only column of A'' in which $\vec{v}$ contains a one is column 4, the only $\mathrm{ADD}_{r,c}$ operations involving the entries of A_0 in A'' that will affect $\vec{v}$ are $\mathrm{ADD}_{r,c}$ operations that involve column 4 of A'' (which corresponds to the first column of A_0). So let us determine what effect an $\mathrm{ADD}_{r,c}(i,4)$ operation has on A'', where $4\le i\le k$. Without loss of generality, let $i=4$. If we perform an $\mathrm{ADD}_{r,c}(4,4)$ on A'' (the boldface entry), and then swap the

first two columns, we will produce the following matrix:

$$A' = \left[\begin{array}{cc|c|c|c} 1 & 0 & 1 & 1 & \vec{v_1} \\ 0 & 1 & 1 & 1 & \vec{v_1} \\ \hline 1 & 1 & 1 & 0 & 0^* \\ \hline 1 & 1 & 0 & 1 & \vec{v_1} \\ \hline 0 & 0 & 0 & 1 & \\ \vdots & \vdots & \vdots & \vdots & M_1' \\ 0 & 0 & 0 & 0 & \\ \vdots & \vdots & \vdots & \vdots & \end{array}\right]$$

Let A_0' be the $(k-3) \times (k-3)$ submatrix in the last $k-3$ rows and last $k-3$ columns of A' and let $\vec{v'} = 1\vec{v_1}$. Then A' is equal to the extension matrix $\mathcal{A}_2(A_0', \vec{v'})$. Notice that $\vec{v'}$ is a row of A_0'. Therefore, if we perform any $\mathrm{ADD}_{r,c}$ operation on an entry of A_0' in A', we will produce a matrix $A''' = \mathcal{A}_2(A_0''', \vec{v'''})$ in which either $\vec{v'''}$ is a row of A_0''' (which has the same structure as A') or $w(\vec{v'''}) = 1$ (which has the same structure as A'').

What we have just shown is that for any $(2k, k, 4)$ e_3-code, there exists an equivalent code C generated by $G = [I_k | \mathcal{A}_2(A_0, \vec{v})]$ where $G_0 = [I_{k-3} | A_0]$ generates a $(2k-6, k-3)$ self-dual code C_0 and $\vec{v}$ is either a row of A_0 or $w(\vec{v}) = 1$. Furthermore, if $G_0' = [I_{k-3} | A_0']$ generates a code equivalent to C_0', then there exists a vector $\vec{v'}$, where either $w(\vec{v'}) = 1$ or $\vec{v'}$ is a row of A_0', such that $G' = [I_k | \mathcal{A}_2(A_0', \vec{v'})]$ generates a code equivalent to C.

Thus, given any list L_0 of $(k-3)\times(k-3)$ matrices A_0, where $G_0 = [I_{k-3} | A_0]$ generates a $(2k-6, k-3)$ self-dual code, if for each A_0 in L_0 and each $(k-3)$-vector $\vec{v}$, where either $w(\vec{v}) = 1$ or $\vec{v}$ is a row in A_0, if we produce a list L of $k \times k$ matrices $\mathcal{A}_2(A_0, \vec{v})$, then for any $(2k, k, 4)$ e_3-code C, there will exist an A in L such that $G = [I_k | A]$ generates a code equivalent to C. This gives us our algorithm for enumerating the $(2k, k, 4)$ e_3-codes.

One last thing we need to consider, before giving a formal description of our algorithm for enumerating the e_3-codes, is the distance of the $(2k-6, k-3)$ self-dual codes C_0 that we use to generate L. Let C_0 be a $(2k-6, k-3)$ self-dual code generated by $G_0 = [I_{k-3} | A_0]$. Let $\vec{v}$ be any $(k-3)$-vector such that either $w(\vec{v}) = 1$ or $\vec{v}$ is a row of A_0. Our algorithm

for enumerating the $(2k, k, 4)$ e_3-codes works by finding all $k \times k$ matrices A, where:

$$A = \left[\begin{array}{cc||c|c} 1 & 0 & 1 & \vec{v} \\ 0 & 1 & 1 & \vec{v} \\ \hline\hline 1 & 1 & 1 & 0^* \\ \hline 1 & 1 & 0 & \\ \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & A_0 \\ \vdots & \vdots & \vdots & \end{array}\right]$$

We will show that for $G = [I_k|A]$ to be a generator matrix for a $(2k, k, 4)$ e_3-code, the $(2k-6, k-3)$ self-dual code code C_0, generated by $G_0 = [I_{k-3}|A_0]$ must have distance ≥ 4. Suppose C_0 has distance 2. Then A_0 has a row r with weight 1 and since the rows of A have weight ≥ 3, this row must intersect $\vec{v}$ (otherwise, the first two columns corresponding to that row in A would be 0 0). Without loss of generality, assume row r is one in the first row of A_0 and row r is one in the first column of A_0. Then the first four rows of A are as follows:

$$\begin{array}{cc||c|cc} 1 & 0 & 1 & 1 & \vec{v} \\ 0 & 1 & 1 & 1 & \vec{v} \\ \hline\hline 1 & 1 & 1 & 0 & 0^* \\ \hline 1 & 1 & 0 & 1 & 0^* \end{array}$$

Suppose $w(\vec{v}) = 1$. Then $\vec{v} = 0^*$, which implies that the first four rows of A form an $e_{4,0}$-block, and therefore C is not an e_3-code. Therefore, $w(\vec{v}) \geq 2$ which implies that $\vec{v}$ must be a row in A_0 (other than row r). But this implies that the weight 1 row in A_0 intersects $\vec{v}$, which is a contradiction, since the rows of A_0 intersect in an even number of positions. Therefore, C_0 cannot have distance 2. Thus, C_0 has distance ≥ 4.

We are now ready to give our algorithm for enumerating the $(2k, k, 4)$ e_3-codes.

Algorithm 4.28 :

- Input: a list L_0 of $(k-3) \times (k-3)$ matrices A_0, where $G_0 = [I_{k-3}|A_0]$ generates a $(2k-6, k-3, d \geq 4)$ self-dual code, such that for any $(2k-6, k-3, d \geq 4)$ self-dual code C_0 there exists one and only one matrix A_0 in L_0 such that $G_0 = [I_{k-3}|A_0]$ generates a code equivalent to C_0.

- Output: a list L of $k \times k$ matrices A, where $G = [I_k|A]$ generates a $(2k, k, 4)$ e_3-code, such that for any $(2k, k, 4)$ e_3-code C there exists at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C.

begin
  for each $A_0 \in L_0$ **do**
    for each $(k-3)$–vector $\vec{v}$ such that $w(\vec{v}) = 1$ or $\vec{v}$ is a row of A_0 **do**
      Let A=$\mathcal{A}_2(A_0, \vec{v})$.
      if A does not contain a block with higher precedence than a e_3-block **then**
        QuickOrderMatrix$(A, 0, A')$.
        if $A' \notin L$ **then**
          Insert A' into L.
      end if
    end for
  end for
end

We now have our algorithm for enumerating the $(2k, k, 4)$ e_3-codes. We have now found algorithms for enumerating every class of $(2k, k, 4)$ self-dual codes.

Summarizing, in this section we have developed algorithms for enumerating each class of $(2k, k, 4)$ self-dual codes: the e_{2i}-codes, the d_i-codes, and the e_3-codes. We have found that:

- the e_{2i}-codes can be produced from the $(2(k-2i), k-2i, d \geq 4)$ self-dual codes,
- the d_i-codes can be produced from the $(2(k-(i+1)), k-(i+1), d \geq 4)$ self-dual codes, and
- the e_3-codes can be produced from the $(2k-6, k-3, d \geq 4)$ self-dual codes.

Our algorithm for enumerating the e_{2i}-codes produces a list L of $k \times k$ matrices such that for any $(2k, k, 4)$ e_{2i}-code C, there exists one and only one A in L such that $G = [I_k|A]$ generates a code equivalent to C. Our algorithms for enumerating the d_i-codes and the e_3-codes each produce a list L of $k \times k$ matrices such that for any $(2k, k, 4)$ d_i-code/e_3-code C, there exists at least one A in L such that $G = [I_k|A]$ generates a code equivalent to C.

These algorithms are an improvement on the algorithm given in Section 4.1, in which we enumerated the $(2k, k, 4)$ self-dual codes by extending the $(2k-4, k-2, 4)$ self-dual codes. The improvements these revised algorithms give us are a reduction in both the length and the number of input codes, and a reduction in both the length and the number of extension vectors. These improvements result in both a reduction in the running time and a reduction in the number of equivalent codes produced.

In the next section, we will look at ways in which we can further reduce the running time and the number of equivalent codes produced by the algorithms for enumerating the d_i-codes and the e_3-codes, by considering the generator blocks for weight 4 sets in the input codes and how they combine with the extension vectors.

4.5 Further Improvements

Our algorithms for enumerating the $(2k, k, 4)$ e_3-codes and the $(2k, k, 4)$ d_i-codes, $i \geq 1$, are both based on the fact that for any such code there exists an equivalent code C,

generated by $G = [I_k|A]$, where:

$$A = \mathcal{A}_i(A_0, \vec{v}) = \left[\begin{array}{cc||ccc|c} 1 & 0 & 1 & 1 & \ldots\ 1 & \vec{v} \\ 0 & 1 & 1 & 1 & \ldots\ 1 & \vec{v} \\ \hline\hline 1 & 1 & & & & \\ 1 & 1 & & & & \\ \vdots & \vdots & & I_{i-1} & & 0 \\ 1 & 1 & & & & \\ \hline & & & & & \\ F & & & 0 & & A_0 \\ & & & & & \end{array}\right]$$

where A_0 is a $(k-i-1) \times (k-i-1)$ matrix, $\vec{v}$ is a $(k-i-1)$–vector, and F is a $(k-i-1) \times 2$ matrix. $G_0 = [I_{k-i-1}|A_0]$ generates a $(2(k-i-1), k-i-1, d \geq 4)$ self-dual code. The vector $\vec{v}$ has weight $w \equiv i+1 \bmod 2$. The rows of the matrix F are determined by A_0 and $\vec{v}$. For the d_i-codes, $i \geq 3$ and $w(\vec{v}) \geq 1$, for the d_2-codes, $w(\vec{v}) \geq 3$, and for the d_1-codes, $w(\vec{v}) \geq 4$. For the e_3-codes, $A = \mathcal{A}_2(A_0, \vec{v})$, where $\vec{v}$ is either a row of A_0 or $w(\vec{v}) = 1$.

Let L_0 be any list of $(k-i-1) \times (k-i-1)$ matrices A_0, such that for any $(2(k-i-1), k-i-1, d \geq 4)$ self-dual code C_0 there exists one and only $A_0 \in L_0$ such that $G_0 = [I_{k-i-1}|A_0]$ generates a code equivalent to C_0. Our algorithms for enumerating the $(2k, k, 4)$ e_3-codes and d_i-codes basically work by producing all $k \times k$ matrices $A = \mathcal{A}_i(A_0, \vec{v})$ for all $A_0 \in L_0$ and all extension vectors $\vec{v}$ whose weight meets the specified parity and lower bounds. Two problems our algorithms suffer from is that they may produce many pairs of matrices A and A' in which $G = [I_k|A]$ and $G' = [I_k|A']$ generate equivalent codes, and they may produce a matrix A in which $G = [I_k|A]$ does not generate a code in the specified class (for example, if we are enumerating the d_3-codes, the algorithm may produce an A in which $G = [I_k|A]$ is not a d_3-code).

In this section, we will look at ways in which we can reduce both of these problems by examining how the generator blocks for weight 4 sets in the input matrices $G_0 = [I_{k-i-1}|A_0]$ interact with the extension vector. The results of this examination will show

us, in certain cases, how we can determine beforehand, for a given A_0:

- when two extension vectors $\vec{v}$ and $\vec{v'}$ will result in matrices $A = \mathcal{A}_i(A_0, \vec{v})$ and $A' = \mathcal{A}_i(A_0, \vec{v'})$ in which $G = [I_k|A]$ and $G' = [I_k|A']$ generate equivalent codes, and
- what extension vectors $\vec{v}$ will result in a matrix $A = \mathcal{A}_i(A_0, \vec{v})$ in which $G = [I_k|A]$ does not generate a code in the specified class.

These results can then be used to reduce the unnecessary work done by our algorithms for enumerating the e_3-codes and d_i-codes.

We will begin by giving examples of the two problems from which our algorithms suffer.

Example 4.29 Let C_0 be the $(8, 4, 4)$ self-dual code generated by $G_0 = [I_4|A_0]$, where:

$$A_0 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

The rows of G_0 form an $e_{4,0}$-block.

Suppose we our enumerating the $(16, 8, 4)$ d_3-codes. Then A_0 may be one of the input matrices which our algorithm will extend. Two of the extension vectors our algorithm will use to extend A_0 are $\vec{v} = 1100$ and $\vec{v'} = 1010$. The matrices $A = \mathcal{A}_3(A_0, \vec{v})$ and

$A' = \mathcal{A}_3(A_0, \vec{v'})$ that result from these extensions are:

$$A = \left[\begin{array}{cc|cc|cccc} 1&0&1&1&1&1&0&0\\ 0&1&1&1&1&1&0&0\\ \hline 1&1&1&0&0&0&0&0\\ 1&1&0&1&0&0&0&0\\ \hline 0&0&0&0&1&1&1&0\\ 0&0&0&0&1&1&0&1\\ 1&1&0&0&1&0&1&1\\ 1&1&0&0&0&1&1&1 \end{array}\right] \quad A' = \left[\begin{array}{cc|cc|cccc} 1&0&1&1&1&0&1&0\\ 0&1&1&1&1&0&1&0\\ \hline 1&1&1&0&0&0&0&0\\ 1&1&0&1&0&0&0&0\\ \hline 0&0&0&0&1&1&1&0\\ 1&1&0&0&1&1&0&1\\ 0&0&0&0&1&0&1&1\\ 1&1&0&0&0&1&1&1 \end{array}\right]$$

Notice that if we swap the second and third last rows and the second and third last columns of A', A is produced. Therefore, $G = [I_k|A]$ and $G' = [I_k|A']$ generate equivalent codes. Thus, our algorithm only needed to try one of the two vectors $\vec{v}$ and $\vec{v'}$. As we shall see, this is something we could have determined beforehand, based on the fact that A_0 contains an $e_{4,0}$-block.

Example 4.30 Let C_0 be the $(8,4,4)$ self-dual code generated by $G_0 = [I_4|A_0]$, where:

$$A_0 = \begin{bmatrix} 1&1&1&0\\ 1&1&0&1\\ 1&0&1&1\\ 0&1&1&1 \end{bmatrix}$$

The rows of G_0 form an $e_{4,0}$-block.

Suppose we our enumerating the $(12,6,4)$ d_1-codes. Then A_0 may be one of the input matrices our algorithm will extend. $\vec{v} = 1111$ is a vector our algorithm will use to extend

A_0. The matrix $A = \mathcal{A}_1(A_0, \vec{v})$ that results from this extension is:

$$A = \left[\begin{array}{cc|cccc} 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

This A contains an e_6-block, and thus will be rejected. As we shall see, this too could have been determined beforehand, based on the fact that A_0 contained an $e_{4,0}$-block.

The improvements we will make are based on the concept of the an extension of a generator block for a weight 4 set in the input code, which we will now define. Let C_0 be any $(2(k-i-1), k-i-1, 4)$ self-dual code generated by a matrix of the form $G_0 = [I_{k-i-1}|A_0]$. Recall that the rows of G_0 can be partitioned into the $m+1$ disjoint sets $G_{B_1}, G_{B_2}, \ldots, G_{B_m}, G_R$ where each G_{B_i} is a generator block for a weight 4 set B_i and G_R is the set of remaining rows of G_0 (if any). Further recall that the columns of G_0 may also be partitioned into the $m+1$ disjoint sets $S_{B_1}, S_{B_2}, \ldots, S_{B_m}, S_R$ where each S_{B_i} are the columns of the weight 4 set B_i and S_R is the set of remaining columns of G_0 (if any). Let $\vec{v}$ be an extension vector. We will denote the portion of $\vec{v}$ that corresponds to the subset of columns S_{B_i} that are columns in A_0 by $v\vec{}_{B_i}$. We shall refer to $v\vec{}_{B_i}$ as an *extension for the generator block* G_{B_i}.

Example 4.31 Let C_0 be the $(20, 10, 4)$ self-dual code generated by $G_0 = [I|A_0]$ where:

$$A_0 = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} \end{matrix} \\ \left[\begin{array}{cccccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{matrix} \end{array}$$

C_0 contains two weight 4 sets B_0 and B_1. B_0 is generated by the $d_{5,1}$-block $G_{B_0} = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ and B_1 is generated by the $d_{3,1}$-block $G_{B_1} = \{r_7, r_8, r_9, r_{10}\}$. The columns of G_{B_0} are $S_{B_0} = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ and the columns of G_{B_1} are $S_{B_1} = \{c_7, c_8, c_9, c_{10}\}$. Let $\vec{v} = (v_1, v_2, v_3, \ldots, v_{10})$ be any extension vector. Then the extension of G_{B_0} is $v\vec{}_{B_0} = (v_1, v_2, v_3, v_4, v_5, v_6)$ and the extension of G_{B_1} is $v\vec{}_{B_1} = (v_7, v_8, v_9, v_{10})$.

For each of the different generator blocks G_{B_i} that may occur in G_0, we will have a look at how all the different extensions $v\vec{}_{B_i}$ intersects the generator block G_{B_i}. From this, we will be able to determine, for any two extension vectors $\vec{v}$ and $\vec{v'}$ that only differ in $v\vec{}_{B_i}$ and $v'\vec{}_{B_i}$, whether or not $G = [I_k|\mathcal{A}_i(A_0, \vec{v})]$ and $G' = [I_k|\mathcal{A}_i(A_0, \vec{v'})]$ will always generate equivalent codes (no matter how the remaining columns of $\vec{v}$ and $\vec{v'}$ are equally filled). If $v\vec{}_{B_i}$ and $v'\vec{}_{B_i}$ always result in generator matrices for equivalent codes, then they will be referred to as *equivalent extensions* for the generator block G_{B_i}. We will also be able to determine the lowest precedence generator block that $G = [I_k|\mathcal{A}_i(A_0, \vec{v})]$ must contain, given any $\vec{v}$ containing a fixed extension $v\vec{}_{B_i}$ for a given generator block G_{B_i} in G_0.

4.5.1 Equivalent Extensions of Generator Blocks

In this subsection, we will see how we can find all inequivalent extensions for a given generator block G_B for a weight 4 set B. We will show that two extension vectors $\vec{v_B}$ and $\vec{v'_B}$ for the generator block G_B are equivalent by showing that for any $(2(k-i-1), k-i-1, 4)$ self-dual code C_0, generated by $G_0 = [I_{k-i-1}|A_0]$, where G_0 contains a G_B-block, that for any two extension vectors $\vec{v}$ and $\vec{v'}$, where $\vec{v}$ and $\vec{v'}$ only differ in $\vec{v_B}$ and $\vec{v'_B}$, there exists a sequence of $\text{ADD}_{r,c}$, SWAP_r, and/or SWAP_c operations on the matrix A_0 in $\mathcal{A}_i(A_0, \vec{v})$ that results in the matrix $\mathcal{A}_i(A_0, \vec{v'})$. This tells us that $G = [I_k|\mathcal{A}_i(A_0, \vec{v})]$ and $G' = [I_k, \mathcal{A}_i(A_0, \vec{v'}])$ generate equivalent codes, which in turn implies that $\vec{v_B}$ and $\vec{v'_B}$ are equivalent extensions for the generator block G_B. We will demonstrate how this is done by finding all the equivalent extensions for a $d_{2,0}$-block.

Let C_0 be a $(2(k-i-1), k-i-1, 4)$ self-dual code, generated by $G_0 = [I_{k-i-1}|A_0]$, where G_0 contains a generator block G_B that is a $d_{2,0}$-block:

$$A_0 = \left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ \hline 1 & 1 & 0 & 0 & X \\ 1 & 0 & 1 & 1 & Y \\ 0 & 1 & 1 & 1 & Z \\ 0 & 0 & 0 & 0 & W \end{array}\right]$$

Let $\vec{v}$ be any extension vector. Let $\vec{v_B} = (v_1, v_2, v_3, v_4)$ be the columns of $\vec{v}$ that correspond to the columns of the $d_{2,0}$-block and let $\vec{v_R}$ be the remaining columns of $\vec{v}$. Let C be the $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$, where $A = \mathcal{A}_i(A_0, \vec{v})$. To simplify our

demonstration, let us assume $i = 1$. Then:

$$A = \left[\begin{array}{cc|cccc|c} 1 & 0 & v_1 & v_2 & v_3 & v_4 & \vec{vR} \\ 0 & 1 & v_1 & v_2 & v_3 & v_4 & \vec{vR} \\ \hline x & x & 1 & 1 & 1 & 0 & 0^* \\ y & y & 1 & 1 & 0 & 1 & 0^* \\ \hline & & 1 & 1 & 0 & 0 & X \\ & F & 1 & 0 & 1 & 1 & Y \\ & & 0 & 1 & 1 & 1 & Z \\ & & 0 & 0 & 0 & 0 & W \end{array}\right]$$

where the first two columns of the last $k-2$ two rows of A are determined by how the last $k-2$ columns of each row intersect with $\vec{v}$. There are $2^4 = 16$ different possibilities for $\vec{vB} = (v_1, v_2, v_3, v_4)$. We must find all the different pairs $\vec{vB}$ and $\vec{v'_B}$ for which there exists a sequence of $\mathrm{ADD}_{r,c}$, SWAP_r, and/or SWAP_c operations on $\mathcal{A}(A_0, \vec{vB}\vec{vR})$ that result in the matrix $\mathcal{A}(A_0, \vec{v'_B}\vec{vR})$.

First note that if we swap columns 5 and 6 of A, and then swap rows 3 and 4 of A, then A_0 and $\vec{vR}$ remain unchanged, and (v_1, v_2, v_3, v_4) is permuted to (v_1, v_2, v_4, v_3). Therefore, $\vec{vB} = (v_1, v_2, v_3, v_4)$ and $\vec{v'_B} = (v_1, v_2, v_4, v_3)$ are equivalent extensions for a $d_{2,0}$-block.

Suppose $\vec{vB} = (1, 1, 1, 1)$:

$$A = \left[\begin{array}{cc|cccc|c} 1 & 0 & 1 & 1 & 1 & 1 & \vec{vR} \\ 0 & 1 & 1 & 1 & 1 & 1 & \vec{vR} \\ \hline 1 & 1 & 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 1 & 1 & 0 & 1 & 0^* \\ \hline & & 1 & 1 & 0 & 0 & X \\ & F & 1 & 0 & 1 & 1 & Y \\ & & 0 & 1 & 1 & 1 & Z \\ & & 0 & 0 & 0 & 0 & W \end{array}\right]$$

Then if we perform an $\mathrm{ADD}_{r,c}(3,3)$ on A, producing A', and then perform a $\mathrm{SWAP}_c(4,5)$ on A', producing A'', we get:

$$A' = \left[\begin{array}{cc|cccc|c} 1 & 0 & 1 & 0 & 0 & 1 & \vec{v_R} \\ 0 & 1 & 1 & 0 & 0 & 1 & \vec{v_R} \\ \hline 1 & 1 & 1 & 1 & 1 & 0 & 0^* \\ 0 & 0 & 1 & 0 & 1 & 1 & 0^* \\ \hline & & 1 & 0 & 1 & 0 & X \\ & F' & 1 & 1 & 0 & 1 & Y \\ & & 0 & 1 & 1 & 1 & Z \\ & & 0 & 0 & 0 & 0 & W \end{array}\right] \qquad A'' = \left[\begin{array}{cc|cccc|c} 1 & 0 & 1 & 0 & 0 & 1 & \vec{v_R} \\ 0 & 1 & 1 & 0 & 0 & 1 & \vec{v_R} \\ \hline 1 & 1 & 1 & 1 & 1 & 0 & 0^* \\ 0 & 0 & 1 & 1 & 0 & 1 & 0^* \\ \hline & & 1 & 1 & 0 & 0 & X \\ & F' & 1 & 0 & 1 & 1 & Y \\ & & 0 & 1 & 1 & 1 & Z \\ & & 0 & 0 & 0 & 0 & W \end{array}\right]$$

Thus, the sequence of operations $\mathrm{ADD}_{r,c}(3,3)$, $\mathrm{SWAP}_c(4,5)$ leaves A_0 and $\vec{v_R}$ alone, but changes $\vec{v_B}$ from $(1,1,1,1)$ to $(1,0,0,1)$. Therefore, $(1,1,1,1)$ and $(1,0,0,1)$ are equivalent extensions for a $d_{2,0}$-block. Furthermore, since (v_1,v_2,v_3,v_4) and (v_1,v_2,v_4,v_3) are equivalent extensions, $(1,0,1,0)$ is equivalent to $(1,0,0,1)$, and thus is also equivalent to $(1,1,1,1)$. Therefore, $(1,1,1,1)$, $(1,0,0,1)$, and $(1,0,1,0)$ are all equivalent extensions for a $d_{2,0}$-block. Similarly, if $\vec{v_B} = (1,1,1,1)$, then the sequence of operations $\mathrm{ADD}_{r,c}(3,4)$, $\mathrm{SWAP}_c(3,5)$ leaves A_0 and $\vec{v_R}$ alone, but changes $\vec{v_B}$ from $(1,1,1,1)$ to $(0,1,0,1)$. Therefore, $(1,1,1,1)$, $(0,1,0,1)$, and $(0,1,1,0)$ are all equivalent extensions for a $d_{2,0}$-block. Finally, if $\vec{v_B} = (1,1,1,1)$, then the sequence of operations $\mathrm{ADD}_{r,c}(3,5)$, $\mathrm{SWAP}_c(3,4)$ leaves A_0 and $\vec{v_R}$ alone, but changes $\vec{v_B}$ from $(1,1,1,1)$ to $(0,0,1,1)$. Therefore, $(1,1,1,1)$ and $(0,0,1,1)$ are equivalent extensions of a $d_{2,0}$-block. Thus, the six extensions $(1,1,1,1)$, $(1,0,0,1)$, $(1,0,1,0)$, $(0,1,0,1)$, $(0,1,1,0)$, and $(0,0,1,1)$ are all equivalent extensions for a $d_{2,0}$-block.

In the same way we can show that the six extensions $(1,1,1,0)$, $(1,1,0,1)$, $(1,0,0,0)$, $(0,1,0,0)$, $(0,0,1,0)$, and $(0,0,0,1)$ are all equivalent extensions for a $d_{2,0}$-block. Unfortunately, there are no equivalent extensions among the remaining possibilities: $(1,1,0,0)$, $(1,0,1,1)$, $(0,1,1,1)$, and $(0,0,0,0)$.

What we have shown is that for any $(2(k-i-1), k-i-1, 4)$ self-dual code C_0, generated by $G_0 = [I_{k-i-1}|A_0]$, where G_0 contains a $d_{2,0}$-block, and any extension vector $\vec{v} = \vec{v_B}\vec{v_R}$, of the 16 possibilities for $\vec{v_B}$, at most 6 will result in a matrix $G = [I_k|\mathcal{A}_i(A_0, \vec{v})]$ that generates an inequivalent $(2k, k, 4)$ self-dual code. Therefore, our algorithms need only try 6 of the 16 extensions $\vec{v_B}$ for each $d_{2,0}$-block in G_0.

Using the same method we used for the $d_{2,0}$-block, we can find all the equivalent extensions for each of the different generator blocks for weight 4 sets. A list of all inequivalent extensions for each of the different generator blocks is given at the end of this section.

4.5.2 Minimal Blocks for Extensions of Generator Blocks

In this subsection, we will see how we can determine the lowest precedence generator block that $G = [I_k|\mathcal{A}_i(A_0, \vec{v})]$ must contain, for a given generator block G_B in $G = [I_{k-i-1}|A_0]$ and an extension vector $\vec{v}$ that is fixed in the columns of $\vec{v_B}$. We shall call this generator block the *minimal block* for the extension $\vec{v_B}$ of the generator block G_B. We will demonstrate how this is done by finding the minimal block for all the inequivalent extensions of a $d_{2,0}$-block.

Let C_0 be a $(2(k-i-1), k-i-1, 4)$ self-dual code, generated by $G_0 = [I_{k-i-1}|A_0]$, where G_0 contains a generator block G_B that is a $d_{2,0}$-block. Let $\vec{v}$ be any extension vector. Let $\vec{v_B} = (v_1, v_2, v_3, v_4)$ be the columns of $\vec{v}$ that correspond to the columns of the $d_{2,0}$-block and let $\vec{v_R}$ be the remaining columns of $\vec{v}$. Let C be the $(2k, k, 4)$ self-dual code generated by $G = [I_k|A]$, where $A = \mathcal{A}_i(A_0, \vec{v})$. Then, with just the columns in A of the rows of the $d_{2,0}$-block and the extension vector $\vec{v}$ given, we have:

1	0	v_1	v_2	v_3	v_4	$\vec{v_R}$
0	1	v_1	v_2	v_3	v_4	$\vec{v_R}$
x	x	1	1	1	0	0^*
y	y	1	1	0	1	0^*

There are six inequivalent extensions $\vec{v_B}$ of a $d_{2,0}$-block. They are: $(1,1,1,1)$, $(1,1,1,0)$, $(1,1,0,0)$, $(1,0,1,1)$, $(0,1,1,1)$, and $(0,0,0,0)$. We want to find the minimal block for each of these extensions.

Suppose $\vec{v_B} = (1,1,1,1)$:

1	0	1	1	1	1	$\vec{v_R}$
0	1	1	1	1	1	$\vec{v_R}$
1	1	1	1	1	0	0^*
1	1	1	1	0	1	0^*

The two rows that form the $d_{2,0}$-block in $G = [I_k|A_0]$ now form a $d_{1,3}$-block in $G = [I_k|A]$. Therefore, the minimal block for the extension $(1,1,1,1)$ of a $d_{2,0}$-block is a d_1-block. Note that the d_1-block block may in fact be a part of a larger generator block in $G = [I_k|A]$. For example, if $\vec{v_R} = 0^*$ then the d_1-block is in fact a part of a $d_{3,2}$-block. However, whenever a $d_{2,0}$-block is extended with $(1,1,1,1)$, the resulting $G = [I_k|A]$ must contain a generator block that is at least a d_1-block.

Suppose $\vec{v_B} = (1,1,1,0)$:

1	0	1	1	1	0	$\vec{v_R}$
0	1	1	1	1	0	$\vec{v_R}$
1	1	1	1	1	0	0^*
0	0	1	1	0	1	0^*

The second row of the two rows that form the $d_{2,0}$-block in $G = [I_k|A_0]$ now forms a $d_{1,0}$-block in $G = [I_k|A]$. Therefore, the minimal block for the extension $(1,1,1,0)$ of a $d_{2,0}$-block is a d_1-block.

Suppose $\vec{v_B} = (1,1,0,0)$:

1	0	1	1	0	0	$\vec{v_R}$
0	1	1	1	0	0	$\vec{v_R}$
0	0	1	1	1	0	0^*
0	0	1	1	0	1	0^*

The two rows that form the $d_{2,0}$-block in $G = [I_k|A_0]$ still form a $d_{2,0}$-block in $G = [I_k|A]$. Therefore, the minimal block for the extension $(1,1,0,0)$ of a $d_{2,0}$-block is a d_2-block.

Suppose $\vec{v_B} = (1,0,1,1)$:

1	0	1	0	1	1	$\vec{v_R}$
0	1	1	0	1	1	$\vec{v_R}$
0	0	1	1	1	0	0*
0	0	1	1	0	1	0*

The two rows that form the $d_{2,0}$-block in $G = [I_k|A_0]$ still form a $d_{2,0}$-block in $G = [I_k|A]$. Therefore, the minimal block for the extension $(1,0,1,1)$ of a $d_{2,0}$-block is a d_2-block.

Suppose $\vec{v_B} = (0,1,1,1)$:

1	0	0	1	1	1	$\vec{v_R}$
0	1	0	1	1	1	$\vec{v_R}$
0	0	1	1	1	0	0*
0	0	1	1	0	1	0*

The two rows that form the $d_{2,0}$-block in $G = [I_k|A_0]$ still form a $d_{2,0}$-block in $G = [I_k|A]$. Therefore, the minimal block for the extension $(0,1,1,1)$ of a $d_{2,0}$-block is a d_2-block.

Suppose $\vec{v_B} = (0,0,0,0)$:

1	0	0	0	0	0	$\vec{v_R}$
0	1	0	0	0	0	$\vec{v_R}$
0	0	1	1	1	0	0*
0	0	1	1	0	1	0*

The two rows that form the $d_{2,0}$-block in $G = [I_k|A_0]$ still form a $d_{2,0}$-block in $G = [I_k|A]$. Therefore, the minimal block for the extension $(0,0,0,0)$ of a $d_{2,0}$-block is a d_2-block.

We have now found the minimal block for each of the 6 inequivalent extensions of a $d_{2,0}$-block. For the extensions $(1,1,1,1)$ and $(1,1,1,0)$, the minimal block is a d_1-block,

and for the extensions $(1,1,0,0)$, $(1,0,1,1)$, $(0,1,1,1)$, and $(0,0,0,0)$, the minimal block is a d_2-block.

As an example of the advantage this provides us, suppose we are enumerating the $(2k,k,4)$ d_1-codes. Then for any $(2k-4,k-2,4)$ self-dual code C_0, generated by $G_0 = [I_{k-2}|A_0]$, where G_0 contains a $d_{2,0}$-block, the only extensions of the $d_{2,0}$-block we need to try are $(1,1,1,1)$ and $(1,1,1,0)$ since the other 4 extensions of the $d_{2,0}$-block will result in a $G = [I_k|\mathcal{A}_1(A_0,\vec{v})]$ that contains a generator block that is at least a d_2-block, and thus G will not be a generator matrix for a d_1-code.

Using the same method we used for the $d_{2,0}$-block, we can find the minimal block for all the inequivalent extensions for each of the different generator blocks for weight 4 sets.

What now follows is a list of all the inequivalent extensions $v\vec{_B}$ for each of the different generator blocks for weight 4 sets. Listed with each extension $v\vec{_B}$ for a generator block G_B are the number of extensions of G_B that are equivalent to G_B. Also listed is the minimal block for each extension $v\vec{_B}$.

generator blocks	inequivalent extensions	number of equivalent extensions	minimal blocks
$e_{3,0}$	1111	7	d_2
	1110	7	d_2
	0111	1	e_3
	0000	1	e_3
$e_{3,1}$	111	4	d_2
	110	3	d_2
	000	1	e_3

Figure 4.1: the number of equivalent extensions and the minimal block for each inequivalent extension of an e_3-block.

generator blocks	inequivalent extensions	number of equivalent extensions	minimal blocks
$e_{4,0}$	1111	7	d_3
	1110	8	e_3
	0000	1	e_4
$e_{2i,0}$, $i \geq 3$	111^a0^{2i-a-2}, $1 \leq a \leq 2i-2$	$4\binom{2i-2}{a}$	$max(d_{a-1}, d_{2i-a-1})$
	110^{2i-2}	1	d_{2i-1}
	101^{2i-2}	2	d_{2i-2}
	000^{2i-2}	1	e_{2i}
$e_{2i,1}$, $i \geq 3$	1^a0^{2i-a}, $1 \leq a \leq 2i$	$\binom{2i}{a}$	$max(d_{a-1}, d_{2i-a-1})$
	0^{2i}	1	e_{2i}

Figure 4.2: the number of equivalent extensions and the minimal block for each inequivalent extension of an e_{2i}-block, $i \geq 2$.

generator blocks	inequivalent extensions	number of equivalent extensions	minimal blocks
$d_{1,0}$	111	4	–
	110	1	d_1
	101	1	d_1
	011	1	d_1
	000	1	d_1
$d_{1,3}$	11	1	d_1
	10	2	–
	00	1	d_1
$d_{1,4}$	1	1	–
	0	1	d_1

Figure 4.3: the number of equivalent extensions and the minimal block for each inequivalent extension of a d_1-block.

generator blocks	inequivalent extensions	number of equivalent extensions	minimal blocks
$d_{i,0},\ i \geq 2$	$111^a0^{i-a},\ \ 1 \leq a \leq i$	$4\binom{i}{a}$	$max(d_{a-1}, d_{i-a})$
	110^i	1	d_i
	101^i	1	d_i
	011^i	1	d_i
	000^i	1	d_i
$d_{i,1},\ i \geq 2$	$111^a0^{i-a-1},\ \ 1 \leq a \leq i-1$	$4\binom{i-1}{a}$	$max(d_{a-1}, d_{i-a})$
	110^{i-1}	1	d_i
	101^{i-1}	2	d_{i-1}
	000^{i-1}	1	d_i
$d_{i,2},\ i \geq 2$	$1^a0^{i-a},\ \ 0 \leq a \leq i+1$	$\binom{i+1}{a}$	$max(d_{a-1}, d_{i-a})$
$d_{i,3},\ i \geq 2$	$1^a0^{i-a},\ \ 0 \leq a \leq i+1$	$\binom{i+1}{a}$	$max(d_{a-1}, d_{i-a})$
$d_{i,4},\ i \geq 2$	$1^a0^{i-a},\ \ 0 \leq a \leq i$	$\binom{i}{a}$	$max(d_{a-1}, d_{i-a})$

Figure 4.4: the number of equivalent extensions and the minimal block for each inequivalent extension of a d_i-block, $i \geq 2$.

4.6 Results of Enumeration

Based on the ideas presented in this chapter, programs for enumerating the $(2k, k, 4)$ self-dual codes have been implemented. These programs have been run to enumerate lists of $(2k, k, 4)$ self-dual codes of length up to and including $2k = 32$. We will conclude this chapter by listing the results of this enumeration.

For each k, three different programs were used to enumerate the $(2k, k, 4)$ self-dual codes. Each of these programs enumerate a different class of $(2k, k, 4)$ self-dual codes. The first program was used to enumerate both the decomposable $(2k, k, d \geq 4)$ self-dual codes and the $(2k, k, 4)$ e_k-code, $k \equiv 0 \bmod 2$, $k \geq 4$. (for $k \leq 16$, all decomposable $(2k, k, d \geq 4)$ self-dual codes have distance 4). The second program was used to enumerate both the indecomposable $(2k, k, 4)$ e_3-codes and the indecomposable $(2k, k, 4)$ d_i-codes, $i \geq 2$. Finally, the third program was used to enumerate the indecomposable $(2k, k, 4)$ d_1-codes. The program to enumerate the d_1-codes uses a method discussed at the end of Section 5.1 of Chapter 5 to further reduce the number of equivalent codes produced. For each of the three enumerations, tables are given listing the results. The information given in each table is:

- k: the dimension of the self-dual codes enumerated.
- running time: the (user) time it took to enumerate each list. The programs were run on an IBM RS/6000.
- number of codes: the number of codes produced.
- average number of equivalent codes: the number of codes produced divided by the number of inequivalent codes for each class of codes. The number of inequivalent codes is given at the end of Chapter 6.

The programs were run for $k = 4, 6, 7, 8, \ldots, 16$ (there do not exist $(2k, k, 4)$ self-dual codes for $k = 1, 2, 3, 5$).

k	running time	number of codes produced	average number of equivalent codes
4	0.0 sec	1	1
6	0.0 sec	1	1
7	–	0	–
8	0.0 sec	2	1
9	–	0	–
10	0.0 sec	2	1
11	0.0 sec	1	1
12	0.1 sec	5	1
13	0.0 sec	3	1
14	0.1 sec	11	1
15	0.2 sec	13	1
16	0.6 sec	42	1

Figure 4.5: results of enumeration of the decomposable $(2k, k, 4)$ self-dual codes and the $(2k, k, 4)$ e_k-code, where $k \equiv 0 \bmod 2$.

k	running time	number of codes produced	average number of equivalent codes
4	–	0	–
6	–	0	–
7	0.0 sec	1	1
8	0.0 sec	1	1
9	0.1 sec	2	1
10	1.1 sec	5	1
11	1.8 sec	6	1
12	6.7 sec	19	1
13	37.5 sec	41	1.11
14	2.4 min	135	1.16
15	19.3 min	549	1.59
16	93.8 min	3812	2.33

Figure 4.6: results of the enumeration of the indecomposable $(2k, k, 4)$ e_3-codes and the indecomposable $(2k, k, 4)$ d_i-codes, $i \geq 2$.

k	running time	number of codes produced	average number of equivalent codes
4	–	0	–
6	–	0	–
7	–	0	–
8	–	0	–
9	–	0	–
10	–	0	–
11	0.1 sec	1	1
12	3.0 sec	4	1
13	22.3 sec	8	1.14
14	2.7 min	44	1.57
15	7.7 min	213	2.17
16	79.7 min	2690	3.35

Figure 4.7: results of the enumeration of the indecomposable $(2k, k, 4)$ d_1-codes.

Chapter 5

A Recursive Enumeration of the $(2k, k, d \geq 6)$ Self-Dual Codes

In this chapter we will develop an algorithm that produces a list L_0 of $(2k, k, d \geq 6)$ self-dual codes with the property that for any $(2k, k, d \geq 6)$ self-dual code C_0 there exists at least one code in the list that is equivalent to C_0. The algorithm is recursive and assumes a complete list L_1 of inequivalent $(2k, k, 4)$ self-dual codes C_1 is available, where C_1 does not contain any intersecting weight 4 words.

In order to develop our algorithm, we will first need to further explore the extension method of Chapter 4 which we used to enumerate the $(2k, k, 4)$ self-dual codes. Recall, from Section 4.1 of Chapter 4, that for any $(2k, k, 4)$ self-dual code C there exists an equivalent code generated by $G = [I_k | A(A_0, \vec{v})]$, where $G_0 = [I_{k-2} | A_0]$ generates a $(2k - 4, k - 2)$ self-dual code C_0. In other words, for any $(2k, k, 4)$ self-dual code C, there exists a $(2k - 4, k - 2)$ self-dual code C_0 that can be *extended* to a code that is equivalent to C. In Section 5.1, for a given $(2k, k, 4)$ self-dual code C, a method will be given which will find all the inequivalent $(2k - 4, k - 2)$ self-dual codes C_0 that can be extended to a code that is equivalent to C. The results of this section will be used to develop our algorithm for enumerating the $(2k, k, d \geq 6)$ self-dual codes from the $(2k, k, 4)$ self-dual codes that

do not contain any intersecting weight 4 words.

In Section 5.2, it will be shown that we can extend any $(2k, k, d \geq 6)$ self-dual code C_0 to a $(2k+4, k+2, 4)$ self-dual code C, in which there exist at most three inequivalent $(2k, k)$ self-dual codes that can be extended to a code equivalent to C. One is the original $(2k, k, d \geq 6)$ self-dual code C_0. Another will be a $(2k, k, 4)$ self-dual code that does not contain any intersecting weight 4 words. In Section 5.3, we will use this result to develop our algorithm for enumerating a list of $(2k, k, d \geq 6)$ self-dual codes from any complete list of inequivalent $(2k, k, 4)$ self-dual codes containing no intersecting weight 4 words. Since the list of $(2k, k, d \geq 6)$ self-dual codes which our algorithm produces may contain many equivalent codes, in Section 5.4 we will look at ways in which we can reduce the number of equivalent codes which our algorithm produces. Finally, in Section 5.5, we will give the results of our enumeration.

5.1 The Three Codes Under a Weight 4 Word In A Self-Dual Code

Recall, by Theorem 4.2 of Chapter 4, that for any $(2k, k, 4)$ self-dual code C and weight 4 word $\vec{w} \in C$, there exists an equivalent code C', generated by $G' = [I_k | \mathcal{A}(A'_0, \vec{v'})]$, where the first two rows of G' sum to the weight 4 word $\vec{w'}$ of C' that corresponds to the weight 4 word $\vec{w}$ of C (see Figure 5.1). The four columns of G' labeled by c_1, c_2, c_3, c_4 in Figure 5.1 correspond to the four columns of C' in which the weight 4 word $\vec{w'}$ has a value of one. The matrix $G'_0 = [I_{k-2} | A'_0]$ generates a $(2k-4, k-2)$ self-dual code C'_0. The $(k-2)$–vector $\vec{v'}$ is such that $w(\vec{v'}) \equiv 0 \bmod 2$ and $w(\vec{v'}) \geq 2$. The $(k-2) \times 2$ matrix F' is determined by A'_0 and $\vec{v'}$ as follows: If a row in A'_0 intersects $\vec{v'}$ in an even number of positions then the corresponding row in F is 0 0, and if a row in A'_0 intersects $\vec{v'}$ in an odd number of positions then the corresponding row in F is 1 1.

In this section, for a given C and $\vec{w}$, we will find generator matrices $G''_0 = [I_{k-2} | A''_0]$ for

$$G' = [I_k|\mathcal{A}(A'_0, \vec{v'})] = \begin{array}{c} \begin{array}{cc} c_1 & c_2 \end{array} \qquad\qquad\qquad\qquad\qquad\qquad \begin{array}{cc} c_3 & c_4 \end{array} \\ \left[\begin{array}{cc|ccccc||cc|c} 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \vec{v'} \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 1 & \vec{v'} \\ \hline 0 & 0 & & & & & & & \\ 0 & 0 & & & & & & & \\ \vdots & \vdots & & & I_{k-2} & & & F' & A'_0 \\ 0 & 0 & & & & & & & \end{array}\right] \end{array}$$

Figure 5.1: The generator matrix for C'.

all the inequivalent $(2k-4, k-2)$ self-dual codes C''_0 for which there exists an extension vector $\vec{v''}$ in which $G'' = [I_k|\mathcal{A}(A''_0, \vec{v''})]$ generates a code C'' that is equivalent to C and such that the first two rows of G'' sum to the weight 4 word $\vec{w''}$ of C'' that corresponds to the weight 4 word $\vec{w}$ of C. We will refer to each of the $(2k-4, k-2)$ self-dual codes C''_0 we find as *codes under the weight 4 word $\vec{w}$ of C*. Note that once we know how to find all the codes under a specific weight 4 word of C, we will know how to find a list containing all the inequivalent $(2k-4, k-2)$ self-dual codes that can be extended to a code that is equivalent to C. That is, if for each weight 4 word $\vec{w}$ in C, we find all the inequivalent codes under the weight 4 word $\vec{w}$ of C, then the list we produce will contain all the inequivalent $(2k-4, k-2)$ self-dual codes that can be extended to a code that is equivalent to C.

In order to find all the inequivalent codes under the weight 4 word $\vec{w}$ of C, let us begin by examining the relationship between the codewords of the $(2k, k, 4)$ self-dual code C', generated by the matrix G' in Figure 5.1, and the $(2k-4, k-2)$ self-dual code C'_0, generated by the matrix $G'_0 = [I_{k-2}|A'_0]$. Let c_1, c_2, c_3, c_4 denote the columns in C' in which the weight 4 word $\vec{w'}$ has a value of one. By examining G' in Figure 5.1, we can see that the codewords of C'_0 are exactly the codewords of C', with columns c_1, c_2, c_3, c_4 deleted, that contain 0 0 in columns c_1, c_2. Furthermore, for any code equivalent to C', the set of codewords (with the permuted columns c_1, c_2, c_3, c_4 deleted) that contain 0 0

in the permuted columns c_1, c_2 form a code that is equivalent to C_0'. Therefore, any code under the weight 4 word $\vec{w}$ of C must be equivalent to a code C_0'' consisting of the set of codewords of C' (with columns c_1, c_2, c_3, c_4 deleted) that contain 0 0 in columns c_a, c_b, where $c_a, c_b \in \{c_1, c_2, c_3, c_4\}$. We will refer to C_0'' as a *code under* c_a, c_b. Since there are $\binom{4}{2} = 6$ choices for the two columns c_a and c_b, there are at most 6 inequivalent codes under the weight 4 word $\vec{w}$ of C. However, since the rows of F' in G' contain either 0 0 or 1 1, if we swap the two columns in G' labeled by c_1, c_3 and then swap the two columns labeled by c_2, c_4 and then row-reduce to recover the identity, G' is reproduced. In other words, the permutation $(c_1\ c_3)(c_2\ c_4)$ is an automorphism of C'. Therefore, since any code under c_a, c_b is generated by a matrix with the same form as G', any code under c_a, c_b is equivalent to any code under c_c, c_d where $\{c_a, c_b, c_c, c_d\} = \{c_1, c_2, c_3, c_4\}$. Thus, there are in fact at most three inequivalent codes under the weight 4 word $\vec{w}$ of C. They are any code under c_1, c_2 (or c_3, c_4), any code under c_1, c_3 (or c_2, c_4), and any code under c_1, c_4 (or c_2, c_3).

Let us now find generator matrices for each of these cases. Given the $(2k, k, 4)$ self-dual code C' generated by the matrix $G' = [I_k | \mathcal{A}(A_0', \vec{v'})]$, we immediately have a generator matrix for a code under c_1, c_2. That is, the matrix $G' = [I_{k-2} | A_0']$ generates a $(2k-4, k-2)$ self-dual code C' that is a code under c_1, c_2. The vector $\vec{v'}$ is an extension vector that extends C_0' to C'. Let us now find generator matrices and extension vectors for codes under c_1, c_3 and c_1, c_4 in terms of the matrix A_0' and the extension vector $\vec{v'}$. This will be done by applying sequences of $\mathrm{ADD}_{r,c}$, SWAP_r, and SWAP_c operations to the matrix $\mathcal{A}(A_0', \vec{v'})$, producing a matrix $\mathcal{A}(A_0'', \vec{v''})$, in which either the two columns of G' labeled by c_1, c_3, or the two columns labeled by c_1, c_4, have both been permuted to the identity matrix in $G'' = [I_k | \mathcal{A}(A_0'', \vec{v''})]$. This will give us A_0'' and $\vec{v''}$ in terms of A_0' and $\vec{v'}$.

In order to do this, we will first have to reorganize the matrix $\mathcal{A}(A_0', \vec{v'})$. Let us have

a closer look at the structure of the matrix:

$$\mathcal{A}(A'_0, \vec{v'}) = \left[\begin{array}{cc||c} 1 & 0 & \vec{v'} \\ 0 & 1 & \vec{v'} \\ \hline\hline F' & & A'_0 \end{array}\right]$$

Recall that the code C' generated by $G' = [I_k|\mathcal{A}(A'_0, \vec{v'})]$ has distance 4. The $(k-2)$–vector $\vec{v'}$ is such that $w(\vec{v'}) \equiv 0 \bmod 2$ and $w(\vec{v'}) \geq 2$. The rows of the $(k-2) \times 2$ matrix F' are either 0 0 or 1 1 and are determined by A'_0 and $\vec{v'}$.

Since the code C' has distance 4, by Theorem 2.4 of Chapter 2, the columns of $\mathcal{A}(A'_0, \vec{v'})$ must all have weight ≥ 3. Therefore, the first two columns of $\mathcal{A}(A'_0, \vec{v'})$ must both have weight ≥ 3, and thus there must exist a row, say row r, in the last $k-2$ rows of $\mathcal{A}(A'_0, \vec{v'})$ in which row r is equal to 1 1 in F'. Since row r is equal to 1 1 in F', row r must intersect $\vec{v'}$ in an odd number of positions, and thus, since $\vec{v'}$ has even weight, there must exist a column, say column c, in the last $k-2$ columns of $\mathcal{A}(A'_0, \vec{v'})$ in which $\vec{v'}$ has a value of 1 in column c and row r has a value of 0 in column c.

Rewriting the matrix $\mathcal{A}(A'_0, \vec{v'})$ with row r permuted to the third row, column c permuted to the third column, and all the possibilities for the values of the first three columns in the last $k-3$ rows of $\mathcal{A}(A'_0, \vec{v'})$, we get the following:

$$\mathcal{A}(A'_0, \vec{v'}) = \begin{array}{c} \\ (c_1) \\ (c_2) \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cc} c_3 & c_4 \end{array} \\ \left[\begin{array}{cc||c|c} 1 & 0 & 1 & \vec{v} \\ 0 & 1 & 1 & \vec{v} \\ \hline\hline 1 & 1 & 0 & \vec{w} \\ \hline 1 & 1 & 1 & X_0 \\ 1 & 1 & 0 & X_1 \\ 0 & 0 & 1 & X_2 \\ 0 & 0 & 0 & X_3 \end{array}\right] \end{array} \begin{array}{l} \\ w(\vec{v}) \equiv 1 \bmod 2 \\ \vec{w} \bullet \vec{v} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_0} \in X_0,\ \vec{x_0} \bullet \vec{v} \equiv 0 \bmod 2 \\ \text{for any } \vec{x_1} \in X_1,\ \vec{x_1} \bullet \vec{v} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_2} \in X_2,\ \vec{x_2} \bullet \vec{v} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_3} \in X_3,\ \vec{x_3} \bullet \vec{v} \equiv 0 \bmod 2 \end{array}$$

We now have $\mathcal{A}(A'_0, \vec{v'})$ in a form that will allow us to find generator matrices and extension vectors for codes under c_1, c_3 and c_1, c_4.

In order to keep track of the four columns of $G' = [I_k|\mathcal{A}(A'_0, \vec{v'})]$ that correspond to the four columns c_1, c_2, c_3, c_4 of C' that contain the four ones of the weight 4 word $\vec{w'}$ of C', the first two rows and columns of $\mathcal{A}(A'_0, \vec{v'})$ have been labeled appropriately. That is, the first two columns of $\mathcal{A}(A'_0, \vec{v'})$ are labeled with c_3 and c_4, and the first two rows of $\mathcal{A}(A'_0, \vec{v'})$ are labeled with (c_1) and (c_2) which indicates that the corresponding columns in the identity matrix of $G' = [I_k|\mathcal{A}(A'_0, \vec{v'})]$ are labeled with c_1 and c_2.

We will now find generator matrices and extension vectors for codes under c_1, c_3 and c_1, c_4 by applying appropriate sequences of $\mathrm{ADD}_{r,c}$, SWAP_r, and SWAP_c operations to $\mathcal{A}(A'_0, \vec{v'})$. Recall that for a given A and non-zero entry (i, j) in A, an $\mathrm{ADD}_{r,c}(i, j)$ on A essentially swaps column i of I_k with column j of A in $G = [I_k|A]$ and then row-reduces. A $\mathrm{SWAP}_r(i_0, i_1)$ on A swaps columns i_0 and i_1 of I_k in $G = [I_k|A]$ and then row-reduces, and a $\mathrm{SWAP}_c(j_0, j_1)$ swaps columns j_0 and j_1 of A in $G = [I_k|A]$. Also recall the rows and columns of A are indexed by $1, 2, \ldots, k$.

We can find a generator matrix and extension vector for a code under c_1, c_4 by performing an $\mathrm{ADD}_{r,c}(2, 2)$ on $\mathcal{A}(A'_0, \mathcal{A}(v'))$, which produces the matrix:

$$\mathcal{A}(A''_0, \vec{v''}) = \begin{array}{c} \\ (c_1) \\ (c_4) \\ \\ \\ \\ \\ \end{array} \left[\begin{array}{cc||c|c} c_3 & c_2 & & \\ \hline 1 & 0 & 1 & \vec{v} \\ 0 & 1 & 1 & \vec{v} \\ \hline\hline 1 & 1 & 1 & \vec{v} \oplus \vec{w} \\ \hline 1 & 1 & 0 & \vec{v} \oplus X_0 \\ 1 & 1 & 1 & \vec{v} \oplus X_1 \\ 0 & 0 & 1 & X_2 \\ 0 & 0 & 0 & X_3 \end{array}\right]$$

The notation $\vec{v'} \oplus X_0$ means add componentwise (modulo 2) $\vec{v'}$ to each row in X_0. The matrix $G''_0 = [I_{k-2}|A''_0]$ generates a code C''_0 that is a code under c_1, c_4. The vector $\vec{v''}$

is an extension vector that extends C_0'' to the $(2k, k, 4)$ self-dual code C'' generated by $G'' = [I_k | \mathcal{A}(A_0'', \vec{v''})]$. C'' is equivalent to the original $(2k, k, 4)$ self-dual code C.

To find a generator matrix and extension vector for a code under c_1, c_3 we will need to perform a sequence of $\mathrm{ADD}_{r,c}$, SWAP_r, and SWAP_c operations on $\mathcal{A}(A_0', \vec{v'})$. If we first perform a $\mathrm{SWAP}_r(2,3)$ followed by a $\mathrm{SWAP}_c(1,3)$ on $\mathcal{A}(A_0', \vec{v'})$, we get the following matrix:

	c_4	c_3	
(c_1)	1	1	$\vec{v}$
	0	1	$\vec{w}$
(c_2)	1	0	$\vec{v}$
	1	1	X_0
	0	1	X_1
	1	0	X_2
	0	0	X_3

If we then perform an $\mathrm{ADD}_{r,c}(2,3)$ on this matrix, we get the following matrix:

	c_4		
(c_1)	1	1	$\vec{v} \oplus \vec{w}$
(c_3)	0	1	$\vec{w}$
(c_2)	1	0	$\vec{v}$
	1	1	$\vec{w} \oplus X_0$
	0	1	$\vec{w} \oplus X_1$
	1	0	X_2
	0	0	X_3

Finally, if we perform an $\mathrm{ADD}_{r,c}(3,1)$ on this matrix, the following matrix is produced:

$$\mathcal{A}(A_0''', \vec{v'''}) = \begin{array}{c} \\ (c_1) \\ (c_3) \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cc} c_2 & c_4 \end{array} \\ \left[\begin{array}{cc||c|c} 1 & 0 & 1 & \vec{w} \\ 0 & 1 & 1 & \vec{w} \\ \hline\hline \mathbf{1} & \mathbf{1} & 0 & \vec{v} \\ \hline 1 & 1 & 1 & \vec{v} \oplus \vec{w} \oplus X_0 \\ 0 & 0 & 1 & \vec{w} \oplus X_1 \\ 1 & 1 & 0 & \vec{v} \oplus X_2 \\ 0 & 0 & 0 & X_3 \end{array}\right] \end{array}$$

The matrix $G_0''' = [I_{k-2}|A_0''']$ generates a code C_0''' that is a code under c_1, c_3. The vector $\vec{v'''} = 1\ \vec{w}$ is an extension vector that extends C_0''' to the $(2k, k, 4)$ self-dual code C''' generated by $G''' = [I_k|\mathcal{A}(A_0''', \vec{v'''})]$. C''' is equivalent to the original $(2k, k, 4)$ self-dual code C.

Let us now summarize what we have just found. Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ self-dual code C, and let $\vec{w}$ be any weight 4 in C. Let $G' = [I_k|\mathcal{A}(A_0', \vec{v'})]$ be the generator matrix for a code C', equivalent to C, such that the weight 4 word $\vec{w'}$ is the sum of the first two rows of G', where $\vec{w'}$ is the weight 4 word in C' that corresponds to the weight 4 word $\vec{w}$ of C, the vector $\vec{v'} = 1\vec{v}$, and the matrix A_0' is as follows:

$$A_0' = \left[\begin{array}{c|c} 0 & \vec{w} \\ \hline 1 & X_0 \\ 0 & X_1 \\ 1 & X_2 \\ 0 & X_3 \end{array}\right]$$

Let C_0 be any code under the weight 4 word $\vec{w}$ of C. Then C_0 is equivalent to one of the three $(2k-4, k-2)$ self-dual codes C_0', C_0'', and C_0''', where C_0' is generated by $G_0' = [I_{k-2}|A_0']$, and C_0'', C_0''' are generated by $G_0'' = [I_{k-2}|A_0'']$, $G_0''' = [I_{k-2}|A_0''']$,

respectively, where:

$$A_0'' = \left[\begin{array}{c|c} 1 & \vec{v} \oplus \vec{w} \\ \hline 0 & \vec{v} \oplus X_0 \\ 1 & \vec{v} \oplus X_1 \\ 1 & X_2 \\ 0 & X_3 \end{array}\right] \qquad A_0''' = \left[\begin{array}{c|c} 0 & \vec{v} \\ \hline 1 & \vec{v} \oplus \vec{w} \oplus X_0 \\ 1 & \vec{w} \oplus X_1 \\ 0 & \vec{v} \oplus X_2 \\ 0 & X_3 \end{array}\right]$$

The vectors $\vec{v''} = 1\vec{v}$ and $\vec{v'''} = 1\vec{w}$ are extension vectors that extend the codes C_0'' and C_0''', respectively, to codes that are equivalent to C. Note that the three codes C_0', C_0'', and C_0''' are not necessarily inequivalent.

Example 5.1 Let C be the $(16, 8, 4)$ self-dual code generated by $G = [I_8|A]$, where

$$A = \left[\begin{array}{cccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \begin{array}{l} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \end{array}$$

Let $\vec{w} = r_4 \oplus r_5 \oplus r_6$ be our weight 4 word.

Let A' be the matrix that is obtained by applying an $\mathrm{ADD}_{r,c}(4, 5)$ to A, followed by

the sequence $\mathrm{SWAP}_r(1,5)$, $\mathrm{SWAP}_r(2,6)$, $\mathrm{SWAP}_c(1,5)$, $\mathrm{SWAP}_c(2,8)$, and $\mathrm{SWAP}_c(3,6)$:

$$A' = \left[\begin{array}{cc|c|cccc|l} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & r_4 \oplus r_5 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & r_6 \\ \hline\hline 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & r_3 \oplus r_4 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & r_4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & r_1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & r_2 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & r_4 \oplus r_7 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & r_4 \oplus r_8 \end{array}\right]$$

Then $G' = [I_8|A']$ generates a code C', equivalent to C, such that the first two rows of G' sum to $\vec{w'}$, the weight 4 word of C' that corresponds to the weight 4 word $\vec{w}$ of C. Furthermore, $A' = \mathcal{A}(A'_0, \vec{v'})$ where $G'_0 = [I_6|A'_0]$ generates a $(12,6)$ self-dual code C'_0.

Let C_0 be any code under the weight 4 word $\vec{w}$ of C. Then C_0 is equivalent to one of the three $(12,6)$ self-dual codes C'_0, C''_0, and C'''_0, where C'_0, C''_0, C'''_0 are generated by $G'_0 = [I_6|A'_0]$, $G''_0 = [I_6|A''_0]$, $G'''_0 = [I_6|A'''_0]$, respectively, where:

$$A'_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad A''_0 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad A'''_0 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

The vectors:

$$\begin{aligned} \vec{v'} &= (1\,0\,1\,0\,1\,1) \\ \vec{v''} &= (1\,0\,1\,0\,1\,1) \\ \vec{v'''} &= (1\,0\,0\,0\,1\,0) \end{aligned}$$

are extension vectors that extend the codes C_0', C_0'', C_0''', respectively, to codes that are equivalent to C. Note that in this case C_0'' and C_0''' are equivalent. Therefore, for our $\vec{w}$ and C there are only two inequivalent codes under the weight 4 word $\vec{w}$ of C.

In the remainder of this chapter, we will encounter situations in which we are given the matrix A_0' and the vector $\vec{v'}$ from which we will want to produce the matrices A_0'' and A_0''' and the vectors $\vec{v''}$ and $\vec{v'''}$. To make it clear when we are finding these matrices and vectors, and to make it clear just how these matrices and vectors are formed, we will now define four functions on A_0' and $\vec{v'}$ that produce each of A_0'', A_0''', $\vec{v''}$, and $\vec{v'''}$.

Definition 5.2 Let $G_0' = [I_{k-2}|A_0']$ be a generator matrix for a $(2k-4, k-2)$ self-dual code C_0' and let $\vec{v'}$ be a $(k-2)$–vector that extends C_0' to a $(2k, k, 4)$ self-dual code. Given A_0' and $\vec{v'}$, the function $\mathcal{R}(A_0', \vec{v'})$ is defined as the $(k-2)\times(k-2)$ matrix that is produced as follows:

- Each row $\vec{x}$ in A_0' in which $\vec{x} \bullet \vec{v'} \equiv 1 \bmod 2$ is replaced with $\vec{x} \oplus \vec{v'}$.
- Each row $\vec{x}$ in A_0' in which $\vec{x} \bullet \vec{v'} \equiv 0 \bmod 2$ is left unchanged.

Definition 5.3 Define $v\vec{\mathcal{R}}(A_0', \vec{v'}) = \vec{v'}$.

These two definitions give us A_0'' and $\vec{v''}$. That is, $A_0'' = \mathcal{R}(A_0', \vec{v'})$ and $\vec{v''} = v\vec{\mathcal{R}}(A_0', \vec{v'})$.

Definition 5.4 Let $G_0' = [I_{k-2}|A_0']$ be a generator matrix for a $(2k-4, k-2)$ self-dual code C_0' and let $\vec{v'}$ be a $(k-2)$–vector that extends C_0' to a $(2k, k, 4)$ self-dual code. Let $\vec{r}$ be a fixed row in A_0' such that $\vec{v'} \bullet \vec{r} \equiv 1 \bmod 2$. Let c be a fixed column in which $\vec{v'}$ is 1 in column c and $\vec{r}$ is 0 in column c. Let $\vec{e_c}$ be the $(k-2)$–vector that has a value of 1 in column c and a value of 0 in every other column. Given A_0', $\vec{v'}$, $\vec{r}$, and c, the function $\mathcal{S}_0(A_0', \vec{v'}, \vec{r}, c)$ is defined as the $(k-2)\times(k-2)$ matrix that is produced as follows:

- Row $\vec{r}$ is replaced with $\vec{v'} \oplus \vec{e_c}$.

- Each row $\vec{x}$ in A'_0 in which both $\vec{v'} \bullet \vec{x} \equiv 1 \bmod 2$ and $\vec{x}$ is 1 in column c is replaced with $\vec{v'} \oplus \vec{e_c} \oplus \vec{r} \oplus \vec{x}$.

- Each row $\vec{x}$ in A'_0 (excluding row $\vec{r}$) in which both $\vec{v'} \bullet \vec{x} \equiv 1 \bmod 2$ and $\vec{x}$ is 0 in column c is replaced with $\vec{e_c} \oplus \vec{r} \oplus \vec{x}$.

- Each row $\vec{x}$ in A'_0 in which both $\vec{v'} \bullet \vec{x} \equiv 0 \bmod 2$ and $\vec{x}$ is 1 in column c is replaced with $\vec{v'} \oplus \vec{x}$.

- Each row $\vec{x}$ in A'_0 in which both $\vec{v'} \bullet \vec{x} \equiv 0 \bmod 2$ and $\vec{x}$ is 0 in column c is left unchanged.

Note that there are many choices for $\vec{r}$ and c.

Definition 5.5 Define $v_{\vec{\mathcal{S}_0}}(A'_0, \vec{v'}, \vec{r}, c) = \vec{e_c} \oplus \vec{r}$ where $\vec{e_c}$ and $\vec{r}$ are defined as in Definition 5.4.

These two definitions give us A'''_0 and $\vec{v'''}$. That is, $A'''_0 = \mathcal{S}_0(A'_0, \vec{v'}, \vec{r}, c)$ and $\vec{v'''} = v_{\vec{\mathcal{S}_0}}(A'_0, \vec{v'}, \vec{r}, c)$, where $\vec{r}$ is any row in A'_0 such that $\vec{v'} \bullet \vec{r} \equiv 1 \bmod 2$ and c is any column in which $\vec{v'}$ has a value of 1 and $\vec{r}$ has a value of 0.

Definition 5.4 is dependent on having a row $\vec{r}$ and column c of A'_0 in which $\vec{v'}$ has a value of 1 in column c and $\vec{r}$ has a value of 0 in column c. Later in this chapter, we will encounter situations in which we are given a row $\vec{r}$ and column c of A'_0 in which row $\vec{r}$ has a value of 1 in column c. More specifically, we will encounter situations in which we

are given a $k \times k$ matrix $\mathcal{A}(A_0', \vec{v'})$ and a non-zero entry of A_0' as follows:

$$\mathcal{A}(A_0', \vec{v'}) = \begin{array}{c} \\ (c_1) \\ (c_2) \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cc|c|c} c_3 & c_4 & & \\ \hline 1 & 0 & 1 & \vec{v} \\ 0 & 1 & 1 & \vec{v} \\ \hline 1 & 1 & 1 & \vec{w} \\ \hline 1 & 1 & 1 & X_0 \\ 1 & 1 & 0 & X_1 \\ 0 & 0 & 1 & X_2 \\ 0 & 0 & 0 & X_3 \end{array} \end{array} \begin{array}{l} \\ \\ w(\vec{v}) \equiv 1 \bmod 2 \\ \vec{w} \bullet \vec{v} \equiv 0 \bmod 2 \\ \text{for any } \vec{x_0} \in X_0,\ \vec{x_0} \bullet \vec{v} \equiv 0 \bmod 2 \\ \text{for any } \vec{x_1} \in X_1,\ \vec{x_1} \bullet \vec{v} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_2} \in X_2,\ \vec{x_2} \bullet \vec{v} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_3} \in X_3,\ \vec{x_3} \bullet \vec{v} \equiv 0 \bmod 2 \end{array}$$

If this is the case, then generator matrices and extension vectors for codes under c_1, c_4 and c_1, c_3, respectively, are given by the following matrices:

$\mathcal{A}(A_0'', \vec{v''}) =$

	c_3	c_2		
(c_1)	1	0	1	$\vec{v}$
(c_4)	0	1	1	$\vec{v}$
	1	1	0	$\vec{v} \oplus \vec{w}$
	1	1	0	$\vec{v} \oplus X_0$
	1	1	1	$\vec{v} \oplus X_1$
	0	0	1	X_2
	0	0	0	X_3

$\mathcal{A}(A_0''', \vec{v'''}) =$

	c_4	c_2		
(c_1)	1	0	1	$\vec{v} \oplus \vec{w}$
(c_3)	0	1	1	$\vec{v} \oplus \vec{w}$
	1	1	0	$\vec{v}$
	0	0	1	$\vec{w} \oplus X_0$
	1	1	1	$\vec{v} \oplus \vec{w} \oplus X_1$
	1	1	0	$\vec{v} \oplus X_2$
	0	0	0	X_3

In this case, our definitions still hold for A_0'' and $\vec{v''}$. That is, $A_0'' = \mathcal{R}(A_0', \vec{v'})$ and $\vec{v''} = \vec{v_{\mathcal{R}}}(A_0', \vec{v'})$. However, we will need to define new functions for A_0''' and $\vec{v'''}$.

Definition 5.6 Let $G_0' = [I_{k-2}|A_0']$ be a generator matrix for a $(2k-4, k-2)$ self-dual code C_0' and let $\vec{v'}$ be a $(k-2)$–vector that extends C_0' to a $(2k, k, 4)$ self-dual code. Let $\vec{r}$ be a fixed row in A_0' such that $\vec{v'} \bullet \vec{r} \equiv 1 \bmod 2$. Let c be a fixed column in which $\vec{v'}$ and $\vec{r}$ both have a value of 1 in column c. Let $\vec{e_c}$ be the $(k-2)$–vector that has a value

of 1 in column c and a value of 0 in every other column. Given A'_0 and $\vec{v'}$, the function $\mathcal{S}_1(A'_0, \vec{v'}, \vec{r}, c)$ produces a $(k-2) \times (k-2)$ matrix as follows:

- Row $\vec{r}$ is replaced with $\vec{v'} \oplus \vec{e_c}$.
- Each row $\vec{x}$ in A'_0 (excluding row $\vec{r}$) in which both $\vec{v'} \bullet \vec{x} \equiv 1 \bmod 2$ and $\vec{x}$ is 1 in column c is replaced with $\vec{e_c} \oplus \vec{r} \oplus \vec{x}$.
- Each row $\vec{x}$ in A'_0 in which both $\vec{v'} \bullet \vec{x} \equiv 1 \bmod 2$ and $\vec{x}$ is 0 in column c is replaced with $\vec{v'} \oplus \vec{e_c} \oplus \vec{r} \oplus \vec{x}$.
- Each row $\vec{x}$ in A'_0 in which both $\vec{v'} \bullet \vec{x} \equiv 0 \bmod 2$ and $\vec{x}$ is 1 in column c is replaced with $\vec{v'} \oplus \vec{x}$.
- Each row $\vec{x}$ in A'_0 in which both $\vec{v'} \bullet \vec{x} \equiv 0 \bmod 2$ and $\vec{x}$ is 0 in column c is left unchanged.

Definition 5.7 Define $v\vec{\mathcal{S}}_1(A'_0, \vec{v'}, \vec{r}, c) = \vec{v'} \oplus \vec{e_c} \oplus \vec{r}$ where $\vec{e_c}$ and $\vec{r}$ are defined as in Definition 5.6.

With these two definitions, we have $A'''_0 = \mathcal{S}_1(A'_0, \vec{v'}, \vec{r}, c)$ and $\vec{v'''} = v\vec{\mathcal{S}}_1(A'_0, \vec{v'}, \vec{r}, c)$, where $\vec{r}$ is any row in A'_0 such that $\vec{v'} \bullet \vec{r} \equiv 1 \bmod 2$ and c is any column in which both $\vec{v'}$ and $\vec{r}$ have a value of 1.

We now have two functions, $\mathcal{S}_0(A'_0, \vec{v'}, \vec{r}, c)$ and $\mathcal{S}_1(A'_0, \vec{v'}, \vec{r}, c)$, that give us a generator matrix for a code under c_1, c_3, and we have two functions, $v\vec{\mathcal{S}}_0(A'_0, \vec{v'}, \vec{r}, c)$ and $v\vec{\mathcal{S}}_1(A'_0, \vec{v'}, \vec{r}, c)$, that give us appropriate extension vectors. The functions $\mathcal{S}_0(A'_0, \vec{v'}, \vec{r}, c)$ and $v\vec{\mathcal{S}}_0(A'_0, \vec{v'}, \vec{r}, c)$ are used when we have a row $\vec{r}$ and column c of A'_0 in which $\vec{v'}$ has a value of 1 in column c and $\vec{r}$ has a value of 0 in column c. The functions $\mathcal{S}_1(A'_0, \vec{v'}, \vec{r}, c)$ and $v\vec{\mathcal{S}}_1(A'_0, \vec{v'}, \vec{r}, c)$ are used when we have a row $\vec{r}$ and column c of A'_0 in which both $\vec{v'}$ and $\vec{r}$ have a value of 1 in column c. In order to talk about a generator matrix and extension vector for a code under c_1, c_3 without making a distinction between these two cases, we will now define two functions to cover both cases.

Definition 5.8 Let $G'_0 = [I_{k-2}|A'_0]$ be a generator matrix for a $(2k-4, k-2)$ self-dual code C'_0 and let $\vec{v'}$ be a $(k-2)$-vector that extends C'_0 to a $(2k, k, 4)$ self-dual code. Let $\vec{r}$ be a fixed row in A'_0 such that $\vec{v'} \bullet \vec{r} \equiv 1 \bmod 2$. Let c be a fixed column in which $\vec{v'}$ has a value of 1. The function $\mathcal{S}(A'_0, \vec{v'}, \vec{r}, c)$ produces a $(k-2) \times (k-2)$ matrix as follows:

$$\mathcal{S}(A'_0, \vec{v'}, \vec{r}, c) = \begin{cases} \mathcal{S}_0(A'_0, \vec{v'}, \vec{r}, c) & \text{if } \vec{r} \text{ is 0 in column } c, \\ \mathcal{S}_1(A'_0, \vec{v'}, \vec{r}, c) & \text{if } \vec{r} \text{ is 1 in column } c. \end{cases}$$

The function $v\vec{\mathcal{S}}(A'_0, \vec{v'}, \vec{r}, c)$ produces a $(k-2)$-vector as follows:

$$v\vec{\mathcal{S}}(A'_0, \vec{v'}, \vec{r}, c) = \begin{cases} v\vec{\mathcal{S}}_0(A'_0, \vec{v'}, \vec{r}, c) & \text{if } \vec{r} \text{ is 0 in column } c, \\ v\vec{\mathcal{S}}_1(A'_0, \vec{v'}, \vec{r}, c) & \text{if } \vec{r} \text{ is 1 in column } c. \end{cases}$$

Let us now summarize what we have found in this section. Let C be any $(2k, k, 4)$ self-dual code and let $\vec{w}$ be a weight 4 word in C. Then there are at most three inequivalent codes under the weight 4 word $\vec{w}$ of C. That is, there are at most three inequivalent $(2k-4, k-2)$ self-dual codes C_0, generated by $G_0 = [I_k|A_0]$, for which there exists an extension vector $\vec{v}$ such that $G' = [I_k|\mathcal{A}(A_0, \vec{v})]$ generates a code equivalent C', equivalent to C, where the first two rows of G' sum to the weight 4 word $\vec{w'}$ of C' that corresponds to the weight 4 word $\vec{w}$ of C. Given a generator matrix $G = [I_k|A]$ for C, and the weight 4 word $\vec{w}$, we can easily find generator matrices and extension vectors for the at most three inequivalent codes under the weight 4 word $\vec{w}$ of C. The first step is to apply a sequence of $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c operations to A, producing a matrix $A' = \mathcal{A}(A'_0, \vec{v'})$, in which $G' = [I_k|A']$ generates a code C' equivalent to C, and the first two rows of $G' = [I_k|A']$ sum to the weight 4 word $\vec{w'}$ of C' that corresponds to the weight 4 word $\vec{w}$ of C. The matrix $A' = \mathcal{A}(A'_0, \vec{v'})$ can then be used to produce generator matrices and extension vectors for the three codes under the weight 4 word $\vec{w}$ of C. That is, $G'_0 = [I_{k-2}|A'_0]$, $G''_0 = [I_{k-2}|A''_0]$, and $G'''_0 = [I_{k-2}|A'''_0]$, where $A''_0 = \mathcal{R}(A'_0, \vec{v'})$ and $A'''_0 = \mathcal{S}(A'_0, \vec{v'}, \vec{r}, c)$ are generator matrices for the at most three inequivalent codes under the weight 4 word $\vec{w}$ of C, and $\vec{v'}$, $\vec{v''} = v\vec{\mathcal{R}}(A'_0, \vec{v'})$, and $\vec{v'''} = v\vec{\mathcal{S}}(A'_0, \vec{v'}, \vec{r}, c)$ are extension vectors that extend each of the codes to a code equivalent to C. More specifically,

$G' = [I_k|\mathcal{A}(A_0', \vec{v'})]$, $G'' = [I_k|\mathcal{A}(A_0'', \vec{v''})]$ and $G''' = [I_k|\mathcal{A}(A_0''', \vec{v'''})]$ all generate codes equivalent to C, and the first two rows of each of G', G'', and G''' sum to a weight 4 word that corresponds to the weight 4 word $\vec{w}$ of C.

Given a generator matrix $G = [I_k|A]$ for a $(2k, k, 4)$ self-dual code C, we can find a list of generator matrices containing at least one generator matrix for each of the inequivalent $(2k-4, k-2)$ self-dual codes that can be extended to a code that is equivalent to C by simply finding generator matrices $G' = [I_k|\mathcal{A}(A_0', \vec{v'})]$ for each of the weight 4 words $\vec{w}$ in C. For each weight 4 word $\vec{w}$ in C, the generator matrix G' we produce can then be used to find generator matrices and extension vectors for the at most three inequivalent codes under the weight 4 word $\vec{w}$ of C.

One use in finding generator matrices for all the inequivalent $(2k-4, k-2)$ self-dual codes that can be extended to a code equivalent to C is to further reduce the number of equivalent $(2k, k, 4)$ self-dual codes produced by our algorithm in Chapter 4 for enumerating the $(2k, k, 4)$ self-dual codes. For example, suppose we produce a $(2k, k, 4)$ d_1-code C from a $(2k-4, k-2, 4)$ d_1-code C_0. If we then find all the $(2k-4, k-2)$ self-dual codes that can be extended to a code equivalent to C, and then discover that one of them contains, say, a d_2-block, then we know there exists another inequivalent $(2k-4, k-2)$ self-dual code that can be extended to a code equivalent to C. Therefore, we can discard C. That is, the only codes produced that are equivalent to C that will be kept are the codes that were produced by extending a $(2k-4, k-2)$ self-dual code that contained a d_2-block.

In this chapter, knowledge of the inequivalent $(2k-4, k-2)$ self-dual codes that can be extended to a code equivalent to C will be used to develop an algorithm for enumerating the $(2k-4, k-2, d \geq 6)$ self-dual codes from the $(2k-4, k-2, 4)$ self-dual codes containing no intersecting weight 4 words.

5.2 An Extension of the $(2k, k, d \geq 6)$ Self-Dual Codes

In this section, it will be shown that for any $(2k, k, d \geq 6)$ self-dual code, there exists an equivalent code C_0', generated by $G_0' = [I_k | A_0']$, and an extension vector $\vec{v'}$, such that $G' = [I_{k+2} | \mathcal{A}(A_0', \vec{v'})]$ generates a $(2k+4, k+2, 4)$ self-dual code C' that contains one and only one weight 4 word. The results of Section 5.1 are then used to find generator matrices for the at most three inequivalent $(2k, k)$ self-dual codes that can be extended to a code equivalent to C'. It is then shown that among these three codes, one is the original $(2k, k, d \geq 6)$ self-dual code, and another is a $(2k, k, 4)$ self-dual code that does not contain any intersecting weight 4 words. The results of this section will be used in Section 5.3 to develop our algorithm for enumerating the $(2k, k, d \geq 6)$ self-dual codes from any complete list of inequivalent $(2k, k, 4)$ self-dual codes containing no intersecting weight 4 words.

Let us begin by finding a canonical generator matrix for the $(2k, k, d \geq 6)$ self-dual codes. Recall that Theorem 2.18 of Section 2.5 states that for any (n, k, d) code C and any weight d word $\vec{w}$ in C, there exists an equivalent code C', generated by $G' = [I_k | A']$, such that the weight d word $\vec{w'}$ of C', that corresponds to the weight d word $\vec{w}$ of C, is a row in G'. Therefore, for any $(2k, k, d \geq 6)$ self-dual code C_0 and weight d word $\vec{w}$ in C, there exists an equivalent code C_0', generated by $G_0' = [I_k | A_0']$, where:

$$A_0' = \left[\begin{array}{c|c|c|c} 0 & 1^{d-3} & 1\ 1 & 0^* \\ \hline 1 & W_0 & W_1 & W_2 \\ 0 & X_0 & X_1 & X_2 \\ \hline 1 & Y_0 & Y_1 & Y_2 \\ 0 & Z_0 & Z_1 & Z_2 \end{array}\right] \begin{array}{l} \\ \\ \text{for any } w \in W_0,\ w(w) \equiv 0 \bmod 2 \\ \text{for any } x \in X_0,\ \ w(x) \equiv 1 \bmod 2 \\ \text{for any } y \in Y_0,\ \ w(y) \equiv 1 \bmod 2 \\ \text{for any } z \in Z_0,\ \ w(z) \equiv 0 \bmod 2 \end{array}$$

$$\underbrace{\qquad\qquad}_{S_0}$$

where the first row of G_0' is the weight d word $\vec{w'}$ of C_0' that corresponds to the weight d

word $\vec{w}$ of C_0. The the last four rows of A'_0 represent all the different ways the remaining rows of A'_0 can intersect the first row of A'_0 in the first $d-2$ columns of A'_0 (i.e. the set of columns S_0).

If we extend C'_0 with the vector $\vec{v'} = 1\ 1^{d-3}\ 0\ 0\ 0^*$, the resulting code is the $(2k+4, k+2, 4)$ self-dual code C', generated by $G' = [I_{k+2}|\mathcal{A}(A'_0, \vec{v'})]$, where:

$$\mathcal{A}(A'_0, \vec{v'}) = \left[\begin{array}{cc||c|c|cc|c} 1 & 0 & 1 & 1^{d-3} & 0 & 0 & 0^* \\ 0 & 1 & 1 & 1^{d-3} & 0 & 0 & 0^* \\ \hline\hline 1 & 1 & 0 & 1^{d-3} & 1 & 1 & 0^* \\ \hline 1 & 1 & 1 & W_0 & W_1 & & W_2 \\ \hline 1 & 1 & 0 & X_0 & X_1 & & X_2 \\ \hline 0 & 0 & 1 & Y_0 & Y_1 & & Y_2 \\ \hline 0 & 0 & 0 & Z_0 & Z_1 & & Z_2 \end{array}\right]$$

We will refer to C' as the *extended code* over the weight d word $\vec{w'}$ of the $(2k, k, d \geq 6)$ self-dual code C'_0.

Theorem 5.9 The extended code C' contains one and only one weight 4 word (the weight 4 word that is the sum of the first two rows of G').

Proof We will prove this by partitioning the codewords of the $(2k+4, k+2, 4)$ self-dual code C' into four disjoint sets and, using the fact that the $(2k, k, d)$ self-dual code C'_0 has distance $d \geq 6$, prove that none of the four sets contain a weight 4 word (other than the one already mentioned).

Let us first set up our proof. Our $(2k+4, k+2, 4)$ self-dual code C' is generated by

a matrix of the form:

$$G' = [I_{k+2}|\mathcal{A}(A'_0, \vec{v'})] =$$

$$\begin{array}{c}
\begin{array}{cccccccccccccccc}
a_1 & a_2 & b_1 & b_2 & b_3 & \cdots & b_k & c_1 & c_2 & d_1 & d_2 & d_3 & \cdots & d_k & & \\
\end{array} \\
\left[\begin{array}{cc||c|cc|c|cc||c|c|cc|c|c}
1 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 1 & 1^{d-3} & 0 & 0 & 0^* & \vec{w_1} \\
0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 1 & 1^{d-3} & 0 & 0 & 0^* & \vec{w_2} \\
\hline
0 & 0 & 1 & 0 & 0 & \cdots & 0 & 1 & 1 & 0 & 1^{d-3} & 1 & 1 & 0^* & \vec{w_3} \\
\hline
0 & 0 & 0 & \multicolumn{3}{c|}{\multirow{3}{*}{I_{k-1}}} & & 1 & 1 & \multicolumn{4}{c|}{\multirow{3}{*}{U_0}} & & \\
0 & 0 & 0 & & & & & \vdots & \vdots & & & & & & \\
\vdots & \vdots & \vdots & & & & & 0 & 0 & & & & & & \\
0 & 0 & 0 & & & & & \vdots & \vdots & & & & & &
\end{array}\right] \begin{array}{l} \\ \\ \\

codeword of C' is in S_A. Since any non-zero codeword in C'_0 has weight greater than or equal to $d > 4$, the non-zero codewords in S_A all have weight > 4. Therefore, S_A does not contain any weight 4 words.

Let us now consider the set of codewords $S_B - \{\vec{w_1} \oplus \vec{w_2}\}$ (i.e. the set S_B with the weight 4 word $\vec{w_1} \oplus \vec{w_2}$ removed). Since $\vec{w_1} \oplus \vec{w_2} = (1,1,0,0,\ldots,0,1,1,0,0,\ldots,0)$, the codewords of $S_B - \{\vec{w_1} \oplus \vec{w_2}\}$ are the non-zero codewords of C'_0 with four extra components added to them, and therefore, have weight $\geq d > 4$. Thus, the only weight 4 word in S_B is $\vec{w_1} \oplus \vec{w_2}$.

Let us now consider the set of codewords $S_C = S_A \oplus \vec{w_1}$. The codewords of S_C consist of all codewords of the form $\vec{c_a} \oplus \vec{w_1}$, where $\vec{c_a}$ is a codeword of S_A. The codewords of S_A consist of the codewords of the distance $d \geq 6$ code C'_0 with four extra components a_1, a_2, c_1, c_2 added to them, where a_1, a_2 are 0 0 and c_1, c_2 are 0 0 or 1 1. We will prove that S_C does not contain any weight 4 words by first assuming that S_C does contain a weight 4 word $\vec{c_c}$ and then showing that $\vec{c_c}$ cannot exist.

Suppose that S_C contains a weight 4 word $\vec{c_c} = \vec{c_a} \oplus \vec{w_1}$. Since $\vec{w_1}$ has weight d, where $d \geq 6$, $\vec{c_a}$ cannot be the zero-codeword, which implies that $\vec{c_a}$ has weight $\geq d$. Since C' is a self-dual code (and thus the codewords of C' intersect in an even number of positions), there are only two ways in which any two non-zero codewords, $\vec{c_1}$ and $\vec{c_2}$, of C' can sum to a weight 4 word:

$\vec{c_1}$	=	1	1	1	1	(1 1)*	0*		$\vec{c_1}$	=	1	1	0	0	(1 1)*	0*
$\vec{c_2}$	=	0	0	0	0	(1 1)*	0*		$\vec{c_2}$	=	0	0	1	1	(1 1)*	0*

Our two non-zero codewords $\vec{c_a}$ and $\vec{w_1}$ cannot be the first case:

- Since $\vec{c_a}$ has weight $\geq d$ and $\vec{w_1}$ has weight d, $\vec{c_a}$ cannot be $\vec{c_2}$, and
- Since $\vec{w_1}$ has a value of 1 in column a_0 and $\vec{c_a}$ has a value of 0 in column a_0, $\vec{w_1}$ cannot be $\vec{c_2}$.

Therefore, $\vec{c_a}$ and $\vec{w_1}$ must intersect as follows:

$$1\ \ 1\ \ 0\ \ 0\ \ (1\,1)^*\ \ 0^*$$
$$0\ \ 0\ \ 1\ \ 1\ \ (1\,1)^*\ \ 0^*$$

It will now be shown such a $\vec{c_a}$ cannot exist.

Consider the codeword $\vec{w_3}$ of C' (i.e. the third row of G'). $\vec{w_3}$ is an element of S_A. Furthermore, since $\vec{w_1} \oplus \vec{w_3}$ has weight 6, $\vec{w_3}$ is not $\vec{c_a}$. Let us now consider all the different ways in which $\vec{c_a}$ may intersect $\vec{w_3}$ using the facts that C' is a self-dual code, $\vec{c_a}$ is an element of S_A, and $\vec{c_a}$ and $\vec{w_3}$ intersect as previously stated. Using these facts, we have three different possibilities for the way in which $\vec{c_a}$ may intersect $\vec{w_3}$. These three possibilities are given in Table 5.1.

	a_1	a_2	c_1	c_2	S_1	S_2	S_3	S_4
$\vec{w_1}$	1	0	1	0	1	1^{d-3}	0 0 0	0^*
$\vec{w_3}$	0	0	1	1	0	1^{d-3}	1 1 1	0^*
Case 1	0	0	0	0	1	1^{d-3}	1 0 0	$1\,0^*$
Case 2	0	0	1	1	1	$1^{d-4}\,0$	0 0 0	$1\,0^*$
Case 3	0	0	1	1	0	1^{d-3}	1 0 0	0^*

Table 5.1: The three different ways in which $\vec{c_a}$ can intersect $\vec{w_3}$.

The information given in Table 5.1 is as follows: The first row of the table gives us the column indices of the codewords in C'. The labels S_1, S_2, S_3, and S_4 represent sets of column indices. $S_1 = \{d_1\}$ is the column in C' (excluding a_1) in which $\vec{w_1}$ has a value of 1 and $\vec{w_3}$ has a value of 0. $S_2 = \{d_2, d_3, \ldots, d_{d-2}\}$ is the set of columns (excluding c_1) in which both $\vec{w_1}$ and $\vec{w_3}$ have a value of 1. $S_3 = \{b_1, d_{d-1}, d_d\}$ is the set of columns (excluding c_2) in which $\vec{w_1}$ has a value of 0 and $\vec{w_3}$ has a value of 1. $S_4 = \{b_2, b_3, \ldots, b_k, d_{d+1}, d_{d+2}, \ldots, d_k\}$ is the set of columns (excluding a_2) in which both $\vec{w_1}$ and $\vec{w_3}$ have a value of 0. The second row of the table is the codeword $\vec{w_1}$. The

third row of the table is the codeword $\vec{w_3}$. The last three rows in the table are the three different possibilities for the codeword $\vec{c_a}$. The three different possibilities are determined by the boldface entries. That is, the boldface entries fix the remaining columns in each case, based on the following three facts:

- $\vec{c_a}$ is equal to 0 0 in columns a_1, a_2.
- $\vec{c_a}$ intersects $\vec{w_3}$ in an even number of positions.
- $\vec{c_a}$ and $\vec{w_1}$ intersect as follows:

$$\begin{array}{cccccc} 1 & 1 & 0 & 0 & 1^* & 0^* \\ 0 & 0 & 1 & 1 & 1^* & 0^* \end{array}$$

It will now be shown that each of the three possibilities for the codeword $\vec{c_a}$ given in Table 5.1 contradicts the distance of the $(2k, k, d \geq 6)$ self-dual code C_0':

- Suppose $\vec{c_a}$ intersects $\vec{w_3}$ as given in case 1. Consider the codeword $\vec{c}$ of C_0' corresponding to the codeword $c_a \oplus \vec{w_3}$ (which is an element of S_A). $\vec{c}$ is equal to $c_a \oplus \vec{w_3}$ with columns a_0, a_1, c_0, c_1 deleted. However, $\vec{c}$ has weight 4, which contradicts the distance of C_0' being ≥ 6. Therefore, $\vec{c_a}$ cannot intersect $\vec{w_3}$ as given in case 1.
- Suppose $\vec{c_a}$ intersects $\vec{w_3}$ as in case 2 or case 3. Consider the codeword of C_0' corresponding to $\vec{c_a}$. In both cases, it has weight $d-2$, which contradicts the distance of C_0' being d. Therefore, $\vec{c_a}$ cannot intersect $\vec{w_3}$ as given in case 2 and as given in case 3.

Therefore, there does not exist a $\vec{c_a} \in S_A$ such that $\vec{c_c} = \vec{c_a} \oplus \vec{w_1}$ has weight 4. Therefore, S_C does not contain any weight 4 words.

Using this same method we can show that S_D does not contain any weight 4 words. This completes all of our cases.

Therefore, the only weight 4 word in C' is $\vec{w_1} \oplus \vec{w_2}$. □

Since the extended code C', generated by $G' = [I_{k+2}|\mathcal{A}(A'_0, \vec{v'})]$, contains exactly one weight 4 word, we know, from Section 5.1, that there are at most three inequivalent $(2k, k)$ self-dual codes that can be extended to a code equivalent to C'. One of these codes is the original $(2k, k, d \geq 6)$ self-dual code C'_0, generated by $G'_0 = [I_k|A'_0]$:

$$A'_0 = \left[\begin{array}{c|c|c|c} 0 & 1^{d-3} & 1\ 1 & 0^* \\ \hline 1 & W_0 & W_1 & W_2 \\ 0 & X_0 & X_1 & X_2 \\ \hline 1 & Y_0 & Y_1 & Y_2 \\ 0 & Z_0 & Z_1 & Z_2 \end{array}\right]$$

The vector $\vec{v'} = 1\ 1^{d-3}\ 0\,0\ 0^*$ is an extension vector that extends C'_0 to C'. The other two codes are generated by $G''_0 = [I_k|A''_0]$ and $G'''_0 = [I_k|A'''_0]$, where:

$A''_0 = \mathcal{R}(A'_0, \vec{v'}) =$

$$\left[\begin{array}{c|c|c|c} 1 & 0^{d-3} & 1\ 1 & 0^* \\ \hline 0 & \overline{W_0} & W_1 & W_2 \\ 1 & \overline{X_0} & X_1 & X_2 \\ \hline 1 & Y_0 & Y_1 & Y_2 \\ 0 & Z_0 & Z_1 & Z_2 \end{array}\right]$$

$A'''_0 = \mathcal{S}(A'_0, \vec{v'}, \vec{r}, c) =$

$$\left[\begin{array}{c|c|c|c} 0 & 1^{d-3} & 0\ 0 & 0^* \\ \hline 1 & W_0 & \overline{W_1} & W_2 \\ 1 & \overline{X_0} & \overline{X_1} & X_2 \\ \hline 0 & \overline{Y_0} & Y_1 & Y_2 \\ 0 & Z_0 & Z_1 & Z_2 \end{array}\right]$$

(where, for the function $\mathcal{S}(A'_0, \vec{v'}, \vec{r}, c)$, we have used the first row of A'_0 for $\vec{r}$ and column 1 for c). $\overline{W_0}$ is the matrix W_0 with all the rows complemented. $G''_0 = [I_k|A''_0]$ generates a $(2k, k, 4)$ self-dual code C''_0. $G'''_0 = [I_k|A'''_0]$ generates a $(2k, k, d_0)$ self-dual code C'''_0, where $4 \leq d_0 \leq d - 2$. Note, since C' contains only one weight 4 word, neither C''_0 nor C'''_0 have distance 2. The vector $\vec{v''} = \vec{v_{\mathcal{R}}}(A'_0, \vec{v'}) = 1\ 1^{d-3}\ 0\,0\ 0^*$ extends C''_0 to a code equivalent to C'. The vector $\vec{v'''} = \vec{v_{\mathcal{S}}}(A'_0, \vec{v'}, \vec{r}, c) = 1\ 1^{d-3}\ 1\,1\ 0^*$ extends C'''_0 to a code equivalent to C'.

We will refer to the $(2k, k, 4)$ self-dual code C_0'' as an *extension mate* of the $(2k, k, d \geq 6)$ self-dual code C_0'. We will refer to all three codes C_0', C_0'', and C_0''' as *related codes.*

Theorem 5.10 The extension mate C_0'' does not contain any intersecting weight 4 words.

Proof The codewords of C_0'' are generated by the matrix $G_0'' = [I_k | A_0'']$. Suppose C_0'' contains two intersecting weight 4 words $\vec{c_a}$ and $\vec{c_b}$:

$$\begin{array}{lcccccccc} \vec{c_a} & 1 & 1 & 1 & 1 & 0 & 0 & 0^* \\ \vec{c_b} & 1 & 1 & 0 & 0 & 1 & 1 & 0^* \end{array}$$

Let R_a and R_b denote the rows of G_0'' that sum to $\vec{c_a}$ and $\vec{c_b}$, respectively.

Now, C_0'' can be extended to a $(2k+4, k+2, 4)$ self-dual code C'', equivalent to C', such that C'' contains one and only one weight 4 word. C'' is generated by the matrix $G'' = [I_{k+2} | \mathcal{A}(A_0'', \vec{v''})]$. The first two rows of G'' sum to the one and only one weight 4 word of C''. Let R_a'' and R_b'' denote the rows of G'' that correspond to the rows R_a and R_b, respectively, of G_0''. Note that neither R_a'' nor R_b'' include the first two rows of G''. Consider the two codewords of C'' that are generated by R_a'' and R_b'', respectively. The two codewords generated by these sets of rows are equal to the codewords $\vec{c_a}$ and $\vec{c_b}$ of C_0'' with four extra components added to them. Two of these components will be 0 0 in both codewords (the two components corresponding to the first two columns of I_{k+2}). The other two components will be either 0 0 or 1 1 (the two components corresponding to the first two columns of $\mathcal{A}(A_0'', \vec{v''})$). Therefore, with the four extra components given first, C'' will contain the following two codewords:

$$\begin{array}{cccc|ccccccc} 0 & 0 & a & a & 1 & 1 & 1 & 1 & 0 & 0 & 0^* \\ 0 & 0 & b & b & 1 & 1 & 0 & 0 & 1 & 1 & 0^* \end{array}$$

However, if $a = b$ then these two codewords sum to a weight 4 word (which is not the weight 4 word that is the sum of the first two rows of G'') and if $a \neq b$ then one of these

codewords is a weight 4 word (which is not the weight 4 word that is the sum of the first two rows of G''). In either case, we have found a weight 4 word in C'' that is not the weight 4 word of C'' that is the sum of the first two rows of G''. This contradicts the fact that C'' contains one and only one weight 4 word.

Therefore, the extension mate C_0'' does not contain any intersecting weight 4 words.

□

We will now give an example of the two related codes C_0'' and C_0''' for a given $(2k, k, d \geq 6)$ self-dual code C_0' and extension vector $\vec{v'}$.

Example 5.11 Let C_0' be the $(24, 12, 8)$ self-dual code generated by $G_0' = [I_{12}|A_0']$ where:

$$A_0' = \left[\begin{array}{cccccc|cc|cccc} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{array}\right]$$

Let $\vec{v'} = (1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0)$. Then $G' = [I_{14}|\mathcal{A}(A_0', \vec{v'})]$ generates a $(28, 14, 4)$ self-dual

code C' that contains one and only one weight 4 word, where:

$$\mathcal{A}(A'_0, \vec{v'}) = \left[\begin{array}{cc||c|cccccc|cc|cccc}
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline\hline
1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1
\end{array}\right]$$

There are exactly two other $(24, 12)$ self-dual codes that can be extended to a code that is equivalent to C'. They are the $(24, 12, 4)$ self-dual code C''_0, generated by $G''_0 = [I_{12}|A''_0]$,

and the $(24, 12, 6)$ self-dual code C_0''', generated by $G_0''' = [I_{12}|A_0''']$, where:

$\mathcal{R}(A_0', \vec{v'}) =$

1	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	0	1	1	0	0
0	0	0	0	1	0	0	0	0	0	1	1
0	1	0	0	1	1	1	1	1	0	1	0
1	0	0	1	0	1	1	0	1	0	1	1
1	0	1	0	1	0	1	0	1	1	1	0
1	1	0	0	1	1	1	0	0	1	0	1
1	1	0	0	0	0	0	1	1	1	1	1
1	0	1	0	1	1	0	1	1	0	0	1
1	0	0	1	1	1	0	1	0	1	1	0
0	0	1	0	0	1	1	1	0	1	1	1
0	0	0	1	1	0	1	1	1	1	0	1

$\mathcal{S}(A_0', \vec{v'}, \vec{r}, c) =$

0	1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	0	1	1	1	0	0	1	1
1	0	1	1	0	0	0	0	1	0	1	0
1	0	0	1	0	1	0	1	1	0	1	1
1	0	1	0	1	0	0	1	1	1	1	0
0	0	1	1	0	0	1	0	0	1	0	1
0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	0	0	0	1	1	0	0	1
0	1	1	0	0	0	0	1	0	1	1	0
0	0	1	0	0	1	1	1	0	1	1	1
0	0	0	1	1	0	1	1	1	1	0	1

(where, for the function $\mathcal{S}(A_0', \vec{v'}, \vec{r}, c)$, we have use the first row of A_0' for $\vec{r}$ and column 1 for c). The code C_0'' does not contain any intersecting weight 4 words. Extension vectors that extend C_0'' and C_0''' to codes that are equivalent to C' are $\vec{v''} = \vec{v_{\mathcal{R}}}(A_0', \vec{v'}) =$ (1 1 1 1 1 1 0 0 0 0 0 0) and $\vec{v'''} = \vec{v_{\mathcal{S}}}(A_0', \vec{v'}, \vec{r}, c) =$ (1 1 1 1 1 1 1 1 0 0 0 0), respectively.

Let us now summarize what we have found in this section. Let C be any $(2k, k, d \geq 6)$ self-dual code. Then there exists an equivalent code C_0', generated by $G_0' = [I_k|A_0']$, such that C_0' can be extended to a $(2k+4, k+2, 4)$ self-dual code C' that contains one and only one weight 4 word. Furthermore, there are at most two other inequivalent $(2k, k)$ self-dual codes, C_0'' and C_0''', that can be extended to a code equivalent to C'. C_0'' is a $(2k, k, 4)$ self-dual code that does not contain any intersecting weight 4 words. C_0''' is a $(2k, k, d_0)$ self-dual code, where $4 \leq d_0 \leq d-2$. In the next section, we will use the results of this section to develop our recursive algorithm for enumerating the $(2k, k, d \geq 6)$ self-dual codes from any complete list of inequivalent $(2k, k, 4)$ self-dual codes containing no

intersecting weight 4 words.

5.3 Enumeration of the $(2k, k, d \geq 6)$ Self-Dual Codes

We can use the results of Section 5.2 to enumerate a list L of $(2k, k, d \geq 6)$ self-dual codes containing at least one representative for each inequivalent code as follow:

- for each inequivalent $(2k, k, 4)$ self-dual code C_0 that does not contain any intersecting weight 4 words, find all of the ways that C_0 can be extended to a $(2k+4, k+2, 4)$ self-dual code C which contains one and only one weight 4 word.
- for each C found, find the other two $(2k, k)$ self-dual codes, C_r and C_s, that can be extended to a code equivalent to C.
- if one of the two codes, say C_r, has distance $d \geq 6$, and if the other one, say C_s, has distance $4 \leq d_0 \leq d-2$, then add the code C_r to the list L.

Since for any $(2k, k, d \geq 6)$ self-dual code there exists an equivalent code C_0' that can be extended to a $(2k+4, k+2, 4)$ self-dual code C' (that contains one and only one weight 4 word) for which there exists at most two other inequivalent $(2k, k)$ self-dual codes C_0'' and C_0''' that can be extended to a code equivalent to C', where C_0'' has distance 4 (and contains no intersecting weight 4 words) and C_0''' has distance $4 \leq d_0 \leq d-2$, we know that the list L of $(2k, k, d \geq 6)$ self-dual codes we produce will contain at least one representative for each inequivalent $(2k, k, d \geq 6)$ self-dual code. That is, for any $(2k, k, d \geq 6)$ self-dual code C there will exist at least one code in L that is equivalent to C.

In this section, we will use the basic outline just presented to develop our algorithm for enumerating the $(2k, k, d \geq 6)$ self-dual codes from any complete list of inequivalent $(2k, k, 4)$ self-dual codes containing no intersecting weight 4 words. We will begin by determining for a given $G_0 = [I_k | A_0]$, where G_0 generates a $(2k, k, 4)$ self-dual code C_0

containing no intersecting weight 4 words, subsets of the different extension vectors $\vec{v}$ that always result in equivalent codes being produced. This will allow us to reduce the number of extension vectors that we will need to consider. We will then find all of the different extension vectors $\vec{v}$ that result in a matrix $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$ that generates a $(2k+4, k+2, 4)$ self-dual code C containing one and only one weight 4 word. This will allow us to produce our extension vectors $\vec{v}$, and thus find generator matrices for the two related codes C_r and C_s, without actually producing the matrix G. This will also allow us to further reduce the number of extension vectors that we will need to consider. We will then show how we can produce generator matrices for both C_r and C_s using the matrix A_0, the vector $\vec{v}$, and the function $\mathcal{R}(A_0, \vec{v})$ we defined in Section 5.1. We will then conclude this section by giving a formal description of our algorithm for enumerating the $(2k, k, d \geq 6)$ self-dual codes.

5.3.1 Equivalent Extensions of The d_1-Codes

Let $G_0 = [I_k|A_0]$ be a generator matrix for a $(2k, k, 4)$ self-dual code C_0 that does not contain any intersecting weight 4 words. Suppose C_0 contains exactly m weight 4 words. Each weight 4 word in C_0 is either a row of G_0 or is generated by two or three rows of G_0. Furthermore, since none of the weight 4 words in C_0 intersect, the weight 4 words of C_0 are generated by disjoint sets of rows of G_0. Therefore, the rows of G_0 can be partitioned into the $m+1$ disjoint sets $G_{B_1}, G_{B_2}, \ldots, G_{B_m}, G_R$ where G_{B_i}, $1 \leq i \leq m$, generates a weight 4 word and G_R is the set of the remaining rows of G_0. Furthermore, since none of the weight 4 words in C_0 intersect, the columns of G_0 can be partitioned into the $m+1$ disjoint sets $S_{B_1}, S_{B_2}, \ldots, S_{B_m}, S_R$ where S_{B_i}, $1 \leq i \leq m$, are the *columns* of the weight 4 word generated by G_{B_i} and S_R is the set of the remaining columns of G_0. What is meant by the columns of the weight 4 word generated by G_{B_i} is the four columns of G_0 in which the rows of G_{B_i} sum to one.

In terms of the terminology introduced in Sections 4.2 and 4.3 of Chapter 4, the

$(2k, k, 4)$ self-dual code C_0, generated by $G_0 = [I_k|A_0]$, is a d_1-code. The m disjoint sets of rows $G_{B_1}, G_{B_2}, \ldots, G_{B_m}$ are d_1-blocks. If $|G_{B_i}| = 1$ then G_{B_i} is a $d_{1,0}$-block, if $|G_{B_i}| = 2$ then G_{B_i} is a $d_{1,3}$-block, and if $|G_{B_i}| = 3$ then G_{B_i} is a $d_{1,4}$-block. The columns in the set S_{B_i} are the columns of the d_1-block G_{B_i}.

Let $\vec{v}$ be an extension vector. Then since the columns of $G_0 = [I_k|A_0]$ can be partitioned into the $m+1$ disjoint sets $S_{B_1}, S_{B_2}, \ldots, S_{B_m}, S_R$ the columns of $\vec{v}$ can be partitioned into $m+1$ disjoint sets $v_{\vec{B}_1}, v_{\vec{B}_2}, \ldots, v_{\vec{B}_m}, v_{\vec{R}}$ where $v_{\vec{B}_i}$ consists of the columns of $\vec{v}$ that correspond to the subset of columns of S_{B_i} that are columns in A_0 and $v_{\vec{R}}$ consists of the remaining columns of $\vec{v}$. In terms of the terminology introduced in Section 4.5 of Chapter 4, $v_{\vec{B}_i}$ is an extension of the generator block G_{B_i}.

Example 5.12 Let C_0 be the $(26, 13, 4)$ d_1-code generated by $G_0 = [I_k|A_0]$, where:

$$A_0 = \left[\begin{array}{ccc|ccc|ccc|cc|cc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline
1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1
\end{array}\right]$$

C_0 contains exactly five weight 4 words. The rows of G_0 partition into the following six disjoint sets:

$$G_{B_1} = \{1\}$$

$$
\begin{aligned}
G_{B_2} &= \{2\} \\
G_{B_3} &= \{3\} \\
G_{B_4} &= \{4,5\} \\
G_{B_5} &= \{6,7,8\} \\
G_R &= \{9,10,11,12,13\}
\end{aligned}
$$

The first five sets each generate a weight 4 word while the last set consists of the remaining rows of G_0. $G_{B_1}, G_{B_2}, G_{B_3}$ are all $d_{1,0}$-blocks, G_{B_4} is a $d_{1,3}$-block, and G_{B_5} is a $d_{1,4}$-block. The columns of G_0 also partition into six disjoint sets $S_{B_1}, S_{B_2}, S_{B_3}, S_{B_4}, S_{B_5}, S_R$. The set S_{B_i}, $1 \leq i \leq 5$, consists of the columns of the d_1-block G_{B_i}, while the set S_R consists of the remaining columns of G_0. The subset of columns in each set that are columns in A_0 are:

$$
\begin{aligned}
S_{B_1} &\supset \{1,2,3\} \\
S_{B_2} &\supset \{4,5,6\} \\
S_{B_3} &\supset \{7,8,9\} \\
S_{B_4} &\supset \{10,11\} \\
S_{B_5} &\supset \{12\} \\
S_R &\supset \{13\}
\end{aligned}
$$

Let $\vec{v}$ be any extension vector. Then the columns of $\vec{v}$ partition into six disjoint sets $v\vec{B}_1, v\vec{B}_2, v\vec{B}_3, v\vec{B}_4, v\vec{B}_5, v\vec{R}$, where $v\vec{B}_i$ consists of the columns of $\vec{v}$ corresponding to the subset of columns in S_{B_i} that are also columns of A_0 and $v\vec{R}$ consists of the remaining columns of $\vec{v}$. For example, consider the following extension vector:

$$\vec{v} = (1\,0\,0 | 0\,1\,0 | 0\,0\,1 | 1\,0 | 1 | 1)$$

For this vector $v\vec{B}_1 = (1\,0\,0)$, $v\vec{B}_2 = (0\,1\,0)$, $v\vec{B}_3 = (0\,0\,1)$, $v\vec{B}_4 = (1\,0)$, $v\vec{B}_5 = (1)$, and $v\vec{R} = (1)$.

Let $\vec{v_B}$ be an extension of a generator block G_B that is a d_1-block. If G_B is a $d_{1,0}$-block then there are $2^3 = 8$ possibilities for $\vec{v_B}$, if G_B is a $d_{1,3}$-block then there are $2^2 = 4$ possibilities for $\vec{v_B}$, and if G_B is a $d_{1,4}$-block then there are $2^1 = 2$ possibilities for $\vec{v_B}$. Let $\vec{v'}$ and $\vec{v''}$ be two extension vectors that differ only in the columns of the extension $\vec{v_B}$ of the d_1-block G_B. Then, as was pointed out in Section 4.5 of Chapter 4, for some of the different possibilities for $\vec{v_B}$, the codes generated by $[I_k|\mathcal{A}(A_0, \vec{v'})]$ and $[I_k|\mathcal{A}(A_0, \vec{v''})]$ will always be equivalent, no matter what the remaining columns of $\vec{v'}$ and $\vec{v''}$ are.

Example 5.13 Let $G_0 = [I_k|A_0]$ be a generator matrix for a $(2k, k, 4)$ d_1-code C_0, and let G_B be a generator block in G_0 that is a $d_{1,0}$-block. Without loss of generality, assume that the $d_{1,0}$-block is in the first row of A_0 and that the columns of the $d_{1,0}$-block are in the first three columns of A_0. Then:

$$A_0 = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0^* \\ \hline 1 & 1 & 0 & X \\ 1 & 0 & 1 & Y \\ 0 & 1 & 1 & Z \\ 0 & 0 & 0 & W \end{array}\right]$$

The rows below the horizontal line represent all the different ways the remaining rows of A_0 may intersect the $d_{1,0}$-block.

Let $\vec{v'} = \vec{v'_B}\vec{v'_R}$ be an extension vector for A_0, where $\vec{v'_B} = (1\,0\,0)$ is an extension of the $d_{1,0}$-block. Consider the matrix:

$$\mathcal{A}(A_0, \vec{v'}) = \left[\begin{array}{cc||ccc|c} 1 & 0 & 1 & 0 & 0 & \vec{v'_R} \\ 0 & 1 & 1 & 0 & 0 & \vec{v'_R} \\ \hline\hline 1 & 1 & 1 & 1 & 1 & 0^* \\ \hline & & 1 & 1 & 0 & X \\ & F' & 1 & 0 & 1 & Y \\ & & 0 & 1 & 1 & Z \\ & & 0 & 0 & 0 & W \end{array}\right]$$

If we perform an $\mathrm{ADD}_{r,c}(3,3)$ on $\mathcal{A}(A_0, \vec{v'})$ (i.e. an $\mathrm{ADD}_{r,c}$ on the boldface entry) followed by a $\mathrm{SWAP}_r(1,2)$ and $\mathrm{SWAP}_c(4,5)$ then the resulting matrix is:

$$\mathcal{A}(A_0, \vec{v''}) = \left[\begin{array}{cc||ccc|c} 1 & 0 & 1 & 1 & 1 & \vec{v'_R} \\ 0 & 1 & 1 & 1 & 1 & \vec{v'_R} \\ \hline\hline 1 & 1 & 1 & 1 & 1 & \mathbf{0^*} \\ \hline & & 1 & 1 & 0 & X \\ & F'' & 1 & 0 & 1 & Y \\ & & 0 & 1 & 1 & Z \\ & & 0 & 0 & 0 & W \end{array}\right]$$

where $\vec{v''} = \vec{v''_B}\vec{v'_R}$, where $\vec{v''_B} = (1\,1\,1)$. What this tells us is that no matter what the vector $\vec{v'_R}$ is, the two extensions $\vec{v'_B} = (1\,0\,0)$ and $\vec{v''_B} = (1\,1\,1)$ of a $d_{1,0}$-block will always result in matrices that generate equivalent codes.

Such extensions of the generator block G_B are referred to as equivalent extensions of the generator block G_B. In Section 4.5 of Chapter 4 we found all the equivalent extensions of a $d_{2,0}$-block using the method of Example 5.13. Using the same method, we can find all the equivalent extensions of the d_1-blocks. For the $d_{1,0}$-block, the odd weight extensions $1\,1\,1$, $1\,0\,0$, $0\,1\,0$, and $0\,0\,1$ are all equivalent extensions. Similarly, for the $d_{1,3}$-block, the odd weight extensions $1\,0$ and $0\,1$ are equivalent extensions. For the $d_{1,4}$-block, there is only one odd weight extension.

Therefore, in order to find all inequivalent $(2k+4, k+2, 4)$ self-dual codes generated by a matrix of the form $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$, where $G_0 = [I_k|A_0]$ generates a $(2k, k, 4)$ d_1-code, we only need to consider the extension vectors $\vec{v}$ in which the extensions of each $d_{1,0}$-block, $d_{1,3}$-block, and $d_{1,4}$-block have either even weight or are any one of the odd weight extensions. If $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$ generates a $(2k+4, k+2, 4)$ self-dual code that contains one and only one weight 4 word then, as we shall see in the next subsection, we in fact only need to consider the extension vectors $\vec{v}$ in which the extensions of each $d_{1,0}$,

$d_{1,3}$, and $d_{1,4}$-block have odd weight. In other words, we can fix the extensions of each d_1-block in $\vec{v}$.

5.3.2 Extensions of the d_1-Codes

Let $G_0 = [I_k|A_0]$ be a generator matrix for a $(2k, k, 4)$ self-dual code C_0 that does not contain any intersecting weight 4 words. In this subsection, we will find all the extension vectors $\vec{v}$ that result in a matrix $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$ that generates a $(2k+4, k+2, 4)$ self-dual code that contains one and only one weight 4 word (i.e. the weight 4 word that is generated by the first two rows of G).

Let C by any $(2k+4, k+2, 4)$ self-dual code generated by:

$$G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})] = \begin{array}{c} \begin{array}{cc} a_1 & a_2 \end{array} \quad \begin{array}{ccccc} & & & & \end{array} \quad \begin{array}{cc} c_1 & c_2 \end{array} \quad \begin{array}{c} \\ \\ \end{array} \\ \left[\begin{array}{cc|ccccc||cc|c} 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \vec{v} \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 1 & \vec{v} \\ \hline 0 & 0 & & & & & & & \\ 0 & 0 & & & & & & & \\ \vdots & \vdots & & & I_k & & & F & A_0 \\ 0 & 0 & & & & & & & \end{array}\right] \end{array} \begin{array}{c} \vec{w_1} \\ \vec{w_2} \end{array}$$

where $G_0 = [I_k|A_0]$ generates a $(2k, k, 4)$ self-dual code C_0 containing no intersecting weight 4 words. In order to find all the extension vectors $\vec{v}$ that result in C containing one and only one weight 4 word, we will first partition the codewords of C into two disjoint sets, and then for each of these sets we will find all the different extension vectors that result in C that contains at least one weight 4 word (other than the weight 4 word already mentioned).

Let $\vec{w_1}$ and $\vec{w_2}$ denote the codewords in the first two rows of G. Partition the codewords of C into the following two disjoint sets:

- T_A = the codewords generated by the last k rows of G.
- $T_B = (T_A \oplus \vec{w_1}) \cup (T_A \oplus \vec{w_2}) \cup (T_A \oplus (\vec{w_1} \oplus \vec{w_2}))$.

We will now find all the different extension vectors $\vec{v}$ that result in either T_A or T_B containing a weight 4 word (other than the weight 4 word $\vec{w_1} \oplus \vec{w_2} \in T_B$). From this, we will be able to determine all the extension vectors $\vec{v}$ that result in both T_A and $T_B - \{\vec{w_1} \oplus \vec{w_2}\}$ containing no weight 4 words, and thus, all the extension vectors that result in G being a generator matrix for a $(2k+4, k+2, 4)$ self-dual code C that contains one and only one weight 4 word.

Let us first consider the set of codewords T_A. The codewords of T_A are the codewords of the $(2k, k, 4)$ d_1-code C_0 with the addition of four extra components, a_1, a_2, c_1, c_2. The codewords of T_A are all equal to 0 0 in components a_1, a_2. The codewords of T_A are either equal to 0 0 or 1 1 in components c_1, c_2. Therefore, for any weight w word $\vec{c_0}$ in C_0, the corresponding codeword $\vec{c}$ in T_A will either have weight w or weight $w+2$. Since every codeword in C must intersect both the codewords $\vec{w_1}$ and $\vec{w_2}$ in an even number of positions, if the last k columns of $\vec{c_0}$ intersect the extension vector $\vec{v}$ in an even number of positions then $\vec{c}$ will have weight w, else $\vec{c}$ will have weight $w+2$. Since C_0 has distance 4, a codeword $\vec{c}$ in T_A will have weight 4 if and only if the corresponding codeword $\vec{c_0}$ in C_0 has weight 4 and the last k columns of $\vec{c_0}$ intersect $\vec{v}$ in an even number of positions. Therefore, T_A will *not* contain any weight 4 words if and only if the extension vector $\vec{v}$ intersects the last k columns of every weight 4 word in C_0 in an odd number of positions.

Suppose C_0 contains exactly m weight 4 words. Then, as we have already seen, the rows of G_0 can be partitioned into $m+1$ disjoint sets $G_{B_1}, G_{B_2}, \ldots, G_{B_m}, G_R$ where each G_{B_i} is a d_1-block and G_R is the set of the remaining rows of G_0. Furthermore, the columns of $\vec{v}$ can be partitioned into $m+1$ disjoint sets $v_{\vec{B}_1}, v_{\vec{B}_2}, \ldots, v_{\vec{B}_m}, v_{\vec{R}}$ where $v_{\vec{B}_i}$ is an extension of the d_1-block G_{B_i} and $v_{\vec{R}}$ is the set of the remaining columns of $\vec{v}$. Let $\vec{c_i}'$ be the weight 4 word in C_0 generated by G_{B_i} and let $\vec{c_i}$ be the corresponding codeword of C in the set T_A. Then, as we've just seen, $\vec{c_i}$ will not have weight 4 if and only if the last k columns of $\vec{c_i}'$ intersect $\vec{v}$ in an odd number of positions. In other words, since the only columns of $\vec{v}$ that $\vec{c_i}'$ intersects are the columns of the extension $v_{\vec{B}_i}$ of the d_1-block G_{B_i}, the codeword $\vec{c_i}$ will not have weight 4 if and only if the extension $v_{\vec{B}_i}$ has odd

weight. Therefore, the set of codewords T_A in the $(2k+4, k+2, 4)$ self-dual code C will not contain any weight 4 words if and only if all the extensions $v\vec{B}_i$ have odd weight, $1 \leq i \leq m$.

Example 5.14 Let C_0 be the $(26, 13, 4)$ d_1-code of Example 5.12 which is generated by $G_0 = [I_k | A_0]$, where:

$$A_0 = \left[\begin{array}{ccc|ccc|ccc|cc|cc|l}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_1 \\ \hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_2 \\ \hline
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & r_3 \\ \hline
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & r_4 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & r_5 \\ \hline
1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & r_6 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & r_7 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & r_8 \\ \hline
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & r_9 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & r_{10} \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & r_{11} \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & r_{12} \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & r_{13}
\end{array}\right]$$

C_0 contains exactly five weight 4 words:

$$\begin{aligned}
\vec{c_1'} &= r_1 \\
\vec{c_2'} &= r_2 \\
\vec{c_3'} &= r_3 \\
\vec{c_4'} &= r_4 \oplus r_5 \\
\vec{c_5'} &= r_6 \oplus r_7 \oplus r_8
\end{aligned}$$

each of which are generated by d_1-blocks. Let $\vec{v} = (1\,0\,0 \,|\, 0\,1\,0 \,|\, 0\,0\,1 \,|\, 1\,0 \,|\, 1 \,|\, 1)$. The five extensions of the d_1-blocks in $\vec{v}$ are $v\vec{B}_1 = (1\,0\,0)$, $v\vec{B}_2 = (0\,1\,0)$, $v\vec{B}_3 = (0\,0\,1)$, $v\vec{B}_4 = (1\,0)$,

and $v\vec{B}_5 = (1)$. Notice that each of these extensions has odd weight. Consider the $(30, 15, 4)$ self-dual code C, generated by $G = [I_{15}, \mathcal{A}(A_0, \vec{v})]$, where:

$$\mathcal{A}(A_0, \vec{v}) = \left[\begin{array}{cc|ccc|ccc|ccc|cc|cc}
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
\hline
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\hline
1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
\hline
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1
\end{array}\right]$$

Let T_A be the set of codewords generated by the last k rows of G. Each of the five codewords $\vec{c_1}$, $\vec{c_2}$, $\vec{c_3}$, $\vec{c_4}$, and $\vec{c_5}$ in T_A that correspond to the five weight 4 words in C_0 have weight 6. That is, the three codewords $\vec{c_1}$, $\vec{c_2}$, and $\vec{c_3}$ in rows 3, 4, and 5 of G, respectively, are weight 6 rows, the codeword $\vec{c_4}$ generated by rows 6 and 7 of G has weight 6, and the codeword $\vec{c_5}$ generated by rows 8, 9, and 10 of G has weight 6.

We have now shown that the subset of codewords T_A of the $(2k+4, k+2, 4)$ self-dual code C, generated by $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$, does not contain any weight 4 words if and only if for each $v\vec{B}$ in $\vec{v}$, where $v\vec{B}$ is an extension of a d_1-block in $G_0 = [I_k|A_0]$, the extension $v\vec{B}$ has odd weight.

Let us now determine the extension vectors $\vec{v}$ which result in the set of codewords

$T_B = (T_A \oplus \vec{w_1}) \cup (T_A \oplus \vec{w_2}) \cup (T_A \oplus (\vec{w_1} \oplus \vec{w_2}))$ (where $\vec{w_1}$ and $\vec{w_2}$ are the first two rows of G) containing weight 4 words (other than the weight 4 word $\vec{w_1} \oplus \vec{w_2}$). To do this, we will find all the different ways in which a set of rows of G, that includes at least one of $\vec{w_1}$ and $\vec{w_2}$, can sum to a weight 4 word.

For the case of one row of G being a weight 4 word that is an element of T_B, the row must either be $\vec{w_1}$ or $\vec{w_2}$. This can only happen if $w(\vec{v}) = 2$. Therefore, T_B will contain a weight 4 word that is a row of G if and only if $w(\vec{v}) = 2$.

Suppose two rows of G sum to a weight 4 word $\vec{c}$ that is an element of $T_B - \{\vec{w_1} \oplus \vec{w_2}\}$. Then, $\vec{c}$ must be the sum of $\vec{w_1}$ (or $\vec{w_2}$) with a row $\vec{w_3}$ from the last k rows of G. Now, the only way two rows of G can sum to a weight 4 word is if the columns in $\mathcal{A}(A_0, \vec{v})$ of these two rows are:

$$\begin{array}{ccc} 1 & 0 & x \\ 0 & 1 & x \end{array}$$

Using the facts that the rows of $\mathcal{A}(A_0, \vec{v})$ have odd weight and intersect evenly, and the first two columns in $\mathcal{A}(A_0, \vec{v})$ of the row $\vec{w_3}$ must be either 0 0 or 1 1, this gives us two different possibilities in which $\vec{w_1}$ (or $\vec{w_2}$) and a row $\vec{w_3}$ from the last k rows of G can sum to a weight 4 word. With just the columns of $\mathcal{A}(A_0, \vec{v})$ given, these two possibilities are:

1	0	1	1^*	0^*	$\vec{w_1}$
0	1	1	1^*	0^*	$\vec{w_2}$
1	1	0	1^*	0^*	$\vec{w_3}$

1	0	0	1^*	0^*	$\vec{w_1}$
0	1	0	1^*	0^*	$\vec{w_2}$
0	0	1	1^*	0^*	$\vec{w_3}$

In both cases, both $\vec{w_1} \oplus \vec{w_3}$ and $\vec{w_2} \oplus \vec{w_3}$ sum to a weight 4 word. Notice that one or the other of these two cases will occur if and only if the extension vector $\vec{v}$ differs from a row in the matrix A_0 in exactly one position. Therefore, $T_B - \{\vec{w_1} \oplus \vec{w_2}\}$ will contain a weight 4 word that is the sum of two rows of G if and only if $\vec{v}$ differs from a row in A_0 in one and only one position.

Suppose three rows of G sum to a weight 4 word $\vec{c}$ that is an element of T_B. Then $\vec{c}$ is either the sum of $\vec{w_1}$ and $\vec{w_2}$ and one row $\vec{w_3}$ from the last k rows of G, or $\vec{c}$ is the

sum of $\vec{w_1}$ (or $\vec{w_2}$) and two rows $\vec{w_3}$ and $\vec{w_4}$ from the last k rows of G. Now, the only way three rows of G can sum to a weight 4 word is if the columns in $\mathcal{A}(A_0, \vec{v})$ of these three rows are:

$$\begin{matrix} 1 & x \\ 1 & y \\ 1 & x \oplus y \end{matrix}$$

Using the facts that the rows of $\mathcal{A}(A_0, \vec{v})$ have odd weight and intersect evenly, and that the first two columns of any row in the last k rows of $\mathcal{A}(A_0, \vec{v})$ are either 0 0 or 1 1, we see that there are two different possibilities in which three rows of G can sum to a weight 4 word that is an element of T_B. With just the columns of $\mathcal{A}(A_0, \vec{v})$ given, these two possibilities are:

1	0	1	1^*	0^*	$\vec{w_1}$
0	1	1	1^*	0^*	$\vec{w_2}$
1	1	1	0^*	0^*	$\vec{w_3}$

1	0	1^*	1^*	0^*	0^*	$\vec{w_1}$
0	1	1^*	1^*	0^*	0^*	$\vec{w_2}$
1	1	1^*	0^*	1^*	0^*	$\vec{w_3}$
1	1	0^*	1^*	1^*	0^*	$\vec{w_4}$

In the first case, $\vec{w_1} \oplus \vec{w_2} \oplus \vec{w_3}$ sum to a weight 4 word. However, the first case cannot occur since the row $\vec{w_3}$ implies that the $(2k, k, 4)$ self-dual code C_0 has distance 2, which is a contradiction. In the second case, both $\vec{w_1} \oplus \vec{w_3} \oplus \vec{w_4}$ and $\vec{w_2} \oplus \vec{w_3} \oplus \vec{w_4}$ sum to a weight 4 word. This case occurs if and only if two rows of A_0 sum to the extension vector $\vec{v}$. Therefore, T_B will contain a weight 4 word that is the sum of three rows of G if and only if A_0 contains two rows that sum to the extension vector $\vec{v}$.

Summarizing what we have just found, the set of codewords T_B in the $(2k+4, k+2, 4)$ self-dual code C, generated by $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$, will contain weight 4 words (other than $\vec{w_1} \oplus \vec{w_2}$) if and only if any one of the following three statements about the extension vector $\vec{v}$ is true: $w(\vec{v}) = 2$, $\vec{v}$ differs from a row in A_0 in one and only one position, or $\vec{v}$ is the sum of two rows in A_0. Thus, the set of codewords $T_B - \{\vec{w_1} \oplus \vec{w_2}\}$ will *not* contain any weight 4 words if and only if the following three statements about the extension vector

are all true: $w(\vec{v}) \geq 4$, $\vec{v}$ differs from every row in A_0 in at least two positions, and $\vec{v}$ is not the sum of any pair of rows in A_0.

We have now found all the conditions which the extension vector $\vec{v}$ must satisfy in order for the $(2k+4, k+2, 4)$ self-dual code C to contain one and only one one weight 4 word. That is, if $G_0 = [I_k|A_0]$ is a generator matrix for a d_1-code, then any extension vector $\vec{v}$ results in a matrix $G_0 = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$ that generates a $(2k+4, k+2, 4)$ self-dual code C containing one and only one weight 4 word if and only if:

- for each d_1-block in G_0, the portion of $\vec{v}$ that is the extension $v\vec{_B}$ of the d_1-block has odd weight,
- $w(\vec{v}) \geq 4$,
- $\vec{v}$ differs from every row in A_0 in at least two positions, and,
- no two rows of A_0 sum to $\vec{v}$.

As was stated earlier, the odd weight extensions of each of the d_1-blocks are equivalent extensions. Therefore, if $\vec{v} = v\vec{_{B_1}} v\vec{_{B_2}} \ldots v\vec{_{B_m}} v\vec{_R}$, where $v\vec{_{B_i}}$ is the extension of a d_1-block G_{B_i} in G_0, then we can fix the columns of each $v\vec{_{B_i}}$ to one of the odd weight extensions. For example, if G_{B_i} is a $d_{1,0}$-block, we can fix $v\vec{_{B_i}} = (1\,0\,0)$, if G_{B_i} is a $d_{1,3}$-block, we can fix $v\vec{_{B_i}} = (1\,0)$, and if G_{B_i} is a $d_{1,4}$-block, we can fix $v\vec{_{B_i}} = (1)$. Thus, we only have to consider all the different vectors $v\vec{_R}$ that result in an extension vector $\vec{v}$ that satisfies the last three conditions.

Let us now summarize what we have just found in terms of our algorithm for enumerating the $(2k, k, d \geq 6)$ self-dual codes. Let $G_0 = [I_k|A_0]$ be the generator matrix for a $(2k, k, 4)$ self-dual code C_0 that does not contain any intersecting weight 4 words (i.e. C_0 is a d_1-code). Let $G_{B_1}, G_{B_2}, \ldots, G_{B_m}$ denote the d_1-blocks in G_0. Then we can produce a set of vectors $\vec{v}$ with the property that for any $(2k+4, k+2, 4)$ self-dual code C that is an extension of C_0 (where C contains one and only one weight 4 word) at least one

vector $\vec{v}$ is produced in which $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$ generates a code equivalent to C, as follows:

- For each d_1-block G_B in G_0, fix the extension $v\vec{}_{B_i}$ of the d_1-block as follows:
  - if G_B is a $d_{1,0}$-block, fix $v\vec{}_{B_i} = (1\,0\,0)$,
  - if G_B is a $d_{1,3}$-block, fix $v\vec{}_{B_i} = (1\,0)$, and
  - if G_B is a $d_{1,4}$-block, fix $v\vec{}_{B_i} = (1)$.
- Our set of extensions vector $\vec{v}$ is found by producing all vectors $v\vec{}_R$ that result in a vector $\vec{v} = v\vec{}_{B_1} v\vec{}_{B_2} \ldots v\vec{}_{B_m} v\vec{}_R$ with the following properties:
  - $w(\vec{v}) \geq 4$,
  - $w(\vec{v}) \equiv 0 \bmod 2$ (since $\vec{v}$ is an extension vector),
  - $\vec{v}$ differs from every row in A_0 in at least two positions, and
  - $\vec{v}$ is not the sum of any two rows of A_0.

For each vector $\vec{v}$ we produce, we can use $\vec{v}$ and A_0 to find generator matrices for the two related codes C_r and C_s. This is the subject of the next subsection.

5.3.3 Finding The Two Related Codes

Let $G_0 = [I_k|A_0]$ be a generator matrix for a $(2k, k, 4)$ d_1-code C_0. Let $\vec{v}$ be an extension vector that extends C_0 to a $(2k+4, k+2, 4)$ self-dual code C that contains one and only one weight 4 word. Then, given A_0 and $\vec{v}$, we can use the functions we developed in Section 5.1 to find generator matrices for the two related codes C_r and C_s. That is, a generator matrix for C_r is $G_r = [I_k|\mathcal{R}(A_0, \vec{v})]$, and a generator matrix for C_s is $G_s = [I_k|\mathcal{S}(A_0, \vec{v}, \vec{r}, c)]$, where $\vec{r}$ is any row in A_0 such that $\vec{v} \bullet \vec{r} \equiv 1 \bmod 2$ and c is any column in which $\vec{v}$ has a value of 1. In this subsection, we will show how we can find a generator matrix for a code equivalent to C_s by simply altering the vector $\vec{v}$ to produce

a vector $\vec{v'}$ that results in a matrix $G'_s = [I_k|\mathcal{R}(A_0, \vec{v'})]$ that generates a code C'_s that is equivalent to C_s. The reason we will do this is that the function $\mathcal{R}(A_0, \vec{v'})$ is a much simpler function than the function $\mathcal{S}(A_0, \vec{v}, \vec{r}, c)$.

Let us begin by finding a canonical A_0 for our input matrices. Recall, by Theorem 2.18 of Chapter 2, that for any $(2k, k, 4)$ self-dual code C there exists an equivalent code generated by a matrix of the form $G = [I_k|A]$, where G contains a weight 4 row. Therefore, for any $(2k, k, 4)$ d_1-code, there exists an equivalent code C_0, generated by $G_0 = [I_k|A_0]$, such that the first row of G_0 is a weight 4 row (i.e. a $d_{1,0}$-block). Let $\vec{v} = 1\,0\,0\,\vec{v_0}$ be our extension vector, where 1 0 0 is the extension of the $d_{1,0}$-block, and $\vec{v_0}$ consists of the remaining columns of $\vec{v}$. Then:

$$A_0 = \left[\begin{array}{c|cc|c} 1 & 1 & 1 & 0^* \\ \hline 1 & 1 & 0 & X_1 \\ 1 & 0 & 1 & X_2 \\ \hline 0 & 1 & 1 & X_3 \\ 0 & 0 & 0 & X_4 \\ \hline 1 & 1 & 0 & X_5 \\ 1 & 0 & 1 & X_6 \\ \hline 0 & 1 & 1 & X_7 \\ 0 & 0 & 0 & X_8 \end{array}\right] \begin{array}{l} \\ \text{for any } \vec{x_1} \in X_1, \quad \vec{x_1} \bullet \vec{v_0} \equiv 0 \bmod 2 \\ \text{for any } \vec{x_2} \in X_2, \quad \vec{x_2} \bullet \vec{v_0} \equiv 0 \bmod 2 \\ \text{for any } \vec{x_3} \in X_3, \quad \vec{x_3} \bullet \vec{v_0} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_4} \in X_4, \quad \vec{x_4} \bullet \vec{v_0} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_5} \in X_5, \quad \vec{x_5} \bullet \vec{v_0} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_6} \in X_6, \quad \vec{x_6} \bullet \vec{v_0} \equiv 1 \bmod 2 \\ \text{for any } \vec{x_7} \in X_7, \quad \vec{x_7} \bullet \vec{v_0} \equiv 0 \bmod 2 \\ \text{for any } \vec{x_8} \in X_8, \quad \vec{x_8} \bullet \vec{v_0} \equiv 0 \bmod 2 \end{array}$$

The last eight rows of A_0 represent all the different ways that a row of A_0 may intersect both the $d_{1,0}$-block in the first row of A_0 and the extension vector $\vec{v} = 1\,0\,0\,\vec{v_0}$.

The first related code C_r is generated by the matrix $G_r = [I_k|\mathcal{R}(A_0, 1\,0\,0\,\vec{v_0})]$. The second related code C_s is generated by the matrix $G_s = [I_k|\mathcal{S}(A_0, 1\,0\,0\,\vec{v_0}, \vec{r}, c)]$ (where $\vec{r}$ is the first row in A_0 and c is the first column), where:

$$\mathcal{S}(A_0, 1\,0\,0\,\vec{v_0}, \vec{r}, c) = \left[\begin{array}{c|cc|c} 0 & 0 & 0 & \vec{v_0} \\ \hline 1 & 0 & 1 & X_1 \\ 1 & 1 & 0 & X_2 \\ \hline 1 & 0 & 0 & \vec{v_0} + X_3 \\ 1 & 1 & 1 & \vec{v_0} + X_4 \\ \hline 0 & 1 & 0 & \vec{v_0} + X_5 \\ 0 & 0 & 1 & \vec{v_0} + X_6 \\ \hline 0 & 1 & 1 & X_7 \\ 0 & 0 & 0 & X_8 \end{array}\right]$$

Consider the extension vector $\vec{v'} = 1\,1\,1\,\vec{v_0}$. That is, consider the extension vector $\vec{v'}$ that is equal to $\vec{v}$ with the exception that the extension of the d_1-block in $\vec{v'}$ is 1 1 1 instead of 1 0 0. Then $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$ and $G' = [I_{k+2}|\mathcal{A}(A_0, \vec{v'})]$ generate equivalent codes. Consider the code C'_s generated by $G'_s = [I_k|\mathcal{R}(A_0, 1\,1\,1\,\vec{v_0})]$, where:

$$\mathcal{R}(A_0, 1\,1\,1\,\vec{v_0}) = \left[\begin{array}{c|cc|c} 0 & 0 & 0 & \vec{v_0} \\ \hline 1 & 1 & 0 & X_1 \\ 1 & 0 & 1 & X_2 \\ \hline 1 & 0 & 0 & \vec{v_0} + X_3 \\ 1 & 1 & 1 & \vec{v_0} + X_4 \\ \hline 0 & 0 & 1 & \vec{v_0} + X_5 \\ 0 & 1 & 0 & \vec{v_0} + X_6 \\ \hline 0 & 1 & 1 & X_7 \\ 0 & 0 & 0 & X_8 \end{array}\right]$$

Notice that if we swap the second and third columns of $\mathcal{R}(A_0, 1\,1\,1\,\vec{v_0})$ we produce the matrix $\mathcal{S}(A_0, 1\,0\,0\,\vec{v_0}, \vec{r}, c)$. Therefore, the second related code C_s is equivalent to the code C'_s.

What we have just shown is that given our canonical A_0 and $\vec{v} = 1\,0\,0\,\vec{v_0}$, $G_r = [I_k|\mathcal{R}(A_0, 1\,0\,0\,\vec{v_0})]$ generates the related code C_r and $G'_s = [I_k|\mathcal{R}(A_0, 1\,1\,1\,\vec{v_0})]$ generates a code equivalent to the related code C_s. In other words, we can find a generator matrix for a code equivalent to the related code C_s by simply altering the extension of one of the $d_{1,0}$-blocks in G_0.

5.3.4 Enumeration of the $(2k, k, d \geq 6)$ self-dual codes

We are now ready to give a formal description of our algorithm for producing a list L of $(2k, k, d \geq 6)$ self-dual codes with the property that for any $(2k, k, d \geq 6)$ self-dual code C_1, there exists at least one code in L equivalent to C_1. The algorithm uses the *QuickOrder* algorithm of Chapter 3 to reduce the number of equivalent codes produced.

Algorithm 5.15 :

- Input: a list L_0 of $k \times k$ matrices A_0, where $G_0 = [I_k|A_0]$ generates a $(2k, k, 4)$ d_1-code, such that for any $(2k, k, 4)$ d_1-code C_0 there exists one and only one matrix A_0 in L_0 such that $G_0 = [I_k|A_0]$ generates a code equivalent to C_0. Without loss of generality, it is assumed each G_0 contains at least one $d_{1,0}$-block.

- Output: a list L_1 of $k \times k$ matrices A_1, where $G_1 = [I_k|A_1]$ generates a $(2k, k, d \geq 6)$ self-dual code, such that for any $(2k, k, d \geq 6)$ self-dual code C_1 there exists at least one matrix A_1 in L_1 such that $G_1 = [I_k|A_1]$ generates a code equivalent to C_1.

begin

for each $A_0 \in L_0$ **do**

Let $G_{B_1}, G_{B_2}, \ldots, G_{B_m}$ be the d_1-blocks in $G_0 = [I_k|A_0]$.

Let $\vec{v}$ be an extension vector with columns $v_{\vec{B}_1}, v_{\vec{B}_2}, \ldots, v_{\vec{B}_m}, v_{\vec{R}}$, where $v_{\vec{B}_i}$ is an extension of G_{B_i} and $v_{\vec{R}}$ consist of the remaining columns of $\vec{v}$.

Fix the columns of each $v_{\vec{B}_i}$ as follows:

if G_{B_i} is a $d_{1,0}$-block, $v_{\vec{B}_i} = 1\,0\,0$,

if G_{B_i} is a $d_{1,3}$-block, $v_{\vec{B}_i} = 1\,0$, and

if G_{B_i} is a $d_{i,4}$-block, $v_{\vec{B}_i} = 1$.

for each $v_{\vec{R}}$ that results in a $\vec{v}$ such that $w(\vec{v}) \equiv 0 \bmod 2$, $w(\vec{v}) \geq 4$, $\vec{v}$ differs from every row in A_0 in at least two columns, and $\vec{v}$ is not the sum of any pair of rows in A_0 **do**

Let $A_r = \mathcal{R}(A_0, \vec{v})$.

Pick any G_{B_i} that is a $d_{1,0}$-block.

Let $\vec{v'}$ be $\vec{v}$ with $v_{\vec{B}_i}$ changed to $1\,1\,1$.

Let $A_s = \mathcal{R}(A_0, \vec{v'})$.

Let d_r be the distance of the code generated by $G_r = [I_k|A_r]$, and let d_s be the distance of the code generated by $G_s = [I_k|A_s]$.

if $d_r > d_s \geq 4$ **then**

QuickOrderMatrix$(A_r, 0, A_1)$.

if $A_1 \notin L_1$ **then** insert A_1 into L_1.

end if

else if $d_s > d_r \geq 4$ **then**

QuickOrderMatrix$(A_s, 0, A_1)$.

if $A_1 \notin L_1$ **then** insert A_1 into L_1.

end else if

end for

end for

end

5.4 Further Improvements

In this section, we will look at ways in which we can reduce both the number of equivalent $(2k, k, d \geq 6)$ self-dual codes produced and the number of $(2k, k, 4)$ d_1-codes input into Algorithm 5.15. We will do this by considering the extension mates of codes that are equivalent to a given $(2k, k, d \geq 6)$ self-dual code.

Let C_1 be any $(2k, k, d \geq 6)$ self-dual code. Let $\vec{w_1}$ be any weight d word in C_1. Then, by Theorem 2.18 of Chapter 2, there exists an equivalent code C_1', generated by $G_1' = [I_k | A_1']$, where

$$A_1' = \left[\begin{array}{c|c|c|c} 0 & 1^{d-3} & 1\ 1 & 0^* \\ \hline 1 & W_0 & W_1 & W_2 \\ 0 & X_0 & X_1 & X_2 \\ \hline 1 & Y_0 & Y_1 & Y_2 \\ 0 & Z_0 & Z_1 & Z_2 \end{array}\right] \begin{array}{l} \\ \\ \text{for any } w \in W_0,\ w(w) \equiv 0 \bmod 2 \\ \text{for any } x \in X_0,\ \ w(x) \equiv 1 \bmod 2 \\ \text{for any } y \in Y_0,\ \ w(y) \equiv 1 \bmod 2 \\ \text{for any } z \in Z_0,\ \ w(z) \equiv 0 \bmod 2 \end{array}$$

where the first row of G_1' is the weight d word $\vec{w_1'}$ of C_1' that corresponds to the weight d word $\vec{w_1}$ of C_1. The extension mate of C_1' is the $(2k, k, 4)$ d_1-code C_0', generated by $G_0' = [I_k|\mathcal{R}(A_1', \vec{v'})]$, where $\vec{v'} = 1\ 1^{d-3}\ 0\ 0\ 0^*$ and:

$$\mathcal{R}(A_1', \vec{v'}) = \left[\begin{array}{c|c|c|c} 1 & 0^{d-3} & 1\ 1 & 0^* \\ \hline 0 & \overline{W_0} & W_1 & W_2 \\ 1 & \overline{X_0} & X_1 & X_2 \\ \hline 1 & Y_0 & Y_1 & Y_2 \\ 0 & Z_0 & Z_1 & Z_2 \end{array}\right]$$

In this section we will use the term "extension mate" to refer to the $(2k, k, 4)$ d_1-code C_0' only, and not to the codes that are equivalent to C_0'. With this convention, only a subset of the codes equivalent to the $(2k, k, d \geq 6)$ self-dual code C_1 have extension mates. That is, the only codes equivalent to C_1 that have extension mates are the codes generated by matrices with the same structure as G_1'.

Notice that the extension mate C_0' is dependent on both the weight d word we select to be $\vec{w'}$, and the columns of G_1' we select to be the columns of A_1' that intersect the extension vector $\vec{v'}$. Therefore, for any two equivalent $(2k, k, d \geq 6)$ self-dual codes C_1'' and C_1''', where $C_1'' \neq C_1'''$ and C_1'' and C_1''' are both generated by matrices with the same structure as $G_1' = [I_k|A_1']$, their extension mates, C_0'' and C_0''', respectively, may not necessarily be equal. Furthermore, C_0'' and C_0'''' may not even be equivalent. In other words, there may exist many inequivalent $(2k, k, 4)$ d_1-codes that are extension mates of codes equivalent to the $(2k, k, d \geq 6)$ self-dual code C_1.

Example 5.16 Let C_1' and C_1'' be the equivalent $(24, 12, 6)$ self-dual codes generated by

$G'_1 = [I_{12}|A'_1]$ and $G''_1 = [I_{12}|A''_1]$, respectively, where:

$$A'_1 = \begin{bmatrix} 0&1&1&1&1&1&0&0&0&0&0&0 \\ 1&1&1&0&0&0&1&1&0&0&0&0 \\ 1&1&0&0&1&0&0&0&1&1&0&0 \\ 1&0&1&0&0&1&0&0&1&0&1&0 \\ 1&0&0&0&1&1&1&0&0&0&0&1 \\ 1&0&0&0&0&0&0&1&0&1&1&1 \\ 0&1&0&1&0&0&1&0&0&1&0&1 \\ 0&1&0&0&0&1&0&1&1&0&0&1 \\ 0&0&1&1&0&0&0&1&1&1&0&0 \\ 0&0&1&0&1&0&1&0&0&1&1&0 \\ 0&0&0&1&1&0&0&0&1&0&1&1 \\ 0&0&0&1&0&1&1&1&0&0&1&0 \end{bmatrix} \quad A''_1 = \begin{bmatrix} 0&1&1&1&1&1&0&0&0&0&0&0 \\ 1&0&1&0&0&1&1&1&0&0&0&0 \\ 1&0&0&0&1&1&0&0&1&1&0&0 \\ 1&1&1&0&0&0&0&0&1&0&1&0 \\ 0&0&1&0&1&0&1&0&1&0&0&1 \\ 0&0&1&1&0&0&0&1&0&0&1&1 \\ 0&0&0&1&0&1&1&0&0&1&0&1 \\ 0&1&0&0&0&1&0&1&1&0&0&1 \\ 1&1&0&1&0&0&0&1&0&1&0&0 \\ 1&0&0&1&1&0&1&0&0&0&1&0 \\ 0&1&0&0&1&0&0&0&0&1&1&1 \\ 0&0&0&0&0&0&1&1&1&1&1&0 \end{bmatrix}$$

Note that A''_1 can be produced from A'_1 by first swapping rows 1 and 3, and then swapping columns 1 and 3, columns 4 and 10, columns 2 and 6, and then columns 2 and 9. Let $\vec{v} = (1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0)$. Then $G'_0 = [I_{12}|A'_0] = [I_{12}|\mathcal{R}(A'_1, \vec{v})]$ generates the extension mate C'_0 of C'_1, and $G''_0 = [I_{12}|A''_0] = [I_{12}|\mathcal{R}(A''_1, \vec{v})]$ generates the extension mate C''_0 of C''_1, where:

$$A'_0 = \begin{bmatrix} 1&0&0&0&1&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&1&1&0&0&0&0 \\ 1&1&0&0&1&0&0&0&1&1&0&0 \\ 1&0&1&0&0&1&0&0&1&0&1&0 \\ 0&1&1&1&1&1&1&0&0&0&0&1 \\ 0&1&1&1&0&0&0&1&0&1&1&1 \\ 0&1&0&1&0&0&1&0&0&1&0&1 \\ 1&0&1&1&0&1&0&1&1&0&0&1 \\ 0&0&1&1&0&0&0&1&1&1&0&0 \\ 1&1&0&1&1&0&1&0&0&1&1&0 \\ 1&1&1&0&1&0&0&0&1&0&1&1 \\ 1&1&1&0&0&1&1&1&0&0&1&0 \end{bmatrix} \quad A''_0 = \begin{bmatrix} 1&0&0&0&1&1&0&0&0&0&0&0 \\ 1&0&1&0&0&1&1&1&0&0&0&0 \\ 0&1&1&1&1&1&0&0&1&1&0&0 \\ 0&0&0&1&0&0&0&0&1&0&1&0 \\ 1&1&0&1&1&0&1&0&1&0&0&1 \\ 0&0&1&1&0&0&0&1&0&0&1&1 \\ 1&1&1&0&0&1&1&0&0&1&0&1 \\ 1&0&1&1&0&1&0&1&1&0&0&1 \\ 0&0&1&0&0&0&0&1&0&1&0&0 \\ 1&0&0&1&1&0&1&0&0&0&1&0 \\ 1&0&1&1&1&0&0&0&0&1&1&1 \\ 0&0&0&0&0&0&1&1&1&1&1&0 \end{bmatrix}$$

The $(24, 12, 4)$ d_1-code C_0', generated by $G_0' = [I_{12}|A_0']$, contains exactly two weight 4 words. One is the codeword in the first row of G_0'. The other is the codeword in the second row of G_0'. On the other hand, the $(24, 12, 4)$ d_1-code C_0'', generated by $G_0'' = [I_{12}|A_0'']$, contains exactly four weight 4 words. Three are the codewords in the first, fourth, and ninth rows of G_0''. The other is the sum of the third, fifth, and seventh rows of G_0''. Since C_0' and C_0'' do not contain the same number of weight 4 words, we know that C_0' and C_0'' are inequivalent. Thus, the extension mates of the two equivalent $(24, 12, 6)$ self-dual codes C_1' and C_1'' are inequivalent.

As Example 5.16 demonstrates, not only do there exist equivalent $(2k, k, d \geq 6)$ self-dual codes whose extension mates are inequivalent, but their extension mates may not even contain the same number of weight 4 words. This leads us to the following two definitions:

Definition 5.17 Let C_1' be a $(2k, k, d \geq 6)$ self-dual code that has an extension mate C_0'. Then C_0' is said to be a *maximal extension mate* if for any C_1'', equivalent to C_1', that has an extension mate C_0'', the number of weight 4 words in C_0' is greater than or equal to the number of weight 4 words in C_0''.

Definition 5.18 Let C_1 be a $(2k, k, d \geq 6)$ self-dual code. Let C_1' be any code equivalent to C_1 that has a maximal extension mate C_0'. Then the *extension index* of C_1 is the number of weight 4 words in C_0'.

We will reduce the number of equivalent codes produced by Algorithm 5.15 by keeping only the $(2k, k, d \geq 6)$ self-dual codes C_1 that are produced from $(2k, k, 4)$ d_1-codes C_0 in which the number of weight 4 words in C_0 is equal to the extension index of C_1. That is, C_1 will be kept if and only if C_0 is equivalent to a code C_0' that is a maximal extension mate of a code C_1' that is equivalent to C_1. Since for each inequivalent $(2k, k, 4)$ d_1-code C_0' that is an extension mate of a code C_1', Algorithm 5.15 will use a code C_0 that is

equivalent to C_0' to produce at least one code C_1 that is equivalent to C_1', we will not lose any information by keeping only the codes C_1 produced from codes C_0 in which the number of weight 4 words in C_0 is equal to the extension index of C_1. That is, the list L_1 produced by the algorithm will still contain at least one representative for each inequivalent $(2k, k, d \geq 6)$ self-dual code. As we shall later see, keeping only the codes C_1 produced from codes C_0, in which the number of weight 4 words in C_0 is equal to the extension index of C_1, will also allow us to reduce the number of $(2k, k, 4)$ d_1-codes that are input to our algorithm.

Let us now consider the problem of computing the extension index of a $(2k, k, d \geq 6)$ self-dual code C_1. In this section, we will develop two different methods for computing the extension index of C_1. The reason we will develop two methods is that there exist codes for both methods in which one method is better than the other. The first method simply finds all the inequivalent extension mates of codes equivalent to C_1 and counts the number of weight 4 words in each. The largest number of weight 4 words counted is returned as the extension index. The second method uses the weight d words of C_1 to determine the extension index of C_1.

Before giving the details of our two methods for computing the extension index of a given $(2k, k, d \geq 6)$ self-dual code, we will prove a result that both of our methods will require. The result will show us that, for a given $(2k, k, d \geq 6)$ self-dual code C_1 with extension mate C_0, we can use the codewords of C_1 and the extension vector $\vec{v}$ to find the codewords of C_0.

Theorem 5.19 Let C_1 be a $(2k, k, d \geq 6)$ self-dual code that is generated by a matrix of the form $G_1 = [I_k | A_1]$, where the first row of A_1 is $0\ 1^{d-3}\ 1\ 1\ 0^*$. Let $\vec{v} = 1\ 1^{d-3}\ 0\ 0\ 0^*$. Let C_0 be the extension mate of C_1. That is, let C_0 be the $(2k, k, 4)$ d_1-code generated by $G_0 = [I_k | A_0]$, where $A_0 = \mathcal{R}(A_1, \vec{v})$. Let R_1 be any subset of the rows of G_1 and let R_0 be the corresponding subset of the rows of G_0. Let $\vec{c_1}$ be the codeword of C_1 that R_1 generates and let $\vec{c_0}$ be the codeword of C_0 that R_0 generates. Finally, let $\vec{u} = 0^k\, \vec{v}$.

Then:

$$\vec{c_0} = \begin{cases} \vec{c_1} & \text{if } \vec{c_1} \bullet \vec{u} \equiv 0 \bmod 2, \\ \vec{c_1} \oplus \vec{u} & \text{if } \vec{c_1} \bullet \vec{u} \equiv 1 \bmod 2. \end{cases}$$

Similarly:

$$\vec{c_1} = \begin{cases} \vec{c_0} & \text{if } \vec{c_0} \bullet \vec{u} \equiv 0 \bmod 2, \\ \vec{c_0} \oplus \vec{u} & \text{if } \vec{c_0} \bullet \vec{u} \equiv 1 \bmod 2. \end{cases}$$

Proof Let $G_1 = [I_k|A_1]$ be the generator matrix for the $(2k, k, d \geq 6)$ self-dual code C_1, where:

$$A_1 = \left[\begin{array}{c|c} W_0 & W_1 \\ \hline Y_0 & Y_1 \end{array}\right] \begin{array}{l} \text{for any } w \in W_0,\ w(w) \equiv 1 \bmod 2 \\ \\ \text{for any } y \in Y_0, \quad w(y) \equiv 0 \bmod 2 \end{array}$$

$$\underbrace{}_{d-2} \quad \underbrace{}_{k-(d-2)}$$

Let $\vec{v} = 1^{d-2}\, 0^*$. Then C_0, the extension mate of C_1, is generated by the matrix $G_0 = [I_k|A_0]$, where:

$$A_0 = \mathcal{R}(A_1, \vec{v}) = \left[\begin{array}{c|c} \overline{W_0} & W_1 \\ \hline Y_0 & Y_1 \end{array}\right]$$

Let R_1 be any subset of the rows of G_1, and let R_0 be the corresponding subset of the rows of G_0. Let $\vec{c_1}$ and $\vec{c_0}$ be the codewords generated by R_1 and R_0, respectively, and let $\vec{u} = 0^k\, \vec{v}$. What we want to show is that if $\vec{c_1}$ intersects $\vec{u}$ in an even number of positions then $\vec{c_0} = \vec{c_1}$ and if $\vec{c_1}$ intersects $\vec{u}$ in an odd number of positions then $\vec{c_0} = \vec{c_1} \oplus \vec{u}$.

Let $\vec{r_1}$ be a row in R_1 and let $\vec{r_0}$ be the corresponding row in R_0. Then, by the definition of $\mathcal{R}(A_1, \vec{v})$, if $\vec{r_1}$ intersects $\vec{u}$ in an even number of positions then $\vec{r_0} = \vec{r_1}$, else if $\vec{r_1}$ intersects $\vec{u}$ in an odd number of positions then $\vec{r_0} = \vec{r_1} \oplus \vec{u}$. Therefore, if the number

of rows in R_1 that intersect $\vec{u}$ in an odd number of positions is even then $\vec{c_0} = \vec{c_1}$, else if the number of rows in R_1 that intersect $\vec{u}$ in an odd number of positions is odd then $\vec{c_0} = \vec{c_1} \oplus \vec{u}$. Thus, all we have to show is that the codeword $\vec{c_1}$ generated by the set of rows R_1 intersects $\vec{u}$ in an odd number of positions if and only if the number of rows in R_1 that intersects $\vec{u}$ in an odd number of positions is odd.

Consider C, the $(2k+4, k+2, 4)$ self-dual code generated by $G = [I_{k+2} | \mathcal{A}(A_1, \vec{v})]$, where:

$$\mathcal{A}(A_1, \vec{v}) = \begin{array}{c} \begin{array}{cc} c_1 & c_2 \end{array} \\ \left[\begin{array}{cc||c|c} 1 & 0 & 1^{d-2} & 0^* \\ 0 & 1 & 1^{d-2} & 0^* \\ \hline\hline 1 & 1 & W_0 & W_1 \\ \hline 0 & 0 & Y_0 & Y_1 \end{array}\right] \end{array}$$

Let R be the set of rows in the last k rows of G that corresponds to the set of rows R_1 of G_1. Let $\vec{c}$ be the codeword of C generated by R and let $\vec{w}$ be the codeword in the first row of G. $\vec{c}$ is equal to $\vec{c_1}$ with four extra components. Two of these components are 0 0 and correspond to the first two columns of I_{k+2}, and the other two components, which we will denote by c_1 and c_2, are either 0 0 or 1 1 and correspond to the first two columns of $\mathcal{A}(A_1, \vec{v})$. Since the codewords $\vec{c}$ and $\vec{w}$ must intersect in an even number of positions, the two components c_1, c_2 of $\vec{c}$ will be equal to 1 1 if and only if the last k components of $\vec{c}$ intersect $\vec{v}$ (i.e. the last k components of $\vec{w}$) in an odd number of positions. Furthermore, since for any row $\vec{r} \in R$, the two components c_1, c_2 of $\vec{r}$ are equal to 1 1 if and only if the last k components of $\vec{r}$ intersect $\vec{v}$ in an odd number of positions, the two components c_1, c_2 of $\vec{c}$ will be equal to 1 1 if and only if the set of rows R that generates $\vec{c}$ contains an odd number of rows whose last k components intersect $\vec{v}$ in an odd number of positions. In other words, the last k components of $\vec{c}$ intersect $\vec{v}$ in an odd number of positions if and only if R contains an odd number of rows whose last k components intersect $\vec{v}$ in an odd number of positions. Therefore, since $\vec{u} = 0^k \vec{v}$ and since the last k components

of $\vec{c}$ and the rows of R correspond to the last k components of $\vec{c_1}$ and the rows of R_1, respectively, the codeword $\vec{c_1}$ generated by the set of rows R_1 intersects $\vec{u}$ in an odd number of positions if and only if the number of rows in R_1 that intersect $\vec{u}$ in an odd number of positions is odd.

Thus, if $\vec{c_1}$ intersects $\vec{u}$ in an even number of positions then $\vec{c_0} = \vec{c_1}$ and if $\vec{c_1}$ intersects $\vec{u}$ in an odd number of positions then $\vec{c_0} = \vec{c_1} \oplus \vec{u}$. Furthermore, since $\vec{u} = 0^k\,\vec{v}$ has even weight, if $\vec{c_0}$ intersects $\vec{u}$ in an even number of positions then $\vec{c_1} = \vec{c_0}$ and if $\vec{c_0}$ intersects $\vec{u}$ in an odd number of positions then $\vec{c_1} = \vec{c_0} \oplus \vec{u}$. This completes our proof. □

Let us now develop our first method for finding the extension index of a $(2k, k, d \geq 6)$ self-dual code C_1. Our first method works by initially finding all of the generator matrices $G_1' = [I_k|A_1']$ for the $(2k, k, d \geq 6)$ self-dual codes C_1', equivalent to C_1, that may have inequivalent extension mates. Each generator matrix $G_1' = [I_k|A_1']$ produced is then used to find the generator matrix $G_0' = [I_k|A_0']$ for C_0', the extension mate of C_1'. That is, for each G_1' produced, G_0' is produced by finding $A_0' = \mathcal{R}(A_1', \vec{v})$, where $\vec{v} = 1\ 1^{d-3}\ 0\ 0\ 0^*$. G_0' is then used to count the number of weight 4 words in C_0'. The largest number of weight 4 words counted is then returned as the extension index of C_1.

Let us now consider how we can find generator matrices for all the codes equivalent to C_1 that may have inequivalent extension mates. Let $\vec{w}$ be a weight d word in C_1. Let $G_1' = [I_k|A_1']$ generate a code C_1' that has an extension mate and is equivalent to C_1, such that the first row of G_1' is the weight d word $\vec{w'}$ of C_1' that corresponds to the weight d word $\vec{w}$ of C_1. Let S' denote the set of $d-2$ columns of A_1' that intersects the extension

vector $\vec{v} = 1\ 1^{d-3}\ 0\ 0\ 0^{*}$:

$$A_1' = \left[\begin{array}{c|c|c|c} 0 & 1^{d-3} & 1\ 1 & 0^{*} \\ \hline 1 & W_0 & W_1 & W_2 \\ 0 & X_0 & X_1 & X_2 \\ \hline 1 & Y_0 & Y_1 & Y_2 \\ 0 & Z_0 & Z_1 & Z_2 \end{array}\right] \vec{w}'$$

$\underbrace{\qquad\qquad}_{S'}$

Let T_d' denote the d columns of G_1' that correspond to the ones of the weight d word $\vec{w}'$ and let T_{2k-d}' denote the remaining $2k-d$ columns of G_1'. Let C_1'' denote any code that has an extension mate and is equivalent to C_1', such that C_1'' is generated by a matrix $G_1'' = [I_k|A_1'']$ in which the first row of G_1'' is the weight d word of C_1'' that corresponds to the weight d word $\vec{w}'$ of C_1'. Finally, let S'' denote the set of $d-2$ columns of A_1'' that intersect the extension vector $\vec{v}$. Then if π is a permutation of the columns of G_1' that results in a matrix whose row-reduced echelon form is $G'' = [I_k|A'']$ then π must permute one and only one column of T_{2k-d}' to the first column of S'' and $d-3$ of the d columns of T_d' to the remaining $d-3$ columns of S''. Furthermore, if the extension mates of C_1' and C_1'' are inequivalent then S' and S'' cannot be equal since, by Theorem 5.19, the codewords of the extension mate of C_1' are determined by the parity of the $d-2$ components of the codewords of C_1' that correspond to the set of columns S'. Therefore, to find generator matrices G_1'' for all the codes C_1'' that may have inequivalent extensions, where the first row of G_1'' is the weight d word of C_1'' that corresponds to the weight d word $\vec{w}'$ of C_1', we need only perform one permutation of the columns G_1' for each of the $\binom{2k-d}{1}\binom{d}{d-3} = (2k-d)\binom{d}{3}$ possibilities for the set S'' of columns in $G_1'' = [I_k|A_1'']$.

This leads us to our first method for finding the extension index of the $(2k, k, d \geq 6)$ self-dual code C_1. Let $G_1 = [I_k|A_1]$ be the generator matrix for C_1. Let x denote the number of weight d words in C_1. Our first step is to find all the weight d words in C_1.

This can be accomplished by simply finding the weights of the $\sum_{i=1}^{d-1}\binom{k}{i}$ codewords that are generated by i rows of G_1, for $i = 1, 2, \ldots, d-1$ (since, by Theorem 2.8 of Chapter 2, any weight w word is the sum of at most $w-1$ rows of $G = [I_k|A]$). Our next step is to produce, for each weight d word $\vec{w} \in C_1$, one generator matrix $G_1' = [I_k|A_1']$ for a code C_1', where C_1' has an extension mate and is equivalent to C_1, such that the first row of G_1' is the weight d word of C_1' that corresponds to the weight d word $\vec{w}$ of C_1. For each G_1' produced, the $(2k-d)\binom{d}{3}$ generator matrices $G_1'' = [I_k|A_1'']$ are produced, one for each of the different choices for the set S''. Each G_1'' produced is then used to find $G_0'' = [I_k|\mathcal{R}(A_1'', \vec{v''})]$, the generator matrix of the extension mate C_0'' of C_1''. The number of weight 4 words of C_0'' are then counted by finding the weights of the $\sum_{i=1}^{3}\binom{k}{i}$ codewords that are generated by one, two, or three rows of G_0''. The largest number of weight 4 words counted is then returned as the extension index of C_1.

Our second method for finding the extension index of a $(2k, k, d \geq 6)$ self-dual code C_1 is based on the result that there exists an extension mate with m weight 4 words for a code equivalent to C_1 if and only if C_1 contains m weight d codewords that are a permutation of the following block of m weight d words:

$$F_{d,m} = \overbrace{\begin{cases} 1\ 1\ \cdots\ 1\ 1\ \cdots\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0\ 0^* \\ 1\ 1\ \cdots\ 1\ 1\ \cdots\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ \cdots\ 0\ 0\ 0\ 0^* \\ \vdots\ \vdots\ \quad\ \vdots\ \vdots\ \quad\ \vdots\ \vdots\ \quad\ \vdots\ \quad\ \quad\ \vdots\ \quad\ \quad\ \vdots\ \quad\ \vdots \\ 1\ 1\ \cdots\ 1\ 0\ \cdots\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 1\ 1\ 1\ 0^* \end{cases}}^{m}$$

$$\underbrace{\qquad\qquad}_{d-m-2}\ \underbrace{\qquad\qquad}_{m}\ \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{3m}$$

where m is as large as possible. The range of the variable m is $1 \leq m \leq d-2$ if $d = 6$ or $d = 8$ and $m = 1$ if $d \geq 10$. The reason that $m = 1$ if $d \geq 10$ is that any two codewords in the $F_{d,m}$-block sum to a weight 8 word.

In order to prove this result, we first require the following lemma:

Lemma 5.20 Let C be a $(2k, k, d)$ self-dual code. Let S be any subset of j columns of C, where $1 \leq j \leq d-1$. Then there exists a code C', equivalent to C, generated by a matrix of the form $G' = [I_k|A']$, such that the subset of the columns of G' that correspond to the subset S of the columns of G, are columns of the matrix A'.

Proof Let $G = [I_k|A]$ be a generator matrix for a code equivalent to C. Let S_I and S_A denote the subset the of columns of I_k and A, respectively, that correspond to the set of j columns of S. If $|S_A| = j$ then we are done. So, assume $|S_A| < j \leq d-1$. Let M denote the remaining columns of A. Then, since $|S_A| < d-1$ and since every row in A must have weight $\geq d-1$, the rows of M must all be non-zero. Therefore, if i_0 is a row in A that corresponds to a column of I_k that is in S_I, there exists a column j_0 in M such that entry (i_0, j_0) of A is non-zero. Therefore, we can apply an $\mathrm{ADD}_{r,c}(i_0, j_0)$ operation to A which swaps the column of I_k that corresponds to row i_0 of A with column j_0 of A and then row-reduces. In other words, if we apply an $\mathrm{ADD}_{r,c}(i_0, j_0)$ operation to A, producing the matrix A', then the matrix $G' = [I_k|A']$ has the property that $|S'_I| = |S_I| - 1$ and $|S'_A| = |S_A| + 1$, where S'_I and S'_A are the columns of I_k and A', respectively, that correspond to the set of columns S. If $|S'_A| = j$, we are done. Otherwise, since $|S'_A| < j \leq d-1$, we can continue to apply $\mathrm{ADD}_{r,c}$ operations to A' until a matrix A'' is produced in which $|S''_A| = j$, where S''_A are the columns of A'' that correspond to the set of columns S. □

We are now ready to prove the following:

Theorem 5.21 Let C_1 be any $(2k, k, d \geq 6)$ self-dual code. Then there exists an extension mate with m weight 4 words for a code equivalent to C_1 if and only if C_1 contains m

weight d codewords that are a permutation of the following block of m weight d words:

$$F_{d,m} = \overset{m}{\left\{ \begin{array}{cccccccccccccccccccc} 1 & 1 & \cdots & 1 & 1 & \cdots & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0^* \\ 1 & 1 & \cdots & 1 & 1 & \cdots & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots & 0 & 0 & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & & \vdots & & & & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 & 0 & \cdots & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 1 & 0^* \end{array} \right.}$$

$$\underbrace{\qquad\qquad}_{d-m-2} \quad \underbrace{\qquad}_{m} \quad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{3m}$$

where m is as large as possible. If $d = 6$ or $d = 8$ then $1 \leq m \leq d-2$ and if $d \geq 10$ then $m = 1$. We will use the notation $F_{d,m}$ to denote any permutation of the columns of the above block of m weight d words.

Proof Let C_0' be the extension mate of a code C_1' that is equivalent to C_1. We will first show that if C_0' contains exactly m weight 4 words then C_1' must contain an $F_{d,m}$-block.

Let $\vec{c_0}$ be any weight 4 word in C_0' and let $\vec{c_1}$ be the corresponding codeword in C_1'. Let $\vec{u} = 0^k \vec{v}$, where $\vec{v} = 1^{d-2}0^*$. Then, by Theorem 5.19, if $\vec{c_0}$ intersects $\vec{u}$ in an even number of positions then $\vec{c_1} = \vec{c_0}$, else if $\vec{c_0}$ intersects $\vec{u}$ in an odd number of positions then $\vec{c_1} = \vec{c_0} \oplus \vec{u}$. However, since $\vec{c_0}$ has weight 4 and $\vec{c_1}$ has weight ≥ 6, $\vec{c_1} \neq \vec{c_0}$. Therefore, $\vec{c_0}$ intersects $\vec{u}$ in an odd number of positions and $\vec{c_1} = \vec{c_0} \oplus \vec{u}$. Thus, $\vec{c_0}$ intersects $\vec{u}$ in either one column or in three columns. However, if $\vec{c_0}$ intersects $\vec{u}$ in three columns then $\vec{c_1} = \vec{c_0} \oplus \vec{u}$ has weight $d-4$:

$$\begin{array}{ll} \vec{u} & 1\,1\,1\,1^{d-5}\;0^* \\ \vec{c_0} & 1\,1\,1\,0^{d-5}\;1\,0^* \\ \hline \vec{c_1} & 0\,0\,0\,1^{d-5}\;1\,0^* \end{array}$$

This contradicts the fact that the distance of C_1' is d. Therefore, $\vec{c_0}$ must intersect $\vec{u}$ in

exactly one column and $\vec{c_1} = \vec{c_0} \oplus \vec{u}$ is a weight d word:

$$\begin{array}{ll} \vec{u} & 1\,1^{d-3}\,0^* \\ \vec{c_0} & 1\,0^{d-3}\,1\,1\,1\,0^* \\ \hline \vec{c_1} & 0\,1^{d-3}\,1\,1\,1\,0^* \end{array}$$

Furthermore, since none of the weight 4 words in C_0' intersect, the m weight 4 words in C_0' must intersect $\vec{u}$ as follows:

$\vec{u}$	1	1	$\cdots$	1	1	$\cdots$	1	1	0	0	0	0	0	0	$\cdots$	0	0	0	0^*
	0	0	$\cdots$	0	0	$\cdots$	0	1	1	1	1	0	0	0	$\cdots$	0	0	0	0^*
	0	0	$\cdots$	0	0	$\cdots$	1	0	0	0	0	1	1	1	$\cdots$	0	0	0	0^*
	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$			$\vdots$				$\vdots$		$\vdots$
	0	0	$\cdots$	0	1	$\cdots$	0	0	0	0	0	0	0	0	$\cdots$	1	1	1	0^*
	$\underbrace{\qquad\qquad}_{d-m-2}$				$\underbrace{\qquad\qquad}_{m}$														

Therefore, the weight d words in C_1' that correspond to the weight 4 words in C_0' form an $F_{d,m}$-block:

$\vec{u}$	1	1	$\cdots$	1	1	$\cdots$	1	1	0	0	0	0	0	0	$\cdots$	0	0	0	0^*
	1	1	$\cdots$	1	1	$\cdots$	1	0	1	1	1	0	0	0	$\cdots$	0	0	0	0^*
	1	1	$\cdots$	1	1	$\cdots$	0	1	0	0	0	1	1	1	$\cdots$	0	0	0	0^*
	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$			$\vdots$				$\vdots$		$\vdots$
	1	1	$\cdots$	1	0	$\cdots$	1	1	0	0	0	0	0	0	$\cdots$	1	1	1	0^*
	$\underbrace{\qquad\qquad}_{d-m-2}$				$\underbrace{\qquad\qquad}_{m}$														

Furthermore, C_1' does not contain any other weight d words that can be adjoined to the $F_{d,m}$-block to form an $F_{d,m+1}$-block, for if C_1' did contain such a weight d word, then the corresponding codeword in C_0' would be a weight 4 word, which would imply that C_0' had $m+1$ weight 4 words, which contradicts the assumption that C_0' contains exactly m weight 4 words.

Thus, if C_0' contains exactly m weight 4 words then C_1', and thus C_1, contains an $F_{d,m}$-block.

Let us now show that if C_1 contains an $F_{d,m}$-block, then there exists a code C_1', equivalent to C_1, whose extension mate contains exactly m weight 4 words.

Let C_1 be a $(2k, k, d \geq 6)$ self-dual code that contains an $F_{d,m}$-block:

$$F_{d,m} = \overset{m}{\left\{\begin{array}{c} \overbrace{}^{S}\ \overbrace{}^{} \\ 1\ 1\ \cdots\ 1\ 1\ \cdots\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0\ 0^* \\ 1\ 1\ \cdots\ 1\ 1\ \cdots\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ \cdots\ 0\ 0\ 0\ 0^* \\ \vdots\ \vdots\ \quad\ \vdots\ \vdots\ \quad\ \vdots\ \vdots\ \quad\ \vdots\ \quad\ \vdots\ \quad\ \quad\ \vdots\ \quad\ \vdots \\ 1\ 1\ \cdots\ 1\ 0\ \cdots\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 1\ 1\ 1\ 0^* \\ \underbrace{}_{d-m-2}\ \underbrace{}_{m}\ \underbrace{}_{3m} \end{array}\right.}$$

where m is as large possible. The set of columns S consists of the columns in which the codewords of the $F_{d,m}$-block intersect. Since $|S| = d - 2 < d - 1$, by Lemma 5.20, we know there exists a code C_1'', equivalent to C_1, that is generated by a matrix of the form $G_1'' = [I_k|A_1'']$, in which the columns of G_1'' that correspond to the set of columns S of C are columns of the matrix A_1''.

We will use the generator matrix $G_1'' = [I_k|A_1'']$ to prove that there exists a code C_1''', equivalent to C_1, that is generated by a matrix of the form $G_1''' = [I_k|A_1''']$ such that the columns of G_1''' that correspond to the set of columns S of C are columns of A_1''', and such that G_1''' contains a weight d row that corresponds to a weight d word of the $F_{d,m}$-block in C_1. This in turn shows us that there exists a generator matrix $G_1' = [I_k|A_1']$ for a code C_1', where C_1' has an extension mate and is equivalent to C_1, such that the first $d-2$ columns of A_1' correspond to the set of $d-2$ columns S of C_1. Once we have proved that such a C_1' exists, we will prove that the extension mate of C_1' must contain exactly m weight 4 words.

Let S'' denote the set of columns of the $F_{d,m}$-block in C_1'' that corresponds to the set of columns S of the $F_{d,m}$-block in C_1. Then, by our definition of C_1'', the set of columns S'' corresponds to columns of the matrix A_1'' in $G_1'' = [I_k|A_1'']$. Therefore, since each weight

d word $\vec{w''}$ of the $F_{d,m}$-block in C_1'' contains exactly three non-zero columns that are not columns of the set S'', we know there are at most three columns in I_k that correspond to non-zero columns of $\vec{w''}$. In other words, any weight d word $\vec{w''}$ of the $F_{d,m}$-block in C_1'' is the sum of at most three rows of the generator matrix $G_1'' = [I_k|A_1'']$. We can use this fact to prove that for any weight d word $\vec{w''}$ of the $F_{d,m}$-block in C_1'', there exists a sequence of $\text{ADD}_{r,c}$ operations on A_1'' ,that does not involve any of the columns of A_1'' that correspond to the set of columns S'', that results in a matrix A_1''' in which the weight d word $\vec{w'''}$ is a row of $G_1''' = [I_k|A_1''']$, where $\vec{w'''}$ corresponds to the weight d word $\vec{w''}$. This shows us there exists a code C_1''', equivalent to C_1, that is generated by a matrix of the form $G_1''' = [I_k|A_1''']$ such that the columns of G_1''' that correspond to the set of columns S of C are columns of A_1''', and such that G_1''' contains a weight d row that corresponds to a weight d word of the $F_{d,m}$-block in C_1.

Let us now show that such sequences of $\text{ADD}_{r,c}$ operations on A_1'' exist for any weight d word in the $F_{d,m}$-block of C_1''. Let $\vec{w''}$ be any weight d word in the $F_{d,m}$-block of C_1''. If $\vec{w''}$ is a row of G_1'' then we are done. Therefore, we have two cases to consider: the case in which $\vec{w''}$ is the sum of two rows of G_1'' and the case in which $\vec{w''}$ is the sum of three rows of G_1''.

Let T_1 denote the columns of A_1'' that correspond to the set of columns S'' and let T_2 denote the remaining columns of A_1''. Suppose $\vec{w''}$ is the sum of three rows $\vec{r_1}$, $\vec{r_2}$, and $\vec{r_3}$ of $G_1'' = [I_k|A_1'']$. Then, since three columns in the identity I_k and $d-3$ columns in the set T_1 sum to one in rows $\vec{r_1}$, $\vec{r_2}$, and $\vec{r_3}$, none of the columns in T_2 sum to one in rows $\vec{r_1}$, $\vec{r_2}$, and $\vec{r_3}$. Therefore, every column in T_2 has even weight. Furthermore, since $|T_1| = d-2$ and since every row in A_1'' has weight $\geq d-1$ (since C_1'' has distance d) there must exist a column in T_2 with weight 2. Let j be such a column, and let i be a row of A_1'' that has a value of 1 in column j, where i corresponds to one of the three rows $\vec{r_1}$, $\vec{r_2}$, and $\vec{r_3}$. Then an $\text{ADD}_{r,c}(i,j)$ operation on A_1'' will produce a matrix A_1''' in which the weight d word $\vec{w'''}$, that corresponds to $\vec{w''}$, is the sum of two rows of $G_1''' = [I_k|A_1''']$.

Assume $\vec{w''}$ is the sum of two rows $\vec{r_1}$ and $\vec{r_2}$ of $G_1'' = [I_k|A_1'']$. Then, since two columns in the identity I_k and $d-3$ columns in the set T_1 sum to one in rows $\vec{r_1}$ and $\vec{r_2}$, exactly one column of T_2 sums to one in rows $\vec{r_1}$ and $\vec{r_2}$. Therefore, all but one of the columns in T_2 has even weight. Furthermore, since $|T_1| = d-2$ and since every row in A_1'' has weight $\geq d-1$, there must exist a column in T_2 with weight 2. Let j be such a column and let i be a row of A_1'' that has a value of 1 in column j, where i corresponds to one of the two rows $\vec{r_1}$ and $\vec{r_2}$. Then an $\mathrm{ADD}_{r,c}(i,j)$ operation on A_1'' will produce a matrix A_1''' in which the weight d word $\vec{w'''}$ that corresponds to $\vec{w''}$ is a row of $G_1''' = [I_k|A_1''']$, and we are done.

Therefore, there exists a code C_1''', equivalent to C_1, that is generated by a matrix of the form $G_1''' = [I_k|A_1''']$ such that the columns of G_1''' that correspond to the set of columns S of C are columns of A_1''', and such that G_1''' contains a weight d row that corresponds to a weight d word of the $F_{d,m}$-block in C_1. Thus, there exists a code C_1', where C_1' has an extension mate and is equivalent to C_1, where C_1' is generated by a matrix $G_1' = [I_k|A_1']$, such that the first $d-2$ columns of A_1' correspond to the set of $d-2$ columns S of C_1 (i.e. the set S of $d-2$ columns in C_1 in which the weight d words of the $F_{d,m}$-block intersect).

Let C_0' be the $(2k,k,4)$ d_1-code generated by $G_0' = [I_k|\mathcal{R}(A_1',\vec{v})]$, where $\vec{v} = 1^{d-2}0^*$. That is, let C_0' be the extension mate C_1'. Then, by Theorem 5.19, for each codeword $\vec{c_1}$ in C_1', if $\vec{c_1}$ intersects $\vec{u} = 0^k\,\vec{v}$ in an odd number of positions then the corresponding codeword in C_0' is $\vec{c_0} = \vec{c_1} \oplus \vec{u}$. Since the non-zero columns of $\vec{u}$ correspond to the set of $d-2$ columns S of C, every weight d word in the $F_{d,m}$-block of C_1' intersects $\vec{u} = 0^k\,\vec{v}$ in an odd number of positions. Therefore, the corresponding codewords in C_0' are permutations

of the following m weight 4 words:

$\vec{u}$	1	1	$\cdots$	1	1	$\cdots$	1	1	0	0	0	0	0	0	$\cdots$	0	0	0	0^*
	0	0	$\cdots$	0	0	$\cdots$	0	1	1	1	1	0	0	0	$\cdots$	0	0	0	0^*
	0	0	$\cdots$	0	0	$\cdots$	1	0	0	0	0	1	1	1	$\cdots$	0	0	0	0^*
	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$			$\vdots$				$\vdots$		$\vdots$
	0	0	$\cdots$	0	1	$\cdots$	0	0	0	0	0	0	0	0	$\cdots$	1	1	1	0^*
	$\underbrace{\quad\quad\quad}_{d-m-2}$				$\underbrace{\quad\quad\quad}_{m}$														

Furthermore, since the $F_{d,m}$-block is as large as possible, these are the only weight 4 words C_0' may contain. That is, if C_0' contained another weight 4 word $\vec{c_0'}$, then since the corresponding codeword $\vec{c_1'}$ in C_1' must have weight $\geq d$, $\vec{c_0'}$ must intersect $\vec{u}$ in exactly one position. However, since the weight 4 words of C_0' do not intersect, this would imply that the corresponding codeword $\vec{c_1'}$ and the $F_{d,m}$-block in C_1' would form an $F_{d,m+1}$-block, which contradicts the assumption that m is as large as possible.

Thererfore, if C_1 contains an $F_{d,m}$-block, there exists a code C_1', equivalent to C_1, whose extension mate contains exactly m weight 4 words.

Thus, for any $(2k, k, d \geq 6)$ self-dual code C_1, C_1 contains an $F_{d,m}$-block if and only if there exists a code C_1', equivalent to C_1, such that the extension mate of C_1 contains exactly m weight 4 words. □

Theorem 5.21 gives us our second method for finding the extension index of a $(2k, k, d \geq 6)$ self-dual code C_1. That is, if the largest $F_{d,m}$-block in C_1 is an F_{d,m_0}-block, then the extension index of C_1 is m_0.

Let us now consider how we can find the largest $F_{d,m}$-block that C_1 contains. Let $G_1 = [I_k|A_1]$ be the generator matrix for the $(2k, k, d \geq 6)$ self-dual code C_1. Our first step is to find the set W of all weight d words in C_1. Since any weight d word in C_1 is the sum of at most $d-1$ rows in G_1, we can find W by simply generating all the codewords of C_1 that are the sum of i rows of G_1, where $1 \leq i \leq d-1$, and then finding their

weights. The next step is to find all pairs $\vec{w_1}, \vec{w_2} \in W$ that form an $F_{d,2}$-block. For each pair (if any), we will then find all $\vec{w_3} \in W - \{\vec{w_1}, \vec{w_2}\}$ that can be adjoined to $\vec{w_1}$ and $\vec{w_2}$ to form an $F_{d,3}$-block. For each triple $\vec{w_1}, \vec{w_2}, \vec{w_3} \in W$ found that form an $F_{d,3}$-block (if any), we will then find all the words in $W - \{\vec{w_1}, \vec{w_2}, \vec{w_3}\}$ that can be adjoined to the $F_{d,3}$-block to form an $F_{d,4}$-block. As demonstrated by Table 5.2, if there are exactly n words in $W - \{\vec{w_1}, \vec{w_2}, \vec{w_3}\}$ that can be adjoined to the $F_{d,3}$-block to form an $F_{d,4}$-block, then these n words, together with $\vec{w_1}$, $\vec{w_2}$, and $\vec{w_3}$, form an $F_{d,3+n}$-block.

	$d-5$				
$\vec{w_1}$	111...11	110	111000000	000	0^*
$\vec{w_2}$	111...11	101	000111000	000	0^*
$\vec{w_3}$	111...11	011	000000111	000	0^*
$\vec{w_a}$	011...11	111	000000000	111	0^*
$\vec{w_{b_1}}$	011...11	111	000000000	100	110^*
$\vec{w_{b_2}}$	101...11	111	000000000	110	10^*
$\vec{w_{b_3}}$	101...11	111	000000000	000	1110^*

Table 5.2: The last three rows represent all the ways a weight d word can be adjoined to the $F_{d,3}$-block (formed by $\vec{w_1}, \vec{w_2}, \vec{w_3}$) to form an $F_{d,4}$-block, and intersect the weight d word $\vec{w_a}$ in an even number of positions.

The first three rows in Table 5.2 form an $F_{d,3}$-block. The fourth row, $\vec{w_a}$, is any weight d word that can be adjoined to the $F_{d,3}$-block to form an $F_{d,4}$-block. The last three rows, $\vec{w_{b_1}}, \vec{w_{b_2}}, \vec{w_{b_3}}$, represent all the different ways any other weight d word can be adjoined to the $F_{d,3}$-block to form an $F_{d,4}$-block, and intersect $\vec{w_a}$ in an even number of positions. Notice that both $\vec{w_a} \oplus \vec{w_{b_1}}$ and $\vec{w_a} \oplus \vec{w_{b_2}}$ have weight 4. Therefore, since C_1 has distance $d \geq 6$, the only weight d words that can be adjoined to both the $F_{d,3}$-block and $\vec{w_a}$ are the weight d words that have the same form as $\vec{w_{b_3}}$. If W contains $n-1$ such weight d words, then together with $\vec{w_1}$, $\vec{w_2}$, $\vec{w_3}$, and $\vec{w_a}$, they will form an $F_{d,3+n}$-block. Therefore,

if C_1 contains x weight d words, finding the largest $F_{d,m}$-block in C_1 will require no more than $(x-3)\binom{x}{3}$ iterations through the list W of weight d words of C_1.

We now have our two different methods for finding the extension index of a $(2k, k, d \geq 6)$ self-dual code C_1. Let $G_1 = [I_k|A_1]$ be the generator matrix for C_1 and let x be the number of weight d words in C_1. Both methods first find all the weight d words in C_1 by finding the weight of the $\sum_{i=1}^{d-1}\binom{k}{i}$ codewords that are generated by i rows of G_1. for $i = 1, 2, \ldots, d-1$. The first method then finds generator matrices $G_1' = [I_k|A_1']$ for the $x(2k-d)\binom{d}{3}$ codes C_1', equivalent to C_1, whose extension mates may be inequivalent. For each G_1' found, the generator matrix G_0' for the extension mate C_0' of C_1' is then produced. The number of weight 4 words in C_0' is then found by finding the weight of the $\sum_{i=1}^{3}\binom{k}{i}$ codewords that are generated by i rows of G_0', for $i = 1, 2, 3$. The largest number of weight 4 words found is the extension index. The second method performs at most $(x-3)\binom{x}{3}$ iterations through the weight d words of C_1 to find the largest $F_{d,m}$-block in C_1. By Theorem 5.21, if $F_{d,m}$ is the largest block, then the extension index of C_1 is m. For a given C_1, the method to use is dependent on factors such as the distance of C_1, the number of weight d words in C_1, and the dimension of C_1.

Let us now consider how we can use the extension index of a $(2k, k, d \geq 6)$ self-dual code to reduce the number of equivalent codes produced by Algorithm 5.15 (i.e. our algorithm for enumerating the $(2k, k, d \geq 6)$ self-dual codes). Recall, Algorithm 5.15 works by producing, for each inequivalent $(2k, k, 4)$ d_1-code C_0 generated by $G_0 = [I_k|A_0]$, all the extension vectors $\vec{v}$ that result in a generator matrix $G = [I_{k+2}|\mathcal{A}(A_0, \vec{v})]$ that generates a $(2k+4, k+2, 4)$ self-dual code C that contains one and only one weight 4 word. For each C_0 and $\vec{v}$, Algorithm 5.15 then finds the two related codes, C_r and C_s, that can be extended to a code equivalent to C. If one of the related codes, which we will denote by C_1, has distance d, and the other related code has distance d_0, where $d > d_0 \geq 4$, then the $(2k, k, d \geq 6)$ self-dual code C_1 is recorded in the list L_1. Now, let C_0' be any code, equivalent to C_0, such that C_0' is the extension mate of a $(2k, k, d \geq 6)$ self-dual code C_1'. Then C_0' and C_1' can be extended to equivalent $(2k+4, k+2, 4)$ self-dual

codes that contain one and only one weight 4 word. Therefore, for each C_0', equivalent to C_0, where C_0' is an extension mate of a $(2k, k, d \geq 6)$ self-dual code C_1', Algorithm 5.15 will use C_0 to produce at least one code C_1 that is equivalent C_1'. Thus, if we only keep the $(2k, k, d \geq 6)$ self-dual codes C_1 that are produced by a code C_0 in which the number of weight 4 words in C_0 is equal to the extension index of C_1, then the list L_1 produced by the algorithm will still contain at least one representative for each of the inequivalent $(2k, k, d \geq 6)$ self-dual codes.

Since we are only keeping codes C_1 that are produced from codes C_0 such that the number of weight 4 words in C_0 is equal to the extension index of C_1, we may not necessarily have to "completely" compute the extension index of C_1. For example, if $d = 6$ and the number of weight 4 words in C_0 is three, then if either one of our two methods for computing the extension index "finds" an extension mate with four weight 4 words, then we can stop since we would know at this point that C_0 does not have the same number of weight 4 words as the extension index of C_1. Furthermore, Theorem 5.21 provides us with an upper bound for the extension index of any $(2k, k, d \geq 6)$ self-dual code. That is, by Theorem 5.21, for any $(2k, k, d \geq 6)$ self-dual code C_1, if $d = 6$ then the extension index of C_1 is ≤ 4, if $d = 8$ then the extension index of C_1 is ≤ 6, and if $d \geq 10$ then the extension index of C_1 is 1. Thus, if C_0 is an extension mate then the number of weight 4 words in C_0 is $\leq \max(4, 6, 1) = 6$. Therefore, our algorithm does not have to consider any of the $(2k, k, 4)$ d_1-codes that contain greater than or equal to seven weight 4 words.

We can also use the upper bounds on the extension index for the $(2k, k, d \geq 6)$ self-dual codes to avoid computing the extension index of C_1. That is, we can avoid computing the extension index if $d = 6$ and C_0 has four weight 4 words, $d = 8$ and C_0 has six weight 4 words, or $d \geq 10$ and C_0 has one weight 4 word. However, it must be pointed out that one aspect of Algorithm 5.15 that we have not investigated is whether or not it is possible for the algorithm to produce a code C_1, from a code C_0, in which C_0 is not equivalent to a code that is an extension mate of a code that is equivalent to C_1.

That is, it has not been investigated whether or not there exist codes C_0 and C_1, where C_0 is not equivalent to a code C_0' that is an extension mate of a code C_1' equivalent to C_1, such that C_0 and C_1 can be extended to equivalent $(2k+4, k+2, 4)$ self-dual codes containing one and only one weight 4 word. Therefore, by not computing the extension mate for codes C_1 produced from codes C_0 that contain the same number of weight 4 words as the upper bound for the extension index of C_1, we may in fact be keeping codes C_1 that were not produced from codes with the same number of weight 4 words as the extension index of C_1. However, for the enumerations in which we have checked whether or not this happens, it has not happened. That is, for each $(2k, k, 6)$ self-dual code C_1 produced from a code C_0 with four weight 4 words, the extension index of C_1 has been four, and, for each $(2k, k, 8)$ self-dual code C_1 produced from a code C_0 with six weight 4 words, the extension index of C_1 has been six. Note that no $(2k, k, d \geq 10)$ self-dual codes were found. Therefore, in our algorithm, for each $(2k, k, d \geq 6)$ self-dual code C_1 produced from a code C_0 in which the number of weight 4 words in C_0 is equal to the upper bound on the extension index of C_1, the extension index of C_1 is not computed. For completeness sake, our algorithm also checks to make sure the number of weight 4 words in C_0 is not greater than the upper bound on the extension index of C_1.

Let us now reorganize/rewrite Algorithm 5.15 with our improvements included. The algorithm makes use of the boolean function *MaxExtensionMate*(A_1, d, m), where $G_1 = [I_k|A_1]$ is a generator matrix for a $(2k, k, d \geq 6)$ self-dual code C_1 and m is the number of weight 4 words in the $(2k, k, 4)$ d_1-code C_0 that was used to produce A_1. *MaxExtensionMate* uses one of our two methods for finding the extension index of C_1 to determine whether or not the code C_0 is a maximal extension mate (i.e. whether or not m is equal to the extension index of C_1). If m is equal to the extension index of C_1 then *MaxExtensionMate* returns *true*, otherwise it returns *false*. The function assumes $d = 6$ or $d = 8$ and $m < d - 2$.

Algorithm 5.22 :

- Input: a list L_0 of $k \times k$ matrices A_0, where $G_0 = [I_k|A_0]$ generates a $(2k, k, 4)$ d_1-code, such that for any $(2k, k, 4)$ d_1-code C_0 there exists one and only one matrix A_0 in L_0 such that $G_0 = [I_k|A_0]$ generates a code equivalent to C_0. Without loss of generality, it is assumed each G_0 contains at least one $d_{1,0}$-block.
- Output: a list L_1 of $k \times k$ matrices A_1, where $G_1 = [I_k|A_1]$ generates a $(2k, k, d \geq 6)$ self-dual code, such that for any $(2k, k, d \geq 6)$ self-dual code C_1 there exists at least one matrix A_1 in L_1 such that $G_1 = [I_k|A_1]$ generates a code equivalent to C_1.

begin

for each $A_0 \in L_0$ **do**

Let m be the number of weight 4 words in the code generated by $G_0 = [I_k|A_0]$.

if $m \leq 6$ **then**

Let $G_{B_1}, G_{B_2}, \ldots, G_{B_m}$ be the d_1-blocks in $G_0 = [I_k|A_0]$.

Let $\vec{v}$ be an extension vector with columns $v\vec{}_{B_1}, v\vec{}_{B_2}, \ldots, v\vec{}_{B_m}, v\vec{}_R$, where $v\vec{}_{B_i}$ is an extension of G_{B_i} and $v\vec{}_R$ consists of the remaining columns of $\vec{v}$.

Fix the columns of each $v\vec{}_{B_i}$ as follows:

if G_{B_i} is a $d_{1,0}$-block, $v\vec{}_{B_i} = 1\,0\,0$,

if G_{B_i} is a $d_{1,3}$-block, $v\vec{}_{B_i} = 1\,0$, and,

if G_{B_i} is a $d_{i,4}$-block, $v\vec{}_{B_i} = 1$.

for each $v\vec{}_R$ that results in a $\vec{v}$ such that $w(\vec{v}) \equiv 0 \bmod 2$, $w(\vec{v}) \geq 4$, $\vec{v}$ differs from every row in A_0 in at least two columns, and $\vec{v}$ is not the sum of any pair of rows in A_0 **do**

Let $A_r = \mathcal{R}(A_0, \vec{v})$.

Pick any G_{B_i} that is a $d_{1,0}$-block.

Let $\vec{v'}$ be $\vec{v}$ with $v\vec{}_{B_i}$ changed to $1\,1\,1$.

Let $A_s = \mathcal{R}(A_0, \vec{v'})$.

Let d_r be the distance of the code generated by $G_r = [I_k|A_r]$, and let d_s be the distance of the code generated by $G_s = [I_k|A_s]$.

if $max(d_r, d_s) > min(d_r, d_s) \geq 4$ **then**

Let $d = max(d_r, d_s)$.

if $d = d_r$ **then**

Let $A_1 = A_r$.

else

Let $A_1 = A_s$.

if $((d = 6)$ **and** $(m \leq 4))$ **or** $((d = 8)$ **and** $(m \leq 6))$ **or** $((d \geq 10)$ **and** $(m = 1))$ **then**

if $((d = 6)$ **and** $(m = 4))$ **or** $((d = 8)$ **and** $(m = 6))$ **or** $(d \geq 10)$ **or** *MaxExtensionMate*(A_1, d, m) **then**

QuickOrderMatrix$(A_1, 0, A_1)$.

if $A_1 \notin L_1$ **then**

Insert A_1 into L_1.

end if

end if

end if

end for

end if

end for

end

Notice that the only $(2k, k, 4)$ d_1-codes that Algorithm 5.22 does not consider are the codes with greater than six weight 4 words. By considering the weight ≥ 6 words in the input codes, we can also reduce the number of input codes with less than or equal to six weight 4 words that the algorithm needs to consider. As a demonstration of how we can do this, we will conclude this section by showing how we can reduce the number of input

codes with one and only one weight 4 word that Algorithm 5.22 must consider.

In order to show how we can reduce the number of input codes with one and only one weight 4 word that Algorithm 5.22 must consider, we will first prove the following theorem:

Theorem 5.23 Let C_1 be a $(2k, k, d \geq 6)$ self-dual code that does not contain any intersecting weight 6 words (i.e. either $d \geq 8$ or if $d = 6$ then C_1 does not contain any intersecting weight 6 words). Let C_0 be any $(2k, k, 4)$ d_1-code that is an extension mate of some code equivalent to C_1, such that C_0 contains one and only one weight 4 word $\vec{w}$. Then C_0 does not contain any weight 6 words that intersect $\vec{w}$.

Proof We will prove that C_0 does not contain a weight 6 word that intersects $\vec{w}$ by assuming it does and then showing that this leads us to a contradiction.

Let C_1' be the $(2k, k, d \geq 6)$ self-dual code, equivalent to C_1, whose extension mate is C_0. C_1' is generated by a matrix $G_1' = [I_k | A_1']$, where the first row of A_1' is $0\ 1^{d-3}\ 1\ 1\ 0^*$. C_0 is generated by a matrix $G_0 = [I_k | A_0]$, where $A_0 = \mathcal{R}(A_1', \vec{v})$, $\vec{v} = 1\ 1^{d-3}\ 0\ 0\ 0^*$, and the first row of A_0 is $1\ 0^{d-3} 1\ 1\ 0^*$. Let $\vec{u} = 0^* \vec{v}$. Let $\vec{c_0}$ be any codeword in C_0 and let $\vec{c_1}$ be the corresponding codeword in C_1'. Then, by Theorem 5.19, if $\vec{c_0}$ intersects $\vec{u}$ in an even number of positions then $\vec{c_1} = \vec{c_0}$, else if $\vec{u}$ intersects $\vec{u}$ in an odd number of positions then $\vec{c_1} = \vec{c_0} \oplus \vec{c_1}$. We will use this fact to prove that C_0 cannot contain a weight 6 word that intersects the weight 4 word $\vec{w}$ of C. Note that $\vec{u}$ intersects the weight 4 word $\vec{w}$ in the first row of G_0 in exactly one position.

Suppose C_0 contains a weight 6 word $\vec{c_0}$ that intersects $\vec{w}$. Then, since C_0 has distance 4, $\vec{w}$ and $\vec{c_0}$ must intersect as follows:

$$\begin{array}{cccccccccc} \vec{w} & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \vec{c_0} & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

Let us now consider all the different ways the vector $\vec{u}$ (which has weight $d - 2$) can

intersect these two codewords of C_0, keeping in mind that $\vec{u}$ intersects $\vec{w}$ in exactly one position. We will separate these into two cases, based on the distance of C_1'.

Suppose C_1' has distance $d \geq 8$. Then since the codeword $\vec{c_0}$ has weight 6, the corresponding codeword in C_1 must be $\vec{c_1} = \vec{c_0} \oplus \vec{u}$. That is, $\vec{u}$ must intersect $\vec{c_0}$ in an odd number of positions. Suppose $\vec{u}$ intersects $\vec{c_0}$ in five positions:

$$\begin{array}{lcccccccc} \vec{u} & 1 & 1 & 1 & 1 & 1 & 0 & 1^{d-7} & 0^* \\ \vec{c_0} & 1 & 1 & 1 & 1 & 1 & 1 & 0^{d-7} & 0^* \end{array}$$

Then the codeword $\vec{c_1} = \vec{c_0} \oplus \vec{u}$ would have weight $d-6$, which contradicts the assumption that the distance of C_1' is d. Similarly, if $\vec{u}$ intersects $\vec{c_0}$ in three positions, then $\vec{c_1} = \vec{c_0} \oplus \vec{u}$ would have weight $d-2$, which is also a contradiction. Therefore, $\vec{u}$ intersects $\vec{c_0}$ in exactly one position. This gives us two different ways in which $\vec{u}$ can intersect both $\vec{w}$ and $\vec{c_0}$:

	$\vec{w}$	1	1	1	1	0	0	0	0	0^*
	$\vec{c_0}$	1	1	0	0	1	1	1	1	0^*
Case 1:	$\vec{u}$	1	0	0	0	0	0	0	0	$1^{d-3}\,0^*$
Case 2:	$\vec{u}$	0	0	1	0	1	0	0	0	$1^{d-4}\,0^*$

For both of these cases, consider the two codewords of C_1' that correspond to $\vec{w}$ and $\vec{c_0}$:

Case 1:	$\vec{w} \oplus \vec{u}$	0	1	1	1	0	0	0	0	$1^{d-3}\,0^*$
	$\vec{c_0} \oplus \vec{u}$	0	1	0	0	1	1	1	1	$1^{d-3}\,0^*$
Case 2:	$\vec{w} \oplus \vec{u}$	1	1	0	1	1	0	0	0	$1^{d-4}\,0^*$
	$\vec{c_0} \oplus \vec{u}$	1	1	1	0	0	1	1	1	$1^{d-4}\,0^*$

In both cases, the two codewords $\vec{w} \oplus \vec{u}$ and $\vec{c_0} \oplus \vec{u}$ of C_1' sum to a weight 6 word, which contradicts the assumption that the distance of C_1' is $d \geq 8$. Therefore, if the distance of C_1' is $d \geq 8$, then C_0 cannot contain a weight 6 word that intersects the weight 4 word $\vec{w}$.

Suppose C_1' has distance $d = 6$ and does not contain any intersecting weight 6 words. Then, since $\vec{u}$ has weight $d-2 = 4$, and since $\vec{u}$ intersects $\vec{w}$ in exactly one position, there

are exactly eight different ways in which $\vec{u}$ may intersect both $\vec{u}$ and the weight 6 word $\vec{c_0}$:

	$\vec{w}$	1	1	1	1	0	0	0	0	0	0	0	0^n
	$\vec{c_0}$	1	1	0	0	1	1	1	1	0	0	0	0^n
Case 1:	$\vec{u}$	1	0	0	0	1	1	1	0	0	0	0	0^n
Case 2:	$\vec{u}$	1	0	0	0	1	1	0	0	1	0	0	0^n
Case 3:	$\vec{u}$	1	0	0	0	1	0	0	0	1	1	0	0^n
Case 4:	$\vec{u}$	1	0	0	0	0	0	0	0	1	1	1	0^n
Case 5:	$\vec{u}$	0	0	1	0	1	1	1	0	0	0	0	0^n
Case 6:	$\vec{u}$	0	0	1	0	1	1	0	0	1	0	0	0^n
Case 7:	$\vec{u}$	0	0	1	0	1	0	0	0	1	1	0	0^n
Case 8:	$\vec{u}$	0	0	1	0	0	0	0	0	1	1	1	0^n

For each of these cases, consider the two codewords of C_1' that correspond to $\vec{w}$ and $\vec{c_0}$ (keeping in mind that if $\vec{c_0}$ intersects $\vec{u}$ in an even number of positions then the corresponding codeword in C_1' is $\vec{c_0}$, else if $\vec{c_0}$ intersects $\vec{u}$ in an odd number of positions

then the corresponding codeword in C_1' is $\vec{c_0} \oplus \vec{u}$):

Case 1:	$\vec{w} \oplus \vec{u}$	0	1	1	1	1	1	1	0	0	0	0	0*
	$\vec{c_0}$	1	1	0	0	1	1	1	1	0	0	0	0*
Case 2:	$\vec{w} \oplus \vec{u}$	0	1	1	1	1	1	0	0	1	0	0	0*
	$\vec{c_0} \oplus \vec{u}$	0	1	0	0	0	0	1	1	1	0	0	0*
Case 3:	$\vec{w} \oplus \vec{u}$	0	1	1	1	1	0	0	0	1	1	0	0*
	$\vec{c_0}$	1	1	0	0	1	1	1	1	0	0	0	0*
Case 4:	$\vec{w} \oplus \vec{u}$	0	1	1	1	0	0	0	0	1	1	1	0*
	$\vec{c_0} \oplus \vec{u}$	0	1	0	0	1	1	1	1	1	1	1	0*
Case 5:	$\vec{w} \oplus \vec{u}$	1	1	0	1	1	1	1	0	0	0	0	0*
	$\vec{c_0} \oplus \vec{u}$	1	1	1	0	0	0	0	1	0	0	0	0*
Case 6:	$\vec{w} \oplus \vec{u}$	1	1	0	1	1	1	0	0	1	0	0	0*
	$\vec{c_0}$	1	1	0	0	1	1	1	1	0	0	0	0*
Case 7:	$\vec{w} \oplus \vec{u}$	1	1	0	1	1	0	0	0	1	1	0	0*
	$\vec{c_0} \oplus \vec{u}$	1	1	1	0	0	1	1	1	1	1	0	0*
Case 8:	$\vec{w} \oplus \vec{u}$	1	1	0	1	0	0	0	0	1	1	1	0*
	$\vec{c_0}$	1	1	0	0	1	1	1	1	0	0	0	0*

Cases 1, 2, and 5 all contradict the assumption that the distance of C_1' is $d = 6$: in Case 1, $\vec{w} \oplus \vec{u}$ and $\vec{c_0}$ sum to a weight 4 word, and in Cases 2 and 5, $\vec{c_0} \oplus \vec{u}$ are weight 4 words. Cases 3, 4, 6, 7, and 8 all contradict the assumption that C_1' does not contain any intersecting weight $d = 6$ words: in Cases 3, 6, and 8, $\vec{w} \oplus \vec{u}$ and $\vec{c_0}$ are intersecting weight 6 words, and in Cases 4 and 7, $\vec{w} \oplus \vec{u}$ and $(\vec{w} \oplus \vec{u}) \oplus (\vec{c_0} \oplus \vec{u})$ are intersecting weight 6 words. Therefore, if the distance of C_1' is $d = 6$ and if C_1' does not contain any intersecting weight 6 words, then C_0 cannot contain a weight 6 word that intersects the weight 4 word $\vec{w}$.

Thus, if C_1 is a $(2k, k, d \geq 6)$ self-dual code that does not contain any intersecting weight 6 words, and if C_0 is a $(2k, k, 4)$ d_1-code that is an extension mate of some code

that is equivalent to C_1, where C_0 contains one and only one weight 4 word, then C_0 does not contain any weight 6 words that intersect the weight 4 word. □

In terms of Algorithm 5.22, what Theorem 5.23 tells us is that for any $(2k, k, 4)$ d_1-code C_0, where C_0 contains exactly one weight 4 word and C_0 contains a weight 6 word that intersects the weight 4 word, then if C_0 is an extension mate of a $(2k, k, d \geq 6)$ self-dual code C_1, then $d = 6$ and C_0 contains intersecting weight 6 words. However, any pair of intersecting weight 6 words in a $d = 6$ code form an $F_{6,2}$-block:

$$F_{6,2} = \begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

Therefore, by Theorem 5.21, there exists a $(2k, k, 4)$ d_1-code C_0' that is an extension mate of a code C_1' that is equivalent to C_1, such that C_0' contains two weight 4 words. Therefore, Algorithm 5.22 does not have to consider any of the $(2k, k, 4)$ d_1-codes that contain one and only one weight 4 word and a weight 6 word that intersects the weight 4 word.

5.5 Results of Enumeration

Based on the ideas presented in this chapter, programs for enumerating the $(2k, k, d \geq 6)$ self-dual codes have been implemented. These programs have been run to enumerate lists of $(2k, k, d \geq 6)$ self-dual codes of length up to and including $2k = 32$. We will conclude this chapter by listing the results of this enumeration.

Table 5.2 gives the results of our enumeration. The information provided in the table is:

- k: the dimension of the self-dual codes enumerated.
- running time: the (user) time it took to enumerate each list. The programs were run on an IBM RS/6000.

- number of input codes: the number of inequivalent $(2k, k, 4)$ d_1-codes.
- number of output codes: the number of $(2k, k, d \geq 6)$ self-dual codes produced.
- average number of equivalent codes: the number of $(2k, k, d \geq 6)$ self-dual codes our algorithm produced divided by the number of inequivalent $(2k, k, d \geq 6)$ self-dual codes.

The programs were run for $k = 11, 12, 13, 14, 15, 16$ (there do not exist $(2k, k, d \geq 6)$ self-dual codes for $k = 1, 2, \ldots, 10$). The number of inequivalent $(2k, k, 4)$ d_1-codes and the number of inequivalent $(2k, k, d \geq 6)$ self-dual codes are both given at the end of Chapter 6.

k	running time	number of input codes	number of output codes	average number of equivalent codes
11	0.4 sec	1	1	1
12	0.4 sec	4	3	1.5
13	12.8 sec	7	2	2
14	2.5 min	28	7	2.33
15	19.0 min	98	32	2.46
16	7.8 hrs	802	215	2.62

Figure 5.2: results of enumeration of the $(2k, k, d \geq 6)$ self-dual codes.

Chapter 6

Finding A Unique Representative For Each Equivalence Class of $(2k, k)$ Self-Dual Codes

In the last two chapters we developed algorithms for enumerating the $(2k, k, 4)$ self-dual codes and the $(2k, k, d \geq 6)$ self-dual codes. In Chapter 4, we developed an algorithm that produces a list L_1 of $k \times k$ matrices A, where $G = [I_k|A]$ generates a $(2k, k, 4)$ self-dual code, such that for any $(2k, k, 4)$ self-dual code C there exists at least one A in L_1 such that $G = [I_k|A]$ generates a code equivalent to C. In Chapter 5, we developed an algorithm that produces a list L_2 of $k \times k$ matrices A, where $G = [I_k|A]$ generates a $(2k, k, d \geq 6)$ self-dual code, such that for any $(2k, k, d \geq 6)$ self-dual code C there exists at least one A in L_2 such that $G = [I_k|A]$ generates a code equivalent to C. However, the lists produced by both algorithms may contain many pairs of matrices, A_1 and A_2, in which $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes. We are really interested in producing a list L of $k \times k$ matrices such that for any $(2k, k)$ self-dual code C, there exists one and only one A in L such that $G = [I_k|A]$ generates a code equivalent to C. Therefore, we would like an algorithm that will allow us to determine whether or not two

matrices $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent $(2k, k)$ self-dual codes.

In this chapter, we will develop an algorithm that, given any $k \times k$ matrix A, where $G = [I_k|A]$ generates a $(2k, k)$ self-dual code C, produces a unique matrix A_0, where $G_0 = [I_k|A_0]$ generates a code equivalent to C. That is, we will develop an algorithm that, for any pair of matrices A_1 and A_2, where $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ generate equivalent codes, produces the same $k \times k$ matrix A_0 when run on either one of A_1 or A_2. The algorithm will also determine the size of the automorphism group of C. We will refer to the unique matrix A_0 produced by the algorithm as the *unique representative* for the equivalence class of codes which C belongs to.

The algorithm we develop in this chapter can be used to find a list L of $k \times k$ matrices such that for any $(2k, k, d \geq 4)$ self-dual code C there exists one and only one A in L such that $G = [I_k|A]$ generates a code equivalent to C, by running the algorithm on the lists L_1 and L_2 (produced by the algorithms we developed in the last two chapters) and recording every distinct matrix produced in L. As we shall see later, once we have produced our list L, we can then use the sizes of the automorphism groups of the codes we produced to aid us in verifying the results of our enumeration.

Section 6.1 will review some of the definitions, notation, and concepts we will use throughout this chapter. In Section 6.2, we will discuss the simple, but impractical, approach of finding a unique representative and counting the size of the automorphism group by performing all $(2k)!$ permutations of $G = [I_k|A]$. Two general ideas are then introduced on how we can find a unique representative and count the size of the automorphism group while only performing a subset of the permutations of the columns of $G = [I_k|A]$. The algorithm we will develop in this chapter is based on the ideas discussed in this section.

In Section 6.3, we will develop a two stage method for performing the permutations of the columns of $G = [I_k|A]$ that result in matrices that row-reduce to a matrix with the form $[I_k|A']$. In Section 6.4, we will use this two stage method for performing permutations

to develop our algorithm for finding a unique representative and counting the size of the automorphism group of any class of $(2k, k)$ self-dual codes. As it turns out, this algorithm becomes impractical at around $k = 12$. Therefore, in Sections 6.5–6.9, we will look at improvements we can make to our algorithm for specific classes of $(2k, k)$ self-dual codes.

As we shall see, for the decomposable codes, and the indecomposable distance 2 and e_{2i}-codes, we have no need for our unique representative algorithm. Therefore, in Section 6.5 we will find formulas for the size of the automorphism group for each of these classes of self-dual codes. In Sections 6.6–6.8, we will look at ways we can improve our algorithm for finding a unique representative and counting the size of the automorphism group for the remaining classes of $(2k, k, 4)$ self-dual codes: the indecomposable e_3-codes and d_i-codes. In Section 6.9 we will look at ways we can improve our algorithm for the remaining classes of self-dual codes: the $(2k, k, d \geq 6)$ self-dual codes.

We will conclude this chapter by discussing, in Section 6.10, how we can use the sizes of the automorphism groups we computed to check the results of our enumeration. Finally, in Section 6.11, we will give the results of running our unique representative algorithm on the lists L_1 and L_2 of self-dual codes produced by our enumeration algorithms from the previous two chapters.

6.1 Definitions, Concepts, and Notation

In this section, we will review some of the definitions, concepts, and notation that we will use throughout this chapter in developing our algorithm for finding a unique representative for any equivalence class of $(2k, k)$ self-dual codes.

We will begin by reviewing some of our permutation conventions and notation. Let G be a generator matrix for a $(2k, k)$ self-dual code C. We reference the columns of G

by the sequence of integers $1, 2, 3, \ldots, 2k$. Let

$$\pi = \begin{pmatrix} 1 & 2 & 3 & \cdots & 2k \\ b_1 & b_2 & b_3 & \cdots & b_{2k} \end{pmatrix}$$

be a permutation of the columns of G. We denote the matrix that results from applying π to G by $\pi(G)$. Our convention for carrying out π is that column b_i of G becomes column i of $\pi(G)$. Our convention for products of permutations is they are carried out from right to left. That is, $\pi_1\pi_2(G)$ means $\pi_1(\pi_2(G))$. We will perform our permutations as a product of disjoint cycles. Whenever we are interested in keeping track of what happens to the columns of G we will label the columns of G with the sequence $c_1, c_2, \ldots, c_{2k}$. Whenever we are interested in keeping track of the rows of G we will label the rows of G with the sequence $r_1, r_2, \ldots, r_k$.

When we are working with matrices of the form $G = [I_k|A]$ we will usually describe permutations of the columns of G in terms of the columns of I_k and/or the columns of A. Whenever this is the case, both the columns of I_k and the columns of A will be referenced by the sequence of integers $1, 2, 3, \ldots, k$. Whenever we are interested in keeping track of what happens to the columns of I_k and/or A we will label the columns of I_k with the sequence $c_1, c_2, \ldots, c_k$ and the columns of A with the sequence $a_1, a_2, \ldots, a_k$. This completes our review of our permutation conventions and notation.

We will now review some of the definitions, concepts, and notation we will use regarding equivalence classes of $(2k, k)$ self-dual codes. Let G be a generator matrix for a $(2k, k)$ self-dual code. In this chapter we will mainly be interested in the row-reduced echelon form of G. We will use the notation $\mathcal{RREF}(G)$ to denote the row-reduced echelon form of G.

Example 6.1 Let C be the $(6,3)$ self-dual code generated by:

$$G = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \end{matrix}$$

Let $\pi = (1\ 6\ 3\ 5)$ be a permutation of the columns of G. Then:

$$\pi(G) = \begin{array}{c} \begin{matrix} c_6 & c_2 & c_5 & c_4 & c_1 & c_3 \end{matrix} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \end{matrix} \qquad \mathcal{RREF}(\pi(G)) = \begin{array}{c} \begin{matrix} c_6 & c_2 & c_5 & c_4 & c_1 & c_3 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} r_3 \\ r_2 \\ r_1 \end{matrix}$$

The row-reduced echelon form of G is unique.

We can use the row-reduced echelon form of generator matrices to define equivalence of codes. Let C_1 and C_2 be any two $(2k,k)$ self-dual codes, generated by G_1 and G_2, respectively. C_1 and C_2 are *equivalent* if and only if there exists a permutation π_1 of the columns of C_1 such that $\mathcal{RREF}(\pi_1(G_1)) = \mathcal{RREF}(G_2)$. Let C be any $(2k,k)$ self-dual code. The *equivalence class* of C, which we will denote by $\mathcal{C}(C)$, is the set of all codes that are equivalent to C. By definition of equivalence, if C_1 and C_2 are equivalent codes then $\mathcal{C}(C_1) = \mathcal{C}(C_2)$, and if C_1 and C_2 are inequivalent then $\mathcal{C}(C_1) \cap \mathcal{C}(C_2) = \emptyset$.

Let C be any $(2k,k)$ self-dual code. The set of row-reduced generator matrices that generates the codes in $\mathcal{C}(C)$ will be denoted by $\mathcal{G}(C)$.

Example 6.2 Let C be the $(4,2)$ self-dual code generated by:

$$G = \left[\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right]$$

$$\mathcal{G}(C) = \left\{ \left[\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right], \left[\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right], \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right] \right\}$$

For any code $C' \in \mathcal{C}(C)$ there is one and only one $G' \in \mathcal{G}(C)$ such that G' generates C'.

In terms of the set $\mathcal{G}(C)$, our goal in this chapter is to develop an algorithm which, given any $G \in \mathcal{G}(C)$, produces some unique $G_0 \in \mathcal{G}(C)$. We will also want to have the algorithm determine the size of the *automorphism group* of C.

Let C be any $(2k, k)$ self-dual code and let G be the row-reduced generator matrix for C. The *automorphism group* of C, which we will denote by $Aut(C)$, is the set of all permutations π of the columns of G such that $\mathcal{RREF}(\pi(G)) = G$. For any $G' \in \mathcal{G}(C)$, there are exactly $|Aut(C)|$ permutations π of the columns of G such that $\mathcal{RREF}(\pi(G)) = G'$. Since the total number of permutations of the columns of G is $(2k)!$, this implies that the number of codes in $\mathcal{C}(C)$ is $(2k)!/|Aut(C)|$. In other words, $|Aut(C)|$ can be used to find $|\mathcal{C}(C)|$. Note that this is really just Lagrange's Theorem. Finding $|\mathcal{C}(C)|$ will prove useful in verifying the *correctness* of the implementation of our algorithm.

Example 6.3 Consider the $(4, 2)$ self-dual code C of Example 6.2, which is generated by the matrix:

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

The automorphism group of C is:

$$Aut(C) = \{I, (1\ 3), (2\ 4), (1\ 3)(2\ 4), (1\ 2)(3\ 4), (1\ 2\ 3\ 4), (1\ 4\ 3\ 2), (1\ 4)(2\ 3)\}$$

Since $|Aut(C)| = 8$, we know that the number of codes in $\mathcal{C}(C)$ must be $4!/8 = 24/8 = 3$.

This completes our review of some of the definitions, concepts, and notation that we will use throughout this chapter.

6.2 Finding a Unique Representative for any Equivalence Class of Self-Dual Codes

Let $\mathcal{C}(C)$ be any equivalence class of $(2k, k)$ self-dual codes. Our goal in this chapter is to develop an algorithm that, given any generator matrix $G \in \mathcal{G}(C)$, produces a unique $G_0 \in \mathcal{G}(C)$. In other words, we want an algorithm which, for any two matrices $G_1, G_2 \in \mathcal{G}(C)$, produces permutations π_1 and π_2 of the columns of G_1 and G_2, respectively, such that $\mathcal{RREF}(\pi_1(G_1)) = \mathcal{RREF}(\pi_2(G_2)) = G_0$. We will also want the algorithm to determine the size of the automorphism group of C. In this section, we will first discuss a straightforward (but impractical) algorithm for finding both a unique representative G_0 and the size of the automorphism group. We will then conclude the section by discussing two ideas for improving the algorithm. We will use both of these ideas in the algorithms we will develop later in this chapter.

Our straightforward (but impractical) algorithm is to perform all $(2k)!$ permutations of the columns of G, producing the set of generator matrices $\mathcal{G}(C)$, from which our unique representative G_0 can be selected. Which matrix to select as G_0 can be determined by defining some set of properties $\mathcal{P}$ on generator matrices for self-dual codes, in such a way that for any equivalence class $\mathcal{C}(C)$ of self-dual codes, there exists one and only one $G_0 \in \mathcal{G}(C)$ with every property in $\mathcal{P}$. For example, we can define an ordering on generator matrices and select as G_0 the *largest* matrix in $\mathcal{G}(C)$. The size of the automorphism group $Aut(C)$ can be determined by counting the number of permutations of the columns of G that produce a specific matrix in $\mathcal{G}(C)$. For example, we can count the number of permutations that produce the input matrix G. This lead us to the following algorithm:

Algorithm 6.4 :

- Input: A generator matrix G for a $(2k, k)$ self-dual code C .
- Output: A unique $G_0 \in \mathcal{G}(C)$ and *acnt*, the size of C's automorphism group.

begin

Let $\mathcal{G} = \emptyset$ and let *acnt* $= 0$.

Let Π_{2k} be the set of all $(2k)!$ permutations of the columns of G.

for each $\pi \in \Pi_{2k}$ **do**

Let $G' = \mathcal{RREF}(\pi(G))$.

if $G' \notin \mathcal{G}$ **then** insert G' into $\mathcal{G}$.

if $G' = \mathcal{RREF}(G)$ **then** increment *acnt* by 1.

end for

Return *acnt* and the unique $G_0 \in \mathcal{G}$ that $\mathcal{P}$ determines.

end

The reason this algorithm is impractical is the fact that we perform all $(2k)!$ permutations of the columns of G, and $(2k!)$ is too large a number and requires too much time. So, let us now discuss two methods we can use to reduce the number of permutations of the columns of G that Algorithm 6.4 needs to perform in order to find the unique representative G_0 and the size of the automorphism group *acnt*.

The first idea is to look for ways in which we can reduce the number of permutations of G that result in duplicate matrices being produced. Let Π_{2k} be the set of $(2k)!$ permutations of the columns of G. Then for any $G' \in \mathcal{G}(C)$ there are exactly $|Aut(C)|$ permutations $\pi \in \Pi_{2k}$ such that $\mathcal{RREF}(\pi(G)) = G'$. However, to find $\mathcal{G}(C)$, and thus G_0, we only need to perform one of the permutations of the columns of G that results in G'. Therefore, if for some subset of permutations $\Pi \subseteq \Pi_{2k}$ we can determine beforehand (i.e. without performing each permutation $\pi \in \Pi$ of the columns of G) that for any pair of permutations $\pi_1, \pi_2 \in \Pi$ we have $\mathcal{RREF}(\pi_1(G)) = \mathcal{RREF}(\pi_2(G))$, then we need only perform one of the permutations $\pi \in \Pi$ of the columns of G.

Example 6.5 Let C be the $(6,3)$ self-dual code generated by:

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

Consider the set of permutations

$$\Pi = \left\{ \begin{pmatrix} 1 & 2 & 3 \\ a & b & c \end{pmatrix} \begin{pmatrix} 4 & 5 & 6 \\ a+3 & b+3 & c+3 \end{pmatrix} \right\}$$

where $a\ b\ c$ is a permutation of 1 2 3. The permutations in Π are automorphisms of C. Therefore, to find the set $\mathcal{G}(C)$, and thus G_0, for the subset of permutations $\Pi \subseteq \Pi_6$, we only need to perform one of the permutations $\pi \in \Pi$ of the columns of G.

Note that when we reduce the number of permutations of the columns of G by eliminating permutations that result in duplicates, we may have to compute the number of permutations that we do not perform in order to count the size of the automorphism group.

The second idea for reducing the number of permutations of the columns of G we need to perform is to look for ways in which we can reduce the number of different matrices produced. As given, Algorithm 6.4 produces every matrix in the set $\mathcal{G}(C)$. However, we are only interested in the one unique $G_0 \in \mathcal{G}(C)$ that possesses all the properties in $\mathcal{P}$. Therefore, if for some subset of permutations $\Pi \subseteq \Pi_{2k}$ we can determine beforehand (i.e. without performing each permutation $\pi \in \Pi$ of the columns of G) that for any $\pi \in \Pi$, the matrix $\mathcal{RREF}(\pi(G))$ does not possess all the properties in $\mathcal{P}$ (i.e. is not G_0), then we do not have to perform any of the permutations $\pi \in \Pi$ of the columns of G.

Example 6.6 Let C be the $(8,4)$ self-dual code generated by:

$$G = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 \end{matrix} \\ \left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{matrix}$$

Suppose one of our properties in $\mathcal{P}$ is that G_0 has the form $[I_k|A_0]$. Let π be any permutation of the columns of G such that $G' = \mathcal{RREF}(\pi(G))$ does not have the form $[I_k|A]$. Consider the set of permutations Π consisting of all permutations of the form $\pi_I\pi_A\pi$, where π_I is a permutation of the first 4 columns of G' and π_A is a permutation of the last 4 columns of G'. Then for any $\pi' \in \Pi$, since G' does not have the form $[I_k|A]$ and since π_I only permutes the first 4 columns of $\pi(G)$ and π_A only permutes the last 4 columns of $\pi(G)$, $\mathcal{RREF}(\pi'(G))$ will not have the form $[I_k|A]$. Therefore, we do not have to perform any of the permutations $\pi' \in \Pi$ of the columns of G. For example, if $\pi = (4\ 5)$ then

$$\mathcal{RREF}(\pi(G)) = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_5 & c_4 & c_6 & c_7 & c_8 \end{matrix} \\ \left[\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{matrix}$$

does not have the form $[I_k|A]$. Therefore, any permutation $\pi_I\pi_A(4\ 5)$ of the columns of G, where π_I is a permutation of columns $1,2,3,4$ and π_A is a permutation of columns $5,6,7,8$, will also result in a matrix that does not have the form $[I_k|A]$. Therefore, to find our unique representative G_0, we do not have to perform any permutation of the form $\pi_I\pi_A(4\ 5)$ of the columns of G.

In the next two sections we will progressively work toward developing an algorithm for finding a unique representative G_0 for any equivalence class of $(2k,k)$ self-dual codes.

In later sections, we will look at ways we can alter the algorithm in order to develop algorithms for finding a unique representative G_0 for specific equivalence classes of $(2k, k)$ self-dual codes, namely, the $(2k, k, 4)$ self-dual codes and the $(2k, k, d \geq 6)$ self-dual codes. In their most basic form, the algorithms we will develop are all essentially instances of Algorithm 6.4 together with the two ideas discussed in this section for reducing the number of permutations of the columns of the input matrix G that we need to perform.

6.3 Enumerating the Subset of Generator Matrices with the form $G = [I_k|A]$ in $\mathcal{G}(C)$

Let C be any $(2k, k)$ self-dual code. Let $\mathcal{G}_{[I|A]}(C)$ denote the set of all generator matrices of the form $G = [I_k|A]$ where G generates a code equivalent to C. That is, $\mathcal{G}_{[I|A]}(C)$ is the subset of generator matrices $G \in \mathcal{G}(C)$ where G has the form $[I_k|A]$. Since for any equivalence class $\mathcal{C}$ of $(2k, k)$ self-dual codes there exists at least one $C_0 \in \mathcal{C}$ where C_0 is generated by a matrix of the form $G_0 = [I_k|A_0]$, we know $\mathcal{G}_{[I|A]}(C) \neq \emptyset$. Therefore, to find a unique representative for C, instead of producing the entire set $\mathcal{G}(C)$, we need only produce the subset $\mathcal{G}_{[I|A]}(C)$ from which we can select our unique representative G_0.

As with the set $\mathcal{G}(C)$ we can find the set $\mathcal{G}_{[I|A]}(C)$ by simply performing each of the $(2k)!$ permutations $\pi \in \Pi_{2k}$ of the columns of G. However, since there may be many permutations $\pi \in \Pi_{2k}$ in which $\mathcal{RREF}(\pi(G)) \notin \mathcal{G}_{[I|A]}(C)$, by performing every permutation $\pi \in \Pi_{2k}$ we may be doing a lot of unnecessary work. In this section, we will develop an algorithm that, given any $G \in \mathcal{G}_{[I|A]}(C)$, produces the set $\mathcal{G}_{[I|A]}(C)$ while avoiding much of this unnecessary work. The algorithm can also be used to find the size of the automorphism group of C. The algorithm is based on a two stage method for performing each of the permutations $\pi \in \Pi_{2k}$ of the columns of G. Performing our permutations using this two stage method will allow us to avoid performing many of the permutations $\pi \in \Pi_{2k}$ that result in a matrix that is not in $\mathcal{G}_{[I|A]}(C)$. It will also allow

us to avoid performing some of the permutations $\pi \in \Pi_{2k}$ that result in the same matrix.

We will begin this section by first describing the two stage method for performing each of the permutations $\pi \in \Pi_{2k}$ of the columns of G. We will then use this two stage method to develop our algorithm for producing the set $\mathcal{G}_{[I|A]}(C)$. Since we will only be working with generator matrices of the form $G = [I_k|A]$, we only need to work with the $k \times k$ matrix A. Therefore, we will then rewrite our algorithm for producing the set $\mathcal{G}_{[I|A]}(C)$ in terms of the $k \times k$ matrix A and the three operations $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c.

In the next section, we will develop an algorithm for finding a unique representative for any equivalence class of self-dual codes, and in later sections we will develop algorithms for finding a unique representative for specific equivalence classes of self-dual codes (i.e. the distance 4 and the distance ≥ 6 self-dual codes). Each of these algorithms will make use of the basic ideas behind the algorithm for enumerating the set $\mathcal{G}_{[I|A]}(C)$ (and determining the size of the automorphism group of C) we will develop in this section. Therefore, we will conclude this section by stating the basic properties of the algorithm developed in this section that are used by each of the algorithms we will develop in later sections for finding a unique representative.

6.3.1 A Two Stage Method for Performing the Permutations in Π_{2k}

Let $G = [I_k|A]$ be the generator matrix for a $(2k, k)$ self-dual code C. Let Π_{2k} denote the set of all $(2k)!$ permutations of the columns of G. In order to develop our two stage method for performing each of the permutations $\pi \in \Pi_{2k}$ we will first partition the set Π_{2k} based on the set of k columns of G that each π permutes to the first k columns of $\pi(G)$. Let S_I denote the set of the first k columns of G. Let S_A denote the set of the last k columns of G. Then the set Π_{2k} of all $(2k)!$ permutations of the columns of G can be partitioned into $\binom{2k}{k}$ disjoint sets, each of size $(k!)^2$, as follows:

$$\Pi_{2k} = \{\Pi(T_I, T_A) \mid T_I \subseteq S_I, T_A \subseteq S_A, 0 \leq |T_I| = |T_A| \leq k\}, \text{ where}$$

$$\Pi(T_I, T_A) = \{\pi \in \Pi_{2k} \mid \text{the first } k \text{ columns of } \pi(G) \text{ are a permutation of the set of columns } (S_I - T_I) \cup T_A\}.$$

In other words, Π_{2k} is partitioned into the $\binom{2k}{k}$ disjoint sets $\Pi(T_I, T_A)$, where $\Pi(T_I, T_A)$ is the subset of permutations $\pi \in \Pi_{2k}$ that permutes the set of columns $(S_I - T_I) \cup T_A$ of G to the first k columns of $\pi(G)$.

Example 6.7 Let C be the $(4, 2)$ self-dual code generated by:

$$G = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

Then the set Π_4 of all $4! = 24$ permutations of the columns of G can be partitioned into the following $\binom{4}{2} = 6$ disjoint sets of permutations:

$$\begin{aligned} \Pi(\emptyset, \emptyset) &= \{I, (1\ 2), (3\ 4), (1\ 2)(3\ 4)\} \\ \Pi(\{1\}, \{3\}) &= \{(1\ 3), (1\ 2\ 3), (1\ 3\ 4), (1\ 2\ 3\ 4)\} \\ \Pi(\{1\}, \{4\}) &= \{(1\ 4), (1\ 2\ 4), (1\ 4\ 3), (1\ 2\ 4\ 3)\} \\ \Pi(\{2\}, \{3\}) &= \{(2\ 3), (1\ 3\ 2), (2\ 3\ 4), (1\ 3\ 4\ 2)\} \\ \Pi(\{2\}, \{4\}) &= \{(2\ 4), (1\ 4\ 2), (2\ 4\ 3), (1\ 4\ 3\ 2)\} \\ \Pi(\{1,2\}, \{3,4\}) &= \{(1\ 3)(2\ 4), (1\ 4\ 2\ 3), (1\ 3\ 2\ 4), (1\ 4)(2\ 3)\} \end{aligned}$$

where each set $\Pi(T_I, T_A)$ consists of all permutations π of the columns of G such that the first 2 columns of $\pi(G)$ are a permutation of the set of 2 columns $(\{1,2\} - T_I) \cup T_A$. For example, the 4 matrices that result from performing each of the 4 permutations in the set $\Pi(\{1\}, \{3\})$ are:

$$\begin{array}{c} \begin{matrix} c_3 & c_2 & c_1 & c_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array} \quad \begin{array}{c} \begin{matrix} c_2 & c_3 & c_1 & c_4 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{array} \quad \begin{array}{c} \begin{matrix} c_3 & c_2 & c_4 & c_1 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} \begin{matrix} c_2 & c_3 & c_4 & c_1 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

The first 2 columns in each of these matrices are a permutation of columns $(\{1,2\} - \{1\}) \cup \{3\} = \{2,3\}$ of G.

Let us now consider how we can perform each of the $(k!)^2$ permutations of the columns of G in the sets $\Pi(T_I, T_A)$. Let $\Pi(T_I, T_A)$ be any one of our $\binom{2k}{k}$ subsets of Π_{2k}. Let π be any permutation in the set $\Pi(T_I, T_A)$. Let $\Pi_I(\pi)$ denote the set of all $k!$ permutations of the first k columns of $\pi(G)$ and let $\Pi_A(\pi)$ denote the set of all $k!$ permutations of the last k columns of $\pi(G)$. Then $\Pi(T_I, T_A) = \{\pi_I \pi_A \pi \mid \pi_I \in \Pi_I(\pi), \pi_A \in \Pi_A(\pi)\}$. In other words, given any permutation $\pi \in \Pi(T_I, T_A)$, we can produce every permutation $\pi' \in \Pi(T_I, T_A)$ of the columns of G by performing all $(k!)^2$ permutations of the first k columns and the last k columns of $\pi(G)$.

Example 6.8 Let G be the generator matrix for the $(4,2)$ self-dual code in Example 6.7. Consider the set of permutations $\Pi(\{1\},\{3\}) = \{(1\ 3),(1\ 2\ 3),(1\ 3\ 4),(1\ 2\ 3\ 4)\}$ of the columns of G. If we are given any permutation $\pi \in \Pi(\{1\},\{3\})$, then we can use π to produce each of the 4 permutations in $\Pi(\{1\},\{3\})$. For example, if $\pi = (1\ 3)$, then if we perform each of the $(2!)(2!) = 4$ permutations of the first 2 columns and last 2 columns of $\pi(G)$, the corresponding permutations on the columns of G are:

$$\begin{aligned}
&\{\pi, (1\ 2)\pi, (3\ 4)\pi, (1\ 2)(3\ 4)\pi\} \\
&\quad = \{(1\ 3),(1\ 2)(1\ 3),(3\ 4)(1\ 3),(1\ 2)(3\ 4)(1\ 3)\} \\
&\quad = \{(1\ 3),(1\ 2\ 3),(1\ 3\ 4),(1\ 2\ 3\ 4)\} \\
&\quad = \Pi(\{1\},\{3\})
\end{aligned}$$

This observation leads us to our two stage method for performing each of the permutations in Π_{2k}.

The first stage of our two stage method for performing the permutations in Π_{2k} is to select one permutation π from each of the $\binom{2k}{k}$ sets of permutations $\Pi(T_I, T_A)$. Selecting a permutation from each set $\Pi(T_I, T_A)$ can be accomplished in a straightforward

manner. For example, if $T_I = \{u_0, u_1, \ldots, u_m\}$ and $T_A = \{v_0, v_1, \ldots, v_m\}$, then $\pi = (u_0\ v_0)(u_1\ v_1)\ldots(u_m\ v_m)$ is an element of $\Pi(T_I, T_A)$. The second stage of our two stage method is to perform each of the $(k!)^2$ permutations of the columns of G, in each set of permutations $\Pi(T_I, T_A)$, by performing each of the $(k!)^2$ permutations of the first k and the last k columns of $\pi(G)$, where π is the permutation in $\Pi(T_I, T_A)$ selected in the first stage.

Thus, our two stage method for performing all $(2k)!$ permutations of the columns of G works by first selecting $\binom{2k}{k}$ permutations of the columns of G, and then uses each permutation selected to perform $(k!)^2$ of the $(2k)!$ permutations of the columns of G. We will use this two stage method in our algorithm for producing the set $\mathcal{G}_{[I|A]}(C)$ of all row-reduced generator matrices of the form $[I_k|A]$ that generate codes equivalent to C. As we shall see, the advantage in using the two stage method in our algorithm is that it will allow us to produce the set $\mathcal{G}_{[I|A]}(C)$, and to find the size of the automorphism group of C, without having to perform the $(k!)^2$ permutations of every set of permutations $\Pi(T_I, T_A)$.

One final point on performing permutations of the columns of G is that if $\pi = \pi_1\pi_2$ is any permutation of the columns of G, then $\mathcal{RREF}(\pi_1\pi_2(G)) = \mathcal{RREF}(\pi_1(\mathcal{RREF}(\pi_2(G))))$.

6.3.2 Enumerating the Set $\mathcal{G}_{[I|A]}(C)$

We are now ready to develop our algorithm which, given any $G \in \mathcal{G}_{[I|A]}(C)$, produces the set of generator matrices $\mathcal{G}_{[I|A]}(C)$. The algorithm will use our two stage method for performing each permutation in the set Π_{2k} of all $(2k)!$ permutations of the columns of G. That is, the algorithm will first select/perform one permutation π from each of the $\binom{2k}{k}$ sets of permutations $\Pi(T_I, T_A)$, producing a list L' of generator matrices $G' = \mathcal{RREF}(\pi(G))$. For each $G' \in L'$, the algorithm will then perform each of the $(k!)^2$ permutations of the first k and the last k columns of G'. In other words, for each

$G' = \mathcal{RREF}(\pi(G)) \in L'$, the algorithm will use $\mathcal{RREF}(\pi(G))$ to perform each of the $(k!)^2$ permutations of the columns of G in the set $\Pi(T_I, T_A)$, where $\Pi(T_I, T_A)$ is the set of permutations of the columns of G from which the permutation π was selected in the first stage of the algorithm.

One advantage in using the two stage method to perform permutations in our algorithm is that whenever we select a permutation π in the first stage of our algorithm in which $G' = \mathcal{RREF}(\pi(G))$ does not have the form $[I_k|A]$, then we do not have to insert G' into our list L'. The reason for this is that for any set of permutations $\Pi(T_I, T_A)$, either $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A]$ for all $\pi \in \Pi(T_I, T_A)$, or $\mathcal{RREF}(\pi(G))$ does not have the form $[I_k|A]$ for all $\pi \in \Pi(T_I, T_A)$, as the next theorem demonstrates.

Theorem 6.9 Let π_1 and π_2 be any two permutations in the set $\Pi(T_I, T_A)$. Then $\mathcal{RREF}(\pi_1(G))$ has the form $[I_k|A]$ if and only if $\mathcal{RREF}(\pi_2(G))$ has the form $[I_k|A]$.

Proof Suppose $\mathcal{RREF}(\pi_1(G))$ has the form $[I_k|A]$. Then since π_1 and π_2 are both elements of $\Pi(T_I, T_A)$, the first k columns of $\pi_2(G)$ are a permutation of the first k columns of $\pi_1(G)$, and therefore $\mathcal{RREF}(\pi_2(G))$ must also have the form $[I_k|A]$. Similarly, if $\mathcal{RREF}(\pi_1(G))$ does not have the form $[I_k|A]$, then since the first k columns of $\pi_2(G)$ are a permutation of the first k columns of $\pi_1(G)$, $\mathcal{RREF}(\pi_2(G))$ does not have the form $[I_k|A]$. This completes our proof. □

Therefore, for each set of permutations $\Pi(T_I, T_A)$, if π is the permutation in $\Pi(T_I, T_A)$ that we select in the first stage of our algorithm, then if $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A]$, then for each of the $(k!)^2$ permutations $\pi' \in \Pi(T_I, T_A)$, the matrix $\mathcal{RREF}(\pi'(G))$ must also have the form $[I_k|A]$, and if $\mathcal{RREF}(\pi(G))$ does not have the form $[I_k|A]$, then for each of the $(k!)^2$ permutations $\pi' \in \Pi(T_I, T_A)$, the matrix $\mathcal{RREF}(\pi'(G))$ does not have the form $[I_k|A]$. Therefore, since we are only interested in producing the subset $\mathcal{G}_{[I|A]}(C)$ of generator matrices in $\mathcal{G}(C)$ with the form $[I_k|A]$, for each permutation π selected in the first stage our algorithm will insert $G' = \mathcal{RREF}(\pi(G))$ into the list L' if and only if G'

has the form $[I_k|A]$. Therefore, the second stage of our algorithm will only perform the permutations of the columns of G that result in a matrix whose row-reduced echelon form is $[I_k|A]$. Thus, for each set of permutations $\Pi(T_I, T_A)$ in which for each $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ does not have the form $[I_k|A]$, the second stage of our algorithm avoids performing the $(k!)^2$ unnecessary permutations of the columns of G in the set $\Pi(T_I, T_A)$.

Another advantage in using the two stage method to perform permutations in our algorithm is that whenever a permutation π is selected in the first stage of our algorithm in which $G' = \mathcal{RREF}(\pi(G))$ has already been seen (i.e. is already an element of L'), then we do not have to perform any of the $(k!)^2$ permutations of the first k and last k columns of G', as the next theorem demonstrates:

Theorem 6.10 Let $\Pi(T_{I_1}, T_{A_1})$ and $\Pi(T_{I_2}, T_{A_2})$ be any two of our subsets of permutations of Π_{2k}. Suppose there exist permutations $\pi_1 \in \Pi(T_{I_1}, T_{A_1})$ and $\pi_2 \in \Pi(T_{I_2}, T_{A_2})$ such that $\mathcal{RREF}(\pi_1(G)) = \mathcal{RREF}(\pi_2(G))$. Then the set of row-reduced matrices that result from performing each of the permutations of the columns of G in $\Pi(T_{I_1}, T_{A_1})$ (and then row-reducing) is equal to the set of row-reduced matrices that result from performing each of the permutations of the columns of G in $\Pi(T_{I_2}, T_{A_2})$ (and then row-reducing).

Proof This follows immediately from the fact that for any set $\Pi(T_I, T_A)$ and given any $\pi \in \Pi(T_I, T_A)$, $\Pi(T_I, T_A) = \{\pi_I \pi_A \pi \mid \pi_I$ is a permutation of the first k columns of $\pi(G)$ and π_A is a permutation of the last k columns of $\pi(G)\}$. Therefore, if there exist two permutations $\pi_1 \in \Pi(T_{I_1}, T_{A_1})$ and $\pi_2 \in \Pi(T_{I_2}, T_{A_2})$ such that $\mathcal{RREF}(\pi_1(G)) = \mathcal{RREF}(\pi_2(G))$, then for each pair of permutations $\pi_1' = \pi_I \pi_A \pi_1 \in \Pi(T_{I_1}, T_{A_1})$ and $\pi_2' = \pi_I \pi_A \pi_2 \in \Pi(T_{I_2}, T_{A_2})$, we have:

$$\begin{aligned}
\mathcal{RREF}(\pi_1'(G)) &= \mathcal{RREF}(\pi_I \pi_A \pi_1(G)) \\
&= \mathcal{RREF}(\pi_I \pi_A(\mathcal{RREF}(\pi_1(G)))) \\
&= \mathcal{RREF}(\pi_I \pi_A(\mathcal{RREF}(\pi_2(G)))) \\
&= \mathcal{RREF}(\pi_I \pi_A \pi_2(G))
\end{aligned}$$

$$= \mathcal{RREF}(\pi_2'(G))$$

Thus, the set of row-reduced matrices that results from performing the permutations in each of $\Pi(T_{I_1}, T_{A_1})$ and $\Pi(T_{I_2}, T_{A_2})$ are the same. □

Therefore, if for any two sets, $\Pi(T_{I_1}, T_{A_1})$ and $\Pi(T_{I_2}, T_{A_2})$, there exist permutations $\pi_1 \in \Pi(T_{I_1}, T_{A_1})$ and $\pi_2 \in \Pi(T_{I_2}, T_{A_2})$ such that $\mathcal{RREF}(\pi_1(G)) = \mathcal{RREF}(\pi_2(G))$, then the set of row-reduced generator matrices that result from performing each of the $(k!)^2$ permutations in each set are the same. Thus, in the first stage of our enumeration, if we are fortunate enough to select permutations π_1 and π_2 such that $\mathcal{RREF}(\pi_1(G)) = \mathcal{RREF}(\pi_2(G))$, then we need only insert one of the two matrices $\mathcal{RREF}(\pi_1(G))$ and $\mathcal{RREF}(\pi_2(G))$ into our list L'. Furthermore, we can still find the size of the automorphism group of C. That is, for any matrix $G'' \in \mathcal{G}_{[I|A]}(C)$, we can still compute the total number of permutations π of the columns of G such that $\mathcal{RREF}(\pi(G)) = G''$, without actually performing every such permutation of the columns of G, as the next theorem demonstrates:

Theorem 6.11 Let $\Pi(T_I, T_A)$ be any one of our subsets of permutations of Π_{2k}. Let $\pi \in \Pi(T_I, T_A)$ and let $G'' = \mathcal{RREF}(\pi(G))$. Then the number of permutations $\pi' \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi'(G)) = G''$ is equal to the number of permutations $\pi_I\pi_A$ of the first k and the last k columns of G'' such that $\mathcal{RREF}(\pi_I\pi_A(G'')) = G''$.

Proof Again, this follows from the fact that $\Pi(T_I, T_A) = \{\pi_I\pi_A\pi \mid$ where π_I is a permutation of the first k columns of $\pi(G)$ and π_A is a permutation of the last k columns of $\pi(G)\}$. Therefore, for each permutation $\pi_I\pi_A$ of the columns of G'' such that $\mathcal{RREF}(\pi_I\pi_A(G'')) = G''$, we know that the permutation $\pi' = \pi_I\pi_A\pi$ is an element of $\pi(T_I, T_A)$ and:

$$\begin{aligned} \mathcal{RREF}(\pi'(G)) &= \mathcal{RREF}(\pi_I\pi_A\pi(G)) \\ &= \mathcal{RREF}(\pi_I\pi_A(\mathcal{RREF}(\pi(G)))) \end{aligned}$$

$$= \mathcal{RREF}(\pi_I \pi_A(G''))$$
$$= G''$$

Similarly, for each permutation $\pi' = \pi_I \pi_A \pi \in \Pi(T_I, T_A)$ where $\mathcal{RREF}(\pi'(G)) = G''$, we have $\mathcal{RREF}(\pi_I \pi_A \pi(G)) = \mathcal{RREF}(\pi_I \pi_A(\mathcal{RREF}(\pi(G)))) = \mathcal{RREF}(\pi_I \pi_A(G'')) = G''$. Therefore, the number of permutations $\pi_I \pi_A$ of the first k columns and the last k columns of $G'' = \mathcal{RREF}(\pi(G))$ such that $\mathcal{RREF}(\pi_I \pi_A(G'')) = G''$ is equal to the number of permutations $\pi' \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi'(G)) = G''$. □

Therefore, for any matrix $G'' \in \mathcal{G}_{[I|A]}(C)$, if *pcnt* is the number of permutations $\pi_I \pi_A$ of the first k columns and the last k column of G'' such that $\mathcal{RREF}(\pi_I \pi_A(G'')) = G''$ and if *ccnt* is the number of sets $\Pi(T_I, T_A)$ for which there exists a permutation $\pi' \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi'(G)) = G''$, then the total number of permutations π of the columns of G in which $\mathcal{RREF}(\pi(G)) = G''$ is $pcnt \times ccnt$. Thus, if for each matrix G' inserted into L', we also record the number of permutations we select in the first stage of our algorithm that result in G', then we can still compute the size of the automorphism group of C. That is, if the output to the first stage of our algorithm is a list L' of pairs $(G'_1, cnt'_1), (G'_2, cnt'_2), \ldots, (G'_m, cnt'_m)$, where cnt'_i is the number of times G'_i was produced in the first stage, then for any $G'' \in \mathcal{G}_{[I|A]}(C)$, if *pcnt* is the number of permutations $\pi_I \pi_A$ of the first k columns and the last k columns of G'' such that $\mathcal{RREF}(\pi_I \pi_A(G'')) = G''$, and if $G'_{i_1}, G'_{i_2}, \ldots, G'_{i_n}$ are the matrices in L' for which there exists a permutation π'_{i_j} of the first k columns and the last k columns G'_{i_j} such that $\mathcal{RREF}(\pi'_{i_j}(G'_{i_j})) = G''$, then the number of permutations π of the first k and last k columns of G such that $\mathcal{RREF}(\pi(G)) = G''$ is equal to $pcnt(\sum_{j=1}^{n} cnt'_{i_j})$. Since for any matrix $G'' \in \mathcal{G}_{[I|A]}(C)$, the size of the the automorphism group of C is equal to the number of permutations π of the columns of G in which $\mathcal{RREF}(\pi(G)) = G''$, if we record the number of times each matrix is produced in the first stage of our algorithm, we can still compute the size of the automorphism group of C.

We are now ready to give a formal description our algorithm that, given any $G \in$

$\mathcal{G}_{[I|A]}(C)$, produces the set of matrices $\mathcal{G}_{[I|A]}(C)$. The unique representative G_0 is then selected from $\mathcal{G}_{[I|A]}(C)$. The matrix in $\mathcal{G}_{[I|A]}(C)$ which the algorithm uses to find the size of the automorphism group of C is the input matrix G. That is, the algorithm finds the size of the automorphism group of C by computing the number of permutations of the columns G that result in G being reproduced.

Algorithm 6.12 :

- Input: A generator matrix $G = [I_k|A]$ for a $(2k, k)$ self-dual code C.
- Output: A unique matrix $G_0 = [I_k|A_0]$ selected from the list L'' of all generator matrices of the form $[I_k|A'']$ that generate a code equivalent to C (i.e. $L'' = \mathcal{G}_{[I|A]}(C)$), and *acnt*, the size of the automorphism group of C.

begin

Stage 1:

Let $L' = \emptyset$.

Let S_I be the set of the first k columns of G.

Let S_A be the set of the last k columns of G.

Let $\Pi_1 = \{(T_I, T_A) \mid T_I \subseteq S_I, T_A \subseteq S_A, 0 \leq |T_I| = |T_A| \leq k\}$.

for each $(T_I, T_A) \in \Pi_1$ **do**

Let π be any permutation in the set $\Pi(T_I, T_A)$.

Let $G' = \mathcal{RREF}(\pi(G))$.

if G' has the form $[I_k|A']$ **then**

if $(G', cnt') \in L'$ **then** increment cnt' by one.

else insert $(G', 1)$ into L'.

end if

end for

Stage 2:

Let $acnt = 0$.

Let $L'' = \emptyset$.

for each $(G', cnt') \in L'$ **do**

Let $\Pi_2 = \{(\pi_I, \pi_A) \mid \pi_I$ is a permutation of the first k columns of G' and π_A is a permutation of the last k columns of $G'\}$.

for each $(\pi_I, \pi_A) \in \Pi_2$ **do**

Let $G'' = \mathcal{RREF}(\pi_I \pi_A(G'))$.

if $G'' \notin L''$ **then** insert G'' into L''.

if $G'' = G$ **then** increment $acnt$ by cnt'.

end for

end for

Select a unique matrix $G_0 \in L''$.

Return $acnt$ and G_0.

end

The algorithm computes the product $acnt = pcnt(\sum_{j=1}^{n} cnt'_{i_j})$ by initializing $acnt$ to 0, and for each pair $(G', cnt') \in L'$, adding cnt' to $acnt$ every time a permutation $\pi_I \pi_A$ of the first k and last k columns of G' is performed in which $\mathcal{RREF}(\pi_I \pi_A)(G') = G$. For any G' in L', the number of such permutations will be either 0 or $pcnt$. Thus, if $G'_{i_1}, G'_{i_2}, \ldots, G'_{i_n}$ are the matrices in L' in which the number of such permutations is $pcnt$, then the algorithm computes $\sum_{j=1}^{n} \sum_{x=1}^{pcnt} cnt'_{i_j} = pcnt(\sum_{j=1}^{n} cnt'_i) = acnt$.

Example 6.13 Let us use Algorithm 6.12 to produce L'' and $acnt$ for the $(2k, k)$ self-dual code C of Example 6.7, which is generated by the matrix:

$$G = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} \\ r_1 \\ r_2 \end{matrix}$$

The first stage of our algorithm selects one permutation from each of the 6 sets of permutations $\Pi(\emptyset,\emptyset)$, $\Pi(\{1\},\{3\})$, $\Pi(\{1\},\{4\})$, $\Pi(\{2\},\{3\})$, $\Pi(\{2\},\{4\})$, and $\Pi(\{1,2\},\{3,4\})$. Suppose the permutations selected are I, (1 3), (1 4), (2 3), (2 4), and (1 3 2 4), respectively. Then the 6 row-reduced generator matrices that result from performing each of these 6 permutations of the columns of G are:

$$\mathcal{RREF}(I(G)) = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \end{matrix} \qquad \mathcal{RREF}((1\ 3)(G)) = \begin{array}{c} \begin{matrix} c_3 & c_2 & c_1 & c_4 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \end{matrix}$$

$$\mathcal{RREF}((1\ 4)(G)) = \begin{array}{c} \begin{matrix} c_4 & c_2 & c_3 & c_1 \end{matrix} \\ \left[\begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{matrix}\right] \end{array} \begin{matrix} r_2 \\ r_1 \end{matrix} \qquad \mathcal{RREF}((2\ 3)(G)) = \begin{array}{c} \begin{matrix} c_1 & c_3 & c_2 & c_4 \end{matrix} \\ \left[\begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \end{matrix}$$

$$\mathcal{RREF}((2\ 4)(G)) = \begin{array}{c} \begin{matrix} c_1 & c_4 & c_3 & c_2 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \end{matrix} \qquad \mathcal{RREF}((1\ 3\ 2\ 4)(G)) = \begin{array}{c} \begin{matrix} c_3 & c_4 & c_1 & c_2 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \end{matrix}$$

Only three of these six matrices are distinct:

$$\begin{aligned} G'_1 &= \mathcal{RREF}(I(G)) = \mathcal{RREF}((1\ 3)(G)) = \mathcal{RREF}((2\ 4)(G)) \\ G'_2 &= \mathcal{RREF}((1\ 3\ 2\ 4)(G)) \\ G'_3 &= \mathcal{RREF}((1\ 4)(G)) = \mathcal{RREF}((2\ 3)(G)) \end{aligned}$$

Both G'_1 and G'_2 have the form $[I_k|A]$ while G'_3 does not have the form $[I_k|A]$. Thus, the output from the first stage of our algorithm is the list $L' = \{(G'_1, 3), (G'_2, 1)\}$.

The second stage of our algorithm performs all permutations of the first 2 columns and last 2 columns of every matrix in the list L'. Each distinct matrix produced is recorded in the list L''. The second stage of the algorithm also computes *acnt*, the number of permutations of G that result in G.

Let $acnt = 0$. For the pair $(G'_1, 3)$, the 4 row-reduced matrices that result from performing each of the 4 permutations of the first 2 columns and last 2 columns of G'_1

are:

$$\mathcal{RREF}((I)(G_1')) = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \end{matrix} \qquad \mathcal{RREF}((1\ 2)(G_1')) = \begin{array}{c} \begin{matrix} c_2 & c_1 & c_3 & c_4 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{matrix}\right] \end{array} \begin{matrix} r_2 \\ r_1 \end{matrix}$$

$$\mathcal{RREF}((3\ 4)(G_1')) = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_4 & c_3 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{matrix}\right] \end{array} \begin{matrix} r_1 \\ r_2 \end{matrix} \quad \mathcal{RREF}((1\ 2)(3\ 4)(G_1')) = \begin{array}{c} \begin{matrix} c_2 & c_1 & c_4 & c_3 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}\right] \end{array} \begin{matrix} r_2 \\ r_1 \end{matrix}$$

Only two of these four matrices are distinct:

$$\begin{aligned} G_1'' &= \mathcal{RREF}((I)(G_1')) = \mathcal{RREF}((1\ 2)(3\ 4)(G_1')) \\ G_2'' &= \mathcal{RREF}((1\ 2)(G_1')) = \mathcal{RREF}((3\ 4)(G_1')) \end{aligned}$$

The algorithm will insert both G_1'' and G_2'' into L'' and since $G_1'' = G$, and G_1'' was produced twice, the algorithm will increment $acnt$ by $cnt_1' = 3$ twice. That is, once the algorithm is finished performing the 4 permutations of the columns of G_1', $acnt = 3 + 3 = 6$.

For the pair $(G_2', 1)$, the 4 permutations of the first 2 columns and last 2 columns of G_2' will also result in G_1'' being produced twice, and G_2'' being produced twice. Therefore, none of the matrices produced are inserted into the list L'' and since $G = G_1''$ was produced twice, the algorithm will increment $acnt$ by $cnt_2' = 1$ twice. That is, once the algorithm is finished performing the 4 permutations of the columns of G_2', $acnt = 6 + 1 + 1 = 8$.

Thus, for the $(4, 2)$ self-dual code C, generated by:

$$G = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right]$$

the output of Algorithm 6.12 is the set of all generator matrices of the form $[I_k|A]$ that generate a code equivalent to C:

$$L'' = \left\{ \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right], \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] \right\}$$

and $acnt = 8$, the size of the automorphism group of C.

6.3.3 Using the $k \times k$ Matrix A to Enumerate $\mathcal{G}_{[I|A]}(C)$

Let us now consider how we can rewrite Algorithm 6.12 in terms of the $k \times k$ matrix A and the three operations $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c. Recall that if entry (i,j) of A is 1, then an $\text{ADD}_{r,c}(i,j)$ operation adds (mod 2) row i of A (with a 0 in place of the 1 in column j) to all other rows of A that contain a 1 in column j. The $\text{ADD}_{r,c}(i,j)$ operation is only defined on the non-zero entries (i,j) of A. A $\text{SWAP}_r(i_0,i_1)$ operation swaps rows i_0 and i_1 of A and a $\text{SWAP}_c(j_0,j_1)$ operation swaps columns j_0 and j_1 of A. As we shall see, the first stage of Algorithm 6.12 can be accomplished with the $\text{ADD}_{r,c}$ operation and the second stage of Algorithm 6.12 can be accomplished with the SWAP_r and SWAP_c operations.

Let $G = [I_k|A]$ be the input matrix to Algorithm 6.12. For each of the $\binom{2k}{k}$ sets $\Pi(T_I, T_A)$ of permutations of the columns of G, the first stage of Algorithm 6.12 selects one permutation $\pi \in \Pi(T_I, T_A)$ and produces the matrix $G' = \mathcal{RREF}(\pi(G))$. If G' has the form $[I_k|A']$ it is recorded in the list L', otherwise it is discarded. As we shall see, this stage can be accomplished by applying sequences of $\text{ADD}_{r,c}$ operations to A.

Recall that an $\text{ADD}_{r,c}(i,j)$ on A produces the matrix A' where $[I_k|A'] = \mathcal{RREF}(\pi([I_k|A]))$ where π is the transposition of the columns of $[I_k|A]$ that swaps column j of A with column i of I_k (i.e. the column of I_k that contains a one in row i). If we apply the sequence $\text{ADD}_{r,c}(i_1,j_1), \text{ADD}_{r,c}(i_2,j_2), \ldots, \text{ADD}_{r,c}(i_m,j_m)$ to A where $i_1 \neq i_2 \neq \cdots \neq i_m$ and $j_1 \neq j_2 \neq \cdots \neq j_m$ then the matrix that results is the matrix A' where $[I_k|A'] = \mathcal{RREF}(\pi([I_k|A]))$ where $\pi = (i_1\ j_1)(i_2\ j_2)\ldots(i_m\ j_m)$. Therefore, the $\text{ADD}_{r,c}$ operation can be used to accomplish permutations of the columns of $G = [I_k|A]$ that swap a subset T_I of columns of I_k with an equally sized subset T_A of columns of A. However, since it is only defined on the non-zero entries of A, the $\text{ADD}_{r,c}$ operation cannot be used to accomplish every such permutation of the columns of G. But, as the next theorem demonstrates, for any set $\Pi(T_I, T_A)$ with the property that for any $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$, there exists a sequence

$\text{ADD}_{r,c}(i_1, j_1), \text{ADD}_{r,c}(i_2, j_2), \ldots, \text{ADD}_{r,c}(i_m, j_m)$ of m $\text{ADD}_{r,c}$ operations on A, where $m = |T_I|$, such that $(i_1\ j_1)(i_2\ j_2)\ldots(i_m\ j_m) \in \Pi(T_I, T_A)$.

Theorem 6.14 Let $G = [I_k|A]$ be the generator matrix for a $(2k, k)$ self-dual code C. Let T_I be any set of m columns of I_k and let T_A be any set of m columns of A, $0 \leq m \leq k$. Let T_r be the set of rows of A corresponding to T_I and let $T_c = T_A$. Then for any $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$ if and only if there exists a sequence S of m $\text{ADD}_{r,c}$ operations $\text{ADD}_{r,c}(i_1, j_1), \text{ADD}_{r,c}(i_2, j_2), \ldots, \text{ADD}_{r,c}(i_m, j_m)$ on A where $T_r = \{i_1, i_2, \ldots, i_m\}$ and $T_c = \{j_1, j_2, \ldots, j_m\}$. Note that the sequence S takes A and produces A'.

Proof If for any permutation $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$, then the fact that there exists a sequence of m $\text{ADD}_{r,c}$ operations $\text{ADD}_{r,c}(i_1, j_1), \text{ADD}_{r,c}(i_2, j_2), \ldots, \text{ADD}_{r,c}(i_m, j_m)$ on A, where $T_r = \{i_1, i_2, \ldots, i_m\}$ and $T_c = \{j_1, j_2, \ldots, j_m\}$, is a direct result of Lemma 2.15 in Section 2.4 of Chapter 2, which states the same result, but for the general case of any (n, k) linear code generated by $G = [I_k|A]$.

Conversely, suppose for any $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ does not have the form $[I_k|A']$. Then there cannot exist a sequence S of m $\text{ADD}_{r,c}$ operations $\text{ADD}_{r,c}(i_1, j_1), \text{ADD}_{r,c}(i_2, j_2), \ldots, \text{ADD}_{r,c}(i_m, j_m)$ on A where $T_r = \{i_1, i_2, \ldots, i_m\}$ and $T_c = \{j_1, j_2, \ldots, j_m\}$, for if such a sequence did exist then the matrix A' that results from applying S to A is such that $[I_k|A'] = \mathcal{RREF}(\pi(G))$, where $\pi = (i_1\ j_1)(i_2\ j_2)\ldots(i_m\ j_m)$, which is an element of $\Pi(T_I, T_A)$, which implies that there exists a permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$, which is a contradiction. □

Therefore, for any set of m rows T_r and m columns T_c of A, there exists a sequence S of m $\text{ADD}_{r,c}$ operations $\text{ADD}_{r,c}(i_1, j_1), \text{ADD}_{r,c}(i_2, j_2), \ldots, \text{ADD}_{r,c}(i_m, j_m)$ on A, where $T_r = \{i_1, i_2, \ldots, i_m\}$ and $T_c = \{j_1, j_2, \ldots, j_m\}$, if and only if for any $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$. For a given pair of sets (T_r, T_c), we can determine whether or not such a sequence S of $\text{ADD}_{r,c}$ operations exists, recursively. That is, for a

given matrix A and pair of sets (T_r, T_c), if there exists a pair $(i,j) \in (T_r, T_c)$ such that entry (i,j) of A is 1, then there exists a sequence S of m $\text{ADD}_{r,c}$ operations on A that involves all the elements in (T_r, T_c) if and only if there exists a sequence S' of $m-1$ $\text{ADD}_{r,c}$ operations on $\text{ADD}_{r,c}(i,j)(A)$ where S' involves all the elements in $(T_r - \{i\}, T_c - \{j\})$. Thus, the first stage of Algorithm 6.12 can be accomplished by applying sequences of $\text{ADD}_{r,c}$ operations to A.

Let us now consider the second stage of Algorithm 6.12. Let L' be the list of pairs (G', cnt') produced by the first stage of Algorithm 6.12. For each $G' = [I_k|A']$ in L', the second stage of Algorithm 6.12 performs all $(k!)^2$ permutations of the first k columns and the last k columns of G'. Since G' has the form $[I_k|A']$, the $k!$ permutations of the last k columns of G' can be accomplished by simply applying sequences of SWAP_c operations to A'. Furthermore, since a $\text{SWAP}_r(i_0, i_1)$ on A' produces the matrix A'', where $[I_k|A''] = \mathcal{RREF}(\pi(G))$, where π is the transposition of the columns of $[I_k|A']$ that swaps columns i_0 and i_1 of I_k, the $k!$ permutations of the first k columns of G' can be accomplished by performing the $k!$ permutations of the rows of A' using the SWAP_r operation. Thus, the second stage of Algorithm 6.12 can be accomplished by applying sequences of SWAP_r and SWAP_c operations to A'.

We are now ready to rewrite Algorithm 6.12 in terms of the $k \times k$ matrix A and the three operations $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c. One additional step we have included in the algorithm is to order, in descending order, the rows and columns of every matrix A' produced in the first stage of the algorithm using the *QuickOrder* algorithm of Chapter 3. In terms of permutations of the columns of G, the reason this step is included is that it reduces the number of permutations $\pi \in \Pi(T_I, T_A)$ that may be selected in the first stage of the algorithm. Thus, for any two sets $\Pi(T_{I_1}, T_{A_1})$ and $\Pi(T_{I_2}, T_{A_2})$, for which there exist permutations $\pi_1 \in \Pi(T_{I_1}, T_{A_1})$ and $\pi_2 \in \Pi(T_{I_2}, T_{A_2})$ such that $\mathcal{RREF}(\pi_1(G)) = \mathcal{RREF}(\pi_2(G))$, the chances of us selecting two such permutations is increased (since we are now only selecting permutations $\pi \in \Pi(T_I, T_A)$ in which the rows and columns of the matrix A' in $[I_k|A'] = \mathcal{RREF}(\pi(G))$ are ordered in descending order). Note that there

is no need for our algorithm to perform elementary row-operations since the $\mathrm{ADD}_{r,c}$, SWAP_r, and SWAP_c operations take care of this for us.

Algorithm 6.15 :

- Input: Any $k \times k$ matrix A such that $G = [I_k|A]$ generates a $(2k, k)$ self-dual code C.
- Output: The set L'' of all $k \times k$ matrices A'' such that $G'' = [I_k|A'']$ generates a code equivalent to C, and *acnt*, the size of the automorphism group of C.

begin

Stage 1:

Let $L' = \emptyset$.

Let S_r be the set of the k rows of A.

Let S_c be the set of the k columns of A.

Let $\Pi_1 = \{(T_r, T_c) \mid T_r \subseteq S_r, T_c \subseteq S_c, 0 \leq |T_r| = |T_c| \leq k\}$.

for each $(T_r, T_c) \in \Pi_1$ **do**

Let $m = |T_r|$.

if there exists a sequence S of m $\mathrm{ADD}_{r,c}$ operations on A that includes each row of T_r and each column of T_c **then**

Let $A' = S(A)$.

QuickOrderMatrix$(A', 0, A')$.

if $(A', cnt') \in L'$ **then** increment cnt' by one.

else insert $(A', 1)$ into L'.

end if

end for

Stage 2:

Let $acnt = 0$.

Let $L'' = \emptyset$.

for each $(A', cnt') \in L'$ **do**

Let $\Pi_2 = \{(\pi_r, \pi_c) \mid \pi_r$ is a permutation of the rows of A' and π_c is a permutation of the columns of $A'\}$.

for each $(\pi_r, \pi_c) \in \Pi_2$ **do**

Let $A'' = \pi_r \pi_c(A')$.

if $A'' \notin L''$ **then** insert A'' into L''.

if $A'' = A$ **then** increment *acnt* by cnt'.

end for

end for

Return L'' and *acnt*.

end

Given any $k \times k$ matrix A, where $G = [I_k|A] \in \mathcal{G}_{[I|A]}(C)$, Algorithm 6.15 can easily be modified to produce a unique representative $G_0 = [I_k|A_0]$ for any equivalence class of codes. For example, if we define $G_0 = [I_k|A_0]$ to be the unique matrix in $\mathcal{G}_{[I|A]}(C)$ such that A_0 is larger than every other $k \times k$ matrix A, where $G = [I_k|A] \in \mathcal{G}_{[I|A]}(C)$, then in the second stage of the algorithm, if instead of building the list L'' of every distinct matrix A'' we find, we just record the largest matrix A'' seen thus far, the final matrix recorded will be A_0.

6.3.4 Concluding Remarks

In this section, we developed an algorithm which, given any $k \times k$ matrix A, where $G = [I_k|A]$ generates a $(2k, k)$ self-dual code C, produces a list L'' of all $k \times k$ matrices A'', where $G'' = [I_k|A''] \in \mathcal{G}_{[I|A]}(C)$. The algorithm also finds *acnt*, the size of the automorphism group of C. The algorithm uses a two stage method for performing permutations of the columns of G.

The first stage of the algorithm performs $\binom{2k}{k}$ permutations π of the columns of $G = [I_k|A]$, one for each of the different choices for the first k columns of $\pi(G)$. If $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$ then it is recorded in the list L' produced by the algorithm, otherwise it is discarded. The output of the first stage of the algorithm is a list $L' = \{(A'_1, cnt'_1), (A'_2, cnt'_2), \ldots, (A'_m, cnt'_m)\}$, where cnt'_i is the number of times A'_i is produced in the first stage. The list L' has the two properties that for any $G'' = [I_k|A''] \in \mathcal{G}_{[I|A]}(C)$:

1. There exists at least one matrix A'_i in L' such that the rows and columns of A'_i are permutations of the rows and columns of A''.

2. If $pcnt$ is the number of permutations $\pi_r\pi_c$ of the rows and columns of A'' such that $\pi_r\pi_c(A'') = A''$, and if $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ are the matrices in L' whose rows and columns are permutations of the rows and columns of A'', then $\sum_{j=1}^{n} cnt'_{i_j} = acnt/pcnt$, where $acnt$ is the size of the automorphism group of C.

The first property allows us to find all matrices in the set $\mathcal{G}_{[I|A]}(C)$ by performing permutations of the rows and columns of every matrix A'_i in L'. The second property allows us to compute the size of the automorphism group of C.

For each pair (A'_i, cnt'_i) in the list L' produced by the first stage of the algorithm, the second stage of the algorithm performs each of the $(k!)^2$ permutations $\pi_r\pi_c$ of the rows and columns of A'_i. The output of the second stage is a list L'' of all $k \times k$ matrices A'', where $G'' = [I_k|A''] \in \mathcal{G}_{[I|A]}(C)$, and it also produces $acnt$, the size of the automorphism group of C. Given any $G'' = [I_k|A''] \in \mathcal{G}_{[I|A]}(C)$, the size of the automorphism group of C is found by computing $acnt = pcnt \times ccnt$, where $pcnt$ is the number of permutations $\pi_r\pi_c$ of the rows and columns of A'' such that $\pi_r\pi_c(A'') = A''$ and $ccnt$ is the number of sets of permutations $\Pi(T_I, T_A)$ in which there exists a permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = G''$.

In the next section, we will develop an algorithm for finding a unique representative for

any equivalence class of self-dual codes, and in later sections we will develop algorithms for finding a unique representative for specific equivalence classes of self-dual codes (i.e. the distance 4 and the distance ≥ 6 self-dual codes). Each of these algorithms will use the two stage method for performing permutations of the columns of G that Algorithm 6.15 uses. However, whereas Algorithm 6.15 produces every matrix in the set $\mathcal{G}_{[I|A]}(C)$, each of these algorithms will only produce matrices that are elements of some subset $\mathcal{G}$ of $\mathcal{G}_{[I|A]}(C)$. The unique representative $G_0 = [I_k|A_0]$ produced by each algorithm will be selected from the subset $\mathcal{G}$.

The first stage of each of the algorithms we will develop will also produce a list of pairs $L' = \{(A'_1, cnt'_1), (A'_2, cnt'_2), \ldots, (A'_m, cnt'_m)\}$ with the two properties that for any $G'' = [I_k|A''] \in \mathcal{G}$:

1. There exists at least one matrix A'_i in L' such that the rows and columns of A'_i are permutations of the rows and columns of A''.

2. If *pcnt* is the number of permutations $\pi_r\pi_c$ of the rows and columns of A'' such that $\pi_r\pi_c(A'') = A''$, and if $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ are the matrices in L' whose rows and columns are permutations of the rows and columns of A'', then $\sum_{j=1}^{n} cnt'_{i_j} = acnt/pcnt$, where *acnt* is the size of the automorphism group of C.

We will refer to the first stage of each of our algorithms as the *Combination algorithm.*

The second stage of each of the algorithms will use the list L' produced by the Combination algorithm to find the unique representative $G_0 = [I_k|A_0]$, and *acnt*, the size of the automorphism group of C. A_0 is found by performing a subset of the permutations of the rows and columns of each A'_i in L' that result in a matrix A'', where $G'' = [I_k|A''] \in \mathcal{G}$. The size of the automorphism group is found by computing $acnt = pcnt \times ccnt$. The second stage of each of our algorithms will be referred to as the *Permutation algorithm.*

Each of the algorithms we develop are designed to reduce the number of sets $\Pi(T_I, T_A)$ from which we select permutations, the number of matrices A' we insert into the list L',

and/or the number of permutations of the rows and columns of each A' in L' we perform.

Each of the algorithms we develop will make use of several properties of the $\binom{2k}{k}$ sets of permutations $\Pi(T_I, T_A)$, where $\Pi(T_I, T_A)$ is the set of all permutations π of the columns of G in which the first k columns of $\pi(G)$ are a permutation of the set of k columns $(S_I - T_I) \cup T_A$ of G. Therefore, we will conclude this section with a summary of some of the properties the set of permutations $\Pi(T_I, T_A)$ possesses.

Let $\Pi(T_I, T_A)$ be the set of $(k!)^2$ permutations of the columns of G such that the first k columns of $\pi(G)$ are a permutation of the set of k columns $(S_I - T_I) \cup T_A$ of G. Then:

1. If $\pi_1, \pi_2 \in \Pi(T_I, T_A)$ then $\mathcal{RREF}(\pi_1(G))$ has the form $[I_k|A_1]$ if and only if $\mathcal{RREF}(\pi_2(G))$ has the form $[I_k|A_2]$.

2. Given any $\pi \in \Pi(T_I, T_A)$, $\Pi(T_I, T_A)$ is equal to the set of $(k!)^2$ permutations $\pi_I \pi_A \pi$ of the columns of G, where π_I is a permutation of the first k columns of $\pi(G)$ and $\pi(A)$ is a permutation of the last k columns of $\pi(G)$.

Let $\Pi(T_I, T_A)$ be a set in which for any $\pi' \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi'(G))$ has the form $[I_k|A']$. Let $\pi_1, \pi_2 \in \Pi(T_I, T_A)$, let $G_1 = [I_k|A_1] = \mathcal{RREF}(\pi_1(G))$, and let $G_2 = [I_k|A_2] = \mathcal{RREF}(\pi_2(G))$. Then by property 2:

3. The rows and columns of A_1 are permutations of the rows and columns of A_2.

4. The number of permutations $\pi_1' \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi_1'(G)) = G_1$ is equal to the number of permutations $\pi_2' \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi_2'(G)) = G_2$.

5. The number of permutations $\pi_1' \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi_1'(G)) = G_1$ is equal to the number of permutations $\pi_r \pi_c$ of the rows and columns of A_1 such that $\pi_r \pi_c(A_1) = A_1$.

Finally, let $G' = [I_k|A'] \in \mathcal{G}_{[I|A]}(C)$. Let *acnt* be the size of the automorphism group of C. Let *pcnt* be the number of permutations $\pi_r \pi_c$ of the rows and columns of A' such that $\pi_r \pi_c(A') = A'$. Then:

6. The number of pairs (T_I, T_A) in which there exists a permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = G'$ is equal to $ccnt = acnt/pcnt$.

This last property allows us to compute the size of the automorphism group of C without actually performing every permutation of the columns of G that results in G'.

6.4 A General Unique Representative Algorithm

Let C be any $(2k, k)$ self-dual code. In this section, we will develop an algorithm which, given any $G = [I_k|A] \in \mathcal{G}_{[I|A]}(C)$, produces a unique $G_0 = [I_k|A_0] \in \mathcal{G}_{[I|A]}(C)$. The algorithm will also find $acnt$, the size of the automorphism group of C. The algorithm uses the same two stage method to perform permutations of the columns of G that Algorithm 6.15 of Section 6.3 uses. However, the algorithm will only perform permutations of the columns of G that result in matrices that are in the subset $\mathcal{G}$ of generator matrices in $\mathcal{G}_{[I|A]}(C)$, where $G'' = [I_k|A''] \in \mathcal{G}$ if and only if the row weights of A'' are *minimal*. The unique representative G_0 returned by the algorithm is the unique $G_0 = [I_k|A_0] \in \mathcal{G}$ where A_0 is the largest matrix whose rows and columns are in descending order. By limiting our permutations of the columns of G to only those that result in matrices that are in the subset $\mathcal{G}$, we will reduce both the size of the list L' produced by the Combination algorithm (which corresponds to the first stage of Algorithm 6.15) and the number of permutations of the rows and columns of each matrix A' in L' performed by the Permutation algorithm of our algorithm (which corresponds to the second stage of Algorithm 6.15).

6.4.1 The Combination Algorithm

The Combination algorithm for our unique representative algorithm is the same as the first stage of Algorithm 6.15 with the exception that instead of recording every $k \times k$ matrix A' produced in L', only the $k \times k$ matrices A' with *minimal row weights* will be recorded in L'.

Let us begin by defining what we mean by a matrix having *minimal row weights*. In order to do so, we will first define a comparison relation between the row weights of two matrices:

Definition 6.16 Let $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ be generator matrices for any two $(2k, k)$ self-dual codes. Let w be the smallest integer for which the number of rows with weight w in A_1 and the number of rows with weight w in A_2 are different (if such a w exists). Then:

- If A_1 has *more* rows with weight w than A_2 then the row weights of A_1 are said to be *smaller* than the row weights of A_2.
- If A_1 has *fewer* rows with weight w than A_2 then the row weights of A_1 are said to be *larger* than the row weights of A_2.
- If such a w does not exist, then the row weights of A_1 are said to be *equal* to the row weights of A_2.

This leads us to our definition of minimal row weights:

Definition 6.17 Let $G = [I_k|A]$ be the generator matrix for a $(2k, k)$ self-dual code C. Then A is said to have *minimal row weights* if for any $G' = [I_k|A'] \in \mathcal{G}_{[I|A]}(C)$, the row weights of A are smaller than or equal to the row weights of A'.

Let us now consider how we can alter the first stage of Algorithm 6.15 in order to produce a list L' of pairs (A', cnt'), where A' has minimal row weights, such that for any $G'' = [I_k|A''] \in \mathcal{G}$, there exists at least one A' in L' such that the rows and columns of A' are permutations of the rows and columns of A''.

Let S_I denote the set of the first k columns of G and let $\Pi(T_I, T_A)$ denote the set of all permutations π of the columns of G in which the first k columns of $\pi(G)$ are a permutation of the set $(S_I - T_I) \cup T_A$ of k columns of G. Suppose that for any $\pi \in \Pi(T_I, T_A)$,

$\mathcal{RREF}(\pi(G))$ has the form $[I_k|A]$. Let π_1 and π_2 be any two permutations in $\Pi(T_I, T_A)$ and let $[I_k|A_1] = \mathcal{RREF}(\pi_1(G))$ and $[I_k|A_2] = \mathcal{RREF}(\pi_2(G))$. Then, since the rows and columns of A_1 are permutations of the rows and columns of A_2, the row weights of A_1 and A_2 are equal. In other words, no matter what permutation $\pi \in \Pi(T_I, T_A)$ we select in the first stage of our algorithm, the rows weights of A' in $[I_k|A'] = \mathcal{RREF}(\pi(G))$ will be the same.

Therefore, the only alteration we need to make to the first stage of Algorithm 6.15 is to restrict the matrices A' in the list L' produced by the algorithm to just those matrices that have minimal row weights. We can produce such a list by simply comparing the row weights of each matrix A' produced during the algorithm to the row weights of any matrix that has already been inserted into the list. That is, for each matrix A' produced, if the row weights of A' are greater than the row weights of any matrix in L' then A' can be discarded, else if the row weights of A' are equal to the row weights of any matrix in L' then A' is recorded in L', else if the row weights of A' are less than the row weights of any matrix in L' then the list L' is emptied and then A' is recorded in L'. With this alteration, once the first stage has completed the list L' will only contain the matrices A' that have minimal row weights.

We are now ready to describe the Combination algorithm of our algorithm for finding a unique representative for any equivalence class of $(2k, k)$ self-dual codes.

Algorithm 6.18 *CombinationAlgorithm*(A, L'):

- Input: Any $k \times k$ matrix A such that $G = [I_k|A]$ generates a $(2k, k)$ self-dual code C.
- Output: A list L' of pairs (A', cnt'), where A' is a $k \times k$ matrix with minimal row weights, and cnt' is the number of times A' was produced. The list L' returned has the two properties that for any $G'' = [I_k|A''] \in \mathcal{G}_{[I|A]}(C)$, where A'' has minimal row weights:

1. There exists at least one A' in L' such that the rows and columns of A' are permutations of the rows and columns of A''.
2. If *pcnt* is the number of permutations $\pi_r\pi_c$ of the rows and columns of A'' such that $\pi_r\pi_c(A'') = A''$, and if $A'_1, A'_2, \ldots, A'_n$ are the subsets of matrices in L' whose rows and columns are permutations of the rows and columns of A'', and if cnt'_i is the number of times A'_i was produced, then $\sum_{i=1}^{n} cnt'_i = acnt/pcnt$. where *acnt* is the size of the automorphism group of C.

begin

Let $L' = \emptyset$.

Let S_r be the set of the k rows of A.

Let S_c be the set of the k columns of A.

Let $\Pi_1 = \{(T_r, T_c) \mid T_r \subseteq S_r, T_c \subseteq S_c, 0 \leq |T_r| = |T_c| \leq k\}$.

for each $(T_r, T_c) \in \Pi_1$ **do**

Let $m = |T_r|$.

if there exists a sequence S of m $\mathrm{ADD}_{r,c}$ operations on A that includes each row of T_r and each column of T_c **then**

Let $A' = S(A)$.

QuickOrderMatrix$(A', 0, A')$.

if $(L' = \emptyset)$ **or** (A' has smaller row weights than any $A'_1 \in L'$) **then**

$L' = \{(A', 1)\}$.

else if A' has row weights equal to any $A'_1 \in L'$ **then**

if $(A', cnt') \in L'$ **then** increment cnt' by 1.

else insert $(A', 1)$ into L'.

end else if

end if

end for

end

Example 6.19 Let C be the $(20, 10)$ self-dual code generated by $G = [I_{10}|A]$, where

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Before giving the output which Algorithm 6.18 gives us for this A, a few facts about the code C will be given that will help us better understand the output produced by the algorithm.

Let $\mathcal{G}$ be the subset of generator matrices $G'' = [I_{10}|A'']$ in $\mathcal{G}_{[I|A]}(C)$ in which the row weights of A'' are minimal. Then if $G'' = [I_{10}|A''] \in \mathcal{G}$, the rows and columns of A'' are permutations of the rows and columns of one (and only one) of the following two matrices:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \quad A' = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Notice that both A and A' have four rows with weight 4, two rows with weight 5, and four rows with weight 7. Furthermore, for any $G'' = [I_k|A''] \in \mathcal{G}_{[I|A]}(C)$, if the rows and columns of A'' are not permutations of the rows and columns of either one of A and A', and if $w = 3$ is the smallest integer in which the number of weight w rows in A and A'' differ, then A'' will have less than four rows with weight 3. If $w = 5$ then A'' will have less than two rows with weight 5, and if $w = 7$ then A'' will have less than four rows with weight 7. Also note that the number of permutations $\pi_r\pi_c$ of the rows and columns of A such that $\pi_r\pi_c(A) = A$ is 16, and the number of permutations $\pi_r\pi_c$ of the rows and columns of A' such that $\pi_r\pi_c(A') = A'$ is also 16. Finally, the size of the automorphism group of C is 82944.

If we run Algorithm 6.18 on A then the list of pairs produced by the algorithm is $L' = \{(A'_1, 192), (A'_2, 2112), (A'_3, 2880), (A'_4, 5184)\}$, where:

$$A'_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \quad A'_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$A'_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad A'_4 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Notice that the rows and columns of A'_1, A'_2, and A'_3 are all permutations of the rows and columns of A', and the rows and columns of A'_4 are permutations of the rows and columns of A. Therefore, the list L' has the property that for any $G'' = [I_{10}|A''] \in \mathcal{G}$, there exists at least one A'_i in L' such that the rows and columns of A'_i are permutations of the rows and columns of A''. Furthermore, if the rows and columns of A'' are permutations of the rows and columns of A', and thus permutations of the rows and columns of A'_1, A'_2, and A'_3, then the number of sets $\Pi(T_I, T_A)$ in which there exists a permutations $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = [I_k|A'']$ is:

$$\begin{aligned} ccnt &= \sum_{i=1}^{3} cnt'_i \\ &= 192 + 2112 + 2880 \\ &= 5184 \\ &= 82944/16 \\ &= acnt/pcnt \end{aligned}$$

If the rows and columns of A'' are permutations of the rows and columns of A, and thus permutations of the rows and columns of A'_4, then the number of sets $\Pi(T_I, T_A)$ in which

there exists a permutations $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = [I_k|A'']$ is:

$$\begin{aligned} ccnt &= \sum_{i=4}^{4} cnt'_i \\ &= 5184 \\ &= 82944/16 \\ &= acnt/pcnt \end{aligned}$$

Note that the fact that *pcnt*, and thus *ccnt*, is the same for both A and A' is just a coincidence and is not generally the case.

6.4.2 The Permutation Algorithm

The unique representative our algorithm will return is the unique $G_0 = [I_k|A_0] \in \mathcal{G}$ in which A_0 is the largest such matrix whose rows and columns are both in descending order. Therefore, for each A' in the list L' produced by the Combination algorithm, the Permutation algorithm for our unique representative algorithm needs only to perform the permutations $\pi_r\pi_c$ of the rows and columns of A' that result in the matrix $A_{max} = \pi_r\pi_c(A')$, where A_{max} is the unique matrix whose rows and columns are in descending order, with the property that: for any permutation $\pi'_r\pi'_c$ of the rows and columns of A', that results in a matrix $A'' = \pi'_r\pi'_c(A')$ whose rows and columns are in descending order, A_{max} is larger than or equal to A''. In other words, we want an algorithm that, given any A', permutes the rows and columns of A' in such a way that the unique matrix A_{max} is produced. The *UniqueOrder* algorithm of Chapter 3 is such an algorithm.

The *UniqueOrder* algorithm is based on the fact that for any permutation π_r of the columns of A' there exists one and only one permutation π_c of the columns of A' such that the columns of $\pi_r\pi_c(A')$ are in descending order. Therefore, the *UniqueOrder* algorithm only considers permutations π_r of the rows of A'. For each permutation π_r of the rows of A', the *UniqueOrder* algorithm finds the unique permutation π_c of the columns A' that results in a matrix $\pi_r\pi_c(A')$ whose columns are in descending order, by ordering the

columns of the matrix $\pi_r(A')$ in descending order. The permutations π_r of the rows of A' that the *UniqueOrder* algorithm performs are all the permutations that result in A_{max}. plus a subset of the permutations that result in matrices A'' where A'' is smaller than A_{max} and the rows and columns of A'' are in descending order.

For each A' in the list L' produced by the Combination algorithm, the Permutation algorithm for our unique representative algorithm will find the unique matrix A_{max}. Since the list L' may contain matrices A'_1 and A'_2 such that the rows and columns of A'_1 are not simply permutations of the rows and columns of A'_2, the Permutation algorithm finds A_0 by selecting the largest A_{max} found by the *UniqueOrder* algorithm for each A' in L'.

In order to find the automorphism group of C, the Permutation algorithm must also find *pcnt*, the number of permutations $\pi_r\pi_c$ of the rows and columns of A_0 such that $\pi_r\pi_c(A_0) = A_0$. One way to find *pcnt* is to use the fact that if A' is the matrix input to the *UniqueOrder* algorithm that produces the unique A_{max} that is assigned to A_0, then *pcnt* is equal to the number of permutations $\pi_r\pi_c$ of the rows and columns of A' such that $\pi_r\pi_c(A') = A_{max}$. Since the *UniqueOrder* algorithm performs all the permutations $\pi_r\pi_c$ of the rows and columns of A' such that $\pi_r\pi_c(A') = A_{max}$, we can use the *UniqueOrder* algorithm to find *pcnt*.

Let us now rewrite the *UniqueOrder* algorithm with the additional steps needed to count the number of permutations of the rows and columns of A' that results in A_{max}:

Algorithm 6.20 *UniqueOrderMatrix*$(A_{m-1}, m, A_{max}, pcnt, larger)$:

- Input: the $k \times k$ matrix A_{m-1}, the integer m, $i \leq m \leq k-1$, and the Boolean *larger*.
- Output: the $k \times k$ matrix A_{max} and the integer *pcnt*. A_{max} is the largest matrix whose rows and columns are permutations of the rows and columns of A_{m-1} such that the first $m-1$ rows of A_{max} and A_{m-1} are equal, and both the last $k-(m-1)$ rows and the columns of A_{max} are in descending order. The integer *pcnt* is the total

number of permutations $\pi_r\pi_c$ of the last $k-(m-1)$ rows and the columns of A_{m-1} such that $\pi_r\pi_c(A_{m-1}) = A_{max}$.

begin
  if $m \geq k$ **then**
    if (A_{max} is undefined) **or** ($A_{m-1} > A_{max}$) **then**
      $A_{max} = A_{m-1}$.
      $pcnt = 1$.
    end if
    else if $A_{m-1} = A_{max}$ **then**
      $pcnt = pcnt + 1$.
  end if
  else begin
    FindLargestRemainingRows(A_{m-1}, m, S_m).
    for each $r \in S_m$ **do begin**
      $A'_{m-1} = \text{SWAP}_r(m, r)(A_{m-1})$.
      OrderMatrixRow(A'_{m-1}, m, A_m).
      if *larger* **or** (row m of A_m) > (row m of A_{max}) **then**
        UniqueOrderMatrix$(A_m, m+1, A_{max}, pcnt, true)$.
      else if (row m of A_m) = (row m of A_{max}) **then**
        UniqueOrderMatrix$(A_m, m+1, A_{max}, pcnt, false)$.
    end for
  end else
end

Note that the *UniqueOrder* algorithm is recursive. For our purposes in this section, it is invoked with the call *UniqueOrderMatrix*$(A', i, A_{max}, pcnt, false)$, where $i = 1$ and A_{max} is initially *undefined.* In later sections, the *UniqueOrder* algorithm will be used with a starting row $i > 1$. In this case, *pcnt* is equal to the number of permutations $\pi_r\pi_c$

of the last $k-(i-1)$ rows and the columns of A' such that $\pi_r\pi_c(A') = A_{max}$.

Let us now give the output of the *UniqueOrder* for the matrices in the list L' produced by the *CombinationAlgorithm* in Example 6.19

Example 6.21 Consider the four matrices A'_1, A'_2, A'_3, and A'_4 in the list L' of Example 6.19.

The rows and columns of A'_1, A'_2, and A'_3 are permutations of each other, and therefore the matrix returned by the *UniqueOrder* algorithm for each of them will be the same. As it turns out, the matrix returned by the *UniqueOrder* algorithm for each of A'_1, A'_2 and A'_3 is:

$$A'_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

For each of these matrices, the algorithm also returns $pcnt = 16$, the number of permutations $\pi_r\pi_c$ of the columns of A'_3 in which $\pi_r\pi_c(A'_3) = A'_3$.

The matrix returned by the *UniqueOrder* algorithm for A'_4 is:

$$A'_4 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

The *UniqueOrder* algorithm also returns $pcnt = 16$, the number of permutations $\pi_r\pi_c$ of the columns of A'_4 in which $\pi_r\pi_c(A'_4) = A'_4$.

Note that while, in this example, the matrices A'_3 and A'_4 returned by the *UniqueOrder* algorithm were already elements of the list L', since the matrices in L' where ordered by the *QuickOrder* algorithm (which does not generally produce the unique matrix returned by the *UniqueOrder* algorithm), this is not generally the case.

We are now ready to give a formal description of the Permutation algorithm for our unique representative algorithm.

Algorithm 6.22 *PermutationAlgorithm*(L', A_0, *acnt*):

- Input: the list L' of pairs (A', cnt') produced by the Combination algorithm.
- Output: the $k \times k$ matrix A_0 and *acnt*. A_0 is the largest matrix with minimal row weights whose rows and columns are in descending order such that $G_0 = [I_k|A_0] \in \mathcal{G}_{[I|A]}(C)$. The integer *acnt* is the size of the automorphism group of C.

begin

Initially A_0 is *undefined*.

for each $(A', cnt') \in L'$ **do**

UniqueOrderMatrix$(A', A'', 0, pcnt, false)$.

if (A_0 is *undefined*) or ($A'' > A_0$) **then**

$A_0 = A''$.

$acnt = pcnt * cnt'$.

end if

else if ($A'' = A_0$) **then**

$acnt = acnt + pcnt * cnt'$.

end for

end

Example 6.23 Consider the list $L' = \{(A'_1, 192), (A'_2, 2112), (A'_3, 2880), (A'_4, 5184)\}$ produced by the Combination algorithm in Example 6.19.

The first pair processed by the algorithm is $(A'_1, 192)$. The call to the procedure *UniqueOrderMatrix*$(A'_1, A_{max}, 0, pcnt, false)$ returns $A_{max} = A'_3$ and $pcnt = 16$. Since A_0 is undefined, the algorithm will set $A_0 = A'_3$ and $acnt = 16 * 192 = 3072$.

The second pair processed by the algorithm is $(A'_2, 2112)$. The call to the procedure *UniqueOrderMatrix*$(A'_2, A_{max}, 0, pcnt, false)$ returns $A_{max} = A'_3$ and $pcnt = 16$. Since $A'_3 = A_0$, the algorithm will compute $acnt = 3072 + 16 * 2112 = 36864$.

The third pair processed by the algorithm is $(A'_3, 2880)$. The call to the procedure *UniqueOrderMatrix*$(A'_3, A_{max}, 0, pcnt, false)$ returns $A_{max} = A'_3$ and $pcnt = 16$. Since $A'_3 = A_0$, the algorithm will compute $acnt = 36864 + 16 * 2880 = 82944$.

The fourth and final pair processed by the algorithm is $(A'_4, 5184)$. The call to *UniqueOrderMatrix*$(A'_4, A_{max}, 0, pcnt, false)$ returns $A_{max} = A'_4$ and $pcnt = 16$. Since $A'_4 > A'_3$, the algorithm resets $A_0 = A'_4$ and recomputes $acnt = 16 * 5184 = 82944$.

Thus, the unique A_0 returned by the algorithm is:

$$A_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

and the size of automorphism group of C is computed to be $acnt = 82944$.

6.4.3 The Unique Representative Algorithm

Together, the Combination and Permutation algorithms described in this section give us an algorithm which, given any $k \times k$ matrix A, where $G = [I_k|A] \in \mathcal{G}_{[I|A]}(C)$, produces a unique representative A_0, where A_0 is the largest matrix with minimal row weights whose rows and columns are in descending order, such that $G_0 = [I_k|A_0] \in \mathcal{G}_{[I|A]}(C)$:

Algorithm 6.24 *UniqueRepresentative*$(A, A_0, acnt)$:

- Input: any $k \times k$ matrix A such that $G = [I_k|A]$ generates a $(2k, k)$ self-dual code C.
- Output: the $k \times k$ matrix A_0 and *acnt*. $G_0 = [I_k|A_0]$ is the unique matrix in $\mathcal{G}_{[I|A]}(C)$ where A_0 is the largest matrix with minimal row weights, and whose rows and columns are in descending order. *acnt* is the size of the automorphism group of C.

begin

CombinationAlgorithm(A, L').

PermutationAlgorithm$(L', A_0, acnt)$.

return A_0 and $acnt$.

end

We now have our Unique Representative algorithm for any equivalence class of $(2k, k)$ self-dual codes. One major drawback of the algorithm is that the Combination algorithm still needs to apply $\binom{2k}{k}$ permutations to the input matrix. To get around this, we will start to consider the different classes of codes. In the next section, we will have a look at the decomposable codes, and the indecomposable distance 2 and e_{2i}-codes. As we shall see, for these classes of codes there is no need for our Unique Representative algorithm. In Sections 6.6, 6.7, and 6.8, we will develop Combination and Permutation algorithms for the remaining classes of distance 4 codes, namely the e_3-codes and the d_i-codes. Finally, in Section 6.9, we will develop Combination and Permutation algorithms for the distance ≥ 6 codes.

6.5 The Decomposable, Distance 2, and e_{2i}-Codes

There are two reasons we want a unique representative algorithm. The first reason is to find and remove the equivalent codes from our lists of $(2k, k)$ self-dual codes we produce using the methods described in Chapters 4 and 5. The second reason is to find the size of the automorphism group of C.

For some classes of codes, namely the decomposable codes, and the indecomposable distance 2 and $(2k, k, 4)$ e_{2i}-codes, we can easily find a list L of such codes, such that for any C that is either decomposable, has distance 2, or is an e_{2i}-code, there exists one and only one code in L that is equivalent to C (for the indecomposable and distance 2 codes see Chapter 1 and for the $(2k, k, 4)$ e_{2i}-codes, see Section 4.4 of Chapter 4). Therefore, we

do not need a unique representative algorithm to find and remove equivalent codes from the list L. In this section, we will show how to compute the size of the automorphism group for each of the decomposable, distance 2, and e_{2i}-codes, thereby eliminating the need for a unique representative algorithm for these classes of codes. Note that the results in this section are not new. They are included for completeness sake.

6.5.1 The Decomposable and Distance 2 Self-Dual Codes

Let C be a decomposable $(2k, k)$ self-dual code. In order to compute the size of the automorphism group of C, we will first define some notation. C_i will denote an indecomposable self-dual code. For $m \geq 1$, we will use $C_{i,m}$ to denote the composition of C_i, m times. That is:

$$C_{i,m} = \underbrace{C_i \oplus C_i \oplus \cdots \oplus C_i}_{m}$$

Using this notation, C is the composition of one or more codes:

$$C = C_{1,m_1} \oplus C_{2,m_2} \oplus \cdots \oplus C_{j,m_j}$$

where each pair of indecomposable codes C_{i_1} and C_{i_2}, $i_1 \neq i_2$, are inequivalent.

Let $Aut(C)$ denote the size of the automorphism group of C. For each inequivalent code C_i in C, let G_i be a generator matrix for C_i and let $G_{i,m}$ denote the composition of G_i, m times. Then $G = G_{1,m_1} \oplus G_{2,m_2} \oplus \cdots G_{j,m_j}$ is a generator matrix for C. Let π be any permutation that is an automorphism of G. Then π can be written as the product of one or more permutations π' where π' is either an automorphism of one of the m_i occurrences of G_i in G_{i,m_i}, or π' completely swaps the columns of two of the occurrences of G_i in G_{i,m_i}. Therefore:

$$Aut(C) = \prod_{i=1}^{j} (m_i)!\ Aut(C_i)^{m_i} \tag{6.1}$$

Thus, given the size of the automorphism group of each of the inequivalent indecomposable codes in C, we can compute the size of the automorphism group of C.

Let C be any $(2k, k, 2)$ self-dual code. Then, with the exception of the $(2, 1, 2)$ self-dual code generated by $G = [1\ 1]$, C is decomposable, and thus we can find the size of the automorphism group of C with Equation 6.1. The size of the automorphism group of the $(2, 1, 2)$ self-dual code generated by $G = [1\ 1]$ is 2.

6.5.2 The e_{2i}-Codes

Let C be any $(2k, k, 4)$ e_{2i}-code. Then, with the exception of the $(2k, k, 4)$ e_k-code, where $k \equiv 0 \bmod 2$ and $k \geq 4$, C is decomposable, and thus we can find the size of the automorphism group of C with Equation 6.1. Therefore, we only need to find the size of the automorphism group of the $(2k, k, 4)$ e_k-codes.

Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ e_k-code. We will find the size of the automorphism group for these codes by computing $acnt = ccnt \times pcnt$, where $pcnt$ is the number of permutations $\pi_r\pi_c$ of the rows and columns of A such that $\pi_r\pi_c(A) = A$, and $ccnt$ is the number of sets of permutations $\Pi(T_I, T_A)$ for which there exists a permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = G$.

Let us first find $acnt$ for the e_4-code. The $(8, 4, 4)$ e_4-code C is generated by the matrix:

$$G = [I_4|A] = \begin{array}{c} \begin{array}{cccc|cccc} a_1 & a_2 & a_3 & a_4 & c_1 & c_2 & c_3 & c_4 \end{array} \\ \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{l} r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}$$

Let us first find $ccnt$ for this G. Let $\vec{w}$ be any weight 4 word in the e_4-code C. We will use the phrase *the columns of $\vec{w}$ in G* to refer to four columns c of G in which the rows of G that sum to $\vec{w}$, sum to a one in column c. For example, the four columns of the weight 4 word $\vec{w} = r_1 \oplus r_2$ are a_1, a_2, c_3, and c_4. It can be proved that for any permutation π of the columns of G, $\mathcal{RREF}(\pi(G)) = [I_k|A']$, where the rows and columns

of A' are permutations of the rows and columns of A, if and only if the last four columns of $\pi(G)$ contain three of the four columns of some weight 4 row $\vec{w}$ in C. (Note that the last four columns of $\pi(G)$ will in fact contain three of the four columns of exactly four of the weight 4 rows in C). Therefore, to find the number of sets $\Pi(T_I, T_A)$ in which there exists a permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = G$, we need only count the number of different combinations for the last four columns of $\pi(G)$ in which three of the last four columns of $\pi(G)$ are the columns of some weight 4 $\vec{w}$ in C. There are exactly 14 weight 4 words in C, and thus $\binom{14}{1}$ choices for $\vec{w}$. For each of these choices, there are $\binom{4}{3}$ choices for the three columns of $\vec{w}$ that are in the last four columns of $\pi(G)$ and $\binom{4}{1}$ choices for the remaining column in the last four columns of $\pi(G)$. Since for each of these choices there are in fact four weight 4 words in C with three of their four columns in $\pi(G)$, this gives us a total of $\binom{14}{1}\binom{4}{3}\binom{4}{1}/4 = 14 * 4 = 56$. Therefore, the number of sets $\Pi(T_I, T_A)$ in which there exists a permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = G$ is 56. Thus, for the $(8, 4, 4)$ e_4-code, $ccnt = 56$.

For the e_4-code generated by $G = [I_4|A]$, $pcnt$ is equal to the number of permutations of the rows of A. That is, for any permutation π_r of the rows of A there exists one and only one permutation π_c of the columns of A such that $\pi_r\pi_c(A) = A$. Therefore, for the $(8, 4, 4)$ e_4-code, $pcnt = 4! = 24$.

Thus, the size of the automorphism group of the $(8, 4, 4)$ e_4-code is:

$$\begin{aligned} acnt &= ccnt \times pcnt \\ &= 56 \times 24 \\ &= 1344 \end{aligned}$$

Let us now find $acnt$ for the e_{2k}-codes, where $k \geq 6$ and $k \equiv 0 \bmod 2$. The $(2k, k, 4)$

e_k-code, $k \geq 6$, is generated by the matrix:

$$G = [I_k|A] = \begin{array}{c} \begin{matrix} a_1 & a_2 & & a_{k-2} & a_{k-1} & a_k & c_1 & c_2 & c_3 & c_4 & & c_k \end{matrix} \\ \left[\begin{array}{cccccc|cccccc} 1 & 0 & \cdots & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & 1 & 1 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 & 1 & 0 & 1 & 0 & 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 & 1 & 1 & 1 & \cdots & 1 \end{array}\right] \end{array} \begin{array}{c} r_1 \\ r_2 \\ \vdots \\ r_{k-2} \\ r_{k-1} \\ r_k \end{array}$$

Let us first find *ccnt* for this G. Consider W, the set of weight 4 words contained in the codewords of the e_k-code. Note that if $\vec{w} \in W$ then $\vec{w}$ is either one of the first $k-2$ rows of G, the sum of any pair of the first $k-2$ rows of G, the sum of the last 2 rows of G, or the sum of one of the first $k-2$ rows of G and the last 2 rows of G. Now consider the following set S of k disjoint pairs of columns of G:

$$S = \{(a_1, c_3), (a_2, c_4), \ldots, (a_{k-2}, c_k), (a_{k-1}, a_k), (c_1, c_2)\}$$

Notice that for each of these pairs of columns, if $\vec{w} \in W$ then $\vec{w}$ has the same value in both of the columns in each pair. Furthermore, these are the only pairs of columns of G for which this is true.

Example 6.25 For the pair (c_1, c_2), any $\vec{w} \in W$ that is one of the first $k-2$ rows of G has a value of 1 in columns c_1 and c_2, any $\vec{w} \in W$ that is the sum of any two of the first $k-2$ rows of G has a value of 0 in columns c_1 and c_2, the $\vec{w} \in W$ that is the sum of the last 2 rows of G has a value of 1 in columns c_1 and c_2, and any $\vec{w} \in W$ that is the sum of one of the first $k-2$ rows of G and the last 2 rows of G has a value of 0 in columns c_1 and c_2. This is not true for the pair (c_1, c_3) since the weight 4 word $\vec{w} = r_2$ has a value of 1 in column c_1 and a value of 0 in column c_3.

Thus, the columns of G can be uniquely partitioned into the set S of k pairs of columns such that for any weight 4 word $\vec{w} \in W$, the value of $\vec{w}$ is the same in both columns of

each pair. Notice that one of the pairs of columns in S (namely (a_{k-1}, a_k)) consists of two columns from I_k, another one of these pairs (namely (c_1, c_2)) consists of two columns from A, and the remaining $k-2$ pairs consists of one column from I_k and one column from A. Now, if π is a permutation of the columns of G such that $\mathcal{RREF}(\pi(G)) = G$, this must also be true of the columns of $\pi(G)$. That is, the first k columns of $\pi(G)$ must contain both columns of one and only one of the pairs of columns in S, and, the last k columns of $\pi(G)$ must contain both columns of one and only one of the pairs of columns in S. Furthermore, for any set of permutations $\Pi(T_I, T_A)$ such that for any $\pi \in \Pi(T_I, T_A)$ the first k columns of $\pi(G)$ contain both columns of one and only one of the pairs of columns in S (which implies the last k columns of $\pi(G)$ will contain both columns of one and only one of the pairs of columns in S), it can be shown that there exists a permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = G$.

Example 6.26 Let $S_I = \{1, 2, \ldots, k\}$ denote the set of columns of I_k in G and let $S_A = \{k+1, k+2, \ldots, 2k\}$ denote the set of columns of A in G. Consider the set of permutations $\Pi(\{k\}, \{k+3\})$. The permutation $\pi = (k\ k+3)$ (i.e. the permutation that swaps the last column of I_k with the third column of A) is an element of $\Pi(\{k\}, \{k+3\})$ and:

$$\mathcal{RREF}(\pi(G)) = G = \begin{array}{c} \begin{array}{cccccc|cccccc} a_{k-1} & a_2 & & a_{k-2} & a_1 & c_3 & c_1 & c_2 & a_k & c_4 & & c_k \end{array} \\ \left[\begin{array}{cccccc|cccccc} 1 & 0 & \cdots & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & 1 & 1 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 & 1 & 0 & 1 & 0 & 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 & 1 & 1 & 1 & \cdots & 1 \end{array}\right] \end{array} \begin{array}{l} r_{k-1} \oplus r_k \\ r_2 \\ \vdots \\ r_{k-2} \\ r_1 \oplus r_k \\ r_k \end{array}$$

The only pair in S that has both columns in the first k columns of $\pi(G)$ is (a_1, c_3) and the only pair in S that has both columns in the last k columns of $\pi(G)$ is (c_1, c_2).

Thus, to find *ccnt*, we need only count the number of different combinations for the

first k columns of $\pi(G)$ in which the first k columns of $\pi(G)$ contains both columns of one and only one pair of columns in S. Now, there are $\binom{k}{1}$ choices for the pair of columns in S whose columns are both in I_k, and for each of these choices, there are $\binom{k-1}{1}$ choices for the pair of columns in S whose columns are both in A, and for each of the $k-2$ remaining pairs of columns in S, there are $\binom{2}{1}$ choices for the one column in the pair that is a column of I_k. Therefore, the number of sets $\Pi(T_I, T_A)$ in which there exists a permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = G$ is equal to $\binom{k}{1}\binom{k-1}{1}\binom{2}{1}^{k-2} = k(k-1)2^{k-2}$. Thus, for the $(2k, k, 4)$ e_k-code, $k \geq 6$, $ccnt = k(k-1)2^{k-2}$.

Let us now find $pcnt$. Now, for any permutation π_r of the first $k-2$ rows of A there exists one and only one permutation π_c of the columns of $\pi_r(A)$ such that $\pi_r\pi_c(A) = A$. Also, for any permutation π_r of the last 2 rows of A there exists one and only one permutation π_c of the columns of $\pi_r(A)$ such that $\pi_r\pi_c(A) = A$. Furthermore, if π_r is any permutation of the rows of A that permutes one or both of the last 2 rows of A into the first $k-2$ rows of $\pi_r(A)$, then there does not exist a permutation π_c of the columns of $\pi_r(A)$ such that $\pi_r\pi_c(A) = A$. Therefore, for the $(2k, k, 4)$ e_k-code, $k \geq 6$, $pcnt = 2!(k-2)! = 2(k-2)!$.

Thus, the size of the automorphism group of the $(2k, k, 4)$ e_k-code, $k \geq 6$, is:

$$\begin{aligned} acnt &= ccnt \times pcnt \\ &= (k(k-1)2^{k-2})(2(k-2)!) \\ &= k(k-1)(k-2)!2^{k-1} \end{aligned}$$

Thus, we now have formulas to compute the size of the automorphism group of any decomposable code and any indecomposable distance 2 and e_{2i}-code. Therefore, we can now restrict our unique representative algorithm to just the indecomposable distance 4 codes (excluding the e_k-codes) and distance ≥ 6 self-dual codes.

6.6 Generator Blocks with Weight 4 Rows

In this section, and the next two sections, we will look at ways in which we can improve the Combination and Permutation algorithms of Section 6.4 in order to find a unique representative for any code in two subclasses of $(2k, k, 4)$ self-dual codes: the e_3-codes and the d_i-codes. In this section, we will prove several results on permutations involving the columns of a subset of the generator blocks for weight 4 sets that may occur in the generator matrices for the e_3 and d_i-codes. In Section 6.7, we will use the results proved in this section to develop our Combination algorithm for the e_3 and d_i-codes, and in Section 6.8, we will develop our Permutation algorithm for the e_3 and d_i-codes.

6.6.1 Generator Blocks for Weight 4 Sets

We will begin this section by reviewing some of the basic facts we have learned about generator blocks for weight 4 sets.

Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ self-dual code C. Recall that the rows of G can be partitioned into the $m+1$ disjoint sets $G_{B_1}, G_{B_2}, \ldots, G_{B_m}, G_R$ where each G_{B_j} is a generator block for a weight 4 set B_j and G_R is the set of the remaining rows of G (if any). Furthermore, the columns of G can be partitioned into the $m+1$ disjoint sets $S_{B_1}, S_{B_2}, \ldots, S_{B_m}, S_R$ where each S_{B_j} are the columns of the generator block G_{B_j} and S_R is the set of the remaining columns of G (if any). Recall that the columns of a generator block G_B for a weight 4 set B is the set S_B of columns c of G for which there exists at least one weight 4 word $\vec{w} \in B$ such that the subset of rows of G_B that sum to $\vec{w}$, sum to one in column c.

Example 6.27 Let C be the $(18, 9, 4)$ self-dual code generated by:

$$G = [I_9|A] = \begin{array}{c} \begin{array}{ccccccccc|ccccccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 \end{array} \\ \left[\begin{array}{ccccccccc|ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{array}\right] \begin{array}{c} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \end{array} \end{array}$$

The rows of G partition into the 3 disjoint sets:

$$\begin{aligned} G_{B_1} &= \{r_1, r_2, r_3\} \\ G_{B_2} &= \{r_4, r_5, r_6, r_7, r_8\} \\ G_R &= \{r_9\} \end{aligned}$$

where G_{B_1} is an $e_{3,0}$-block and G_{B_2} is a $d_{4,1}$-block. The columns of G partition into the 3 disjoint sets:

$$\begin{aligned} S_{B_1} &= \{a_1, a_2, a_3, c_1, c_2, c_3, c_4\} \\ S_{B_2} &= \{a_4, a_5, a_6, a_7, a_8, c_5, c_6, c_7, c_8, c_9\} \\ S_R &= \{a_9\} \end{aligned}$$

where S_{B_1} are the columns of the $e_{3,0}$-block and S_{B_2} are the columns of the $d_{4,1}$-block.

In Section 4.2 we found *all the different* generator blocks that may occur in a generator matrix $G = [I_k|A]$, where G generates a $(2k, k, 4)$ self-dual code. They were the $e_{3,0}$ and $e_{3,1}$-blocks, the $e_{4,0}$-block, the $e_{2i,0}$ and $e_{2i,1}$-blocks, $i \geq 3$, the $d_{1,0}$, $d_{1,3}$, and $d_{1,4}$-blocks,

and the $d_{i,0}$, $d_{i,1}$, $d_{i,2}$, $d_{i,3}$, and $d_{i,4}$-blocks, $i \geq 2$. What is meant by *all the different* generator blocks is that if G_B is any generator block in the generator matrix $G = [I_k|A]$, then the rows and columns of the G_B-block are permutations of the rows and columns of one (and only one) of these generator blocks. Note that we will use the notation given (i.e. $e_{3,0}, e_{3,1}, e_{4,0}, \ldots$) to refer to any permutation of the rows and columns of these blocks. For example, we will refer to both of the following sets of rows as $e_{3,0}$-blocks:

$$\begin{array}{cccc|ccccc|cccc|ccccc}
1 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & 0^* & 1 & 0 & 0 & 0^* & 0 & 1 & 1 & 1 & 0^* \\
0 & 1 & 0 & 0^* & 1 & 1 & 0 & 1 & 0^* & 0 & 1 & 0 & 0^* & 1 & 0 & 1 & 1 & 0^* \\
0 & 0 & 1 & 0^* & 1 & 0 & 1 & 1 & 0^* & 0 & 0 & 1 & 0^* & 1 & 1 & 1 & 0 & 0^*
\end{array}$$

The different generator blocks can be grouped into different classes of generator blocks: the e_3-blocks, the e_4-blocks, the e_{2i}-blocks, the d_1-blocks, and the d_i-blocks. The $e_{3,0}$ and $e_{3,1}$-blocks are both instances of an e_3-block. The $e_{4,0}$-blocks is an instance of an e_4-block. The $e_{2i,0}$ and $e_{2i,1}$-blocks are both instances of an e_{2i}-block. The $d_{i,0}$, $d_{i,1}$, $d_{i,2}$, $d_{i,3}$, and $d_{i,4}$-blocks are all instances of a d_i-block. And the $d_{1,0}$, $d_{1,3}$, and $d_{1,4}$-blocks are all instances of a d_1-block.

The reason we group the different generator blocks into different classes of generator blocks is as follows: Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ self-dual code C and let G_B be a generator block in G for a weight 4 set B. Let π be any permutation of the columns of G such that $G' = \mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$. Then the permutation π of the columns of G may also permute columns of the G_B-block, and when the elementary row operations are applied to $\pi(G)$, producing G', the rows of the G_B-block may also be altered (for example, a row of the G_B-block in $\pi(G)$ may have added to it a row in $\pi(G)$ that is not a row of the G_B-block). Therefore, the generator block G_B in G may be transformed into a generator block $G_{B'}$ in G' in which the rows and columns of $G_{B'}$ are not simply permutations of the rows and columns of the G_B-block. The reason for our grouping of the different generator blocks is that the generator block $G_{B'}$ must be a member of the same group as G_B (as we learned in Chapter 4). For example, if G_B

is either an $e_{3,0}$-block or an $e_{3,1}$-block, then $G_{B'}$ is either an $e_{3,0}$-block or an $e_{3,1}$-block. Thus, the $e_{3,0}$-block and the $e_{3,1}$-block are both considered instances of an e_3-block.

In order to classify the $(2k, k, 4)$ self-dual codes, the different classes of generator blocks for weight 4 sets are given a precedence. In descending order, this precedence is:

$$\ldots, e_8, e_6, e_4, e_3, \ldots, d_i, d_{i-1}, \ldots, d_4, d_3, d_2, d_1.$$

Each $(2k, k, 4)$ self-dual code is classified by the largest precedence block it contains. That is, for each $(2k, k, 4)$ self-dual code C, generated by $G = [I_k|A]$, if the highest precedence block in G is an e_{2i}-block then C is classified as an e_{2i}-code, if the highest precedence block in G is an e_3-block then C is classified as an e_3-code, and if the highest precedence block in G is a d_i-block then C is classified as a d_i-code.

As was demonstrated in Section 6.5, there is no need for a unique representative algorithm for the e_{2i}-codes, $i \geq 2$. Therefore, in this section, and the next two sections, we will only consider the e_3-codes and the d_i-codes, $i \geq 1$. That is, we will only consider generator matrices $G = [I_k|A]$ for $(2k, k, 4)$ self-dual codes in which the generator blocks for weight 4 sets in G are either e_3-blocks and/or d_i-blocks, $i \geq 1$. Moreover, in this section, we will only consider a subset of the different generator blocks in these two classes of generator blocks. That is, the only generator blocks we will consider in this section are the $e_{3,0}$-blocks, the $e_{3,1}$-blocks, the $d_{i,0}$-blocks, $i \geq 1$, and the $d_{i,1}$-blocks, $i \geq 2$ (see Figure 6.1). These are the only instances of the e_3-blocks and d_i-blocks that contain weight 4 rows (i.e. all the rows in each of the $d_{i,2}$, $d_{i,3}$, and $d_{i,4}$-blocks, all have weights ≥ 6). Therefore, in order to distinguish them from the generator blocks we are not considering in this section, we will refer to the $e_{3,0}$, $e_{3,1}$, $d_{i,0}$, and $d_{i,1}$-blocks as *generator blocks with weight 4 rows.*

$$\left[\begin{array}{cccc|ccccc} 1 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & 0^* \\ 0 & 1 & 0 & 0^* & 1 & 1 & 0 & 1 & 0^* \\ 0 & 0 & 1 & 0^* & 1 & 0 & 1 & 1 & 0^* \end{array}\right]$$

an $e_{3,0}$-block

$$\left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0^* & 1 & 1 & 1 & 0^* \\ 0 & 1 & 0 & 0 & 0^* & 1 & 1 & 0 & x \\ 0 & 0 & 1 & 0 & 0^* & 1 & 0 & 1 & x \\ 0 & 0 & 0 & 1 & 0^* & 0 & 1 & 1 & x \end{array}\right]$$

an $e_{3,1}$-block

$$\left[\begin{array}{\overbrace{ccccc}^{i}|ccccc} 1 & 0 & \cdots & 0 & 0^* & 1 & 1 & \overbrace{1}^{i} & 0 & \cdots & 0 & 0^* \\ 0 & 1 & \cdots & 0 & 0^* & 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0^* & 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \end{array}\right]_{i}$$

a $d_{i,0}$-block, $i \geq 1$

$$\left[\begin{array}{\overbrace{ccccccc}^{i-1}|ccccc} 1 & 0 & \cdots & 0 & 0 & 0 & 0^* & 1 & 1 & \overbrace{1}^{i-1} & 0 & \cdots & 0 & 0^* \\ 0 & 1 & \cdots & 0 & 0 & 0 & 0^* & 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & 0^* & 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \\ 0 & 0 & \cdots & 0 & 1 & 0 & 0^* & 1 & 0 & 1 & 1 & \cdots & 1 & x \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0^* & 0 & 1 & 1 & 1 & \cdots & 1 & x \end{array}\right]_{i-1}$$

a $d_{i,1}$-block, $i \geq 2$

Figure 6.1: Generator Blocks with Weight 4 Rows.

6.6.2 Generator Blocks with Weight 4 Rows in the Combination Algorithm

In the remainder of this section, we will prove several results on permutations involving the columns of the generator blocks with weight 4 rows. We will begin by introducing or reviewing some of the notations and concepts we will use throughout this section.

Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, 4)$ self-dual code such that G contains at least one generator block with weight 4 rows. Let G_B be a specific generator block with weight 4 rows in G. Let π be a permutation of the columns of G such that $G' = \mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$. We will use the notation $\pi(G_B)$ to denote the generator block in G' that corresponds to the generator block G_B of G.

Example 6.28 Let C be the $(14, 7, 4)$ self-dual code generated by:

$$G = [I_7|A] = \begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{matrix} \\ \left[\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right] \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \end{matrix} \end{array}$$

Let G_B be the $e_{3,0}$-block in the first three rows of G. Let $\pi = (5\ 9)$ be a permutation of

the columns of G. Then:

$$G' = \mathcal{RREF}(\pi(G)) = \left[\begin{array}{ccccccc|ccccccc|l} a_1 & a_2 & a_3 & a_4 & c_2 & a_6 & a_7 & c_1 & a_5 & c_3 & c_4 & c_5 & c_6 & c_7 & \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & r_1 \oplus r_5 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & r_2 \oplus r_5 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & r_4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & r_5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & r_6 \oplus r_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & r_7 \oplus r_5 \end{array}\right]$$

$\pi(G_B)$ is the $e_{3,1}$-block located in rows 1, 2, 3, and 5 of G'.

For a given G and generator block G_B with weight 4 rows, we will say a permutation π of the columns of G *preserves* the generator block G_B if $G' = \mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$ and if the generator block $\pi(G_B)$ in G' is still a G_B-block. That is, π preserves the generator block G_B if the rows and columns of the generator block $\pi(G_B)$ are simply permutations of the rows and columns of G_B.

Example 6.29 Consider the $(14, 7, 4)$ self-dual code C generated by the matrix $G = [I_7|A]$ given in Example 6.28. The permutation $\pi = (5\ 9)$ of Example 6.28 does *not* preserve the $e_{3,0}$-block in the first three rows of G since $\pi(G_B)$ is an $e_{3,1}$-block. Now consider the permutation $\pi = (2\ 9)$ of the columns of G:

$$G' = \mathcal{RREF}(\pi(G)) = \left[\begin{array}{ccccccc|ccccccc|l} a_1 & c_2 & a_3 & a_4 & a_5 & a_6 & a_7 & c_1 & a_2 & c_3 & c_4 & c_5 & c_6 & c_7 & \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & r_1 \oplus r_2 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & r_2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & r_4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & r_5 \oplus r_2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & r_6 \oplus r_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & r_7 \oplus r_2 \end{array}\right]$$

The permutation $\pi = (2\ 9)$ does preserve the $e_{3,0}$-block in the first three rows of G since the generator block $\pi(G_B)$ in G' is also an $e_{3,0}$-block.

Let us now describe some of the notation we will use for various subsets of the columns of $G = [I_k|A]$. We will continue to use S_I to denote the set of k columns of I_k in G, and we will continue to use S_A to denote the set of k columns of A in G. For a specific generator block G_B with weight 4 rows in G, we will use the following notation to denote the following four subsets of the columns of G:

- S_{I_B} will denote the subset of columns in S_I that are also columns of the G_B-block.
- S_{A_B} will denote the subset of columns in S_A that are also columns of the G_B-block.
- S_{I_R} will denote the remaining columns in S_I. That is, $S_{I_R} = S_I - S_{I_B}$.
- S_{A_R} will denote the remaining columns in S_A. That is, $S_{A_R} = S_A - S_{A_B}$.

We will use the following notation to denote the following pairs of subsets of the columns in $G = [I_k|A]$:

- (T_I, T_A) will denote any pair of subsets $T_I \subseteq S_I$ and $T_A \subseteq S_A$ in which $0 \leq |T_I| = |T_A| \leq k$.
- (T_{I_B}, T_{A_B}) will denote any pair of subsets $T_{I_B} \subseteq S_{I_B}$ and $T_{A_B} \subseteq S_{A_B}$ in which $0 \leq |T_{I_B}| = |T_{A_B}| \leq \min(|S_{I_B}|, |S_{A_B}|)$.
- (T_{I_R}, T_{A_R}) will denote any pair of subsets $T_{I_R} \subseteq S_{I_R}$ and $T_{A_R} \subseteq S_{A_R}$ in which $0 \leq |T_{I_R}| = |T_{A_R}| \leq \min(|S_{I_R}|, |S_{A_R}|)$.

Example 6.30 Let C be the $(18, 9, 4)$ self-dual code generated by:

$$G = [I_9|A] = \begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 \end{matrix} \\ \left[\begin{array}{ccccccccc|ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \end{matrix}$$

Let G_B be the $d_{4,1}$-block in rows r_1, r_2, r_3, r_4, r_5. The columns of the G_B-block in G are columns a_1, a_2, a_3, a_4, a_5 in I_9 and columns c_1, c_2, c_3, c_4, c_5 in A.

The six sets of columns S_I, S_A, S_{I_B}, S_{A_B}, S_{I_R}, and S_{A_R} for this G and G_B are:

$$\begin{aligned} S_I &= \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\} & S_A &= \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\} \\ S_{I_B} &= \{a_1, a_2, a_3, a_4, a_5\} & S_{A_B} &= \{c_1, c_2, c_3, c_4, c_5\} \\ S_{I_R} &= \{a_6, a_7, a_8, a_9\} & S_{A_R} &= \{c_6, c_7, c_8, c_9\} \end{aligned}$$

We will use (T_I, T_A) to denote any pair of equally sized subsets of S_I and S_A, respectively. We will use (T_{I_B}, T_{A_B}) to denote any pair of equally sized subsets of S_{I_B} and S_{A_B}, respectively. And, we will use (T_{I_R}, T_{A_R}) to denote any pair of equally sized subsets of S_{I_R} and S_{A_R}, respectively.

We will also continue to use the notation $\Pi(T_I, T_A)$ to denote the set of $(k!)^2$ permutations π of the columns of G in which the first k columns of $\pi(G)$ are a permutation of the set of k columns $(S_I - T_I) \cup T_A$ of G. One additional property of these sets of permutations we will make use of is that if $\Pi(T_I, T_A)$ is a set in which for any $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$, then if for a particular generator block G_B in G, there

exists a permutation $\pi_1 \in \Pi(T_I, T_A)$ such that π_1 preserves the G_B-block, then every permutation $\pi_2 \in \Pi(T_I, T_A)$ preserves the G_B-block. For example, if G_B is an $e_{3,0}$-block, then if $\pi_1(G_B)$ is also an $e_{3,0}$-block (and not an $e_{3,1}$-block) then $\pi_2(G_B)$ must also be an $e_{3,0}$-block (and not an $e_{3,1}$-block). The reason this is true is that for any $\pi_2 \in \Pi(T_I, T_A)$, if $[I_k|A_1] = \mathcal{RREF}(\pi_1(G))$ and $[I_k|A_2] = \mathcal{RREF}(\pi_2(G))$, then the rows and columns of A_2 must be permutations of the rows and columns of A_1, and thus the rows and columns of $\pi_2(G_B)$ are permutations of the rows and columns of $\pi_1(G_B)$.

Therefore, for a given generator block G_B, for each of the sets of permutations $\Pi(T_I, T_A)$ (such that for any $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$) either every permutation in $\Pi(T_I, T_A)$ preserves the generator block G_B, or none of the permutations in $\Pi(T_I, T_A)$ preserves the generator block G_B. This leads us to the following definition:

Definition 6.31 Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, 4)$ self-dual code such that G contains at least one generator block with weight 4 rows. Let G_B be a particular generator block with weight 4 rows in G. Then the set of permutations $\Pi(T_I, T_A)$ of the columns of G is said to *preserve the generator block* G_B if for any $\pi \in \Pi(T_I, T_A)$:

- $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A]$, and
- π preserves the generator block G_B.

The results we will prove in this section all deal with sets of pairs (T_I, T_A) in which the corresponding set of permutations $\Pi(T_I, T_A)$ preserves generator blocks in G.

Before stating the results, we will describe some of the notation we will use to denote various sets of pairs (T_I, T_A) in which the set of permutations $\Pi(T_I, T_A)$ preserves a given generator block in G. Let G_B be a particular generator block in G. We will use the following notation to denote the following sets of pairs:

- T will denote the set of all pairs (T_I, T_A) such that the set of permutations $\Pi(T_I, T_A)$ preserves the generator block G_B.

- T_B will denote the set of all pairs (T_{I_B}, T_{A_B}) such that the set of permutations $\Pi(T_{I_B}, T_{A_B})$ preserves the generator block G_B. (Recall that (T_{I_B}, T_{A_B}) denotes any pair of equally sized subsets of the columns of I_k and A, respectively, that are also columns of the G_B-block).

- T_R will denote the set of all pairs (T_{I_R}, T_{A_R}) such that the set of permutations $\Pi(T_{I_R}, T_{A_R})$ preserves the generator block G_B. (Recall that (T_{I_R}, T_{A_R}) denotes any pair of equally sized subsets of the columns of I_k and A, respectively, that do not contain any of the columns of the G_B-block).

Note that T_B and T_R are both subsets of T. Also note that since for any pair $(T_{I_R}, T_{A_R}) \in T_R$, neither T_{I_R} nor T_{A_R} contain columns of the G_B-block, T_R can also be defined as the set of all pairs (T_{I_R}, T_{A_R}) such that for any $\pi \in \Pi(T_{I_R}, T_{A_R})$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$.

Example 6.32 Consider the $(18, 9, 4)$ self-dual code C of Example 6.30, which is generated by:

$$G = [I_9|A] = \begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 \end{matrix} \\ \left[\begin{array}{ccccccccc|ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \end{matrix}$$

Recall that if G_B is the $d_{4,1}$-block in the first five rows of G then:

$$\begin{aligned} S_I &= \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\} & S_A &= \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\} \\ S_{I_B} &= \{a_1, a_2, a_3, a_4, a_5\} & S_{A_B} &= \{c_1, c_2, c_3, c_4, c_5\} \\ S_{I_R} &= \{a_6, a_7, a_8, a_9\} & S_{A_R} &= \{c_6, c_7, c_8, c_9\} \end{aligned}$$

T is the subset of pairs (T_I, T_A), where $T_I \subseteq S_I$, $T_A \subseteq S_A$, and $|T_I| = |T_A|$, such that for any $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ has the form $[I_9|A']$ and $\pi(G_B)$ is a $d_{4,1}$-block. T_B is the subset of pairs (T_{I_B}, T_{A_B}), where $T_{I_B} \subseteq S_{I_B}$, $T_{A_B} \subseteq S_{A_B}$, and $|T_{I_B}| = |T_{A_B}|$, such that for any $\pi \in \Pi(T_{I_B}, T_{A_B})$, $\mathcal{RREF}(\pi(G))$ has the form $[I_9|A']$ and $\pi(G_B)$ is a $d_{4,1}$-block. T_R is the subset of pairs (T_{I_R}, T_{A_R}), where $T_{I_R} \subseteq S_{I_R}$, $T_{A_R} \subseteq S_{A_R}$, and $|T_{I_R}| = |T_{A_R}|$, such that for any $\pi \in \Pi(T_{I_R}, T_{A_R})$, $\mathcal{RREF}(\pi(G))$ has the form $[I_9|A']$.

For example, consider the following list of pairs:

$$\begin{aligned} (T_{I_1}, T_{A_1}) &= (\{a_1\}, \{c_3\}) \\ (T_{I_2}, T_{A_2}) &= (\{a_1, a_2\}, \{c_1, c_4\}) \\ (T_{I_3}, T_{A_3}) &= (\{a_6, a_7\}, \{c_6, c_8\}) \\ (T_{I_4}, T_{A_4}) &= (\{a_1, a_6\}, \{c_3, c_6\}) \\ (T_{I_5}, T_{A_5}) &= (\{a_1\}, \{c_5\}) \\ (T_{I_6}, T_{A_6}) &= (\{a_9\}, \{c_6\}) \\ (T_{I_7}, T_{A_7}) &= (\{a_4\}, \{c_1\}) \\ (T_{I_8}, T_{A_8}) &= (\{a_4\}, \{c_7\}) \end{aligned}$$

- each of of the pairs (T_{I_1}, T_{A_1}), (T_{I_2}, T_{A_2}), (T_{I_3}, T_{A_3}), and (T_{I_4}, T_{A_4}) are elements of the set T. That is, for any permutation $\pi \in \Pi(T_{I_i}, T_{A_i})$, where $1 \leq i \leq 4$, $\mathcal{RREF}(\pi(G))$ has the form $[I_9|A']$ and $\pi(G_B)$ is a $d_{4,1}$-block.
- (T_{I_1}, T_{A_1}) and (T_{I_2}, T_{A_2}) are also elements of T_B.
- (T_{I_3}, T_{A_3}) is an element of T_R.

- (T_{I_4}, T_{A_4}) is neither an element of T_B nor an element of T_R.
- (T_{I_5}, T_{A_5}) and (T_{I_6}, T_{A_6}) are not elements of T since for any $\pi \in \Pi(T_{I_i}, T_{A_i})$, where $i = 5$ or $i = 6$, $\mathcal{RREF}(\pi(G))$ does not have the form $[I_9|A']$.
- Even though for any $\pi \in \Pi(T_{I_7}, T_{A_7})$, $\mathcal{RREF}(\pi(G))$ has the form $[I_9|A']$, (T_{I_7}, T_{A_7}) is not an element of T since $\pi(G_B)$ is a $d_{4,2}$-block and not a $d_{4,1}$-block.
- Similarly, even though for any $\pi \in \Pi(T_{I_8}, T_{A_8})$, $\mathcal{RREF}(\pi(G))$ has the form $[I_9|A']$, (T_{I_8}, T_{A_8}) is not an element of T since $\pi(G_B)$ is a $d_{4,0}$-block and not a $d_{4,1}$-block.

Our main goal in this section is to prove the following three lemmas on the three sets of pairs T, T_B, and T_R:

Lemma 6.33 The set T is equal to the set of all pairs $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R})$ where $(T_{I_B}, T_{A_B}) \in T_B$ and $(T_{I_R}, T_{A_R}) \in T_R$.

In other words, for any pair (T_I, T_A), where $T_I \subseteq S_I$, $T_A \subseteq S_A$, and $|T_I| = |T_A|$, if T_{I_B} and T_{A_B} are the columns in T_I and T_A, respectively, that are also columns of the G_B-block, then $(T_I, T_A) \in T$ if and only if $(T_{I_B}, T_{A_B}) \in T_B$ and $(T_I - T_{I_B}, T_A - T_{A_B}) \in T_R$.

Lemma 6.34 For any pair $(T_{I_B}, T_{A_B}) \in T_B$, there exists a permutation $\pi \in \Pi(T_{I_B}, T_{A_B})$ such that $\mathcal{RREF}(\pi(G)) = G$.

In other words, for any $\pi' \in \Pi(T_{I_B}, T_{A_B})$, if $\mathcal{RREF}(\pi'([I_k|A])) = [I_k|A']$ then the rows and columns of A' are permutations of the rows and columns of A.

Lemma 6.35 $|T_B|$ is a constant, which we will denote by $f_c(G_B)$.

That is, given a matrix $G = [I_k|A]$ and generator block G_B in G: the value of $|T_B|$ is the same no matter what the rows and columns of A that are not rows and columns of G_B are.

Together Lemmas 6.33, 6.34, and 6.35 give us the following theorem that we will make use of in the next section when we develop our Combination algorithm for the e_3 and d_i-codes:

Theorem 6.36 Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, 4)$ self-dual code such that G contains at least one generator block with weight 4 rows. Let G_B be any particular generator block with weight 4 rows in G. Let $(T_I, T_A) \in T$. Let T_{I_B} be the subset of columns of T_I that are also columns of the G_B-block, and let T_{A_B} be the subset of columns of T_A that are also columns of the G_B-block. Let $T_{I_R} = T_I - T_{I_B}$ and let $T_{A_R} = T_A - T_{A_B}$. Then:

- $|T_{I_B}| = |T_{A_B}|$ and $|T_{I_R}| = |T_{A_R}|$ (by Lemma 6.33),
- $(T_{I_R}, T_{A_R}) \in T_R$ (by Lemma 6.33), and
- for any $\pi \in \Pi(T_{I_B}, T_{A_B})$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$, where the rows and columns of A' are permutations of the rows and columns of A (by Lemmas 6.33 and 6.34).

Furthermore, for any pair $(T_{I_R}, T_{A_R}) \in T_R$, there are exactly $f_c(G_B)$ pairs (T_{I_B}, T_{A_B}) such that $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \in T$ (by Lemma 6.35).

Now, in order to prove Lemmas 6.33, 6.34, and 6.35, we will in fact prove Theorem 6.37 (given below) for the cases in which G_B is an $e_{3,0}$, $e_{3,1}$, $d_{i,0}$, or $d_{i,1}$-block. For Theorem 6.37 we will assume, without loss of generality, the following facts regarding the location of the generator block G_B in the matrix G: the set of columns S_{I_B} are in the first $|S_{I_B}|$ columns of I_k, the set of columns S_{A_B} are in the first $|S_{A_B}|$ columns of A, and the rows of the G_B-block are in the first $|G_B|$ rows of G. We will also assume that the rows and columns of the G_B-block are ordered as given in Figure 6.1. With these conventions, if for any $G_1 = [I_{k_1}|A_1]$ and $G_2 = [I_{k_2}|A_2]$ in which the G_B-block is the same (for example, the

G_B-block in both G_1 and G_2 is an $e_{3,0}$-block) then the sets of columns S_{I_B} and S_{A_B} are the same for both G_1 and G_2, and any pair of subsets of columns (T_{I_B}, T_{A_B}) correspond to the same subsets of columns in both G_1 and G_2.

Theorem 6.37 Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ self-dual code C, where G contains a generator block G_B that is either an $e_{3,0}$, $e_{3,1}$, $d_{i,0}$, or $d_{i,1}$-block. Then for any pair (T_{I_B}, T_{A_B}), where $T_{I_B} \in S_{I_B}$, $T_{A_B} \in S_{A_B}$, and $|T_{I_B}| = |T_{A_B}|$, one and only one of the following is true:

1. $(T_{I_B}, T_{A_B}) \in T_B$ and for any $\pi \in \Pi(T_{I_B}, T_{A_B})$, the rows and columns of A' in $[I_k|A'] = \mathcal{RREF}(\pi(G))$ are permutations of the rows and columns of A, or
2. $(T_{I_B}, T_{A_B}) \notin T_B$ and for any pair (T_{I_R}, T_{A_R}), where $T_{I_R} \in S_{I_R}$, $T_{A_R} \in S_{A_R}$, and $|T_{I_R}| = |T_{A_R}|$, $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \notin T$.

Example 6.38 Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ self-dual code where G contains a generator block G_B that is an $e_{3,0}$-block. Then with our conventions we have:

$$G = [I_k|A] = \left[\begin{array}{ccc|c||cccc|c|l} a_1 & a_2 & a_3 & & c_1 & c_2 & c_3 & c_4 & & \\ \hline 1 & 0 & 0 & 0^* & 1 & 1 & 1 & 0 & 0^* & r_1 \\ 0 & 1 & 0 & 0^* & 1 & 1 & 0 & 1 & 0^* & r_2 \\ 0 & 0 & 1 & 0^* & 1 & 0 & 1 & 1 & 0^* & r_3 \\ \hline 0 & 0 & 0 & & & & & & & \\ 0 & 0 & 0 & & & & & & & \\ \vdots & \vdots & \vdots & I_{k-3} & & & M & & & \\ 0 & 0 & 0 & & & & & & & \end{array}\right]$$

Note that $S_{I_B} = \{a_1, a_2, a_3\}$ and $S_{A_B} = \{c_1, c_2, c_3, c_4\}$.

Theorem 6.37 tells us that no matter what the integer k or the $(k-3) \times k$ matrix M is, for any pair (T_{I_B}, T_{A_B}), where $T_{I_B} \subseteq S_{I_B}$, $T_{A_B} \subseteq S_{A_B}$, and $|T_{I_B}| = |T_{A_B}|$, one and only one of the following is true:

- $(T_{I_B}, T_{A_B}) \in T_B$ and for any $\pi \in \Pi(T_{I_B}, T_{A_B})$, the rows and columns of A' in $[I_k|A'] = \mathcal{RREF}(\pi(G))$ are permutations of the rows and columns of A, or,

- $(T_{I_B}, T_{A_B}) \notin T_B$ and for any pair (T_{I_R}, T_{A_R}), $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \notin T$.

For example, $(\{a_1\}, \{c_1\}) \in T_B$, and thus, for any $\pi \in \Pi(\{a_1\}, \{c_1\})$, the rows and columns of A' in $[I_k|A'] = \mathcal{RREF}(\pi(G))$ are permutations of the rows and columns of A. The pair $(\{a_1\}, \{c_4\}) \notin T_B$, and thus, for any pair (T_{I_R}, T_{A_R}), the pair $(\{a_1\} \cup T_{I_R}, \{c_4\} \cup T_{A_R}) \notin T$.

Once we have proved Theorem 6.37, we will have proved Lemmas 6.33, 6.34, and 6.35, and thus we will have proved Theorem 6.36. That is, by Theorem 6.37, we have Lemma 6.33 since:

- for any $(T_{I_B}, T_{A_B}) \in T_B$, by the first point in Theorem 6.37 we know that for any $\pi \in \Pi(T_{I_B}, T_{A_B})$ the rows and columns of A' in $[I_k|A'] = \mathcal{RREF}(\pi(G))$ are permutations of the rows and columns of A, which implies that for any pair $(T_{I_R}, T_{A_R}) \in T_R$, $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \in T$. Therefore, the set of all pairs $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R})$ is contained in the set T.

- for any pair $(T_I, T_A) \in T$, if T_{I_B} and T_{A_B} are columns in T_I and T_A, respectively, that are also columns of the G_B-block, then since for any $\pi \in \Pi(T_I, T_A)$, $\pi(G_B)$ is still a G_B-block, we must have $|T_{I_B}| = |T_{A_B}|$, and, by the second point in Theorem 6.37, (T_{I_B}, T_{A_B}) must be in T_B (for if $(T_{I_B}, T_{A_B}) \notin T_B$ then the fact that $(T_I, T_A) \in T$ contradicts the second point of Theorem 6.37). Furthermore, since $(T_{I_B}, T_{A_B}) \in T_B$, by the first point of Theorem 6.37, $(T_I - T_{I_B}, T_A - T_{A_B}) \in T_R$. Therefore, T is contained in the set of all pairs $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R})$.

Thus, the set T is equal to the set of all pairs $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R})$, where $(T_{I_B}, T_{A_B}) \in T_B$ and $(T_{I_R}, T_{A_R}) \in T_R$, which gives us Lemma 6.33. Lemma 6.34 follows immediately from

the first point in Theorem 6.37. Lemma 6.35 follows immediately from the if and only if condition in Theorem 6.37.

Let us now prove Theorem 6.37 for each of the cases when G_B is an $e_{3,0}$-block, an $e_{3,1}$-block, a $d_{i,0}$-block, or a $d_{i,1}$-block. We will consider the cases when G_B is either an $e_{3,0}$-block or an $e_{3,1}$-block first, and then the cases when G_B is either a $d_{i,0}$-block or a $d_{i,1}$-block.

The $e_{3,0}$ and $e_{3,1}$-blocks

Let us now prove Theorem 6.37 for the case when G_B is an $e_{3,0}$-block:

a_1	a_2	a_3		c_1	c_2	c_3	c_4		
1	0	0	0*	1	1	1	0	0*	r_1
0	1	0	0*	1	1	0	1	0*	r_2
0	0	1	0*	1	0	1	1	0*	r_3

Note that for an $e_{3,0}$-block, $|S_{I_B}| = 3$ and $|S_{A_B}| = 4$.

Consider B, the weight 4 set generated by the $e_{3,0}$-block. There are exactly 7 weight 4 words in B:

$$r_1, r_2, r_3, r_1 \oplus r_2, r_1 \oplus r_3, r_2 \oplus r_3, r_1 \oplus r_2 \oplus r_3$$

For each $\vec{w} \in B$, we will use the phrase "the columns of $\vec{w}$" to refer to the four columns c of G in which the rows of G that sum to $\vec{w}$, sum to a one in column c. For example, the columns of $\vec{w} = r_1$ are a_1, c_1, c_2, c_3 and the columns of $\vec{w} = r_1 \oplus r_2$ are a_1, a_2, c_3, c_4.

Notice that in the weight 4 set B there is exactly one weight 4 word $\vec{w}$ such that three of the four columns of $\vec{w}$ are columns of S_{I_B} (namely, $\vec{w} = r_1 \oplus r_2 \oplus r_3$). Now, for any pair $(T_{I_B}, T_{A_B}) \in T_B$ and permutation $\pi \in \Pi(T_{I_B}, T_{A_B})$, this must also be true for the weight 4 set B' generated by $\pi(G_B)$. That is, there must exist one and only one $\vec{w'} \in B'$ such that three of the four columns of $\vec{w'}$ are in $S'_{I_B} = (S_{I_B} - T_{I_B}) \cup T_{I_A}$. Therefore, for each of the $\sum_{i=0}^{3} \binom{3}{i}\binom{4}{i} = 35$ possibilities for (T_{I_B}, T_{A_B}), we know that if $(S_{I_B} - T_{I_B}) \cup T_{I_A}$

does not contain three of the columns of one of the seven weight 4 words in B, then for any $\pi \in \Pi(T_{I_B}, T_{A_B})$, $\pi(G_B)$ cannot be an $e_{3,0}$-block, and therefore $(T_{I_B}, T_{A_B}) \notin T_B$. Furthermore, for any pair (T_{I_R}, T_{A_R}), we know $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \notin T$ since, for any $\pi \in \Pi(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R})$, $\pi(G_B)$ still cannot be an $e_{3,0}$-block. Thus, we now only have to consider the pairs (T_{I_B}, T_{A_B}) in which the set $(S_{I_B} - T_{I_B}) \cup T_{A_B}$ contains three of the four columns of one of the seven weight 4 words in B.

We have $\binom{7}{1} = 7$ choices for the weight 4 word $\vec{w} \in B$ that has three columns in $(S_{I_B} - T_{I_B}) \cup T_{A_B}$, and for each $\vec{w}$, we have $\binom{4}{3} = 4$ choices for the three columns in I_k. Therefore, the number of pairs (T_{I_B}, T_{A_B}) in which one of the seven weight 4 words in B has three of its four columns in $(S_{I_B} - T_{I_B}) \cup T_{A_B}$ is $7 \times 4 = 28$. We will prove Theorem 6.37 for the case when G_B is an $e_{3,0}$-block by showing that for each of these 28 possibilities for the set (T_{I_B}, T_{A_B}) there exists a permutation $\pi \in \Pi(T_{I_B}, T_{A_B})$ such that $\mathcal{RREF}(\pi(G)) = G$. Note that this also shows us that $(T_{I_B}, T_{A_B}) \in T_B$.

Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ self-dual code C, where G contains an $e_{3,0}$-block. Then, with just the columns of A given, we have:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \\ \hline 0 & 1 & 1 & 1 & X \\ 0 & 0 & 0 & 0 & Y \end{array}\right] \end{array} \begin{array}{ll} r_1 & (a_1) \\ r_2 & (a_2) \\ r_3 & (a_3) \end{array}$$

Above the horizontal line are the rows of the $e_{3,0}$-block in A and below the horizontal line are all the different ways the remaining rows of A may intersect the $e_{3,0}$-block.

In our proof, we will perform our permutations of the columns of G (and elementary row operations) by applying sequences of $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c operations on A. Recall that if entry (i, j) of A is non-zero, then an $\text{ADD}_{r,c}(i, j)$ on A results in the matrix A' where $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$, where π is the transposition that swaps column

i of I_k with column j of A. A SWAP_r operation simply swaps two columns of I_k and row-reduces, and a SWAP_c operation simply swaps two columns of A.

Let us begin by first showing that for each of the seven weight 4 words in B there exists a permutation π (only involving the columns of the $e_{3,0}$-block) that permutes three of the four columns of the weight 4 word into I_k. In terms of the matrix A and the three operations $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c what we want to show is that there exists a sequence S of $\text{ADD}_{r,c}$ operations on A that results in a matrix $S(A)$ in which three of the four columns of $\vec{w}$ are columns of I_k in $[I_k|S(A)]$. We then want to show that there exists a sequence of SWAP_r and/or SWAP_c operations on $S(A)$ that results in A being reproduced.

For the weight 4 word $r_1 \oplus r_2 \oplus r_3$ we are done, since three of its four columns are already in I_k. Let us now consider the weight 4 word $r_1 \oplus r_2$. The four columns of $r_1 \oplus r_2$ in G are a_1, a_2, c_3, c_4. If we perform an $\text{ADD}_{r,c}(3,3)$ on A, then the matrix that results is:

$$A' = \begin{array}{c} \begin{matrix} c_1 & c_2 & a_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc|c} 0 & 1 & 1 & 1 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \\ \hline 1 & 1 & 1 & 0 & X \\ 0 & 0 & 0 & 0 & Y \end{array}\right] \end{array} \begin{array}{ll} r_1 \oplus r_3 & (a_1) \\ r_2 & (a_2) \\ r_3 & (c_3) \end{array}$$

Notice that three of the four columns of the weight 4 word $r_1 \oplus r_2$ in A are now columns of I_k in $[I_k|A']$, and if we swap the first and fourth columns of A' then A is reproduced. Similar sequences of $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c operations exist for the weight 4 words $r_1 \oplus r_3$ and $r_2 \oplus r_3$. This completes the case when $\vec{w}$ is the sum of two rows of G. Let us now consider the weight 4 word r_1. If we perform an $\text{ADD}_{r,c}(2,1)$ on A followed by an

$\mathrm{ADD}_{r,c}(3,2)$, then the matrix that results is:

$$A'' = \begin{array}{c} \begin{matrix} a_2 & a_3 & c_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc|c} 1 & 0 & 1 & 1 & 0^* \\ 0 & 1 & 1 & 1 & 0^* \\ 1 & 1 & 1 & 0 & 0^* \\ \hline 1 & 1 & 0 & 1 & X \\ 0 & 0 & 0 & 0 & Y \end{array}\right] \end{array} \begin{array}{ll} r_1 \oplus r_2 & (a_1) \\ r_3 & (c_1) \\ r_3 \oplus r_2 & (c_2) \end{array}$$

Notice that three of the four columns of the weight 4 word r_1 in A are now columns of I_k in $[I_k|A'']$, and if we swap the first and third columns of A'' then A is reproduced. Similar sequences of $\mathrm{ADD}_{r,c}$, SWAP_r, and SWAP_c operations exist for the weight 4 words r_2 and r_3. This completes the case when $\vec{w}$ is a row of G. Thus, for each of the seven weight 4 words $\vec{w}$ in B, there exists a sequence S of $\mathrm{ADD}_{r,c}$ operations on A that results in a matrix $S(A)$, where the rows and columns of $S(A)$ are permutations of the rows and columns of A, such that three of the four columns of $\vec{w}$ are columns of I_k in $[I_k|S(A)]$.

Since for each $\vec{w} \in B$ there exists a sequence S of $\mathrm{ADD}_{r,c}$ operations that results in a matrix $S(A)$ whose row and columns are permutations of the rows and columns of A, such that three of the four columns of $\vec{w}$ are columns of I_k in $[I_k|S(A)]$, all we have left to show is that for each of the $\binom{4}{3} = 4$ combinations of three columns chosen from the four columns a_1, a_2, a_3, c_1 of $r_1 \oplus r_2 \oplus r_3$, there exists a sequence S of $\mathrm{ADD}_{r,c}$ operations that results in a matrix $S(A)$, whose rows and columns are permutations of the rows and columns of A, in which our choice of three columns are columns of I_k in $[I_k|S(A)]$. For the combination a_1, a_2, a_3 we are done since they are already columns of I_k in $[I_k|A]$. For the combination a_1, a_2, c_1, if we perform an $\mathrm{ADD}_{r,c}(3,1)$ operation on A, then the resulting

matrix is:

$$A' = \begin{array}{c} \begin{matrix} a_3 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc|c} 1 & 1 & 0 & 1 & 0^* \\ 1 & 1 & 1 & 0 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \\ \hline 0 & 1 & 1 & 1 & X \\ 0 & 0 & 0 & 0 & Y \end{array}\right] \end{array} \begin{array}{ll} r_1 \oplus r_3 & (a_1) \\ r_2 \oplus r_3 & (a_2) \\ r_3 & (c_1) \end{array}$$

Notice that the three columns a_1, a_2, c_1 are now columns of I_k in $[I_k|A']$, and if we swap the third and fourth columns of A' then A is reproduced. Similar sequences of $\text{ADD}_{r,c}$, SWAP_r, and SWAP_c operations exist for the combinations a_1, a_3, c_1 and a_2, a_3, c_1. Thus, for each of the 4 combinations of three columns chosen from the four columns of $r_1 \oplus r_2 \oplus r_3$, there exists a sequence S of $\text{ADD}_{r,c}$ operations on A that results in a matrix $S(A)$ whose rows and columns are permutations of the rows and columns of A, in which our choice of three columns are columns of I_k in $[I_k|S(A)]$.

What we have shown is that for each of the $7 \times 4 = 28$ pairs (T_{I_B}, T_{A_B}), where $(S_{I_B} - T_{I_B}) \cup T_{A_B}$ contains three of the four ones of some $\vec{w} \in B$, there exists a permutation $\pi \in \Pi(T_{I_B}, T_{A_B})$ such that $\mathcal{RREF}(\pi(G)) = G$. Since, for each of the remaining $35-28 = 7$ pairs (T_{I_B}, T_{A_B}), we have already shown that for any pair (T_{I_R}, T_{A_R}), $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \notin T$, we have proved Theorem 6.37 for the case when G_B is an $e_{3,0}$-block. Note that we have also shown that $f_c(e_{3,0}) = 28$.

We can also prove Theorem 6.37 for the case when G_B is an $e_{3,1}$-block in a very similar manner, with the major differences being that for an $e_{3,1}$-block $|S_{I_B}| = 4, |S_{A_B}| = 3$, and for the weight 4 set B generated by the $e_{3,1}$-block, there is exactly one weight 4 word $\vec{w} \in B$ such that three of the four columns of $\vec{w}$ are columns of A. For the $e_{3,1}$-block we also have $f_c(e_{3,1}) = 28$.

The $d_{i,0}$ and $d_{i,1}$-blocks

Let us now prove Theorem 6.37 for our remaining two cases: the case when G_B is a $d_{i,0}$-block, $i \geq 1$, and the case when G_B is a $d_{i,1}$-block, $i \geq 2$.

We will first prove Theorem 6.37 for the case when G_B is a $d_{i,1}$-block, $i \geq 2$:

a_1	a_2		a_{i-1}	a_i	a_{i+1}		c_1	c_2	c_3	c_4		c_{i+1}		
1	0	$\cdots$	0	0	0	0*	1	1	1	0	$\cdots$	0	0*	r_1
0	1	$\cdots$	0	0	0	0*	1	1	0	1	$\cdots$	0	0*	r_2
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	
0	0	$\cdots$	1	0	0	0*	1	1	0	0	$\cdots$	1	0*	r_{i-1}
0	0	$\cdots$	0	1	0	0*	1	0	1	1	$\cdots$	1	x	r_i
0	0	$\cdots$	0	0	1	0*	0	1	1	1	$\cdots$	1	x	r_{i+1}

Note that for a $d_{i,1}$-block, $|S_{I_B}| = |S_{A_B}| = i + 1$.

Consider B, the weight 4 set generated by the $d_{i,1}$-block. The weight 4 words in B consist of the first $i - 1$ rows of G, the sum of any pair of the first $i - 1$ rows of G, the sum of the last two rows of G, and the sum of the last two rows of G with any of the first $i - 1$ rows of G. Consider the following set S of $i + 1$ disjoint pairs of columns of G:

$$S = \{(a_1, c_3), (a_2, c_4), \ldots, (a_{i-1}, c_{i+1}), (a_i, a_{i+1}), (c_1, c_2)\}$$

Note that the $2i + 2$ columns in this list are the columns of the $d_{i,1}$-block. Notice that for each of these pairs of columns, for any $\vec{w} \in B$, $\vec{w}$ has the same value in both columns. For example, every $\vec{w} \in B$ is either 0 0 or 1 1 in columns (c_1, c_2). Furthermore, for any other pair of columns of the $d_{i,1}$-block this is not true. For example, for the pair (c_3, c_4), $\vec{w} = r_1$ has a value of 1 in column c_3 and a value of 0 in column c_4. Therefore, the columns of the $d_{i,1}$-block in G can be uniquely partitioned into the set S of disjoint pairs of columns such that for any weight 4 word $\vec{w} \in B$, the value of $\vec{w}$ is the same in both columns of each pair.

Notice that one of the pairs of columns in S (namely (a_i, a_{i+1})) consists of two columns from I_k, another one of the pairs (namely (c_1, c_2)) consists of two columns from A, and the remaining $i-1$ pairs each consist of one column from I_k and one column from A. Thus, for any set $(T_{I_B}, T_{A_B}) \in T_B$, the set of $i+1$ columns $(S_{I_B} - T_{I_B}) \cup T_{A_B}$ must also contain both columns of one (and only one) of the $i+1$ pairs of columns in S. Therefore, for each of the $\sum_{j=0}^{i+1} \binom{i+1}{j}\binom{i+1}{j} = \binom{2i+2}{i+1}$ possibilities for (T_{I_B}, T_{A_B}), we know that if $(S_{I_B} - T_{I_B}) \cup T_{I_A}$ does not contain one (and only one) of the pairs of columns in S, then for any $\pi \in \Pi(T_{I_B}, T_{A_B})$, $\pi(G_B)$ cannot be a $d_{i,1}$-block, and therefore $(T_{I_B}, T_{A_B}) \notin T_B$. Furthermore, for any pair (T_{I_R}, T_{A_R}), we know $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \notin T$ since, for any $\pi \in \Pi(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R})$, $\pi(G_B)$ still cannot be a $d_{i,1}$-block. Thus, we now only have to consider the pairs (T_{I_B}, T_{A_B}) in which the set $(S_{I_B} - T_{I_B}) \cup T_{A_B}$ contains one (and only one) of the $(i+1)$ pairs of columns in S.

We have $\binom{i+1}{1}$ choices for the one and only pair of columns in S that has both columns in $(S_{I_B} - T_{I_B}) \cup T_{A_B}$, and for each these choices we have $\binom{i}{1}$ choices for the one and only pair of columns in S that has neither column in $(S_{I_B} - T_{I_B}) \cup T_{A_B}$, and for the remaining $i-1$ pairs we have 2^{i-1} choices for the one column that occurs in $(S_{I_B} - T_{I_B}) \cup T_{A_B}$. Therefore, the number of pairs (T_{I_B}, T_{A_B}) in which the set $(S_{I_B} - T_{I_B}) \cup T_{A_B}$ contains both columns of one and only one of the $i+1$ pairs is $\binom{i+1}{1}\binom{i}{1}2^{i-1} = i(i+1)2^{i-1}$.

Using the same method we used for the $e_{3,0}$-block, it can be shown that for each of these $i(i+1)2^{i-1}$ possibilities for the sets (T_{I_B}, T_{A_B}) there exists a permutation $\pi \in \Pi(T_{I_B}, T_{A_B})$ such that $\mathcal{RREF}(\pi(G)) = G$. Note that this also shows us that $(T_{I_B}, T_{A_B}) \in T_B$. Since for each of the remaining pairs (T_{I_B}, T_{A_B}) we have already shown that, for any pair (T_{I_R}, T_{A_R}), $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \notin T$, we have proved Theorem 6.37 for the case when G_B is an $d_{i,1}$-block. We have also shown that $f_c(d_{i,1}) = i(i+1)2^{i-1}$.

We can also prove Theorem 6.37 for the case when G_B is an $d_{i,0}$-block, $i \geq 1$, in a very similar manner, with the major differences being that for an $d_{i,0}$-block $|S_{I_B}| = i$, $|S_{A_B}| = i+2$, and for the $(i+1)$ pairs of columns, one and only one pair has both columns

in A and the remaining i pairs have one column in I_k and the other in A. This gives us $f_c(d_{i,0}) = \binom{i+1}{1} 2^i = (i+1)2^i$.

This completes our proof of Theorem 6.37.

Thus, we have now proved Lemmas 6.33, 6.34, and 6.35, which together give us Theorem 6.36.

Furthermore, for any pair $(T_{I_R}, T_{A_R}) \in T_R$, there are exactly $f_c(G_B)$ pairs (T_{I_B}, T_{A_B}) such that $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \in T$, where $f_c(G_B)$ is defined as follows:

$$\begin{aligned} f_c(e_{3,0}) &= 28 \\ f_c(e_{3,1}) &= 28 \\ f_c(d_{i,0}) &= (i+1)2^i \qquad i \geq 1 \\ f_c(d_{i,1}) &= i(i+1)2^{i-1} \qquad i \geq 2. \end{aligned}$$

Theorem 6.36 can be easily extended to consider sets of permutations that preserve more than one generator block in G. For example, suppose G contains two generator blocks, G_{B_1} and G_{B_2}, with weight 4 rows. Let T now denote the set of all pairs (T_I, T_A) in which the set of permutations $\Pi(T_I, T_A)$ preserves both G_{B_1} and G_{B_2} (i.e. for any $\pi \in \Pi(T_I, T_A)$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A]$ and π preserves both of the generator blocks G_{B_1} and G_{B_2}). Let (T_I, T_A) be any pair in T. Let $T_{I_{B_1}}$, $T_{I_{B_2}}$ denote the subsets of columns of T_I that are also columns of the generator blocks G_{B_1} and G_{B_2}, respectively, and let $T_{A_{B_1}}$, $T_{A_{B_2}}$ denote the subsets of columns of T_A that are also columns of the generator blocks G_{B_1} and G_{B_2}, respectively. Let $T_{I_B} = T_{I_{B_1}} \cup T_{I_{B_2}}$, let $T_{A_B} = T_{A_{B_1}} \cup T_{A_{B_2}}$, let $T_{I_R} = T_I - T_{I_B}$, and let $T_{A_R} = T_A - T_{A_B}$. Then by Theorem 6.36:

- $|T_{I_{B_1}}| = |T_{A_{B_1}}|$ and $|T_{I_{B_2}}| = |T_{A_{B_2}}|$, which implies that $|T_{I_R}| = |T_{A_R}|$,
- $(T_{I_R}, T_{A_R}) \in T_R$, where T_R is the set of all pairs (T_{I_R}, T_{A_R}) in which the set of permutations $\Pi(T_{I_R}, T_{A_R})$ preserves both of the generator blocks G_{B_1} and G_{B_2}, and

- for any $\pi_1 \in \Pi(T_{I_{B_1}}, T_{A_{B_1}})$, if $[I_k|A_1] = \mathcal{RREF}(\pi_1(G))$ then the rows and columns of A_1 are permutations of the rows and columns of A, and for any $\pi_2 \in \Pi(T_{I_{B_2}}, T_{A_{B_2}})$, if $[I_k|A_2] = \mathcal{RREF}(\pi_2(G))$ then the rows and columns of A_2 are permutations of the rows and columns of A, which together imply that, for any $\pi \in \Pi(T_{I_B}, T_{A_B})$, if $[I_k|A'] = \mathcal{RREF}(\pi(G))$ then the rows and columns of A' are permutations of the rows and columns of A.

Furthermore, for any pair $(T_{I_R}, T_{A_R}) \in T_R$, there are exactly $f_c(G_{B_1}) \times f_c(G_{B_2})$ pairs (T_{I_B}, T_{A_B}) such that $(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \in T$.

Generalizing this, we have our final result of this section:

Theorem 6.39 Let G be the generator matrix for a $(2k, k, 4)$ self-dual code such that G contains at least one generator block with weight 4 rows. Let $G_{B_1}, G_{B_2}, \ldots, G_{B_q}$ be the generator blocks with weight 4 rows in G. Let $(T_I, T_A) \in T$, where T is the set of all pairs (T_I, T_A) in which the set of permutations $\Pi(T_I, T_A)$ preserves each of the generator blocks G_{B_j}, for $j = 1, 2, \ldots, q$. Let T_{I_B} be the subset of all columns in T_I that are also columns of the generator blocks G_{B_j}, and let T_{A_B} be the subset of columns in T_A that are also columns of the generator blocks G_{B_j}. Let $T_{I_R} = T_I - T_{I_B}$ and let $T_{A_R} = T_A - T_{A_B}$. Then:

- $|T_{I_B}| = |T_{A_B}|$ and $|T_{I_R}| = |T_{A_R}|$,
- $(T_{I_R}, T_{A_R}) \in T_R$, where T_R is the set of all pairs (T_{I_R}, T_{A_R}) in which the set of permutations $\Pi(T_{I_R}, T_{A_R})$ preserves each of the generator blocks G_{B_j}, for $j = 1, 2, \ldots, q$, and
- for any $\pi \in \Pi(T_{I_B}, T_{A_B})$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$, where the rows and columns of A' are permutations of the rows and columns of A.

Furthermore, for any pair $(T_{I_R}, T_{A_R}) \in T_R$ there are exactly $\prod_{j=1}^{q} f_c(G_{B_j})$ pairs (T_{I_B}, T_{A_B})

such that $\Pi(T_{I_B} \cup T_{I_R}, T_{A_B} \cup T_{A_R}) \in T$, where $f_c(G_B)$ is defined as follows:

$$\begin{aligned} f_c(e_{3,0}) &= 28 \\ f_c(e_{3,1}) &= 28 \\ f_c(d_{i,0}) &= (i+1)2^i \qquad i \geq 1 \\ f_c(d_{i,1}) &= i(i+1)2^{i-1} \qquad i \geq 2. \end{aligned}$$

Note that, as was pointed out earlier, since for any pair $(T_{I_R}, T_{A_R}) \in T_R$, neither T_{I_R} nor T_{A_R} contains any of the columns of the generator blocks G_B, T_R is just the set of all pairs (T_{I_R}, T_{A_R}) such that for any $\pi \in \Pi(T_{I_R}, T_{A_R})$, $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$.

In the next section, we will see how we can use Theorem 6.39 to reduce the number of sets of permutations $\Pi(T_I, T_A)$ we need to consider in the the Combination algorithm we will develop for the e_3 and d_i-codes.

6.7 Generator Blocks with Weight 4 Rows In the Combination Algorithm

In this section, we will develop our Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes, $i \geq 1$.

Let us begin by reviewing the role of the Combination algorithm. Let C be any $(2k, k, 4)$ self-dual code. Let $\mathcal{G}$ denote some subset of the generator matrices in $\mathcal{G}_{[I|A]}(C)$, with the property that for any pair $G_1 = [I_k|A_1], G_2 = [I_k|A_2] \in \mathcal{G}_{[I|A]}(C)$, where the rows and columns of A_1 are permutations of the rows and columns of A_2, G_1 is an element of $\mathcal{G}$ if and only if G_2 is an element of $\mathcal{G}$. Recall that given any $G = [I_k|A] \in \mathcal{G}$, the role of the Combination algorithm is to produce a list $L' = \{(A'_1, cnt'_1), (A'_2, cnt'_2), \ldots, (A'_m, cnt'_m)\}$ such that for any $G'' = [I_k|A''] \in \mathcal{G}$:

- There exists at least one A'_i in L' such that the rows and columns of A'_i are permutations of the rows and columns of A''.

- If *pcnt* is the number of permutations $\pi_r\pi_c$ of the rows and columns of A'' such that $\pi_r\pi_c(A'') = A''$, and if $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ are the matrices in L' whose rows and columns are permutations of the rows and columns of A'', then $\sum_{j=1}^{n} cnt'_{i_j} = acnt/pcnt$ where *acnt* is the size of the automorphism group of C.

So far our Combination algorithm has found such a list L' by selecting and performing one permutation π from each of the $\binom{2k}{k}$ sets of permutations $\Pi(T_I, T_A)$. For each $G' = \mathcal{RREF}(\pi(G))$ produced, if G' has the form $[I_k|A']$ and is an element of $\mathcal{G}$ then A' is recored in L', otherwise it is discarded. The main problem with this algorithm is that we have to select and perform one permutation from each of the $\binom{2k}{k}$ sets of permutations $\Pi(T_I, T_A)$.

Our Combination algorithm is determining, for each of the $\binom{2k}{k}$ sets $\Pi(T_I, T_A)$, whether or not each of the permutations in $\Pi(T_I, T_A)$ results in a matrix that is an element of $\mathcal{G}$. However, if for some subset T of pairs (T_I, T_A) we can determine for each pair (without actually selecting/performing permutations) that for any $\pi \in \Pi(T_I, T_A)$ the resulting matrix is not an element of $\mathcal{G}$, then the Combination algorithm does not have to consider any member of the set of permutations $\Pi(T_I, T_A)$ where $(T_I, T_A) \in T$. Furthermore, if for some subset T of pairs (T_I, T_A), we can determine (without actually selecting or performing permutations) that for any two pairs $(T_{I_1}, T_{A_1}), (T_{I_2}, T_{A_2}) \in T$ there exist permutations $\pi_1 \in \Pi(T_{I_1}, T_{A_1})$ and $\pi_2 \in \Pi(T_{I_2}, T_{A_2})$ such that $\mathcal{RREF}(\pi_1(G)) = \mathcal{RREF}(\pi_2(G))$ then, as long as we know $|T|$, the Combination algorithm need only consider one of these sets of permutations. In other words, the Combination algorithm does not necessarily have to consider each of the $\binom{2k}{k}$ sets of permutations $\Pi(T_I, T_A)$.

In this section, we will develop a Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes that will allow us to reduce the number of sets of permutations $\Pi(T_I, T_A)$ that the algorithm has to consider. The set of generator matrices $\mathcal{G}$ for which the algorithm will produce a list L', is the set of all generator matrices $G'' \in \mathcal{G}_{[I|A]}(C)$ where G'' has a ***maximal combination of generator blocks with weight 4 rows*** (which we will define later).

As we shall see, limiting our algorithm to such generator matrices will allow us to make use of Theorem 6.39 of Section 6.6 in order to reduce the number of sets $\Pi(T_I, T_A)$ considered by the algorithm.

6.7.1 Maximal Combinations of Generator Blocks With Weight 4 Rows

We will begin this section by defining the subset $\mathcal{G}$ of generator matrices in $\mathcal{G}_{[I|A]}(C)$ for which our Combination algorithm will produce the list L'. The set will be based on the generator blocks with weight 4 rows contained in each matrix.

Let C be any $(2k, k, 4)$ self-dual code that is either an e_3-code or a d_i-code, $i \geq 1$. Then the set of generator matrices $\mathcal{G}_{[I|A]}(C)$ may contain many pairs of matrices in which at least one of their corresponding generator blocks differ. That is, for each e_3-block (if any), there may exist matrices in $\mathcal{G}_{[I|A]}(C)$ in which it is an $e_{3,0}$-block, and there may exist matrices in which it corresponds to an $e_{3,1}$-block. For each d_1-block (if any), there may exist matrices in which it corresponds to a $d_{1,0}$-block, a $d_{1,3}$-block, or a $d_{1,4}$-block. And, for each d_i-block, $i \geq 2$, there may exist matrices in which it corresponds to a $d_{i,0}$-block, a $d_{i,1}$-block, a $d_{i,2}$-block, a $d_{i,3}$-block, or a $d_{i,4}$-block. Furthermore, for any pair of matrices $G_1, G_2 \in \mathcal{G}_{[I|A]}(C)$, the number of each of the different generator blocks in G_1 and G_2 may differ. For example, G_1 may contain two $e_{3,0}$-blocks and zero $e_{3,1}$-blocks, and G_2 may contain one $e_{3,0}$-block and one $e_{3,1}$-block. In other words, there may exist many different combinations of the amounts of each of the different generator blocks that occur in the generator matrices in $\mathcal{G}_{[I|A]}(C)$.

Let $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ be any two generator matrices in $\mathcal{G}_{[I|A]}(C)$. Suppose G_1 and G_2 do not contain the same amounts of each of the different generator blocks. For example, suppose G_1 contains exactly two $e_{3,0}$-blocks and suppose G_2 contains exactly one $e_{3,0}$-block and one $e_{3,1}$-block. Then the rows and columns of A_1 cannot be permutations of the rows and columns of A_2. Therefore, we can define our set of generator matrices $\mathcal{G}$ to be some subset of generator matrices in $\mathcal{G}_{[I|A]}(C)$ that all have

the same amounts of each of the different generator blocks. In other words, we can define $\mathcal{G}$ by defining how many of the e_3-blocks in each matrix are $e_{3,0}$-blocks, how many of the e_3-blocks in each matrix are $e_{3,1}$-blocks, how many of the d_1-blocks in each matrix are $d_{i,0}$-blocks, and so on.

Ideally, it would be nice if we could define our set $\mathcal{G}$ to be the set of all generator matrices G in $\mathcal{G}_{[I|A]}(C)$ in which all the e_3-blocks in G are $e_{3,0}$-blocks, and in which all the d_i-blocks, $i \geq 1$, are $d_{i,0}$-blocks. If this were possible, then our Combination algorithm could produce the list L' by first permuting the columns of the input matrix until a $G \in \mathcal{G}$ is produced, and then selecting and performing permutations from just the sets of permutations $\pi \in \Pi(T_{I_R}, T_{A_R})$ in which T_{I_R} and T_{A_R} do not contain any of the columns of the generator blocks in G. If $G_{B_1}, G_{B_2}, \ldots, G_{B_p}$ are the generator blocks in G (which are all generator blocks with weight 4 rows) then for each π selected that results in a matrix $G' = \mathcal{RREF}(\pi(G))$ that has the form $[I_k|A']$, if we record the pair $(A', \prod_{i=1}^{p} f_c(G_B))$ in L', then by Theorem 6.39 of Section 6.6, the resulting list L' will have the desired properties. However, as the following example demonstrates, this is not generally possible.

Example 6.40 Let C be the $(14, 7, 4)$ self-dual code generated by:

$$G = [I_7|A] = \left[\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right]$$

There are two generator blocks for weight 4 sets in G: the e_3-block (which is an $e_{3,0}$-block) in the first three rows of G and the e_3-block in the last four rows of G (which is an $e_{3,1}$-block). Let G_{B_1} be the e_3-block in the first three rows of G and let G_{B_2} be the

e_3-block in the last four rows of G. Let $G' = [I_7|A']$ be any generator matrix in $\mathcal{G}_{[I|A]}(C)$, and let G'_{B_1} and G'_{B_2} be the generator blocks in G' that correspond to G_{B_1} and G_{B_2}, respectively. Then one possibility for the generator blocks in G' is that G'_{B_1} may be an $e_{3,0}$-block and G'_{B_2} may be an $e_{3,1}$-block. Another possibility is that G'_{B_1} may be an $e_{3,1}$-block and G'_{B_2} may be an $e_{3,0}$-block. However, G'_{B_1} and G'_{B_2} cannot both be $e_{3,0}$-blocks, since each $e_{3,0}$-block takes up four distinct columns in A but there are only 7 columns. It is also not possible for both G'_{B_1} and G'_{B_2} to be $e_{3,1}$-blocks, since each $e_{3,1}$-block takes up four distinct columns in I_7.

Therefore, we cannot define our set $\mathcal{G}$ to be the set of all generator matrices G in $\mathcal{G}_{[I|A]}(C)$ in which all the e_3-blocks are $e_{3,0}$-blocks and all the d_i-blocks, $i \geq 1$, are $d_{i,0}$-blocks. Furthermore, $\mathcal{G}_{[I|A]}(C)$ may only contain generator matrices whose generator blocks are *not* all generator blocks with weight 4 rows. In other words, it is not generally possible to define our set $\mathcal{G}$ to contain generator matrices whose generator blocks are all generator blocks with weight 4 rows. Keeping this in mind, one property that the generator matrices in our set $\mathcal{G}$ will possess is that they will all have a *full combination of generator blocks with weight 4 rows*:

Definition 6.41 Let $G = [I_k|A]$ be any matrix in $\mathcal{G}_{[I|A]}(C)$. Let $G_{B_1}, G_{B_2}, \ldots, G_{B_q}$ be the subset of generator blocks in G that are generator blocks with weight 4 rows. For any $G' \in \mathcal{G}_{[I|A]}(C)$, let $G'_{B_1}, G'_{B_2}, \ldots, G_{B'_q}$ denote the generator blocks in G' that correspond to the generator blocks with weight 4 rows in G. Then G is said to have a *full combination of generator blocks with weight 4 rows* if and only if, for any $G' \in \mathcal{G}_{[I|A]}(C)$ in which each of the generator blocks G'_{B_j} in G', $j = 1, 2, \ldots, q$, is the same as the corresponding generator block G_{B_j} in G, none of the remaining generator blocks in G' are generator blocks with weight 4 rows.

In other words, G has a *full combination of generator blocks with weight 4 rows* if and only if for any permutation π of the columns of G that results in a matrix $G' = \mathcal{RREF}(\pi(G))$ that has the form $[I_k|A]$, if π transforms one of the generator blocks in G

that is *not* a generator block with weight 4 rows into a generator block with weight 4 rows, then π must also transform at least one of the generator blocks with weight 4 rows in G into a different generator block.

Example 6.42 Let C be the $(20, 10, 4)$ self-dual code generated by:

$$G = [I_{10}|A] = \left[\begin{array}{cccccccccc|cccccccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{array}$$

G contains the following three generator blocks for weight 4 sets:

- G_{B_1}: the $d_{3,0}$-block in the first three rows of G,
- G_{B_2}: the $d_{3,1}$-block in rows 4, 5, 6, and 7 of G, and
- G_{B_3}: the $d_{1,4}$-block in the last three rows of G.

The generator blocks with weight 4 rows in G are the $d_{3,0}$-block G_{B_1} and the $d_{3,1}$-block G_{B_2}. This G has a full combination of generator blocks with weight 4 rows. That is, there does not exist a permutation π of the columns of G, where $\mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$, in which $\pi(G_{B_1})$ is a $d_{3,0}$-block, $\pi(G_{B_2})$ is a $d_{3,1}$-block, and $\pi(G_{B_3})$ is a $d_{1,0}$-block (i.e. the d_1-block that is a generator block with weight 4 rows). The reason for this is that the columns of each generator block in G do not intersect, and since the $d_{3,0}$

and $d_{3,1}$-blocks together take up 10 of the 11 columns in A, there is only one column left for the d_1-block, and therefore it cannot be a $d_{1,0}$-block since a $d_{1,0}$-block would take up exactly 3 columns in A.

There may exist matrices G_1 and G_2 in $\mathcal{G}_{[I|A]}(C)$ in which both G_1 and G_2 have full combinations of generator blocks with weight 4 rows, but G_1 and G_2 do not contain the same amounts of each of the different generator blocks with weight 4 rows. For example, if the generator matrices in $\mathcal{G}_{[I|A]}(C)$ contain only one generator block, and if this generator block is a d_2-block, then the generator block in G_1 may be a $d_{2,0}$-block, and the generator block in G_2 may be a $d_{2,1}$-block. Therefore, we will define a *unique* full combination of generator blocks with weight 4 rows, which we will name a *maximal combination of generator blocks with weight 4 rows.*

In order to define a maximal combination of generator blocks with weight 4 rows, we will first define a precedence on the generator blocks with weight 4 rows. From highest to lowest, this precedence is:

$$e_{3,0}, e_{3,1}, \ldots, d_{i,0}, d_{i,1}, d_{i-1,0}, d_{i-1,1}, \ldots, d_{2,0}, d_{2,1}, d_{1,0}.$$

Using this precedence, we will now define a comparison relationship for the different combinations of amounts of generator blocks with weight 4 rows:

Definition 6.43 Let G_1 and G_2 be any two matrices in $\mathcal{G}_{[I|A]}(C)$. If G_1 and G_2 have the same amount of each of the different generator blocks with weight 4 rows then G_1 and G_2 are said to have *equal* combinations of generator blocks with weight 4 rows. Otherwise, if G_B is the highest precedence generator block with weight 4 rows in which G_1 and G_2 do not contain the same amount, then if G_1 has more G_B-blocks than G_2 then G_1 is said to have a *larger* combination of generator blocks with weight 4 rows than G_2, else G_1 is said to have a *smaller* combination of generator blocks with weight 4 rows than G_2.

This definition leads us to our definition of a maximal combination of generator blocks with weight 4 rows:

Definition 6.44 Let $G = [I_k|A]$ be any matrix in $\mathcal{G}_{[I|A]}(C)$ such that G has a full combination of generator blocks with weight 4 rows. Then G is said to have a *maximal combination of generator blocks with weight 4 rows* if and only if for any $G' \in \mathcal{G}_{[I|A]}(C)$ that has a full combination of generator blocks with weight 4 rows, either G and G' have equal combinations of generator blocks with weight 4 rows, or G has a larger combination of generator blocks with weight 4 rows than G'.

Example 6.45 Consider the generator matrix $G = [I_{10}|A]$ for the $(20, 10, 4)$ self-dual code C of Example 6.42. G contains three generator blocks:

- the $d_{3,0}$-block G_{B_1},
- the $d_{3,1}$-block G_{B_2}, and,
- the $d_{1,4}$-block G_{B_3}.

Let π be any permutation of the columns of G such that $G' = \mathcal{RREF}(\pi(G))$ has the form $[I_{10}|A']$. Then one of the following two statements will be true about the generator blocks in G'.

- G' has the same amount of each of the different generator blocks with weight 4 rows as G. That is, one of the d_3-blocks in G' is a $d_{3,0}$-block, the other d_3-block in G' is a $d_{3,1}$-block, and the d_1-block in G' is not a $d_{1,0}$-block.
- G' does not contain the same amount of each of the different generator blocks with weight 4 rows as G. Furthermore, if G_B is the highest precedence generator block with weight 4 rows in which G and G' do not contain the same amount, then G has more. In other words, there do not exist permutations π in which, for example, the two d_3-blocks in G' are both $d_{3,0}$-blocks.

Therefore, G has a maximal combination of generator blocks with weight 4 rows.

The set of generator matrices $\mathcal{G}$ for which our Combination algorithm will find a list L' for is the subset of all generator matrices in $\mathcal{G}_{[I|A]}(C)$ that have a maximal combination of generator blocks with weight 4 rows. We will denote this subset of generator matrices by $\mathcal{G}_{max}(C)$.

6.7.2 The Combination Algorithm for the e_3 and d_i-Codes

Our Combination algorithm for the e_3 and d_i-codes will find a list L' for the set of generator matrices $\mathcal{G}_{max}(C)$. The algorithm will produce L' in two stages. The first stage will produce a list L_{max} of $k \times k$ matrices A' containing exactly one representative for each of the *different* maximal combinations of generator blocks with weight 4 rows. That is, for any $G'' = [I_k|A''] \in \mathcal{G}_{max}(C)$, if $G''_{B_1}, G''_{B_2}, \ldots, G''_{B_q}$ are the generator blocks with weight 4 rows in G'', then L_{max} will contain exactly one matrix A' such that each of the corresponding generator blocks G'_{B_j} in $G' = [I_k|A']$ is the same as the generator block G''_{B_j} in G'', for $j = 1, 2, \ldots, q$. The second stage of our algorithm will apply our original Combination algorithm to each $A' \in L_{max}$, but will only consider the sets of permutations $\Pi(T_{I_R}, T_{A_R})$ of the columns of $G' = [I_k|A']$ that do not include any of the columns of the generator blocks with weight 4 rows in G'. In other words, for each $A' \in L_{max}$, the second stage of our algorithm uses A' (and Theorem 6.39 of Section 6.6) to perform and select permutations from the sets of permutations $\Pi(T_I, T_A)$ of the columns of the input matrix $G = [I_k|A]$, that result in matrices with the same combination of generator blocks with weight 4 rows as $G' = [I_k|A']$.

Let $G = [I_k|A]$ be the input matrix to our Combination algorithm. The first stage of our algorithm will produce the list L_{max} by performing one permutation π of the columns of G for each of the different possible combinations for the generator blocks with weight 4 rows in $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$. The algorithm will find and perform each of these permutations recursively. That is, the algorithm will first find one permutation π_1 for each of the different possible combinations for the generator blocks with weight 4 rows

contained in the first $j-1$ generator blocks of $G_1 = \mathcal{RREF}(\pi_1(G))$. The algorithm will then either skip to the next level of the recursion (indicating the j^{th} generator block will not be a generator block with weight 4 rows in the final matrix), or the algorithm will find one permutation π_2 (if it exists) of the columns of G_1 for each of the different possibilities for the generator block $\pi_2(G_{B_j})$, where G_{B_j} is the j^{th} generator block in G_1 and π_2 preserves the generator blocks with weight 4 rows contained in the first $j-1$ generator blocks of G_1. For example, if G_{B_j} is a d_3-block, then the algorithm will either skip to the next level of the recursion (indicating that G_{B_j} will be neither a $d_{3,0}$ nor $d_{3,1}$-block in the final matrix), or the algorithm will find one permutation π_2 that results in $\pi_2(G_{B_j})$ being a $d_{3,0}$-block, and one permutation that results in $\pi_2(G_{B_j})$ being a $d_{3,1}$-block.

As the next theorem demonstrates, we can search for the permutation π_2 by considering only permutations of the columns of G_1 that do not include any of the columns of the generator blocks with weight 4 rows contained in the first $j-1$ generator blocks of G_1:

Theorem 6.46 Suppose there exists a permutation π_2' of the columns of G_1 that preserves the generator blocks with weight 4 rows contained in the first $j-1$ generator blocks of G_1. Then there exists a permutation π_2 of the columns of G_1 that does not involve any of the columns of the generator blocks with weight 4 rows contained in the first $j-1$ generator blocks of G_1 such that $\pi_2'(G_{B_j})$ and $\pi_2(G_{B_j})$ are the same.

Proof This follows immediately from Theorem 6.39 of Section 6.6. That is, if $\Pi(T_I, T_A)$ is the set of permutations of the columns of G_1 to which π_2' belongs, then if T_{I_R}, T_{A_R} are the subsets of columns of T_I, T_A, respectively, that are not columns of the generator blocks with weight 4 rows contained in the first $j-1$ generator blocks of G_1, then by Theorem 6.39, for any $\pi_2 \in \Pi(T_{I_R}, T_{A_R})$, the rows and columns of A_2 are permutations of the rows and columns of A_2', where $[I_k|A_2] = \mathcal{RREF}(\pi_2(G_1))$ and $[I_k|A_2'] = \mathcal{RREF}(\pi_2'(G_2))$. Therefore, π_2 is a permutation of the columns of G_1 that does not involve any of the columns of the generator blocks with weight 4 rows contained in the first $j-1$ generator

blocks of G_1 such that $\pi_2(G_{B_j})$ is the same as $\pi_2'(G_{B_j})$. □

As an example of how we can find such a permutation π_2, suppose G_{B_j} is an e_3-block and suppose we want $\pi_2(G_{B_j})$ to be an $e_{3,0}$-block. If G_{B_j} is already an $e_{3,0}$-block then $\pi_2 = I$ will do, where I is the identity permutation. Suppose G_{B_j} is an $e_{3,1}$-block:

a_1	a_2	a_3	a_4		c_1	c_2	c_3		
1	0	0	0	0*	1	1	1	0*	r_1
0	1	0	0	0*	1	1	0	x	r_2
0	0	1	0	0*	1	0	1	x	r_3
0	0	0	1	0*	0	1	1	x	r_4

Then it can be proved that a permutation π_2 of the columns of $G_1 = [I_k|A_1]$ exists such that $\pi_2(G_{B_j})$ is an $e_{3,0}$-block, $\mathcal{RREF}(\pi_2(G))$ has the form $[I_k|A_2]$, and π_2 preserves the generator blocks with weight 4 rows contained in the first $j-1$ generator blocks of G_1 if and only if there exists at least one column c in A_1 that contains a one in row r_4, where c is neither a column of the $e_{3,1}$-block G_{B_j} nor a column of any of the generator blocks with weight 4 rows contained in the first $j-1$ generator blocks of G_1. If this is the case, then $\pi_2 = (a_4\ c)$ is such a permutation.

We will now give a formal description of our algorithm for producing the list L_{max} of $k \times k$ matrices A' containing exactly one representative for each of the different maximal combinations of generator blocks with weight 4 rows. The algorithm makes use of a Boolean array $FourBlock[1..p]$, where p is the number of generator blocks in G, to keep track of the subset of generator blocks for which a permutation π_2 was found that resulted in a generator block with weight 4 rows. The algorithm also makes us of the following two functions (of which formal descriptions will not be given):

- $AddToList(A', FourBlock, L_{max})$: This function inserts A' into L_{max}, if appropriate. That is, if the combination of generator blocks with weight 4 rows in $G' = [I_k|A']$ is equal (i.e. has the same amount of each of the different types)

to the combination of generator blocks with weight 4 rows of any $G'' = [I_k|A'']$, where $A'' \in L_{max}$, then A'' is added to L_{max}, else if the combination of generator blocks with weight 4 rows in $G' = [I_k|A']$ is larger than the combination of generator blocks with weight 4 rows of any $G'' = [I_k|A'']$, where $A'' \in L_{max}$, then L_{max} is first emptied, and then A' is inserted. In order to prevent us from inserting more that one representative for each of the different maximal combinations of generator blocks with weight 4 rows, for each entry $FourBlock[j]$ that has a value of $false$, the function will make sure that the generator block G'_{B_j} in $G' = [I_k|A']$ is *not* a generator block with weight 4 rows.

- **Boolean** *PermuteBlockToForm*$(A', j, form, FourBlock, A'')$: This function finds a permutation π_2 (if it exists) of the columns of $G'' = [I_k|A'']$ that results in $\pi_2(G_{B_j})$ being the desired generator block with weight 4 rows (which is indicated by the variable $form$), where π_2 does not involve any of the columns of each generator block $G'_{B_{j_0}}$ in $G' = [I_k|A']$ in which $FourBlock[j_0] = true$. If such a permutation π_2 is found, then the function returns a value of $true$, otherwise the function returns a value of $false$. If a value of $true$ is returned then the function will set $FourBlock[j]$ to $true$, and the function will set the $k \times k$ matrix A'' by finding $[I_k|A''] = \mathcal{RREF}(\pi_2([I_k|A']))$.

We are now ready to give a formal description of our algorithm. The algorithm is recursive and is invoked with the call *FindMaximalList*$(A, 1, p, FourBlock, L_{max})$ where $FourBlock[j] = false$, $j = 1, 2, \ldots, p$, and $L_{max} = \emptyset$.

Algorithm 6.47 *FindMaximalList*$(A', j, p, FourBlock, L_{max})$:

- Input: The $k \times k$ matrix A', the integers j and p, where $1 \leq j \leq p$, and the Boolean array $FourBlock[1..p]$. $G' = [I_k|A']$ generates a $(2k, k, 4)$ self-dual code C, where C is either an e_3-code or a d_i-code, $i \geq 1$. G' contains exactly p generator blocks for weight 4 sets: $G'_{B_1}, G'_{B_2}, \ldots, G'_{B_p}$.

- Output: A list L_{max} of $k \times k$ matrices A' containing exactly one representative for each of the different maximal combinations of generator blocks with weight 4 rows.

begin

if $j > p$ **then**

$AddToList(A', FourBlock, L_{max})$.

else begin

if G_{B_j} is an e_3-block **then begin**

if $PermuteBlockToForm(A', j, e_{3,0}, FourBlock, A'')$ **then**

$FindMaximalList(A'', j+1, p, FourBlock, L_{max})$.

if $PermuteBlockToForm(A', j, e_{3,1}, FourBlock, A'')$ **then**

$FindMaximalList(A'', j+1, p, FourBlock, L_{max})$.

end if

else if G_{B_j} is a d_i-block, $i \geq 2$ **then begin**

if $PermuteBlockToForm(A', j, d_{i,0}, FourBlock, A'')$ **then**

$FindMaximalList(A'', j+1, p, FourBlock, L_{max})$.

if $PermuteBlockToForm(A', j, d_{i,1}, FourBlock, A'')$ **then**

$FindMaximalList(A'', j+1, p, FourBlock, L_{max})$.

$FourBlock[j] = false$.

$FindMaximalList(A', j+1, p, FourBlock, L_{max})$.

end else if

else

if $PermuteBlockToForm(A', j, d_{1,0}, FourBlock, A'')$ **then**

$FindMaximalList(A'', j+1, p, FourBlock, L_{max})$.

$FourBlock[j] = false$.

$FindMaximalList(A', j+1, p, FourBlock, L_{max})$.

end else

end else

end

Let us now consider how we can use the list L_{max} produced by Algorithm 6.47 in order to produce the list L' produced by our Combination algorithm for the e_3 and d_i-codes, $i \geq 1$.

Let $G = [I_k|A]$ be the input matrix to our Combination algorithm, and let $L_{max} = \{A'_1, A'_2, \ldots, A'_r\}$ be the list produced by Algorithm 6.47. Let T denote the set of all pairs (T_I, T_A), where T_I and T_A are equally sized subsets of the columns of I_k and A, respectively, such that $(T_I, T_A) \in T$ if and only if for any $\pi \in \Pi(T_I, T_A)$, $G' = \mathcal{RREF}(\pi(G))$ has the form $[I_k|A']$, and G' has a maximal combination of generator blocks with weight 4 rows. For each $A'_i \in L_{max}$, let $T(A'_i)$ denote the set of all pairs $(T_I, T_A) \in T$, such that for any $\pi \in \Pi(T_I, T_A)$, the generator blocks with weight 4 rows in $G' = \mathcal{RREF}(\pi(G))$ are the same as their corresponding generator blocks in $G'_i = [I_k|A'_i]$. Then, for any pair $A'_{i_1}, A'_{i_2} \in L_{max}$, $T(A_{i_1}) \cap T(A_{i_2}) = \emptyset$, and $T = T(A_1) \cup T(A_2) \cup \ldots T(A_r)$. In other words, for any pair $(T_I, T_A) \in T$, there exists one and only one $A'_i \in L_{max}$ such that $(T_I, T_A) \in T(A'_i)$.

For each $A'_i \in L_{max}$, our Combination algorithm will use A'_i to perform and select permutations from the each set of permutations $\Pi(T_I, T_A)$, where $(T_I, T_A) \in T(A'_i)$. It will do this by performing permutations of the columns of $G'_i = [I_k|A'_i]$ that preserve the generator blocks with weight 4 rows in G'_i. Why we can do this is explained next.

Let T'_i denote the set of all pairs (T'_I, T'_A), where T'_I and T'_A are equally sized subsets of the columns of I_k and A'_i, respectively, such that $(T'_I, T'_A) \in T'_i$ if and only if for any $\pi' \in \Pi(T'_I, T'_A)$, $G'' = \mathcal{RREF}(\pi'(G))$ has the form $[I_k|A'']$, and π' preserves the generator blocks with weight 4 rows in G'_i. Then there is a one-to-one mapping between the set $T(A'_i)$ and the set T'_i. That is, if π_i is the permutation of the columns of G that results in G'_i (i.e. $\mathcal{RREF}(\pi_i(G)) = G'_i$) and if π_i^{-1} is the inverse of π_i (which implies that $\mathcal{RREF}(\pi_i^{-1}(G'_i)) = G$) then:

- For any $\pi \in \Pi(T_I, T_A)$, where $(T_I, T_A) \in T(A_i')$, if we rewrite π as $\pi'\pi_i$, then $\mathcal{RREF}(\pi(G)) = \mathcal{RREF}(\pi'\pi_i(G)) = \mathcal{RREF}(\pi'(G_i'))$, which implies that $(T_I', T_A') \in T_i'$, where $\Pi(T_I', T_A')$ is the set of permutations to which π' belongs. In other words, (T_I, T_A) is mapped to (T_I', T_A'). Furthermore, every element in $T(A_i')$ is mapped to a unique element of T_i'.

- Conversely, for any $\pi' \in \Pi(T_I', T_A')$, where $(T_I', T_A') \in T_i'$, if we rewrite π' as $\pi\pi_i^{-1}$, then $\mathcal{RREF}(\pi\pi_i^{-1}(G_i')) = \mathcal{RREF}(\pi\pi_i^{-1}\pi_i(G)) = \mathcal{RREF}(\pi(G))$, which implies that $(T_I, T_A) \in T(A_i')$, where $\Pi(T_I, T_A)$ is the set of permutations to which π belongs. In other words, (T_I', T_A') is mapped to (T_I, T_A). Furthermore, every element of T_i' is mapped to a unique element of $T(A_i')$.

In other words, if we select and perform permutations from the sets of permutations $\Pi(T_I', T_A')$ of the columns of G_i' in order to determine whether or not $(T_I', T_A) \in T_i'$, what we are doing is selecting and performing permutations from the sets of permutations $\Pi(T_I, T_A)$ of the columns of G in order to determine whether or not $(T_I, T_A) \in T(A_I')$.

The advantage in selecting and performing permutations from the sets of permutations $\Pi(T_I', T_A')$ of the columns of G_i' to determine whether or not $(T_I', T_A') \in T_I'$ is that, by Theorem 6.39 of Section 6.6, we only have to select and perform permutations from the sets of permutations $\Pi(T_{I_R}', T_{A_R}')$, where T_{I_R}' and T_{A_R}' do not contain any of the columns of the generator blocks with weight 4 rows in G_i'. That is, for every pair $(T_I', T_A') \in T_i'$, if (T_{I_R}', T_{A_R}') are the columns of (T_I', T_A') that are not columns of the generator blocks with weight 4 rows in G_i', then (T_{I_R}', T_{A_R}') is also an element of T_i'. Furthermore, if $G_{B_1}', G_{B_2}', \ldots, G_{B_q}'$ are the generator blocks with weight 4 rows in G_i', then for any pair $(T_{I_R}', T_{A_R}') \in T_i'$, we know there are exactly $\prod_{j=1}^{q} f_c(G_{B_j}')$ pairs $(T_I', T_A') \in T_i'$ in which (T_{I_R}', T_{A_R}') is the subset of columns of (T_I', T_A') that are not columns of the generator blocks with weight 4 rows in G_i', and for any $\pi_1 \in \Pi(T_{I_R}', T_{A_R}')$ and $\pi_2 \in \Pi(T_I', T_A')$, the rows and columns of A_1 are permutations of the rows and columns of A_2, where $\mathcal{RREF}(\pi_1(G_i')) = [I_k|A_1]$ and $\mathcal{RREF}(\pi_2(G_i')) = [I_k|A_2]$. Thus, we do not lose any

information by considering only the sets of permutations $\Pi(T'_{I_R}, T'_{A_R})$ that do not involve any of the columns of the generator blocks with weight 4 rows in G'_i.

We are now ready to give a formal description of our two stage Combination algorithm for the e_3 and d_i-codes. The algorithm is given in terms of the $k \times k$ matrix A and the $\text{ADD}_{r,c}$ operation. The algorithm makes use of two functions, *OrderGenBlocks* and *QuickOrderMatrix*, in order to reduce the number of pairs A'_{i_1} and A'_{i_2} that are inserted into L' in which the rows and columns of A'_{i_1} are simply permutations of the rows and columns of A'_{i_2}. The function *OrderGenBlocks*(A') (which we will not define formally) orders the rows and columns of A' in such a way that:

- The rows and columns (in A') of each generator block with weight 4 rows occur in consecutive rows and columns.
- The rows and columns (in A') of each generator block are ordered as given in Figure 6.1 of Section 6.6.
- If G_{B_1} and G_{B_2} are generator blocks with weight 4 rows such that G_{B_1} has a higher precedence than G_{B_2}, then G_{B_1} occurs before G_{B_2} (i.e. the rows of G_{B_1} are located above the rows of G_{B_2} and the columns (in A') of G_{B_1} are located to the left of the columns (in A') of G_{B_2}).
- The rows and columns (in A') of the generator blocks with weight 4 rows all occur before the remaining rows and columns of A'.

The *QuickOrderMatrix* function (which is defined in Chapter 3) is used to order the remaining rows and columns of A'.

Algorithm 6.48 *CombinationAlgorithm*(A, L'):

- Input: Any $k \times k$ matrix A such that $G = [I_k|A]$ generates a $(2k, k)$ self-dual code C that is either an e_3-code or a d_i-code, $i \geq 1$.

- Output: A list L' of pairs (A', cnt'), where A' is a $k \times k$ matrix and cnt' is an integer. The list $L' = \{(A_1, cnt_1), (A_2, cnt_2), \ldots, (A_m, cnt_m)\}$ returned has the two properties that for any $G'' = [I_k|A''] \in \mathcal{G}_{max}(C)$:

  1. There exists at least one A'_i in L' such that the rows and columns of A'_i are permutations of the rows and columns of A''.
  2. If $pcnt$ is the number of permutations $\pi_r\pi_c$ of the rows and columns of A'' such that $\pi_r\pi_c(A'') = A''$, and if $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ are the subsets of matrices in L' whose rows and columns are permutations of the rows and columns of A'', then $\sum_{j=1}^{n} cnt'_{i_j} = acnt/pcnt$, where $acnt$ is the size of the automorphism group of C.

begin

Let $L' = \emptyset$.

Let $L_{max} = \emptyset$.

Let p be the number of generator blocks for weight 4 sets in $G = [I_k|A]$.

FindMaximalList$(A, 1, p, FourBlock, L_{max})$.

for each $A_1 \in L_{max}$ **do begin**

OrderGenBlocks(A_1).

Let $G_{B_1}, G_{B_2}, \ldots, G_{B_q}$ be the generator blocks with weight 4 rows in $G_1 = [I_k|A_1]$.

Let r be the last row in A_1 of the last generator with weight 4 rows in $G_1 = [I_k|A_1]$.

Let c be the last column in A_1 of the last generator block with weight 4 rows in $G_1 = [I_k|A_1]$.

Let S_r be the set of the last $k - r$ rows of A_1.

Let S_c be the set of the last $k - c$ columns of A_1.

Let $\Pi_1 = \{(T_r, T_c) \mid T_r \subseteq S_r, T_c \subseteq S_c, 0 \leq |T_r| = |T_c| \leq \min(k - r, k - c)\}$.

for each $(T_r, T_c) \in \Pi_1$ **do**

if there exists a sequence S of $|T_r|$ $\text{ADD}_{r,c}$ operations on A_1 that includes each row of T_r and each column of T_c **then**

Let $A' = S(A_1)$.

QuickOrderMatrix$(A', r+1, A')$.

if there exists a pair (A', cnt') in L' **then**

Increment cnt' by $\prod_{j=1}^{q} f_c(G_{B_j})$.

else

Insert the pair $(A', \prod_{j=1}^{q} f_c(G_{B_j}))$ into L'.

end if

end for

end

As an example of the advantage Algorithm 6.48 has over our original Combination algorithm, consider the following matrix:

$$G = [I_7|A] = \left[\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right]$$

Our original Combination algorithm will select and perform permutations from each of the $\binom{14}{7} = 3432$ sets of permutations $\Pi(T_I, T_A)$ of the columns of G. Let G_{B_1} be the generator block in the first three rows of G and let G_{B_2} be the generator block in the last four rows of G. Algorithm 6.48 will first find the list $L_{max} = \{A_1, A_2\}$, where the generator blocks G'_{B_1}, G'_{B_2} in $G_1 = [I_7|A_1]$ are $e_{3,0}$ and $e_{3,1}$-blocks, respectively, and the generator blocks G''_{B_1}, G''_{B_2} in $G_2 = [I_7|A_2]$ are $e_{3,1}$ and $e_{3,0}$-blocks, respectively. For

both A_1 and A_2, since every column in both A_1 and A_2 is a column of a generator block with weight 4 rows, the algorithm only considers the trivial set of permutations $\Pi(\emptyset, \emptyset)$. That is, instead of selecting and performing permutations from 3432 sets of permutations, Algorithm 6.48 will only select and perform permutations from $1 + 1 = 2$ sets of permutations.

We now have our Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes. In the next section, we will develop our Permutation algorithm for the $(2k, k, 4)$ e_3 and d_i-codes, which, together with the Combination algorithm, gives us our Unique Representative algorithm for the $(2k, k, 4)$ e_3 and d_i-codes.

6.8 Weight 4 Blocks In the Permutation Algorithm

In this section, we will develop our Permutation algorithm for the $(2k, k, 4)$ e_3 and d_i-codes, $i \geq 1$.

Let $L' = \{(A'_1, cnt'_1), (A'_2, cnt'_2), \ldots, (A'_m, cnt'_m)\}$ be the list of pairs produced by our Combination algorithm for the e_3 and d_i-codes. Recall that for each $A'_i \in L'$, the role of the Permutation algorithm is to order to rows and columns of A'_i, producing a unique matrix A''_i. The algorithm then selects, as our unique representative A_0, the largest A''_i produced. The Permutation algorithm also finds *acnt*, the size of the automorphism group of C, by counting the number of permutations $\pi_r\pi_c$ of the rows and columns of A'_i such that $\pi_r\pi_c(A'_i) = A''_i$. That is, if $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ are the matrices in L' whose rows and columns are permutations of the rows and columns of A_0 (i.e. $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ are the matrices in L' that are uniquely ordered to A_0) then the Permutation algorithm computes $acnt = pcnt(\sum_{j=1}^{n} cnt'_{i_j})$, where *pcnt* is the number of permutations $\pi_r\pi_c$ of the rows and columns of any one of the matrices A'_{i_j}, $1 \leq j \leq n$, such that $\pi_r\pi_c(A'_{i_j}) = A_0$.

Whereas our Combination algorithm for the e_3 and d_i-codes reduced the number of permutations it performed by considering the generator blocks with weight 4 rows in

$G = [I_k|A]$, our Permutation algorithm for the e_3 and d_i-codes will reduce the number of permutations it performs by considering the weight 4 blocks in $G = [I_k|A]$ (i.e. the weight 4 rows contained in each of the generator blocks with weight 4 rows). More specifically, our Permutation algorithm will reduce the number of permutations $\pi_r\pi_c$ of the rows and columns of A it performs by considering the *weight 3 blocks* in A.

6.8.1 The Weight 3 Blocks in A

Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ self-dual code C. In Section 4.2 of Chapter 4, we found all the different weight 3 blocks that may occur in A. They are the E_3-block, the E_4-block, and the D_i-blocks, $i \geq 1$. If C is either an e_3-code or a d_i-code, then the only weight 3 blocks that may occur in A are the E_3-blocks and the D_i-blocks (see Figure 6.2). Since we are only working with the e_3 and d_i-codes, in the remainder of this section whenever we use the term *weight 3 blocks* we will just mean the E_3 and D_i-blocks.

$$\begin{array}{ccc} \begin{matrix} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \end{matrix} & & \begin{matrix} \overbrace{}^{i} \\ 1\ 1\ 1\ 0\ \cdots\ 0\ 0^* \\ 1\ 1\ 0\ 1\ \cdots\ 0\ 0^* \\ \vdots\ \vdots\ \vdots\ \vdots\ \quad\ \vdots\ \vdots \\ 1\ 1\ 0\ 0\ \cdots\ 1\ 0^* \end{matrix} \Bigg\} i \\ \text{the } E_3\text{-block} & & \text{the } D_i\text{-blocks, } i \geq 1 \end{array}$$

Figure 6.2: The E_3 and D_i weight 3 blocks.

Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, 4)$ self-dual code C, where C is either an e_3-code or a d_i-code, $i \geq 1$. Then the weight 3 rows of A can be partitioned into the m disjoint sets $A_{B_1}, A_{B_2}, \ldots, A_{B_m}$, where each A_{B_j} is a weight 3 block. Furthermore, since the rows of any two weight 3 blocks in A do not intersect, the subset of columns of A that contain at least one 1 in the weight 3 rows of A can be partitioned into the

m disjoint sets $S_{B_1}, S_{B_2}, \ldots, S_{B_m}$, where each S_{B_j} are the *columns* of the weight 3 block A_{B_j}. That is, each S_{B_j} is the set of the columns of A that are non-zero in the rows of the weight 3 block A_{B_j}. In other words, any weight 3 row in A belongs to one and only one weight 3 block in A, and any column of A that intersects at least one weight 3 row, intersects one and only one weight 3 block, and thus belongs to one and only one weight 3 block in A.

Let r be the number of weight 3 rows in A and let c be the number of columns in A that intersect any one of the weight 3 blocks in A. We will say the weight 3 blocks of A are in *descending order* if and only if:

- the first r rows of A contain the weight 3 rows of A,
- the first c columns of A contain the columns of the weight 3 blocks of A,
- the rows and columns of each weight 3 block occur in consecutive rows and columns,
- the rows and columns of each weight 3 block are ordered as in Figure 6.2,
- the E_3-blocks in A occur before the D_i-blocks, and
- the D_{i_1}-blocks in A occur before the D_{i_2}-blocks, where $i_1 > i_2 \geq 1$.

In other words, we will say the weight 3 blocks in A are in descending order if and only if:

$$A = \left[\begin{array}{cccc|c} A_{B_1} & 0 & \cdots & 0 & \\ 0 & A_{B_2} & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \\ 0 & 0 & \cdots & A_{B_m} & \\ \hline & & M & & \end{array}\right]$$

where the rows in M all have weight ≥ 5, the rows and columns of each weight 3 block A_{B_j} are ordered as in Figure 6.2, and, if $1 \leq j_1 < j_2 \leq m$, then $A_{B_{j_1}}$ has *higher precedence* than $A_{B_{j_2}}$, where our precedence of the weight 3 blocks is, from highest to lowest:

$$E_3, \ldots, D_i, D_{i-1}, \ldots, D_2, D_1.$$

The weight 3 blocks in the matrices produced in our Permutation algorithm will be in descending order.

Note that our ordering of the weight 3 blocks is unique. That is, if the rows and columns of A_1 are permutations of the rows and columns of A_2, and if the weight 3 blocks in both A_1 and A_2 are in descending order, then the first r rows of both A_1 and A_2 are equal (where r is the number of weight 3 rows in both A_1 and A_2). Also note that if the weight 3 blocks in A are in descending order, then the first r rows and the first r components of the columns of A are in descending order.

6.8.2 The Permutation Algorithm for the e_3 and d_i-Codes

For each A' in the list L' produced by our Combination algorithm, our Permutations algorithm for the e_3 and d_i-codes will produce the largest matrix A'' whose rows and columns are permutations of the rows and columns of A', such that:

- the weight 3 blocks in A'' are in descending order,
- the columns of A'' are in descending order, and
- the last $k - r$ rows of A'' are in descending order, where r is the number of weight 3 rows in A''.

The unique matrix A_0 returned by our Permutation algorithm will be the largest A'' produced.

One of the advantages in defining our unique A'' in this manner is that since the columns of A'' are in descending order we only need to consider permutations of the rows of A' (as we learned in Chapter 3). That is, since the columns of A'' are in descending order, and since for any permutation π_r of the rows of A' there exists one and only one permutation π_c of the columns of A' such that the columns of $\pi_r\pi_c(A')$ are in descending order, our Permutation algorithm only needs to consider the different permutations π_r of the rows of A'. For each permutation π_r of the rows of A' considered, the algorithm only needs to produce the one permutation π_c of the columns of A' that orders the columns of $\pi_r(A')$ in descending order. Therefore, in developing our Permutation algorithm, we need only consider the different permutations π_r of the rows of A'.

Another advantage in the way we have defined our unique A'' is as follows: Let r be the number of weight 3 rows in A'. Suppose we first order the weight 3 blocks of A' in descending order. Then the first r rows of A' all have weight 3, and the last $k-r$ rows of A' all have weight ≥ 5. Therefore, if $\pi_r\pi_c$ is a permutation of the rows and columns of A' such that $\pi_r\pi_c(A') = A''$, then π_r can be written as a product of two disjoint permutations $\pi_{r_2}\pi_{r_1}$, where π_{r_1} is a permutation of the first r rows of A' (i.e. the weight 3 rows of A') and π_{r_2} is a permutation of the last $k-r$ rows of A' (i.e. the weight ≥ 5 rows of A'). Thus, if we first order the weight 3 blocks of A' in descending order, our Permutation algorithm need only consider the different permutations of the first r rows and the last $k-r$ rows of A'.

We are now ready to describe the general method our Permutation algorithm will use to perform permutations of the rows and columns of A'. Let r denote the number of weight 3 rows in A'. For each A' in L', our Permutation algorithm will first order the weight 3 blocks of A' in descending order. The algorithm will then perform each permutation π_r of the first r rows of A' that result in a matrix $\pi_r\pi_c(A')$ whose weight 3 rows are in descending order, where π_c is any permutation of the columns of A' that orders the first r components of the columns of $\pi_r(A')$ in descending order. For each $\pi_r\pi_c(A')$ produced, the algorithm will then use the *UniqueOrder* algorithm (given in Section 6.4)

to perform permutations of the last $k - r$ rows and the columns of $\pi_r\pi_c(A')$. In other words, for each $\pi_r\pi_c(A')$ produced, our Permutation algorithm will use the *UniqueOrder* algorithm to find the largest matrix whose last $k - r$ rows and columns are permutations of the last $k - r$ rows and the columns of $\pi_c\pi_r(A')$.

For each A' in L', let us now consider how our Permutation algorithm can produce the unique matrix A'', and compute *pcnt*, the number of permutations $\pi_r\pi_c$ of the rows and columns of A' such that $\pi_c\pi_r(A') = A''$. Unless stated otherwise, in the remainder of this section, for any permutation π_r of the first r rows of A' (where r is the number of weight 3 rows in A'), whenever we write the matrix $\pi_r\pi_c(A')$, the permutation π_c will be any permutation of the columns of A' that orders the first r components of the columns of $\pi_r(A')$ in descending order.

Let P_r denote the set of all permutations π_r of the first r rows of A' such that the weight 3 blocks of $\pi_r\pi_c(A')$ are in descending order. Let $A'_1 = \pi_{r_1}\pi_{c_1}(A')$ and let $A'_2 = \pi_{r_2}\pi_{c_2}(A')$, where $\pi_{r_1}, \pi_{r_2} \in \Pi_r$. Let A''_1, A''_2 denote the unique matrices that result from applying the *UniqueOrder* algorithm to the last $k-r$ rows of A'_1 and A'_2, respectively. Then A''_1 may or may not be equal to A''_2.

Example 6.49 Consider the following two 11×11 matrices:

$$A_1' = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \quad A_2' = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

The rows and columns of A_1' are permutations of the rows and columns of A_2'. Note that A_2' is just A_1' with the two D_2-blocks in the first four rows of A_1' *swapped*. That is, $A_2' = \pi_r\pi_c(A_1')$, where $\pi_r = (1\ 3)(2\ 4)$ and $\pi_c = (1\ 5)(2\ 6)(3\ 7)(4\ 8)$.

The weight 3 blocks in both A_1' and A_2' are in descending order. However, for any permutation π_r' of the last 6 rows of A_1' (i.e. the rows of A_1' with weight ≥ 5), there does not exist a permutation of the columns of $\pi_r'(A_1')$ that results in A_2'. Therefore, the

unique matrices A_1'' and A_2'' returned by the *UniqueOrder* algorithm will not be the same:

$$A_1'' = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \quad A_2'' = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Let $A_1'', A_2'', \ldots, A_n''$ denote the distinct matrices that result from applying the *UniqueOrder* algorithm to the last $k-r$ rows of each of the matrices $\pi_r\pi_c(A')$, where $\pi_r \in P_r$. Our Permutation algorithm produces the unique matrix A'' by simply returning the largest matrix in the list $A_1'', A_2'', \ldots, A_n''$.

Our Permutation algorithm finds *pcnt* by computing the number of permutations of the rows and columns of A' that result in A'' as follows: Let Q_r denote the subset of permutations $\pi_r \in P_r$ in which A'' is the matrix that results when the *UniqueOrder* algorithm is applied to the last $k-r$ rows of $\pi_r\pi_c(A')$. In other words, Q_r is the subset of permutations $\pi_r \in P_r$ for which there exists a permutation π_r' of the last $k-r$ rows of $\pi_r(A')$ such that $\pi_r'\pi_r\pi_c'(A') = A''$, where π_c' is the permutation of the columns of A' that orders the columns of $\pi_r'\pi_r(A')$ in descending order. These are the only permutations of the rows and columns of A' that result in A''. Furthermore, for any pair $A_1' = \pi_{r_1}\pi_{c_1}(A')$ and $A_2' = \pi_{r_2}\pi_{c_2}(A')$, where $\pi_{r_1}, \pi_{r_2} \in Q_2$, the number of permutations of the last $k-r$ rows and the columns of A_1' that result in A'' is equal to the number of permutations of the last $k-r$ rows and the columns of A_2' that result in A''. Therefore, if $pcnt_{k-r}$ is the

number of permutations of the last $k-r$ rows and the columns of $\pi_r\pi_c(A')$ that result in A'', where π_r is any element of Q_r, then $pcnt = |Q_r| \times pcnt_{k-r}$. Note that, in addition to producing the matrix A'', the *UniqueOrder* algorithm also computes the number of permutations of the last $k-r$ rows and columns of $\pi_r\pi_c(A')$ that result in A''. That is, the *UniqueOrder* algorithm also gives us $pcnt_{k-r}$. Therefore, our Permutation algorithm can compute *pcnt* by simply adding $pcnt_{k-r}$ to *pcnt* every time the algorithm produces a matrix $\pi_r\pi_c(A')$ that is uniquely ordered to A'' by the *UniqueOrder* algorithm.

One last thing we will consider before giving a formal description of our Permutation algorithm for the e_3 and d_i-codes is how the algorithm can find the set of permutations P_r. That is, we will show how the algorithm can produce only the permutations π_r of the first r rows of A' that result in a matrix $\pi_r\pi_c(A')$ whose weight 3 blocks are in descending order. We will also show that the algorithm, in fact, only needs to consider a subset of these permutations.

Let $A_{B_1}, A_{B_2}, \ldots, A_{B_m}$ denote the weight 3 blocks in A'. Recall that if the weight 3 blocks of A' are in descending order then:

$$A' = \left[\begin{array}{cccc|c} A_{B_1} & 0 & \cdots & 0 & \\ 0 & A_{B_2} & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \\ 0 & 0 & \cdots & A_{B_m} & \\ \hline & & M & & \end{array}\right]$$

where A_{B_j} either has the same precedence or has a higher precedence than $A_{B_{j+1}}$ (i.e. all E_3-blocks occur first, followed by the D_i-blocks, followed by the D_{i-1}-blocks, and so on). Let π_r be any permutation of the weight 3 rows of A'. Then the weight 3 blocks in $\pi_r\pi_c(A')$ are in descending order if and only if π_r can be written as the product of two permutations $\pi_{r_2}\pi_{r_1}$, where π_{r_1} is a product of permutations of the form $\pi(A_{B_j})$ and π_{r_2} is a product of permutations of the form $\pi(A_{B_{j_1}}, A_{B_{j_2}})$, where:

- $\pi_r(A_{B_j})$ is any permutation of the rows of the weight 3 block A_{B_j} (and just the rows of A_{B_j}), and,

- $\pi_r(A_{B_{j_1}}, A_{B_{j_2}})$ is a permutation that swaps all the rows of the weight 3 block $G_{B_{j_1}}$ with all the rows of the weight 3 block $G_{B_{j_2}}$, where $G_{B_{j_1}}$ and $G_{B_{j_2}}$ are the same type of blocks (i.e. $G_{B_{j_1}}$ and $G_{B_{j_2}}$ are either both E_3-blocks, or, both D_i-blocks).

Thus, to produce only the permutations $\pi_r \in P_r$, our Permutation algorithm only needs to produce all the different combinations of the two permutations $\pi_{r_2}\pi_{r_1}$, where π_{r_1} is a product of permutations of the form $\pi_r(A_{B_j})$ and π_{r_2} is a product of permutations of the form $\pi_r(A_{B_{j_1}}, A_{B_{j_2}})$.

Let us consider all the different possibilities for the permutation π_{r_1}. For each of the different types of weight 3 blocks, let $f_p(A_{B_j})$ denote the number of different permutations of the form $\pi_r(A_{B_j})$. That is, if A_{B_j} consists of i weight 3 rows, then $f_p(A_{B_j})$ is equal to the number of permutations of the i rows of A_{B_j}, which is equal to $i!$. This gives us a total of $\prod_{j=1}^{m} f_p(A_{B_j})$ permutations π_{r_1}. As we shall see, our Permutation algorithm does not have to consider any of these permutations. The reason for this, as will be shown next, is that for each of the permutations $\pi_r = \pi_r(A_{B_j})$, there exists a permutation π_c of the columns of A' such that $\pi_r\pi_c(A') = A'$. In other words, for any permutation π_{r_2} (where π_{r_2} is a product of permutations of the form $\pi_r(A_{B_{j_1}}, A_{B_{j_2}})$), the unique matrix returned by the *UniqueOrder* algorithm is the same for each of the $\prod_{j=1}^{m} f_p(A_{B_j})$ matrices $\pi_{r_2}\pi_{r_1}\pi_c(A')$ (where π_c is a permutation of columns that orders the first r components of the columns of $\pi_{r_2}\pi_{r_1}$ in descending order).

Let us now show that for each of the permutations $\pi_r = \pi_r(A_{B_j})$, where A_{B_j} is either an E_3-block or a D_i-block, $i \geq 1$, there exists a permutation π_c of the columns of A' such that $\pi_r\pi_c(A') = A'$.

We will first consider permutations of the rows of the E_3-block. Let A' be any matrix that contains an E_3-block. Without loss of generality, assume the E_3-block is located in

the first three rows and the first four columns of A'. Then:

$$A' = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 0^* \\ 1 & 1 & 0 & 1 & 0^* \\ 1 & 0 & 1 & 1 & 0^* \\ \hline 0 & 1 & 1 & 1 & X \\ 0 & 0 & 0 & 0 & Y \end{array}\right] \end{array} \begin{matrix} \\ r_1 \\ r_2 \\ r_3 \\ \\ \end{matrix}$$

To the left of the vertical line are the columns of the E_3-block. The rows below the horizontal line represent all the ways the remaining rows of A' may intersect the rows of an E_3-block.

Let π_r be any permutation of the rows of the E_3-block. Then there exists one and only one permutation π_c of the first four columns of A' that orders the columns of the E_3-block in $\pi_r(A')$ in descending order. For example, if $\pi_r = (1\ 3)$ then $\pi_c = (2\ 4)$. Furthermore, for any of the permutations π_r of the rows of the E_3-block, since all three rows of the E_3-block are one in the first column of A', the permutation π_c will be a permutation of columns 2, 3, and 4 of A'. Since each of the remaining rows of A' have the same values in columns 2, 3, and 4 (i.e. either 0 0 0 or 1 1 1), this implies that $\pi_r\pi_c(A') = A'$. Thus, for each of the $f_p(E_3) = 3!$ permutations $\pi_r = \pi_r(A_{B_j})$, where A_{B_j} is an E_3-block, there exists a permutation π_c of the columns of A' such that $\pi_r\pi_c(A') = A'$.

Let us now consider permutations of the rows of the D_i-blocks, $i \geq 1$. Let A' be any matrix that contains a D_i-block. Without loss of generality, assume the D_i-block is

located in the first i rows and the first $i+2$ columns of A'. Then:

$$A' = \begin{array}{c} \begin{array}{cccccc|c} c_1 & c_2 & c_3 & c_4 & & c_{i+2} & \end{array} \\ \left[\begin{array}{cccccc|c} 1 & 1 & 1 & 0 & \cdots & 0 & 0^* \\ 1 & 1 & 0 & 1 & \cdots & 0 & 0^* \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & 1 & 0 & 0 & \cdots & 1 & 0^* \\ \hline 1 & 1 & 0 & 0 & \cdots & 0 & W \\ 1 & 0 & 1 & 1 & \cdots & 1 & X \\ 0 & 1 & 1 & 1 & \cdots & 1 & Y \\ 0 & 0 & 0 & 0 & \cdots & 0 & Z \end{array}\right] \end{array} \begin{array}{l} r_1 \\ r_2 \\ \vdots \\ r_i \end{array}$$

To the left of the vertical line are the columns of the D_i-block. The rows below the horizontal line represent all the ways the remaining rows of A' may intersect the rows of a D_i-block.

Let π_r be any permutation of the rows of the D_i-block. Then there exists one and only one permutation π_c of the last i columns of the D_i-block that orders the columns of the D_i-block in $\pi_r(A')$ in descending order. For example, if $\pi_r = (1 \;\; i)$ then $\pi_c = (3 \;\; i+2)$. Furthermore, since each of the remaining rows of A' have the same values in columns $3, 4, \ldots, i+2$ of A' (i.e. the last i columns of the D_i-block in A'), $\pi_r\pi_c(A') = A'$. Thus, for each of the $f_p(D_i) = i!$ permutations $\pi_r = \pi_r(A_{B_j})$, where A_{B_j} is a D_i-block, $i \geq 1$, there exists a permutation π_c of the columns of A' such that $\pi_r\pi_c(A') = A'$.

Therefore, for any permutation π_{r_2}, where π_{r_2} is a product of permutations of the form $\pi_r(A_{B_{j_1}}, A_{B_{j_2}})$ (where $A_{B_{j_1}}$ and $A_{B_{j_2}}$ are the same types of weight 3 blocks), the unique matrix returned by the *UniqueOrder* algorithm is the same for each of the $\prod_{j=1}^{m} f_p(A_{B_j})$ matrices $\pi_{r_2}\pi_{r_1}\pi_c(A')$, where π_{r_1} is a product of permutations of the form $\pi_r(A_{B_j})$ (and π_c is a permutation of columns that orders the first r components of the columns of $\pi_{r_2}\pi_{r_1}(A')$ in descending order).

Let us now summarize what this means in terms of our Permutation algorithm: Let π_{r_2} be any permutation that is a product of permutations of the form $\pi_r(A_{B_{j_1}}, A_{B_{j_2}})$. Let π_{c_2} be any permutation of the columns of A' that orders the weight 3 blocks of $\pi_{r_2}(A')$ in descending order. Let $A'_2 = \pi_{r_2}\pi_{c_2}(A')$. Let A'' be the unique matrix that results from applying the *UniqueOrder* algorithm to the last $k-r$ rows and the columns of A'_2. Let $pcnt_{k-r}$ be the number of permutations $\pi'_r\pi'_c$ of the last $k-r$ rows and the columns of A'_2 such that $\pi'_r\pi'_c(A'_2) = A''$. Then for our distinct permutation π_{r_2}, there are exactly $pcnt_{k-r}(\prod_{j=1}^{m} f_p(A_{B_j}))$ permutations $\pi_r\pi_c$ of the columns of A' in which $\pi_r\pi_c(A') = A''$, such that $\pi_r = \pi'_r\pi_{r_2}\pi_{r_1}$, where π_{r_1} is the product of permutations of the form $\pi_r(A_{B_j})$. π'_r is a permutation of the last $k-r$ rows of A', and, $f_p(A_{B_j})$ is defined as follows:

$$\begin{aligned} f_p(E_3) &= 3! \\ f_p(D_i) &= i!, \quad i \geq 1. \end{aligned}$$

We are now ready to give a formal description of our Permutation algorithm for the e_3 and d_i-codes.

Algorithm 6.50 *PermutationAlgorithm*$(L', A_0, acnt)$:

- Input: the list L' of pairs (A', cnt') produced by Algorithm 6.48 (i.e. our Combination algorithm for the e_3 and d_i-codes).
- Output: the $k \times k$ matrix A_0 and $acnt$. $G_0 = [I_k|A_0]$ is the largest matrix in $\mathcal{G}_{[I|A]}(C)$ such that:
  - G_0 has a maximal combination of generator blocks with weight 4 rows (defined in Section 6.7),
  - the weight 3 blocks in A_0 are ordered in descending order,
  - the columns of A_0 are in descending order, and
  - the last $k-r$ rows of A_0 are in descending order, where r is the number of weight 3 rows in A_0.

The integer *acnt* is the size of the automorphism group of C.

begin

Initially A_0 is *undefined.*

for each $(A', cnt') \in L'$ **do**

Order the weight 3 blocks of A' in descending order.

Let r denote the number of weight 3 rows in A'.

Let $A_{B_1}, A_{B_2}, \ldots, A_{B_m}$ denote the weight 3 blocks in A'.

Let $1 = p_1 < p_2 < p_3 < \ldots < p_q < p_{q+1} = m + 1$, be the sequence of weight 3 block indices in which $A_{B_{p_i}}$, $A_{B_{p_{i+1}-1}}$ are the same type and $A_{B_{p_i}}$, $A_{B_{p_{i+1}}}$ are not, for $1 \leq i \leq q$.

Let π_{r_2} denote any combination of the permutations $\pi_r(A_{B_{j_1}}, A_{B_{j_2}})$, where $A_{B_{j_1}}$ and $A_{B_{j_2}}$ are the same type of weight 3 blocks.

for each of the $\prod_{i=1}^{q}(p_{i+1} - p_i)!$ permutations π_{r_2} **do**

Let π_{c_2} be any permutation of columns that orders the columns of the weight 3 blocks of $\pi_{r_2}(A')$ in descending order.

Let $A'_2 = \pi_{r_2}\pi_{c_2}(A')$.

UniqueOrderMatrix$(A'_2, r, A'', pcnt_{k-r})$

if (A_0 is *undefined*) **or** $(A'' > A_0)$ **then**

$A_0 = A''$.

$pcnt = pcnt_{k-r} * \prod_{j=1}^{m} f_p(G_{B_j})$.

$acnt = cnt' * pcnt$.

end if

else if $(A'' = A_0)$ **then**

$pcnt = pcnt_{k-r} * \prod_{j=1}^{m} f_p(G_{B_j})$.

$acnt = acnt + cnt' * pcnt$.

end for

end for

end

We now have our Permutation algorithm for the e_3 and d_i-codes, which together with the Combination algorithm of Section 6.7, gives us our Unique Representative algorithm for the e_3 and d_i-codes.

6.9 A Unique Representative Algorithm for the Distance ≥ 6 Codes

In the last two sections, we developed the Combination and Permutation algorithms for our Unique Representative algorithm for the $(2k, k, 4)$ self-dual codes. The Combination algorithm was designed to reduce both the number of sets of permutations $\Pi(T_I, T_A)$ the algorithm needs to consider and the number of matrices produced by the algorithm (compared to the Combination algorithm we developed in Section 6.4). The Permutation algorithm was designed to reduce the number of permutations of the rows and columns of the matrices produced by the Combination algorithm that the algorithm needs to perform (compared to the Permutation algorithm we developed in Section 6.5). If the number of weight 4 words in the input code is large enough then both algorithms will substantially reduce the amount of work that is performed. However, if the input code contains only a few weight 4 words, the algorithms offer little improvement over our original Combination and Permutation algorithms. Furthermore, if the input code has distance $d \geq 6$, the algorithms offer no improvement.

In this section, we will develop Combination and Permutation algorithms for the $(2k, k, d \geq 6)$ self-dual codes. The Combination algorithm will be designed to reduce both the number of sets of permutations the algorithm needs to consider and the number of matrices produced by the algorithm. The Permutation algorithm will be designed to reduce the number of permutations of the rows and columns of the matrices produced by

the Combination algorithm that the algorithm needs to perform. Once we have designed our Combination and Permutation algorithms for the $(2k, k, d \geq 6)$ self-dual codes, we will then show how we can combine them with our Combination and Permutation algorithms for the $(2k, k, 4)$ self-dual codes in order to reduce the amount of work that needs to be done when the input codes contains only a few weight 4 words.

6.9.1 The $f_{d-1,i}$-Blocks

Let $G = [I_k|A]$ be the generator matrix for a $(2k, k, d \geq 6)$ self-dual code C. Our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes will be based on the weight $d-1$ rows in A. Unlike the weight 3 rows of the $(2k, k, 4)$ self-dual codes, the number of different ways in which the weight $d-1$ rows may interact in such an A is quite large. Therefore, our Combination and Permutation algorithms for the $(2k, k, d \geq 6)$ self-dual codes will only consider a subset of the different blocks of weight $d-1$ rows that may occur in A. The only blocks of weight $d-1$ rows of A that our algorithms will consider are blocks whose rows and columns are permutations of the following:

$$f_{d-1,i} = \left.\begin{array}{cccccccccccccccc} 1 & 1 & \cdots & 1 & 1 & \cdots & 1 & 0 & 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0^* \\ 1 & 1 & \cdots & 1 & 1 & \cdots & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & 0 & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 0 & \cdots & 1 & 1 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 0^* \end{array}\right\} i$$

$$\underbrace{}_{d-2-i} \quad \underbrace{}_{i} \quad \underbrace{}_{2i}$$

We will refer to any permutation of the rows and columns of such blocks of weight $d-1$ rows as $f_{d-1,i}$-blocks. When we are not concerned with the number of rows in the block, we will refer to each of the different $f_{d-1,i}$-blocks as simply f_{d-1}-blocks. Note that if A contains an $f_{d-1,i}$-block, then the i codewords in C that correspond to the i rows of the $f_{d-1,i}$-block in A form an $F_{d,i}$-block (which was introduced in Section 5.4 of Chapter 5). Also note that if $d = 6$ or $d = 8$ then $1 \leq i \leq d-2$, and if $d \geq 10$, since for any pair of rows in an $f_{d-1,i}$-block, the corresponding rows in G sum to a weight 8 word, we must

have $i = 1$.

Now, unlike the weight 3 blocks, the rows of any two f_{d-1}-blocks may intersect (i.e. there may exist a column in A that has a value of one in rows of both blocks). Furthermore, a row of one f_{d-1}-block may also be a row of another f_{d-1}-block.

Example 6.51 Let C be the $(26, 13, 6)$ self-dual code generated by $G = [I_{13}|A]$, where:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} & c_{13} \end{matrix} \\ \left[\begin{array}{cccccccccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \\ r_{11} \\ r_{12} \\ r_{13} \end{matrix} \end{array}$$

The matrix A contains many pairs of f_5-blocks that intersect. For example, the $f_{5,4}$-block in rows r_1, r_2, r_3, r_4 intersects the $f_{5,3}$-block in rows r_5, r_6, r_7. Also, each of the weight 5 rows in A belong to more than one f_5-block. For example, the first row of A is a part of both the $f_{5,4}$-block in rows r_1, r_2, r_3, r_4 and the $f_{5,3}$-block in rows r_1, r_5, r_6.

We will limit the f_{d-1}-blocks that our algorithms consider to the combinations of f_{d-1}-blocks that form what we will call *non-intersecting sets of f_{d-1}-blocks*:

Definition 6.52 Let $G = [I_k|A]$ be a generator for a $(2k, k, d \geq 6)$ self-dual code C, such that A contains at least one weight $d-1$ row (i.e. A contains at least one f_{d-1}-block). Let $F_1, F_2, \ldots, F_m$ denote any m disjoint sets of weight $d-1$ rows in A, where $m \geq 1$. Then $S = \{F_1, F_2, \ldots, F_m\}$ is a *non-intersecting set of f_{d-1}-blocks* if and only if:

- each $F_j \in S$ forms an $f_{d-1,i}$-block, where $i = |F_j|$,
- if $r_1 \in F_{j_1}$ and $r_2 \in F_{j_2}$, where $j_1 \neq j_2$, then rows r_1 and r_2 do not intersect,
- there does not exist a weight $d-1$ row r in A in which r does not intersect any of the m disjoint sets in S and r is not an element of any of the sets in S (i.e. S contains as many non-intersecting f_{d-1}-blocks as possible), and
- there does not exist a weight $d-1$ row r in A that intersects rows of one and only one set F_j in which $F_j \cup \{r\}$ forms an $f_{d-1,i+1}$-block, where $i = |F_j|$ (i.e. each non-intersecting f_{d-1}-block in S is as large as possible).

Example 6.53 Let C be the $(26, 13, 6)$ self-dual code generated by $G = [I_{13}|A]$, where:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} & c_{13} \end{matrix} \\ \left[\begin{array}{cccccccccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \\ r_{11} \\ r_{12} \\ r_{13} \end{matrix} \end{array}$$

Let $F_1 = \{r_1\}$ and let $F_2 = \{r_{11}, r_{12}\}$. F_1 is an $f_{5,1}$-block and F_2 is a $f_{5,2}$-block. The set $S = \{F_1, F_2\}$ is a non-intersecting set of f_5-blocks. Let $F_1' = \{r_2\}$ and let $F_2' = \{r_9, r_{10}\}$. F_1' is an $f_{5,1}$-block and F_2' is a $f_{5,2}$-block. The set $S' = \{F_1', F_2'\}$ is also a non-intersecting set of f_5-blocks. Note that S and S' contain the same number of each of the different f_5-blocks (i.e. both contain one $f_{5,1}$-block, one $f_{5,2}$-block, zero $f_{5,3}$-blocks, and zero $f_{5,4}$-blocks). Let $F_1'' = \{r_1, r_2, r_3, r_4\}$, which is an $f_{5,4}$-block. The set $S'' = \{F_1''\}$ is a non-intersecting set of f_5-blocks. Note that S and S'' do not contain the same number of each of the different f_5-blocks.

As Example 6.53 demonstrates, A may contain many non-intersecting sets of f_{d-1}-blocks. Furthermore, if S and S' are two different non-intersecting sets of f_{d-1}-blocks in A, then the number of each of the different f_{d-1}-blocks in S and S' may or may not be the same. In our algorithms, for a given A, we will only be interested in non-intersecting sets

of f_{d-1}-blocks that contain a specific number of each of the different f_{d-1}-blocks. Now, in order to define how many of each of the different f_{d-1}-blocks that our non-intersecting sets of f_{d-1}-blocks will have, we will first define a comparison relation between two non-intersecting sets of f_{d-1}-blocks:

Definition 6.54 Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, d \geq 6)$ self-dual code, where A contains at least two non-intersecting sets of f_{d-1}-blocks. Let S_1 and S_2 be any two non-intersecting sets of f_{d-1}-blocks in A. If S and S' contain the same number of each of the different f_{d-1}-blocks then we will say S_1 and S_2 have an *equal combination of f_{d-1}-blocks*. Suppose S_1 and S_2 do not have an equal combination of f_{d-1}-blocks. Let i be the largest integer in which S_1 and S_2 do not contain the same number of $f_{d-1,i}$-blocks. Then if S_1 has more $f_{d-1,i}$-blocks than S_2 then S_1 is said to have a *larger combination of f_{d-1}-blocks* than S_2, otherwise S_1 is said to have a *smaller combination of f_{d-1}-blocks* than S_2.

For a given A, the only non-intersecting sets of f_{d-1}-blocks our algorithms will consider are the sets that have either equal or larger combinations of f_{d-1}-blocks than all other f_{d-1}-blocks in A. We will refer to such sets as *maximal non-intersecting sets of f_{d-1}-blocks*. More formally, we have the following definition:

Definition 6.55 Let $G = [I_k|A]$ generate a $(2k, k, d \geq 6)$ self-dual code C, where A contains at least one weight $d-1$ row. Let S be a non-intersecting set of f_{d-1}-blocks in A. Then S is a *maximal non-intersecting set of f_{d-1}-blocks* if and only if for any other non-intersecting set of f_{d-1}-blocks S' in A, either S and S' have equal combinations of f_{d-1}-blocks or S has a larger combination of f_{d-1}-blocks than S'.

Example 6.56 Let C be the $(26, 13, 6)$ self-dual code generated by $G = [I_{13}|A]$, where:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} & c_{13} \end{matrix} \\ \left[\begin{array}{cccccccccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \\ r_{11} \\ r_{12} \\ r_{13} \end{matrix}$$

Let $S_1 = \{\{r_1, r_2, r_3, r_4\}\}$, let $S_2 = \{\{r_1\}, \{r_{11}, r_{12}\}\}$, and let $S_3 = \{\{r_1, r_5, r_6\}\}$. S_1, S_2, and S_3 are all non-intersecting sets of f_5-blocks. Furthermore, every non-intersecting set of f_5-blocks in A has an equal combination of f_5-blocks to one and only one of S_1, S_2, and S_3. Now, S_1 consists of an $f_{5,4}$-block, S_2 consists of an $f_{5,1}$-block and an $f_{5,2}$-block, and S_3 consists of an $f_{5,3}$-block. Therefore, S_1 has a larger combination of f_5-blocks than both S_2 and S_3. Thus, S_1 is a maximal non-intersecting set of f_5-blocks. Furthermore, if S is any non-intersecting set of f_5-blocks in A such that S and S_1 have an equal combination of f_5-blocks then S is also a maximal non-intersecting set of f_5-blocks.

Let C be a $(2k, k, d \geq 6)$ self-dual code and let $\mathcal{G}_{[I|A]}(C)$ denote the set of all generator matrices of the form $G = [I_k|A]$ that generate a code equivalent to C. We have just defined, for a given $k \times k$ matrix A, where A contains at least one weight $d-1$ row and $G = [I_k|A] \in \mathcal{G}_{[I|A]}(C)$, the combination of f_{d-1}-blocks that we will consider. That is, for

a given A, we will only consider the combinations of the weight $d-1$ rows of A that form a maximal non-intersecting set of f_{d-1}-blocks. Now, let $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ be any two matrices in $\mathcal{G}_{[I|A]}(C)$, where A_1 and A_2 both contain at least one weight $d-1$ row. Let S_1 and S_2 by any two maximal non-intersecting sets of f_{d-1}-blocks in A_1 and A_2, respectively. Then S_1 and S_2 do not necessarily contain equal combinations of f_{d-1}-blocks. In our algorithms, we will only be interested in the subset of the matrices in $\mathcal{G}_{[I|A]}(C)$ whose maximal non-intersecting sets of f_{d-1}-blocks have either equal or larger combinations of f_{d-1}-blocks than the maximal non-intersecting sets of f_{d-1}-blocks in all other matrices in $\mathcal{G}_{[I|A]}(C)$. We will refer to the maximal non-intersecting sets of f_{d-1}-blocks in such a matrix as *maximal combinations of non-intersecting f_{d-1}-blocks*. More formally, we have the following definition:

Definition 6.57 Let $G = [I_k|A]$ be a $(2k, k, d \geq 6)$ self-dual code C, such that A contains at least one weight $d-1$ row. Let S be a maximal non-intersecting set of f_{d-1}-blocks in A. Then S is a *maximal combination of non-intersecting f_{d-1}-blocks* if and only if for any $G' = [I_k|A'] \in \mathcal{G}_{[I|A](C)}$, where A' contains at least one weight $d-1$ row, if S' is a maximal non-intersecting set of f_{d-1}-blocks in A' then either S and S' have equal combinations of f_{d-1}-blocks or S has a larger combination of f_{d-1}-blocks than S'.

If the maximal non-intersecting sets of f_{d-1}-blocks in A are also maximal combinations of non-intersecting f_{d-1}-blocks then we will say A contains a maximal combination of non-intersecting f_{d-1}-blocks.

Let C be a $(2k, k, d \geq 6)$ self-dual code. We will use the notation $\mathcal{G}_{max}(C)$ to denote the set of all matrices $G = [I_k|A] \in \mathcal{G}_{[I|A]}(C)$ in which A contains a maximal combination of non-intersecting f_{d-1}-blocks. Note that if S_1 and S_2 are maximal combinations of non-intersecting f_{d-1}-blocks in the matrices A_1 and A_2, respectively, where $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ are both elements of $\mathcal{G}_{max}(C)$, then the total number of weight $d-1$ rows in the f_{d-1}-blocks of S_1 is equal to the total number of weight $d-1$ rows in the f_{d-1}-blocks of S_2. For a given $(2k, k, d \geq 6)$ self-dual code C, we will use the notation $h(C)$ to denote

the total number of weight $d-1$ rows in the f_{d-1}-blocks of any maximal combination of non-intersecting f_{d-1}-blocks in any matrix A, where $G = [I_k|A] \in \mathcal{G}_{max}(C)$.

Now, by Theorem 2.8 of Chapter 2, if $G = [I_k|A]$ generates a $(2k, k, d \geq 6)$ self-dual code C then there exists at least one $G' = [I_k|A'] \in \mathcal{G}_{[I|A]}(C)$ such that A' contains at least one weight $d-1$ row. In other words, there exists an A' that contains a non-intersecting set of f_{d-1}-blocks, and thus there exists an A' that contains a maximal combination of non-intersecting f_{d-1}-blocks. Therefore, $\mathcal{G}_{max}(C) \neq \emptyset$. Thus, we can select the unique matrix $G_0 = [I_k|A_0]$, produced by our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes, from the set of generator matrices $\mathcal{G}_{max}(C)$.

6.9.2 $f_{d-1,i}$-Blocks and the Unique Representative Algorithm

Let C be a $(2k, k, d \geq 6)$ self-dual code generated by $G = [I_k|A]$. The unique matrix that our Unique Representative algorithm will return is the largest matrix $G_0 = [I_k|A_0] \in \mathcal{G}_{max}(C)$ such that:

- the first $h(C)$ rows of A_0 form a maximal combination of non-intersecting f_{d-1}-blocks,
- the first $h(C)$ rows of A_0 are in descending order,
- the last $k - h(C)$ rows of A_0 are in descending order, and
- the columns of A_0 are in descending order.

In this subsection, we will develop general outlines of the Combination and Permutation algorithms for our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes.

We will begin by defining several sets of permutations and generator matrices that we will need for our Unique Representative algorithm. We will also give some of the properties of these sets that we will use in our algorithm.

Let $\mathcal{G}^*_{max}(C)$ denote the set of all generator matrices $G' = [I_k|A'] \in \mathcal{G}_{max}(G)$ in which the first $h(C)$ rows of A' contain a maximal combination of non-intersecting f_{d-1}-blocks. Let $\Pi^*_{max}(C)$ denote the set of all permutations π of the columns of G such that $\mathcal{RREF}(\pi(G)) \in \mathcal{G}^*_{max}(C)$. Note that since $\mathcal{G}^*_{max}(C)$ contains the unique matrix $G_0 = [I_k|A_0]$, our Unique Representative algorithm need only perform permutations that are in the set $\Pi^*_{max}(C)$.

Let $T(G)$ denote the set of all pairs (T_I, T_A), where T_I and T_A are equally sized subsets of the columns of I_k and A, respectively. Recall that for each $(T_I, T_A) \in T(G)$:

- $\Pi(T_I, T_A)$ is the set of all permutations π of the columns of G such that the first k columns of $\pi(G)$ are a permutation of the set of columns $(S_I - T_I) \cup T_A$, where S_I is the set of all k columns of I_k.
- $\mathcal{G}(T_I, T_A)$ is the set of all generator matrices $\mathcal{RREF}(\pi(G))$ such that $\pi \in \Pi(T_I, T_A)$.

Let $T^*_{max}(G)$ denote the set of all pairs $(T_I, T_A) \in T(G)$ for which there exists at least one permutation $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) \in \mathcal{G}^*_{max}(C)$. For each $(T_I, T_A) \in T^*_{max}(G)$:

- $\Pi^*_{max}(T_I, T_A)$ will denote the set of all permutations $\pi \in \Pi(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) \in \mathcal{G}^*_{max}(C)$.
- $\mathcal{G}^*_{max}(T_I, T_A)$ will denote the set of all generator matrices $\mathcal{RREF}(\pi(G))$ such that $\pi \in \Pi^*_{max}(T_I, T_A)$.

In terms of any $k \times k$ matrix A', where $[I_k|A'] \in \mathcal{G}(T_I, T_A)$ and $(T_I, T_A) \in T^*_{max}(G)$, the set of generator matrices $\mathcal{G}^*_{max}(T_I, T_A)$ is equal to the set of all matrices $[I_k|\pi_r\pi_c(A')]$, where $\pi_r\pi_c$ is any permutation of the rows and columns of A', such that π_r permutes any one of the maximal combination of non-intersecting f_{d-1}-blocks in A' to the first $h(C)$ rows of $\pi_r(A')$.

The relationships between the sets of permutations $\Pi^*_{max}(G)$ and $\Pi^*_{max}(T_I, T_A)$, and the sets of generator matrices $\mathcal{G}^*_{max}(C)$ and $\mathcal{G}^*_{max}(T_I, T_A)$ is:

$$\begin{aligned} \Pi^*_{max}(G) &= \bigcup_{(T_I,T_A)\in T^*_{max}(G)} \Pi^*_{max}(T_I, T_A) \\ \mathcal{G}^*_{max}(G) &= \bigcup_{(T_I,T_A)\in T^*_{max}(G)} \mathcal{G}^*_{max}(T_I, T_A) \end{aligned}$$

That is, the set of permutations $\Pi^*_{max}(G)$ is equal to the union of the disjoint sets of permutations $\Pi^*_{max}(T_I, T_A)$, where $(T_I, T_A) \in T^*_{max}(G)$, and the set of generator matrices $\mathcal{G}^*_{max}(C)$ is equal to the union of the sets of generator matrices $\mathcal{G}^*_{max}(T_I, T_A)$, where $(T_I, T_A) \in T^*_{max}(G)$.

Let π be any permutation in $\Pi^*_{max}(G)$. Let C' be the $(2k, k, d \geq 6)$ self-dual code generated by $G' = [I_k|A']$, where $G' = \mathcal{RREF}(\pi(G))$. Let S' be the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A'. Then each of the $h(C)$ weight $d-1$ rows of S' corresponds to a unique weight d word in C. That is, for each weight $d-1$ row r of A' that is in the maximal combination of non-intersecting f_{d-1}-blocks S', there exists a unique weight d word $\vec{w}$ in C that is, essentially, permuted by the permutation π to the unique weight d word $\vec{w'}$ of C' that is in row r of $G' = [I_k|A']$. Therefore, C contains a unique set V of $h(C)$ weight d words that corresponds to the $h(C)$ weight $d-1$ rows of the maximal combination of non-intersecting f_{d-1}-blocks S'.

Example 6.58 Let C be the $(22, 11, 6)$ self-dual code generated by $G = [I_k|A]$, where:

$$A = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} \end{matrix} \\ \left[\begin{array}{ccccccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{array}{ll} r_1 & (a_1) \\ r_2 & (a_2) \\ r_3 & (a_3) \\ r_4 & (a_4) \\ r_5 & (a_5) \\ r_6 & (a_6) \\ r_7 & (a_7) \\ r_8 & (a_8) \\ r_9 & (a_9) \\ r_{10} & (a_{10}) \\ r_{11} & (a_{11}) \end{array}$$

Let a_1 denote the first column of I_k in G and let c_1 denote the first column of A in G. Let $\pi = (a_1\ c_1)$. Let C' be the $(22, 11, 6)$ self-dual code generated $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$. Then:

$$A' = \begin{array}{c} \begin{matrix} a_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} \end{matrix} \\ \left[\begin{array}{ccccccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{array}{ll} r_1 & (c_1) \\ r_2 \oplus r_1 & (a_2) \\ r_3 \oplus r_1 & (a_3) \\ r_4 \oplus r_1 & (a_4) \\ r_5 \oplus r_1 & (a_5) \\ r_6 & (a_6) \\ r_7 & (a_7) \\ r_8 & (a_8) \\ r_9 & (a_9) \\ r_{10} & (a_{10}) \\ r_{11} & (a_{11}) \end{array}$$

The first three rows of A' form a maximal combination of non-intersecting f_5-blocks S' (that, in this case, consists of a single $f_{5,3}$-block). The weight 6 word of C' in the first row of G' corresponds to the weight 6 word $\vec{r_1}$ of C. The weight 6 word of C' in the second row of G' corresponds to the weight 6 word $\vec{r_1} \oplus \vec{r_2}$ of C. The weight 6 word of C' in the third row of G' corresponds to the weight 6 word $\vec{r_1} \oplus \vec{r_3}$ of C. The set $V = \{r_1, r_1 \oplus r_2, r_1 \oplus r_3\}$ is the unique set of $h(C) = 3$ weight d words in C that corresponds to the weight 5 rows of A' that form the maximal combination of non-intersecting f_5-blocks S'.

Given S' and π, we will refer to V as the unique set of weight d words in C that *corresponds* to S'. Given V and π, we will refer to S' as the unique maximal combination of non-intersecting f_{d-1}-blocks in A' that *corresponds* to V.

Let us now examine the structure of the set of m weight d words V. Recall that for each $f_{d-1,i}$-block in S', the corresponding set of i codewords in C' forms an $F_{d,i}$ (which was introduced in Section 5.4 of Chapter 5). Let $f_{d-1,i_1}, f_{d-1,i_2}, \ldots, f_{d-1,i_m}$ denote the f_{d-1}-blocks in S'. Then, since each f_{d-1,i_j}-block in S' corresponds to an F_{d,i_j}-block in C', each f_{d-1,i_j}-block in S' corresponds to an F_{d,i_j}-block in C. In other words, the $h(C)$ weight d words in V form m F_d-blocks: $F_{d,i_1}, F_{d,i_2}, \ldots, F_{d,i_m}$. Furthermore, since the f_{d-1}-blocks in S' do not intersect, the F_d-blocks in V do not intersect. Thus, the unique set of $h(C)$ weight d words V in C that corresponds to S' consists of m non-intersecting F_{d-1}-blocks: $F_{d,i_1}, F_{d,i_2}, \ldots, F_{d,i_m}$.

Now, given any set V of m non-intersecting F_d-blocks, $F_{d,i_1}, F_{d,i_2}, \ldots, F_{d,i_m}$, in C, there may or may not exist a permutation $\pi \in \Pi^*_{max}(C)$ that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ in which the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A' corresponds to V. If for a given set V of m non-intersecting F_d-blocks, $F_{d,i_1}, F_{d,i_2}, \ldots, F_{d,i_m}$, in C, there does exist a permutation $\pi \in \Pi^*_{max}(C)$ that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ in which the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A' corresponds to V, we will refer to V as a ***maximal combination of non-intersecting F_d-blocks*** in C.

Let $\mathcal{V}_{max}(C)$ denote the set of all sets of $h(C)$ weight d words V in C that form a maximal combination of non-intersecting F_d-blocks. For each $V \in \mathcal{V}_{max}(C)$:

- $\Pi^*_V(G)$ will denote the set of all permutations $\pi \in \Pi^*_{max}(G)$ that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ in which the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A' corresponds to V.
- $\mathcal{G}^*_V(C)$ will denote the set of all generator matrices $\mathcal{RREF}(\pi(G))$ such that $\pi \in \Pi^*_V(G)$.
- $T^*_V(G)$ will denote the set of all pairs $(T_I, T_A) \in T^*_{max}(G)$ for which there exists at least one permutation $\pi \in \Pi^*_{max}(T_I, T_A)$ such that $\pi \in \Pi^*_V(G)$.
- For each $(T_I, T_A) \in T^*_V(G)$:
  - $\Pi^*_V(T_I, T_A)$ will denote the set of all permutations $\pi \in \Pi^*_{max}(T_I, T_A)$ such that $\pi \in \Pi^*_V(G)$.
  - $\mathcal{G}^*_V(T_I, T_A)$ will denote the set of all generator matrices $\mathcal{RREF}(\pi(G))$ such that $\pi \in \Pi^*_V(T_I, T_A)$.

In terms of any $k \times k$ matrix A', where $G' = [I_k|A'] \in \mathcal{G}^*_V(T_I, T_A)$, $(T_I, T_A) \in T^*_V(G)$, and $V \in \mathcal{V}_{max}(C)$, the set of generator matrices $\mathcal{G}^*_V(T_I, T_A)$ is equal to the set of all generator matrices $[I_k|\pi_{r_1}\pi_{r_2}\pi_c(A')]$, where π_{r_1} is a permutation of the first $h(C)$ rows of A', π_{r_2} is a permutation of the last $k - r$ rows of A', and π_c is a permutation of the k columns of A'. Note that given any $G' = [I_k|A'] \in \mathcal{G}^*_V(T_I, T_A)$, there is a one-to-one correspondence between the permutations in $\Pi^*_V(T_I, T_A)$ and the permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A', where π_{r_1} is a permutation of the first $h(C)$ rows of A', π_{r_2} is a permutation of the last $k - h(C)$ rows of A', and π_c is a permutation of the k columns of A'.

For each $V \in \mathcal{V}_{max}(C)$, the relationships between the sets of permutations $\Pi^*_V(G)$ and $\Pi^*_V(T_I, T_A)$, and the sets of generator matrices $\mathcal{G}^*_V(C)$ and $\mathcal{G}^*_V(T_I, T_A)$ is:

$$\begin{aligned} \Pi^*_V(G) &= \bigcup_{(T_I,T_A)\in T^*_V(G)} \Pi^*_V(T_I, T_A) \\ \mathcal{G}^*_V(C) &= \bigcup_{(T_I,T_A)\in T^*_V(G)} \mathcal{G}^*_V(T_I, T_A) \end{aligned}$$

That is, for each $V \in \mathcal{V}_{max}(C)$, the set of permutations $\Pi^*_V(G)$ is equal to the union of the disjoint sets of permutations $\Pi^*_V(T_I, T_A)$, where $(T_I, T_A) \in T^*_V(G)$, and the set of generator matrices $\mathcal{G}^*_V(C)$ is equal to the union of the sets of generator matrices $\mathcal{G}^*_V(T_I, T_A)$, where $(T_I, T_A) \in T^*_V(G)$.

The relationships between the sets of permutations $\Pi^*_{max}(G)$ and $\Pi^*_V(G)$, and the sets of generator matrices $\mathcal{G}^*_{max}(C)$ and $\mathcal{G}^*_V(C)$ is:

$$\begin{aligned} \Pi^*_{max}(G) &= \bigcup_{V\in\mathcal{V}_{max}(C)} \Pi^*_V(G) \\ \mathcal{G}^*_{max}(C) &= \bigcup_{V\in\mathcal{V}_{max}(C)} \mathcal{G}^*_V(C) \end{aligned}$$

That is, the set of permutations $\Pi^*_{max}(G)$ is equal to the union of the disjoint sets of permutations $\Pi^*_V(G)$, where $V \in \mathcal{V}_{max}(C)$. The set of generator matrices $\mathcal{G}^*_{max}(C)$ is equal to the union of the sets of generator matrices $\mathcal{G}^*_V(T_I, T_A)$, where $V \in \mathcal{V}_{max}(C)$.

Therefore, the set of permutations $\Pi^*_{max}(T_I, T_A)$ is equal to the union of the disjoint sets of permutations $\Pi^*_V(T_I, T_A)$, where $V \in \mathcal{V}_{max}(C)$ and $(T_I, T_A) \in T^*_V(G)$:

$$\Pi^*_{max}(G) = \bigcup_{V\in\mathcal{V}_{max}(C)} \bigcup_{(T_I,T_A)\in T^*_V(G)} \Pi^*_V(T_I, T_A)$$

Our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes will use the disjoint sets $\Pi^*_V(T_I, T_A)$ to perform the permutations in $\Pi^*_{max}(G)$. As the next theorem demonstrates, by performing the permutations in $\Pi^*_{max}(G)$ with the sets $\Pi^*_V(T_I, T_A)$, we may be able to avoid performing many of the permutations in $\Pi^*_{max}(G)$:

Theorem 6.59 For any $V_1, V_2 \in \mathcal{V}_{max}(C)$, $(T_{I_1}, T_{A_1}) \in T^*_{V_1}(G)$, and $(T_{I_2}, T_{A_2}) \in T^*_{V_2}(C)$ either:

- $\mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1}) = \mathcal{G}^*_{V_2}(T_{I_2}, T_{A_2})$, or
- $\mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1}) \cap \mathcal{G}^*_{V_2}(T_{I_2}, T_{A_2}) = \emptyset$.

Furthermore, if $\mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1}) = \mathcal{G}^*_{V_2}(T_{I_2}, T_{A_2})$ then for any $G' \in \mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1})$, the number of permutations $\pi_1 \in \Pi^*_{V_1}(T_{I_1}, T_{A_1})$ such that $\mathcal{RREF}(\pi_1(G)) = G'$ is equal to the number of permutations $\pi_2 \in \Pi^*_{V_2}(T_{I_2}, T_{A_2})$ such that $\mathcal{RREF}(\pi_2(G)) = G'$.

Proof Suppose $\mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1}) \cap \mathcal{G}^*_{V_2}(T_{I_2}, T_{A_2}) \neq \emptyset$. Then $\mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1}) = \mathcal{G}^*_{V_2}(T_{I_2}, T_{A_2})$ since, for any $G' = [I_k|A'] \in \mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1}) \cap \mathcal{G}^*_{V_2}(T_{I_2}, T_{A_2})$, both sets are equal to the set of all matrices $[I_k|\pi_{r_1}\pi_{r_2}\pi_c(A')]$, where π_{r_1} is a permutation of the first $h(C)$ rows of A', π_{r_2} is a permutation of the last $k - h(C)$ rows of A', and π_c is a permutation of the k columns of A'.

Suppose $\mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1}) = \mathcal{G}^*_{V_2}(T_{I_2}, T_{A_2})$. Let π' and π'' be any two permutations in $\Pi^*_{V_1}(T_{I_1}, T_{A_1})$ and $\Pi^*_{V_2}(T_{I_2}, T_{A_2})$, respectively, such that $\mathcal{RREF}(\pi'(G)) = \mathcal{RREF}(\pi''(G))$. Let $G''' = [I_k|A'''] = \mathcal{RREF}(\pi'(G))$. Then there is a one-to-one correspondence between the permutations in $\Pi^*_{V_1}(T_{I_1}, T_{A_1})$ and the permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A''', where π_{r_1} is a permutation of the first $h(C)$ rows of A''', π_{r_2} is a permutation of the last $k - h(C)$ rows of A', and π_c is a permutation of the k columns of A'''. Since $G''' = [I_k|A'''] = \mathcal{RREF}(\pi''(G))$, there is also a one-to-one correspondence between the permutations in $\Pi^*_{V_2}(T_{I_2}, T_{A_2})$ and the permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A''', where π_{r_1} is a permutation of the first $h(C)$ rows of A''', π_{r_2} is a permutation of the last $k - h(C)$ rows of A''', and π_c is a permutation of the k columns of A'''. Therefore, for any $G' \in \mathcal{G}^*_{V_1}(T_{I_1}, T_{A_1})$, the number of permutations $\pi_1 \in \Pi^*_{V_1}(T_{I_1}, T_{A_1})$ such that $\mathcal{RREF}(\pi_1(G)) = G'$ is equal to the number of permutations $\pi_2\Pi^*_{V_2}(T_{I_2}, T_{A_2})$ such that $\mathcal{RREF}(\pi_2(G)) = G'$. □

Corollary 6.60 Let G' be any matrix in $\mathcal{G}^*_{max}(C)$. Let $S(G')$ denote the set of all pairs $(V,(T_I,T_A))$, where $V \in \mathcal{V}_{max}(C)$ and $(T_I,T_A) \in T^*_V(G)$, such that $G' \in \mathcal{G}^*_V(T_I,T_A)$. Let $fcnt(G') = |S(G')|$. For any pair $(V,(T_I,T_A)) \in S(G')$, let $qcnt(G')$ denote the number of permutations $\pi \in \Pi^*_V(T_I,T_A)$ such that $\mathcal{RREF}(\pi(G)) = G'$. Then the size of the automorphism group of C is equal to $fcnt(G') \times qcnt(G')$.

We are now ready to give general outlines of the Combination and Permutation algorithms for our Unique Representative algorithm for the $(2k,k,d \geq 6)$ self-dual codes. For each maximal combination of non-intersecting F_d-blocks $V \in \mathcal{V}(C)$ and for each pair of sets of columns $(T_I,T_A) \in T^*_V(G)$, the Combination algorithm will, essentially, find one matrix $G' = [I_k|A'] \in \mathcal{G}^*_V(T_I,T_A)$. For each matrix $G' = [I_k|A']$ produced, the Permutation algorithm will use $G' = [I_k|A']$ to:

- produce a unique matrix $G'' = [I_k|A''] \in \mathcal{G}^*_V(T_I,T_A)$, and
- count the number of permutations $\pi \in \Pi^*_V(T_I,T_A)$ such that $\mathcal{RREF}(\pi(G)) = G''$.

The largest matrix produced by the Permutation algorithm is the unique matrix $G_0 = [I_k|A_0]$ returned by the Unique Representative algorithm. Let $S(G_0)$ denote the set of all pairs $(V,(T_I,T_A))$ such that $G_0 \in \mathcal{G}^*_V(T_I,T_A)$. The size of the automorphism group of C is found by computing $fcnt(G_0) \times qcnt(G_0)$, where $fcnt(G_0) = |S(G_0)|$ and $qcnt(G_0)$ is the number of permutations $\pi \in \Pi^*_V(T_I,T_A)$ such that $\mathcal{RREF}(\pi(G)) = G_0$, where $(V,(T_I,T_A))$ is any element in the set $S(G_0)$.

Notice that the basic functions of the Combination and Permutation algorithms differs slightly from our Combination and Permutation algorithms in our general Unique Representative algorithm. The way in which we compute the size of the automorphism group of C also differs slightly. This difference is due to the fact that for a given $(2k,k,d \geq 6)$ self-dual code C, if $G' = [I_k|A'] \in \mathcal{G}^*_{max}(C)$, there may exist more than one combination of $h(C)$ rows in A' that forms a maximal combination of non-intersecting f_{d-1}-blocks.

Let *vcnt* denote the number of different combinations of $h(C)$ rows in A' that form a maximal combination of f_{d-1}-blocks, where $G' = [I_k|A'] \in \mathcal{G}^*_{max}(C)$. Then for each pair $(T_I, T_A) \in T^*_{max}(G)$, there are exactly *vcnt* maximal combinations of non-intersecting F_d-blocks $V \in V_{max}(C)$ such that $(T_I, T_A) \in T^*_V(G)$. Let $\mathcal{V}_{max}(T_I, T_A)$ denote the set of all such V. The basic difference between our Combination algorithm for the $(2k, k, d \geq 6)$ self-dual codes and our general Combination algorithm is that for each pair $(T_I, T_A) \in T^*_{max}(G)$, instead of performing one and only one permutation in $\Pi(T_I, T_A)$ our Combination algorithm will perform one and only one permutation in each of the disjoints sets $\Pi^*_V(T_I, T_A)$, for each $V \in \mathcal{V}^*_{max}(T_I, T_A)$, for a total of $vcnt = |\mathcal{V}^*_{max}(T_I, T_A)$ permutations in $\Pi(T_I, T_A)$.

The difference between our Permutation algorithm for the $(2k, k, d \geq 6)$ self-dual codes and our general Permutation algorithm is that for each $G' = [I_k|A']$ produced by the Combination algorithm, instead of considering permutations of all the rows and columns of A' that result in a matrix $G'' = [I_k|A'']$ such that the first $h(C)$ rows of A'' contains a maximal combination of non-intersecting f_{d-1}-blocks, our Permutation algorithm will only consider permutations of the rows and columns of A' that leave the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows in A' in the first $h(C)$ rows of A'. The permutations that do not leave the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A' will be handled by the other $vcnt - 1$ matrices produced by the Combination algorithm.

Let us now consider the difference in the way we compute the size of the automorphism group of C. Let *acnt* denote the size of the automorphism group of C. Let $G_0 = [I_k|A_0]$ be the unique matrix in $\mathcal{G}^*_{max}(C)$ that is returned by our Unique Representative algorithm. Let *ccnt* denote the number of pairs $(T_I, T_A) \in T(G)$ such that $G_0 \in \mathcal{G}(T_I, T_A)$. Let *pcnt* denote the number of permutations $\pi_r\pi_c$ of the rows and columns of A_0 such that $\pi_r\pi_c(A_0) = A_0$. Our general Unique Representative algorithm finds *acnt* by computing $ccnt \times pcnt$. Let $S(G_0)$ denote the set of all pairs $(V, (T_I, T_A))$ such that $G_0 \in \mathcal{G}^*_V(T_I, T_A)$. Let $fcnt = |(G_0)|$. Let *qcnt* denote the number of permutations $\pi \in \Pi^*_V(T_I, T_A)$ such that

$\mathcal{RREF}(\pi(G)) = G_0$, where $(V, (T_I, T_A))$ is any pair in $S(G_0)$. Our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes finds *acnt* by computing $fcnt \times qcnt$.

In order to better understand what we are really doing when we find *acnt* by computing $fcnt \times qcnt$, we will find formulas that relate *fcnt* to *ccnt* and *qcnt* to *ccnt*. Now, *qcnt* is equal to the number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A_0 such that $\pi_{r_1}\pi_{r_2}\pi_c(A_0) = A_0$, where π_{r_1} is a permutation of the first $h(C)$ rows of A_0, π_{r_2} is a permutation of the last $k - h(C)$ rows of A_0, and π_c is a permutation of the k columns of A_0. Let *vcnt* be the number of different combinations of $h(C)$ rows in A_0 that form a maximal combinations of non-intersecting f_{d-1}-blocks. For each of these *vcnt* maximal combination of non-intersecting f_{d-1}-blocks in A_0, there may or may not exist a permutation $\pi_r\pi_c$ of the rows and columns of A_0 that permutes the maximal combination of non-intersecting f_{d-1}-blocks into the first $h(C)$ rows of $\pi_r\pi_c(A_0)$, such that $\pi_r\pi_c(A_0) = A_0$. Let *wcnt* denote the number of different maximal combinations of non-intersecting f_{d-1}-blocks for which such a permutation $\pi_r\pi_c$ exists. Then $pcnt = wcnt \times qcnt$. Therefore:

$$pcnt/qcnt = wcnt$$

Furthermore, since $pcnt = acnt/ccnt$ and $fcnt = acnt/qcnt = acnt/(pcnt/wcnt) = ccnt \times wcnt$:

$$fcnt/ccnt = wcnt$$

Therefore, when we find *acnt* by computing $fcnt \times qcnt$, what we are really computing is $(ccnt \times wcnt) \times (pcnt/wcnt)$.

Example 6.61 Let C be a $(22, 11, 6)$ self-dual code, generated by a matrix $G = [I_{11}|A]$, in which the unique matrix returned by our Unique Representative algorithm is $G_0 =$

$[I_{11}|A_0]$, where:

$$A_0 = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The first 4 rows of A_0 form a maximal combination of f_5-blocks (that, in this case, consists of one $f_{5,4}$-block). Therefore, $h(C) = 4$.

Our general Unique Representative algorithm finds *acnt* by computing *ccnt* $\times$ *pcnt*. In this case, *ccnt* $= 1344$ and *pcnt* $= 660$. Therefore, *acnt* $= 1344 \times 660 = 887040$.

Now, let $S(G_0)$ denote the set of all pairs $(V, (T_I, T_A))$ such that $G_0 \in \mathcal{G}_V^*(T_I, T_A)$. Let *fcnt* $= |S(G_0)|$, and for any pair $(V, (T_I, T_A)) \in S(G_0)$, let *qcnt* be the number of permutations $\pi \in \Pi_V^*(T_I, T_A)$ such that $\mathcal{RREF}(\pi(G)) = G_0$. Our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes finds *acnt* by computing *fcnt* $\times$ *qcnt*. In this case, *fcnt* $= 147840$ and *qcnt* $= 6$. Therefore, *acnt* $= 147840 \times 6 = 887040$.

Let *wcnt* denote the number of different maximal combinations of non-intersecting f_5-blocks in A_0 for which there exists a permutation $\pi_r\pi_c$ of the rows and columns of A_0 that permutes the maximal combination of non-intersecting f_5-blocks to the first $h(C) = 4$ rows of $\pi_r\pi_c(A_0)$, such that $\pi_r\pi_c(A_0) = A_0$. In this case, *wcnt* $= 110$. The relationships between *ccnt* and *fcnt*, and *pcnt* and *qcnt* is:

$$fcnt/ccnt = 147840/1344$$

$$
\begin{aligned}
&= 110 \\
&= wcnt \\
pcnt/qcnt &= 660/6 \\
&= 110 \\
&= wcnt
\end{aligned}
$$

Therefore, when we find $acnt$ by computing $fcnt \times qcnt$, what we are really computing is $(ccnt \times wcnt) \times (pcnt/wcnt)$.

We will now give a more detailed outline of the Combination and Permutation algorithms for our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes.

The Combination algorithm produces matrices in each of the sets $\Pi_V^*(T_I, T_A)$ in two stages. For each $V \in \mathcal{V}_{max}(C)$, the first stage of the Combination algorithm will produce one and only one matrix $G_V = [I_k|A_V] \in \mathcal{G}_V^*(C)$. For each matrix $G_V = [I_k|A_V]$ produced in the first stage, the second stage of the Combination algorithm will use $G_V = [I_k|A_V]$ to produce, for each $(T_I, T_A) \in \Pi_V^*(T_I, T_A)$, one and only one matrix $G' = [I_k|A'] \in \mathcal{G}_V(T_I, T_A)$. Note that by Theorem 6.59, the Combination algorithm needs only record the distinct matrices produced, along with the number of times each was produced. That is, if the algorithm produces matrices $G'_1 \in \mathcal{G}_{V_1}(T_{I_1}, T_{A_1})$ and $G'_2 \in \mathcal{G}_{V_2}(T_{I_2}, T_{A_2})$ such that $G'_1 = G'_2$ then we need only record one of them since, by Theorem 6.59, $\mathcal{G}_{V_1}(T_{I_1}, T_{A_1}) = \mathcal{G}_{V_2}(T_{I_2}, T_{A_2})$, and therefore the unique matrix produced by the Permutation algorithm will be the same for both. Furthermore, by recording the number of times G'_1 was produced, we can still compute the size of the automorphism group since, by Theorem 6.59, for any $G'' \in \mathcal{G}_{V_1}(T_{I_1}, T_{I_2})$, the number of permutations in $\Pi_{V_2}^*(T_{I_1}, T_{A_1})$ that result in G'' is equal to the number of permutations in $\Pi_{V_2}^*(T_{I_2}, T_{A_2})$ that result in G''.

For a given matrix A'', where $G'' = [I_k|A''] \in \mathcal{G}_{max}^*(G)$, let $\pi_{r_1}\pi_{r_2}\pi_c$ denote any permutation of the first $h(C)$ rows, the last $k - h(C)$ rows, and the k columns of A''. The out-

put of the Combination algorithm is a list. $L' = \{(A'_1, cnt'_1), (A'_2, cnt'_2), \ldots, (A'_m, cnt'_m)\}$, where each $G_i = [I_k|A_i] \in \mathcal{G}^*_{max}(C)$. For any $G'' = [I_k|A''] \in \mathcal{G}^*_{max}(C)$, the list L' has the following two properties.

- There exists at least one A'_i in L' for which there exists a permutation $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A'_i such that $\pi_{r_1}\pi_{r_2}\pi_c(A'_i) = A''$.
- Let $qcnt$ be the number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A'' such that $\pi_{r_1}\pi_{r_2}\pi_c(A'') = A''$. Let $fcnt$ denote the number of pairs $(V, (T_I, T_A))$ such that $G'' \in \mathcal{G}^*_V(T_I, T_A)$. Let $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ be the matrices in L' for which there exists a permutation $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A'_{i_j} such that $\pi_{r_1}\pi_{r_2}(A_{i_j})' = A''$. Then:

$$\begin{aligned} \sum_{j=1}^{n} cnt'_{i_j} &= fcnt \\ &= acnt/qcnt \end{aligned}$$

  where $acnt$ is the size of the automorphism group of C.

For each pair (A'_i, cnt'_i) in the list L' produced by the Combination algorithm, the Permutation algorithm will find the largest matrix $G''_i = [I_k|A''_i]$ such that:

- the first $h(C)$ rows of A''_i are a permutation of the first $h(C)$ rows of A'_i,
- the last $k - h(C)$ rows of A''_i are a permutation of the last $k - h(C)$ rows of A'_i,
- the columns of A''_i are a permutation of the columns of A'_i,
- the first $h(C)$ rows of A''_i are in descending order,
- the last $k - h(C)$ rows of A''_i are in descending order, and
- the columns of A''_i are in descending order.

The Permutation algorithm will also count the number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A'_i that result in A''_i. The largest $G''_i = [I_k|A'']$ produced by the Permutation algorithm is the unique matrix $G_0 = [I_k|A_0]$ returned by the Unique Representative algorithm. Let $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ denote the matrices in L' that are ordered to the unique matrix A_0. Let *qcnt* denote the number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of any one of the matrices A'_{i_j} such that $\pi_{r_1}\pi_{r_2}\pi_c(A'_i) = A_0$. The Permutation algorithm finds the size of the automorphism group of C by computing:

$$qcnt \times \sum_{j=1}^{n} cnt'_{i_j} \quad = \quad qcnt \times fcnt$$

where *fcnt* is the number of pairs $(V, (T_I, T_A))$ such that $G_0 = [I_k|A_0] \in \mathcal{G}^*_V(T_I, T_A)$.

This completes the general outlines of the Combination and Permutation algorithms for our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes. We will now look at the finer details of the Combination and Permutation algorithms.

6.9.3 The Combination Algorithm for the $(2k, k, d \geq 6)$ Self-Dual Codes

Let C be a $(2k, k, d \geq 6)$ self-dual code, generated by $G = [I_k|A]$. Let $\mathcal{V}_{max}(C)$ be the set of all maximal combinations of non-intersecting F_d-blocks in C. Our Combination algorithm for the $(2k, k, d \geq 6)$ self-dual codes works in two stages. For each $V \in \mathcal{V}_{max}(C)$, the first stage our of Combination algorithm finds one and only one permutation $\pi \in \Pi^*_V(G)$, producing a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$. The result of the first stage of the algorithm is a list L_{max} of $|\mathcal{V}_{max}(C)|$ pairs (A', V), where $G' = [I_k|A'] \in \mathcal{G}^*_V(C)$. For each pair (A', V) in the list L_{max}, the second stage of our Combination algorithm will use A' to produce one and only one permutation $\pi' \in \Pi^*_V(T_I, T_A)$ for each set $(T_I, T_A) \in T^*_V(G)$. For each π produced, the matrix $G'' = [I_k|A''] = \mathcal{RREF}(\pi'(G))$ is recorded in the list L' produced by the Combination algorithm. In this subsection, we will describe how our Combination algorithm will accomplish these two stages.

We will begin with a short review of some of the concepts we will use in describing our algorithm.

Let C be a $(2k, k, d \geq 6)$ self-dual code, generated by $G = [I_k|A]$. Let $G_1 = [I_k|A_1]$ and $G_2 = [I_k|A_2]$ be matrices in $\mathcal{G}_{[I|A]}(C)$ (where G_1 and G_2 may or may not be equal) such that both A_1 and A_2 contain at least one weight $d-1$ row. Let S_1 and S_2 be any non-intersecting sets of f_{d-1}-blocks in A_1 and A_2, respectively. Then S_1 and S_2 have an *equal combination* of f_{d-1}-blocks if and only if S_1 and S_2 contain the same number of each of the different f_{d-1}-blocks (i.e. the same number of $f_{d-1,1}$-blocks, the same number of $f_{d-1,2}$-blocks, etc.). If S_1 and S_2 do not have equal combinations of f_{d-1}-blocks then if i is the largest integer for which S_1 and S_2 do not contain the same number of $f_{d-1,i}$-blocks, then S_1 has a *larger combination* of $f_{d-1,i}$-blocks than S_2 if S_1 contains more $f_{d-1,i}$-blocks, otherwise, S_1 has a *smaller combination* of f_{d-1}-blocks than S_2.

Let $G_1 = [I_k|A_1]$ be any matrix in $\mathcal{G}_{[I|A]}(C)$ such that A_1 contains at least one weight $d-1$ row. Let S_1 be any set of non-intersecting f_{d-1}-blocks in A_1. Then S_1 is a *maximal combination of non-intersecting f_{d-1}-blocks* if and only if for any $G_2 = [I_k|A_2] \in \mathcal{G}_{[I|A]}(C)$ (where G_2 may or may not be equal to G_1) and set S_2 of non-intersecting f_{d-1}-blocks in A_2, either S_1 and S_2 have equal combinations of f_{d-1}-blocks, or S_1 has a larger combination of f_{d-1}-blocks than S_2.

Let $\mathcal{G}_{max}(C)$ denote the set of all matrices $G' = [I_k|A'] \in \mathcal{G}_{[I|A]}(C)$ such that A' contains at least one maximal combination of non-intersecting f_{d-1}-blocks. The integer $h(C)$ is the number of rows in any maximal combination of f_{d-1}-blocks in A', where $G' = [I_k|A']$ is any matrix in $\mathcal{G}_{max}(C)$. The set $\mathcal{G}^*_{max}(C)$ is the set of all matrices $G' = [I_k|A'] \in \mathcal{G}_{max}(C)$ in which the first $h(C)$ rows of A' contain a maximal combination of f_{d-1}-blocks.

Let $\Pi^*_{max}(G)$ denote the set of all permutations π of the columns of G such that $\mathcal{RREF}(\pi(G)) \in \mathcal{G}^*_{max}(C)$. Let V be any set of $h(C)$ weight d words in C. Then V is a *maximal combination of non-intersecting F_d-blocks* in C if and only if there exists a

permutation $\pi \in \Pi^{*}_{max}(G)$ that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ in which S', the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A', corresponds to the set of $h(C)$ weight d words V in C. The reason V is called a maximal combination of non-intersecting F_d-blocks is that each $f_{d-1,i}$-block in S' corresponds to an $F_{d,i}$-block in V.

Let $\mathcal{V}_{max}(C)$ denote the set of all maximal combinations of non-intersecting F_d-blocks in C. For any $V \in \mathcal{V}_{max}(C)$, the set $\Pi^{*}_{V}(G)$ is the set of all permutations $\pi \in \Pi^{*}_{max}(G)$ that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ in which the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A' corresponds to V. The set $\mathcal{G}^{*}_{V}(G)$ is the set of all matrices $\mathcal{RREF}(\pi(G))$, such that $\pi \in \Pi^{*}_{V}(G)$. For each $V \in \mathcal{V}_{max}(C)$, the first stage of our Combination algorithm will find one and only one permutation $\pi \in \Pi^{*}_{V}(G)$, producing a matrix $G' = [I_k|A'] \in \mathcal{G}^{*}_{V}(G)$. The output of the first stage of our Combination algorithm is a list L_{max} of $|\mathcal{V}_{max}(C)|$ matrices (A', V), where $G' = [I_k|A'] \in \mathcal{G}^{*}_{V}(C)$.

In order to find the list L_{max}, the first stage of our algorithm has two tasks it must perform: finding the list $\mathcal{V}_{max}(C)$ of all maximal combinations of non-intersecting F_d-blocks in C, and, for each $V \in \mathcal{V}_{max}(C)$, finding one permutation $\pi \in \Pi^{*}_{V}(G)$. Our algorithm will accomplish both of these tasks by finding sets of non-intersecting F_d-blocks in C that form what we will call *row-reducible sets of non-intersecting F_d-blocks*:

Definition 6.62 Let C be a $(2k, k, d \geq 6)$ self-dual code, generated by $G = [I_k|A]$. Let R be any set of non-intersecting F_d-blocks in C. Then R is a *row-reducible set of non-intersecting F_d-blocks* if and only if there exists a permutation π of the columns of G that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ in which each weight d word of each F_d-block in R corresponds to a row in G'. In other words, R is a row-reducible set of non-intersecting F_d-blocks if and only if there exists a permutation π of the columns of G that results in a matrix $G' = [I_k|A']$ in which each $F_{d,i}$-block in R corresponds to an $f_{d-1,i}$-block in A'.

Note that a maximal combination of non-intersecting F_d-blocks is also a row-reducible set of non-intersecting F_d-blocks.

We will now define a comparison relation on the row-reducible sets of non-intersecting F_d-blocks that will allow us to define the maximal combination of non-intersecting F_d-blocks in C in terms of the row-reducible sets of non-intersecting F_d-blocks in C.

Definition 6.63 Let R_1 and R_2 be any two row-reducible sets of non-intersecting F_d-blocks in C. Then R_1 and R_2 are said to have an *equal combination* of F_d-blocks if and only if R_1 and R_2 contain the same number of each of the different F_d-blocks (i.e. the same number of $F_{d,1}$-blocks, the same number of $F_{d,2}$-blocks, etc.). Suppose R_1 and R_2 do not have an equal combination of F_d-blocks. Let i be the largest integer in which R_1 and R_2 do not contain the same number of $F_{d-1,i}$-blocks. If R_1 has more $F_{d,i}$-blocks than R_2 then R_1 is said to have a *larger combination* of F_d-blocks than R_2, otherwise R_1 has a *smaller combination* of F_d-blocks than R_2.

With this definition we can define when a row-reducible set of non-intersecting F_d-blocks is a maximal combination of non-intersecting F_d-blocks. That is, any row-reducible set of non-intersecting F_d-blocks R in C is a maximal combination of non-intersecting F_d-blocks if and only if for any other row-reducible set of non-intersecting F_d-blocks R' in C, either R and R' have an equal combination of F_d-blocks or R has a larger combination of F_d-blocks than R'.

We will now define several sets involving row-reducible sets of non-intersecting F_d-blocks that we will use in describing the first stage of our algorithm. Let $\mathcal{R}(C)$ denote the set of all row-reducible sets of non-intersecting F_d-blocks in C. For each $R \in \mathcal{R}(C)$:

- $h(R)$ will denote the total number of weight d words in the F_d-blocks of R.
- $\Pi_R^*(G)$ will denote the set of all permutations π of the columns of G that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ such that:

- each $F_{d,i}$-block in R corresponds to an $f_{d-1,i}$-block in A', and
- the set S' of non-intersecting f_{d-1}-blocks in A', that correspond to the non-intersecting F_d-blocks in R, occur in the first $h(R)$ rows of A'.

- $\mathcal{G}^*_R(C)$ will denote the set of all generator matrices $\mathcal{RREF}(\pi(G))$, where $\pi \in \Pi^*_R(G)$.

Note that for any $R \in \mathcal{R}(C)$, if R is also a maximal combination of non-intersecting F_d-blocks V, then $\Pi^*_R(G) = \Pi^*_V(G)$, and $\mathcal{G}^*_R(C) = \mathcal{G}^*_V(C)$.

Before we describe our algorithm for producing the list L_{max} of pairs (A', V), we will describe how we can enumerate the set $\mathcal{V}_{max}(C)$ using the row-reducible sets of non-intersecting F_d-blocks in C. Our method for enumerating the set $\mathcal{V}_{max}(C)$ is recursive and is based on the fact that if R is a row-reducible set of non-intersecting F_d-blocks then any (non-empty) subset of R is also a row-reducible set of non-intersecting F_d-blocks.

Let F be an F_d-block in C. Then, since there does not necessarily exist a permutation π of the columns of G that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ such that A' contains an f_{d-1}-block that corresponds to F, it is not necessarily true that $\{F\}$ is a row-reducible set of non-intersecting F_d-blocks. Let R be a row-reducible set of non-intersecting F_d-blocks in C and let F be an F_d-block in C (that is also a row-reducible set of non-intersecting F_d-blocks) such that F does not intersect any of the F_d-blocks in R. Then, since there does not necessarily exist a permutation π of the columns of G that results in a matrix $G' = [I_k|A'] = \mathcal{RREF}(\pi(G))$ such that A' contains a set of non-intersecting f_{d-1}-blocks that corresponds to $R \cup \{F\}$, it is also not necessarily true that $R \cup \{F\}$ is a row-reducible set of non-intersecting F_d-blocks. In other words, not every combination of non-intersecting F_d-blocks in C forms a row-reducible set of non-intersecting F_d-blocks. However, determining whether or not a combination of non-intersecting F_d-blocks forms a row-reducible set of non-intersecting F_d-blocks is not trivial.

In our recursive algorithm for enumerating the the set $\mathcal{V}_{max}(C)$, we will make the assumption that given any F_d-block F and any set R, where R is either the empty set or

a row-reducible set of non-intersecting F_d-blocks, we can immediately determine whether or not $R \cup \{F\}$ is a row-reducible set of non-intersecting F_d-blocks in C. How we can in fact determine whether or not $R \cup \{F\}$ is a row-reducible set of non-intersecting F_d-blocks will be discussed later, when we describe our algorithm for producing the list L_{max} of pairs (A', V).

Now, since our method for enumerating the set $\mathcal{V}_{max}(C)$ will use the row-reducible sets of non-intersecting F_d-blocks in C, we will now define the set $\mathcal{V}_{max}(C)$ in terms of the row-reducible sets of non-intersecting F_d-blocks in C: Let $\mathcal{R}(C)$ denote the set of all row-reducible sets of non-intersecting F_d-blocks in C. Then the set $\mathcal{V}_{max}(C)$ is equal to the set of all $R \in \mathcal{R}(C)$ such that, for any $R' \in \mathcal{R}(C)$, either R and R' have equal combinations of F_d-blocks or R has a larger combination of F_d-blocks than R'.

Our method for recursively enumerating the list $\mathcal{V}_{max}(C)$ works as follows: The input to each recursive step will be a row-reducible set of non-intersecting F_d-blocks R (for the first recursive call $R = \emptyset$). The output of each recursive step will be a set L of row-reducible sets of non-intersecting F_d-blocks that contain R, such that for any pair $R'_1, R'_2 \in L$, R'_1 and R'_2 have equal combinations of F_d-blocks (for the first recursive call, L is initialized to the empty set). The first recursive call will produce a list of all of the largest F_d-blocks F in C in which $R = \{F\}$ is a row-reducible set of non-intersecting F_d-blocks C. For each $R = \{F\}$ produced, the second recursive call will produce a list of all of the largest F_d-blocks F' in C in which $R' = R \cup \{F'\}$ is a row-reducible set of non-intersecting F_d-blocks. For each R' produced, the recursion will continue until we find a row-reducible set R'' of non-intersecting F_d-blocks for which there do not exist any F_d-blocks F in C in which $R'' \cup \{F\}$ is a row-reducible set of non-intersecting F_d-blocks. Whenever such an R'' is produced, one of the following four actions will take place:

- if the list $L = \emptyset$ then R'' is inserted into L,
- else given any $R''' \in L$:

- if R'' and R''' have an equal combination of F_d-blocks then R'' is inserted into L,
- else if R'' has a larger combination of F_d-blocks than R''' then L is emptied and then R'' is inserted,
- else R'' has a smaller combination of F_d-blocks than R''' and is ignored.

Once the recursion has completed, the list L will contain all (and only) the row-reducible sets of non-intersecting F_d-blocks that are also elements of $\mathcal{V}_{max}(C)$.

Now, there is one basic problem with the outline of our algorithm we just described. That is, if the row-reducible sets of non-intersecting F_d-blocks in $\mathcal{V}_{max}(C)$ have more than one of $F_{d,i}$-block, then our recursive algorithm will produce each $R \in \mathcal{V}_{max}(C)$ more than once. For example, if $R \in \mathcal{V}_{max}(C)$ and R contains two F_{d_1}-blocks, F_1 and F_2, then when $R' = R - \{F_1, F_2\}$ is the input to a recursive call, the algorithm will produce R in two different ways:

- The algorithm first produces $R' \cup \{F_1\}$, followed by $(R' \cup \{F_1\}) \cup \{F_2\}$.
- The algorithm first produces $R' \cup \{F_2\}$, followed by $(R' \cup \{F_2\}) \cup \{F_1\}$.

In order to eliminate this problem, we will establish a precedence relation on the F_d-blocks in C. That is, we will develop a method by which we can uniquely identity each F_d-block in C, which will establish a unique ordering of the $F_{d,i}$-blocks in C. We can then use our precedence relation to prevent our algorithm from producing duplicates by only producing sets $R \cup \{F\}$ in which F has lower precedence than every F_d-block in R.

Let $\mathcal{F}(C)$ denote the set of all F_d-blocks in C. Before defining our precedence relation on the F_d-blocks in $\mathcal{F}(C)$, we will first develop a method that will allow us to uniquely identifying each F_d-block in C, without actually producing the set $\mathcal{F}(C)$. Our method for uniquely identifying the F_d-blocks $F \in \mathcal{F}(C)$ is based on the weight d words in F, and works as follows: Let $W(C)$ denote the set of all weight d words in C. For each

$\vec{w} \in W(C)$, assign a unique integer identifier to $\vec{w}$. For each $F \in \mathcal{F}(C)$, let $W(F)$ denote the integer identifiers of the weight d words in F. Then $W(C)$ uniquely identifies each $F \in \mathcal{F}(C)$. We will use $W(C)$ to uniquely identify each $F \in \mathcal{F}(C)$.

Let us now establish our precedence relation on the F_d-blocks in C. For each $F \in \mathcal{F}(C)$, let $|F|$ denote the number of weight d words in F (i.e. if F is an $F_{d,i}$-block then $|F| = i$). Let F_1 and F_2 be any two F_d-blocks in $\mathcal{F}(C)$. Then we will say F_1 has *higher precedence* than F_2 if and only if:

- $|F_1| > |F_2|$, or
- $|F_1| = |F_2|$ and $w_1 > w_2$, where w_1 is the largest integer in $W(F_1) - (W(F_1) \cap W(F_2))$ and w_2 is the largest integer in $W(F_2) - (W(F_1) \cap W(F_2))$.

If F_1 does not have higher precedence than F_2 then we will say F_1 has *lower precedence* than F_2. This establishes our precedence relation on the F_d-blocks in C.

For each $R \in \mathcal{R}(C)$ that is an input to a recursive call of our algorithm, we will now only find the largest F_d-blocks F in C in which $R' = R \cup \{F'\}$ is a row-reducible set of non-intersecting F_d-blocks *and* F' has lower precedence than the lowest precedence block in R. More formally, for a given $R \in \mathcal{R}(C)$ that is the input to a recursive call of our algorithm, we will find the F_d-blocks in the set $\mathcal{F}_{hi(lo)}(C, R)$ which we can define progressively as follows:

- Let $\mathcal{F}(C, R)$ denote the set of all $F \in \mathcal{F}(C)$ such that $R \cup \{F\} \in \mathcal{R}(C)$.
- Let $\mathcal{F}_{lo}(C)$ denote the set of all $F \in \mathcal{F}(C, R)$ such that F has lower precedence than the lowest precedence F_d-block in R.
- Then $\mathcal{F}_{hi(lo)}(C, R)$ is the set of all $F \in \mathcal{F}_{lo}(C, R)$ such that for any $F' \in \mathcal{F}_{lo}(C, R)$, $|F| \geq |F'|$.

For the first invocation of our algorithm, we will let $R = \emptyset$ be our input and will define $\mathcal{F}_{hi(lo)}(C, \emptyset)$ as follows:

- Let $\mathcal{F}(C, \emptyset)$ denote the set of all $F \in \mathcal{F}(C)$ such that $\{F\} \in \mathcal{R}(C)$.
- Then $\mathcal{F}_{hi(lo)}(R, \emptyset)$ is the set of all $F \in \mathcal{F}(C, \emptyset)$ such that for any $F' \in \mathcal{F}(C, \emptyset)$, $|F| \geq |F'|$.

We are now ready to give a formal description of our algorithm for enumerating each element in the set $\mathcal{V}_{max}(C)$ once and only once. The algorithm is recursive and is invoked with the call $EnumerateV_{max}(C, \emptyset, \mathcal{V}_{max}(C))$, where $\mathcal{V}_{max}(C)$ is initially set to the empty set.

Algorithm 6.64 $EnumerateV_{max}(C, R, L)$:

- Input: the $(2k, k, d \geq 6)$ self-dual code C and the row-reducible set of non-intersecting F_d-blocks $R \in \mathcal{R}(C)$.
- Output: a set L of row-reducible sets of non-intersecting F_d-blocks in C.

begin
  Find $\mathcal{F}_{hi(lo)}(C, R)$.
  if $\mathcal{F}_{hi(lo)}(C, R) = \emptyset$ **then**
    if $L = \emptyset$ **then** $L = \{R\}$.
    else
      Let R' be any element of L.
      if R and R' have equal combinations of F_d-blocks **then**
        $L = L \cup R$.
      else if R has a larger combination of F_d-blocks than R' **then**
        $L = \{R\}$.
    end else
  end if
  else

for each $F \in \mathcal{F}_{hi(lo)}(C,R)$ **do**

$Enumerate V_{max}(C, R \cup \{F\}, L)$.

end for

end else

end

We now have our recursive algorithm for enumerating the set $\mathcal{V}_{max}(C)$. However, what we are really interested in is producing a list L_{max} of pairs (A', V) (where $V \in \mathcal{V}_{max}(C)$ and $G' = [I_k|A'] \in \mathcal{G}^*_V(C)$) such that for each $V \in \mathcal{V}_{max}(C)$, there exists one and only one pair (A', V). We also still need to consider how we can determine, for a given $F \in \mathcal{F}(C)$, whether or not $R \cup \{F\} \in \mathcal{R}(C)$. (This is something we will need to do in order to find the set $\mathcal{F}_{hi(lo)}(C,R)$). Therefore, we will now discuss the additional steps needed in Algorithm 6.64 in order to accomplish both of these tasks.

Our input to each recursive call of the algorithm will now consist of a row-reducible set of non-intersecting F_d-blocks $R \in \mathcal{R}(C)$, a $k \times k$ matrix A', where $G' = [I_k|A'] \in \mathcal{G}^*_R(C)$, and the set of all weight d words in C that do not intersect any of the weight d words in C (which we will denote by $W(C,R)$). For the first recursive call of the algorithm, $R = \emptyset$ and $A' = A$, where $G = [I_k|A]$ generates the $(2k, k, d \geq 6)$ self-dual code C. The recursive algorithm will use the matrix A' and the set of weight d words $W(C,R)$ in C to find both the set $\mathcal{F}_{hi(lo)}(C,R)$, and for each $F \in \mathcal{F}_{hi(lo)}(C,R)$, one matrix $G'' = [I_k|A''] \in \mathcal{G}_{R\cup\{F\}}(C)$. This is accomplished as follows:

Set $\mathcal{F}_{hi(lo)}(C,R)$ to the empty set.

if $R \neq \emptyset$ **then**

Let $F_{lo}(R)$ denote the lowest precedence block in R.

Let $i = |F_{lo}(R)|$.

end if

else

Let $i = d$ (i.e. some integer greater than the largest possible F_d-block).

end else

while $((\mathcal{F}_{hi(lo)}(C,R) = \emptyset)$ **and** $(i > 0))$ **do**

Find $\mathcal{F}_i(R)$, the list of all $F_{d,i}$-blocks in $W(R)$.

for each $F \in \mathcal{F}_i(R)$ **do**

Find a permutation π of the columns of $G' = [I_k|A']$ (if such a π exists) that results in a matrix $G'' = [I_k|A''] = \mathcal{RREF}(\pi(G')) \in \mathcal{G}^*_{R\cup\{F\}}(C)$.

If such a π is found (i.e. exists) then insert F into the list $\mathcal{F}_{hi(lo)}(C,R)$.

end for

if $\mathcal{F}_{hi(lo)}(C,R) = \emptyset$ **then**

$i = i - 1$.

end while

Finding the permutation π (if it exists) can be accomplished in polynomial time with respect to k (the dimension of C) and d (the distance of C). If $i \leq 3$, the list $\mathcal{F}_i(R)$ can be found by finding all $\binom{|W(R)|}{i}$ combinations of i weight d words in $W(R)$. If $i \geq 4$, the list $\mathcal{F}_i(R)$ can be found by first finding all $\binom{|W(R)|}{3}$ combinations of 3 weight d words in $W(R)$. For each combination found that forms an $F_{d,3}$-block, we then only need to scan $W(R)$ to find all the remaining words which, when adjoined to the F_{d-3}-block, form an $F_{d,4}$-block (since, as was shown in Section 5.4 of Chapter 5, if there exist i_1 weight d words which, when adjoined to the $F_{d,3}$-block, form an $F_{d,4}$-block, then the i_1 weight d words, together with the $F_{d,3}$-block, form an F_{d,i_1+3}-block).

We are now ready to give a formal description of our recursive algorithm for finding the list L_{max} of pairs (A', V). The recursive driver *FindMaximalF$_d$List*(A, L_{max}) is given first, followed by the recursive procedure *FindNextF$_d$Block*(A', R', W', L_{max}).

Algorithm 6.65 *FindMaximalF$_d$List*(A, L_{max}):

- Input: a $k \times k$ matrix A, where $G = [I_k|A]$ generates a $(2k, k, d \geq 6)$ self-dual code

C.

- Output: a list L_{max} of pairs (A', V), where $G' = [I_k|A'] \in \mathcal{G}^*_V(C)$ and $V \in \mathcal{V}_{max}(C)$. For each $V \in \mathcal{V}_{max}(C)$, L_{max} contains exactly one pair (A', V).

begin

Let $R = \emptyset$.

Let $L_{max} = \emptyset$.

Let W be the set of all weight d words in C.

FindNextF$_d$Block(A, R, W, L_{max}).

end

Algorithm 6.66 *FindNextF$_d$Block*(A', R', W', L_{max}):

- Input: the $k \times k$ matrix A', the row-reducible set of non-intersecting F_d-blocks R', and the set of weight d words W'.
  - $G' = [I_k|A']$ generates a $(2k, k, d \geq 6)$ self-dual code C' that is equivalent to the code C, generated by $G = [I_k|A]$, where A is the input matrix to the recursive driver: FindMaximalF_dList(A, L_{max}).
  - R' is a row-reducible set of non-intersecting F_d-blocks in the code C. The first $h(R')$ rows of A' form a set S' of non-intersecting set f_{d-1}-blocks that corresponds to R'.
  - W' is the subset of all weight d words in C that do not intersect any of the weight d words in R'.
- Output: a list L_{max} of pairs (A'', R''), where R'' is a row-reducible set of non-intersecting F_d-blocks and $G'' = [I_k|A''] \in \mathcal{G}^*_{R''}(C)$. For any pair (A''_1, R''_1), $(A''_2, R''_2) \in L_{max}$, R''_1 and R''_2 have equal combinations of F_d-blocks.

begin

Find $\mathcal{F}_{hi(lo)}(C, R')$.

if $\mathcal{F}_{hi(lo)}(C, R') = \emptyset$ **then**

if $L_{max} = \emptyset$ **then** $L_{max} = \{(A', R')\}$.

else

Let (A'', R'') be any element of L_{max}.

if R' and R'' contain equal combinations of F_d-blocks **then**

Insert (A', R') into L_{max}.

else if R' has a larger combination of F_d-blocks than R'' **then**

$L_{max} = \{(A', R')\}$.

end else

end if

else

for each $F'' \in \mathcal{F}_{hi(lo)}(C, R')$ **do**

Let π be any permutation in $\Pi^*_{R'}(G', F'')$.

Let $G'' = [I_k|A''] = \mathcal{RREF}(\pi(G'))$.

Let $R'' = R' \cup F''$.

Let W'' equal W' with all weight d words that intersect the weight d words in F'' removed.

FindNextF$_d$Block(A'', R'', W'', L_{max}).

end for

end else

end

This completes our look at the first stage of our Combination algorithm for the $(2k, k, d \geq 6)$ self-dual codes. Let us now consider the second stage of our Combination algorithm: for each $(A', V) \in L_{max}$, use the matrix A' to produce one matrix $G'' = [I_k|A''] \in \mathcal{G}^*_V(T_I, T_A)$, for each $(T_I, T_A) \in T^*_V(G)$.

We will begin by describing some of the notation we will use. For each pair (A', V) in the list L_{max} produced by the first stage of our Combination algorithm:

- $\Pi^*_V(G')$ will denote the set of all permutations π of the columns of $G' = [I_k|A']$ that results in a matrix $G'' = \mathcal{RREF}(\pi(G'))$ such that $G'' \in \mathcal{G}^*_V(C)$.
- $T(G')$ will denote the set of all pairs (T'_I, T'_A), where T'_I and T'_A are equally sized subsets of the columns of I_k and A', respectively.
- $T^*_V(G')$ will denote the set of all pairs $(T'_I, T'_A) \in T(G')$ for which there exists at least one permutation $\pi \in \Pi(T'_I, T'_A)$ such that $\pi \in \Pi^*_V(G')$,
- For each $(T'_I, T'_A) \in T^*_V(G')$:
  - $\Pi^*_V(T'_I, T'_A)$ will denote the set of all permutations $\pi \in \Pi(T'_I, T'_A)$ such that $\pi \in \Pi^*_V(G')$.
  - $\mathcal{G}^*_V(T_I, T'_A)$ will denote the set of all generator matrices $\mathcal{RREF}(\pi(G'))$, where $\pi \in \Pi^*_V(T'_I, T'_A)$.

Note that there is a one-to-one correspondence between the sets of permutations $\Pi^*_V(T'_I, T'_A)$ of the columns of G' and the sets of permutations $\Pi^*_V(T_I, T_A)$ of the columns of G.

The second stage of our algorithm will produce one matrix $G'' = [I_k|A''] \in \mathcal{G}^*_V(T_I, T_A)$, for each $(T_I, T_A) \in T^*_V(G)$, by producing one matrix $G'' = [I_k|A''] \in \mathcal{G}^*_V(T'_I, T'_A)$, for each $(T'_I, T'_A) \in T^*_V(G')$.

Let (A', V) be an element in the list L_{max} produced by the first stage of our Combination algorithm. Let (T'_I, T'_A) be any pair in the set $T(G')$. Then the pair (T'_I, T_A) is also an element of $T^*_V(G')$ if and only if:

- for any $\pi \in \Pi(T'_I, T'_A)$, $\mathcal{RREF}(\pi(G'))$ has the form $[I_k|A'']$, and

- there exists a permutation $\pi \in \Pi(T'_I, T'_A)$ that results in a matrix $G'' = [I_k|A''] = \mathcal{RREF}(\pi(G''))$ in which the first $h(C)$ rows of A'' form a maximal combination of non-intersecting f_{d-1}-blocks that corresponds to V.

In other words, $(T'_I, T'_A) \in T^*_V(G)$ if and only if for any permutation $\pi \in \Pi(T'_I, T'_A)$, $G'' = [I_k|A''] = \mathcal{RREF}(\pi(G'))$, and for each of the $h(C)$ weight d words in the first $h(C)$ rows of G', the corresponding weight d word in the code C'', generated by G'', must also be rows in the matrix G''. We will use this fact to find all the different possibilities for the pairs $(T'_I, T'_A) \in T^*_V(G')$ by considering the non-zero columns of the $h(C)$ codewords in the first $h(C)$ rows of $G' = [I_k|A']$ (which correspond to the maximal combination of f_{d-1}-blocks in A').

Let us first describe the notation we will need in order to do this. Let (A', V) be any element in the list L_{max} produced by the first stage of our Combination algorithm. Let $S' = \{f_1, f_2, \ldots, f_m\}$ denote the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A'. Then for each f_i in S':

- $S'_{A,i}$ will denote the set of the columns in A' that are non-zero in the rows of the f_{d-1}-block f_i, and
- $S'_{I,i}$ will denote the set of the columns in I_k (where $G' = [I_k|A']$) that correspond to the rows in A' of the f_{d-1}-block f_i.

Note that since the f_{d-1}-blocks in S' form a non-intersecting set of f_{d-1}-blocks in A', the sets $S'_{I_1}, S'_{I_2}, \ldots, S'_{I_m}$ are all disjoint as are the sets $S'_{A_1}, S'_{A_2}, \ldots, S'_{A_m}$. Let (T'_I, T'_A) be any element in $T^*_V(G')$. For $i = 1, 2, \ldots, m$:

- T'_{I_i} will denote the columns of T'_I that are also columns of the set S'_{I_i}.
- T'_{A_i} will denote the columns of T'_A that are also columns of the set S'_{A_i}.

Finally, T'_{I_R} will denote the remaining columns of T'_I, and $T_{A'_R}$ will denote the remaining columns of T'_A.

Now, let (T'_I, T'_A) be an element of $T^*_V(G')$. Let f_j be any $f_{d-1,i}$-block in S', and let F'_j denote the i weight d rows in $G' = [I_k|A']$ that correspond to the i weight $d-1$ rows in f_j:

$$F_j = \left.\begin{array}{ccccc|ccccccccccccc} 1 & 0 & \cdots & 0 & 0^* & 1 & 1 & \cdots & 1 & 1 & \cdots & 1 & 0 & 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0^* \\ 0 & 1 & \cdots & 0 & 0^* & 1 & 1 & \cdots & 1 & 1 & \cdots & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & 0 & 0 & 0^* \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0^* & 1 & 1 & \cdots & 1 & 0 & \cdots & 1 & 1 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 0^* \end{array}\right\} i$$

Now, since $(T'_I, T'_A) \in T^*_V(G')$, we know that for any permutation $\pi \in \Pi^*_V(T'_I, T'_A)$, each of the weight d rows in F_j must also be weight d words in $G'' = \mathcal{RREF}(\pi(G'))$. Therefore, for any weight d row $r \in F_j$, if the column in the identity matrix that corresponds to r is not an element of T'_{I_j}, then $T_{A'_j}$ cannot contain any columns that have a value of one in the weight d row r. Also, for any weight d row $r \in F_j$, if the column in the identity matrix that corresponds to r is an element of T'_{I_j}, then T'_{A_j} must contain one and only one column c that has a value of one in the weight d row r. Furthermore, since none of the weight d words in F_j intersect in the identity matrix, column c must be non-zero in the remaining rows of F_j. Therefore, if $(T'_I, T'_A) \in T^*_V(G')$, the pairs of sets (T'_{i_j}, T'_{A_j}) must have the following properties:

- $|T'_{I_j}| = |T'_{A_j}|$, and
- for each column in T'_{i_j}, if r is the row of I_k that has a value of one in the column, then T'_{A_j} contains one and only one column c that has a value of one in row r of A'. Furthermore, column c has a value of zero in all other rows of the $f_{d-1,i}$-block f_j in A'.

We will refer to such pairs (T'_{I_j}, T'_{A_j}) as *block preserving pairs*. Note that if f_j is an $f_{d-1,1}$-block, there are a total of $\binom{d}{1} = d$ different block preserving pairs in (S'_{I_j}, S'_{A_j}), if

f_j is an $f_{d-1,2}$-block, there are a total of $2\binom{4}{1} = 8$ different block preserving pairs, and if f_j is an $f_{d-1,i\geq 3}$-block, there are a total of $i\binom{3}{1} = 3i$ different block preserving pairs (T'_{I_j}, T'_{A_j}).

What we have just shown is that if $(T'_I, T'_A) \in T^*_V(G')$, then each pair $(T'_{I_j}, T_{A'_j})$, where $1 \leq j \leq m$, must be a block preserving pair. This, in turn, shows us that $|T'_{I_R}| = |T'_{A_R}|$ (since for $j = 1, 2, \ldots, m$, we have $|T'_{I_j}| = |T'_{A_j}|$). Furthermore, if $(T'_I, T'_A) \in T(G')$ is any pair in which all of the subsets (T'_{I_j}, T'_{A_j}) are block preserving pairs, then if for any $\pi \in \Pi(T'_I, T'_A)$, $\mathcal{RREF}(\pi(G'))$ has the form $[I_k|A'']$, then $(T'_I, T'_A) \in T^*_V(G')$. In other words, if (T'_I, T'_A) is any pair in which all the subsets (T'_{I_j}, T'_{A_j}) are block preserving pairs, either $(T'_I, T'_A) \in T^*_V(G')$, or for any permutation $\pi \in \Pi(T'_I, T'_A)$, $\mathcal{RREF}(\pi(G'))$ does not have the form $[I_k|A'']$.

Therefore, all we need to do in the second stage of our Combination algorithm is as follows: Let $T_{prev}(G')$ denote the set of all pairs $(T'_I, T'_A) \in T(G')$ in which all of the subsets (T'_{I_j}, T'_{A_j}) are block preserving pairs. For each $(T'_I, T'_A) \in T_{prev}(G')$, find one permutation $\pi \in \Pi^*_V(T'_I, T'_A)$. If $G'' = \mathcal{RREF}(\pi(G'))$ has the form $[I_k|A'']$ then A'' is recorded in the list produced by the Combination algorithm, otherwise G'' is discarded.

Our Combination will, in fact, find the set $T_{prev}(G')$ and one permutation in each set $\Pi^*_V(T'_I, T'_A)$ by applying sequences of $\text{ADD}_{r,c}$ operations to the matrix A'. Therefore, we will now rewrite our method in terms of $\text{ADD}_{r,c}$ operations on A''.

Let $(T'_I, T'_A) \in T(G')$. Let T'_r denote the rows of A' that correspond to the columns of T'_I, and let $T'_c = T'_A$. Recall that for any $\pi \in \Pi(T'_I, T'_A)$, $\mathcal{RREF}(\pi(G'))$ has the form $[I_k|A'']$ if and only if there exists a sequence of $|T'_r|$ $\text{ADD}_{r,c}$ operations on A' that includes each element of T'_r once and each element of T'_c once. Furthermore, if such a sequence exists, then if i is any row of T'_r and j is any column of T'_c in which entry (i,j) of A' is non-zero, then there exists a sequence of $|T'_r| - 1$ $\text{ADD}_{r,c}$ operations on the matrix $\text{ADD}_{r,c}(i,j)(A')$ which includes each element of $T'_r - \{i\}$ once and each element of $T'_c - \{j\}$ once. In other words, to find such a sequence, we can select any elements from T'_r and T'_c

at each step.

Let $T_{prev}(A')$ denote the set of all pairs (T'_r, T'_c) such that $(T'_I, T'_A) \in T_{prev}(G')$, where T'_I are the columns of I_k in G' that correspond to the rows of T'_r in A', and $T'_A = T'_c$. For each pair $(T'_r, T'_c) \in T_{prev}(A')$, let $(T'_{r_j}, T_{c'_j})$ denote the rows and non-zero columns of the f_j-block in A', where $1 \leq j \leq m$. That is, $(T'_{r_j}, T_{c'_j})$ are the rows and columns of (T'_r, T'_c) that correspond to the block preserving pair (T'_{I_j}, T'_{A_j}) in (T'_I, T'_A). (Note that we will also refer to the pairs (T'_{r_j}, T'_{c_j}) as block preserving pairs.) For each pair $(T'_r, T'_c) \in T_{prev}(A')$, let (T'_{r_R}, T'_{c_R}) denote the remaining columns of (T'_r, T'_c) (i.e. the rows and columns that are not rows and columns of any of the block preserving pairs in (T'_r, T'_c)).

For each pair $(T'_r, T'_c) \in T_{prev}(A')$, our algorithm can find a sequence S of $|T'_r|$ $\text{ADD}_{r,c}$ operations on A' (if such a sequence exists) that includes each row in T'_r once and each column in T'_c once by first finding a sequence of $|T'_r - T'_{r_R}|$ $\text{ADD}_{r,c}$ operations on A' that includes each row of $T'_r - T'_{r_R}$ once and each column of $T'_c - T'_{c_R}$ once (i.e. the rows and columns of the block preserving pairs in (T'_r, T'_c)). The reason for this is that for each block preserving pair (T'_{r_j}, T'_{c_j}) in (T'_r, T'_c), there always exists a sequence of $|T'_{r_j}|$ $\text{ADD}_{r,c}$ operations on A' that includes every row of T'_{r_j} once and every column of T'_{c_j} once, as the next example demonstrates.

Example 6.67 Consider an $f_{5,3}$-block:

1	2	3	4	5	6	7	8	9	10	
1	1	1	0	1	1	0	0	0	0	1
1	1	0	1	0	0	1	1	0	0	2
1	0	1	1	0	0	0	0	1	1	3

Let $T_1 = \{\emptyset, (\{1\}, \{5\}), (\{1\}, \{6\})\}$, let $T_2 = \{\emptyset, (\{2\}, \{7\}), (\{2\}, \{8\})\}$, and let $T_3 = \{\emptyset, (\{3\}, \{9\}), (\{3\}, \{10\})\}$. Any block preserving pair (T'_{r_j}, T'_{c_j}) of the $f_{5,3}$-block is the union of any three elements, one from each of the sets T_1, T_2, and T_3. For each one of these possibilities for (T'_{r_j}, T'_{c_j}), there exists a sequence of $|T'_{r_j}|$ $\text{ADD}_{r,c}$ operations that includes each element of T'_{r_j} once and each element of T'_{c_j} once.

Therefore, there exists a sequence S of $|T'_r|$ $\text{ADD}_{r,c}$ operations on A' that includes every row of T'_r and every column of T'_c if and only if there exists a sequence S_1 of $|T'_r - T'_{r_R}|$ $\text{ADD}_{r,c}$ operations of A' that includes every row of $T'_r - T'_{r_R}$ and every column of $T'_c - T'_{c_R}$, and a sequence S_2 of $|T'_{r_R}|$ $\text{ADD}_{r,c}$ operations on $S_1(A')$ that includes every row of $T_{r'_R}$ and every column of T'_{c_R}.

This completes our look at the second stage of our Combination algorithm.

We will now give a formal description of our Combination algorithm for the $(2k, k, d \geq 6)$ self-dual codes.

Algorithm 6.68 *CombinationAlgorithm*(A, L'):

- Input: a $k \times k$ matrix A such that $G = [I_k|A]$ generates a $(2k, k, d \geq 6)$ self-dual code C.
- Output: a list $L' = \{(A'_1, cnt'_1), (A'_2, cnt'_2), \ldots, (A'_m, cnt'_m)\}$, where each $G_i = [I_k|A_i] \in \mathcal{G}^*_{max}(C)$. For any given matrix A'', where $G'' = [I_k|A''] \in \mathcal{G}^*_{max}(C)$, let $\pi_{r_1}\pi_{r_2}\pi_c$ denote any permutation of the first $h(C)$ rows of A'', the last $k - h(C)$ rows of A'', and all k columns of A''. Then, for any $G'' = [I_k|A''] \in \mathcal{G}^*_{max}(C)$, the list L' has the following two properties:
  - There exists at least one A'_i in L' for which there exists a permutation $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A'_i such that $\pi_{r_1}\pi_{r_2}\pi_c(A'_i) = A''$.
  - Let $qcnt$ be the number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A'' such that $\pi_{r_1}\pi_{r_2}\pi_c(A'') = A''$. Let $fcnt$ denote the number of pairs $(V, (T_I, T_A))$ such that $G'' \in \mathcal{G}^*_V(T_I, T_A)$. Let $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ be the matrices in L' for which there exists a permutation $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A'_{i_j} such that $\pi_{r_1}\pi_{r_2}\pi_c(A'_{i_j}) = A''$. Then $\sum_{j=1}^{n} cnt'_{i_j} = fcnt = acnt/qcnt$, where $acnt$ is the size of the automorphism group of C.

begin

Let $L' = \emptyset$.

Let $L_{max} = \emptyset$.

FindMaximalF$_d$List(A, L_{max}).

for each $(A', V) \in L_{max}$ **do**

Let $S' = \{f_1, f_2, \ldots, f_m\}$ denote the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A'.

Let $T_P(A')$ denote the set of all pairs (T'_{r_P}, T'_{c_P}), where T'_{r_P} is a subset of the rows of the f_{d-1}-blocks in S', T'_{c_P} is a subset of the non-zero columns of the f_{d-1}-blocks in S', and for $j = 1, 2, \ldots, m$, (T'_{r_j}, T'_{c_j}) is a block preserving pair, where T'_{r_j} are the rows of T'_{r_P} that are also rows of the f_{d-1}-block f_j and T'_{c_j} are the columns of T'_{c_P} that are also the non-zero columns of the f_{d-1}-block f_j.

for each $(T'_{r_P}, T'_{c_P}) \in T_P(A')$ **do**

Let S_1 be any sequence of $|T'_{r_P}|$ ADD$_{r,c}$ operations on A' that includes each row of T'_{r_P} once and each column of T'_{c_P} once.

Let $A'' = S_1(A')$.

Let $T_R(A'')$ denote the set of all pairs (T''_{r_R}, T''_{c_R}) of equally sized subsets of the rows and columns of A'' that do not include any of the rows and non-zero columns of the f_{d-1}-blocks in the first $h(C)$ rows of A''.

for each $(T''_{r_R}, T''_{c_R}) \in T_R(A'')$ **do**

if there exists a sequence S_2 of $|T''_{r_R}|$ ADD$_{r,c}$ operations on A'' that includes each row of T''_{r_R} once and each column of T''_{c_R} once **then**

Let $A''' = S_2(A'')$.

QuickOrderMatrix$(A''', 1, A''')$.

if there exists a pair $(A''', cnt''') \in L'$ **then**

Increment cnt''' by one.

else

Insert the pair $(A''', 1)$ into L'.

end if

end for

end for

end for

end

Compared to our general Combination algorithm, our Combination algorithm for the $(2k, k, d \geq 6)$ self-dual codes may in fact do more work (other than the fact that our Combination performs some of the permutations that our Permutation algorithm normally would). There are two reasons for this: One reason is that if C contains a relatively large number of weight d words, finding the list L_{max} may be very costly. Another reason is that for any pair $(T_I, T_A) \in T(G)$, such that for any $\pi \in \Pi(T_I, T_A)$ $\mathcal{RREF}(\pi(G))$ does not have the form $[I_k|A']$, the second stage of our algorithm may, essentially, perform more than one permutation in $\Pi(T_I, T_A)$. However, in practice, for the codes we enumerated (i.e. the $(2k, k, d \geq 6)$ self-dual codes for k up to and including 16) our Combination (and Permutation) algorithm for the $(2k, k, d \geq 6)$ self-dual codes is substantially better than our general Combination (and Permutation) algorithm, with two exceptions: the $(24, 12, 8)$ Golay code and the $(22, 11, 6)$ shortened Golay code.

6.9.4 The Permutation Algorithm for the $(2k, k, d \geq 6)$ Self-Dual Codes

For each pair (A'_i, cnt'_i) in the list L' produced by our Combination algorithm, the Permutation algorithm finds the largest matrix A''_i such that:

- the first $h(C)$ rows of A''_i are a permutation of the first $h(C)$ rows of A'_i,
- the last $k - h(C)$ rows of A''_i are a permutation of the last $k - h(C)$ rows of A'_i,
- the columns of A''_i are a permutation of the columns of A'_i,
- the first $h(C)$ rows of A''_i are in descending order,

- the last $k - h(C)$ rows of A_i'' are in descending order, and
- the columns of A_i'' are in descending order.

The Permutation algorithm will also find the number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the rows and columns of A_i' such that $\pi_{r_1}\pi_{r_2}\pi_c(A_i') = A_i''$, where π_{r_1} is a permutation of the first $h(C)$ rows of A_i', π_{r_2} is a permutation of the last $k - h(C)$ rows of A_i', and π_c is a permutation of the k columns of A_i'.

The largest matrix produced by the Permutation algorithm is the unique matrix $G_0 = [I_k|A_0] \in \mathcal{G}^*_{max}(C)$ that is returned by our Unique Representative algorithm. Let $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ denote the matrices in L' that are uniquely ordered to the unique matrix A_0. For any A'_{i_j}, $1 \leq j \leq n$, let $qcnt$ denote the number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of the first $h(C)$ rows, last $k-h(C)$ rows, and all k columns of A'_{i_j} such that $\pi_{r_1}\pi_{r_2}\pi_c(A'_{i_j}) = A_0$. The Permutation algorithm finds the size of the automorphism group of C by computing $qcnt \times \sum_{j=1}^{n} cnt'_{i_j}$.

Since for each (A', cnt') in L', the columns of matrix A'' produced by the Permutation algorithm are in descending order, we need only consider the different permutations of the rows of A' (since for any permutation π_r of the rows of A' there exists one and only one permutation π_c of the columns of $\pi_r(A')$ that orders the columns of $\pi_r(A')$ in descending order). Since, for each (A', cnt') in L', the first $h(C)$ rows of of the matrix A'' are a permutation of the first $h(C)$ rows of A', and the last $k - h(C)$ rows of A'' are a permutation of the last $h(C)$ rows of A', we need only consider permutations $\pi_{r_1}\pi_{r_2}$ of the rows of A' in which π_{r_1} is a permutation of the first $h(C)$ rows of A' and π_{r_2} is a permutation of the last $k - h(C)$ rows of A'.

Let A' be any matrix in the list L' produced by the Combination algorithm. Let π_{r_1} denote any permutation of the first $h(C)$ rows of A', let π_{r_2} denote any permutation of the last $k - h(C)$ rows of A', and let π_c denote the unique permutation of the columns of A' that orders the columns of $\pi_{r_1}\pi_{r_2}(A')$ in descending order. Let A'' denote the largest

matrix $\pi_{r_1}\pi_{r_2}\pi_c(A')$ in which the first $h(C)$ rows and the last $k-h(C)$ rows of A'' are both in descending order. Finally, let $qcnt(A')$ denote the number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ of A' such that $\pi_{r_1}\pi_{r_2}\pi_c(A') = A''$. Our Permutation algorithm will find A'' and $qcnt(A')$ in two stages.

Let A' be a matrix in the list L' produced by the Combination algorithm. For any permutation π_{r_1} of the first $h(C)$ rows of A', let π_{c_1} denote any permutation of the columns of A' that orders the first $h(C)$ components of $\pi_{r_1}(A')$ in descending order. The first stage of our Permutation algorithm will perform all permutations π_{r_1} of the first $h(C)$ rows of A' that results in a matrix $\pi_{r_1}\pi_{c_1}(A')$ in which the first $h(C)$ rows of $\pi_{r_1}\pi_{c_1}(A')$ are equal to the first $h(C)$ rows of A''. That is, our first stage will find all permutations π_{r_1} of the first $h(C)$ rows of A' that results in a matrix $\pi_{r_1}\pi_{c_1}(A')$ whose first $h(C)$ rows of A' are as large as possible.

For a given A' in the list L', let $A'_1, A'_2, \ldots, A'_y$ denote the matrices that are produced by the first stage of our Permutation algorithm. For each A'_i, $1 \leq i \leq y$, the second stage of our Permutation algorithm will use the *UniqueOrder* algorithm of Section 6.4 to perform permutations π_{r_2} of the last $k - h(C)$ rows of A'_i. For any permutation π_{r_2} of the last $k - h(C)$ rows of A'_i, let π_{c_2} denote the unique permutation of the columns of A'_i that orders the columns of $\pi_{r_2}(A'_i)$ in descending order. The output of the *UniqueOrder* algorithm will be the largest matrix $A''_i = \pi_{r_2}\pi_{c_2}(A'_i)$ such that the last $k - h(C)$ rows of A''_i are in descending order. The *UniqueOrder* algorithm will also compute cnt''_i, the number of permutations π_{r_2} of the last $k - h(C)$ rows of A'_i such that $\pi_{r_2}\pi_{c_2}(A'_i) = A''_i$.

For a given A' in the list L', let $A''_1, A''_2, \ldots, A''_y$ denote the matrices that are produced by the second stage of our Permutation algorithm. Then the largest matrix A''_i produced by the second stage of our algorithm is the largest matrix $A'' = \pi_{r_1}\pi_{r_2}\pi_c(A')$ whose first $h(C)$ rows, last $k - h(C)$ rows, and all k columns are in descending order. Let $A''_{i_1}, A''_{i_2}, \ldots, A''_{i_z}$ denote the matrices, produced by the second stage of our algorithm, such that $A''_{i_j} = A''$. For any i_j, where $1 \leq j \leq z$, let $cnt'' = cnt''_{i_j}$. Then $qcnt(A')$, the

number of permutations $\pi_{r_1}\pi_{r_2}\pi_c$ such that $\pi_{r_1}\pi_{r_2}\pi_c(A') = A''$, is equal to $z \times cnt''$. This gives us our two stage method for finding A'' and $qcnt(A')$.

The only thing we have left to consider is how to produce all permutations π_{r_1} of the first $h(C)$ rows of A' that results in a matrix $\pi_{r_1}\pi_{c_1}(A')$ whose first $h(C)$ rows are as large as possible.

Let S' be the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A'. Let $f_{d-1,i_1}, f_{d-1,i_2}, \ldots, f_{d-1,i_m}$ denote the f_{d-1}-blocks of S'. Then the first $h(C)$ rows of A' are as large as possible if and only if for $j = 1, 2, \ldots, m-1$:

- The rows and the non-zero columns of f_{d-1,i_j} occur before the rows and the non-zero columns of $f_{d-1,i_{j+1}}$,
- $i_j \geq i_{j+1}$, and
- the rows and the non-zero columns of each f_{d-1}-block are in descending order.

That is, the first $h(C)$ rows of A' are as large as possible if and only if the rows and the non-zero columns of each f_{d-1}-block in S' are in consecutive rows and columns (and in descending order) and occur before the rows and the non-zero columns of all smaller f_{d-1}-blocks in S'. If the f_{d-1}-blocks of the maximal combination of non-intersecting f_{d-1}-blocks S' in the first $h(C)$ rows of A' are ordered in such a manner, then we will say the maximal combination of non-intersecting f_{d-1}-blocks in the first $h(C)$ rows of A' are in *maximal descending order.*

To find all permutations $\pi_{r_1}\pi_{c_1}$ of the first $h(C)$ rows and the columns of A' that result in a matrix $\pi_{r_1}\pi_{c_1}(A')$ such that first maximal combination of f_{d-1}-blocks in the first $h(C)$ rows of $\pi_{r_1}\pi_{c_1}(A')$ are ordered in maximal descending order, we will first order the maximal combination of f_{d-1}-blocks in the first $h(C)$ rows of A' into maximal descending order. We can then use A' to find all such permutations $\pi_{r_1}\pi_{c_1}$ by considering all combinations of the following two types of permutations of the first $h(C)$ rows and the columns of A':

1. For a given pair of $f_{d-1,i}$-blocks F_1 and F_2 in A', permutations $\pi'_{r_1}\pi'_{c_1}$, where π'_{r_1} completely swaps the rows of F_1 and F_2 and π'_{c_1} completely swaps the non-zero columns of F_1 and F_2. We will refer to such permutations as permutations *between the f_{d-1}-blocks F_1 and F_2.*

2. For a given f_{d-1}-block F in A'', permutations $\pi''_{r_1}\pi''_{c_1}$, where π''_{r_1} is any permutation of the rows of F and π''_{c_1} is a permutation that orders the columns of F in descending order. We will refer to such permutations as permutations *within the f_{d-1}-block F.*

Note that for a given pair of $f_{d-1,i}$-blocks F_1 and F_2, there may exist many permutations $\pi'_{r_1}\pi'_{c_1}$ that are permutations between F_1 and F_2. In our algorithm, we will be interested in one and only one of these permutations. Therefore, we will define two permutations between two $f_{d-1,i}$-blocks as *distinct permutations between two $f_{d-1,i}$-blocks* if and only if they are not between the same two f_{d-1}-blocks. Also note that for a given f_{d-1}-block F and permutation π''_{r_1} of the rows of F, there may exist more that one permutation π''_{c_1} of the non-zero columns of F such that $\pi''_{r_1}\pi''_{c_1}$ is a permutation within F. In our algorithm, we will be interested in one and only one of the permutations π''_{c_1} (which one does not matter). Therefore, we will define two permutations within F as *distinct permutations within F* if and only if the permutations π''_{r_1} are not equal.

We are now ready to give a formal description of our Permutation algorithm for the $(2k, k, d \geq 6)$ self-dual codes.

Algorithm 6.69 *PermutationAlgorithm*$(L', A_0, acnt)$:

- Input: the list L' of pairs (A', cnt') produced by Algorithm 6.68 (i.e. our Combination algorithm for the $(2k, k, d \geq 6)$ self-dual codes). Note that the first $h(C)$ rows of each A' contain a maximal combination of non-intersecting f_{d-1}-blocks.

- Output: the $k \times k$ matrix A_0 and $acnt$. $G_0 = [I_k|A_0]$ is the largest matrix in $\mathcal{G}^*_{max}(C)$ such that:

- the first $h(C)$ rows of A_0 are in descending order,
- the last $k - h(C)$ rows of A_0 are in descending order, and
- the columns of A_0 are in descending order.

The integer *acnt* is the size of the automorphism group of C.

begin

Initially A_0 is *undefined.*

for each $(A', cnt') \in L'$ **do**

Order the maximal combination of f_{d-1}-blocks in the first $h(C)$ rows of A' in maximal descending order.

Let Π'_1 denote the set of all distinct permutations $\pi'_{r_1}\pi'_{c_1}$ between any two $f_{d-1,i}$-blocks in the first $h(C)$ rows of A'.

for each $\pi'_{r_1}\pi'_{c_1} \in \Pi'_1$ **do**

Let Π''_1 denote the set of all distinct permutations $\pi''_{r_1}\pi''_{c_1}$ within each f_{d-1}-block in the first $h(C)$ rows of $\pi'_{r_1}\pi'_{c_1}(A')$.

for each $\pi''_{r_1}\pi''_{c_1} \in \Pi''_1$ **do**

Let $A''_1 = \pi''_{r_1}\pi'_{r_1}\pi''_{c_1}\pi'_{c_1}(A')$.

UniqueOrderMatrix$(A''_1, h(C), A'', cnt'')$.

if (A_0 is *undefined*) **or** $(A'' > A_0)$ **then**

$A_0 = A''$.

$acnt = cnt' * cnt''$.

end if

else if $(A'' = A_0)$ **then**

$acnt = acnt + cnt' * cnt''$.

end else if

end for

end for

end for

end

Together, our Combination and Permutation algorithms for the $(2k, k, d \geq 6)$ self-dual codes give us our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes.

6.9.5 Combining our Unique Representative Algorithms

We will conclude this section by discussing how we can combine our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes with our Unique Representative algorithm for the $(2k, k, 4)$ e_3 and d_i-codes. The reason we will do this is that by combining our algorithms we will be able to reduce the amount of work our algorithm needs to do for the $(2k, k, 4)$ e_3 and d_i-codes that contain only a *few* weight 4 words.

We will begin by reviewing the Combination and Permutation algorithms of our Unique Representative algorithm for the $(2k, k, 4)$ e_3 and d_i-codes.

Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, 4)$ self-dual code C, where C is either an e_3-code or a d_i-code. The Combination algorithm for the $(2k, k, 4)$ e_3-codes and d_i-codes works in two stages. The first stage uses the input matrix A to produce a list L_{max} of $k \times k$ matrices A' such that $G' = [I_k|A']$ generates a code C' that is equivalent to C, and $G' = [I_k|A']$ contains a *maximal combination of generator blocks with weight 4 rows*. For each matrix $G' = [I_k|A']$, where A' is an element of the list L_{max} produced by the first stage of the algorithm:

- Let S'_I and S'_A denote the sets of k columns of I_k and A', respectively.
- Let S'_{I_B} and S'_{A_B} denote the subsets of the columns in S'_I and S'_A, respectively, that are also columns of the generator blocks with weight 4 rows in G'.
- Let $S'_{I_R} = S'_I - S'_{I_B}$ and let $S'_{A_R} = S'_A - S'_{A_B}$.

- Let $T_B(G')$ denote the set of all pairs (T'_{I_B}, T'_{A_B}), where T'_{I_B} and T'_{A_B} are equally sized subsets of the columns of S'_{I_B} and S'_{A_B}, respectively, such that for any permutation $\pi \in \Pi(T'_{I_B}, T'_{A_B})$, $\mathcal{RREF}(\pi(G')) = [I_k|A'']$, where the rows and columns of A'' are permutations of the rows and columns of A'.

- Let $T_R(G')$ denote the set of all pairs (T'_{I_R}, T'_{A_R}), where T'_{I_R} and T'_{A_R} are equally sized subsets of the columns of S'_{I_R} and S'_{A_R}, respectively, such that for any permutation $\pi \in \Pi(T'_{I_R}, T'_{A_R})$, $\mathcal{RREF}(\pi(G'))$ has the form $[I_k|A'']$.

For each matrix $G' = [I_k|A']$, where $A' \in L_{max}$, the second stage of the Combination algorithm finds, for each pair $(T'_{I_R}, T'_{A_R}) \in T_R(G')$, one permutation $\pi \in \Pi(T'_{I_R}, T'_{A_R})$ producing a matrix $G'' = [I_k|A''] = \mathcal{RREF}(\pi(G))$. For each permutation found, the pair $(A'', |T_B(G')|)$ is recorded in the list L' of pairs (A'_i, cnt') produced by the Combination algorithm. That is, if L' contains a pair (A'_i, cnt'_i) such that $A'_i = A''$, then cnt'_i is incremented by $|T_B(G')|$, else the pair $(A'', |T_B(G')|)$ is inserted into L'.

Let $L' = \{(A'_1, cnt'_1), (A'_2, cnt'_2), \ldots, (A'_m, cnt'_m)\}$ be the list that is output by the Combination algorithm. Let t denote the number of weight 3 rows in any A'_i in L' (Note: since for each A'_i in L', $G'_i = [I_k|A'_i]$ contains a maximal combination of generator blocks with weight 4 rows, every A'_i is L' contains the same number of weight 3 rows). For each A'_i in the list L', the Permutation algorithm for the $(2k, k, 4)$ e_3 and d_i-codes does the following:

- finds a unique matrix A''_i, whose rows and columns are permutations of the rows and columns of A'_i, and

- finds cnt''_i, the number of permutations $\pi_r\pi_c$ of the rows and columns of A'_i such that $\pi_r\pi_c(A'_i) = A''_i$.

The Permutation algorithm finds the unique matrix A''_i by first permuting the weight 3 rows of A'_i into the first t rows of A'_i. The algorithm then uniquely orders the weight 3

rows in the first t rows of A'_i. The algorithm then calls the *UniqueOrder* algorithm to order the remaining $k - i$ rows of A'_i, producing the unique matrix A''_i.

The largest matrix A''_i found by the Permutation algorithm is the unique matrix A_0 that is returned by our Unique Representative algorithm. The size of the automorphism group of C is computed as follows: Let $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ denote the matrices in L' that are uniquely ordered by the Permutation algorithm to the unique matrix A_0. Then the size of the automorphism group of C is equal to $acnt = \sum_{j=1}^{n} cnt'_{i_j} cnt''_{i_j}$. Note that for each A'_{i_j} that is uniquely ordered to A_0, the count cnt''_{i_j} is equal to $pcnt$, the number of permutations $\pi_r \pi_c$ of the rows and columns of A_0 such that $\pi_r \pi_c(A_0) = A_0$. Therefore, we can rewrite our equation for $acnt$ as $acnt = ccnt \times pcnt$, where $ccnt = \sum_{j=1}^{n} cnt'_{i_j}$.

This completes our review of the Combination and Permutation algorithms of our Unique Representative algorithm for the $(2k, k, 4)$ e_3 and d_i-codes. Let us now consider how we can combine them with the Combination and Permutation algorithms of our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes.

Now, in order to combine our algorithms, we must first define terminologies for the $(2k, k, 4)$ e_3 and d_i-codes that are analogous to the terms: weight d words, row-reducible sets of non-intersecting F_d-blocks, and maximal combinations of non-intersecting F_d-blocks in the $(2k, k, d \geq 6)$ self-dual codes. We will use the list L_{max} of $k \times k$ matrices A' to accomplish this (where L_{max} is the list produced by the first stage of our Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes).

We will begin by first defining some of the notation and terminology we will use. Let L_{max} be the list of $k \times k$ matrices A' produced by the first stage of our Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes. For each $A' \in L_{max}$, G' will denote the matrix $[I_k | A']$ and C' will denote the $(2k, k, 4)$ self-dual code generated by G'. For each $A' \in L_{max}$:

- $\Pi_{max_4}(G')$ will denote the set of permutations π of the columns of G' such that $\pi \in \Pi(T'_{I_R}, T'_{A_R})$, where $(T'_{I_R}, T'_{A_R}) \in T_R(G')$, and

- $\mathcal{G}_{max_4}(C')$ will denote the set of all matrices $G'' = [I_k|A''] = \mathcal{RREF}(\pi(G))$, where $\pi \in \Pi_{max_4}(G')$.

Let C'_R denote the subset of the columns of C' in which the components of all the weight 4 words in B' are zero, where B' is the set of weight 4 words in C' that are generated by the generator blocks with weight 4 words in G'. For each $G'' \in \mathcal{G}_{max_4}(C')$, let G''_R denote the subset of the columns of G'' that are not columns of the generator blocks with weight 4 rows in G''. We will define our analogous terminology in terms of the weight ≥ 6 codewords of C' whose non-zero components are completely contained in the set of columns C'_R, and the weight ≥ 6 rows of G'' whose non-zero components are completely contained in the set of columns G''_R. We will refer to the weight $w \geq 6$ codewords of C' whose non-zero components are completely contained in the columns of C'_R as *non-interfering weight w codewords*. We will refer to the weight $w \geq 6$ rows of G'' whose non-zero components are completely contained in G''_R as *non-interfering weight w rows* (i.e. non-interfering in the sense that any permutation involving the non-zero components of such rows, followed by appropriate sequence of row-operations, will not affect the columns of the generator blocks with weight 4 rows in G'').

Let $A' \in L_{max}$. Then, given any $G'' \in \mathcal{G}_{max_4}(C')$, G'' may or may not contain a non-interfering weight ≥ 6 row. Furthermore, for a given pair $G''_1, G''_2 \in \mathcal{G}_{max_4}(C')$ in which both G''_1 and G''_2 contain non-interfering weight ≥ 6 rows (if such a pair exists), the smallest weights of the non-interfering weight ≥ 6 rows in G''_1 and G''_2 may not be the same. For a given $A' \in L_{max}$, we will define $d(C')$ to be the smallest integer $w \geq 6$ (if such an integer exists) for which there exists a $G'' \in \mathcal{G}_{max_4}(C')$ such that G'' contains a non-interfering weight w row. If there does not exist a $G'' \in \mathcal{G}_{max_4}(C')$ that contains a non-interfering weight ≥ 6 row, we will define $d(C') = 0$.

Let $A' \in L_{max}$, such that $d(C') \neq 0$. Let R' be a set of non-interfering weight $d(C')$ rows in C' that also form a non-intersecting set of $F_{d(C')}$-blocks. Then we will say R' is a *row-reducible set of non-intersecting $F_{d(C')}$-blocks* if and only if there exists

a permutation $\pi \in \Pi_{max_4}(G')$ that results in a matrix $G'' = [I_k|A''] = \mathcal{RREF}(\pi(G'))$ in which A'' contains a non-intersecting set of $f_{d(C')-1}$-blocks that corresponds to R'. Let $\mathcal{R}(C')$ denote the set of all row-reducible sets of non-intersecting $F_{d(C')}$-blocks in C'. Then, for a given $R \in \mathcal{R}(C')$, we will say R is a *maximal combination of non-intersecting $F_{d(C')}$-blocks in C'* if and only if, for any $R' \in \mathcal{R}(C')$, either R and R' have equal combinations of $F_{d(C')}$-blocks or R has a larger combination of $F_{d(C')}$-blocks than R'.

Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, 4)$ self-dual code C (where C is either an e_3-code or d_i-code) such that the list L_{max} produced by the first stage of the Combination algorithm contains at least one A' such that $d(C') \neq 0$. Let A'_1, A'_2 be any pair in L_{max} such that $d(C'_1) \neq 0$ and $d(C'_2) \neq 0$ (where C'_1 and C'_2 are the codes generated by $G'_1 = [I_k|A'_1]$ and $G'_2 = [I_k|A'_2]$, respectively). Then it is not necessarily true that $d(C'_1) = d(C'_2)$ (since $\mathcal{G}_{max_4}(C'_1) \cap \mathcal{G}_{max_4}(C'_2)$ may be the empty set). Let $d(C)$ denote the smallest integer $w \geq 6$ for which there exists an $A' \in L_{max}$ such that $d(C') = w$. Let A'_1, A'_2 be any pair in L_{max} such that $d(C'_1) = d(C'_2) = d(C)$. Let V_1 and V_2 be any maximal combinations of non-intersecting $F_{d(C)}$-blocks in C_1 and C_2, respectively. Then it is not necessarily true that V_1 and V_2 have equal combinations of $F_{d(C)}$-blocks (again, since $\mathcal{G}_{max_4}(C'_1) \cap \mathcal{G}_{max_4}(C'_2)$ may be the empty set). For a given $A'_1 \in L_{max}$, such that $d(C'_1) = d(C)$, we will say that a maximal combination of non-intersecting $F_{d(C)}$-blocks V_1 in C'_1 is a *maximal combination of non-intersecting $F_{d(C)}$-blocks in C* if and only if for any $A'_2 \in L_{max}$, where $d(C'_2) = d(C)$, if V_2 is any maximal combination of non-intersecting $F_{d(C)}$-blocks in V_2, then either V_1 and V_2 have equal combinations of $F_{d(C)}$-blocks, or V_1 has a larger combination of $F_{d(C)}$-blocks than V_2.

We now have our analogous terminology for the $(2k, k, 4)$ e_3 and d_i-codes. We will now describe some of the additional notation will we use when describing how we can combine our Unique Representative algorithm for the $(2k, k, 4)$ e_3 and d_i-codes with our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes.

Let L_{max} be the list of $k \times k$ matrices A' produced by the first stage of our Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes. We will use h_B to denote the number of rows in G' that are rows of the generator blocks with weight 4 rows in G', where $G' = [I_k|A']$ and A' is any matrix in the list L_{max}. We will use $h(C)$ to denote the number of weight d words in any maximal combination of F_d-blocks in C. For each $A' \in L_{max}$, we will use the notation $\mathcal{V}_{max}(C')$ to denote the set of all maximal combinations of non-intersecting $F_{d(C')}$-blocks in C'. We will use $h(C')$ to denote the number of weight $d(C')$ words in any $V \in \mathcal{V}_{max}(C')$. For each $V \in \mathcal{V}_{max}(C')$:

- $\Pi^*_{max_4,V}(G')$ will denote the set of all permutations $\pi \in \Pi_{max_4}(G')$ that results in a matrix $G'' = [I_k|A''] = \mathcal{RREF}(\pi(G'))$ in which the first $h(C')$ rows of A'' form a set of non-intersecting $f_{d(C')-1}$-blocks that correspond to V, and the next h_B rows of G'' are the rows of the generator blocks with weight 4 rows.

- $\mathcal{G}^*_{max_4,V}(G')$ will denote the set of all generator matrices $\mathcal{RREF}(\pi(G))$, where $\pi \in \Pi^*_{max_4,V}(G')$.

- $T^*_{R,V}(G')$ will denote the set of all pairs (T'_{I_R}, T'_{A_R}) for which there exists a permutation $\pi \in \Pi(T'_{I_R}, T'_{A_R})$ such that $\pi \in \Pi^*_{max_4,V}(G')$. Note: T'_{I_R} and T'_{A_R} are equally sized subsets of S'_{I_R} and S'_{A_R}, respectively, where S'_{I_R} and S'_{A_R} are the subsets of the columns of I_k and A', respectively, that are not columns of the generator blocks with weight 4 rows in G'.

- Finally, for each $(T'_{I_R}, T'_{A_R}) \in T^*_{R,V}(G')$:

  - $\Pi^*_{R,V}(T'_{I_R}, T'_{A_R})$ will denote the set of all permutations $\pi \in \Pi(T'_{I_R}, T'_{A_R})$ such that $\pi \in \Pi^*_{R,V}(G')$.

  - $\mathcal{G}^*_{R,V}(T'_{I_R}, T'_{A_R})$ will denote the set of all matrices $\mathcal{RREF}(\pi(G'))$, where $\pi \in \Pi^*_{R,V}(T'_{I_R}, T'_{A_R})$.

This completes our description of the additional notation we will need.

We are now ready to give a general outline of how we can combine the Combination and Permutation algorithms for the $(2k, k, 4)$ e_3 and d_i-codes with the Combination and Permutation algorithms for the $(2k, k, d \geq 6)$ self-dual codes.

Let $G = [I_k|A]$ be a generator matrix for a $(2k, k, 4)$ self-dual code C, where C is either an e_3-code or a d_i-code. Our combined Combination algorithm works in three stages. The first stage of our combined Combination algorithm is the same as the first stage of the Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes. That is, our first stage will produce the list L_{max} of $k \times k$ matrices A' such that $G' = [I_k|A']$ contains a maximal combination of generator blocks with weight 4 rows. The second and third stages of our combined Combination algorithm will, essentially, use the two stages of our Combination algorithm for the $(2k, k, d \geq 6)$ self-dual codes to perform the second stage of our Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes (keeping in mind that we will now be performing some of the permutations in our Combination algorithm that our Permutation algorithm for the $(2k, k, 4)$ e_3 and d_i-codes performs).

For each A' in the list L_{max} produced by the first stage of our combined Combination algorithm, the second stage of our combined Combination algorithm will use the first stage of our Combination algorithm for the $(2k, k, d \geq 6)$ self-dual codes (with the appropriate alterations) to find $d(C')$ and, if $d(C') \neq 0$, a list $L^*_{max}(C')$ of pairs (A'', V), where $V \in \mathcal{V}_{max}(C')$ and $G'' = [I_k|A''] \in \mathcal{G}^*_{max_4,V}(C')$. If, for every $A' \in L_{max}$, $d(C') = 0$, then we will simply proceed to the second stage of the Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes, followed by the Permutation algorithm for the $(2k, k, 4)$ e_3 and d_i-codes (i.e. if for every $A' \in L_{max}$, $d(C') = 0$, then we will simply continue to use our Unique Representative algorithm for the $(2k, k, 4)$ e_3 and d_i-codes).

Suppose there exists a matrix $A' \in L_{max}$ such that $d(C') \neq 0$. Let L'_1 denote the set of all $A' \in L_{max}$ such that $d(C') = d(C)$, and each maximal combination of $F_{d(C)}$-blocks in C' is also a maximal combination of $F_{d(C)}$-blocks in C. For each $A' \in L'_1$ and each pair (A'', V) in the list $L^*_{max}(C')$, the third stage of our combined Combination algorithm

will use the matrix A'' to find one matrix $G''' = [I_k|A'''] \in \mathcal{G}^{*}_{R,V}(T'_{I_R}, T'_{A_R})$ for each pair $(T'_{I_R}, T'_{A_R}) \in T^{*}_{R,V}(G')$. For each matrix A''' produced, the pair $(A''', |T_B(G')|)$ is recorded in the list L' produced by the combined Combination algorithm (where $T_B(G')$ is the set we defined in our review of the Combination algorithm for the $(2k, k, 4)$ e_3 and d_i-codes). That is, if L' contains a pair (A'_i, cnt'_i) such that $A'_i = A'''$, then cnt'_i is incremented by $|T_B(G')|$, else the pair $(A'', |T_B(G')|)$ is inserted into L'.

Let $L' = \{(A'_1, cnt'_1), (A'_2, cnt'_2), \ldots, (A'_m, cnt'_m)\}$ be the list that is produced by our combined Combination algorithm. Each matrix A'_i in L' will have the following structure:

- the first $h(C)$ rows are a set of non-intersecting $f_{d(C)-1}$-blocks.
- the next h_B rows in A'_i form a maximal combination of generator blocks with weight 4 rows in $G'_i = [I_k|A'_i]$.

Note that only a subset of rows $h(C)+1, h(C)+2, \ldots, h(C)+h_B$ may have weight 3. Let t denote the number of these rows that have weight 3 (note that these are the only weight 3 rows in A'_i).

For each A'_i in L', the combined Permutation algorithm will produce a unique matrix A''_i as follows: The first step in the combined Permutation algorithm will be to permute the weight 3 rows in A'_i into rows $h(C)+1, h(C)+2, \ldots, h(C)+t$. The algorithm will then order the first $h(C)$ rows of A'_i, and then order rows $h(C)+1, h(C)+2, \ldots, h(C)+t$. The algorithm will then find all permutations π_{r_1} of the first $h(C)$ rows of A'_i such that the first $h(C)$ rows of $\pi_{r_1}\pi_{c_1}(A'_i)$ are equal to the first $h(C)$ rows of A'_i (where π_{c_1} is any permutation of the first $h(C)$ components of A'_i that orders the columns of $\pi_{r_1}(A'_i)$ in descending order). For each $\pi_{r_1}\pi_{c_1}(A'_i)$ produced, the next t rows are left alone (since for any permutation π_{r_2} of the weight 3 rows in $\pi_{r_1}\pi_{c_1}(A'_i)$ there exists a permutation π_{c_2} of the columns of $\pi_{r_1}\pi_{c_1}(A'_i)$ such that $\pi_{r_2}\pi_{c_2}\pi_{r_1}\pi_{c_1}(A'_i) = \pi_{r_1}\pi_{c_1}(A'_i)$). Therefore, the next step in the algorithm is to call the *UniqueOrder* algorithm with a starting row of $h(C)+t+1$. This gives us our unique matrix A''_i. For each A'_i in L', the combined

Permutation algorithm will also find $qcnt(A'_i)$, the number of permutations of the first $h(C)$ rows, the next t rows, the remaining $k-(h(C)+t)$ rows, and the columns of A'_i, that results in A''_i.

The largest matrix A''_i found by our combined Permutation algorithm is the unique matrix A_0 that is returned by our combined Unique Representative algorithm. The size of the automorphism group of C is computed as follows: Let $A'_{i_1}, A'_{i_2}, \ldots, A'_{i_n}$ denote the matrices in L' that are uniquely ordered by the Permutation algorithm to the unique matrix A_0. Let $qcnt = qcnt(A'_{i_j})$, for any $1 \leq j \leq n$. Then the size of the automorphism group of C is equal to $acnt = qcnt \times fcnt$, where $fcnt = \sum_{j=1}^{n} cnt'_{i_j}$.

This completes our look at how we can combine our Unique Representative algorithm for the $(2k, k, 4)$ e_3 and d_i-codes with our Unique Representative algorithm for the $(2k, k, d \geq 6)$ self-dual codes.

6.10 Using the Sizes of Automorphism Groups to Check Our Results

In this section, it will be shown that we can check the results of our enumeration of the inequivalent $(2k, k)$ self-dual codes by counting the number of *different* $(2k, k)$ self-dual codes in one of two ways. By *different*, we mean codes generated by matrices G_1 and G_2 such that $\mathcal{RREF}(G_1) \neq \mathcal{RREF}(G_2)$, where G_1 and G_2 may or may not generate equivalent codes.

Let $F(k)$ denote the number of different $(2k, k)$ self-dual codes. Then by Theorem 1.12 of Chapter 1 (in which we counted the number of different $(2k, k)$ self-dual codes), we can compute $F(k)$ with the following formula:

$$F(k) = \prod_{i=1}^{k-1} (2^i + 1) \tag{6.2}$$

We can also find $F(k)$ using the sizes of the automorphism groups we computed for each of the inequivalent $(2k, k)$ self-dual codes.

Let L be any complete list of inequivalent $(2k, k)$ self-dual codes. For each $C \in L$, let $\mathcal{C}(C)$ denote the equivalence class of $(2k, k)$ self-dual codes to which C belongs. Recall that, if $Aut(C)$ is the size of the automorphism group of C then the number of codes in $\mathcal{C}(C)$ is equal to $(2k)!/|Aut(C)|$. Therefore, we can also compute $F(k)$, the number of different $(2k, k)$ self-dual codes, as follows:

$$F(k) = \sum_{\forall C \in L} \frac{(2k)!}{|Aut(C)|} \tag{6.3}$$

Therefore, by Equation 6.3, we can use the sizes of the automorphism groups we compute in our enumeration to find the number of different $(2k, k)$ self-dual codes. If the results of our enumeration are correct (i.e. if we have not made any mistakes in our implementation) then the number of different $(2k, k)$ self-dual codes we find using the sizes of the automorphism groups and Equation 6.3 will be the same as the number of different $(2k, k)$ self-dual codes given to us by Equation 6.2.

Example 6.70 Let us count the number of different $(16, 8)$ self-dual codes using both Equation 6.2 and Equation 6.3. By Equation 6.2 the number of different $(16, 8)$ self-dual codes is:

$$\begin{aligned} F(8) &= \prod_{i=1}^{7} (2^i + 1) \\ &= 635037975 \end{aligned}$$

Let us now find the number of different $(16, 8)$ self-dual codes using Equation 6.3. There are exactly seven inequivalent $(16, 8)$ self-dual codes. They are the $(16, 8)$ self-dual codes C_i, $1 \leq i \leq 7$, generated by $G_i = [I_8 | A_i]$, where:

$$A_1 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&1&1&0 \\ 0&0&0&0&1&1&0&1 \\ 0&0&0&0&1&0&1&1 \\ 0&0&0&0&0&1&1&1 \end{bmatrix} \quad A_3 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&1&1&0&0&0 \\ 0&0&1&1&0&1&0&0 \\ 0&0&1&1&0&0&1&0 \\ 0&0&1&1&0&0&0&1 \\ 0&0&1&0&1&1&1&1 \\ 0&0&0&1&1&1&1&1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&1&1&0&0&0&0 \\ 0&1&1&0&1&0&0&0 \\ 0&1&0&1&1&0&0&0 \\ 0&0&0&0&0&1&1&1 \\ 0&0&1&1&1&1&1&0 \\ 0&0&1&1&1&1&0&1 \\ 0&0&1&1&1&0&1&1 \end{bmatrix} \quad A_5 = \begin{bmatrix} 1&1&1&0&0&0&0&0 \\ 1&1&0&1&0&0&0&0 \\ 1&0&1&1&0&0&0&0 \\ 0&1&1&1&0&0&0&0 \\ 0&0&0&0&1&1&1&0 \\ 0&0&0&0&1&1&0&1 \\ 0&0&0&0&1&0&1&1 \\ 0&0&0&0&0&1&1&1 \end{bmatrix} \quad A_6 = \begin{bmatrix} 1&1&1&0&0&0&0&0 \\ 1&1&0&1&0&0&0&0 \\ 1&1&0&0&1&0&0&0 \\ 1&1&0&0&0&1&0&0 \\ 1&1&0&0&0&0&1&0 \\ 1&1&0&0&0&0&0&1 \\ 1&0&1&1&1&1&1&1 \\ 0&1&1&1&1&1&1&1 \end{bmatrix}$$

$$A_7 = \begin{bmatrix} 1&1&1&0&0&0&0&0 \\ 1&1&0&1&0&0&0&0 \\ 0&0&0&0&1&1&1&0 \\ 0&0&0&0&1&1&0&1 \\ 1&1&0&0&1&0&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&0&1&1&1&1&0&0 \\ 0&1&1&1&1&1&0&0 \end{bmatrix}$$

The size of the automorphism group of each of these codes is:

$$|Aut(C_1)| = 10321920 \quad |Aut(C_2)| = 516096 \quad |Aut(C_3)| = 184320$$
$$|Aut(C_4)| = 112896 \quad |Aut(C_5)| = 3612672 \quad |Aut(C_6)| = 5160960$$
$$|Aut(C_7)| = 73728$$

Therefore, by Equation 6.3, the number of different $(16, 8)$ self-dual codes is:

$$\begin{aligned} \sum_{i=1}^{7} \frac{16!}{|Aut(C_i)|} &= 2027025 + 40540500 + 113513400 + 185328000 \\ &\quad + 5791500 + 4054050 + 283783500 \\ &= 635037975 \end{aligned}$$

which is equal to the value we found using Equation 6.2

This method was used to check the results of our enumeration of the inequivalent $(2k, k)$ self-dual codes for $2k$ up to and including 32.

6.11 Results

Based on the ideas presented in this chapter, programs for producing a unique representative A_0 and finding the size of the automorphism group for each of the indecomposable $(2k, k, 4)$ e_3 and d_i-codes, $i \geq 1$, and the $(2k, k, d \geq 6)$ self-dual codes have been implemented. These programs have been run on the lists of $(2k, k, 4)$ and $(2k, k, d \geq 6)$ self-dual codes produced using the methods of Chapters 4 and 5, respectively. The result of these programs is a list L of $k \times k$ matrices A_0 such that for any indecomposable $(2k, k)$ self-dual code C, where C is either a $(2k, k, 4)$ e_3 or d_i-code, or a $(2k, k, d \geq 6)$ self-dual code, there exists one and only one A_0 in L such that $G_0 = [I_k | A_0]$ generates a code equivalent to C. We will conclude this chapter by listing the results produced by our programs.

For each $k \leq 16$, the programs were run on three different lists of codes: the indecomposable $(2k, k, 4)$ e_3 and d_i-codes, $i \geq 2$, the $(2k, k, 4)$ d_1-codes, and the $(2k, k, d \geq 6)$ self-dual codes. For each of these lists, tables are given listing the results. The information given in each table is:

- k: the dimension of the self-dual codes in the list.
- running time: the (user) time it took to find a unique representative for each code in the list. The programs were run on an IBM RS/6000.
- number of input codes: the number of codes in the input list.
- number of output codes: this is equal to the number of inequivalent codes in the input list.

- average time per input code: the average time it took to produce a unique representative for each code in the list.

One interesting point about the table for the $(2k, k, d \geq 6)$ self-dual codes (i.e. the table in Figure 6.5) is that the average time per input code is relatively slower for the $(22, 11, 6)$ and $(24, 12, d \geq 6)$ self-dual codes compared to the average time per input code for the $(2k, k, d \geq 6)$ self-dual codes, for $k = 13, 14, 15, 16$. These relatively slow running times were due to the $(24, 12, 8)$ Golay code and the $(22, 11, 6)$ shortened Golay code.

k	running time	number of input codes	number of output codes	average time per input code
7	0.0 sec	1	1	0.0 sec
8	0.0 sec	1	1	0.0 sec
9	0.0 sec	2	2	0.0 sec
10	0.2 sec	5	5	0.0 sec
11	0.1 sec	6	6	0.0 sec
12	0.8 sec	19	19	0.0 sec
13	5.8 sec	41	37	0.2 sec
14	7.4 min	135	116	3.3 sec
15	1.4 hrs	549	346	9.3 sec
16	2.2 hrs	3812	1638	2.1 sec

Figure 6.3: results of finding a unique representative for each code in our list of indecomposable $(2k, k, 4)$ e_3 and d_i-codes, $i \geq 2$, produced using the methods of Chapter 4.

k	running time	number of input codes	number of output codes	average time per input code
7	–	–	–	–
8	–	–	–	–
9	–	–	–	–
10	–	–	–	–
11	0.1 sec	1	1	0.1 sec
12	0.9 sec	4	4	0.2 sec
13	13.8 sec	8	7	1.7 sec
14	12.0 min	44	28	16.3 sec
15	35.7 min	213	98	10.1 sec
16	10.5 hrs	2690	802	14.0 sec

Figure 6.4: results of finding a unique representative for each code in our list of indecomposable $(2k, k, 4)$ d_1-codes, produced using the methods of Chapter 4.

k	running time	number of input codes	number of output codes	average time per input code
7	–	–	–	–
8	–	–	–	–
9	–	–	–	–
10	–	–	–	–
11	52.8 sec	1	1	52.8 sec
12	7.2 hrs	3	2	2.4 hrs
13	18.1 sec	2	1	9.1 sec
14	20.0 min	7	3	2.9 min
15	2.6 hrs	32	13	5.0 min
16	37.5 hrs	215	82	10.4 min

Figure 6.5: results of finding a unique representative for each code in our list of $(2k, k, d \geq 6)$ self-dual codes produced using the methods of Chapter 5.

Chapter 7

Final Results

We will conclude this thesis by giving some of the results of our enumeration of the $(2k, k)$ self-dual codes, for $2k \leq 32$. Included in this chapter is the following information:

- A table listing the number of inequivalent $(2k, k)$ self-dual codes along with various subclasses of $(2k, k)$ self-dual codes, for $2k \leq 32$.
- Tables giving the weight distribution and the size of the automorphism group of each inequivalent $(2k, k, d \geq 4)$ self-dual code, for $2k \leq 32$.

Generator matrices for each of the inequivalent/indecomposable $(2k, k, d \geq 4)$ self-dual codes will be made available on the world wide web.

7.1 The Number of Inequivalent $(2k, k)$ Self-Dual Codes, $2k \leq 32$

The following information is given in Table 7.1:

- $2k$: the length of the codes, $1 \leq k \leq 16$.

- Self-Dual Codes: the number of inequivalent $(2k, k, d)$ self-dual codes, for $d = 2, 4, 6, 8$. For $k \leq 16$, there are no $(2k, k, d \geq 10)$ self-dual codes.

- Indecomposable: the number of inequivalent indecomposable $(2k, k, d)$ self-dual codes, for $d = 4, 6, 8$. The only indecomposable self-dual code with distance 2 is the trivial $(2, 1, 2)$ self-dual code.

- Doubly-Even: the number of inequivalent doubly-even $(2k, k)$ self-dual codes, for $d = 4, 8$. Note that a $(2k, k)$ self-dual code C is doubly-even if and only if the weight of every codeword in C is divisible by 4.

- Total: the total number of inequivalent $(2k, k)$ self-dual codes.

The number of inequivalent decomposable $(2k, k, d)$ self-dual codes can be found by subtracting the number of inequivalent indecomposable $(2k, k, d)$ self-dual codes from the total number of inequivalent $(2k, k, d)$ self-dual codes. The newly discovered information in Table 7.1 (indicated by the boldface entries) is the number of inequivalent $(32, 16, d \geq 4)$ self-dual codes and the total number of inequivalent $(32, 16)$ self-dual codes.

	Self-Dual Codes				Indecomposable			Doubly-Even		
$2k$	$d=2$	$d=4$	$d=6$	$d=8$	$d=4$	$d=6$	$d=8$	$d=4$	$d=8$	Total
2	1	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0	0	1
6	1	0	0	0	0	0	0	0	0	1
8	1	1	0	0	1	0	0	1	0	2
10	2	0	0	0	0	0	0	0	0	2
12	2	1	0	0	0	0	0	0	0	3
14	3	1	0	0	0	0	0	0	0	4
16	4	3	0	0	2	0	0	2	0	7
18	7	2	0	0	2	0	0	0	0	9
20	9	7	0	0	6	0	0	0	0	16
22	16	8	1	0	7	1	0	0	0	25
24	25	28	1	1	24	1	1	8	1	55
26	55	47	1	0	44	1	0	0	0	103
28	103	155	3	0	145	3	0	0	0	261
30	261	457	13	0	444	13	0	0	0	731
32	731	**2482**	**74**	8	**2441**	**74**	**8**	80	5	**3295**

Table 7.1: The total number of inequivalent codes in various classes of $(2k, k)$ self-dual codes, for $2k \leq 32$.

7.2 The Weight Distribution and Size of the Automorphism Group of each $(2k, k, d \geq 4)$ Self-Dual Code, $2k \leq 32$

We will conclude this chapter by listing the weight distributions and automorphism group sizes for the inequivalent $(2k, k, d \geq 4)$ self-dual codes, for $2k \leq 32$. Given in our listings is the following information:

- $2k$: the length of the codes.
- Weight Distribution: all the different weight distributions for the $(2k, k, d \geq 4)$ self-dual codes.
- Automorphism Group Sizes: for a given weight distribution W, the size of the automorphism group is given for each of the inequivalent $(2k, k, d \geq 4)$ self-dual codes with a weight distribution W. For a given weight distribution W, if there are $m \geq 2$ inequivalent $(2k, k, d \geq 4)$ self-dual codes (with weight distribution W) whose automorphism groups have the same size, say *acnt*, then instead of listing *acnt* m times, we will indicate there are exactly m inequivalent $(2k, k, d \geq 4)$ self-dual codes with weight distribution W and automorphism group size *acnt* by writing *acnt*(m).

For each of the different weight distributions, only the number of weight $2i$ codewords is given, for $4 \leq 2i \leq k$. The reason we only include these weights is that, for any $(2k, k, d \geq 4)$ self-dual code C, the number of weight 0 codewords in C is 1, the number of weight 2 codewords in C is 0, and the number of weight $2i$ codewords in C, where $k < 2i \leq 2k$, is equal to the number of weight $2k - 2i$ codewords in C.

For $k = 1, 2, \ldots, 16$, the sum of all the automorphism group sizes we computed for the $(2k, k)$ self-dual codes (including the distance 2 codes) did sum to the total number of $(2k, k)$ self-dual codes, as given by Theorem 1.12 of Chapter 1.

The newly discovered information given in our listings is the weight distribution and the size of the automorphism group for each of the inequivalent $(32, 16, d \geq 4)$ self-dual codes that are *not* doubly-even. Weight distributions and automorphism group sizes for the inequivalent $(2k, k, d \geq 4)$ self-dual codes, for $8 \leq 2k \leq 20$, were first found by V. Pless [2]. Weight distributions and automorphism group sizes for the inequivalent $(22, 11, d \geq 4)$ self-dual codes and the inequivalent $(24, 12, d \geq 4)$ self-dual codes were first found by V. Pless and N. J. A. Sloane [3]. Weight distributions and automorphism group sizes for the inequivalent doubly-even $(32, 16, d \geq 4)$ self-dual codes were first found by J. H. Conway, V. Pless, and N. J. A. Sloane [4]. The number of inequivalent $(2k, k, d \geq 4)$ self-dual codes, for $2k = 26, 28, 30$ were counted by Conway, Pless, and Sloane using the inequivalent doubly-even $(32, 16)$ self-dual codes in [4] and [5]. The weight distributions and automorphism group sizes for the inequivalent $(2k, k, d \geq 4)$ self-dual codes, for $2k = 26, 28, 30$, can also be derived from the inequivalent doubly-even $(32, 16)$ self-dual codes. In [5], Conway, Pless, and Sloane give the weight distributions for the $(2k, k, d \geq 6)$ self-dual codes, for $2k = 26, 28, 30$. However, we are the first to explicitly write down the weight distributions and automorphism group sizes for every inequivalent $(2k, k, d \geq 4)$ self-dual codes, for $2k = 26, 28, 30$. Note that in [5] there is a typo in TABLE F (which lists the weight distributions for the $(2k, k, d \geq 6)$ self-dual codes, for $2k = 26, 28, 30$). The number of weight 14 words in the codes B_{28} and C_{28} is 4860 and not 4680.

As our tables demonstrate, there are no $(2k, k)$ self-dual codes, $1 \leq k \leq 16$, that have an automorphism group of size 1. More specifically, there are no $(32, 16)$ self-dual codes that have an automorphism group of size 1 (which was not previously known). Therefore, we have proved Conjecture 1.13: the smallest possible length for a self-dual code whose automorphism group has size 1 is 34.

$2k$	Weight Distributions				Automorphism Group Sizes
	4	6	8	10	
8	14				1344
12	15	32			23040
14	14	49			56448
16	28	0	198		5160960,3612672
	12	64	102		73728
18	17	51	187		322560
	9	75	171		82944
20	45	0	210	512	1857945600
	29	32	226	448	30965760
	21	48	234	416	4423680
	17	56	238	400	1354752
	13	64	242	384	294912
	9	72	246	368	82944
	5	80	250	352	122880
22	28	49	246	700	75866112,54190080
	20	57	270	676	7372800
	16	61	282	664	2211840,774144
	12	65	294	652	221184
	8	69	306	640	36864
	4	73	318	628	36864
	0	77	330	616	887040

	Weight Distributions					
$2k$	4	6	8	10	12	Automorphism Group Sizes
24	66	0	495	0	2972	980995276800
	42	0	591	0	2828	14566293504,6936330240
	34	64	239	960	1500	990904320
	30	64	255	960	1476	1061683200
	30	0	639	0	2756	1061683200
	26	64	271	960	1452	99090432
	24	0	663	0	2720	108380160
	22	64	287	960	1428	17694720
	20	64	295	960	1416	10838016,7741440
	18	64	303	960	1404	42467328,14155776
	18	0	687	0	2684	42467328
	16	64	311	960	1392	2211840
	14	64	319	960	1380	1179648,774144
	12	64	327	960	1368	2654208,221184
	12	0	711	0	2648	7962624
	10	64	335	960	1356	393216,110592
	8	64	343	960	1344	36864
	6	64	351	960	1332	196608,18432
	6	0	735	0	2612	8847360
	4	64	359	960	1320	12288
	2	64	367	960	1308	18432
	0	64	375	960	1296	138240
	0	0	759	0	2576	244823040

$2k$	Weight Distributions					Automorphism Group Sizes
	4	6	8	10	12	
26	43	63	378	853	2758	15606743040
	31	51	426	901	2686	619315200,433520640
	29	81	274	1197	2514	1300561920
	27	79	282	1205	2502	371589120
	25	45	450	925	2650	92897280
	23	75	298	1221	2478	176947200,111476736,61931520
	22	42	462	937	2632	113799168
	21	73	306	1229	2466	26542080
	20	72	310	1233	2460	32514048
	19	71	314	1237	2454	8847360
	19	39	474	949	2614	12386304,8847360
	17	69	322	1245	2442	3096576
	16	36	486	961	2596	6967296
	15	67	330	1253	2430	5308416,1769472,1474560
	14	66	334	1257	2424	774144
	13	65	338	1261	2418	884736,442368
	13	33	498	973	2578	884736
	12	64	342	1265	2412	663552
	11	63	346	1269	2406	2654208,1032192,147456
	10	62	350	1273	2400	110592
	10	30	510	985	2560	331776
	9	61	354	1277	2394	165888,147456,73728
	8	60	358	1281	2388	18432
	7	59	362	1285	2382	98304,24576
	7	27	522	997	2542	147456
	6	58	366	1289	2376	9216
	5	57	370	1293	2370	12288,9216
	4	56	374	1297	2364	4096
	4	24	534	1009	2524	110592
	3	55	378	1301	2358	92160,3072
	2	54	382	1305	2352	3456
	1	53	386	1309	2346	4608
	1	21	546	1021	2506	483840
	0	52	390	1313	2340	11232

$2k$	Weight Distributions						Automorphism Group Sizes
	4	6	8	10	12	14	
28	91	0	1001	0	3003	8192	714164561510400
	59	0	841	512	3195	7168	2497078886400
	51	80	481	960	3563	6112	356725555200
	43	32	633	896	3419	6336	118908518400,83235962880
	43	0	761	768	3291	6656	118908518400
	35	80	401	1216	3659	5600	3963617280
	35	48	529	1088	3531	5920	5945425920
	35	0	721	896	3339	6400	18207866880
	31	72	413	1248	3651	5552	4246732800
	31	56	477	1184	3587	5712	1820786688,1300561920
	29	84	355	1328	3711	5368	1300561920
	28	98	294	1400	3773	5196	6372753408
	27	96	297	1408	3771	5184	1698693120(3)
	27	80	361	1344	3707	5344	371589120
	27	64	425	1280	3643	5504	396361728
	27	32	553	1152	3515	5824	849346560
	27	16	617	1088	3451	5984	619315200
	27	0	681	1024	3387	6144	1698693120
	23	72	373	1376	3699	5296	176947200,111476736,61931520
	23	56	437	1312	3635	5456	70778880
	23	40	501	1248	3571	5616	176947200
	23	24	565	1184	3507	5776	260112384
	21	68	379	1392	3695	5272	26542080
	21	52	443	1328	3631	5432	43352064,30965760
	20	66	382	1400	3693	5260	32514048
	19	80	321	1472	3755	5088	165150720,47185920,37158912, 28311552,18579456
	19	64	385	1408	3691	5248	8847360
	19	48	449	1344	3627	5408	169869312,56623104
	19	32	513	1280	3563	5568	53084160,18579456
	19	16	577	1216	3499	5728	56623104
	19	0	641	1152	3435	5888	159252480
	17	76	327	1488	3751	5064	8847360,4423680
	17	60	391	1424	3687	5224	3096576

$2k$	Weight Distributions						Automorphism Group Sizes
	4	6	8	10	12	14	
28	17	44	455	1360	3623	5384	8847360
	16	74	330	1496	3749	5052	4644864
	15	72	333	1504	3747	5040	9437184,5308416(3),4718592
	15	56	397	1440	3683	5200	5308416,1769472,1474560
	15	40	461	1376	3619	5360	4718592,3096576
	15	24	525	1312	3555	5520	5308416
	15	8	589	1248	3491	5680	15925248
	14	54	400	1448	3681	5188	774144
	13	68	339	1520	3743	5016	10616832,1474560,884736(3), 516096
	13	52	403	1456	3679	5176	884736,442368
	13	36	467	1392	3615	5336	10616832,884736
	12	66	342	1528	3741	5004	2654208,1327104
	12	50	406	1464	3677	5164	663552
	11	64	345	1536	3739	4992	1572864,786432,442368(3), 294912,147456
	11	48	409	1472	3675	5152	2654208,1032192,147456
	11	32	473	1408	3611	5312	1572864,442368
	11	16	537	1344	3547	5472	884736
	11	0	601	1280	3483	5632	23592960
	10	62	348	1544	3737	4980	387072,55296
	10	46	412	1480	3673	5140	110592
	9	60	351	1552	3735	4968	294912,147456(2),73728(2), 36864
	9	44	415	1488	3671	5128	165888,147456,73728
	9	28	479	1424	3607	5288	147456
	8	58	354	1560	3733	4956	36864,18432
	8	42	418	1496	3669	5116	18432
	7	56	357	1568	3731	4944	6773760,5505024,786432,229376, 147456,24576(3),18432,12288
	7	40	421	1504	3667	5104	98304,24576
	7	24	485	1440	3603	5264	786432,73728
	7	8	549	1376	3539	5424	884736
	6	54	360	1576	3729	4932	6144,3072

$2k$	Weight Distributions						Automorphism Group Sizes
	4	6	8	10	12	14	
28	6	38	424	1512	3665	5092	9216
	5	52	363	1584	3727	4920	32768,16384,8192,6144,4096, 3072
	5	36	427	1520	3663	5080	12288,9216
	5	20	491	1456	3599	5240	49152
	4	50	366	1592	3725	4908	12288,3456,3072,1024
	4	34	430	1528	3661	5068	4096
	3	48	369	1600	3723	4896	49152,27648,24576,13824,1024, 512
	3	32	433	1536	3659	5056	92160,3072
	3	16	497	1472	3595	5216	73728
	3	0	561	1408	3531	5376	21288960
	2	46	372	1608	3721	4884	768,384
	2	30	436	1544	3657	5044	3456
	1	44	375	1616	3719	4872	12288,768,384
	1	28	439	1552	3655	5032	4608
	1	12	503	1488	3591	5192	552960
	0	42	378	1624	3717	4860	1344,672
	0	26	442	1560	3653	5020	11232

$2k$	Weight Distributions						Automorphism Group Sizes
	4	6	8	10	12	14	
30	62	77	715	1166	3796	10567	6866966937600
	46	61	635	1342	4148	10151	178362777600
	42	97	495	1386	4556	9807	107017666560
	42	49	639	1386	4172	10095	291325870080,208089907200, 203928109056,72831467520
	38	53	595	1430	4324	9943	20808990720
	34	105	407	1474	4860	9503	14863564800,10404495360
	34	89	455	1474	4732	9599	5945425920
	34	57	551	1474	4476	9791	9909043200
	32	83	457	1496	4744	9571	7431782400
	32	35	601	1496	4360	9859	7431782400
	30	125	315	1518	5140	9255	42467328000
	30	61	507	1518	4628	9639	2972712960,1486356480, 1040449536
	30	45	555	1518	4500	9735	1486356480
	28	55	509	1540	4640	9611	1061683200
	28	39	557	1540	4512	9707	1061683200,743178240
	27	28	582	1551	4454	9741	1820786688,1300561920
	26	113	319	1562	5164	9199	4161798144
	26	81	415	1562	4908	9391	707788800,247726080
	26	65	463	1562	4780	9487	371589120,297271296
	26	33	559	1562	4524	9679	1415577600
	25	54	488	1573	4722	9521	433520640
	24	107	321	1584	5176	9171	1911029760
	24	91	369	1584	5048	9267	247726080
	24	75	417	1584	4920	9363	106168320
	24	67	441	1584	4856	9411	682795008
	24	43	513	1584	4664	9555	637009920
	23	80	394	1595	4990	9301	185794560,130056192
	23	32	538	1595	4606	9589	185794560
	22	101	323	1606	5188	9143	424673280
	22	85	371	1606	5060	9239	53084160
	22	69	419	1606	4932	9335	297271296(2),235929600, 212336640,148635648,49545216

	Weight Distributions						
$2k$	4	6	8	10	12	14	Automorphism Group Sizes
30	22	53	467	1606	4804	9431	141557760,106168320
	22	37	515	1606	4676	9527	141557760,106168320
	22	21	563	1606	4548	9623	297271296
	21	58	444	1617	4874	9369	30965760
	21	34	516	1617	4682	9513	260112384
	20	79	373	1628	5072	9211	35389440(2)
	20	63	421	1628	4944	9307	17694720
	20	47	469	1628	4816	9403	65028096,35389440(2),24772608
	20	31	517	1628	4688	9499	35389440
	19	84	350	1639	5142	9149	37158912
	19	68	398	1639	5014	9245	26542080
	19	60	422	1639	4950	9293	21676032
	19	52	446	1639	4886	9341	79626240
	19	36	494	1639	4758	9437	26542080,18579456(2)
	18	89	327	1650	5212	9087	70778880,42467328(3)
	18	73	375	1650	5084	9183	49545216,42467328,17694720, 12386304
	18	57	423	1650	4956	9279	21233664,11796480
	18	41	471	1650	4828	9375	84934656(2),42467328
	18	25	519	1650	4700	9471	84934656(2),21233664
	17	86	328	1661	5218	9073	18579456
	17	78	352	1661	5154	9121	8847360(2)
	17	62	400	1661	5026	9217	18579456,3096576
	17	46	448	1661	4898	9313	4423680
	17	38	472	1661	4834	9361	9289728
	17	30	496	1661	4770	9409	8847360
	17	14	544	1661	4642	9505	55738368
	16	83	329	1672	5224	9059	21233664
	16	75	353	1672	5160	9107	4644864
	16	67	377	1672	5096	9155	4423680,3538944(2),2949120
	16	51	425	1672	4968	9251	21233664,4644864,4128768, 3538944
	16	35	473	1672	4840	9347	3538944
	15	80	330	1683	5230	9045	79626240

	Weight Distributions						
$2k$	4	6	8	10	12	14	Automorphism Group Sizes
30	15	72	354	1683	5166	9093	2654208
	15	64	378	1683	5102	9141	3096576,1548288
	15	56	402	1683	5038	9189	2654208(2),737280
	15	40	450	1683	4910	9285	5308416,1548288
	15	24	498	1683	4782	9381	5308416
	15	16	522	1683	4718	9429	47775744
	14	77	331	1694	5236	9031	1192181760,11796480,8257536, 7864320,3538944(3)
	14	61	379	1694	5108	9127	4718592,2359296,1769472(3)
	14	53	403	1694	5044	9175	774144
	14	45	427	1694	4980	9223	7077888,3538944(2),2359296, 1966080
	14	29	475	1694	4852	9319	11796480,3538944,1769472
	13	74	332	1705	5242	9017	5308416,2654208
	13	66	356	1705	5178	9065	737280,516096,442368(2)
	13	58	380	1705	5114	9113	1327104
	13	50	404	1705	5050	9161	1327104,884736,442368
	13	42	428	1705	4986	9209	2654208,516096
	13	34	452	1705	4922	9257	2654208,884736,442368
	13	18	500	1705	4794	9353	2654208
	12	71	333	1716	5248	9003	5308416,3981312,1327104(2), 1179648,589824(2)
	12	63	357	1716	5184	9051	663552
	12	55	381	1716	5120	9099	15925248,1376256,737280,442368, 294912(2),258048
	12	39	429	1716	4992	9195	663552,589824(2),294912
	12	31	453	1716	4928	9243	1990656
	12	23	477	1716	4864	9291	1327104,589824
	12	7	525	1716	4736	9387	61931520
	11	68	334	1727	5254	8989	516096,442368(3),221184
	11	60	358	1727	5190	9037	221184,147456,73728
	11	52	382	1727	5126	9085	442368(2),221184
	11	44	406	1727	5062	9133	1327104,258048,221184,73728
	11	36	430	1727	4998	9181	442368(2)

	Weight Distributions						
$2k$	4	6	8	10	12	14	Automorphism Group Sizes
30	11	28	454	1727	4934	9229	442368,221184
	11	20	478	1727	4870	9277	442368
	10	65	335	1738	5260	8975	7372800,2359296(3),589824(2), 294912,196608(2)
	10	57	359	1738	5196	9023	129024,55296(2)
	10	49	383	1738	5132	9071	786432,331776,196608,147456, 110592(2),98304,73728
	10	41	407	1738	5068	9119	110592
	10	33	431	1738	5004	9167	1179648(3),196608,110592
	10	17	479	1738	4876	9263	2359296
	9	62	336	1749	5266	8961	442368,147456,73728(2),36864
	9	54	360	1749	5202	9009	73728(4),36864,24576
	9	46	384	1749	5138	9057	193536,82944,36864(3)
	9	38	408	1749	5074	9105	73728(2),49152,36864
	9	30	432	1749	5010	9153	147456,73728
	9	22	456	1749	4946	9201	442368,221184,73728
	8	59	337	1760	5272	8947	387072,98304,73728(2),49152(3), 36864(3),18432
	8	51	361	1760	5208	8995	18432(2),9216
	8	43	385	1760	5144	9043	147456,98304,73728,49152(2), 36864,24576,18432
	8	35	409	1760	5080	9091	9216
	8	27	433	1760	5016	9139	196608,73728,49152,36864
	8	11	481	1760	4888	9235	589824
	7	56	338	1771	5278	8933	98304,24576,12288(5)
	7	48	362	1771	5214	8981	967680,27648,12288(3),6144, 4608
	7	40	386	1771	5150	9029	18432,12288(2),9216
	7	32	410	1771	5086	9077	36864,24576,12288(2)
	7	24	434	1771	5022	9125	294912,12288
	7	16	458	1771	4958	9173	73728
	7	0	506	1771	4830	9269	1713761280
	6	53	339	1782	5284	8919	737280,147456,110592,65536, 49152(2),27648,24576,18432,

	Weight Distributions						
$2k$	4	6	8	10	12	14	Automorphism Group Sizes
30							16384(2),6144(3),3072(2)
	6	45	363	1782	5220	8967	6144(2),3072(2),1536
	6	37	387	1782	5156	9015	1179648,393216,294912,221184, 49152,27648,18432,8192,6144(3), 3072
	6	29	411	1782	5092	9063	3072
	6	21	435	1782	5028	9111	7741440,131072,32768,18432
	6	13	459	1782	4964	9159	165888
	5	50	340	1793	5290	8905	12288,10240,8192,4608,4096, 3072,2048(2),1536(2)
	5	42	364	1793	5226	8953	4096,3072,2048(2),1536,768
	5	34	388	1793	5162	9001	8192,3072,2048,1536
	5	26	412	1793	5098	9049	27648,6144,4608,4096
	5	18	436	1793	5034	9097	12288
	4	47	341	1804	5296	8891	98304,18432,16384,12288(2), 2304,2048(3),1536,1152,1024(2), 512(2)
	4	39	365	1804	5232	8939	3072(2),1152(2),1024,256
	4	31	389	1804	5168	8987	18432,16384,4096(2),2048, 1536(2),1024
	4	23	413	1804	5104	9035	1024
	4	15	437	1804	5040	9083	552960,55296,12288
	3	44	342	1815	5302	8877	9216,5184,2304,1728,1024, 512(2),256(2),128
	3	36	366	1815	5238	8925	4608,3072,1728,256(2),128
	3	28	390	1815	5174	8973	5184,2304,1536,768,384
	3	20	414	1815	5110	9021	138240,1024
	3	4	462	1815	4982	9117	2903040
	2	41	343	1826	5308	8863	24576,4096,1536,768,512,256(4), 192(2),64
	2	33	367	1826	5244	8911	384,128,64
	2	25	391	1826	5180	8959	8192,3072,1152,512,256
	2	17	415	1826	5116	9007	1728
	2	9	439	1826	5052	9055	122880

	Weight Distributions						
$2k$	4	6	8	10	12	14	Automorphism Group Sizes
30	1	38	344	1837	5314	8849	384,64(3),48
	1	30	368	1837	5250	8897	1536,192(2),48
	1	22	392	1837	5186	8945	1728,192
	1	14	416	1837	5122	8993	4608
	0	35	345	1848	5320	8835	645120,43008,40320,1792,1440, 192,96(2),30
	0	27	369	1848	5256	8883	48
	0	19	393	1848	5192	8931	18432,1152,576

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	120	0	1820	0	8008	0	45638	685597979049984000
	80	0	1420	0	9968	0	42598	1318457652019200
	72	96	892	1760	3960	14528	22918	188351093145600
	60	32	900	1952	4036	14400	22774	42807066624000
	60	0	1220	0	10948	0	41078	42807066624000
	56	0	1180	0	11144	0	40774	78308393877504,53271016243200, 18644855685120
	56	0	924	2048	3976	14336	22854	53271016243200
	52	96	692	1760	4940	14528	21398	1426902220800
	50	0	1120	0	11438	0	40318	5243865661440
	48	64	716	1856	4880	14464	21478	1331775406080
	46	64	696	1856	4978	14464	21326	374561832960
	44	96	612	1760	5332	14528	20790	475634073600
	44	64	676	1856	5076	14464	21174	1426902220800
	44	0	1060	0	11732	0	39862	1426902220800
	42	112	560	1712	5558	14560	20446	291325870080
	42	96	592	1760	5430	14528	20638	107017666560
	40	128	508	1664	5784	14592	20102	380507258880(2)
	40	64	636	1856	5272	14464	20870	380507258880,266355081216
	40	32	700	1952	5016	14400	21254	190253629440
	40	0	1020	0	11928	0	39558	380507258880
	38	16	712	2000	4986	14368	21294	104044953600
	38	0	1000	0	12026	0	39406	145662935040,104044953600
	36	144	436	1616	6108	14624	19606	203843174400
	36	80	564	1808	5596	14496	20374	101921587200
	36	64	596	1856	5468	14464	20566	23781703680,15854469120
	36	48	628	1904	5340	14432	20758	101921587200
	36	0	980	0	12124	0	39254	203843174400
	36	0	724	2048	4956	14336	21334	203843174400
	34	80	544	1808	5694	14496	20222	5945425920
	34	64	576	1856	5566	14464	20414	14863564800,14566293504, 10404495360(2)
	34	48	608	1904	5438	14432	20606	14863564800
	32	96	492	1760	5920	14528	19878	10569646080

	Weight Distributions							
$2k$	4	6	8	10	12	14	16	Automorphism Group Sizes
32	32	88	508	1784	5856	14512	19974	31213486080,7431782400
	32	64	556	1856	5664	14464	20262	57076088832,19025362944, 5202247680
	32	48	588	1904	5536	14432	20454	16986931200
	32	0	940	0	12320	0	38950	57076088832,31213486080
	31	100	474	1748	6001	14536	19754	18207866880
	30	80	504	1808	5890	14496	19918	1486356480
	30	64	536	1856	5762	14464	20110	2972712960
	30	40	584	1928	5570	14416	20398	20808990720,5202247680
	30	32	600	1952	5506	14400	20494	14155776000,8918138880
	30	0	920	0	12418	0	38798	42467328000,26754416640
	29	0	910	0	12467	0	38722	76473040896
	28	112	420	1712	6244	14560	19382	4459069440,3397386240
	28	96	452	1760	6116	14528	19574	6794772480,743178240
	28	80	484	1808	5988	14496	19766	3397386240
	28	72	500	1832	5924	14480	19862	1061683200
	28	64	516	1856	5860	14464	19958	4954521600,3397386240, 2229534720,1585446912, 1486356480,1040449536
	28	48	548	1904	5732	14432	20150	3397386240,1486356480
	28	40	564	1928	5668	14416	20246	2229534720
	28	32	580	1952	5604	14400	20342	6794772480(2)
	28	28	588	1964	5572	14392	20390	19118260224
	28	16	612	2000	5476	14368	20534	3397386240
	28	0	900	0	12516	0	38646	6794772480
	27	76	482	1820	6005	14488	19738	1300561920
	26	88	448	1784	6150	14512	19518	2080899072,1486356480(2)
	26	80	464	1808	6086	14496	19614	1415577600
	26	64	496	1856	5958	14464	19806	3567255552,371589120,297271296, 247726080
	26	48	528	1904	5830	14432	19998	1415577600
	26	40	544	1928	5766	14416	20094	1486356480(2)
	26	16	592	2000	5574	14368	20382	4246732800,4161798144(2), 1486356480

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	26	0	880	0	12614	0	38494	10701766656,8493465600, 4161798144,1486356480
	26	0	624	2048	5446	14336	20574	8493465600
	25	64	486	1856	6007	14464	19730	910393344
	24	128	348	1664	6568	14592	18886	10871635968
	24	104	396	1736	6376	14544	19174	1911029760
	24	96	412	1760	6312	14528	19270	8153726976,2717908992, 707788800,637009920,566231040
	24	80	444	1808	6184	14496	19462	637009920,247726080
	24	72	460	1832	6120	14480	19558	637009920
	24	64	476	1856	6056	14464	19654	2717908992,528482304,353894400, 283115520,247726080,148635648
	24	56	492	1880	5992	14448	19750	106168320
	24	48	508	1904	5928	14432	19846	637009920,353894400,283115520
	24	40	524	1928	5864	14416	19942	1040449536,247726080
	24	32	540	1952	5800	14400	20038	2717908992
	24	24	556	1976	5736	14384	20134	637009920
	24	0	860	0	12712	0	38342	32614907904,10871635968, 1911029760
	24	0	604	2048	5544	14336	20422	32614907904,10871635968, 5435817984
	23	124	346	1676	6585	14584	18858	4682022912
	23	100	394	1748	6393	14536	19146	371589120
	23	88	418	1784	6297	14512	19290	780337152
	23	76	442	1820	6201	14488	19434	185794560
	23	52	490	1892	6009	14440	19722	185794560,130056192
	23	0	850	0	12761	0	38266	2341011456
	22	104	376	1736	6474	14544	19022	424673280,212336640
	22	88	408	1784	6346	14512	19214	424673280,212336640,148635648, 106168320
	22	72	440	1832	6218	14480	19406	106168320
	22	64	456	1856	6154	14464	19502	297271296,173408256,148635648, 141557760,106168320,49545216
	22	56	472	1880	6090	14448	19598	424673280,212336640,106168320

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	22	48	488	1904	6026	14432	19694	235929600,212336640(2), 53084160
	22	40	504	1928	5962	14416	19790	424673280,212336640, 148635648(2),123863040, 106168320
	22	32	520	1952	5898	14400	19886	424673280,212336640
	22	8	568	2024	5706	14352	20174	424673280
	22	0	840	0	12810	0	38190	424673280
	21	76	422	1820	6299	14488	19282	130056192
	21	52	470	1892	6107	14440	19570	30965760
	21	40	494	1928	6011	14416	19714	260112384,65028096
	20	112	340	1712	6636	14560	18774	2831155200,1358954496
	20	96	372	1760	6508	14528	18966	226492416
	20	88	388	1784	6444	14512	19062	74317824
	20	80	404	1808	6380	14496	19158	113246208,94371840,35389440(2)
	20	72	420	1832	6316	14480	19254	35389440
	20	64	436	1856	6252	14464	19350	264241152,226492416(3), 212336640,148635648,74317824, 49545216,41287680,35389440, 24772608,17694720
	20	56	452	1880	6188	14448	19446	35389440(2),17694720
	20	52	460	1892	6156	14440	19494	65028096
	20	48	468	1904	6124	14432	19542	226492416(2),212336640, 188743680,35389440
	20	40	484	1928	6060	14416	19638	74317824(3),35389440
	20	32	500	1952	5996	14400	19734	226492416(3),188743680, 35389440
	20	16	532	2000	5868	14368	19926	226492416,148635648
	20	0	820	0	12908	0	38038	1189085184O,2831155200, 679477248,148635648
	19	84	386	1796	6461	14504	19034	53084160
	19	76	402	1820	6397	14488	19130	111476736,18579456
	19	68	418	1844	6333	14472	19226	26542080
	19	64	426	1856	6301	14464	19274	167215104,21676032

	Weight Distributions							
$2k$	4	6	8	10	12	14	16	Automorphism Group Sizes
32	19	60	434	1868	6269	14456	19322	26542080
	19	52	450	1892	6205	14440	19418	53084160,37158912,18579456
	19	44	466	1916	6141	14424	19514	26542080
	19	40	474	1928	6109	14416	19562	55738368
	19	28	498	1964	6013	14392	19706	37158912
	18	104	336	1736	6670	14544	18718	509607936
	18	88	368	1784	6542	14512	18910	42467328,35389440,24772608
	18	80	384	1808	6478	14496	19006	84934656,42467328
	18	72	400	1832	6414	14480	19102	509607936,42467328,23592960, 21233664,17694720(2),8847360
	18	64	416	1856	6350	14464	19198	84934656,42467328,16515072, 12386304(3)
	18	56	432	1880	6286	14448	19294	127401984,42467328,35389440(2), 21233664,17694720,11796480
	18	48	448	1904	6222	14432	19390	84934656(3),42467328(2), 21233664,17694720,11796480
	18	40	464	1928	6158	14416	19486	115605504,35389440(2),17694720. 12386304(3)
	18	32	480	1952	6094	14400	19582	254803968,84934656,70778880, 42467328
	18	24	496	1976	6030	14384	19678	382205952,42467328,21233664, 17694720
	18	16	512	2000	5966	14368	19774	84934656,35389440
	18	0	800	0	13006	0	37886	509607936,84934656,70778880
	18	0	544	2048	5838	14336	19966	509607936
	17	88	358	1784	6591	14512	18834	18579456
	17	76	382	1820	6495	14488	18978	18579456,9289728,4423680
	17	68	398	1844	6431	14472	19074	8847360
	17	64	406	1856	6399	14464	19122	9289728(2)
	17	60	414	1868	6367	14456	19170	4423680
	17	52	430	1892	6303	14440	19266	55738368,21676032,18579456, 4423680,3096576
	17	44	446	1916	6239	14424	19362	4423680
	17	40	454	1928	6207	14416	19410	18579456(2),9289728

	Weight Distributions							
$2k$	4	6	8	10	12	14	16	Automorphism Group Sizes
32	17	36	462	1940	6175	14408	19458	8847360
	17	16	502	2000	6015	14368	19698	18579456
	17	0	790	0	13055	0	37810	55738368
	16	96	332	1760	6704	14528	18662	75497472,21233664
	16	88	348	1784	6640	14512	18758	21233664
	16	80	364	1808	6576	14496	18854	18874368(2),14155776,11796480, 10616832(2),5898240(2)
	16	72	380	1832	6512	14480	18950	21233664,10616832(2),3538944
	16	64	396	1856	6448	14464	19046	75497472(2),24772608(2), 21233664(3),18874368,17694720, 4128768,3538944,2949120
	16	56	412	1880	6384	14448	19142	21233664(2),10616832,3538944
	16	52	420	1892	6352	14440	19190	4644864
	16	48	428	1904	6320	14432	19238	18874368,10616832(2),9437184, 7077888(2),5898240(2),3538944, 2949120
	16	40	444	1928	6256	14416	19334	21233664(3),13934592,10616832, 6193152,4644864,4423680, 4128768,3538944(2),2949120
	16	32	460	1952	6192	14400	19430	75497472,37748736,21233664, 10616832(2),9437184,7077888, 5898240
	16	28	468	1964	6160	14392	19478	13934592
	16	24	476	1976	6128	14384	19526	63700992,21233664(2)
	16	16	492	2000	6064	14368	19622	37748736,21233664
	16	8	508	2024	6000	14352	19718	21233664,10616832
	16	0	780	0	13104	0	37734	150994944,21233664
	16	0	524	2048	5936	14336	19814	150994944
	15	92	330	1772	6721	14520	18634	31850496
	15	80	354	1808	6625	14496	18778	15925248
	15	76	362	1820	6593	14488	18826	3096576,2654208
	15	68	378	1844	6529	14472	18922	5308416,1474560
	15	64	386	1856	6497	14464	18970	3096576,1548288
	15	60	394	1868	6465	14456	19018	15925248,2654208

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	15	56	402	1880	6433	14448	19066	15925248
	15	52	410	1892	6401	14440	19114	3096576,2654208,2064384, 1548288,737280
	15	48	418	1904	6369	14432	19162	15925248
	15	44	426	1916	6337	14424	19210	5308416,2654208(2),737280
	15	40	434	1928	6305	14416	19258	3096576,1548288(3)
	15	36	442	1940	6273	14408	19306	2654208(2)
	15	24	466	1976	6177	14384	19450	47775744,15925248
	15	0	770	0	13153	0	37658	79626240
	14	88	328	1784	6738	14512	18606	24772608,14155776
	14	80	344	1808	6674	14496	18702	3538944(2),1769472
	14	72	360	1832	6610	14480	18798	42467328,7077888(2),3538944(4), 1966080,1769472
	14	64	376	1856	6546	14464	18894	185794560,11796480,3538944(2), 2359296(2),1769472(3),1548288, 1032192
	14	56	392	1880	6482	14448	18990	7077888,3538944,2359296, 1966080,1769472(5),983040, 737280
	14	52	400	1892	6450	14440	19038	387072
	14	48	408	1904	6418	14432	19086	7077888,3538944(3),2359296, 1966080,1769472(2),1179648
	14	40	424	1928	6354	14416	19182	21233664,7077888,4128768, 3538944(3),2064384(2),1966080, 1769472(5),1179648
	14	32	440	1952	6290	14400	19278	7864320,7077888(2),4718592, 3538944(3),1769472(4)
	14	24	456	1976	6226	14384	19374	21233664,7077888,5898240, 3538944(3)
	14	16	472	2000	6162	14368	19470	8257536,7077888,3538944
	14	0	760	0	13202	0	37582	329042165760,254803968, 70778880,24772608,7077888
	14	0	504	2048	6034	14336	19662	84934656
	13	84	326	1796	6755	14504	18578	10616832

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	13	80	334	1808	6723	14496	18626	10616832
	13	76	342	1820	6691	14488	18674	4423680,3096576,2654208(2)
	13	72	350	1832	6659	14480	18722	1327104
	13	68	358	1844	6627	14472	18770	2654208,442368
	13	64	366	1856	6595	14464	18818	7962624,1327104,258048
	13	60	374	1868	6563	14456	18866	2654208,1327104,442368(2), 245760
	13	56	382	1880	6531	14448	18914	2654208,1327104
	13	52	390	1892	6499	14440	18962	5308416,1327104,1032192,884736, 516096(2),442368(3)
	13	48	398	1904	6467	14432	19010	2654208,1327104
	13	44	406	1916	6435	14424	19058	2654208,1327104,884736,491520, 442368(2)
	13	40	414	1928	6403	14416	19106	2654208(2),1327104,774144, 516096(2)
	13	36	422	1940	6371	14408	19154	2654208(2),737280,442368(2)
	13	32	430	1952	6339	14400	19202	2654208,1327104(2)
	13	28	438	1964	6307	14392	19250	1548288,442368
	13	24	446	1976	6275	14384	19298	2654208,1327104
	13	20	454	1988	6243	14376	19346	10616832
	13	16	462	2000	6211	14368	19394	31850496
	13	8	478	2024	6147	14352	19490	5308416
	13	0	750	0	13251	0	37506	10616832
	12	80	324	1808	6772	14496	18550	37748736,12582912,10616832, 6291456,3538944(3)
	12	76	332	1820	6740	14488	18598	2654208
	12	72	340	1832	6708	14480	18646	1769472,589824(2)
	12	64	356	1856	6644	14464	18742	3145728,2752512,1572864, 1179648,884736(2),688128, 663552,589824(6),491520,294912
	12	60	364	1868	6612	14456	18790	1990656
	12	56	372	1880	6580	14448	18838	1769472(2),1327104,884736, 663552,589824(2),294912(3)
	12	52	380	1892	6548	14440	18886	663552,129024

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	12	48	388	1904	6516	14432	18934	37748736,6291456,4718592, 3538944(2),3145728,1572864, 1179648,884736,663552, 589824(5),442368,294912(3), 245760
	12	40	404	1928	6452	14416	19030	4128768,1769472(3),884736, 737280,663552,589824(4),442368, 344064,294912(4),258048(2)
	12	36	412	1940	6420	14408	19078	663552
	12	32	420	1952	6388	14400	19126	18874368,15925248,6291456(2), 3538944,1572864,663552, 589824(5),294912
	12	28	428	1964	6356	14392	19174	2654208
	12	24	436	1976	6324	14384	19222	15925248,1769472(4),1327104, 1179648,884736,589824(2)
	12	20	444	1988	6292	14376	19270	1327104
	12	16	452	2000	6260	14368	19318	94371840,3145728,2654208, 1769472,1179648
	12	12	460	2012	6228	14360	19366	7962624
	12	8	468	2024	6196	14352	19414	5308416,3538944
	12	0	740	0	13300	0	37430	37748736,15925248,5308416, 3538944
	12	0	484	2048	6132	14336	19510	37748736
	11	76	322	1820	6789	14488	18522	2654208,884736
	11	72	330	1832	6757	14480	18570	442368(2)
	11	68	338	1844	6725	14472	18618	294912,221184(2)
	11	64	346	1856	6693	14464	18666	258048,221184(2)
	11	60	354	1868	6661	14456	18714	442368(2),221184(2),73728(2)
	11	56	362	1880	6629	14448	18762	442368(2),221184(3)
	11	52	370	1892	6597	14440	18810	276480,258048,221184(4),193536, 172032,147456(2),73728
	11	48	378	1904	6565	14432	18858	442368(3),221184(4)
	11	44	386	1916	6533	14424	18906	884736,442368(2),221184(2), 73728(3)

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	11	40	394	1928	6501	14416	18954	442368(3),344064,221184(4), 86016
	11	36	402	1940	6469	14408	19002	442368(2),221184,147456(2), 73728(2)
	11	32	410	1952	6437	14400	19050	442368(4),221184(3)
	11	28	418	1964	6405	14392	19098	442368,221184,147456
	11	24	426	1976	6373	14384	19146	442368(3),221184
	11	16	442	2000	6309	14368	19242	2064384,442368,221184
	11	0	730	0	13349	0	37354	18579456,1327104
	10	72	320	1832	6806	14480	18494	77414400,2359296,1572864, 589824(3),294912
	10	64	336	1856	6742	14464	18590	786432,331776,294912(2),258048, 196608,147456(3),110592,98304
	10	60	344	1868	6710	14456	18638	55296
	10	56	352	1880	6678	14448	18686	1179648,1105920,589824(5), 442368,393216,294912(6), 196608(2),147456,110592, 98304(3),73728,55296
	10	52	360	1892	6646	14440	18734	110592,55296,43008
	10	48	368	1904	6614	14432	18782	2359296,1179648,786432,663552, 294912(3),221184,196608(3), 147456(3),110592,98304(2), 36864
	10	44	376	1916	6582	14424	18830	110592,55296
	10	40	384	1928	6550	14416	18878	3538944,2211840,1179648,786432, 774144,589824(2),393216, 294912(3),196608(4),147456(3), 110592(3),98304(3),86016, 73728(2),64512,55296
	10	36	392	1940	6518	14408	18926	110592,55296(2)
	10	32	400	1952	6486	14400	18974	7372800,2359296,1179648(3), 786432(2),589824,294912(3), 221184,196608(2),147456(2), 110592(2),98304(3),73728

	Weight Distributions							
$2k$	4	6	8	10	12	14	16	Automorphism Group Sizes
32	10	28	408	1964	6454	14392	19022	1161216,165888,110592,55296
	10	24	416	1976	6422	14384	19070	1179648,589824(2),294912(4), 196608,147456(2)
	10	16	432	2000	6358	14368	19166	7077888,3538944,2359296(2), 1179648,294912
	10	8	448	2024	6294	14352	19262	2359296,1179648(2),294912
	10	0	720	0	13398	0	37278	1703116800,4718592,2359296, 1179648
	10	0	464	2048	6230	14336	19358	4718592
	9	68	318	1844	6823	14472	18466	884736,442368,147456
	9	64	326	1856	6791	14464	18514	73728,36864
	9	60	334	1868	6759	14456	18562	147456,73728(3),49152(2), 36864(2)
	9	56	342	1880	6727	14448	18610	221184,73728(2),36864(5)
	9	52	350	1892	6695	14440	18658	147456,73728(11),55296,43008, 24576,12288
	9	48	358	1904	6663	14432	18706	147456,110592,82944,73728, 36864(6)
	9	44	366	1916	6631	14424	18754	73728(7),55296,49152(2), 36864(3),12288
	9	40	374	1928	6599	14416	18802	147456,73728(2),36864(7),32256, 27648
	9	36	382	1940	6567	14408	18850	294912,221184,147456,73728(5), 49152,36864(3),24576,12288
	9	32	390	1952	6535	14400	18898	147456,73728(2),36864(8)
	9	28	398	1964	6503	14392	18946	73728(3),36864(2),24576
	9	24	406	1976	6471	14384	18994	147456,73728(3),36864(2)
	9	20	414	1988	6439	14376	19042	73728
	9	16	422	2000	6407	14368	19090	73728
	9	12	430	2012	6375	14360	19138	221184
	9	8	438	2024	6343	14352	19186	442368
	9	0	710	0	13447	0	37202	442368
	8	64	316	1856	6840	14464	18438	12582912,2580480,1048576, 786432(2),589824(2),221184,

	Weight Distributions							
$2k$	4	6	8	10	12	14	16	Automorphism Group Sizes
32								196608(2),147456,98304, 49152(2)
	8	60	324	1868	6808	14456	18486	18432(2)
	8	56	332	1880	6776	14448	18534	98304,73728,49152(3),36864(3), 24576(5),18432(2)
	8	52	340	1892	6744	14440	18582	72576,36864,18432,9216(2)
	8	48	348	1904	6712	14432	18630	3145728,1048576,393216(2), 294912,262144,196608(2),98304, 49152(9),36864(4),24576(2), 18432,12288(2),9216(3)
	8	44	356	1916	6680	14424	18678	36864,18432,9216(5)
	8	40	364	1928	6648	14416	18726	294912,129024,98304(3), 73728(4),49152(7),48384, 36864(3),24576(5),18432(4), 9216,6144
	8	36	372	1940	6616	14408	18774	36864,18432(2),9216(4)
	8	32	380	1952	6584	14400	18822	1048576,524288(2),393216(3), 196608(2),147456(2),98304(2), 73728(2),49152(5),36864(3), 24576(5),9216(2)
	8	28	388	1964	6552	14392	18870	36864(2),18432,9216(2)
	8	24	396	1976	6520	14384	18918	294912,196608,147456,98304(2), 73728,49152(7),36864(4),24576, 18432
	8	20	404	1988	6488	14376	18966	36864
	8	16	412	2000	6456	14368	19014	3870720,1769472,1048576,786432, 589824(2),442368,147456, 98304(2),49152,36864
	8	8	428	2024	6392	14352	19110	98304
	8	0	700	0	13496	0	37126	528482304,81285120,75497472, 22020096,3538944,442368,294912
	8	0	444	2048	6328	14336	19206	25165824,8388608
	7	60	314	1868	6857	14456	18410	49152,24576(3),12288
	7	56	322	1880	6825	14448	18458	12288(5),6144

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	7	52	330	1892	6793	14440	18506	193536,73728,27648(2),24576, 18432,12288(5),9216(2),6144(2)
	7	48	338	1904	6761	14432	18554	12288(5),6144(6),2304
	7	44	346	1916	6729	14424	18602	49152,24576,18432,12288(7), 9216(2),6144(5),4608(2)
	7	40	354	1928	6697	14416	18650	72576,36864,24576,18432, 12288(6),9216,6144(6),2304
	7	36	362	1940	6665	14408	18698	36864,24576(3),12288(6), 9216(3),6144(4),4608(2)
	7	32	370	1952	6633	14400	18746	24576,12288(6),9216,6144(6), 4608
	7	28	378	1964	6601	14392	18794	20321280,36864,27648,24576(2), 13824,12288(4),9216,6144(3), 4608
	7	24	386	1976	6569	14384	18842	73728,36864(2),24576,12288(7), 6144
	7	20	394	1988	6537	14376	18890	36864,12288(2)
	7	16	402	2000	6505	14368	18938	221184,98304,36864,12288(2)
	7	8	418	2024	6441	14352	19034	49152
	7	0	690	0	13545	0	37050	98304
	6	56	312	1880	6874	14448	18382	2211840,786432,589824,221184, 147456,110592,65536(2), 32768(4),24576(2),18432, 16384(2),12288(2),6144
	6	52	320	1892	6842	14440	18430	3072(3),1536(2)
	6	48	328	1904	6810	14432	18478	196608,131072,110592,49152(2), 27648,18432(2),16384(2),12288, 9216,8192(4),6144(12),4096, 3072(4),1536
	6	44	336	1916	6778	14424	18526	6912,6144,3072(3),1536(7)
	6	40	344	1928	6746	14416	18574	737280,73728,65536(2),49152, 32768(4),24576,18432,16384(3), 12288(5),9216,8192(4),6144(6), 4608,4096,3072(4),1536(4)

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	6	36	352	1940	6714	14408	18622	9216,6912,6144,3072(5),1536(6)
	6	32	360	1952	6682	14400	18670	294912,221184,131072(3),65536, 32768,24576,18432(3),16384(5), 13824,12288,9216(2),8192(3), 6912,6144(11),4096,3072(3), 1536(4)
	6	28	368	1964	6650	14392	18718	18432,6144,3072(3),1536(5)
	6	24	376	1976	6618	14384	18766	26542080,147456,131072(2), 110592,73728,65536,55296,49152, 32768,24576,18432(2),16384(5), 13824,12288,8192(2),6144(8), 3072(2),1536
	6	20	384	1988	6586	14376	18814	18432,6144,3072
	6	16	392	2000	6554	14368	18862	393216,147456,131072(2),65536, 49152,36864,24576,16384,8192, 6144
	6	12	400	2012	6522	14360	18910	9216
	6	8	408	2024	6490	14352	18958	170311680,131072,32768,24576
	6	0	680	0	13594	0	36974	26542080,4423680,786432,196608, 73728
	6	0	424	2048	6426	14336	19054	26542080,4423680,786432
	5	52	310	1892	6891	14440	18354	32768,20480,18432,12288(3), 9216,8192,6144,4608,4096(3), 3072,2048
	5	48	318	1904	6859	14432	18402	8192,4096,2048,1536,1024(3), 768(3)
	5	44	326	1916	6827	14424	18450	4096(3),3072,2048(7),1536(6), 1024,768(4)
	5	40	334	1928	6795	14416	18498	6144,4608(2),4096,3072,2304, 2048(3),1024(5),768(5),384(2)
	5	36	342	1940	6763	14408	18546	32768,6144,4096(5),3072(2), 2048(6),1536(4),1024(2),768(6), 384
	5	32	350	1952	6731	14400	18594	4096(2),3072,2048(6),1536(3),

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32								1024(3),768(5),384
	5	28	358	1964	6699	14392	18642	6144,4608(2),4096,3072,2048(8), 1536(3),1024,768(3),384
	5	24	366	1976	6667	14384	18690	9216,4096(3),3072(2),2048(5), 1536(3),1024,768(2)
	5	20	374	1988	6635	14376	18738	12288,8192,4096,2048(3),1536
	5	16	382	2000	6603	14368	18786	49152,27648,13824,12288,8192, 4608,4096,3072,2048
	5	12	390	2012	6571	14360	18834	10240
	5	8	398	2024	6539	14352	18882	8192
	5	0	670	0	13643	0	36898	147456,82944,61440
	4	48	308	1904	6908	14432	18326	65536,49152,24576,18432,12288, 8192,4096(5),3456,3072,2304, 2048(3),1536,1024(3),512(2)
	4	44	316	1916	6876	14424	18374	3456,2048,1024(2),576,512(3), 384(2),256(4)
	4	40	324	1928	6844	14416	18422	16384(2),12288(2),9216,6912, 6144(2),4096(2),3072,2304(2), 2048(4),1536(3),1152,1024(9), 768(5),512(5),384,256(3),192
	4	36	332	1940	6812	14408	18470	1536,1152,1024(2),512(6), 384(2),256(7),192(2)
	4	32	340	1952	6780	14400	18518	196608(2),32768,24576,18432(2), 16384(2),8192,4096(7),3072(2), 2304,2048(7),1536(3),1152, 1024(6),768(2),512(7),384(3), 256(4),192
	4	28	348	1964	6748	14392	18566	36864,10368,9216,3072,1152, 1024(3),576(2),512(7),384, 256(4),192
	4	24	356	1976	6716	14384	18614	184320,110592,18432,16384,8192, 6144,4096(4),3072,2304,2048(5), 1536(2),1152,1024(6),768(3), 576,512(5),256

$2k$	Weight Distributions							Automorphism Group Sizes
	4	6	8	10	12	14	16	
32	4	20	364	1988	6684	14376	18662	3456,2048,1152,1024(3),512(3), 256
	4	16	372	2000	6652	14368	18710	589824,368640,98304,55296, 32768,12288(3),10368,8192(2), 2304,2048(2),1024(2),768
	4	12	380	2012	6620	14360	18758	12288,2048
	4	8	388	2024	6588	14352	18806	3870720,98304,55296,18432,4096
	4	0	660	0	13692	0	36822	786432,110592,16384
	4	0	404	2048	6524	14336	18902	262144
	3	44	306	1916	6925	14424	18298	6144,2304(2),1536,1152(2), 1024(3),768,256(6),128
	3	40	314	1928	6893	14416	18346	576,512,384,288,256(2),128(5), 64(4)
	3	36	322	1940	6861	14408	18394	3072,1536,512(3),384(4),288(2), 256(10),128(10),64(2)
	3	32	330	1952	6829	14400	18442	5184,768,576,512(2),384,288(2), 256(4),192,128(10),64(4)
	3	28	338	1964	6797	14392	18490	4608,1024(3),768(2),512,384(2), 256(7),192,128(8),64(3)
	3	24	346	1976	6765	14384	18538	10368,5184,1728,1152,768, 512(3),384(3),288,256(6), 128(7),64
	3	20	354	1988	6733	14376	18586	13824,1728,1024,768,256(4), 128(4)
	3	16	362	2000	6701	14368	18634	2304(2),1536,1024,768(2),512, 256(2)
	3	12	370	2012	6669	14360	18682	138240,10368,384
	3	8	378	2024	6637	14352	18730	3072(2)
	3	0	650	0	13741	0	36746	269568,9216
	2	40	304	1928	6942	14416	18270	49152,16384,8192,6144,4608, 1536,1024(2),512(3),384,256(7), 192(2),128(2),64(3),32
	2	36	312	1940	6910	14408	18318	192,128,64(5),32(7),16
	2	32	320	1952	6878	14400	18366	4096,2048(2),1024(2),512(2),

	Weight Distributions							
$2k$	4	6	8	10	12	14	16	Automorphism Group Sizes
32								384,256(8),128(9),96,64(7), 32(5),16(2)
	2	28	328	1964	6846	14392	18414	384,256,192(3),128,96(3),64(6), 32(6),16(2)
	2	24	336	1976	6814	14384	18462	49152,8192,4096(2),3072(2), 2304,2048(2),1024(2),768(2), 512(2),256(7),192,128(4),64(7), 32(3),16
	2	20	344	1988	6782	14376	18510	768,576,256,192,128,64(4), 32(3)
	2	16	352	2000	6750	14368	18558	73728,24576,18432,9216,8192, 3072(3),2048(2),1152,768,512, 384,256(4),192,128,64
	2	12	360	2012	6718	14360	18606	768,192
	2	8	368	2024	6686	14352	18654	122880,18432,4096,3072,1024
	2	0	640	0	13790	0	36670	294912,18432,9216
	2	0	384	2048	6622	14336	18750	98304
	1	36	302	1940	6959	14408	18242	512,384,256,192,128(2),64,48, 32(3),16(4)
	1	32	310	1952	6927	14400	18290	192,96,64,48,32,16(4),8(5)
	1	28	318	1964	6895	14392	18338	768,384,192(2),144,128,96,72, 64,48,32(5),24,16(5),8(4)
	1	24	326	1976	6863	14384	18386	192,96,64(3),48,32(3),16(5), 8(3)
	1	20	334	1988	6831	14376	18434	768,512,192(2),128,96,64,48(3), 32(2),16(3),8
	1	16	342	2000	6799	14368	18482	576,288,256,192,144,64(2), 48(3),32
	1	12	350	2012	6767	14360	18530	1536,864,192,64
	1	8	358	2024	6735	14352	18578	384
	1	0	630	0	13839	0	36594	5376
	0	32	300	1952	6976	14400	18214	368640,23040,2048,1536,1440, 192(3),120,64(2),32(2),24,20, 16,12(2),4

	Weight Distributions							
$2k$	4	6	8	10	12	14	16	Automorphism Group Sizes
32	0	28	308	1964	6944	14392	18262	1344,48,16(2),12,8,4(2),2
	0	24	316	1976	6912	14384	18310	18432,6144,1152,512,384,192, 96(2),32(3),24(2),12,8(4),4(2)
	0	20	324	1988	6880	14376	18358	96,48,16(2),12,8,6,4
	0	16	332	2000	6848	14368	18406	6144,1024(2),768,128,64,48, 32(2),16,8
	0	12	340	2012	6816	14360	18454	144(2),64,60
	0	8	348	2024	6784	14352	18502	6144,1536,384
	0	0	620	0	13888	0	36518	319979520,10321920,86016,23040, 14880
	0	0	364	2048	6720	14336	18598	2064384,294912,86016

Bibliography

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

[2] V. Pless, A classification of self-orthogonal codes over $GF(2)$, *Discrete Math.* 3 (1972), 209–246.

[3] V.Pless and N. J. A. Sloane, On the classification and enumeration of self-dual codes. *J. Combin. Theory Ser. A* 18 (1975), 313–335.

[4] J. H. Conway and V. Pless, On the enumeration of self-dual codes, *J. Combin. Theory Ser. A* 28 (1980), 26–53.

[5] J. H. Conway, V. Pless, and N. J. A. Sloane, The binary self-dual codes of length up to 32: a revised enumeration, *J. Combin. Theory Ser. A* 60 (1992) 183–195.

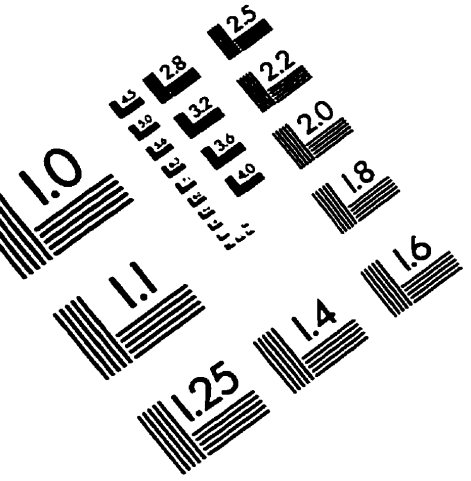
[6] J. H. Conway and N. J. A. Sloane, A new upper bound on the minimal distance of self-dual codes, *IEEE Trans. Inform. Theory*, vol. 36 (Nov. 1990) 1319–1333.

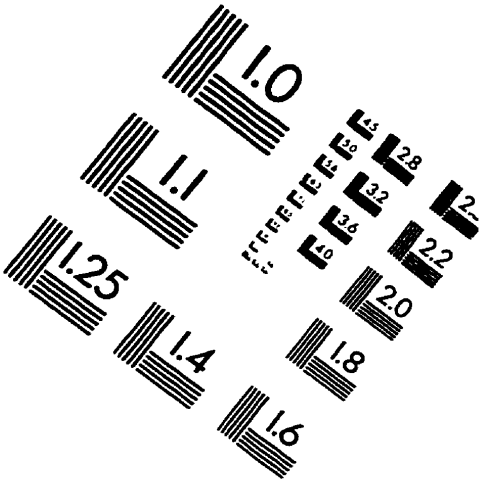
[7] H. Koch, On self-dual, doubly even codes of length 32, *J. Combin. Theory Ser. A* 51 (1989) 63–76.

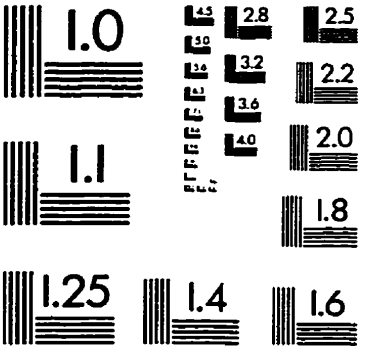
[8] H. Oral, Self-orthogonal codes with distance four are not rigid, *Discrete Mathematics* 104 (1992) 263–271.

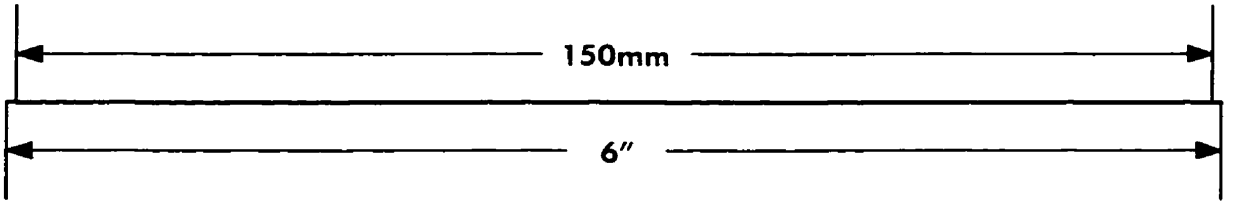
[9] C. W. H. Lam and L. Thiel, Backtrack search with isomorph rejection and consistency check, *J. Symbolic Computation* 7 (1989) 473–485.

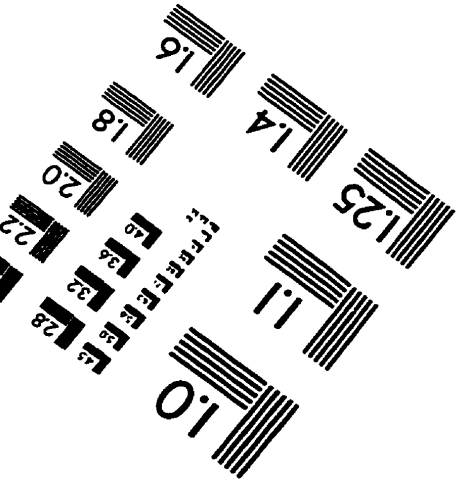
IMAGE EVALUATION TEST TARGET (QA-3)

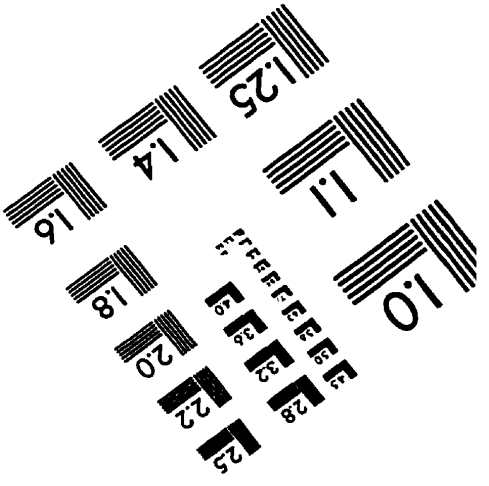












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved