The Relationship between $(16, 6, 3)$-Balanced Incomplete Block Designs and $(25, 12)$ Self-Orthogonal Codes

by

Navid Nasr Esfahani

A thesis 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

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
July 2014

Thesis advisor

G. H. John van Rees

Author

Navid Nasr Esfahani

The Relationship between $(16, 6, 3)$-Balanced Incomplete Block Designs and $(25, 12)$ Self-Orthogonal Codes

Abstract

Balanced Incomplete Block Designs and Binary Linear Codes are two combinatorial designs. Due to the vast application of codes in communication the field of coding theory progressed more rapidly than many other fields of combinatorial designs. On the other hand, Block Designs are applicable in statistics and designing experiments in different fields, such as biology, medicine, and agriculture. Finding the relationship between instances of these two designs can be useful in constructing instances of one from the other. Applying the properties of codes to corresponding instances of Balanced Incomplete Block Designs has been used previously to show the non-existence of some designs. In this research the relationship between $(16, 6, 3)$-designs and $(25, 12)$ codes was determined.

Contents

List of Figures

List of Tables

Acknowledgments

First of all, I would like to show appreciation to my supervisor, Professor van Rees, for all his guidance and support. I would also like to thank my advisory committee members, Professor Bate and Professor Li for their helpful comments, and Professor Spence and Professor Bouykukliev for kindly providing me with the designs and codes. I need to extend thanks to Grigory Shamov and Gilbert Detillieux for their generous help in using computing resources. And last but not least, I want to express my gratitude to my family for their loving support, and also to my dear friends, who gave me their company, happiness, and support from the first day to the present.

To my siblings, Farid and Parinaz

NOTATION

$\mathbb{N}$	$\{0, 1, 2, 3, \dots\}$
$\mathbb{Z}$	$\{\dots, -2, -1, 0, 1, 2, \dots\}$
$\mathbb{Z}^+$	$\{1, 2, 3, 4, \dots\}$
$a \;\; mod \;\; b$	The remainder of a divided by b
$\oplus$	XOR operator
$a[i]$ for a vector or an array a	the i^{th} element of a
$(a_0 a_1 \dots a_n)$	A vector or an array with the elements a_0 to a_n
$[AB]$	A matrix with parts A and B
I_k	The identity matrix of rank k
$\|S\|$ for a set S	The cardinality of S
$\|M\|$ for a matrix M	The determinant of M
M^t for a matrix M	The transpose of M
$a \in S$	a is a member in set S
$T \subset S$	T is a subset of S, and $\|T\| < \|S\|$
$T \subseteq S$	T is a subset of S, and $\|T\| \leq \|S\|$
$T \cap S$	The intersection of set T with set S
$T \cup S$	The union of set T and set S

Chapter 1

Background

This chapter includes the definitions and notations that will be used in the proceeding chapters. However, it only covers the basic knowledge needed in the following chapters. For more detailed information and explanations, there are books available on block designs such as "Combinatorial Designs"[24], "Combinatorial Designs: Construction and Analysis"[20] and on coding theory such as "An Introduction to Error Correcting Codes with Applications"[23]. All the definitions in this chapter are borrowed from the above-mentioned books [23; 20].

1.1 Block Designs

Definition 1. *Let $v, b, k, r,$ and λ be positive integers. For any v-set V and set collection $B = \{B_i \mid B_i \subset V, |B_i| = k, i = 1, 2, \ldots, b\}$, the pair (V, B) is a Balanced Incomplete Block Design (BIBD), if and only if*

$$t \in V, |\{B_i \mid t \in B_i\}| = r$$

and

$$\forall t_1, t_2 \in V, t_1 \neq t_2, |\{B_i \mid t_1 \in B_i, t_2 \in B_i\}| = \lambda.$$

A design with these parameters is called a (v, b, r, k, λ)*-design or* (v, k, λ)*-design in short form.*

Variety or *treatment* are common terms for the elements of V. The elements of B $(B_i, 1 \leq i \leq b)$ are called blocks.

Example 1.1. *Let* $V = \{a, b, c, d\}$ *and* $B_1 = \{a, b\}, B_2 = \{a, c\}, B_3 = \{a, d\}, B_4 = \{b, c\}, B_5 = \{b, d\}$*,and* $B_6 = \{c, d\}$*. If* $B = \{B_1, B_2, B_3, B_4, B_5, B_6\}$*, then* (V, B) *is a* $(4, 6, 3, 2, 1)$*-design.*

Counting the total occurrences of all varieties, first by considering the varieties in each block and doing it once again by counting the occurrences of each variety in the design will result in the following proposition.

Proposition 1.1. *[20, p. 5] In a* (v, b, r, k, λ)*-design* $rv = bk$*.*

Choosing a variety t_1, then counting the total number of pairs that contain t_1, once by calculating the number of pairs that t_1 forms in different blocks, and calculating this value again using the definition of λ, leads to the proposition below.

Proposition 1.2. *[20, p. 4] In a* (v, b, r, k, λ)*-design* $r(k-1) = \lambda(v-1)$*.*

Although the results from Propositions 1.1 and 1.2 can be used to show that no $(6, 3, 3)$ exists ($r = \frac{3\times 5}{2} = 7.5$ which is impossible), they do not guarantee that there is a design for parameters satisfying these conditions.

Definition 2. *Matrix* $M_{v\times b} = \{m_{ij}\}$ *is the incidence matrix of a BIBD if and only if*

$$m_{ij} = \begin{cases} 1 & t_i \in B_j \\ 0 & t_i \notin B_j. \end{cases}$$

If M is the incidence matrix of a (v, b, r, k, λ)-design, then there are exactly k 1s in each column of M and exactly r 1s in each row of M. Also, for each pair of rows r_i and $r_j, i \neq j$ in M, $r_i \cdot r_j = \lambda$.

Example 1.2. *The matrix M is an incidence matrix of the $(4, 2, 1)$-design in Example 1.1.*

$$M = \begin{bmatrix} 1\,1\,1\,0\,0\,0 \\ 1\,0\,0\,1\,1\,0 \\ 0\,1\,0\,1\,0\,1 \\ 0\,0\,1\,0\,1\,1 \end{bmatrix}$$

As mentioned above, each row of M has three 1s and there are two 1s in each column of M. Also, for each distinct pair of rows, their dot product is equal to 1.

Proposition 1.3. *[21] In a (v, b, r, k, λ)-design, $b \geq v$.*

The above proposition can be proven by contradiction. Suppose $b < v$. Add $v - b$ columns of 0 to M to generate matrix M_+. Since $MM^T = M_+M_+^t$, $|MM^t|$ should be equal to $|M_+M_+^t| = |M_+||M_+^t|$. However, calculating the determinant of MM^t will result in $rk(r - \lambda)^{v-1} \neq 0$. On the other hand, due to the added zero columns $|M_+| = 0$. As the calculation of determinant results in both a zero value and a non-zero value, the assumption that $b < v$ is wrong. Hence, in any balanced incomplete block design, $b \geq v$.

Definition 3. *Two block designs (V, B) and $(\hat{V}, \hat{B})$ are isomorphic if and only if $|V| = |\hat{V}|, |B| = |\hat{B}|$ and there exist bijections $\alpha : V \rightarrow \hat{V}$ and $\beta : B \rightarrow \hat{B}$ such that for any variety t in V and any block $B_i \in B$ if t is in B_i then $\alpha(t)$ is in $\beta(B_i)$ and vice versa.*

In other words, two designs are isomorphic if and only if 1) the labeling of the varieties of one is a relabeling of the varieties of the other, 2) the labeling of the blocks of one design is a relabeling of the blocks of the other design, and 3) corresponding varieties appear together in corresponding blocks.

Example 1.3. *Let* $\hat{V} = \{a, c, d, g\}$ *and* $\hat{B} = \{\{a, c\}, \{a, d\}, \{a, g\}, \{c, d\}, \{c, g\}, \{d, g\}\}$. *Define* $\alpha : V \to \hat{V}$ *such that* $\alpha(a) = a, \alpha(b) = c, \alpha(c) = d$, *and* $\alpha(d) = g$ *and define* $\beta : B \to \hat{B}$ *such that* $\beta((a, b)) = (a, c), \beta((a, c)) = (a, d), \beta((a, d)) = (a, g), \beta((b, c)) = (c, d), \beta((b, d)) = (c, g)$, *and* $\beta((c, d)) = (d, g)$. *From the definition,* (V, B) *and* $(\hat{V}, \hat{B})$ *are two isomorphic designs.*

Definition 4. *A* (v, k, λ)*-design is symmetric if and only if* $v = b$.

Example 1.4. *Let* $V = \{a, b, c, d, e, f, g\}$ *and* $B = \{\{a, b, c\}, \{a, d, e\}, \{a, f, g\}, \{b, d, g\}, \{b, e, f\}\{c, d, f\}, \{c, e, g\}\}$. *Since* $|V| = |B|$, (V, B) *is a* $(7, 3, 1)$*-SBIBD.*

It does not necessarily mean that a symmetric BIBD (SBIBD) has a symmetric incidence matrix. From Proposition 1.1, it can be concluded that in an SBIBD, $r = k$.

Theorem 1.4. *(Bruck-Ryser-Chowla Theorem [20, p. 30,32]) For any symmetric* (v, k, λ)*-design, if* v *is even, then* $k - \lambda$ *is a perfect square, otherwise (if* v *is odd) there exist integers* x, y *and* z, *where at least one of them is non-zero, such that*

$$x^2 = (k - \lambda)y^2 + (-1)^{\frac{(v-1)}{2}} \lambda z^2$$

Sometimes it is possible to generate new designs from the known designs. Symmetric designs are a type of design, from which new designs can be obtained.

Definition 5. *For any* (v, v, k, k, λ) *SBIBD, where* $k \geq \lambda + 2$, *one can remove any arbitrary block* $B_0 \in B$ *from the design, as well as all varieties in that block, and construct a new* $(v - k, v - 1, k, k - \lambda, \lambda)$ *design* $(V \setminus B_0, \{B_i \setminus B_0 \mid i = 1, 2, \ldots b\})$.

The resultant design is called a residual design. Any design with the parameters of a residual design is called a quasi-residual design.

Example 1.5. *Consider the* $(7,3,1)$*-SBIBD from Example 1.1. If* $\{b,e,f\}$ *is removed from the blocks, and omit* $b, e,$ *and* f *from all the remaining blocks and* V*, the result* $(\hat{V},\hat{B})$ *with* $\hat{V} = \{a,c,d,g\}$ *and* $\hat{B} = \{\{a,c\},\{a,d\},\{a,g\},\{c,d\},\{c,g\},\{d,g\}\}$ *will be a residual* $(4,2,1)$*-design. It can be seen from Example 1.3 that the resultant* $(4,2,1)$*-design is isomorphic to the* $(4,2,1)$*-design from Example 1.1.*

More examples and properties, as well as a list of designs, up to certain values of parameters, that contains the number of known non-isomorphic copies of each design is provided in the Handbook of Combinatorial Designs [11].

1.2 Codes

If $V_q(n)$ represents the set of all n-tuples over an alphabet of size q, a $[n,M]$ *block code* (or, more simply, a code) is an M-subset of $V_q(n)$. In this research only binary n-tuples ($V_2(n)$) will be used. Any arbitrary n-tuple over the alphabet $\Sigma = \{0,1\}$ is a *word* and if a word is in the code, it is called a *codeword*. Each codeword is said to support the coordinates (or columns), where it has a 1.

Definition 6. *For any two words, the Hamming distance between them is equal to the number of coordinates in which they differ. The Hamming distance of codewords* $a, b \in \{0,1\}^n$ *is represented by* $d(a,b)$*. In other words:*

$$d(a,b) = \sum_{i=1}^{n} (a[i] + b[i] \ mod\ 2)$$

Example 1.6. *Consider the following words over $V_2(7)$:*

$$\vec{a} = (0000000), \vec{b} = (1000111), \vec{c} = (1010101), \vec{d} = (1111110)$$

The the distances between these words will be:

$$d(\vec{a}, \vec{a}) = 0, d(\vec{a}, \vec{b}) = 4, d(\vec{a}, \vec{c}) = 4, d(\vec{a}, \vec{d}) = 6$$

$$d(\vec{b}, \vec{c}) = 2, d(\vec{b}, \vec{d}) = 4, d(\vec{c}, \vec{d}) = 4.$$

Since Hamming distance is the only distance being used in this research, distance will mean Hamming distance.

Definition 7. *A code C has distance d if and only if the minimum distance between any distinct pair of codewords is d.*

Example 1.7. *Consider a $[7, 4]$ block code with all the words from Example 1.6. The distance of the resultant code will be 2, as the minimum distance between distinct pairs of codewords is $d(\vec{b}, \vec{c}) = 2$.*

Definition 8. *The weight of a word is the number of coordinates in it which contain a non-zero value.*

Example 1.8. *Among the words from Example 1.6, the weight of $\vec{a}$ is equal to 0, $\vec{b}$ and $\vec{c}$ are of weight 4, and $\vec{d}$ has weight 6.*

Considering $\vec{0}_n$ to be the n-tuple with all zeros, the weight of a word w is equal to $d(w, \vec{0}_n)$.

Definition 9. *For any code C, its Weight Distribution is an array W of length $n+1$, where, for all $i \in \{0, 1, \ldots, n\}$, $W[i]$ is equal to the number of codewords of weight i.*

Example 1.9. *Consider the block code from Example 1.7. The weight distribution of that code is:*

$$W[0] = 1, W[1] = 0, W[2] = 0, W[3] = 0, W[4] = 2, W[5] = 0, W[6] = 1, W[7] = 0.$$

A code is *even* if all its codewords have even weight. If all codewords in a code C have weights of the form $4k, k \in \mathbb{N}$, then C is *doubly even.* That is, all codewords have weights which are multiples of 4.

Definition 10. *For positive integers n and k with $n \geq k$, a code C is a Linear Binary (n, k)-Code if and only if it forms a k-dimensional subspace of $V_2(n)$.*

So C contains 2^k codewords and for each two codewords $u, v \in C$ their sum $u + v \in C$.

Example 1.10. *Let C_1 be a $[7, 8]$ block code with following codewords.*

$$(0000000), (1000111), (1010101), (1111110), (0010010), (0111001), (0101011), (1101100)$$

Since for any pair of codewords in C_1, their linear combination is a member of C_1, C_1 is a $(7, 3)$ linear code.

A matrix M generates C if its rows span C, i.e. any linear combination of rows of M is a codeword in C and any codeword in C is a linear combination of rows of M.

Proposition 1.5. *[23, p. 49] For any linear binary code C, its distance is equal to the smallest weight of any non-zero codeword in C.*

For each length n and distance d, there is an upper bound on the number of codewords that a code (not necessarily linear) of length n and distance d can have.

This number is represented by $A(n, d)$. The upper bound for the number of codewords of length n that form a linear binary code with distance d is represented by $B(n, d)$.

Definition 11. *A Generator Matrix of an (n, k) linear binary code C is a matrix of n columns and k linearly independent rows that generate C.*

Example 1.11. *Let*

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

M_1 is a generator matrix for the linear code C_2 from Example 1.10. However M_1 is not the only generator matrix for C_1. For example the matrix below is another generator matrix for C_1:

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

Definition 12. *Two words $a, b \in V_2(n)$ are orthogonal if and only if*

$$a \cdot b = \sum_{i=1}^{n} a[i]b[i] = 0(mod\ 2)$$

Example 1.12. *Consider the words from Example 1.6. Since $\vec{a} \cdot \vec{b} = \sum_{i=1}^{7} \vec{a}[i]\vec{b}[i] = 0+0+0+0+0+0+0(mod\ 2) = 0(mod\ 2)$, $\vec{a}$ and $\vec{b}$ are orthogonal. But $\vec{c}$ and $\vec{d}$ are not orthogonal, because $\vec{c} \cdot \vec{d} = \sum_{i=1}^{7} \vec{c}[i]\vec{d}[i] = 1+0+1+0+1+0+0(mod\ 2) = 1(mod\ 2)$.*

Definition 13. *The orthogonal complement of a linear binary code C, represented by $C^{\perp}$, is the subset of $V_2(n)$ that contains all the codewords which are orthogonal to every codeword in of C.*

$$C^{\perp} = \{a \in V_2(n) \mid \forall c \in C, a \cdot c = 0\}$$

Example 1.13. *Let C_2 be a $(7,3)$ linear binary code, with the following generator matrix.*

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Then $C_2^{\perp} = \{(0000000), (1000111), (0101011), (0011101), (1101100), (1011010), (011\,0110), (1110001), (1111111), (0111000), (1010100), (1100010), (0010011), (0100101), (0001110), (0001110)\}$ *is the orthogonal complement for C_2.*

Definition 14. *A linear binary code C is self-orthogonal if and only if each codeword in C is orthogonal to all the other codewords in C ($C \subseteq C^{\perp}$). And C is self-dual if and only if C is self-orthogonal, and for any word w that is orthogonal to every c in C, $w \in C$ ($C = C^{\perp}$).*

Example 1.14. *As can be seen in Example 1.13,* $C_2 = \{(0000000), (1000111), (01010\,11), (0011101), (1101100), (1011010), (0110110), (1110001)\} \subset C_2^{\perp}$. *Therefore, C_2 is self-orthogonal.*

Note that if a linear binary code C is self-orthogonal, it has to be even.

Definition 15. *Two codes, C_1 and C_2, are equivalent if and only if the coordinates of C_2 are a permutation of the coordinates of C_1.*

A BIBD is embedded in a code, if the rows of the incidence matrix of the design are codewords of the code. Since the only difference between two equivalent codes is that the coordinates of one of them are a reordering of the coordinates of the other one, for any BIBD, which is embedded in one code, there will be another BIBD, isomorphic to the first BIBD, embedded in the second code. In other words, an equivalent code does not contain a new set of embedded designs.

Example 1.15. *Let*

$$M_3 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

be the generator matrix for C_3. Then C_2 and C_3 are equivalent because columns of M_3 are a permutation of columns of M_2.

In most cases, a linear (n, k) code can have different generator matrices, which generate equivalent codes. For each linear binary code, C, there is at least one code in the equivalence class of C that has a generator matrix of the form $[I_k \; A_{(n-k)\times k}]$. It is standard for a code to be represented by a generator matrix of the form $[I_k \; A_{(n-k)\times k}]$ that generates either the same code or an equivalent code.

Proposition 1.6. *If two codes are equivalent, they have the same weight distribution. But having the same distribution does not imply that two codes are equivalent.*

Proof. Any two equivalent codes C and C' only differ in the ordering of their coordinates. Since the number of ones in each codeword remains constant after any reordering of bits, corresponding codewords in two equivalent codes have the same weight. Therefore, equivalent codes have the same weight distribution. In order to check the second part, one can consider the $(16, 8)$ self-dual codes generated by the following generator matrices:

Table 1.1 shows the $A_8 \oplus A_8$ code, constructed based on the A_8 matrix from Pless [17], and Table 1.2, copied from the E_{16} matrix in Pless [17], are generator matrices for two inequivalent $(16, 8)$ codes. However, both codes share the following weight distribution array.

$$W[0] = W[16] = 1, W[4] = W[12] = 28, W[8] = 198, 0 < k < 4, i \neq 4k : W[i] = 0$$

But the codes are not equivalent, as they have different automorphism groups [17]. $\square$

$$A_8 \oplus A_8 = \begin{bmatrix} 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1 \end{bmatrix}$$

Table 1.1: $A_8 \oplus A_8$: A generator matrix of a $(16, 8)$ code

$$E_{16} = \begin{bmatrix} 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \end{bmatrix}$$

Table 1.2: E_{16}: A generator matrix of a $(16, 8)$ code, [17]

More information about linear codes, including self-orthogonal codes, can be found in Handbook of Coding Theory [18], Theory of Error Correcting Codes [16]. Pless [17] has also enumerated the binary self-orthogonal codes up to length 20.

Chapter 2

Introduction

Wallis [24, p. 1] informally describes a Combinatorial Design as the result of selecting subsets from a finite set, such that these subsets satisfy some pre-defined conditions. Balanced Incomplete Block Designs (BIBDs) and Binary Linear Codes are two type of such designs that will be considered in this thesis.

Fisher and Yates [13, p. 25] discussed the application of BIBDs in experimental designs. In their book [13, p. 25], they suggested BIBDs as the appropriate experimental design for experiments with specific circumstances. For instance, when the experiment is about pairs of individuals (e.g. mono-zygotic twins), or when the varieties of the experiment are examined in different environments or under various conditions.

Bhattacharya [6] found the first $(16, 6, 3)$-design in 1944. Later, in 1982, Denniston [12] enumerated all the symmetric $(25, 16, 3)$-designs. Using the $(25, 16, 3)$-designs, van Rees generated all the 1281 residual $(16, 6, 3)$-designs [22]. Finally, Spence generated all the $(16, 6, 3)$-designs [11].

The goal of my research is to find the relationship between different classes of $(16, 6, 3)$-designs and $(25, 12)$ binary linear codes. Bilous *et al.* [7], based on previous works such as those by Hall *et al.* [14] and Bate *et al.* [5], used the relationship between other instances of BIBDs and binary linear codes and the properties of those binary linear codes to prove the non-existence of a $(24, 8, 4)$-design.

2.1 Related Work

The $(22, 8, 4)$-design had been the smallest undecided case of BIBDs for about seven decades, before Bilous *et al.* [7] proved the non-existence of this design. Hence, a large amount of work has been done on searching for that design. Since the non-existence was proved through the relationship between possible $(22, 8, 4)$-designs and $(33, 16)$ codes, the majority of the contents of this section will be about $(22, 8, 4)$-designs.

Hall *et al.* [14] discussed the relationship between binary $(33, 16)$ codes and the incidence matrix of a possible $(22, 8, 4)$-design. They knew that the incidence matrix of the design, if it existed, had to be embedded in some $(33, 16)$ codes. This point of view enabled them to search for corresponding codes instead of the designs. For instance, they showed that a code of length 33 should be doubly even and self-orthogonal to contain the incidence matrix of the desired design. They also found other necessary conditions, which cause the elimination of many possible codes. In subsequent works, Bate *et al.* [5] and Bilous and van Rees [9] eliminated more classes of $(33, 16)$ codes.

Bilous and van Rees [8] extended the idea of searching codes for incidence matrices of designs. They proposed searching for a part of an incidence matrix of a $(22, 8, 4)$-design in the $(32, 16)$ codes, which bounds the search domain to doubly even self-dual codes.

In the final step, Bilous *et al.* [7] used computers to search the remaining codes.

Since the search did not find the desired incidence matrix in any code, there is no $(22, 8, 4)$-design. In the future work section, they suggested searching for this kind of relationship in $(16, 6, 3)$-designs, which is the main goal of this research. The $(16, 6, 3)$-design was suggested since it is similar to the $(22, 8, 4)$-design in being a member of the $(6\lambda - 2, 2\lambda, \lambda)$ family of residual designs for $\lambda = 3$ and $\lambda = 4$, respectively; and $(16, 6, 3)$-designs exist, so it is guaranteed that there are relationships to be found. To the best of our knowledge, the search to find the $(22, 8, 4)$-designs, punished on the relationship between codes and designs in 2007, was the latest research related to finding the relationship between $(16, 6, 3)$-designs and $(25, 12)$ self-orthogonal codes.

In order to determine mappings between designs and codes, it is helpful if either the set of all codes or the set of all designs have already been constructed. It would then be possible to consider the existing set as the domain and then generate the range for the mapping. For this case, Spence [19] has generated all $(16, 6, 3)$-designs, up to isomorphism. Having the designs makes it possible to produce the corresponding codes for each BIBD. Therefore, the task is to generate the codes, spanned over incident matrices, and then categorize them into equivalence classes.

As mentioned, $(16, 6, 3)$-designs and $(22, 8, 4)$-designs are related as they are from the family of $(6\lambda - 2, 2\lambda, \lambda)$ residual designs. All the designs in the Residual Design column in Table 2.1 are also members of this class. So, λ is a possible parameter on which to generalize the results.

Table 2.1 presents our knowledge about the first 13 members of this family.

λ	Residual Design	Existence	SBIBD	Existence	Code
1	$(4, 2, 1)$	$\surd$ [†]	$(7, 3, 1)$	$\surd$[11, p.36]	$(7, 3)$
2	$(10, 4, 2)$	$\surd$[11, p.36]	$(16, 6, 2)$	$\surd$[11, p.37]	$(15, 7)$
3	$(16, 6, 3)$	$\surd$ [11, p.37]	$(25, 9, 3)$	$\surd$[11, p.37]	$(25, 12)$
4	$(22, 8, 4)$	$\times$ [11, p.37]	$(34, 12, 4)$	$\times$ B.R.C.[††]	$(33, 16)$
5	$(28, 10, 5)$	$\surd$[11, p.38]	$(43, 15, 5)$	$\times$ B.R.C.	$(43, 21)$
6	$(34, 12, 6)$	$\surd$[11, p.40]	$(52, 18, 6)$	$\times$ B.R.C.	$(51, 25)$
7	$(40, 14, 7)$	?[11, p.41]	$(61, 21, 7)$	$\times$ B.R.C.	$(61, 30)$
8	$(46, 16, 8)$	$\surd$[11, p.43]	$(70, 24, 8)$	$\surd$[11, p.43]	$(69, 34)$
9	$(52, 18, 9)$	$\surd$[11, p.45]	$(79, 27, 9)$	$\surd$[11, p.45]	$(79, 39)$
10	$(58, 20, 10)$	?[11, p.48]	$(88, 30, 10)$	$\times$ B.R.C.	$(87, 43)$
11	$(64, 22, 11)$	?[11, p.50]	$(97, 33, 11)$	?[11, p.50]	$(97, 58)$
12	$(70, 24, 12)$	?[11, p.53]	$(106, 36, 12)$	$\times$ B.R.C.	$(105, 52)$
13	$(76, 26, 13)$	?[11, p.55]	$(115, 39, 13)$	$\times$ B.R.C.	$(115, 57)$

Table 2.1: $(6\lambda - 2, 2\lambda, \lambda)$-Residual Designs and their corresponding SBIBDs
Mathon and Rosa [11] keep track of the number of non-isomorphic BIBDs. The table above contains a selection of the rows of their table.
[†] $V = \{a, b, c, d\}, B = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$
[††] B.R.C. stands for Bruck-Ryser-Chowla theorem.

2.2 Problem Description

The first objective of this research is to completely determine the relationship between the list of inequivalent codes and the non-isomorphic classes of designs generated by Spence [19]. A code C is related to a design D if and only if C contains the rows of the incidence matrix of D among its codewords. To gain more information, it is helpful to have all the $(25, 12)$ self-orthogonal codes. Therefore, generating all $(25, 12)$ self-orthogonal codes is the second goal of this thesis.

Chapter 3

Classification of Designs into Codes Generated by Incidence Matrices

In the first step of finding the relationship between $(16, 6, 3)$ balanced incomplete block designs and $(25, 12)$ self-orthogonal codes, designs were classified based on the binary codes with the smallest dimension in which they are embedded. In this process, there was no need to generate the designs as Spence [19] kindly provided us with his results of generating $(16, 6, 3)-$designs, the number of which were recorded in the Handbook of Combinatorial Designs [11]. In total, there are 18920 different designs represented by their incidence matrices. In this chapter, the procedure of generating codes that are spanned by the incidence matrices of designs will be discussed. The discussion will be followed by explanation of the process of finding equivalent codes and then classifying designs based on their codes.

As mentioned in the previous chapters, each codeword in a self-orthogonal code should have an even weight. In addition to that, each pair of codewords should share an even number of coordinates where they both have the value 1. Since the desirable mapping is from designs with $\lambda = 3$ and $r = 9$, which have 24 columns, to self-orthogonal even binary codes, which have 25 columns, a column of 1s was added to the incidence matrix of each design. As a result, each new incidence matrix spans a self-orthogonal even code. After adding the column of 1s to the incidence matrix of each design, a basis of the new matrix was calculated through multiple row reduction operations. Detailed information can be found in any linear algebra book, such as Elementary Linear Algebra, by Anton [4]. The results of the computation were generator matrices for the codes. According to the results, the ranks of these incidence matrices are either 10, 11, or 12. The rank of an incidence matrix is equal to the dimension of the code that is generated by the design. Table 3.1 shows the number of designs with different ranks of their incidence matrices.

Incidence Matrix Rank	Number of Designs
10	6
11	245
12	18669

Table 3.1: Distribution of Designs with respect to the Rank of their Incidence Matrices

Note that a design with an incidence matrix of rank r can be embedded in codes of dimension d if and only if $d \geq r$. In this chapter only the case where $d = r$ will be considered. Other cases will be explored in Chapter 5, after the $(25, 12)$ self-orthogonal codes are studied in Chapter 4.

In the next step, a program was developed to calculate the weight distribution of each code by generating all the codewords of the code, and classified codes based on their weight distribution. This primary classification revealed more information about the distribution of designs in the codes. Having the codes classified by their weight distribution also made it faster and easier to find the final classification. To calculate the weight distribution of each code, all the codewords of that code were generated and the weight of each codeword was computed. Table 3.2 contains the information about weight distributions and the number of designs embedded in codes with that weight distribution.

It can be seen, from Table 3.2, that there is no code with any codewords of weight 2 or weight 24. This can be explained by the definition of the design and the self-orthogonality of the code.

Consider the weight 24 case. If there is a codeword c of weight 24 in a code C, then any of the other codewords in C cannot have any 1 in the coordinate that is not supported by c. Because the code is self-orthogonal and all the codewords are even, and all the other codewords should have an even number of 1s in coordinates supported by c. This implies that there are an even number of 1s in coordinates not supported by c. Since there is only one such coordinate, this number should be zero. Therefore, that coordinate will be zero in all the codewords, which means that there is an empty block, which contradicts the definition of a BIBD. Hence, there cannot be any codeword of weight 24 in a $(25, 12)$ self-orthogonal code with embedded BIBDs.

Weight Distribution (only even weights)														# of Des.	Dim.
No.	0	2	4	6	8	10	12	14	16	18	20	22	24		
1	1	0	1	4	82	164	346	300	77	44	5	0	0	1	10
2	1	0	3	3	78	166	346	300	81	42	3	1	0	2	10
3	1	0	0	3	90	166	328	300	93	42	0	1	0	3	10
4	1	0	1	9	182	324	654	606	185	84	1	1	0	4	11
5	1	0	1	6	180	334	660	594	179	90	3	0	0	8	11
6	1	0	2	9	178	324	660	606	181	84	2	1	0	8	11
7	1	0	1	11	180	318	660	612	179	82	3	1	0	8	11

Weight Distribution (only even weights)														# of Des.	Dim.
No.	0	2	4	6	8	10	12	14	16	18	20	22	24		
8	1	0	3	9	174	324	666	606	177	84	3	1	0	11	11
9	1	0	2	6	176	334	666	594	175	90	4	0	0	12	11
10	1	0	0	8	182	328	660	600	177	88	4	0	0	12	11
11	1	0	2	11	176	318	666	612	175	82	4	1	0	12	11
12	1	0	7	8	154	328	702	600	149	88	11	0	0	12	11
13	1	0	2	8	174	328	672	600	169	88	6	0	0	16	11
14	1	0	4	6	168	334	678	594	167	90	6	0	0	16	11
15	1	0	4	11	168	318	678	612	167	82	6	1	0	16	11
16	1	0	5	9	166	324	678	606	169	84	5	1	0	20	11
17	1	0	1	8	178	328	666	600	173	88	5	0	0	26	11
18	1	0	3	8	170	328	678	600	165	88	7	0	0	64	11
19	1	0	39	36	234	592	1486	1272	261	144	27	4	0	4	12
20	1	0	21	36	306	592	1378	1272	333	144	9	4	0	8	12
21	1	0	15	36	330	592	1342	1272	357	144	3	4	0	16	12
22	1	0	12	36	342	592	1324	1272	369	144	0	4	0	26	12
23	1	0	0	21	378	640	1288	1218	381	168	0	1	0	30	12
24	1	0	17	26	314	624	1378	1236	325	160	13	2	0	115	12
25	1	0	14	26	326	624	1360	1236	337	160	10	2	0	120	12
26	1	0	19	16	298	656	1414	1200	293	176	23	0	0	172	12
27	1	0	5	26	362	624	1306	1236	373	160	1	2	0	216	12

Weight Distribution (only even weights)														# of Des.	Dim.
No.	0	2	4	6	8	10	12	14	16	18	20	22	24		
28	1	0	13	16	322	656	1378	1200	317	176	17	0	0	292	12
29	1	0	11	26	338	624	1342	1236	349	160	7	2	0	612	12
30	1	0	12	21	330	640	1360	1218	333	168	12	1	0	624	12
31	1	0	1	16	370	656	1306	1200	365	176	5	0	0	836	12
32	1	0	9	21	342	640	1342	1218	345	168	9	1	0	1032	12
33	1	0	8	26	350	624	1324	1236	361	160	4	2	0	1086	12
34	1	0	10	16	334	656	1360	1200	329	176	14	0	0	1408	12
35	1	0	3	21	366	640	1306	1218	369	168	3	1	0	1818	12
36	1	0	7	16	346	656	1342	1200	341	176	11	0	0	2106	12
37	1	0	6	21	354	640	1324	1218	357	168	6	1	0	3814	12
38	1	0	4	16	358	656	1324	1200	353	176	8	0	0	4334	12

Table 3.2: Distribution of Designs in Weight Distribution classes

Now consider the other case. Assume that there is a codeword c of weight 2 in a code C, which is generated by an incidence matrix M. Let i and j be two distinct coordinates in which c has 1s. This will imply that for any of the codewords in C, including rows of M, the value in the i^{th} coordinate is the same as the value in the j^{th} coordinate. If either i or j is the extra column, added to have a self-orthogonal code, then the block represented by the i^{th} column contains all the varieties and $k = v$, which is not true, since $k = 6$ and $v = 16$. Therefore, both column i and column j

are assigned to blocks in the design and they are identical. But it is impossible for a $(16, 6, 3)$-design to have identical blocks. To prove this claim, one can use exactly the same method used by van Rees [21] for the case of $\lambda = 4$. Consider any arbitrary block B^* in a $(16, 6, 3)$-design D. Let a_i be the number of blocks other than B^* in D that intersect with B^* in i varieties. Since there are $b = 24$ blocks in D and B^* is not counted:

$$\sum_{i=0}^{k} a_i = b - 1 = 23. \tag{3.1}$$

From the definition of a BIBD, it can be derived that any of the varieties in B^* occur in $r - 1$ other blocks. If all these occurrences are counted, the result will be as following.

$$\sum_{i=0}^{k} i a_i = k(r - 1) = 48. \tag{3.2}$$

We also know that each pair of varieties in B^* occurs exactly $\lambda - 1$ times.

$$\sum_{i=0}^{k} \binom{i}{2} a_i = \binom{k}{2}(\lambda - 1) = 30 \tag{3.3}$$

From equations 3.2 and 3.3 it can be concluded that

$$\sum_{i=0}^{k} i^2 a_i = 2\sum_{i=0}^{k} \binom{i}{2} a_i + \sum_{i=0}^{k} i a_i = 2\binom{k}{2}(\lambda - 1) + k(r - 1) = 60 + 48 = 108 \tag{3.4}$$

Now consider the following equation.

$$\begin{aligned} 6\sum_{i=0}^{k} a_i - 5\sum_{i=0}^{k} i a_i + \sum_{i=0}^{k} i^2 a_i &= \sum_{i=0}^{k}(i^2 - 5i + 6)a_i = \sum_{i=0}^{k}(i-2)(i-3)a_i \\ &= 6a_0 + 2a_1 + 2a_4 + 6a_5 + 12a_6 = 6(23) - 5(48) + 108 = 6 \end{aligned} \tag{3.5}$$

Since for all values of i, $a_i \geq 0$ and also all the coefficients are non-negative, $a_6 = 0$, which means that B^* cannot be identical to any other block. Therefore, the incidence matrices of $(16, 6, 3)$-designs do not generate any code that has weight 2.

Finally, in order to find out which designs are embedded in each code, the equivalent codes needed to be determined. The search for an available tool that finds the equivalent codes directly, was not successful. However there is a free, well known, and fast enough program called *nauty* [3; 15] developed by B. D. McKay, which finds isomorphisms in graphs. Therefore, each code C was converted into a bipartite graph $G = (V, E, U)$ and then corresponding graphs were examined for isomorphism. In the process of converting a code C to the corresponding graph $G = (V, E, U)$, each codeword $c \in C$ was represented by a vertex v_c and the i^{th} coordinate was represented by a vertex u_i. If $V = \{v_c \mid c \in C\}$ and $U = \{u_i \mid 1 \leq i \leq n\}$, then $E = \{(v_c, u_i) \mid c[i] = 1\}$. In other words, there is an edge between two vertices, if and only if one vertex represents a codeword, and the other corresponds to a coordinate where that codeword is equal to 1. Example 3.1 shows the procedure for generating the corresponding graph on a smaller scale for a $(7, 3)$ code.

Example 3.1. *Consider a $(7, 3)$ code C, with generator matrix G.*

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

Therefore, the code will be $C = \{(0000000), (1001110), (0101101), (0011011), (1100011), (1010101), (0110110), (1111000)\}$. *Now the corresponding graph can be constructed as shown in Figure 3.1. Although in this*

After converting the codes into graphs, the corresponding graphs were compared

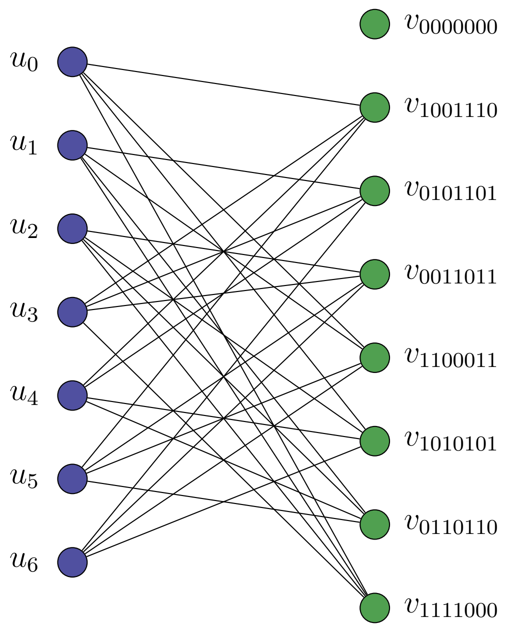

Figure 3.1: Corresponding Graph of Code C

and classified using the *shortg* procedure in *nauty*. The results are provided in Tables 3.3 to 3.46. Each table presents the dimension of the code, the group size of its equivalence class, i.e. the total number of codes that are isomorphic to this code, the Weight Polynomial Number (*WP No.*) to find the weight distribution class of the code in Table 3.2, and the number of designs embedded in the code. The matrix A located below this information can be used to calculate the generator matrix G of each code by setting $G = [I_{dim}\ A]$. In the following chapters, each of these codes will be referenced by its label, which appears in the caption of the corresponding table.

In this chapter, each BIBD was related to the code, generated by the BIBD. In the next step, all the $(25, 12)$ self-orthogonal codes will be enumerated, so that the relationship between the $(16, 6, 3)$-designs and $(25, 12)$ self-orthogonal codes will be determined.

WP *No.* 1	Contains 1 Design
Dim: 10	Group Size: 192

$$\begin{matrix}
1&0&1&0&0&1&1&1&0&1&0&1&0&1&1\\
1&1&1&1&1&1&1&1&0&1&1&0&1&1&1\\
0&0&0&0&0&0&0&0&1&0&1&0&1&1&1\\
1&1&0&1&0&0&1&1&1&0&1&0&1&0&1\\
1&0&0&0&0&1&0&1&1&0&0&0&0&0&1\\
0&0&0&1&0&1&1&1&1&1&0&0&1&1&1\\
1&0&1&0&0&1&1&0&1&1&1&0&1&0&1\\
1&0&0&1&1&0&1&1&1&0&1&0&1&0&1\\
0&0&1&0&0&1&0&0&0&0&1&1&0&0&1\\
1&1&1&1&1&0&0&1&0&0&0&1&0&0&0
\end{matrix}$$

Table 3.3: C_1

WP *No.* 2	Contains 2 Designs
Dim: 10	Group Size: 2034

$$\begin{matrix}
1&0&1&1&1&0&1&0&0&0&1&1&1&0&1\\
1&1&1&0&0&0&0&1&1&1&0&1&0&1&1\\
1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
0&0&0&1&1&1&1&1&0&0&1&1&0&1&1\\
0&0&1&0&0&0&1&0&0&0&0&1&0&1&1\\
0&0&1&1&1&0&1&0&0&1&1&1&1&0&1\\
1&0&1&0&0&0&1&0&1&1&1&0&1&1&1\\
0&1&0&0&0&0&0&0&1&0&0&0&1&1&1\\
1&0&1&0&1&0&1&0&1&1&0&0&1&1&1\\
0&0&0&1&0&1&0&0&0&0&0&1&1&0&1
\end{matrix}$$

Table 3.4: C_2

WP *No.* 3	Contains 2 Designs
Dim: 10	Group Size: 36

$$\begin{matrix}
1&0&0&0&1&1&1&0&0&1&0&0&1&1&0\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&0\\
0&0&1&0&0&1&1&1&0&1&0&1&0&1&0\\
1&1&1&1&0&0&0&1&1&0&0&1&0&1&1\\
1&1&1&0&0&1&0&1&0&0&1&0&1&1&1\\
1&0&1&1&0&1&1&0&0&0&0&1&1&0&0\\
1&1&1&1&1&1&0&1&0&1&0&0&0&0&1\\
0&0&1&1&1&0&1&1&0&0&0&0&1&1&0\\
0&0&1&0&0&0&1&1&1&0&1&1&1&0&0\\
0&1&1&1&1&0&0&0&0&0&1&1&1&0&0
\end{matrix}$$

Table 3.5: C_3

WP *No.* 3	Contains 1 Design
Dim: 10	Group Size: 36

$$\begin{matrix}
1&0&0&0&1&1&1&0&0&1&0&0&1&1&0\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&0\\
0&0&1&0&0&1&1&1&0&1&0&1&0&1&0\\
1&1&1&1&0&0&0&1&1&0&0&1&0&1&1\\
1&1&1&0&0&1&0&1&0&0&1&0&1&1&1\\
1&0&1&1&0&1&1&0&0&0&0&1&1&0&0\\
1&1&1&1&1&1&0&1&0&1&0&0&0&0&1\\
0&0&1&1&1&0&1&1&0&0&0&0&1&1&0\\
0&0&1&0&0&0&1&1&1&0&1&1&1&0&0\\
0&1&1&1&1&0&0&0&0&0&1&1&1&0&0
\end{matrix}$$

Table 3.6: C_4

WP *No.* 4	Contains 4 Designs
Dim: 11	Group Size: 32

0	1	1	0	0	1	1	1	1	1	0	0	1	1
1	1	0	1	0	0	0	0	1	1	1	0	1	0
1	1	1	1	1	1	0	1	1	1	1	1	0	0
1	1	1	1	0	0	1	0	0	1	0	1	0	0
1	0	1	1	0	0	0	1	1	0	0	1	1	0
0	0	0	1	0	1	0	0	0	0	1	1	0	1
1	0	0	0	0	0	1	0	1	0	0	0	0	0
0	0	0	1	1	1	1	1	1	0	1	0	1	1
0	0	1	0	0	1	1	1	1	1	0	1	0	0
0	1	1	1	0	1	0	1	0	1	0	0	1	0
0	1	0	0	1	0	1	0	1	1	1	1	0	0

Table 3.7: C_5

WP *No.* 5	Contains 8 Designs
Dim: 11	Group Size: 48

1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	1	0	0	1	1	1	1
1	0	0	1	1	0	0	1	1	1	1	0	1	1
0	1	0	1	1	1	0	1	1	0	1	1	0	1
1	0	1	1	1	1	0	0	1	0	0	1	1	1
0	1	1	1	1	1	0	1	0	1	0	0	1	1
1	0	1	1	1	0	0	1	1	1	1	0	0	1
0	0	1	0	1	1	1	1	1	0	1	0	1	1
1	0	0	0	1	1	0	1	0	0	0	0	0	1
1	0	0	0	1	1	1	1	0	1	0	1	0	0
0	1	0	0	1	1	0	0	1	1	1	1	0	0

Table 3.8: C_6

WP *No.* 6	Contains 8 Designs
Dim: 11	Group Size: 32

1	1	1	1	0	1	0	1	0	1	0	1	0	1
1	1	0	0	0	0	1	1	0	1	1	0	1	0
0	1	1	1	0	0	1	0	0	0	1	1	1	0
1	1	1	0	0	1	1	0	0	1	0	1	0	0
0	1	0	1	0	1	1	0	1	1	0	0	1	0
0	1	1	0	0	1	1	1	0	1	0	1	0	0
0	1	1	1	1	0	0	0	0	1	1	1	1	1
0	1	0	0	1	1	0	0	0	1	1	1	1	0
1	1	0	1	1	1	0	1	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	1	0
1	1	1	0	1	0	0	1	1	0	1	1	0	1

Table 3.9: C_7

WP *No.* 7	Contains 8 Designs
Dim: 11	Group Size: 48

1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	0	1	1	1	1
0	1	0	1	1	0	0	1	1	1	1	0	1	1
1	1	0	0	1	0	1	0	0	1	1	1	1	1
0	0	0	1	1	0	0	0	0	0	0	1	1	1
0	0	1	0	1	0	0	1	1	0	0	0	0	1
0	1	1	1	1	0	1	0	1	1	0	1	0	1
1	1	1	0	1	1	0	0	1	1	1	0	0	1
0	0	1	0	1	0	1	1	1	1	0	1	0	0
0	1	0	0	1	1	1	0	0	1	1	1	1	1
1	0	1	1	1	1	1	1	1	0	1	1	1	0

Table 3.10: C_8

WP *No.* 8	Contains 7 Designs
Dim: 11	Group Size: 384

```
1 0 0 1 0 1 1 1 1 1 0 0 1 1
1 1 1 0 0 1 1 0 1 1 1 0 0 1
0 1 0 1 0 1 1 1 1 0 1 1 0 1
0 0 1 1 1 1 1 0 1 0 1 0 1 1
0 1 1 1 0 0 0 1 1 1 1 0 1 1
0 0 0 1 1 1 0 1 0 1 1 1 1 1
0 0 0 1 1 0 1 1 0 1 1 1 1 1
0 0 1 0 1 1 1 1 1 0 1 0 1 1
0 1 0 0 0 0 0 0 0 1 1 1 0 1
0 0 1 0 1 1 1 0 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 0 1 1 0 0
```

Table 3.11: C_9

WP *No.* 8	Contains 4 Designs
Dim: 11	Group Size: 288

```
0 1 1 0 1 0 1 1 1 1 0 0 1 1
0 0 0 1 1 0 1 0 0 0 0 0 1 1
1 1 0 0 0 0 0 1 0 0 0 0 1 1
1 1 0 0 1 1 1 0 1 0 1 0 1 1
1 0 0 0 0 0 1 0 0 0 1 1 0 1
1 1 1 0 1 1 1 0 0 0 1 0 1 1
1 1 1 0 1 0 0 1 1 0 1 1 0 1
0 1 0 0 1 0 0 0 0 1 1 0 0 1
1 0 1 1 0 0 1 1 1 0 1 0 1 1
1 0 1 1 0 0 0 0 1 1 1 0 1 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0
```

Table 3.12: C_{10}

WP *No.* 9	Contains 12 Designs
Dim: 11	Group Size: 64

```
0 0 1 1 0 1 1 0 0 1 1 1 1 1
1 1 1 0 0 1 1 1 0 1 1 0 0 1
1 0 0 1 0 1 1 1 1 0 1 1 0 1
0 1 0 1 1 1 1 0 1 0 0 1 1 1
1 0 1 1 0 0 0 1 1 1 1 0 1 1
0 1 1 1 1 1 0 1 0 1 0 0 1 1
0 1 1 1 1 0 1 1 0 1 0 0 1 1
0 0 1 0 1 1 1 1 1 0 1 0 1 1
1 0 0 0 0 0 0 0 0 1 1 1 0 1
0 1 0 0 1 1 1 0 1 1 0 1 0 0
0 1 0 0 0 0 0 1 1 0 0 0 0 0
```

Table 3.13: C_{11}

WP *No.* 10	Contains 12 Designs
Dim: 11	Group Size: 48

```
1 1 1 1 0 1 1 1 1 1 1 1 1 1
1 1 1 0 1 0 0 1 0 0 1 1 1 1
0 1 1 0 1 1 1 0 0 1 1 1 0 1
1 0 1 0 1 1 0 1 1 1 0 1 0 1
1 0 0 0 1 0 1 0 1 1 1 1 1 1
0 0 1 0 1 0 0 0 0 1 0 0 1 1
1 0 1 1 1 1 1 0 0 0 0 1 1 1
1 0 1 1 1 0 1 1 0 1 1 0 0 1
0 1 1 1 1 0 1 0 1 0 1 0 1 1
1 0 1 1 1 1 0 0 1 0 0 1 0 0
0 1 1 1 1 0 0 0 1 1 0 0 1 0
```

Table 3.14: C_{12}

WP *No.* 11	Contains 12 Designs
Dim: 11	Group Size: 64

$$\begin{array}{cccccccccccccc}
0&1&1&0&1&0&1&1&0&1&0&1&1&1\\
1&1&1&0&0&0&0&0&0&1&0&0&0&1\\
1&0&1&1&0&1&0&0&1&0&1&1&0&0\\
1&0&1&0&1&0&0&0&0&1&0&0&0&1\\
0&1&1&1&1&0&0&0&1&1&1&0&1&1\\
1&1&1&0&1&1&1&0&0&0&1&0&1&1\\
0&1&0&0&1&0&0&0&0&0&1&1&0&1\\
1&1&0&0&1&1&0&0&1&1&0&1&1&1\\
1&0&0&0&0&0&1&0&0&0&0&1&1&1\\
0&1&0&0&1&0&1&1&0&1&1&1&0&0\\
1&0&1&1&0&0&1&1&0&1&1&0&1&1
\end{array}$$

Table 3.15: C_{13}

WP *No.* 12	Contains 12 Designs
Dim: 11	Group Size: 12288

$$\begin{array}{cccccccccccccc}
1&1&1&1&0&1&1&1&1&1&1&1&1&1\\
0&0&1&0&1&0&0&0&1&0&1&0&0&1\\
1&1&1&1&1&1&1&0&0&0&1&0&0&1\\
1&1&1&1&1&1&0&0&0&1&1&0&0&1\\
1&1&1&0&1&0&1&0&1&1&0&0&1&1\\
0&1&1&1&1&0&1&0&1&1&0&0&1&1\\
1&1&0&1&1&0&1&1&0&1&0&1&0&1\\
0&0&1&0&1&0&0&0&0&0&1&0&1&1\\
1&0&1&1&1&1&1&0&0&1&1&0&0&1\\
0&0&1&0&1&1&0&0&1&0&1&1&1&0\\
0&0&1&0&1&0&0&1&0&0&0&0&0&0
\end{array}$$

Table 3.16: C_{14}

WP *No.* 13	Contains 16 Designs
Dim: 11	Group Size: 64

$$\begin{array}{cccccccccccccc}
1&1&1&1&0&1&1&1&1&1&1&1&1&1\\
1&1&1&0&1&1&1&1&0&1&0&0&0&1\\
0&1&0&1&1&1&0&0&1&0&1&1&1&1\\
1&1&0&0&1&0&1&0&0&1&1&1&1&1\\
0&1&1&1&1&0&1&0&1&1&1&0&0&1\\
0&1&0&0&1&0&1&1&0&1&1&1&1&1\\
0&1&1&1&1&0&1&0&1&1&0&1&0&1\\
0&1&0&0&1&1&0&0&0&1&1&1&1&0\\
1&1&0&1&1&1&0&1&0&1&0&0&0&0\\
0&0&1&0&1&1&0&0&1&0&0&0&0&1\\
1&1&1&0&1&0&0&1&1&0&1&1&0&1
\end{array}$$

Table 3.17: C_{15}

WP *No.* 14	Contains 16 Designs
Dim: 11	Group Size: 512

$$\begin{array}{cccccccccccccc}
0&1&1&0&0&1&1&1&1&0&1&0&1&1\\
1&0&1&1&0&1&1&0&1&0&1&1&0&1\\
1&0&0&1&0&1&1&1&1&0&1&1&0&1\\
0&1&0&1&1&1&1&0&1&0&0&1&1&1\\
1&0&1&1&0&0&0&1&1&1&1&0&1&1\\
0&1&1&1&1&1&0&1&0&1&0&0&1&1\\
0&1&1&1&1&0&1&1&0&1&0&0&1&1\\
0&0&1&0&1&1&1&1&1&0&1&0&1&1\\
1&0&0&0&0&0&0&0&0&1&1&1&0&1\\
0&1&0&0&1&1&1&0&1&1&0&1&0&0\\
0&0&0&1&0&0&0&0&0&1&0&1&0&0
\end{array}$$

Table 3.18: C_{16}

WP *No.* 15	Contains 16 Designs
Dim: 11	Group Size: 512

1 1 1 1 0 1 1 1 1 1 1 1 1 1
1 1 0 1 1 0 1 1 1 0 0 0 1 1
1 1 1 1 1 1 1 0 0 0 1 0 0 1
0 0 0 0 1 0 1 1 1 1 1 1 1 1
0 0 0 1 1 0 0 1 1 1 1 1 1 1
0 1 1 1 1 0 1 0 1 1 0 1 0 1
0 0 1 0 1 0 0 0 0 1 1 0 0 1
0 0 1 0 1 0 0 0 0 0 0 0 1 0
1 0 1 1 1 0 1 0 1 1 0 1 0 1
1 1 0 1 1 0 1 0 0 0 1 0 1 0
0 0 1 0 1 0 0 0 0 0 1 1 0 1

Table 3.19: C_{17}

WP *No.* 16	Contains 12 Designs
Dim: 11	Group Size: 768

0 1 1 0 0 1 1 1 0 1 0 1 1 1
1 1 1 0 1 0 1 1 0 0 1 0 1 1
1 0 1 1 1 0 0 0 1 0 1 1 0 0
1 0 1 0 1 1 1 1 0 0 1 0 1 1
0 1 1 1 1 1 1 1 1 0 0 0 0 1
1 1 1 0 1 1 1 0 0 0 1 0 1 1
0 1 0 0 1 1 1 1 0 1 0 1 1 1
1 1 0 0 0 1 1 1 1 0 1 1 0 1
1 0 0 0 0 0 1 0 0 0 0 1 1 1
0 1 0 0 0 1 1 1 0 1 1 1 0 0
1 0 1 1 1 0 0 0 0 0 0 0 0 1

Table 3.20: C_{18}

WP *No.* 16	Contains 8 Designs
Dim: 11	Group Size: 1536

1 1 1 1 0 1 1 1 1 1 1 1 1 1
1 0 0 0 1 0 0 1 0 0 0 0 1 1
1 1 0 0 1 1 1 1 1 1 0 0 0 1
1 1 0 0 1 1 1 0 1 1 0 0 1 1
0 1 1 1 1 0 0 1 1 1 0 0 1 1
0 1 0 0 1 0 0 0 0 1 1 0 0 1
0 1 0 0 1 0 0 0 0 1 0 1 0 1
0 0 0 1 1 0 1 0 1 0 0 0 0 1
0 0 0 1 1 1 0 0 1 0 0 0 0 1
1 0 1 1 1 1 1 1 1 0 1 1 1 0
1 0 1 1 1 0 0 1 0 1 0 0 1 0

Table 3.21: C_{19}

WP *No.* 17	Contains 20 Designs
Dim: 11	Group Size: 48

1 1 1 1 0 1 1 1 1 1 1 1 1 1
1 0 0 0 1 0 1 1 1 1 0 1 1 1
0 0 1 1 1 0 0 0 0 0 1 0 0 1
1 0 1 0 1 1 1 0 1 1 1 0 0 1
0 0 0 1 1 1 1 0 0 1 1 1 1 1
0 0 1 0 1 1 1 1 1 1 1 0 0 1
0 1 1 1 1 0 1 0 1 1 0 1 0 1
0 1 0 0 1 1 0 0 0 1 1 1 1 0
1 1 0 1 1 1 0 1 0 1 0 0 0 0
0 1 0 0 1 0 0 0 0 0 0 1 1 1
1 1 1 0 1 0 0 1 1 0 1 1 0 1

Table 3.22: C_{20}

WP *No.* 17	Contains 3 Designs
Dim: 11	Group Size: 256

1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	1	0	0	1	1	1	1
1	1	1	1	1	0	1	0	1	0	0	0	1	1
0	1	0	1	1	1	1	0	1	1	1	0	0	1
0	1	0	0	1	0	0	0	0	1	0	0	1	1
0	1	0	0	1	1	1	1	1	0	0	1	1	1
1	1	0	1	1	0	0	1	1	1	0	1	0	1
0	0	0	1	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	0	0	1	1	1	0	1	0	1
0	1	0	0	1	1	1	1	1	0	1	0	0	0
1	1	1	0	1	1	0	0	0	0	1	1	0	0

Table 3.23: C_{21}

WP *No.* 18	Contains 2 Designs
Dim: 11	Group Size: 1536

1	1	1	1	0	1	1	0	0	1	1	0	0	1
0	1	0	0	0	1	1	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	0	0	1	1	1	1
0	0	0	1	1	1	1	1	1	0	0	1	1	1
1	0	1	1	0	0	1	1	1	0	1	0	1	1
0	0	1	0	1	1	1	1	1	0	0	1	1	1
1	1	1	1	0	1	0	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	0	0	1	1	1
0	1	0	0	1	0	0	0	0	1	1	0	0	1
0	1	0	0	1	0	1	0	1	1	1	0	1	0
1	0	1	1	1	0	0	0	1	0	1	0	1	0

Table 3.24: C_{22}

WP *No.* 18	Contains 52 Designs
Dim: 11	Group Size: 128

1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	1	0	0	1	1	1	1
0	1	0	1	1	0	0	0	0	0	0	0	1	1
1	1	1	1	1	0	1	1	1	0	0	0	0	1
0	1	1	1	1	1	1	0	0	0	1	1	0	1
0	1	0	0	1	1	1	1	1	0	0	1	1	1
1	1	0	1	1	1	1	0	0	0	1	1	0	1
0	1	0	0	1	1	1	1	1	0	1	0	1	1
0	0	0	1	1	1	0	1	0	1	1	1	1	1
0	1	0	0	1	1	1	0	1	0	1	1	0	0
1	1	1	0	1	0	1	0	1	1	0	0	0	0

Table 3.25: C_{23}

WP *No.* 18	Contains 10 Designs
Dim: 11	Group Size: 512

1	1	1	1	0	0	1	1	0	0	1	1	0	1
1	0	0	0	0	0	1	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0	1	1	1	1	0	1
0	0	1	1	1	0	1	0	0	0	0	0	0	1
1	0	0	0	0	0	1	0	0	1	0	1	0	1
1	1	1	0	0	0	0	0	1	1	1	1	1	1
0	1	1	1	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	0	0	0	1	1	1	1	1	1
1	0	1	1	0	0	1	1	1	1	0	0	1	1
0	1	0	0	0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	0	1	1	1	1	1	0	0	0

Table 3.26: C_{24}

WP *No.* 19	Contains 4 Designs
Dim: 12	Group Size: 2229534720

$$\begin{array}{ccccccccccccc}
0&0&0&0&0&0&0&1&1&1&1&0&1\\
1&1&1&1&1&1&1&0&1&0&0&0&1\\
0&0&0&0&0&1&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&1&1&1&0&1&1\\
0&1&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&1&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&1&1&1\\
1&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&1
\end{array}$$

Table 3.27: C_{25}

WP *No.* 20	Contains 8 Designs
Dim: 12	Group Size: 13271040

$$\begin{array}{ccccccccccccc}
0&0&0&0&0&1&1&0&0&1&1&0&1\\
0&0&1&0&0&1&0&1&1&0&0&0&1\\
1&1&0&1&1&1&1&1&0&0&0&0&0\\
0&0&0&0&0&1&1&0&0&1&0&1&1\\
0&1&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&1&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&1&1&0&0&0&1&1&1\\
1&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&1&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&1&0&0&0&0\\
1&1&0&1&1&1&1&0&1&0&0&0&0\\
0&0&1&0&0&0&1&1&1&0&0&0&1
\end{array}$$

Table 3.28: C_{26}

WP *No.* 21	Contains 16 Designs
Dim: 12	Group Size: 1769472

$$\begin{array}{ccccccccccccc}
1&0&0&1&0&1&1&0&0&0&0&0&1\\
1&0&1&1&0&0&1&0&0&0&0&0&1\\
1&1&1&0&0&1&0&1&0&0&1&1&0\\
1&0&0&0&1&0&1&1&1&0&1&1&0\\
1&1&0&1&0&0&1&0&0&0&0&0&1\\
1&0&0&0&0&0&0&1&0&0&1&1&1\\
0&1&1&1&0&1&1&1&0&0&1&1&1\\
1&0&0&1&1&0&0&1&1&0&1&1&0\\
0&0&0&1&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&1&0&1&1&0&0\\
0&0&0&1&0&0&1&0&1&0&0&0&0
\end{array}$$

Table 3.29: C_{27}

WP *No.* 22	Contains 26 Designs
Dim: 12	Group Size: 995328

$$\begin{array}{ccccccccccccc}
1&0&0&1&1&1&0&0&0&0&0&0&1\\
0&1&0&1&1&0&1&1&0&0&1&1&0\\
0&1&1&1&0&0&1&1&1&0&1&1&1\\
1&1&0&1&1&1&1&0&1&0&0&0&0\\
1&0&1&0&1&1&0&1&1&0&1&1&1\\
1&1&0&1&0&0&1&1&0&0&1&1&0\\
1&1&0&0&1&1&0&0&0&0&0&0&1\\
1&1&1&1&1&1&1&0&0&0&0&0&0\\
0&1&0&1&0&1&1&1&0&0&1&1&0\\
0&0&0&0&0&0&0&1&0&1&1&0&0\\
0&0&0&0&0&0&0&1&0&1&0&1&0\\
1&0&0&0&1&1&1&0&0&0&0&0&1
\end{array}$$

Table 3.30: C_{28}

WP *No.* 23	Contains 30 Designs
Dim: 12	Group Size: 120960

1	1	0	1	0	1	1	0	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	0	1	1
0	1	0	0	1	1	0	1	1	1	1	0	0
0	1	1	1	0	1	0	0	0	0	0	0	1
1	1	1	0	1	1	1	0	0	1	0	1	1
0	0	1	0	1	0	1	1	0	0	0	0	1
1	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	1	0	0	0	1	1	0	0	0	1
1	1	1	0	1	0	0	0	1	1	0	1	0
0	1	0	1	1	0	1	0	1	0	1	1	0
1	0	1	1	1	1	0	1	0	0	1	1	1
1	1	1	1	0	0	1	1	0	1	1	0	1

Table 3.31: C_{29}

WP *No.* 24	Contains 115 Designs
Dim: 12	Group Size: 1474560

0	0	0	0	0	1	0	1	0	1	1	0	1
1	0	0	0	0	0	0	1	0	1	1	0	1
1	1	1	0	0	1	0	0	1	1	0	1	0
0	0	0	1	1	0	1	1	1	0	1	1	0
0	0	1	0	0	0	0	1	0	1	1	0	1
0	1	0	1	1	0	1	1	0	0	1	1	0
0	1	0	0	0	0	0	0	1	1	0	1	1
1	1	1	0	0	1	0	1	1	0	1	1	1
0	1	0	1	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	0	1	0	0	0	0
0	1	0	0	0	0	1	0	1	0	0	0	0

Table 3.32: C_{30}

WP *No.* 25	Contains 120 Designs
Dim: 12	Group Size: 387072

0	0	0	0	0	1	0	1	0	1	1	0	1
1	1	0	0	1	0	1	1	0	0	1	1	0
0	1	1	0	1	1	0	0	0	0	0	0	1
0	1	0	1	1	0	1	1	0	0	1	1	0
0	0	1	0	0	0	0	1	0	1	1	0	1
1	1	0	1	1	0	0	1	0	0	1	1	0
0	1	0	0	0	0	0	0	1	1	0	1	1
0	1	1	0	1	1	0	1	0	1	1	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	1	0
0	0	0	0	1	0	0	0	1	1	0	1	1

Table 3.33: C_{31}

WP *No.* 26	Contains 172 Designs
Dim: 12	Group Size: 8847360

1	1	0	0	1	1	1	1	1	0	0	1	1
1	1	1	1	1	1	0	1	0	0	1	0	1
0	1	1	1	0	0	0	0	0	1	0	0	1
0	1	1	1	1	1	0	1	1	0	1	0	1
1	1	1	1	1	1	0	0	1	0	1	0	1
1	0	0	1	1	1	1	1	1	0	0	1	1
1	1	1	1	0	1	0	1	1	0	1	0	1
1	1	1	1	1	0	0	1	1	0	1	0	1
0	1	0	1	0	0	0	0	0	1	1	0	1
0	1	0	1	0	0	0	0	0	0	0	1	0
0	1	0	1	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	1	0	0

Table 3.34: C_{32}

WP *No.* 27	Contains 216 Designs
Dim: 12	Group Size: 24576

$$\begin{matrix}
0&0&0&0&1&0&0&1&0&1&0&1&1\\
0&0&1&1&0&1&0&0&0&0&0&1&1\\
1&0&1&0&1&1&0&1&1&0&1&1&1\\
0&1&1&1&1&0&0&0&1&1&1&1&1\\
0&0&1&1&1&1&0&0&0&0&0&0&1\\
0&1&0&1&1&1&0&0&1&1&1&1&1\\
0&1&1&0&0&1&1&1&0&1&1&0&0\\
1&1&1&0&1&1&0&1&1&0&0&1&1\\
0&1&0&1&1&0&0&1&0&1&1&1&0\\
0&0&0&0&0&0&0&1&1&1&0&0&0\\
0&1&1&1&0&1&1&1&1&0&1&0&1\\
0&0&0&1&0&0&1&0&1&1&0&0&1
\end{matrix}$$

Table 3.35: C_{33}

WP *No.* 28	Contains 292 Designs
Dim: 12	Group Size: 884736

$$\begin{matrix}
1&1&1&0&1&0&1&1&0&1&1&0&1\\
1&0&0&0&0&1&1&0&0&1&0&0&1\\
0&1&1&1&1&0&0&1&1&1&0&1&1\\
0&1&1&1&1&0&0&0&1&1&1&1&1\\
1&1&0&1&1&0&1&1&0&1&1&0&1\\
1&0&1&1&1&0&1&1&0&1&1&0&1\\
1&1&1&1&0&0&0&0&0&0&0&0&1\\
1&0&0&0&1&1&1&0&0&0&0&0&1\\
0&1&1&1&0&0&1&0&0&0&0&0&1\\
0&0&0&0&1&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&1&1&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&1&1&0
\end{matrix}$$

Table 3.36: C_{34}

No.: 29	Contains 612 Designs
Dim: 12	Group Size: 73728

$$\begin{matrix}
0&0&1&0&1&1&0&1&1&1&1&1&1\\
0&0&0&1&1&0&0&1&0&0&1&0&1\\
1&0&1&0&0&1&0&0&0&0&1&0&1\\
1&0&1&0&0&0&0&0&1&0&1&0&1\\
1&1&1&0&0&1&1&1&1&0&0&1&1\\
1&0&0&0&0&1&0&0&1&0&1&0&1\\
0&0&1&1&0&1&0&1&1&1&1&1&1\\
1&1&1&0&0&1&1&1&1&1&0&0&1\\
1&0&0&0&0&0&0&1&0&1&0&1&1\\
1&0&0&0&0&0&0&1&0&0&1&0&0\\
0&1&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&1&0
\end{matrix}$$

Table 3.37: C_{35}

No.: 30	Contains 624 Designs
Dim: 12	Group Size: 221184

$$\begin{matrix}
1&1&1&0&1&1&0&1&0&1&1&0&1\\
0&0&0&1&1&1&0&0&0&1&0&0&1\\
1&1&1&0&0&0&1&1&1&1&0&1&1\\
1&1&1&0&1&0&1&1&1&0&0&1&1\\
0&1&0&0&1&1&1&0&1&1&1&1&1\\
0&0&1&0&1&1&1&0&1&1&1&1&1\\
1&1&1&1&0&0&0&0&0&0&0&0&1\\
1&0&0&0&1&1&1&0&1&1&1&1&1\\
1&1&1&0&0&1&1&1&1&0&0&1&1\\
0&0&0&0&0&0&0&1&1&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&1&1&0\\
0&0&0&0&0&0&1&1&0&0&1&0&0
\end{matrix}$$

Table 3.38: C_{36}

WP *No.* 31	Contains 836 Designs
Dim: 12	Group Size: 23040

0 0 0 0 0 1 0 1 0 1 0 0 1 1
0 1 1 1 1 1 1 1 0 0 0 0 1 1
0 0 1 1 1 1 0 0 1 1 1 0 1 1
0 1 0 1 1 0 0 1 0 1 1 1 1 1
0 0 0 1 1 0 0 1 1 0 0 0 0 1
0 0 1 0 1 0 0 0 0 0 0 1 1 1
0 1 1 1 1 0 1 0 1 1 0 1 0 1
0 0 0 1 1 0 1 1 0 1 1 1 1 1
0 1 0 0 1 1 1 1 1 1 1 0 0 1
0 0 1 0 1 1 0 0 1 0 1 1 1 0
0 1 1 1 0 0 1 0 1 0 1 1 0 0
1 1 0 0 1 1 1 0 1 1 0 1 0 0

Table 3.39: C_{37}

WP *No.* 32	Contains 1032 Designs
Dim: 12	Group Size: 82944

1 0 1 0 1 1 1 1 1 0 1 0 1
1 0 0 1 1 0 1 1 0 1 1 1 1
0 1 1 1 1 1 1 1 0 1 0 0 1
0 1 1 1 1 0 1 1 1 1 0 0 1
0 0 1 1 1 1 0 0 1 1 1 1 1
0 1 0 1 1 1 1 1 1 1 0 0 1
0 0 1 1 0 1 0 1 1 1 1 1 1
0 0 1 1 0 1 1 0 1 1 1 1 1
0 1 0 1 0 0 0 0 0 1 1 0 1
0 1 0 1 0 0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0 0 1 0 1 0
0 1 0 1 1 0 1 1 0 1 1 0 0

Table 3.40: C_{38}

WP *No.* 33	Contains 1086 Designs
Dim: 12	Group Size: 18432

0 0 0 0 0 1 0 1 0 1 1 0 1
0 0 0 1 1 1 1 1 0 1 1 0 0
0 1 1 0 1 1 0 0 0 0 0 0 1
1 0 0 0 1 1 1 1 0 1 1 0 0
1 1 1 1 0 1 0 1 0 0 1 1 1
1 1 0 1 1 0 0 1 0 0 1 1 0
1 0 0 1 0 1 0 0 1 0 0 0 1
1 0 1 1 1 0 0 1 0 0 1 1 0
1 0 0 1 0 0 1 0 0 0 0 0 0
1 0 0 1 1 1 0 0 1 1 0 1 0
0 0 0 0 0 0 0 1 0 1 0 1 0
0 0 0 0 1 0 0 0 1 1 0 1 1

Table 3.41: C_{39}

WP *No.* 34	Contains 1408 Designs
Dim: 12	Group Size: 110592

1 1 0 0 1 1 1 1 0 1 1 0 1
0 0 1 1 1 0 1 1 1 1 1 0 1
0 1 1 1 1 1 0 0 0 1 1 1 1
0 1 1 1 1 0 1 1 1 1 0 0 1
0 0 1 1 1 1 1 1 1 1 0 0 1
0 1 0 1 1 1 0 0 1 1 1 1 1
1 1 1 1 0 1 0 1 1 0 1 0 1
1 1 1 1 0 1 1 0 1 0 1 0 1
1 0 0 1 0 0 1 1 0 0 0 0 1
1 0 0 1 0 0 0 0 0 1 0 0 0
1 0 0 1 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 1 0 0 1 0

Table 3.42: C_{40}

WP *No.* 35	Contains 1818 Designs
Dim: 12	Group Size: 6912

$$\begin{array}{ccccccccccccc}
1&1&0&1&0&1&1&0&0&0&1&1&0\\
0&0&1&1&0&1&1&1&1&1&0&1&1\\
1&1&1&1&1&0&0&1&0&0&0&1&0\\
1&0&1&0&0&1&1&1&1&0&0&1&0\\
1&1&1&0&0&0&1&1&0&1&1&0&0\\
1&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&1&1&0&0&0&0&1&0&1&0&1\\
0&1&1&0&1&1&0&0&1&1&1&1&1\\
0&1&0&1&0&0&1&1&1&1&0&1&0\\
0&1&0&0&0&1&0&1&0&0&0&0&0\\
1&1&1&1&0&1&1&0&0&1&1&0&1\\
0&0&0&0&0&0&1&0&1&0&1&0&0
\end{array}$$

Table 3.43: C_{41}

WP *No.* 36	Contains 2106 Designs
Dim: 12	Group Size: 49152

$$\begin{array}{ccccccccccccc}
1&1&1&0&1&1&1&0&1&0&0&1&1\\
0&0&1&1&0&1&0&0&0&0&0&1&1\\
1&0&0&1&1&1&0&1&1&1&0&1&1\\
1&0&0&1&1&1&0&0&1&1&1&1&1\\
1&1&0&1&1&0&0&0&0&0&0&0&1\\
1&0&1&1&1&0&1&1&0&0&1&1&1\\
1&1&1&1&0&1&0&1&1&0&1&0&1\\
0&0&1&1&0&0&0&0&1&0&0&1&1\\
0&1&0&1&0&0&0&0&0&0&0&1&0\\
0&1&0&1&0&1&1&1&1&0&1&0&0\\
0&1&1&1&1&1&0&1&1&0&1&0&1\\
0&0&0&0&0&0&0&1&0&1&1&0&0
\end{array}$$

Table 3.44: C_{42}

WP *No.* 37	Contains 3814 Designs
Dim: 12	Group Size: 9216

$$\begin{array}{ccccccccccccc}
1&0&0&1&0&0&1&0&1&1&1&1&0\\
0&0&1&1&0&1&0&0&0&0&0&1&1\\
1&0&1&1&1&1&1&0&0&0&0&1&0\\
1&1&1&0&0&0&1&1&0&1&0&1&0\\
1&0&1&0&0&1&1&1&1&0&1&0&0\\
1&1&0&0&0&1&1&1&0&1&0&1&0\\
1&1&1&1&0&1&0&1&1&0&1&0&1\\
0&1&1&0&1&1&0&0&1&1&1&1&1\\
0&1&0&1&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&1&1&1&0&0&0\\
0&1&1&1&0&1&1&1&1&0&1&0&1\\
0&0&0&0&0&0&0&1&0&1&1&0&0
\end{array}$$

Table 3.45: C_{43}

WP *No.* 38	Contains 4334 Designs
Dim: 12	Group Size: 12288

$$\begin{array}{ccccccccccccc}
1&0&0&1&0&0&1&0&1&1&1&1&0\\
0&0&1&1&1&1&0&1&0&1&1&1&1\\
0&0&1&0&0&1&1&1&0&1&1&1&0\\
1&1&1&0&1&0&1&0&0&0&1&1&0\\
1&0&1&0&1&1&1&0&1&1&0&0&0\\
1&1&0&0&1&1&1&0&0&0&1&1&0\\
1&1&1&1&0&1&0&1&1&0&1&0&1\\
1&1&1&1&0&1&0&1&1&0&0&1&1\\
0&1&0&1&1&0&0&1&0&1&1&1&0\\
0&0&0&0&0&0&0&1&1&1&0&0&0\\
0&1&1&1&1&1&1&0&1&1&0&0&1\\
1&0&0&1&1&0&0&0&0&0&0&0&0
\end{array}$$

Table 3.46: C_{44}

Chapter 4

Enumeration of Self-Orthogonal $(25, 12)$ Codes

The next step in searching for the mapping between $(16, 6, 3)$ designs and $(25, 12)$ self-orthogonal codes is to generate the codes. In this chapter, first a proposed iterative process of enumerating self-orthogonal $(25, 12)$ codes will be explained. Although this method is not time efficient, it provides us with results, which are useful in analyzing the codes. After the explanation of the above-mentioned process, the procedure of obtaining $(25, 12)$ self-orthogonal codes from $(26, 13)$ self-dual codes, as well as the distribution of the codes themselves in distance classes, will be provided.

4.1 Theoretical Background for the Iterative Enumeration

In order to reduce the number of occurrences of equivalent codes during the enumeration process for generating $(25, 12)$ binary codes, a particular ordering on the generator matrices was considered. This ordering helped the process to skip over some of the equivalent codes. Each generator matrix generated in this ordering, is called an Ordered Generator Matrix. Here is the definition:

Definition 16. *A Generator Matrix G for a binary code C is said to be an Ordered Generator Matrix if and only if it maintains the following rules:*

1. *For any row r_i in G, the weight of r_i is less than or equal to the weight of any row appearing below r_i.*
2. *For any row r_i in G, the weight of r_i is less than or equal to the weight of any codeword that is a linear combination of rows of G, including r_i .*
3. *If (1) holds with equality, then for any pair of rows r_i and r_j of G, with $i < j$, r_i should be a binary representation of a larger number than the number that r_j represents in binary.*
4. *If (2) holds with equality, then for any row r_i in G and a codeword $c \in C$, if the coefficient of r_i in the linear combination of rows, which results in c, is not zero, then r_i should be a binary representation of a larger number than the number that c represents in binary.*

The following theorems will serve as theoretical validation of our procedure for generating the $(25, 12)$ binary self-orthogonal codes.

Lemma 4.1. *For each (n, k) binary code C, there exists an Ordered Generator Matrix G that generates C.*

Proof. Consider the list of codewords of C, sorted in ascending order based on their weights; and in case of tie, the codeword that represents the larger number in binary appears first. Now, let the first k codewords in our ordering, which are all linearly independent, be the rows of G, preserving their ordering from the sorted list of codewords. By definition, G is an ordered generator matrix. □

Lemma 4.1 guarantees that the search domain can be restricted to the ordered generator matrices, without the loss of any desirable code.

The property below is a straightforward derivation from Condition 4 in Definition 16. During the process of generating the codes, the algorithm will check for a codeword that contradicts this property. If such a codeword c exists, then the algorithm will skip that generator matrix and proceed to the next one.

Proposition 4.2. *Let r_i for $1 \leq i \leq k$ be the i^{th} row of an ordered generator matrix G. Then for any linear combination, c, of the first i rows, if the coefficient of r_i in c is non-zero, then either c has a weight greater than the weight of r_i, or they have equal weights and the binary number represented by r_i is greater than the one represented by c.*

Lemma 4.3. *For any (n, k) binary code C and any positive integer d, the ordered generator matrix of C cannot contain more than $\lfloor \log_2 A(n, d) \rfloor$ rows of weight d.*

Proof. (By contradiction) Assume C has an ordered generator matrix that includes $\hat{k} > \lfloor \log_2 A(n, d) \rfloor$ rows of weight d. Since the rows are linearly independent, those $\hat{k}$ rows can generate a new code $\hat{C}$ of dimension $\hat{k}$. Therefore $\hat{C}$ has $2^{\hat{k}} > A(n, d)$ codewords. By the definition of $A(n, d)$, the distance of $\hat{C}$ is strictly less than d. However, this implies that there is a non-zero linear combination of those $\hat{k}$ rows of weight d that has weight less than d. Since this contradicts Property 4.2, such an ordered generator matrix cannot exist. □

Using Lemma 4.3, the search can be limited to matrices with at most $\lfloor \log_2 A(n, d) \rfloor$ rows of weight d. However, the following lemmas result in better bounds on the weight of rows in an ordered generator matrix.

Proposition 4.4. *For any arbitrary selection of* s *columns and* t *rows in a generator matrix* G*, with* $t > s$*, then there is another generator matrix* $\hat{G}$ *with following properties:*

1) $\hat{G}$ *generates the same code as* G *does.*

2) $\hat{G}$ *is identical to* G *in all the rows other than the* t *chosen rows.*

3) There are at least $t - s$ *of the chosen rows, which are in* $\hat{G}$ *and have all* 0*s in those* s *columns.*

Proof. Form a new matrix $M_{t\times s}$ from the intersection of those t rows and s columns. Since $t > s$, there are at most s rows of M that can be linearly independent. Therefore there are at least $t - s$ linearly dependent rows in M. So, there is a set of elementary row operations on M, which can set $t - s$ rows of M to all-zero rows. If the same elementary row operations are performed on the corresponding rows in G, the result will be $\hat{G}$, because, by the construction, it satisfies the second and the third properties. And since only elementary row operations are used, and the code is a linear code, $\hat{G}$ and G both generate the same code. □

Note that $\hat{G}$ from Proposition 4.4 is still a generator matrix and there is no all-zero row in it.

Lemma 4.5. *For any* (n, k) *binary code, there exists no row of weight* $t > n - k + 1$ *in its ordered generator matrix.*

Proof. Assume a row, r, of weight $t > n-k+1$ exists in the ordered generator matrix of an (n, k) binary code. Consider the $n - t$ coordinates which are not supported by r. Since $k - 1 > n - t$, Proposition 4.4 can be applied and conclude that there is a linear combination, c, of rows in the ordered generator matrix, excluding r, such that c does not support any coordinate that is not supported by r. Now consider the codeword $\hat{r} = r \oplus c$. Notice that the weight of $\hat{r}$ is less than the weight of r and also r has a non-zero coefficient in the linear combination that produces $\hat{r}$. But this contradicts Property 4.2. Therefore, the ordered generator matrix cannot have a row of weight t. $\square$

Since $16 > 25 - 12 + 1 = 14$, the search for ordered generator matrices can skip matrices containing a row of weight $16, 18, 20, 22$, or 24.

Lemma 4.6. *For any* (n, k) *binary code* C *with ordered generator matrix* G*, and for integers* t_1 *and* t_2*, where* $0 < t_1 \leq t_2 \leq n$*, if* $n - (t_2 + \frac{t_1}{2}) < k - 2$*, then* G *cannot have rows of weights* t_1 *and* t_2 *at the same time.*

Proof. Assume r_1 and r_2 are two rows in G with weights t_1 and t_2, respectively. Let $R_1 = \{i \mid r_1[i] = 1\}$ and $R_2 = \{i \mid r_2[i] = 1\}$ be sets of coordinates supported by each of r_1 and r_2. Note that $|R_1 \cap R_2| \leq \frac{t_1}{2}$, otherwise $r_1 \oplus r_2$ can replace r_2. Therefore, $|R_1 \cup R_2| = |R_1| + |R_2| - |R_1 \cap R_2| \geq t_1 + t_2 - \frac{t_1}{2} = t_2 + \frac{t_1}{2}$.

Consider the $k - 2$ rows of G other than r_1 and r_2. Since $k - 2 > n - (t_2 + \frac{t_1}{2}) \geq n - |R_1 \cup R_2|$, Proposition 4.4 is applicable. Therefore, there are at least $u = k - 2 - n + |R_1 \cup R_2|$ linearly independent linear combinations of those rows, $x_j, 1 \leq j \leq u$, such that these linear combinations do not support the coordinates which are not supported by either r_1 or r_2. Define $X_j = \{i \mid x_j[i] = 1\}, 1 \leq j \leq u$, then it can be claimed that for all $j \in \{1, 2, \ldots, u\}, |R_1 \cap X_j| \leq |R_1 \setminus X_j|$ and $|R_2 \cap X_j| \leq |R_2 \setminus X_j|$. Assume that there is an X_j such that either $|R_1 \cap X_j| > |R_1 \setminus X_j|$ or $|R_2 \cap X_j| > |R_2 \setminus X_j|$. Then either the weight of $r_1 \oplus x_j$ is less than t_1 or the weight of $r_2 \oplus x_j$ is less than t_2. But this contradicts Property 4.2, proving the claim.

Now it can be considered true that $R_1 \cap R_2 \cap X_j = \emptyset, j \in \{1, 2, \ldots, u\}$. In order to prove the claim by contradiction, assume that there is an X^* such that $R_1 \cap R_2 \cap X^* \neq \emptyset$, then, considering the fact that $X^* \subseteq R_1 \cup R_2$, either $|R_1 \cap X^*| > |R_1 \setminus X^*|$ or $|R_2 \cap X^*| > |R_2 \setminus X^*|$, which was shown to be impossible. Therefore the proof can be continued with the assumption that for any $j \in \{1, 2, \ldots, u\}, R_1 \cap R_2 \cap X_j = \emptyset$ and $|R_1 \cap X_j| \leq |R_1 \setminus X_j|$ and $|R_2 \cap X_j| \leq |R_2 \setminus X_j|$, which implies that $|R_1 \cap X_j| = |R_2 \cap X_j|, 1 \leq j \leq u$.

Since the rows of G are linearly independent, all the x_js are non-zero. Consider all the linear combinations of x_js. Since there are at least u linearly independent x_js and each of them has at least a 1 in a coordinate in $R_1 \setminus R_2$, there is a linear combination y of x_js that has at least u 1s in coordinates in $R_1 \setminus R_2$. Based on the aforementioned claim, y has the same number of 1s in coordinates in $R_2 \setminus R_1$. Therefore the weight of y is at least $2u$. Now consider the linear combination $r_1 \oplus r_2 \oplus y$ with weight

$$|R_1| + |R_2| - 2|R_1 \cap R_2| - 2u = t_1 + t_2 - 2|R_1 \cap R_2| - 2(k - 2 - n + |R_1 \cup R_2|)$$

$$= t_1 + t_2 - 2|R_1 \cap R_2| - 2(k - 2 - n + |R_1| + |R_2| - |R_1 \cap R_2|)$$

$$= t_1 + t_2 + 2n - (2k - 4) - 2t_1 - 2t_2 = 2n - (2k - 4) - t_1 - t_2.$$

From Proposition 4.2, it can be concluded that weight of $r_1 \oplus r_2 \oplus y$ should be greater than or equal to t_2, therefore

$$2n - (2k - 4) - t_1 - t_2 \geq t_2 \Rightarrow 2n - (2t_2 + t_1) \geq 2k - 4 \Rightarrow n - (t_2 + \frac{t_1}{2}) \geq k - 2.$$

But this contradicts the assumption of the lemma that $n - (t_2 + \frac{t_1}{2}) < k - 2$. Therefore the assumption that r_1 and r_2 are rows of G is incorrect and G cannot have rows of weights t_1 and t_2 at the same time. $\square$

From Lemma 4.6, it can be derived that rows r_1 and r_2 of weighs t_1 and t_2, where $(t_1, t_2) \in \{(4, 14), (6, 14), (8, 12), (8, 14), (10, 12), (10, 14), (12, 12), (12, 14), (14, 14)\}$ cannot appear in the ordered generator matrix of a $(25, 12)$ binary code.

In the special case of self orthogonal (n, k) codes, Lemma 4.6 can also be expanded by the following corollary.

Corollary 1. *For any (n, k) self-orthogonal binary code C with ordered generator matrix G, and for integers t_1 and t_2, where $0 < t_1 \le t_2 \le n$ and t_1 is not divisible by 4, if $n - (t_2 + \frac{t_1}{2}) = k - 2$, then G cannot have rows of weights t_1 and t_2 at the same time.*

Proof. Assume r_1 and r_2 are two rows in G with weights t_1 and t_2, respectively. Let $R_1 = \{i \mid r_1[i] = 1\}$ and $R_2 = \{i \mid r_2[i] = 1\}$ be sets of coordinates supported by each of r_1 and r_2. Note that $|R_1 \cap R_2| \le 2\lfloor \frac{t_1}{4} \rfloor < \frac{t_1}{2}$, otherwise $r_1 \oplus r_2$ can replace r_2 or the code is not self-orthogonal. Therefore,

$$|R_1 \cup R_2| = |R_1| + |R_2| - |R_1 \cap R_2| \ge t_1 + t_2 - 2\lfloor \frac{t_1}{4} \rfloor = t_2 + 2\lceil \frac{t_1}{4} \rceil.$$

Consider the $k - 2$ rows of G other than r_1 and r_2. Since $k - 2 = n - (t_2 + \frac{t_1}{2}) > n - (t_2 + 2\lceil \frac{t_1}{4} \rceil)$ satisfies the condition in Proposition 4.4, there is at least 1 non-zero linear combination of those rows, x, such that this linear combination does not support the coordinates which are supported by neither r_1 nor r_2. Define $X = \{i \mid x[i] = 1\}$, so $X \subseteq R_1 \cup R_2$ and it can be claimed that $|R_1 \cap X| \le |R_1 \setminus X|$ and $|R_2 \cap X| \le |R_2 \setminus X|$. Assume that either $|R_1 \cap X| > |R_1 \setminus X|$ or $|R_2 \cap X| > |R_2 \setminus X|$. These cases imply that either the weight of $r_1 \oplus x$ is less than t_1 or the weight of $r_2 \oplus x$ is less than t_2. But this contradicts Property 4.2 and the assumption is incorrect.

Now the assumption that $R_1 \cap R_2 \cap X = \emptyset$ is true. In order to prove the claim by contradiction, assume that there is an X such that $R_1 \cap R_2 \cap X \neq \emptyset$, then, considering the fact that $X \subseteq R_1 \cup R_2$, either $|R_1 \cap X| > |R_1 \setminus X|$ or $|R_2 \cap X| > |R_2 \setminus X|$, which was shown to be incorrect. Therefore the proof can be continued with the claim that for any $R_1 \cap R_2 \cap X_j = \emptyset$, $|R_1 \cap X| \leq |R_1 \setminus X|$ and $|R_2 \cap X| \leq |R_2 \setminus X|$, which implies that $|R_1 \cap X| = |R_2 \cap X|$.

Since the rows of G are linearly independent, x is non-zero. Therefore, x has at least a 1 in the coordinates supported by either of r_1 or r_2. By the premises, it can be concluded that it has at least a 1 in coordinates supported by each of r_1 and r_2. Since the code is self-orthogonal, x should have at least two 1s in each set of coordinates that are supported by r_1 or r_2. So, the weight of x, u, is at least 4. Now consider the linear combination $r_1 \oplus r_2 \oplus x$, which will have weight

$$|R_1| + |R_2| - 2|R_1 \cap R_2| - |X| < t_1 + t_2 - 2(2\lfloor\frac{t_1}{4}\rfloor) - u < t_1 + t_2 - (t_1 - 2) - 4 < t_2 - 2$$

But this contradicts Proposition 4.2. Therefore, G cannot have r_1 and r_2 at the same time. □

From Corollary 1, it can be concluded that rows r_1 and r_2 with weights t_1 and t_2 cannot appear in the ordered generator matrix of a $(25, 12)$ self-orthogonal binary code if $(t_1, t_2) \in \{(2, 14), (6, 12), (10, 10)\}$.

Considering the results from Lemma 4.5, Lemma 4.6, and Corollary 1, it can be concluded that it is impossible for an ordered generator matrix of a $(25, 12)$ binary self-orthogonal code to have a row of weight 14.

4.2 Iterative Computation of (25,12) Self-Orthogonal Codes

For each code, there are many equivalent codes. During the enumeration of codes these equivalent codes may be generated frequently, and consume both computational and storage resources. In order to limit the multiple occurrence of these equivalent codes, the process of generating $(25, 12)$ self-orthogonal codes was broken into multiple steps. First, the $(25, 6)$ self-orthogonal codes were generated iteratively. The process after that can be described as a branch and bound search, where in each step the process would prune the branches that represent a code, which has already occurred in a form of an equivalent code, and then the dimension of each code would be increased by one. This step will be repeated, until the process reaches a code of dimension 12. Detailed explanation of the process is provided in the following subsections.

4.2.1 Generating (25,6) Self-Orthogonal Codes

In the first step of generating (25,12) self-orthogonal codes, the ordered generator matrices for (25,6) self-orthogonal codes were enumerated. In order to restrict the search domain, the results of previous section were used. The algorithm started from a code of dimension 1 and added up to 5 feasible rows to the generator matrix, checking the self-orthogonality, linearity, and up to some extent the isomorphism conditions mentioned previously. For the computation of the ordered generator matrices of the (25,6) self-orthogonal codes, 17 threads were used on the Helium-01 computing cluster at the University of Manitoba[i]. One thread served as a producer and the other 16 were consumers. Since the server can only run up to 16 threads simultaneously, and

[i]8 Dual-Core AMD Opteron 885 2.6 GHz CPUs and 32 GB of RAM using the Scientific Linux 6.x Operating System[1]

the producer only needed to run, whenever at least one consumer's ordered generator matrix queue was empty. This process could be modified, in order to use more than one server. However, the computing did not take a long time, relative to the next steps of the process, therefore using more servers was unnecessary. The producer enumerated the ordered generator matrices up to 4 rows, and then copied each feasible matrix to a queue for one of the consumers. Each consumer would enumerate the other 2 rows and write the feasible matrices to a file. Finally, there were over 20,000,000 ordered generator matrices for (25,6) self-orthogonal codes. However, many of them were equivalent. In order to remove the duplicate $(25, 6)$ self-orthogonal codes, a procedure like the one in Chapter 3 was used. Each code was transformed into a corresponding graph and *nauty* was used to remove the isomorphic copies. For the computation of this step, 64 cores of Helium(02-05)[ii] servers were exploited for about 3 weeks.

Having all the $(25, 6)$ self-orthogonal codes up to equivalence classes, our next step was to increase the dimension of our codes, step by step, to 12. In each step, the dimension increased by one. This was done by adding a new row, which is not in the code already, to the generator matrix. This operation is called an augmentation of the code by that row. In other words, the rows that maintain the properties of the ordered generator matrix were added to the matrix. In order to reduce the number of equivalent codes, the 1s in each row were forced to locate in the leftmost possible coordinates, as long as the coordinates are supported identically by the previous rows. Then the results were compared against each other to remove isomorphic copies.

However, due to the exponential nature of the problem, all the branch and bound methods used in this algorithm were not efficient enough. Therefore, another method

[ii] 4 servers, each with 8 Dual-Core AMD Opteron 885 2.6 GHz CPUs and 32 GB of RAM using the Scientific Linux 6.x Operating System [1]

was chosen to generate the $(25, 12)$ self-orthogonal binary codes.

4.3 Generating $(25, 12)$ self-orthogonal codes from $(26, 13)$ self-dual codes

Shortening is another method to get a linear code from another linear code. In the process of shortening, a coordinate will be selected and the codewords with the value 1 in that coordinate will be removed from the code. Finally, the selected coordinate, where all the codewords have zero, will be omitted. Therefore, the resultant code will be shorter by one in length and also smaller by one in dimension. When the starting code is self-dual, since all the codewords with the value 1 in the deleted coordinate have been removed from the code, shortening does not harm the self-orthogonality of the code and the shortened code will also be self-orthogonal. However it will no longer be self-dual.

Harada and Munemasa [2] have generated all the $(26, 13)$ self-dual codes up to equivalence. There are 103 such codes. Shortening all of the $(26, 13)$ self-dual codes on every coordinate, resulted in $26 \times 103 = 2678$ $(25, 12)$ self-orthogonal codes. However, after classifying the codes into weight distribution classes and removing equivalent copies of each code, 331 inequivalent $(25, 12)$ self-orthogonal codes remained. Table 4.1 shows the number of resultant $(25, 12)$ self-orthogonal codes with different distances, with or without zero-columns.

These numbers match the results from the $(25, 12)$ self-orthogonal codes, generated by I. Bouyukliev [10], which he kindly provided to us through a personal communication.

Since it is shown in Chapter 6 that codes with a codeword of weight 2 cannot

Distance	With zero Col.	Without zero Col.	Total
2	25	105	130
4	28	168	196
6	1	3	4
8	1	0	1
Total	55	276	331

Table 4.1: (25,12) Self-Orthogonal Codes

contain a $(16,6,3)$-design, the 130 codes with distance 2 can be ignored. Also, as it is known that there are no empty blocks in a design and the added column contains a non-zero number of 1s, it can be implied that a code does not have an all-zero column, if it has an embedded design. Therefore, there are only $168 + 3 = 171$ $(25,12)$ self-orthogonal codes capable of containing $(16,6,3)$-designs.

So far the $(16,6,3)$-designs to self-orthogonal codes of length 25 and dimensions 10, 11, and 12 have been related. All the 331 non-equivalent $(25,12)$ self-orthogonal codes have been also calculated. The only task remaining to be done, in order to find the relationship between the $(16,6,3)$ designs and the $(25,12)$ self-orthogonal codes, is to find out the $(25,12)$ self-orthogonal codes which contain codes of dimensions 10 and 11 with embedded BIBDs.

Chapter 5

Augmenting (25,10) and (25,11) Codes to (25,12) Codes

After classifying the designs based on the codes generated by the incidence matrices of the designs, and then generating all $(25, 12)$ self-orthogonal codes, the last step remaining to determine the relationship between them is to search the codes of dimension 12 for the codes of lower dimensions. However, Instead of directly searching among the codewords of $(25, 12)$ self-orthogonal codes for generator matrices of codes of dimension 10 and 11, the codes of smaller dimension were augmented. In other words, for each code C of dimension k, a program was developed to generate all the codes of dimension $k + 1$ which contain C. This method was more convenient, as similar algorithms has been implemented during this research and also it generated fewer codes and expedited the search.

In the first step, for each code of dimension 10, all the 2^{14} possible rows of even weight that are linearly independent to the rows in the generator matrix were generated. Since the generator matrices were stored in $G = [I_{10}\ A]$ format, the task was only to generate all vectors of the form $(\vec{0}_{10}, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12},$

x_{13}, x_{14}) with even weights. All these rows were examined to see whether or not they maintain the self-orthogonality property of the code. At this point all the self-orthogonal codes of dimension 11 which contain one of the initial codes of dimension 10 are known.

In the next step, the program generated the corresponding graph for each code of dimension 11. Finally, the *shortg* method of *nauty* was used to remove all the duplicates. Since it was important to know all the codes that contain each design, a script program kept track of all the isomorphisms between graphs. The augmentation of the $(25, 10)$ codes with embedded designs in them resulted in 10 new codes of dimension 11 and also 5 codes of dimension 11 from Chapter 3. Among all the augmentations, there is no code of dimension 11 that is augmented from two inequivalent codes of dimension 10 with embedded designs in them. Therefore, there are exactly 30 inequivalent codes of dimension 11 with $(16, 6, 3)-$designs embedded in them. Tables 5.1 to 5.10 contain the information about the new $(25, 11)$ codes and Table 5.11 shows the augmentations which resulted in already existing codes of dimension 11.

Contains 1 Design				Group Size:96					Dim:11				
Augmentation of C_1 by:				(0000000000111011111011000)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	1	20	126	404	646	540	241	60	9	0	0

Table 5.1: C_{45}: An augmentation of (25,10) code C_1

Contains 1 Design				Group Size:64					Dim:11				
Augmentation of C_1 by:				(0000000000110111111001010)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	1	20	126	404	646	540	241	60	9	0	0

Table 5.2: C_{46}: An augmentation of (25,10) code C_1

Contains 1 Design				Group Size:192					Dim:11				
Augmentation of C_1 by:				(0000000000000000111111000)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	1	20	126	404	646	540	241	60	9	0	0

Table 5.3: C_{47}: An augmentation of (25,10) code C_1

Contains 2 Designs				Group Size: 768					Dim: 11				
Augmentation of C_2 by:				(0000000000111011100111010)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	3	23	120	394	652	552	239	54	9	1	0

Table 5.4: C_{48}: An augmentation of (25,10) code C_2

After augmenting all the $(25,10)$ codes to $(25,11)$ codes, the same process was repeated for both the original and the augmented $(25,11)$ codes. In summary, the whole procedure can be explained as: first the 2^{13} possible rows of even weight, starting with 11 zeros, were computed and tested for self-orthogonality. Second, the corresponding graphs were generated. Third, the graphs are used for calculating the codes up to equivalence, similar to the process in Chapter 3. The augmentation resulted in 16 new codes of dimension 12, and 10 $(25,12)$ codes from Chapter 3. Seven of the new codes and eight of the codes from Chapter 3 are augmentations of different codes of dimension 11 which have designs embedded in them. Therefore, there are 36 $(25,12)$ codes, 30 $(25,11)$ codes, and 4 $(25,10)$ codes with embedded incident matrices of $(16,6,3)-$designs. Tables 5.12 to 5.27 present the new codes of dimension 12. Table 5.28 shows the embedding of codes of dimension 11 in codes of dimension 12 from Chapter 3.

Considering the designs from the embedded codes, the number of designs in some of the codes should be updated. Table 5.29 shows the total number of designs in each code.

Contains 2 Designs				Group Size: 3072					Dim: 11			
Augmentation of C_2 by:				(0000000000011001001000110)								
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	4	19	120	406	646	540	247	58	6	1	0

Table 5.5: C_{49}: An augmentation of (25,10) code C_2

Contains 2 Design				Group Size: 48					Dim: 11			
Augmentation of C_3 by:				(0000000000111010011111100)								
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	1	19	132	406	628	540	259	58	3	1	0

Table 5.6: C_{50}: An augmentation of (25,10) code C_3

Finally the relationship between the $(16, 6, 3)$-designs and $(25, 12)$ self-orthogonal codes is determined. In the next chapter, this relationship will be analyzed.

Contains 2 Design				Group Size: 48					Dim: 11			
Augmentation of C_3 by:	(0000000000000001101101100)											
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	0	23	132	394	634	552	251	54	6	1	0

Table 5.7: C_{51}: An augmentation of (25,10) code C_3

Contains 1 Design				Group Size: 48					Dim: 11			
Augmentation of C_4 by:	(0000000000111010011111100)											
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	1	19	132	406	628	540	259	58	3	1	0

Table 5.8: C_{52}: An augmentation of (25,10) code C_4

Contains 1 Design				Group Size: 108					Dim: 11			
Augmentation of C_4 by:	(0000000000101111001101101)											
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	0	9	186	324	648	606	189	84	0	1	0

Table 5.9: C_{53}: An augmentation of (25,10) code C_4

Contains 1 Design				Group Size: 12					Dim: 11			
Augmentation of C_4 by:	(0000000000000001101101100)											
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	0	23	132	394	634	552	251	54	6	1	0

Table 5.10: C_{54}: An augmentation of (25,10) code C_4

Code of Dim. 10	Row added to the generator matrix	Augmentation result
C_1	(0000000000010010100100000)	C_{22}
C_1	(0000000000101001100000000)	C_{24}
C_2	(0000000000100101000100000)	C_{19}
C_3	(0000000000010101000001101)	C_5
C_4	(0000000000101110100000001)	C_9

Table 5.11: Augmentation of dimension 10 codes to codes of dimension 11 from Chapter 3

Contains 9 Design				Group Size: 432					Dim: 12				
Augmentation of C_8 by:				(0000000000001100011011000)									
Augmentation of C_{52} by:				(0000000000000001101101100)									
Augmentation of C_{53} by:				(0000000000000000110110110)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	1	45	270	792	1254	1098	513	112	9	1	0

Table 5.12: C_{55}: An augmentation of (25,11) codes C_8, C_{52}, and C_{53}

Contains 18 Design				Group Size: 256					Dim: 12				
Augmentation of C_5 by:				(0000000000001101100010100)									
Augmentation of C_{13} by:				(0000000000011111101011100)									
Augmentation of C_{50} by:				(0000000000000001101101100)									
Augmentation of C_{51} by:				(0000000000010100011010111)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	2	45	266	792	1260	1098	509	112	10	1	0

Table 5.13: C_{56}: An augmentation of (25,11) codes C_5, C_{13}, C_{50}, and C_{51}

Contains 10 Design				Group Size: 6144					Dim: 12				
Augmentation of C_{19} by:				(0000000000001010001110010)									
Augmentation of C_{48} by:				(0000000000011001001000110)									
Augmentation of C_{49} by:				(0000000000010010001010000)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	6	45	250	792	1284	1098	493	112	14	1	0

Table 5.14: C_{57}: An augmentation of (25,11) codes C_{19}, C_{48}, and C_{49}

Contains 24 Design				Group Size: 1536					Dim: 12				
Augmentation of C_9 by:				(0000000000000110001111000)									
Augmentation of C_{17} by:				(0000000000000110010101010)									
Augmentation of C_{52} by:				(0000000000000000110110110)									
Augmentation of C_{54} by:				(0000000000010100001001011)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	4	45	258	792	1272	1098	501	112	12	1	0

Table 5.15: C_{58}: An augmentation of (25,11) codes C_9, C_{17}, C_{52}, and C_{54}

Contains 23 Design				Group Size: 512					Dim: 12				
Augmentation of C_{11} by:	(0000000000000110001111000)												
Augmentation of C_{24} by:	(0000000000000001111001100)												
Augmentation of C_{45} by:	(0000000000001100000010010)												
Augmentation of C_{46} by:	(0000000000001100000010010)												
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	3	40	258	808	1278	1080	493	120	15	0	0

Table 5.16: C_{59}: An augmentation of (25,11) codes C_{11}, C_{24}, C_{45}, and C_{46}

Contains 3 Design				Group Size: 4608					Dim: 12				
Augmentation of C_{22} by:	(0000000000000000111111000)												
Augmentation of C_{46} by:	(0000000000000000111111000)												
Augmentation of C_{47} by:	(0000000000010010100100000)												
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	3	40	258	808	1278	1080	493	120	15	0	0

Table 5.17: C_{60}: An augmentation of (25,11) codes C_{22}, C_{46}, and C_{47}

Contains 24 Design				Group Size: 192					Dim: 12				
Augmentation of C_{6} by:	(0000000000000110010110010)												
Augmentation of C_{15} by:	(0000000000001100110001100)												
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	2	40	262	808	1272	1080	497	120	14	0	0

Table 5.18: C_{61}: An augmentation of (25,11) codes C_6 and C_{15}

Contains 12 Design				Group Size: 3072					Dim: 12				
Augmentation of C_{18} by:	(0000000000011111101011100)												
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	6	45	250	792	1284	1098	493	112	14	1	0

Table 5.19: C_{62}: An augmentation of (25,11) code C_{18}

Contains 4 Design				Group Size: 1152				Dim: 12					
Augmentation of C_{10} by:				(0000000000010011010010010)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	4	45	258	792	1272	1098	501	112	12	1	0

Table 5.20: C_{63}: An augmentation of (25,11) code C_{10}

Contains 8 Design				Group Size: 256				Dim: 12					
Augmentation of C_7 by:				(0000000000001100100100110)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	3	45	262	792	1266	1098	505	112	11	1	0

Table 5.21: C_{64}: An augmentation of (25,11) code C_7

Contains 52 Design				Group Size: 384				Dim: 12					
Augmentation of C_{23} by:				(0000000000001010001101100)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	3	40	258	808	1278	1080	493	120	15	0	0

Table 5.22: C_{65}: An augmentation of (25,11) code C_{23}

Contains 16 Design				Group Size: 2048				Dim: 12					
Augmentation of C_{16} by:				(0000000000000110001111000)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	5	40	250	808	1290	1080	485	120	17	0	0

Table 5.23: C_{66}: An augmentation of (25,11) code C_{16}

Contains 20 Design				Group Size: 288				Dim: 12					
Augmentation of C_{20} by:				(0000000000001100110001100)									
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	1	40	266	808	1266	1080	501	120	13	0	0

Table 5.24: C_{67}: An augmentation of (25,11) code C_{20}

Contains 3 Design				Group Size: 768					Dim: 12				
Augmentation of C_{21} by:	(0000000000001010001101010)												
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	1	40	266	808	1266	1080	501	120	13	0	0

Table 5.25: C_{68}: An augmentation of (25,11) code C_{21}

Contains 12 Design				Group Size: 12288					Dim: 12				
Augmentation of C_{14} by:	(0000000000001100011011000)												
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	7	40	242	808	1302	1080	477	120	19	0	0

Table 5.26: C_{69}: An augmentation of (25,11) code C_{14}

Contains 12 Design				Group Size: 432					Dim: 12				
Augmentation of C_{12} by:	(0000000000000110010110100)												
Weight Polynomial	0	2	4	6	8	10	12	14	16	18	20	22	24
	1	0	0	40	270	808	1260	1080	505	120	12	0	0

Table 5.27: C_{70}: An augmentation of (25,11) code C_{12}

Code of Dim. 11	Row added to the generator matrix	Augmentation result
C_{17}	(0000000000011110011001010)	C_{36}
C_{18}	(0000000000001101111001100)	C_{36}
C_{53}	(0000000000010100111111101)	C_{29}
C_{23}	(0000000000010100010100000)	C_{40}
C_{16}	(0000000000011001000001000)	C_{40}
C_{24}	(0000000000000001010000110)	C_{42}
C_{22}	(0000000000011110100101010)	C_{42}
C_{21}	(0000000000010100010100000)	C_{42}
C_{14}	(0000000000001100111110010)	C_{42}
C_{15}	(0000000000010010011011110)	C_{42}
C_{23}	(0000000000011110011001100)	C_{44}
C_{16}	(0000000000011111001110000)	C_{44}
C_{11}	(0000000000010101000000100)	C_{44}
C_{20}	(0000000000011110101010010)	C_{44}
C_{15}	(0000000000011110101010010)	C_{44}
C_{12}	(0000000000001010101000000)	C_{44}
C_{6}	(0000000000001010101000000)	C_{44}
C_{14}	(0000000000000000100101010)	C_{32}
C_{21}	(0000000000011110011001010)	C_{37}
C_{12}	(0000000000001100111110100)	C_{37}
C_{6}	(0000000000001100111110010)	C_{37}
C_{8}	(0000000000010010111101100)	C_{43}
C_{13}	(0000000000010010010010000)	C_{43}
C_{18}	(0000000000010010010010000)	C_{43}
C_{10}	(0000000000010010011000000)	C_{43}
C_{17}	(0000000000011000001100000)	C_{43}
C_{7}	(0000000000010111111010101)	C_{43}
C_{8}	(0000000000011110100110100)	C_{41}
C_{5}	(0000000000010100000101011)	C_{41}

Code of Dim. 11	Row added to the generator matrix	Augmentation result
C_9	(0000000000001011001111100)	C_{41}
C_7	(0000000000011011011110011)	C_{41}
C_{53}	(0000000000010100001001011)	C_{41}
C_{19}	(0000000000001101100000000)	C_{38}
C_9	(0000000000001101000000100)	C_{38}

Table 5.28: Augmentation of codes of dimension 11 to codes of dimension 12 from Chapter 3

Code	Dimension	Total # of Designs	Code	Dimension	Total # of Designs
C_1	10	1	C_{36}	12	652
C_2	10	2	C_{37}	12	859
C_3	10	2	C_{38}	12	1049
C_4	10	1	C_{39}	12	1086
C_5	11	6	C_{40}	12	1476
C_6	11	8	C_{41}	12	1848
C_7	11	8	C_{42}	12	2151
C_8	11	8	C_{43}	12	3874
C_9	11	8	C_{44}	12	4470
C_{10}	11	4	C_{45}	11	1
C_{11}	11	12	C_{46}	11	1
C_{12}	11	12	C_{47}	11	1
C_{13}	11	12	C_{48}	11	2
C_{14}	11	12	C_{49}	11	2
C_{15}	11	16	C_{50}	11	2
C_{16}	11	16	C_{51}	11	2
C_{17}	11	16	C_{52}	11	1
C_{18}	11	12	C_{53}	11	1
C_{19}	11	10	C_{54}	11	1
C_{20}	11	20	C_{55}	12	9
C_{21}	11	3	C_{56}	12	18

Code	Dimension	Total # of Designs	Code	Dimension	Total # of Designs
C_{22}	11	3	C_{57}	12	10
C_{23}	11	52	C_{58}	12	24
C_{24}	11	11	C_{59}	12	23
C_{25}	12	4	C_{60}	12	3
C_{26}	12	8	C_{61}	12	24
C_{27}	12	16	C_{62}	12	12
C_{28}	12	26	C_{63}	12	4
C_{29}	12	31	C_{64}	12	8
C_{30}	12	115	C_{65}	12	52
C_{31}	12	120	C_{66}	12	16
C_{32}	12	184	C_{67}	12	20
C_{33}	12	216	C_{68}	12	3
C_{34}	12	292	C_{69}	12	12
C_{35}	12	612	C_{70}	12	12

Table 5.29: Updated values for the number of designs in Codes

Chapter 6

Analysis and Conclusion

In this thesis the relationship between $(16, 6, 3)$-designs and $(25, 12)$ self-orthogonal codes has been found, and in this chapter this relationship will be analyzed. The results may be helpful in searching for new designs in the $(6\lambda-2, 2\lambda, \lambda)$ family of designs. First, the relationship between the smaller cases of $\lambda = 1, 2$ in the $(6\lambda-2, 2\lambda, \lambda)$ family of designs will be discussed, i.e. the $(4, 2, 1)$-design ($\lambda = 1$) and the $(10, 4, 2)$-designs ($\lambda = 2$) will be studied, along with their corresponding self-orthogonal codes, the $(7, 3)$ self-orthogonal codes and the $(15, 7)$ self-orthogonal codes, respectively. Then the distribution of residual $(16, 6, 3)$-designs in different codes will be studied. Finally, the distribution of all the $(16, 6, 3)$-designs in codes will be discussed.

6.1 The Relationship for Smaller Values of λ

There is only one $(4, 2, 1)$ design, which contains all the two subsets of a 4-set [11], which is a residual design. Since λ is odd, a column of 1s should be added to the incidence matrix, which will grant self-orthogonality of the corresponding code. Therefore, the corresponding code is a $(7, 3)$ self-orthogonal code. Pless [17] shows that there are two possible such codes, which have the following weight distributions:

$$W[0] = 1, W[2] = 3, W[4] = 3, W[6] = 1$$

or

$$W[0] = 1, W[4] = 7$$

Considering that $r = 3$ and a 1 is added to each row, the code has to have at least 4 codewords of weight 4. Therefore, the corresponding code will be the code with the second weight distribution.

For the case of $\lambda = 2$, there are three designs, which are all residual designs. Although they generate three codes of dimensions 5, 6, and 7, the codes of dimensions 5 and 6 are embedded in the code with dimension 7. However, if the codes were being searched for designs, Pless [17] shows that there are 10 inequivalent $(15, 7)$ self-orthogonal codes. For a $(10, 15, 6, 4, 2)$-design, using the same reasonings that resulted in Equations 3.1 and 3.4, will result in the following:

$$\sum_{i=0}^{k} a_i = b - 1 \Rightarrow a_0 + a_1 + a_2 + a_3 + a_4 = 14 \tag{6.1}$$

$$\sum_{i=0}^{k} i a_i = k(r-1) \Rightarrow a_1 + 2a_2 + 3a_3 + 4a_4 = 20 \tag{6.2}$$

$$\sum_{i=0}^{k} i^2 a_i = 2\binom{k}{2}(\lambda - 1) + k(r-1) \Rightarrow a_1 + 4a_2 + 9a_3 + 16a_4 = 32 \tag{6.3}$$

Since, for all $i \in \{0, 1, 2, 3, 4\}, a_i \geq 0$, from Equations 6.1 and 6.3 it can be concluded that $a_4 < 2$. Now consider the case that $a_4 = 1$. Suppose that $a_4 = 1$.

Equations 6.2 and 6.3 can be rewritten as:

$$a_1 + 2a_2 + 3a_3 = 16 \tag{6.4}$$

$$a_1 + 4a_2 + 9a_3 = 16 \tag{6.5}$$

Subtracting Equation 6.4 from Equation 6.5 will result in the following.

$$2a_2 + 6a_3 = 0 \Rightarrow a_2 = 0, a_3 = 0 \tag{6.6}$$

From Equations 6.4 and 6.6, it can be concluded that $a_1 = 16$. But it is in contradiction with Equation 6.1. Therefore $a_4 = 0$, which indicates that there are no identical blocks in a $(10, 4, 2)$-design. Using the same method that was used to show that there is no codeword of weight 2 in the corresponding code of a $(16, 6, 3)$-design, it can be concluded that there is no codeword of weight 2 in a code that is generated by a $(10, 4, 2)$-design. This condition will reject 6 of the codes shown by Pless [17]. Two of the remaining codes have no codewords of weight 6, which implies that they do not contain any $(10, 4, 2)$-design. Therefore, it would be only needed to search 2 codes to determine that only the code with the weight distribution below contains $(10, 4, 2)$-designs.

$W[0] = 1, W[2] = 0, W[4] = 9, W[6] = 40, W[8] = 51, W[10] = 24, W[12] = 3, W[14] = 0$

6.2 The Relationship between Residual Designs and Codes

Based on an electronic copy of all 78 of the $(25, 9, 3)$-designs, which Spence kindly supplied to us, all the residual $(16, 6, 3)$-designs were computed. The results of performing the procedure, the same one that was used in Chapters 3 and 5, on the residual designs, shows that all 1281 residual designs enumerated by van Ress [22] are embedded in 12 self-orthogonal codes of dimension 12. The codes with embedded residual design in them are: $C_{28}, C_{29}, C_{31}, C_{33}, C_{37}, C_{38}, C_{39}, C_{40}, C_{41}, C_{42}, C_{43}, C_{44}$. However, these codes have some embedded codes of dimensions 10 and 11. Calculating all the codes of dimension 12 that contain those codes of dimension 10 or 11, and then searching the new codes for embedded designs would result in more designs. Executing this procedure, one can obtain 17883 out of 18920 non-isomorphic $(16, 6, 3)$-designs. More specifically, the resultant designs will be all the designs except for those in codes $C_{25}, C_{26}, C_{27}, C_{30}, C_{34}$, and C_{35}.

6.3 Analysis

Considering the outcome of the previous sections, as well as the results from previous chapters, may be helpful in reinforcing some of methods for searching for designs, and also in finding new approaches for generating new designs. For instance, it is known that for $\lambda = 1, 2$, all the designs are in one code. For the case of $\lambda = 3$, each code which contains a design has, on average, 537 non-isomorphic designs in it, with a minimum of 3 designs and a maximum of 4470 designs. This may be an indication of a method to obtain new designs from known designs, simply by generating the corresponding codes, selecting the codewords of the right weight, and searching those codewords for more embedded designs.

For example, if only the first design found by Bhattacharya [6] was known, computing the code generated by the incidence matrix of the design would show that Bhattacharya's design is embedded in code C_{44}. Searching the corresponding code would result in 4469 more designs in the first step. However, from Table 5.28, it can be seen that $C_6, C_{11}, C_{12}, C_{15}, C_{16}, C_{20}$, and C_{23} are embedded in C_{44}. So, if all the designs in C_{44} are found, some of them generate codes of dimension 11, which are embedded in other codes, such as C_{37}, C_{40}, and C_{42} with 836, 1408, and 2106 new designs, respectively. Repeating this process on these codes will lead to 172 designs in C_{32}.

The previous examples and statistics advert the question that whether or not searching the codes generated by known instances of designs is a doable method of obtaining new designs. For example, for $(28, 10, 5)$-designs ($\lambda = 5$) with 3 known instances [11, p.38] or $(34, 12, 6)$-designs ($\lambda = 6$) with 2 known instances [11, p.40], does searching the codes generated by those known instances result in more non-isomorphic instances of those designs?

Another remark, noticeable in $(4, 2, 1)$-design, $(10, 4, 2)$-designs, and $(16, 6, 3)$-designs, is that the residual designs cover a wide range of codes, from which a large number of designs can be obtained. This observation is important because it raises the question that whether or not, searching the codes, generated by the incident matrices of residual designs, an appropriate method of generating some new quasi-residual designs. This approach can be helpful, as sometimes the symmetric designs are easier to find, or, due to more interest in them, they may be generated before the residual designs. Results might be obtained by searching through the codes corresponding to residual $(46, 16, 8)$-designs or residual $(52, 18, 9)$-designs. The second question, motivated by the above-mentioned observations is on the properties of quasi-residual designs, which are not obtainable from searching the codes of residual designs. In other words, is there any distinction between the quasi-residual designs that are obtainable from residual designs and the quasi-residual designs that are not?

Considering the data in Table 6.1, it can be noticed that the average number of designs per code is relatively high, with respect to the total number of designs. Additionally, considering the fact that Bilous *et al.* [7] searched $(33, 16)$ self-orthogonal codes to prove the non-existence of a $(22, 8, 4)$-design, the question can be asked that is searching the codes an applicable method, in order to enumerate the designs for the larger values of λ? In other words, is it the assumption that the relationship between designs and codes, for larger values of λ, will follow the same pattern, this approach still sounds promising true; and if it is true, is searching the codes still a doable option?

λ	# of Dsgn.s	# of Codes capable of having Dsgn.s	Avg. Dsgn.s per Code
1	1	1	1
2	3	2	$\frac{3}{2}$
3	18920	171	$\frac{19339}{171}\star$

Table 6.1: Average number of designs per code
$\star$ Some of the designs are counted more than once, as they generate codes of smaller dimension and it causes them to appear in more than one code.

It is crucial to note that in order to search for designs in the codes, a specialized clique-finder is needed (a clique is a subgraph, where all pairs of vertices are adjacent). Consider all the codewords of the appropriate weight as vertices in a graph. Define two vertices to be adjacent if and only if their corresponding codewords share exactly λ coordinates where their values are 1, then a clique of order v will be a potential new design. It cannot be expected that a general clique-finder to solve the problem for cases as large as needed. Therefore, the process will not be time efficient. However, a well constructed and specialized clique-finder may execute the process faster than a conventional search, fast enough to find a clique and hence the design. Bilous *et al.* [7] used such a clique-finder to prove the non-existence of $(22, 8, 4)$-designs ($\lambda = 4$).

Finally, one could consider the proportion of the number of codes with known embedded designs to the number of codes with the properties required to contain designs. Since this value seems to be decreasing as the designs become larger, it may be possible that there should be an additional condition on the codes with designs in them. In other words, one may study the properties of the codes which make a code capable of containing designs.

Bibliography

[1] Website: Computer Science CentOS release (http://home.cs.umanitoba.ca/cgi-bin/man?machines) on 01/ 30/ 2014.

[2] Website: Database of self-dual codes (http://www.math.is.tohoku.ac.jp/∼munemasa/selfdualcodes.htm) on 05/ 27/ 2014.

[3] Website: Nauty and Traces (http://cs.anu.edu.au/∼bdm/nauty) on 11/ 13/ 2012.

[4] H. Anton and C. Rorres. *Elementary linear algebra: applications version*. Wiley, 1994.

[5] J. A. Bate, M. Hall Jr., and G. H. J. van Rees. Structures Within $(22, 33, 12, 8, 4)$ Designs. *Journal of Combinatorial Math and Combinatorial Computing*, 4:183–194, 1988.

[6] K. N. Bhattacharya. A new balanced incomplete block design. *Science and Culture*, 9(508):163, 1944.

[7] R. T. Bilous, C. W. H. Lam, L. H. Thiel, P. C. Li, G. H. J. van Rees, S. P. Radzisowski, W. H. Holzmann, and H. Kharaghani. There is no $(22, 8, 4)$ Block Design. *Journal of Combinatorial Designs*, 15:262–267, 2007.

[8] R. T. Bilous and G. H. J. van Rees. An Enumeration of Binary Self-Dual Codes of Length 32. *Codes Cryptography*, 26(1-3):61–86, 2002.

[9] R. T. Bilous and G. H. J. van Rees. Self-Dual Codes and the $(22, 8, 4)$ Balanced Incomplete Block Design. *Jounal of Combinatorial Designs*, 13:363–376, 2005.

[10] I. Bouykukliev. What is Q-Extension? *Serdica Journal of Computing*, vol. 1:115–130.

[11] C. J. Colbourn and J. H. Dinitz, editors. *Handbook of Combinatorial Designs.* Chapman and Hall/CRC, second edition, 2007.

[12] R. H. F. Denniston. Enumeration of symmetric designs (25, 9, 3). *North-Holland Mathematics Studies*, 65:111–127, 1982.

[13] R. A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultureal and Medical Research.* Longman, 1938.

[14] M. Hall Jr., R. Roth, G. H. J. van Rees, and S. A. Vanstone. On Designs $(22, 33, 12, 8, 4)$. *Journal of Combinatorial Theory, Series A*, 47(2):157–175, 1988.

[15] B. D. McKay. Practical Graph Isomorphism. *10th. Manitoba Conference on Numerical Mathematics and Computing*, pages 45–87, 1980.

[16] F. J. McWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes.* North-Holland, 1977.

[17] V. Pless. A Classification of Self-Orthogonal Codes over GF(2). *Discrete Math*, 3:209–246, 1972.

[18] V. S. Pless and W. C. Huffman, editors. *Handbook of Coding Theory.* Elsevier, 1998.

[19] T. Spence. (16,6,3)-designs. Personal Communication.

[20] D. R. Stinson. *Combinatorial Designs: Construction and Analysis.* Springer, 2004.

[21] G. H. J. van Rees. The Investigation of the Structure of a $(22, 33, 12, 8, 4)$-Design, Master Essay, Dept. of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, 1974.

[22] G. H. J. van Rees. All Non-Isomorphic Residual (16,24,9,6,3)-Designs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, pages 183–194, 1988.

[23] S. A. Vanstone and P. C. Van Oorschot. *An Introduction to Error Correcting Codes With Applications.* Springer, 1989.

[24] W. D. Wallis. *Combinatorial Designs*, volume 118 of *Pure and Applied Mathematic, A Series of Monographs and Textbooks.* M. Dekker, 1988.