

Project Number: MA-WJM-6401

# Almost Independent Binary Random Variables

A Major Qualifying Project

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

---

Matthew T. Houde

April 26, 2009

Approved

---

Professor William Martin  
Major Advisor

# Abstract

A collection  $C$  of binary  $n$ -tuples,  $C \subseteq \{0, 1\}^n$ , is considered pairwise independent if for every  $i$  and  $j$  ( $1 \leq i < j \leq n$ ),

$$\Pr_{c \in C}[(c_i, c_j) = \alpha] = 2^{-2}$$

for any  $\alpha \in \{0, 1\}^2$ . Likewise,  $C$  is considered  $t$ -wise independent if the projection onto any  $t$  coordinates is uniformly distributed in  $\{0, 1\}^t$  as  $c$  is chosen uniformly from the sample space  $C$ . Notice that for a collection  $C$  to be  $t$ -wise independent,  $|C|$  must be greater than  $2^t$ ; for certain applications, this is much too large. The aim of this project is to decrease the size of  $|C|$  while still allowing the projection onto any  $t$  coordinates to *appear* uniformly distributed in  $\{0, 1\}^t$ . In this paper we explore several concepts of almost  $t$ -wise independence. For example, we may require that for every  $t$  coordinates  $i_1, \dots, i_t$  and every  $\alpha \in \{0, 1\}^t$ ,

$$|\Pr_{c \in C} [(c_{i_1}, c_{i_2}, \dots, c_{i_t}) = \alpha] - 2^{-t}| \leq \epsilon$$

for some given  $\epsilon > 0$ . I will be presenting two definitions for almost  $t$ -wise independence. Through coding theory tools, such as Delsarte's linear programming bound and sphere-packing bounds, I will show bounds imposed on the size of  $C$  based on those definitions. Through known constructions of almost independent binary random variables, I will then demonstrate how the definitions and bounds I established apply to these pseudo-random collections of binary  $n$ -tuples.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Basics of Coding Theory</b>	<b>7</b>
2.1	Binary $n$ -tuples . . . . .	7
2.2	Inner Distribution . . . . .	8
2.3	Adjacency and Distance Matrices . . . . .	9
2.3.1	Bose-Mesner Algebra . . . . .	13
2.4	Bases and Projectors for Eigenspaces . . . . .	14
2.5	$t$ -wise Independent . . . . .	17
2.6	Delsarte's Theorem . . . . .	17
2.6.1	Delsarte's Linear Programming Bound . . . . .	20
<b>3</b>	<b><math>\epsilon</math>-Almost <math>t</math>-Wise Independence</b>	<b>22</b>
3.1	Delsarte's Theorem Applied to $\epsilon$ -almost $t$ -wise independence . . . . .	22
3.1.1	Delsarte's Linear Programming Bound Applied to $\epsilon$ -almost $t$ -wise independence . . . . .	23
3.1.2	Bound Comparison . . . . .	23
<b>4</b>	<b><math>h</math>-roughly <math>t</math>-wise Independence</b>	<b>26</b>
4.1	Linear Programming Bound for $h$ -roughly $t$ -wise Independence . . . . .	26
4.2	Sphere-packing Bounds on $h$ -roughly $t$ -wise independence . . . . .	27
<b>5</b>	<b>Simple Constructions of almost <math>t</math>-wise independent random variables</b>	<b>31</b>
5.1	The LFSR Construction . . . . .	31
5.2	Quadratic Character Construction . . . . .	31
5.2.1	Evaluated against Delsarte's Theorem . . . . .	32
5.2.2	Evaluated against $h$ -roughly $t$ -wise independent linear program . . . . .	32
<b>6</b>	<b>Conclusion</b>	<b>34</b>
<b>7</b>	<b>References</b>	<b>36</b>
<b>8</b>	<b>Appendix</b>	<b>37</b>
8.1	Maple code . . . . .	37
8.1.1	Delsarte's Linear Programming bound with $t$ -wise independence . . . . .	37
8.1.2	Delsarte's Linear Programming bound with $\epsilon$ -almost $t$ -wise independence . . . . .	37

8.1.3	<i>h</i> -roughly <i>t</i> -wise independent linear programming bound . . . . .	38
8.1.4	LFSR Construction . . . . .	39
8.1.5	Quadratic Character Construction . . . . .	43

# List of Tables

- 2.1 Delsarte’s Linear Programing Bound for  $t$ -wise independence . . . . . 21
- 3.1 Delsarte’s Linear Programing Bound for  $\epsilon$ -Almost  $t$ -wise independence ( $\epsilon = \frac{1}{n}$ ) 24
- 3.2 Bound Comparison . . . . . 24
- 4.1 Linear Programming Bound for  $h$ -roughly  $t$ -wise independent . . . . . 27
- 4.2 Sphere-packing Bound for  $h$ -roughly 3-wise independence . . . . . 30
- 5.1 LFSR constructions evaluated against Delsarte’s Theorem . . . . . 32
- 5.2 Quadratic Character constructions evaluated against Delsarte’s Theorem . . 32
- 5.3 LFSR constructions evaluated against  $h$ -roughly  $t$ -wise independent linear program . . . . . 32
- 5.4 Quadratic Character constructions evaluated against  $h$ -roughly  $t$ -wise independent linear program . . . . . 33

# Chapter 1

## Introduction

From statistical modeling and gambling to the study of disease and even art, random numbers play a key role in a wide range of applications. Many of these applications require collections of independent random numbers of a certain length; however, in order for a collection of random numbers to be considered truly independent each element must occur an equal number of times. Therefore in order to generate a collection of random numbers of length  $k$ , chosen from an alphabet of size  $q$ , the collection must contain at least  $q^k$  elements. For large values of  $k$ , the number of random variables required to generate a truly independent set often make the use of these sets too expensive or too impracticable. Earlier research involving reducing the size of these sample spaces required only limited independence among random variables. In 1990 Joseph Naor and Moni Naor published a paper [3] with the aim of constructing small probability spaces that approximate larger ones. In that paper, Naor and Naor introduced the notion of almost  $t$ -wise independence. Fundamentally with almost  $t$ -wise independence the probability induced on every  $t$ -bit string is close to uniform. This notion allows us to explore the balance between the size of a collection of random variables and how close to independent the collection actually appears. Understanding this balance is an important area of coding theory. The purpose of this paper is to explore ideas of almost  $t$ -wise independence and attempt to establish bounds on the size of a collection of random variables in order to maintain *almost* independence.

In chapter 2 some basic principles of coding theory will be presented which will be used throughout this paper. This chapter will ultimately examine Delsarte's Theorem[2], which can be used to determine the strength of an orthogonal array, and Delsarte's Linear Programming bound, which can be used to determine the minimum number of elements required to create a  $t$ -wise independent set.

In chapter 3 we will explore a definition for almost  $t$ -wise independence which was described in a 1992 paper by Noga Alon, Oded Goldreich, Johan Håstad and René Peralta[1]. By applying the techniques from Delsarte's theorem and Delsarte's linear programming bound, we will then establish a bound for sets of binary random variables based on this definition.

In chapter 4 we will present our own definition for almost  $t$ -wise independence. We

will then construct a linear program bound and a modified sphere-packing bound to establish an upper bound on the length of the binary strings forming a set based on this definition.

In chapter 5 we will explore two simple constructions of almost  $t$ -wise independent random variables. We will generate small almost independent sample spaces based on these constructions and analyze them using the linear programs established on chapters 3 and 4.

# Chapter 2

## Basics of Coding Theory

### 2.1 Binary $n$ -tuples

Throughout this paper we will be working with a collection,  $C$ , of binary  $n$ -tuples (binary strings or vectors of length  $n$ ).

$$C \subseteq \{0, 1\}^n$$

For any element  $x$  in  $\{0, 1\}^n$  such that for  $x = (x_1, x_2, \dots, x_n)$ , we will use  $x_i$  to denote the value of the  $i^{\text{th}}$  entry of the element  $x$ . We will be using the addition operator  $+$  and the scalar multiplication operator  $\times$  such that  $x + y = ((x_1 + y_1) \bmod 2, (x_2 + y_2) \bmod 2, \dots, (x_n + y_n) \bmod 2)$  and  $x \times y = ((x_1 \times y_1) \bmod 2, (x_2 \times y_2) \bmod 2, \dots, (x_n \times y_n) \bmod 2)$ . We will also be introducing two new operators,  $\boxplus$  and  $\boxtimes$ . Where  $\boxplus$  denotes the sum of the entries of the vector produced by the addition of any two elements in  $\{0, 1\}^n$ . Likewise  $\boxtimes$  denotes the sum of the entries of the vector produced by the scalar multiplication of any two elements in  $\{0, 1\}^n$ . More simply put,  $x \boxplus y = \sum_{i=1}^n (x_i + y_i)$  likewise  $x \boxtimes y = \sum_{i=1}^n (x_i \times y_i)$ . Lastly we will be using the operator  $\bar{\phantom{x}}$  to denote the complement of an element. That is for any element  $x$  in  $\{0, 1\}^n$ ,  $\bar{x} = ((x_1 + 1) \bmod 2, (x_2 + 1) \bmod 2, \dots, (x_n + 1) \bmod 2)$ .

Notice that if we take any elements  $x, y$ , and  $z$  in  $\{0, 1\}^n$ ,  $x + y = y + x$ , therefore the commutative law of addition holds. Furthermore  $(x + y) + z = x + (y + z)$ , therefore the associative law of addition holds. There also exists an element  $0$  in  $\{0, 1\}^n$ , (where  $0$  is a vector of length  $n$  consisting of all zeros) such that for any element  $x$ ,  $x + 0 = x$ . We can also see that, for any element  $x$ ,  $x + x = 0$ ; thus there exists an additive inverse under the  $+$  operator. Now if we multiply any element  $x$  by the scalar  $1$  taken from the field  $\{0, 1\}$ , we can see  $1x = x$ . Also if we take any two elements  $a$  and  $b$  from the field  $\{0, 1\}$ , we can see the following three statements hold:

$$(ab)x = a(bx)$$

$$a(x + y) = ax + ay$$

$$(a + b)x = ax + bx$$

With this we have shown that  $\{0, 1\}^n$  is a vector space over the field  $\{0, 1\}$ .

## 2.2 Inner Distribution

The Hamming weight of an element in  $C$ , written  $wt(x)$  for a given element  $x$ , refers to the number of non-zero entries of that element. The Hamming distance,  $d_H(x, y) = |\{i : 1 \leq i \leq n, x_i \neq y_i\}|$ , is the length of shortest path between the vertices  $x$  and  $y$  in the  $n$ -cube. More simply put, the Hamming distance is equal to the number of coordinates in which  $x$  and  $y$  differ. Since we will be working exclusively with binary  $n$ -tuples  $d_H(x, y) = x \boxplus y$ . The minimum distance,  $d(C)$  of  $C$ , is the minimum number of coordinates in which two distinct elements from  $C$  differ. Lastly the maximum distance, or width, is equal to the maximum number of coordinates in which two distinct elements from  $C$  differ. We can now introduce the inner distribution.

**Definition 2.1 (Inner Distribution)** *The inner distribution, or Hamming distance distribution, of the non-empty set  $C$ , is the set  $a_i$ , for all  $i$ ,  $0 \leq i \leq n$  where*

$$a_i = \frac{1}{|C|} |\{(x, y) : x, y \in C, d_H(x, y) = i\}|$$

We also have the following lemma.

**Lemma 2.1**

$$\sum_{i=0}^n a_i = |C|$$

**Proof:** Notice that the set containing all pairs of elements contained in  $C$ ,  $\{(x_i, x_j) : x_i, x_j \in C \forall 0 \leq i, j \leq n\}$ , has cardinality  $|C|^2$ . Furthermore, given any two integers  $i$  and  $j$ , where  $0 \leq i, j \leq n$  and  $i \neq j$ , then  $\{(x, y) : x, y \in C, d_n(x, y) = i\}$  and  $\{(x, y) : x, y \in C, d_n(x, y) = j\}$  are disjoint sets. Lastly notice that we can partition the set  $\{(x_i, x_j) : x_i, x_j \in C \forall 0 \leq i, j \leq n\}$  into  $n+1$  disjoint subsets  $\{(x, y) : x, y \in C, d_n(x, y) = i\}$  for all integers  $i$ ,  $0 \leq i \leq n$ . Thus

$$\begin{aligned} \sum_{i=0}^n a_i &= \frac{1}{|C|} \sum_{i=0}^n |\{(x, y) : x, y \in C, d_h(x, y) = i\}| \\ &= \frac{|C|^2}{|C|} \\ &= |C| \end{aligned}$$

**Done.**

**Example 2.1** *Let  $n = 4$  and let  $C = \{x_1, x_2, x_3, x_4\} = \{001, 010, 100, 111\}$ . If we take  $x_1$  and  $x_2$  we can see the the Hamming distance is*

$$d_H(x_1, x_2) = 001 \boxplus 010 = 2$$

continuing on with this example

$$\begin{aligned}
\{d_H(x, y) = 0\} &= \{(x_1, x_1), (x_2, x_2), (x_3, x_3), (x_4, x_4)\} \\
|\{d_H(x, y) = 0\}| &= 4 \\
\{d_H(x, y) = 1\} &= \emptyset \\
|\{d_H(x, y) = 1\}| &= 0 \\
\{d_H(x, y) = 2\} &= \{(x_1, x_2), (x_1, x_3), (x_1, x_4), (x_2, x_1), (x_2, x_3), (x_2, x_4), \\
&\quad (x_3, x_1), (x_3, x_2), (x_3, x_4), (x_4, x_1), (x_4, x_2), (x_4, x_3)\} \\
|\{d_H(x, y) = 2\}| &= 12 \\
\{d_H(x, y) = 3\} &= \emptyset \\
|\{d_H(x, y) = 3\}| &= 0
\end{aligned}$$

With this we can generate the following inner distribution

$$\begin{aligned}
a_0 &= \frac{1}{|C|} |\{d_H(x, y) = 0\}| = \frac{1}{4} \cdot 4 = 1 \\
a_1 &= \frac{1}{|C|} |\{d_H(x, y) = 1\}| = \frac{1}{4} \cdot 0 = 0 \\
a_2 &= \frac{1}{|C|} |\{d_H(x, y) = 2\}| = \frac{1}{4} \cdot 12 = 3 \\
a_3 &= \frac{1}{|C|} |\{d_H(x, y) = 3\}| = \frac{1}{4} \cdot 0 = 0
\end{aligned}$$

Notice as proved in Lemma 2.1

$$a_0 + a_1 + a_2 + a_3 = 1 + 0 + 3 + 0 = 4 = |C|$$

## 2.3 Adjacency and Distance Matrices

Suppose we have a matrix  $A$  such that every column represents a different vertex on the  $n$ -cube and every row represents a different vertex on the  $n$ -cube. Furthermore we shall arrange the matrix such that every row represents the same vertex on the  $n$ -cube as its corresponding column, and all vertices on the  $n$ -cube are represented. Since there are  $2^n$  distinct vertices on the  $n$ -cube,  $A$  is a matrix of size  $2^n \times 2^n$ . From this we construct the adjacency matrix.

**Definition 2.2 (Adjacency Matrix)** *The adjacency matrix  $A_1$  is a  $2^n \times 2^n$  matrix with entries*

$$(A_1)_{cb} = \begin{cases} 1 & \text{if } d_h(c, b) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where  $c$  is the vertex on the  $n$ -cube corresponding to  $c^{\text{th}}$  column in  $A$  and  $b$  is the vertex on the  $n$ -cube corresponding to the  $b^{\text{th}}$  row in  $A$ .

**Example 2.2** For  $n = 2$  with the fixed ordering of  $n$ -tuples  $\{00, 01, 10, 11\}$ , we get the following adjacency matrix

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

For  $n = 3$  with the fixed ordering of  $n$ -tuples  $\{000, 001, 010, 011, 100, 101, 110, 111\}$  we get the following adjacency matrix

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

Before we go any further it is important that we also understand the concept of a character as it pertains to an abelian group. Given an abelian group  $G$ , a character is any group homomorphism that maps  $G$  to the non-zero complex numbers under complex multiplication.

**Example 2.3** Suppose we take the abelian group  $\mathbb{Z}_n$ , and let  $w$  be a primitive  $n^{\text{th}}$  root of unity

$$w = e^{\frac{2\pi i}{n}}$$

and for  $a$  in  $\mathbb{Z}_n$ ,  $\chi_a$  maps  $G$  to the non zero complex numbers via

$$b \mapsto w^{ab \bmod n}$$

notice

$$\begin{aligned} \chi_a(b+c) &= w^{a(b+c)} \\ &= w^{ab+ac} \\ &= w^{ab}w^{ac} \\ &= \chi_a(b)\chi_a(c) \end{aligned}$$

with this we can see that  $\chi_a$  is a group homomorphism that maps the abelian group  $\mathbb{Z}_n$  to the non-zero complex numbers, therefore  $\chi_a$  is a character for the group  $\mathbb{Z}_n$ .

We will now introduce  $\nu_a$ , a vector of length  $2^n$ ; where for a given  $n$ -tuple  $a$ ,  $\nu_a = \chi_a(b)$ , for a fixed ordering of all  $n$ -tuples  $b$ .  $\chi_a$  being a character of the abelian group  $\{0, 1\}^n$ , that maps an element  $a$ , in  $\{0, 1\}^n$ , to the non-zero complex numbers via the group homomorphism  $b \mapsto (-1)^{a \otimes b}$ . Now, if we multiply  $A_1$  by  $\nu_a$  for a given  $n$ -tuple  $a$ , then for every row  $c$  in  $A_1$  we have

$$(A_1)_c \nu_a = \sum_b A_{cb} \chi_a(b)$$

for all  $n$ -tuples  $b$ . Notice  $(A_1)_{cb} = 1$  if and only if  $c \boxplus b = 1$ , otherwise  $(A_1)_{cb} = 0$ . Also  $c \boxplus b = 1$  if and only if  $b = c + e_i$ . Where  $e_i$  is a string of length  $n$  containing all zeros except a 1 in the  $i^{\text{th}}$  position. Thus

$$\begin{aligned}
(A_1)_c \nu_a &= \sum_{i=1}^n \chi_a(c + e_i) \\
&= \sum_{i=1}^n -1^{a \boxtimes (c + e_i)} = \sum_{i=1}^n (-1^{a \boxtimes c}) (-1^{a \boxtimes e_i}) \\
&= \chi_a(c) \left[ \sum_{i:a_i=1} (-1) + \sum_{i:a_i=0} (1) \right] \\
&= (n - 2wt(a)) \chi_a(c)
\end{aligned}$$

therefore each  $\nu_a$  is an eigenvector for  $A_1$  and if  $wt(a) = m$ , we have

$$A_1 \nu_a = (n - 2m) \nu_a$$

Notice that the maximal eigenspaces for  $A_1$  have dimensions  $\binom{n}{0}, \binom{n}{1}, \dots, \binom{n}{n}$  with associated eigenvalues  $n, n - 2, \dots, n - 2n$ .

**Example 2.4** *If we take  $n = 2$  with fixed ordering of 2-tuples  $\{00, 01, 10, 11\}$ , we get the following values for  $\nu_a$*

$$\begin{aligned}
\nu_{00} &= \begin{bmatrix} \chi_{00}(00) \\ \chi_{00}(01) \\ \chi_{00}(10) \\ \chi_{00}(11) \end{bmatrix} = \begin{bmatrix} -1^{00 \boxtimes 00} \\ -1^{00 \boxtimes 01} \\ -1^{00 \boxtimes 10} \\ -1^{00 \boxtimes 11} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\
\nu_{01} &= \begin{bmatrix} \chi_{01}(00) \\ \chi_{01}(01) \\ \chi_{01}(10) \\ \chi_{01}(11) \end{bmatrix} = \begin{bmatrix} -1^{01 \boxtimes 00} \\ -1^{01 \boxtimes 01} \\ -1^{01 \boxtimes 10} \\ -1^{01 \boxtimes 11} \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \\
\nu_{10} &= \begin{bmatrix} \chi_{10}(00) \\ \chi_{10}(01) \\ \chi_{10}(10) \\ \chi_{10}(11) \end{bmatrix} = \begin{bmatrix} -1^{10 \boxtimes 00} \\ -1^{10 \boxtimes 01} \\ -1^{10 \boxtimes 10} \\ -1^{10 \boxtimes 11} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \\
\nu_{11} &= \begin{bmatrix} \chi_{11}(00) \\ \chi_{11}(01) \\ \chi_{11}(10) \\ \chi_{11}(11) \end{bmatrix} = \begin{bmatrix} -1^{11 \boxtimes 00} \\ -1^{11 \boxtimes 01} \\ -1^{11 \boxtimes 10} \\ -1^{11 \boxtimes 11} \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}
\end{aligned}$$

Now if we multiply  $A_1$  by  $\nu_a$  for each value  $a$  we get the following

$$A_1 \nu_{00} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \quad A_1 \nu_{01} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad A_1 \nu_{10} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad A_1 \nu_{11} = \begin{bmatrix} -2 \\ 2 \\ 2 \\ -2 \end{bmatrix}$$

furthermore notice

$$\begin{aligned}
(2 - 2wt(00))\nu_{00} &= 2 \cdot \nu_{00} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} = A_1\nu_{00} \\
(2 - 2wt(01))\nu_{01} &= 0 \cdot \nu_{01} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = A_1\nu_{01} \\
(2 - 2wt(10))\nu_{10} &= 0 \cdot \nu_{10} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = A_1\nu_{10} \\
(2 - 2wt(11))\nu_{11} &= -2 \cdot \nu_{11} = \begin{bmatrix} -2 \\ 2 \\ 2 \\ -2 \end{bmatrix} = A_1\nu_{11}
\end{aligned}$$

with this we can see  $\nu_{00}, \nu_{01}, \nu_{10}$ , and  $\nu_{11}$ , are eigenvectors of  $A_1$  with  $2, 0, 0$ , and  $-2$  as their respective eigenvalues.

**Definition 2.3 (Distance Matrix)** The distance matrix  $A_j$  is an  $2^n \times 2^n$  matrix with entries

$$(A_j)_{cb} = \begin{cases} 1 & \text{if } d_h(c, b) = j \\ 0 & \text{otherwise} \end{cases}$$

where  $c$  is the coordinate on the  $n$ -cube corresponding to  $c^{\text{th}}$  column of  $A$  and  $b$  is the the coordinate on the  $n$ -cube corresponding to the  $b^{\text{th}}$  row in  $A$ .

**Example 2.5** For  $n = 3$  with the fixed ordering of  $n$ -tuples  $\{000, 001, 010, 011, 100, 101, 110, 111\}$  we get the following 2 distance matrices for  $j = 2$  and  $j = 3$

$$A_2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad A_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

As we did with the adjacency matrix, if we multiply our matrix  $A_j$  by  $\nu_a$ , for every row  $c$  in  $A_j$  we have

$$(A_j)_c \nu_a = \sum_b (A_j)_{cb} \chi_a(b)$$

for all  $n$ -tuples  $b$ . Unlike with the adjacency matrix in which  $(A_1)_{cb} = 1$  if and only if  $c \boxplus b = 1$ , with the distance matrix  $(A_j)_{cb} = 1$  if and only if  $c \boxplus b = j$ . So for any given  $n$ -tuple  $c$ ,  $c \boxplus b = j$  if and only if  $b = c + x_i$ , where  $x_i$  is an  $n$ -tuple of weight  $j$ . There are  $\binom{n}{j}$  distinct  $n$ -tuples,  $x_i$ , of weight  $j$ , thus

$$(A_j)_c \nu_a = \sum_{i=1}^{\binom{n}{j}} \chi_a(c + x_i) = \chi_a(c) \sum_{i=1}^{\binom{n}{j}} (-1)^{a \boxtimes x_i}$$

We know  $a$  is a fixed  $n$ -tuple, and let us assume it has weight  $m$ . Notice that for some position  $k$ ,  $a_k \times x_{i_k} = 1$  if and only if  $a_k = x_{i_k} = 1$ . Now let  $l$  be a non-negative integer such that  $l = a \boxtimes x_i$ , therefore there are exactly  $l$  positions in which  $a_k = x_{i_k} = 1$ . Since there are  $m$  positions in which  $a_k = 1$ , then there are  $\binom{m}{l}$  combinations of the  $l$  positions in  $x_i$  such that  $a_k = x_{i_k} = 1$ . Furthermore since  $a \boxtimes x_i$  is exactly  $l$ , for the remaining  $n - l$  positions if  $x_{i_k} = 1$  then  $a_k = 0$ . Thus with  $x_i$  having weight  $j$ , there are  $j - l$  positions such that  $x_{i_k} = 1$  and  $a_k = 0$ . Since  $a$  has  $n - m$  positions such that  $a_k = 0$ , there are  $\binom{n-m}{j-l}$  combinations of positions  $x_i = 1$  such that  $a_k = 0$ . Hence for a given  $l$ , and a given  $n$ -tuple  $a$  of weight  $m$ , there are  $\binom{m}{l} \binom{n-m}{j-l}$  distinct  $n$ -tuples  $x_i$  of weight  $j$  such that  $a \boxtimes x_i = l$ . Therefore

$$\sum_{i=1}^{\binom{n}{j}} (-1)^{a \boxtimes x_i} = \sum_{l=0}^j (-1)^l \binom{m}{l} \binom{n-m}{j-l}$$

**Definition 2.4 (Krawtchouk Polynomials)** *The Krawtchouk Polynomials are a set of orthogonal polynomials defined as*

$$K_j(m) = \sum_{l=0}^j (-1)^l \binom{m}{l} \binom{n-m}{j-l}$$

With this we can see

$$A_j \nu_a = K_j(m) \nu_a$$

therefore  $\nu_a$  is an eigenvector of  $A_j$ , with the Krawtchouk polynomials as eigenvalues.

### 2.3.1 Bose-Mesner Algebra

If we examine the collection of  $n + 1$  symmetric zero-one matrices  $\{A_0, A_1, \dots, A_n\}$  we can see that  $A_0$  is the identity matrix and  $A_0 + A_1 + \dots, A_n$  forms a  $2^n \times 2^n$  matrix where all entries are 1. With this we can see that this collection is an association scheme as the vector space  $\mathbf{A}$ , spanned by  $\{A_0, A_1, \dots, A_n\}$  over the reals is closed under matrix multiplication. Furthermore we can see that for any two elements  $A_i$  and  $A_j$  in  $\mathbf{A}$ ,  $A_i A_j = A_j A_i$  and for any three elements  $A_i, A_j$ , and  $A_k$  in  $\mathbf{A}$ ,  $A_i (A_j A_k) = (A_i A_j) A_k$ . For these reasons the set  $\mathbf{A}$  forms a commutative and associative algebra over the reals. This algebra is referred to as the Bose-Mesner algebra.

## 2.4 Bases and Projectors for Eigenspaces

Suppose we are given a fixed  $j$ , ( $0 \leq j \leq n$ ) and a fixed ordering of binary  $n$ -tuples of weight  $j : a_1, a_2, \dots, a_{\binom{n}{j}}$ . We shall define  $U_j$  such that

**Definition 2.5** ( $U_j$ )

$$U_j = 2^{-\frac{n}{2}}[\nu_{a_1} \mid \nu_{a_2} \mid \dots \mid \nu_{a_{\binom{n}{j}}}]$$

Thus  $U_j$  has  $\binom{n}{j}$  columns and  $2^n$  rows. Furthermore since  $\chi_a(c) = 1$  if and only if  $(a \boxtimes c) \bmod 2 = 0$  then there are  $2^{n-j} \sum_{k=0}^j \binom{j}{2k} = 2^{\frac{n}{2}}$  distinct  $n$ -tuples  $c$ , such that  $\chi_a(c) = 1$ . Since by definition we multiply each entry in  $U_j$  by  $2^{-\frac{n}{2}}$ , the sum of all the entries in a given column is 1. Notice that if we multiply the transpose of  $U_j$  with itself,  $U_j^\top U_j$ , for any entry  $\langle \chi_a, \chi_b \rangle$  we have

$$\langle \chi_a, \chi_b \rangle = (2^{-\frac{n}{2}})^2 \sum_{c \in \mathbb{Z}_2^n} \chi_a(c) \chi_b(c) = 2^{-n} \sum_{c \in \mathbb{Z}_2^n} (-1)^{(a+b) \boxtimes c}$$

Now if  $a = b$ , then  $a+b$  is the zero vector, thus  $2^{-n} \sum (-1)^0 = 1$ . So if  $a = b$  then  $\langle \chi_a, \chi_b \rangle = 1$ . If  $a \neq b$  then  $a + b$  is a vector with at least 1 non-zero entry, thus

$$|\{c : ((a+b) \boxtimes c) \bmod 2 = 0\}| = |\{c : ((a+b) \boxtimes c) \bmod 2 = 1\}|$$

So for  $a \neq b$

$$\langle \chi_a, \chi_b \rangle = 2^{-n} (|\{c : ((a+b) \boxtimes c) \bmod 2 = 0\}| - |\{c : ((a+b) \boxtimes c) \bmod 2 = 1\}|) = 0$$

Hence

$$U_j^\top U_j = I_{\binom{n}{j}}$$

**Definition 2.6 (Primitive Idempotent( $E_j$ ))**

$$E_j = U_j U_j^\top$$

Notice that since  $U_j$  has dimensions  $2^n \times \binom{n}{j}$  then  $E_j$  must have dimensions  $2^n \times 2^n$ . Furthermore given any entry  $E_{j_{xy}}$  in  $E_j$  we have

$$E_{j_{xy}} = \frac{1}{2^n} (\chi_{a_1}(x) \chi_{a_1}(y) + \chi_{a_2}(x) \chi_{a_2}(y) + \dots + \chi_{a_{\binom{n}{j}}}(x) \chi_{a_{\binom{n}{j}}}(y))$$

Where  $x$  and  $y$  are  $n$ -tuples corresponding to their respective row and column on the distance matrix. Thus

$$E_{j_{xy}} = \frac{1}{2^n} \sum_{i=1}^{\binom{n}{j}} (-1)^{a_i \boxtimes c}$$

Where  $c = x + y$ ,  $c$  having a fixed weight  $m$  and for all  $n$ -tuples  $a$  of weight  $j$ . Therefore

$$E_{j_{xy}} = \frac{1}{2^n} \sum_{l=0}^j (-1)^l \binom{m}{l} \binom{n-m}{j-l} = \frac{a}{2^n} K_j(m)$$

Hence

$$E_j = \frac{1}{2^n} \sum_{m=0}^n (K_j(m) A_m)$$

**Lemma 2.2**  $A_j E_i = K_j(i) E_i$

**Proof:**

$$\begin{aligned} A_j U_i U_i^\top &= K_j(i) U_i U_i^\top \\ A_j U_i &= K_j(i) U_i \end{aligned}$$

Recall  $U_i = \frac{1}{2^n} [V_{a_1} | V_{a_2} | \dots | V_{a_{\binom{n}{j}}}]$  for all distinct  $n$ -tuples  $a_1, a_2, \dots, a_{\binom{n}{j}}$  having weight  $i$ . Hence  $\frac{1}{2^n} [A_j V_{a_1} | A_j V_{a_2} | \dots | A_j V_{a_{\binom{n}{j}}}] = \frac{1}{2^n} [K_j(i) V_{a_1} | K_j(i) V_{a_2} | \dots | K_j(i) V_{a_{\binom{n}{j}}}]$  and as we showed earlier  $A_j V_a = K_j(i) V_a$ , therefore

$$A_j E_j = K_j(i) E_i$$

**Done.**

**Lemma 2.3**  $E_j \cdot E_j = E_j$

**Proof:**

$$\begin{aligned} E_j \cdot E_j &= (U_j U_j^\top)(U_j U_j^\top) \\ &= U_j (U_j^\top U_j) U_j^\top \\ &= U_j (I) U_j^\top \\ &= U_j U_j^\top \\ &= E_j \end{aligned}$$

**Done.**

**Definition 2.7** ( $\delta_{jk}$ )

$$\delta_{jk} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases}$$

**Lemma 2.4**  $E_j E_k = \delta_{jk} E_j$

**Proof:** Let us assume  $j = k$ . Thus  $E_j \cdot E_k = E_j \cdot E_j$  and from lemma 2.3 we can see  $E_j \cdot E_j = E_j$ . Furthermore since  $j = k$ ,  $\delta_{jk} = 1$ , so  $\delta_{jk} \cdot E_j = E_j$ . Therefore  $E_j \cdot E_k = \delta_{jk} E_j$  for  $j = k$ . Now let us assume  $j \neq k$ . Thus  $\delta_{jk} = 0$ , so  $\delta_{jk} \cdot E_j = 0$ . As we proved earlier  $\langle \nu_a, \nu_b \rangle = 0$  for all  $a \neq b$  thus  $U_j^\top U_k = 0$ . Furthermore we know  $E_j \cdot E_k = U_j U_j^\top U_k U_k^\top$  hence  $E_j \cdot E_k = U_j (0) U_k^\top$ . Therefore  $E_j \cdot E_k = 0$ .

**Done.**

**Example 2.6** If we take  $n = 3$  and  $j = 1$  we can construct  $U_j$  as follows

$$U_j = 2^{-\frac{3}{2}}[\nu_{001}|\nu_{010}|\nu_{100}] = 2^{-\frac{3}{2}} \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} = 2^{-\frac{3}{2}} \begin{bmatrix} 1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \\ -1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Notice

$$U_1^\top U_1 = (2^{-\frac{3}{2}})^2 \begin{bmatrix} 8 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$U_1 U_1^\top = E_1 = (2^{-\frac{3}{2}})^2 = \begin{bmatrix} 3 & 1 & 1 & -1 & 1 & -1 & -1 & -3 \\ 1 & 3 & -1 & 1 & -1 & 1 & -3 & -1 \\ 1 & -1 & 3 & 1 & -1 & -3 & 1 & -1 \\ -1 & 1 & 1 & 3 & -3 & -1 & -1 & 1 \\ 1 & -1 & -1 & -3 & 3 & 1 & 1 & -1 \\ -1 & 1 & -3 & -1 & 1 & 3 & -1 & 1 \\ -1 & -3 & 1 & -1 & 1 & -1 & 3 & 1 \\ -3 & -1 & -1 & 1 & -1 & 1 & 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 1 & -1 & 1 & -1 & -1 & -3 \\ 1 & 3 & -1 & 1 & -1 & 1 & -3 & -1 \\ 1 & -1 & 3 & 1 & -1 & -3 & 1 & -1 \\ -1 & 1 & 1 & 3 & -3 & -1 & -1 & 1 \\ 1 & -1 & -1 & -3 & 3 & 1 & 1 & -1 \\ -1 & 1 & -3 & -1 & 1 & 3 & -1 & 1 \\ -1 & -3 & 1 & -1 & 1 & -1 & 3 & 1 \\ -3 & -1 & -1 & 1 & -1 & 1 & 1 & 3 \end{bmatrix}$$

Now if take two  $n$ -tuples  $x = 101$  and  $y = 010$ , where  $x$  and  $y$  correspond to their respective row and column on the distance matrix, we can construct  $E_{j_{xy}}$  as follows

$$E_{j_{(101,010)}} = \frac{1}{2^n} \sum_{l=0}^j (-1)^l \binom{m}{l} \binom{n-m}{j-l}$$

where  $m$  is equal to  $wt(c)$ , for which  $c = x + y = 101 + 010 = 111$ . Therefore  $m = 3$  and we have

$$\begin{aligned} E_{j_{(101,010)}} &= \frac{1}{2^3} \sum_{l=0}^j (-1)^l \binom{3}{l} \binom{3-3}{j-l} \\ &= \frac{1}{2^3} \left( \binom{3}{0} \binom{0}{1} + (-1) \binom{3}{1} \binom{0}{0} \right) \\ &= \frac{1}{2^3} (0 + (-1)(3)) = -\frac{3}{8} \end{aligned}$$

## 2.5 $t$ -wise Independent

A sample space is considered  $t$ -wise independent if the probability distribution on every  $t$  bit location in a string randomly selected from the sample space is uniform. If we are working with strings of binary random numbers in order to achieve  $t$ -wise independence it requires a probability space containing at least  $2^t$  elements.

**Definition 2.8 ( $t$ -wise Independent)** *A sample space  $S_n$  is considered  $t$ -wise independent if when  $X = x_1, \dots, x_n$  is chosen uniformly at random from  $S_n$ , then for any  $t$  positions  $i_1 < i_2 < \dots < i_t$  and any  $t$  bit string  $\alpha$  we have*

$$Pr[x_{i_1}, x_{i_2}, \dots, x_{i_t} = \alpha] = 2^{-t}$$

## 2.6 Delsarte's Theorem

In a paper published in 1973, P. Delsarte [2] described a method for determining the strength of an orthogonal array using a MacWilliams transformation of the inner distribution.

**Theorem 2.1 (Delsarte's Theorem)** *Let  $C$  denote the set of rows of the  $m \times n$  array  $M$ .  $M$  is an orthogonal array of strength  $t$ , if and only if  $b_1 = b_2 = \dots = b_t = 0$  where*

$$b_j = \sum_{i=0}^n a_i k_j(i)$$

for

$$a_i = \frac{1}{|C|} |(x, y) : x, y \in C, d_h(x, y) = i|$$

and

$$k_j(i) = \sum_{l=0}^j (-1)^l \binom{i}{l} \binom{n-i}{j-l}$$

**Proof:** Let  $X$  be a vector of length  $2^n$  such that  $X_b = f$ , where  $f$  is the number of times the  $n$ -tuple corresponding to the  $b^{\text{th}}$  row of the distance matrix occurs in  $C$ . Notice that

$$a_i = \frac{1}{|C|} X^\top A_j X$$

therefore

$$\begin{aligned} b_j &= \sum_{i=0}^n \left( \frac{1}{|C|} X^\top A_j X k_j(i) \right) \\ b_j &= \frac{1}{|C|} \sum_{i=0}^n (X^\top k_j(i) A_j X) \end{aligned}$$

and as we proved earlier

$$E_j = \frac{1}{2^n} \sum_{i=0}^n k_j(i) A_i$$

Hence

$$b_j = \frac{1}{|C|} 2^n X^\top E_j X$$

Thus  $b_j = 0$  if and only if  $X^\top E_j X = 0$ . Furthermore

$$X^\top E_j X = X^\top U_j U_j^\top X = (X^\top U_j)(X^\top U_j)^\top = \|X^\top U_j\|^2$$

Hence  $b_j = 0$  if and only if  $X^\top U_j = 0$ .

$\Rightarrow$  Let us assume  $M$  is an orthogonal array of strength 1.  $X^\top U_1$  is a  $1 \times n$  array with entries  $X^\top \nu_a$ , for all  $n$ -tuples  $a$  where  $wt(a) = 1$ . Since  $M$  is an orthogonal array of strength 1, then for any given column  $j$  in  $M$  there are an equal number of 1's and 0's. Thus for all  $j$ ,  $1 \leq j \leq n$

$$\sum_{c \in C} (-1)^{e_j \boxtimes c} = 0$$

So  $X^\top \nu_a = 0$  for all  $a$ , hence  $X^\top U_1 = 0$ , therefore  $b_1 = 0$ . Thus the theorem holds for  $t = 1$ . Now let us assume  $M$  has strength  $t$  and  $b_1 = b_2 = \dots = b_{t-1} = 0$ . Thus for any  $n$ -tuple  $a$  having weight  $t$

$$\begin{aligned} X^\top \nu_a &= \sum_{c \in C} (-1)^{a \boxtimes c} \\ &= |c : c \in C, (-1)^{a \boxtimes c} = 1| - |c : c \in C, (-1)^{a \boxtimes c} = -1| \end{aligned}$$

Since  $M$  has strength  $t$ , given any  $t$  columns in  $M$  there are an equal number of rows  $c$  with even and odd hamming weights. therefore  $|c : c \in C, (-1)^{a \boxtimes c} = 1| = |c : c \in C, (-1)^{a \boxtimes c} = -1|$ , so  $X^\top \nu_a = 0$ , thus  $X^\top U_t = 0$ , hence  $b_t = 0$ . therefore by induction if  $M$  is an orthogonal array of strength  $t$ , then  $b_1 = b_2 = \dots = b_t = 0$ .

$\Leftarrow$  Let us assume  $b_1 = 0$ , thus  $X^\top U_1 = 0$ . Thus for all  $e_i$ ,  $1 \leq i \leq n$

$$X^\top \nu_{e_i} = \sum_{c \in C} (-1)^{e_i \boxtimes c} = \sum_{i:c_i=1} (-1) + \sum_{i:c_i=0} (1) = 0$$

therefore given any column  $i$  in  $M$  there are an equal number of 0's and 1's. Hence  $M$  is an orthogonal array of strength 1.

Now let us assume  $M$  is an orthogonal array that is at least  $t - 1$ -wise independent, and where  $b_1 = b_2 = \dots b_{t-1} = b_t = 0$ . Let us choose a set  $T$  consisting of all  $t$ -tuples,  $\tau_i$ , formed on any given set of  $t$  columns in  $M$ . Now let  $F$  be the vector of length  $2^t$  consisting of entries  $f_i$ , where  $f_i$  is the number of times the  $t$ -tuple,  $\tau_i$ ,  $1 \leq i \leq 2^t$ , occurs in  $T$ . Furthermore since we can construct a Hamilton circuit on the  $t$ -cube, we shall construct  $F$  such that given any two adjacent  $t$ -tuple,  $\tau_j$  and  $\tau_k$ ,  $k = (j \pm 1) \bmod (2^t + 1)$ . Notice that any 2 adjacent  $t$ -tuples have exactly  $t - 1$  coordinates that are equal. Since we know  $M$  is at least  $t - 1$  wise independent thus  $f_j + f_k = \frac{|C|}{2^{t-1}}$ . Now suppose there exist a  $t$ -tuple,  $\tau_j$ , of even weight that occurs  $\frac{|C|}{2^t} + g$  times in  $T$ , for some integer  $g$ . therefore  $f_j = \frac{|C|}{2^{t-1}} + g$ , thus  $f_{j+1} = \frac{|C|}{2^{t-1}} - g$ . Furthermore we can see that for any  $t$ -tuple  $\tau_i$

$$f_i = \begin{cases} \frac{|C|}{2^{t-1}} + g & \text{if } wt(\tau_i) \text{ is even} \\ \frac{|C|}{2^{t-1}} - g & \text{otherwise} \end{cases}$$

but we know  $b_t = 0$ , so given any set of  $t$  columns in  $M$ , there are an equal number of even and odd  $t$ -tuples thus

$$\begin{aligned} \sum_{i:\text{even}} f_i &= \sum_{i:\text{odd}} f_i \\ \sum_{i:\text{even}} \left( \frac{|C|}{2^{t-1}} + g \right) &= \sum_{i:\text{odd}} \left( \frac{|C|}{2^{t-1}} - g \right) \end{aligned}$$

but this can only occur if  $g = 0$ . So for any set  $T$  consisting of  $t$ -tuples of formed on a set of  $t$  columns in  $M$ , any given  $t$ -tuple  $\tau_j$  in  $T$ ,  $\tau_j$  must occur exactly  $\frac{|C|}{2^t}$  times. therefore by induction  $M$  must be  $t$ -wise independent.

**Done.**

**Example 2.7** Let us use the same collection of 3-tuples,  $C = \{001, 010, 100, 111\}$ , we used in Example 2.1. As we saw in Example 2.1 we get the following inner distribution for  $C$

$$a_0 = 1, a_1 = 0, a_2 = 3, a_3 = 0$$

With this we can construct the values  $b_0, b_1, b_2$  and  $b_3$  as follows

$$\begin{aligned} b_0 &= \sum_{i=0}^3 a_i k_0(i) = \sum_{i=0}^3 a_i \left( \sum_{l=0}^0 (-1)^l \binom{i}{l} \binom{4-i}{0-l} \right) \\ &= a_0(1) + a_1(1) + a_2(1) + a_3(1) \\ &= 1 \cdot (1) + 0 \cdot (1) + 3 \cdot (1) + 0 \cdot (1) = 4 \end{aligned}$$

$$\begin{aligned}
b_1 &= \sum_{i=0}^3 a_i k_1(i) = \sum_{i=0}^n a_i \left( \sum_l^1 (-1)^l \binom{i}{l} \binom{4-i}{1-l} \right) \\
&= a_0(3+0) + a_1(2-1) + a_2(1-2) + a_3(0-3) \\
&= 1 \cdot (3) + 0 \cdot (1) + 3 \cdot (-1) + 0 \cdot (-3) = 0 \\
b_2 &= \sum_{i=0}^3 a_i k_2(i) = \sum_{i=0}^n a_i \left( \sum_l^2 (-1)^l \binom{i}{l} \binom{4-i}{2-l} \right) \\
&= a_0(3+0+0) + a_1(1-2+0) + a_2(0-2+1) + a_3(0+0+3) \\
&= 1 \cdot (3) + 0 \cdot (-1) + 3 \cdot (-1) + 0 \cdot (3) = 0 \\
b_3 &= \sum_{i=0}^3 a_i k_3(i) = \sum_{i=0}^n a_i \left( \sum_l^3 (-1)^l \binom{i}{l} \binom{4-i}{3-l} \right) \\
&= a_0(1+0+0+0) + a_1(0-1+0+0) + \\
&\quad a_2(0+0+1+0) + a_3(0+0+0-1) \\
&= 1 \cdot (1) + 0 \cdot (-1) + 3 \cdot (1) + 0 \cdot (-1) = 4
\end{aligned}$$

Notice that  $b_1 = b_2 = 0$ . So by Delsarte's theorem the collection of 3-tuples  $C$  forms an orthogonal array  $M$  of strength 2. We can verify this is true by examining each pair of columns of the array  $M$  shown below.

$$\begin{bmatrix} 00 \\ 01 \\ 10 \\ 11 \end{bmatrix}, \begin{bmatrix} 10 \\ 10 \\ 00 \\ 11 \end{bmatrix}, \begin{bmatrix} 01 \\ 00 \\ 10 \\ 11 \end{bmatrix}$$

We can see that within each pair of columns, each 2-tuple occurs an equal number of times,  $C$  is therefore 2-wise independent.

### 2.6.1 Delsarte's Linear Programming Bound

Notice that for  $b_0$ ,

$$\begin{aligned}
b_0 &= \sum_{i=0}^n a_i k_0(i) \\
&= \sum_{i=0}^n a_i \\
&= |C|
\end{aligned}$$

With this we can set up a linear programming bound where  $b_0$  is the objective function to be minimized, subject to the constraints  $b_1 = b_2 = \dots = b_t = 0$  and  $b_{t+1} \geq 0, b_{t+2} \geq 0, \dots, b_n \geq 0$ . From this we can determine the minimum number of  $n$ -tuples the set  $C$  can contain while still maintaining  $t$ -wise independence. Table 2.1 displays the minimum number of  $n$ -tuples a  $t$ -wise independent set  $C$  must contain.

$n$	$t$	Minimum $ C $
8	2	10
8	4	64
8	6	112
16	2	18
16	4	154
16	6	1293
32	2	34
32	4	580
32	6	6912
64	2	66
64	4	2146
64	6	48931

Table 2.1: Delsarte's Linear Programming Bound for  $t$ -wise independence

# Chapter 3

## $\epsilon$ -Almost $t$ -Wise Independence

As we showed in chapter 3, if we are working with a collection of binary random numbers, in order to achieve  $t$ -wise independence it requires a probability space containing at least  $2^t$  elements. In 1990 Joseph and Moni Naor presented a paper [3] with the aim of constructing much smaller probability spaces that will behave similarly to probability spaces that are  $t$ -wise independent. In order to achieve this Naor and Naor introduced the notion of almost  $t$ -wise independence. Fundamentally with almost  $t$ -wise independence the probability distribution induced on every  $t$ -bit location is close to uniform. Based on the paper by Naor and Naor, in 1992 Noga Alon, Oded Goldreich, Johan Håstad and René Peralta presented a paper [1] exploring simple constructions of almost  $t$ -wise independent random variables. In that paper they presented the following definition for almost  $t$ -wise independence:

**Definition 3.1 ( $\epsilon$ -almost  $t$ -wise Independent)** *A sample space  $S_n$  is considered  $\epsilon$ -almost  $t$ -wise independent if when  $X = x_1, \dots, x_n$  is chosen uniformly at random from  $S_n$ , then for any  $t$  positions  $i_1 < i_2 < \dots < i_t$  and any  $t$  bit string  $\alpha$  we have*

$$|Pr[x_{i_1}, x_{i_2}, \dots, x_{i_t} = \alpha] - 2^{-t}| \leq \epsilon$$

for some  $\epsilon, \epsilon \geq 0$ .

We will now try to use Delsarte's Linear Programming bound to establish a lower bound in the number of  $n$ -tuples required to establish an  $\epsilon$ -almost  $t$ -wise independent set.

### 3.1 Delsarte's Theorem Applied to $\epsilon$ -almost $t$ -wise independence

Suppose a set of  $n$ -tuples  $C$ , is  $\epsilon$ -almost  $t$ -wise independent. Thus given any element  $x$  in  $C$ , and any set of  $t$  positions  $i_1 \leq i_2 \leq \dots \leq i_t$

$$Pr[x_{i_1}, x_{i_2}, \dots, x_{i_t} = \alpha] - 2^{-t} \leq \epsilon$$

for some  $\epsilon \geq 0$  and any  $t$ -bit string  $\alpha$ . Thus any set of identical  $t$ -tuples, formed on  $t$ -coordinates in  $C$  must contain at least  $|C|(2^{-t} - \epsilon)$  elements and may contain at most

$|C|(2^{-t} + \epsilon)$  elements. Recall  $b_t = \frac{1}{|C|}2^n X^\top E_t X$ . Further more we can see  $X^\top E_t X = (X^\top U_t)(X^\top U_t)^\top = 2^{-n} \sum_{i=1}^{\binom{n}{t}} \Gamma_i^2$ , where  $\Gamma_i = [\text{number of even } t\text{-tuples}] - [\text{number of odd } t\text{-tuples}]$  for a distinct set,  $i$ , of  $t$ -columns in  $C$ . Notice that  $\Gamma_i$  is greatest when each even  $t$ -tuple occurs the maximum number of times and each odd  $t$ -tuple occurs the minimum number of times while still maintaining  $\epsilon$ -almost  $t$ -wise independence. Conversely  $\Gamma_i$  is smallest when each even  $t$ -tuple occurs the minimum number of times and each odd  $t$ -tuple occurs the maximum number of times while still maintaining  $\epsilon$ -almost  $t$ -wise independence. Thus

$$\begin{aligned} |C|(2^{-t} - \epsilon) \cdot 2^{t-1} - |C|(2^{-t} + \epsilon) \cdot 2^{t-1} &\leq \Gamma_i \leq |C|(2^{-t} + \epsilon) \cdot 2^{t-1} - |C|(2^{-t} - \epsilon) \cdot 2^{t-1} \\ -2^t |C| \epsilon &\leq \Gamma_i \leq 2^t |C| \epsilon \end{aligned}$$

so

$$\Gamma_i^2 \leq 2^{2t} |C|^2 \epsilon^2$$

therefore

$$b_t \leq \binom{n}{t} 4^t |C|^2 \epsilon^2$$

But notice that if  $b_t = \binom{n}{t} 4^t |C|^2 \epsilon^2$ , then given any set of  $t$ -tuples formed on  $t$ -columns in  $C$ , either each even  $t$ -tuple occurs  $|C|(2^{-t} + \epsilon)$  times and each odd  $t$ -tuple occurs  $|C|(2^{-t} - \epsilon)$  times or each even  $t$ -tuple occurs  $|C|(2^{-t} - \epsilon)$  times and each odd  $t$ -tuple occurs  $|C|(2^{-t} + \epsilon)$  times. Furthermore since every pair of adjacent  $t$ -tuples consist of an even and odd  $t$ -tuple which have  $t - 1$  coordinates in common, then given any  $t - 1$  columns in  $C$ , each distinct  $t - 1$  tuple occurs exactly  $|C|(2^{-t} - \epsilon) + |C|(2^{-t} + \epsilon) = |C|2^{t-1}$  times. therefore  $C$  is  $t - 1$  wise independent, so  $b_1 = b_2 = \dots = b_{t-1} = 0$ .

### 3.1.1 Delsarte's Linear Programing Bound Applied to $\epsilon$ -almost $t$ -wise independence

As we did with  $t$ -wise independence, we can set up a linear programming bound where  $b_0$  is the objective function to be minimized. However, with  $\epsilon$ -almost  $t$ -wise independence the constraints are now  $b_1 = b_2 = \dots = b_{t-1} = 0$ ,  $b_t \leq \binom{n}{t} 4^t |C|^2 \epsilon^2$  and  $b_t \geq 0, b_{t+1} \geq 0, \dots, b_n \geq 0$ . With this we can determine the minimum number of  $n$ -tuples the set  $C$  can contain while still maintaining  $\epsilon$ -almost  $t$ -wise independence. Table 3.1 displays the minimum number of  $n$ -tuples the set  $\epsilon$ -almost  $t$ -wise independent set  $C$  can contain for  $\epsilon = \frac{1}{n}$ .

### 3.1.2 Bound Comparison

Now if we compare the bound for  $t$ -wise independence with the bound for  $\epsilon$ -almost  $t$ -wise independence (for  $\epsilon = 1/n$ ), the minimum number of  $n$ -tuples contained in  $C$  is obviously smaller with  $\epsilon$ -almost  $t$ -wise independence. Furthermore with Table 3.2 showing consecutive values of  $t$ , we can see that the the minimum number of  $n$ -tuples contained in  $C$  for  $\epsilon$ -almost  $t$ -wise independence is the same as the the minimum number of  $n$ -tuples contained in  $C$  for  $t - 1$ -wise independence.

$n$	$t$	Minimum $ C $
8	2	5
8	4	16
8	6	86
16	2	11
16	4	32
16	6	256
32	2	24
32	4	64
32	6	1158
64	2	54
64	4	128
64	6	4060

Table 3.1: Delsarte's Linear Programming Bound for  $\epsilon$ -Almost  $t$ -wise independence ( $\epsilon = \frac{1}{n}$ )

$n$	$t$	Minimum $ C $ $t$ -wise independence	Minimum $ C $ $\epsilon$ -Almost $t$ -wise independence
8	4	64	16
8	5	86	64
8	6	112	86
16	4	154	32
16	5	256	154
16	6	1293	256
32	4	580	64
32	5	1158	580
32	6	6912	1158
64	4	2146	128
64	5	4060	2146
64	6	48931	4060

Table 3.2: Bound Comparison

This shows for at least the values presented in this table, with our current linear programming bound for  $\epsilon$ -almost  $t$ -wise independence, the minimum number of  $n$ -tuples contained in  $C$  is bounded by the equation  $b_{t-1} = 0$  rather than the equation  $b_t \leq \binom{n}{t} 4^t |C|^2 \epsilon^2$ . Unfortunately with  $\epsilon$ -almost  $t$ -wise independence, Delsarte's linear programming bound does not create an effective bound, as the minimum for  $|C|$  does not decrease uniformly as epsilon goes to 0. For this reason we shall now introduce an alternate definition for almost independence.

# Chapter 4

## $h$ -roughly $t$ -wise Independence

**Definition 4.1** ( *$h$ -roughly  $t$ -wise Independence*) *A sample space  $S_n$  is considered  $h$ -roughly  $t$ -wise independent if when  $X = x_1, \dots, x_n$  is chosen uniformly at random from  $S_n$ , then for any  $t$  positions  $i_1 < i_2 < \dots < i_t$  and any  $t$  bit string  $\alpha$  we have*

$$Pr[x_{i_1}, x_{i_2}, \dots, x_{i_t} = \alpha] \leq \frac{1}{|S_n|} \left\lceil \frac{h|S_n|}{2^t} \right\rceil$$

for some  $h$ ,  $h \geq 1$ .

As with  $\epsilon$ -almost  $t$ -wise independence, with  $h$ -roughly  $t$ -wise independence the probability distribution induced on every  $t$ -bit location is close to uniform. If we examine a sample space  $C$ ,  $C \subseteq \{0, 1\}^n$ , of size  $m$ , if  $C$  is  $t$ -wise independent then given any set of  $t$ -positions in  $C$  each  $t$ -tuple will occur exactly  $\frac{m}{2^t}$  times. If  $C$  is  $h$ -roughly  $t$ -wise independent then given any set of  $t$ -positions in  $C$  no  $t$ -tuple can occur more than  $\lceil \frac{hm}{2^t} \rceil$  times for some value  $h$ ,  $h \geq 1$ . Our goal is to establish an effective upper bound on  $n$ , the length of the binary strings forming an  $h$ -roughly  $t$ -wise independent sample space  $C$ , based on the number of elements in  $C$ .

### 4.1 Linear Programming Bound for $h$ -roughly $t$ -wise Independence

Let us consider an  $h$ -roughly  $t$ -wise independent set of binary  $n$ -tuples  $C$ , having cardinality  $m$ . Let  $V_\lambda$  be a vector of length  $\binom{m}{\lambda}$  consisting of all ordered  $m$ -tuples of weight  $\lambda$ , where  $\lambda = \lceil \frac{hm}{2^t} \rceil + 1$ ; let  $A_m$  be a vector of length  $2^m$  consisting of all ordered  $n$ -tuples; and let  $M$  be an  $\binom{m}{\lambda} \times 2^m$  matrix, with entries:

$$M_{cb} = \begin{cases} 1 & \text{if } (c \boxplus b) \bmod (\lambda) = 0 \\ 0 & \text{otherwise} \end{cases}$$

where  $c$  is the  $c^{th}$  element in  $V_\lambda$  and  $b$  is the  $b^{th}$  element in  $A_m$ .

Now let  $X$  be a vector of length  $2^m$ , with entries  $x_b$ , where  $x_b$  is the number of positions  $i$ ,  $i \leq i \leq n$ , in  $C$  where the  $m$ -tuple formed by the  $i^{th}$  position of all ordered  $n$ -tuples in

$C$  is equal to the  $m$ -tuple corresponding to the  $b^{th}$  element in  $A_m$ . Notice that the sum of entries in  $X$  is equal to  $n$ . Furthermore if we multiply  $M$  by  $X$ , then for every row  $c$  in  $M$ ,  $M_c X$  is equal to  $j$ , the length of a binary string that occurs at least  $\lambda$  times on  $j$  positions in  $C$ . However, since  $C$  is  $h$ -roughly  $t$ -wise independent, then across any  $t$  positions in  $C$  no  $t$ -tuple can occur more than  $\lceil \frac{hm}{2^t} \rceil$  times. Thus for all rows  $c$ ,  $M_c X$  must be less than  $t$ . From this we can establish a linear programming bound subject to the constraints:

$$\begin{aligned} MX &< t \\ X &\geq 0 \end{aligned}$$

with an objective function to be maximized being the sum of entries in  $X$ , thereby establishing an upper bound on the size of  $n$ .

Moreover, we can see that each complementary pair or  $m$ -tuples can be used interchangeably without effecting the value of  $MX$ , so without loss of generality we can re-construct  $A_m$  as a vector of length  $2^{m-1}$  with entries consisting of a single  $m$ -tuple taken from each complimentary pair of  $m$ -tuples. As a result  $M$  becomes a matrix of size  $\binom{m}{\lambda} \times 2^{m-1}$ . We must therefore reconstruct  $X$  such that each entry  $x_b$  is the number of positions  $i$ , in  $C$ , where the  $m$ -tuple formed by the  $i^{th}$  position of all ordered  $n$ -tuples in  $C$  is equal to the  $m$ -tuple, or its complementary pair, corresponding to the  $b^{th}$  element in  $A_m$ .

Table 4.1 displays the maximum size of  $n$ , for 2-roughly  $t$ -wise independent set containing  $|C|$  elements.

$ C $	$t$	maximum size of $n$
4	3	6
4	4	9
4	5	12
6	3	20
6	4	7
6	5	10
8	3	14
8	4	7
8	5	8

Table 4.1: Linear Programming Bound for  $h$ -roughly  $t$ -wise independent

Unfortunately since  $M$  is a matrix of size  $\binom{m}{\lambda} \times 2^{m-1}$ , as  $m$  becomes large the number of computations required to solve this linear programming bound grows exponentially, thereby making this an ineffective method for establishing a bound on  $n$ .

## 4.2 Sphere-packing Bounds on $h$ -roughly $t$ -wise independence

Now let us try a different approach for establishing a bound on  $n$  for  $h$ -roughly  $t$ -wise independence. Consider the set  $C$ , consisting of  $m$  binary  $n$ -tuples. Let us construct the

$m \times n$  matrix  $M$ , where each row represents a different  $n$ -tuple in  $C$ . Now suppose  $C$  is  $h$ -roughly *pair*-wise independent. therefore if we examine any 2 columns in  $M$ , no 2-tuple can occur more than  $\lambda_2$  times, where  $\lambda_2 = \lceil \frac{hm}{4} \rceil$ . Now let  $a$  and  $b$  represent the  $m$ -tuples formed by a given pair of columns in  $M$  and let  $\lambda_{00}, \lambda_{01}, \lambda_{10}$ , and  $\lambda_{11}$  represent the number of times the the 2-tuples 00, 01, 10, and 11 occur respectively across those two columns. Notice that

$$\lambda_{00} + \lambda_{01} + \lambda_{10} + \lambda_{11} = m$$

Furthermore since no 2-tuple can occur more than  $\lambda_2$  times then

$$\lambda_{00} + \lambda_{11} \leq 2\lambda_2$$

therefore

$$d_h(a, b) = \lambda_{01} + \lambda_{10} \geq 2\lambda_2$$

where  $d_h(a, b)$  is the hamming distance between the  $m$ -tuples  $a$  and  $b$ ; however, since  $\lambda_2 \geq \frac{m}{4}$ , this can not be used to form an effective bound on  $n$ .

Now suppose the set  $C$  is  $h$ -roughly 3-wise independent. therefore if we examine any 3 columns in  $M$ , no 3-tuples can occur more than  $\lambda_3$  times, where  $\lambda_3 = \lceil \frac{hm}{8} \rceil$ . Now let  $a, b$ , and  $c$  represent the  $m$ -tuples formed by 3 columns in  $C$  and let  $\lambda_{000}, \lambda_{001}, \lambda_{010}, \lambda_{011}, \lambda_{100}, \lambda_{101}, \lambda_{110}$ , and  $\lambda_{111}$  represent the number of times the 3-tuples 000, 001, 010, 011, 100, 101, 110, and 111 occur respectively across those three columns. Notice that

$$d_h(a, b) = \lambda_{010} + \lambda_{011} + \lambda_{100} + \lambda_{101}$$

$$d_h(a, c) = \lambda_{001} + \lambda_{011} + \lambda_{100} + \lambda_{110}$$

$$d_h(b, c) = \lambda_{001} + \lambda_{010} + \lambda_{101} + \lambda_{110}$$

therefore

$$\begin{aligned} d_h(a, b) + d_h(a, c) + d_h(b, c) &= 2\lambda_{010} + 2\lambda_{011} + 2\lambda_{100} + 2\lambda_{101} + 2\lambda_{001} + \lambda_{110} \\ &= 2m - 2\lambda_{000} - 2\lambda_{111} \end{aligned}$$

but we know that no 3-tuple can occur more than  $\lambda_3$  times, thus

$$d_h(a, b) + d_h(a, c) + d_h(b, c) \geq 2m - 4\lambda_3$$

Now let  $r$  be the smallest possible radius of a ball containing all 3  $m$ -tuples  $a, b$  and  $c$ , centered around some  $m$ -tuple  $d$ . therefore

$$d_h(a, d) \leq r$$

$$d_h(b, d) \leq r$$

$$d_h(c, d) \leq r$$

By the triangle inequality we can see that

$$d_h(a, b) \leq 2r$$

$$d_h(a, c) \leq 2r$$

$$d_h(b, c) \leq 2r$$

Hence

$$\begin{aligned} 6r &\geq d_h(a, b) + d_h(a, c) + d_h(b, c) \\ r &\geq \frac{2m - 4\lambda_3}{6} \end{aligned}$$

Therefore a ball of radius  $\frac{2m-4\lambda_3-1}{6}$ , can contain no more than 2 of the 3  $m$ -tuples  $a, b$ , and  $c$ . Now consider the collection of balls,  $B_r(c)$ , having a radius of size  $r$ ,  $r = \frac{2m-4\lambda_3-1}{6}$ , centered around  $c$ , for all  $m$ -tuples  $c$  formed by columns in  $M$ . As we have shown above, no  $m$ -tuple can be contained in more than 2 of these balls. Thus

$$\sum_{c \in M} |B_r(c)| \leq 2 \cdot 2^m$$

where  $|B_r(c)|$  is the number of  $m$ -tuples contained in the ball  $B_r(c)$ . Since for any given  $m$ -tuple,  $c$ , there are  $\binom{m}{i}$  distinct  $m$ -tuples of distance  $i$  from  $c$ , then there are  $\sum_{i=0}^r \binom{m}{i}$   $m$ -tuples contained in the ball  $B_r(c)$ . therefore

$$\begin{aligned} |M| \cdot \sum_{i=0}^r \binom{m}{i} &\leq 2^{m+1} \\ |M| &\leq \frac{2^{m+1}}{\sum_{i=0}^r \binom{m}{i}} \end{aligned}$$

where  $|M|$  is the number of columns in  $M$ . Recall that the number of columns in  $M$  is equal to  $n$ , the length of the binary strings contained in the  $h$ -roughly 3-wise independent set  $C$ . therefore we have established the following bound:

**Theorem 4.1 (bound for  $h$ -roughly 3-wise independence)** *Let  $C$  be a set of binary  $n$ -tuples. If  $C$  is  $h$ -roughly 3-wise independent then the following equation must hold.*

$$n \leq \frac{2^{m+1}}{\sum_{i=0}^r \binom{m}{i}}$$

$$\begin{aligned} \text{for } r &= \frac{2|C|-4\lambda_3-1}{6} \\ \text{where } \lambda_3 &= \left\lceil \frac{hm}{8} \right\rceil \end{aligned}$$

Table 4.2 displays the maximum size of  $n$ , for  $h$ -roughly 3-wise indepent set containing  $|C|$  elements.

$h$	$ C $	maximum size of $n$
1	4	32
2	4	32
1	8	56
2	8	56
1	16	188
2	16	956
1	32	1,902
2	32	35,375
1	64	164,172
2	64	200,350,791

Table 4.2: Sphere-packing Bound for  $h$ -roughly 3-wise independence

# Chapter 5

## Simple Constructions of almost $t$ -wise independent random variables

We will now examine two methods for construction almost  $t$ -wise independent sets of random variables. Both methods were presented in the 1992 paper Simple Constructions of Almost  $k$ -wise Independent Random Variables by Noga Alon, Oded Goldreich, Johan Håstad and René Peralta [1].

### 5.1 The LFSR Construction

Our first construction is based on linear feed back shift register (LFSR) sequences. With this construction we begin with a start sequence,  $\bar{s} = s_0, s_1, \dots, s_{m-1}$ , a binary string of length  $m$ , and a feedback sequence,  $\bar{f} = f_0, f_1, \dots, f_{m-1}$ , also a binary string of length  $m$ . Where  $f_0 = 1$  and  $f_0 + f_1t + f_2t^2 + \dots + f_{m-1}t^{m-1}$  is an irreducible polynomial. The shift register sequence  $\bar{r} = r_0, r_1, \dots, r_{n-1}$  is generated by the following feedback rule:

$$r_i = \begin{cases} s_i & \text{for } i < m \\ \sum_{j=0}^{m-1} f_j \cdot r_{i-m+j} & \text{for } i \geq m \end{cases}$$

With our constructions we will use the set of all binary strings of length  $m$  as start sequences, and will choose an arbitrary irreducible polynomial for a feedback sequence.

### 5.2 Quadratic Character Construction

Our second construction is based on Weil's Theorem regarding character sums. With this construction we begin with an odd prime number  $p$ . We will generate  $p$  binary strings  $r(x) = r_0(x), r_1(x), \dots, r_{p-1}(x)$ , of length  $n$  via the following equation:

$$r_i(x) = \frac{1 - \chi_p(x+i)}{2}$$

for all  $i = 0, 1, \dots, p-1$  and all  $x = 0, 1, \dots, p-1$ . Where  $\chi_p(y)$ , is 1 if  $y$  is a quadratic residue module  $p$  and -1 otherwise. For  $y$  a multiple of  $p$  we define  $\chi(y) = 0$ .

### 5.2.1 Evaluated against Delsarte's Theorem

Table 5.1 displays the length of the start and feedback sequences  $m$ , the length of the generated sequence  $n$ , the irreducible polynomial being used  $f(t)$ , and the values corresponding to  $b_1, b_2, \dots, b_n$  of Delsarte's Theorem.

$m$	$n$	$f(t)$	$b_0, b_1, \dots, b_n$
3	5	$1 + t$	0, 8, 16, 0, 0
3	6	$1 + t + t^2$	0, 16, 32, 8, 0, 0
4	8	$1 + t + t^3$	0, 48, 0, 176, 0, 16, 0, 0
6	10	$1 + t^4 + t^5$	0, 0, 0, 0, 576, 0, 384, 0, 0, 0, 0

Table 5.1: LFSR constructions evaluated against Delsarte's Theorem

Table 5.2 displays the odd prime number used  $p$  the length of the generated sequence  $n$ , and the values corresponding to  $b_1, b_2, \dots, b_n$  of Delsarte's Theorem.

$p$	$n$	$b_0, b_1, \dots, b_n$
7	11	5, 2, 52, 62, 112, 116, 74, 26, 13
11	16	11, 2, 66, 644, 1942, 4162, 7866, 11716, 12956, 7990, 4470, 1813, 534, 125, 19
13	18	13, 1, 123, 858, 3042, 8622, 18669, 31511, 43717, 49069, 43618, 31536, 18752, 8637, ...

Table 5.2: Quadratic Character constructions evaluated against Delsarte's Theorem

As we expected based on what we proved in section 3, since the distribution of  $b_1, b_2, \dots, b_t$  does not move uniformly towards 0 as  $\epsilon$  gets small, the values received are  $b_1, b_2, \dots, b_t$  are relatively large.

### 5.2.2 Evaluated against $h$ -roughly $t$ -wise independent linear program

Table 5.3 displays the length of the start and feedback sequences  $m$ , the length of the generated sequence  $n$ , the irreducible polynomial being used  $f(t)$ , and the minimum values  $h$  and the maximum values  $t$  such that the generated collection is of random variables is  $h$ -roughly  $t$ -wise independent.

$m$	$n$	$f(t)$	$h$ -roughly $t$ -wise independent
3	4	$1 + t + t^2$	1-roughly 3-wise independent 0.5-roughly 1-wise independent
3	5	$1 + t$	2-roughly 4-wise independent 1-roughly 2-wise independent
3	6	$1 + t + t^2$	4-roughly 5-wise independent 1-roughly 2-wise independent

Table 5.3: LFSR constructions evaluated against  $h$ -roughly  $t$ -wise independent linear program

Table 5.4 displays the odd prime number used  $p$  the length of the generated sequence  $n$ , and the minimum values  $h$  and the maximum values  $t$  such that the generated collection is of random variables is  $h$ -roughly  $t$ -wise independent.

$p$	$n$	$h$ -roughly $t$ -wise independent
7	11	9.1-roughly 6-wise independent 1.1-roughly 2-wise independent
11	16	46.5-roughly 9-wise independent 2.9-roughly 4-wise independent 1.1-roughly 2-wise independent
13	18	314-roughly 12-wise independent 19.7-roughly 7-wise independent 3.7-roughly 4-wise independent 1.2-roughly 2-wise independent

Table 5.4: Quadratic Character constructions evaluated against  $h$ -roughly  $t$ -wise independent linear program

This program was more effective at determining how independent the collection of random variables were compared to Delsarte's Linear Program; however, as the number of elements contained in a set increases the number of computations grows exponentially, making this program ineffective for large sets.

# Chapter 6

## Conclusion

In this paper we have explored two definitions for almost independent sets and examined various methods for establishing bounds imposed on the size of these sets.

With the first definition we examined,  $\epsilon$ -almost  $t$ -wise independence, we ran into some critical shortcomings. As we applied Delsarte's Linear programming bound to an  $\epsilon$ -almost  $t$ -wise independent set, in an attempt to form an effective lower bound on the number of elements contained in such a set, we could see that the resulting bound did not decrease uniformly as  $\epsilon$  went to 0. Furthermore for practical values of  $\epsilon$ , the resulting bound on  $b_t$  was so large the linear program was bounded by  $b_{t-1}$  rather than  $b_t$ . For simplicity we examined the bound imposed on  $b_t$  independently of the other values  $b_1, b_2, \dots, b_{t-1}$ . However, we also showed that  $b_t$  reaches its upper bound only when  $b_1 = b_2 = \dots = b_{t-1} = 0$ . Further research examining the upper bound on  $b_t$  in relationship to  $b_1, b_2, \dots, b_{t-1}$  may be able to provide a more effective bound on the number of elements contained in an  $\epsilon$ -almost  $t$ -wise independent set.

With the second definition we examined,  $h$ -roughly  $t$ -wise independence, we were able to avoid the shortcomings of  $\epsilon$ -almost  $t$ -wise independence by developing a definition that incorporates the variable  $h$ , which is based on the number of elements in the set. In exploring this definition we introduced a new method for examining the elements in a set. Coding theory in general studies the direct relationship between all the elements of a given set. In this paper we study the relationship between the elements in a set indirectly by studying the strings formed on distinct positions across all the ordered elements of a given set. From this we were able to construct a linear programming bound establishing an upper bound on the length of the strings forming as set of a given size. While this linear program was able to establish an effective bound, the number of computations required to execute this program makes it impractical for large sets.

In this paper we also introduced a new approach for sphere packing bounds. Unlike with traditional sphere-packing bounds in which there is a minimum distance between strings in a set, in our approach there was no minimum distance between any two strings but rather a minimum value for the perimeter of a triangle formed from 3 strings in a set. Using this modified sphere-packing bound we were able to form an effective bound on the length of the

strings forming an  $h$ -roughly 3-wise independent set of a given length. Further research into this sphere-packing approach should be able to expand this bound for values of  $t$  greater than 3.

# Chapter 7

## References

- [1] N. Alon, O. Goldreich, J. Håstad and R. Peralta, Simple constructions for almost  $k$ -wise independent random variables, *Random Structures and Algorithms*, Vol. 3 (1992), pp. 289-304.
- [2] P. Delsarte. An Algebraic Approach to the Association Schemes of Coding Theory. *Philips Research Reports*, 10, 1973.
- [3] J. Naor and M. Naor, Small-bias Probability Spaces: Efficient Constructions and Applications, 22nd STOC, 1990, pp. 213-223.

# Chapter 8

## Appendix

### 8.1 Maple code

#### 8.1.1 Delsarte's Linear Programming bound with $t$ -wise independence

```
with(simplex):
with(linalg):
with(Spread):
n := 50:
t := 50:
kraw := proc (j, i)
sum((-1)^l*binomial(i, l)*binomial(n-i, j-l), l = 0 .. min(i, j)) end proc:
A := matrix(1, n+1, [1,(j, i) -> a[i]]):
Q := matrix(n+1, n+1, (i, j) -> kraw(j-1, i-1)):
b := linalg[multiply](A, Q):
obj := b[1, 1]-1:
const := {}:
for j from 2 to t+1 do const := const union{b[1, j] = 0}: od:
for j from t+2 to n+1 do const := const union{b[1, j] >= 0}: od:
minimize(obj, const, NONNEGATIVE):
assign(%):
op(A):
```

#### 8.1.2 Delsarte's Linear Programming bound with $\epsilon$ -almost $t$ -wise independence

```
with(simplex):
with(linalg):
with(Spread):
n := 50:
t := 50:
```

```

kraw := proc (j, i)
sum((-1)^l*binomial(i, l)*binomial(n-i, j-l), l = 0 .. min(i, j)) end proc:
A := matrix(1, n+1, [1,(j, i) -> a[i]]):
Q := matrix(n+1, n+1, (i, j) -> kraw(j-1, i-1)):
b := linalg[multiply](A, Q):
obj := b[1, 1]-1:
const := {}:
for j from 2 to t do const := const union{b[1, j] = 0}: od:
const := const union{b[1, t+1] <= (4^t*(binomial(n, t))(b[1, 1]-1))/n}:
for j from t+2 to n+1 do const := const union{b[1, j] >= 0}: od:
minimize(obj, const, NONNEGATIVE):
assign(%):
op(A):

```

### 8.1.3 $h$ -roughly $t$ -wise independent linear programming bound

```

with(simplex):
with(linalg):
with(Spread):
C := 4:
t := 3:
h := 2:
E := matrix(2^C, C, (j, i) -> 0):
F := matrix(binomial(C, t), C, (j, i) -> 0):
for i from 1 to C do
j := 1:
for k from 1 to 2^(C-i+1) do
for j from j to j+2^(i-1)-1 do
E[j, i] := k mod 2:
od:
od:
od:
i := 1: j := 1: k := 1:
for i from 1 to 2^C do
if sum(E[i, m], m = 1 .. C) = t then
for k from 1 to C do
F[j, k] := E[i, k]:
od:
j := j+1:
fi:
od:
E := matrix(2^(C-1), C, (j, i) -> E[j, i]):
i := 1: j := 1: k := 1:
M := matrix(binomial(C, t), 2^(C-1), (j, i) -> 0):
for i from 1 to binomial(C, t) do

```

```

for j to 2^(C-1) do
if (sum(F[i, m]*E[j, m], m = 1 .. C) mod t = 0 then
  M[i, j] := 1:
fi:
od:
od:
X := matrix(2^(C-1), 1, (j, i) -> x[j]):
b := linalg[multiply](M, X):
i := 1: j := 1: k := 1:
k := matrix(1, 2^(C-1), (j, i) -> 1):
o := linalg[multiply](k, X):
obj := o[1, 1]:
const := {}:
for j from 1 to binomial(C, t) do
const := const union{b[j, 1] <= h}:
od:
maximize(obj, const, NONNEGATIVE);

```

### 8.1.4 LFSR Construction

```

restart;
interface(rtablesize=infinity):
with(linalg):
with(combinat, numbcomb):
n:=4:
p := 14:
f := 3:
S:= Matrix(2^n,n,0):
for i from 1 to n do
for j from (2^(i-1)+1) by 2^(i) to 2^n do
for k from 0 to 2^(i-1)-1 do
S[j+k,i]:= 1:
od:
od:
od:
a:= Matrix(2^n,1,0):
g(x) := 0:
c := 0:
for j from 1 to 2^n do
for i from 1 to n do
g(x) := g(x) + (x^(i-1))*S[j,i]:
od:
if (Irreduc(g(x)) mod 2) = true then
a[j,1] := 1:
c := c + 1:

```

```

fi:
g(x) := 0:
od:

F := Matrix(c, n,0):
d := 1:
for i from 1 to 2^n do
if a[i,1] = 1 then
for k from 1 to n do
F[d,k] := S[i,k]:
od:
d := d+1:
fi:
od:

r := Matrix((2^n),p,0):
a:=1:
for j from 1 to 2^n do
q:= 0:
for k from 1 to n do
r[a,k] := S[j,k]:
od:
for z from n+1 to p do
for w from 1 to n do
q:= F[f,w]*r[a,z-n+w]+q:
od:
r[a,z] := q mod 2:
q := 0:
od:
a:=a+1:
od:
o := (2^n):
repcount :=0:
for j from 1 to o-1 do
for i from j+1 to o do
if i <= o then
for k from 1 to p do
if(r[j,k] = r[i,k]) then
repcount := repcount + 1:
fi:
od:
if(repcount = p) then
o := o-1:
r := delrows(r, i..i):
fi:

```

```

repcount := 0:
fi:
od:
od:
//Evaluated against Delsatre's linear program:
n := p:
p := 0:

innerdist := Matrix(1, n+1, 0):
w := 0:
for i from 1 to p do
for j from i to p do
for k from 1 to n do
w := (r[i, k] + r[j, k]) mod 2 + w:
od:
if (i = j) then
innerdist[1,1+w] := innerdist[1,w+1]+1:
else
innerdist[1,w+1] := innerdist[1,w+1]+2:
fi:
w := 0:
od:
od:

for i from 1 to n+1 do
innerdist[1,i] := innerdist[1,i]/p:
od:

kraw := proc(i,j) RETURN( add( (-1)^l*binomial(j,l)*binomial(n-j,i-1),l=0..min(i,j) ) )
dualdist := matrix(n+1,n+1,(i,j)->kraw(j-1,i-1) ):

b := linalg[multiply](innerdist,dualdist):
print(b):
//Evaluated against the h-roughly t-wise independent program
E := matrix(2^p, p, (j, i) -> 0):
X := matrix(2^(p-1),1, (j,i) -> 0):
T := matrix(1, p-1, (i,j) -> 0):
h := matrix(2, p-1, (j,i) -> 0):

for i from 1 to p do
j := 1:
for k from 1 to 2^(p-i+1) do
for j from j to j+2^(i-1)-1 do
E[j, i] := k mod 2:
od:
od:

```

```

od:
od:
E1 := matrix(2^(p-1), p, (j, i) -> E[j, i]):
for i from 1 to n do
for j from 1 to 2^(p-1) do
x := 0:
for k from 1 to p do
x := x + ((r[k, i]+E1[j, k]) mod 2):
od:
if (x mod p) = 0 then
X[j, 1] := X[j, 1]+1:
fi:
od:
od:

for H from 2 to p do:
F := matrix(binomial(p, H), p, (j, i) -> 0):
l := 1:
for i from 1 to 2^p do
m := 0:
for j from 1 to p do
m := m + E[i, j]:
od:
if (m = H) then
for k from 1 to p do
F[l, k] := E[i, k]:
od:
l := l+1:
fi:
od:

M := matrix(binomial(p, H), 2^(p-1), (j, i) -> 0):
for i from 1 to binomial(p, H) do
for j from 1 to 2^(p-1) do
m := 0:
for k from 1 to p do
m := m + F[i, k]*E1[j, k]:
od:
if (m mod H) = 0 then
M[i, j] := 1:
fi:
od:
od:
t := linalg[multiply](M,X):
for i from 1 to (binomial(p,H)) do

```

```

if T[1,H-1] < t[i,1] then
T[1,H-1] := t[i,1]:
fi:
od:
od:
for i from 1 to (p-1) do
h[1,i] := i*2^(T[1,i])/p:
h[2,i] := (i-1)*2^(T[1,i])/p:
od:
print(T);
print(h);

```

### 8.1.5 Quadratic Character Construction

```

restart:
interface(rtablesize=infinity):
with(numtheory):
with(linalg):
with(combinat, numbcomb):
p := 11:
n := 5:
r := Matrix(p,n,0):
for j from 0 to p-1 do
for i from 0 to n-1 do
if ((i + j) mod p) = 0 then
r[j+1,i+1] := 1:
else
r[j+1,i+1] := (1 - legendre(i+j, p))/2:
fi:
od:
od:
repcount :=0:
for j from 1 to p-1 do
for i from j+1 to p-1 do
for k from 1 to n do
if(r[j,k] = r[i,k]) then
repcount := repcount + 1:
fi:
od:
if(repcount = n) then
p := p-1:
r := delrows(r, i..i):
fi:
repcount := 0:
od:

```

```

od:
//Evaluated against Delsatre's linear program:
innerdist := Matrix(1, n+1, 0):
w := 0:
for i from 1 to p do
for j from i to p do
for k from 1 to n do
w := (r[i, k] + r[j, k]) mod 2 + w:
od:
if (i = j) then
innerdist[1,1+w] := innerdist[1,w+1]+1:
else
innerdist[1,w+1] := innerdist[1,w+1]+2:
fi:
w := 0:
od:
od:

for i from 1 to n+1 do
innerdist[1,i] := innerdist[1,i]/p:
od:

kraw := proc(i,j) RETURN( add( (-1)^l*binomial(j,l)*binomial(n-j,i-1),l=0..min(i,j) ) )
dualdist := matrix(n+1,n+1,(i,j)->kraw(j-1,i-1) ):

b := linalg[multiply](innerdist,dualdist):
print(b):
//Evaluated against the h-roughly t-wise independent program
E := matrix(2^p, p, (j, i) -> 0):
X := matrix(2^(p-1),1, (j,i) -> 0):
T := matrix(1, p-1, (i,j) -> 0):
h := matrix(2, p-1, (j,i) -> 0):

for i from 1 to p do
j := 1:
for k from 1 to 2^(p-i+1) do
for j from j to j+2^(i-1)-1 do
E[j, i] := k mod 2:
od:
od:
od:
E1 := matrix(2^(p-1), p, (j, i) -> E[j, i]):
for i from 1 to n do
for j from 1 to 2^(p-1) do
x := 0:

```

```

for k from 1 to p do
x := x + ((r[k, i]+E1[j, k]) mod 2):
od:
if (x mod p) = 0 then
X[j, 1] := X[j, 1]+1:
fi:
od:
od:

for H from 2 to p do:
F := matrix(binomial(p, H), p, (j, i) -> 0):
l := 1:
for i from 1 to 2^p do
m := 0:
for j from 1 to p do
m := m + E[i,j]:
od:
if (m = H) then
for k from 1 to p do
F[l, k] := E[i, k]:
od:
l := l+1:
fi:
od:

M := matrix(binomial(p, H), 2^(p-1), (j, i) -> 0):
for i from 1 to binomial(p, H) do
for j from 1 to 2^(p-1) do
m := 0:
for k from 1 to p do
m := m + F[i, k]*E1[j, k]:
od:
if (m mod H) = 0 then
M[i, j] := 1:
fi:
od:
od:
t := linalg[multiply](M,X):
for i from 1 to (binomial(p,H)) do
if T[1,H-1] < t[i,1] then
T[1,H-1] := t[i,1]:
fi:
od:
od:
for i from 1 to (p-1) do

```

```
h[1,i] := i*2^(T[1,i])/p:  
h[2,i] := (i-1)*2^(T[1,i])/p:  
od:  
print(T);  
print(h);
```