Almost Perfect Nonlinear functions and related combinatorial structures

by

Mandi S. Maxwell

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Applied Mathematics

Program of Study Committee:
Sung-Yell Song, Co-major Professor
Clifford Bergman, Co-major Professor
Jennifer Davidson
Scott Hansen
Jonathan D. H. Smith

Iowa State University

Ames, Iowa

2005

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Mandi S. Maxwell

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

For the Major Program

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A map $f(x)$ from the finite field $\mathbb{F}_{p^n}$ to itself is said to be differentially k-uniform if k is the maximum number of solutions of the equation $f(x+a) - f(x) = b$ where $a, b \in \mathbb{F}_{p^n}$, $a \neq 0$. In particular, 2-uniform maps over $\mathbb{F}_{2^n}$ are called almost perfect nonlinear (APN) maps. These maps are of interest in cryptography because they offer optimum resistance to linear and differential attacks on certain cryptosystems. They can also be used to construct several combinatorial structures of interest.

In this dissertation, we characterize and classify all known power maps $f(x) = x^d$ over $\mathbb{F}_{2^n}$, which are APN or of low uniformity. We discuss some basic properties of APN maps, collect all known APN power maps, and give a classification of APN power maps up to equivalence. We also give some insight regarding efforts to find other APN functions or prove that others do not exist and classify all power maps according to their degree of uniformity for n up to 13.

In the latter part of this dissertation, through the introduction of an incidence structure, we study how these functions can be used to construct semi-biplanes utilizing the method of Robert S. Coulter and Marie Henderson. We then consider a particular class of APN functions, from which we construct symmetric association schemes of class two and three. Using the result of E. R. van Dam and D. Fon-Der-Flaass, we can see that the relation graphs of some of these association schemes are distance-regular graphs. We discuss the local structure of these distance-regular graphs and characterize them.

INTRODUCTION

0.1 Classification of nonlinear functions

We will denote the finite prime field with p elements by $\mathbb{F}_p$ and we consider maps $f : \mathbb{F}_p^n \to \mathbb{F}_p^n$ defined on the n-dimensional vector space over $\mathbb{F}_p$ and let $N_f(a,b)$ be the number of solutions $x \in \mathbb{F}_p^n$ of the equation $f(x+a) - f(x) = b$, where $a, b \in \mathbb{F}_p^n$. Let

$$\Delta_f = \max\{N_f(a,b) : a, b \in \mathbb{F}_p^n,\ a \neq 0\}.$$

Nyberg [30] defined a map f to be *differentially k-uniform* if $\Delta_f = k$. With $p = 2$, this concept is of interest in cryptography because differential and linear cryptanalysis exploit weaknesses related to the uniformity of the functions used in Data Encryption Standard (DES) like block ciphers.

In $\mathbb{F}_2^n$, the solutions of $f(x+a) - f(x) = b$ come in pairs, x and $x+a$. Thus, 2 is the smallest possible k value; i.e., $\Delta_f \geq 2$. In cryptology, 2-uniform maps are referred to as *almost perfect nonlinear* maps (cf. [29, 14]), whereas in geometry, they are often called *semi-planar* functions [13]. Almost perfect nonlinear functions are of interest in cryptography because they offer optimum resistance to linear and differential attacks on DES-like cryptosystems [4, 9]. We can consider the relationship between the input and output data of an S-box (a round of encryption) for a DES-like cryptosystem to be a map. Let $f : \mathbb{F}_2^r \to \mathbb{F}_2^s$ be such a map that we wish to crypt-analyze. If we use differential cryptanalysis, we will need non-empty sets

$$N_f(a,b) = \{x \in \mathbb{F}_2^r : f(x+a) + f(x) = b\},$$

where $a \in \mathbb{F}_2^r - \{0\}$ and $b \in \mathbb{F}_2^s$. The efficiency of differential cryptanalysis is measured by the cardinality of the set, $|N_f(a,b)|$ and the resistance of the function f, to differential cryptanalysis, can be measured by $\Delta_f = \max_{a \neq 0, b} |N_f(a,b)|$. The lower this value is, the more resistant the function f will be to differential cryptanalysis.

Our purpose in Chapter 1 of this dissertation is to provide a systematic study of almost perfect nonlinear power maps on $\mathbb{F}_2^n$. We begin by reviewing basic terms and general results that will be needed in the following chapters. In Section 1.1 and 1.2, we recall definitions and basic facts about APN functions defined over fields and vector spaces, and then, illustrate some representations of maps over vector spaces and the corresponding finite fields. Section 1.3 contains some specific examples of APN functions over certain binary vector spaces.

In Section 1.4 we will take a slight detour to review the properties of a special class of APN functions known as the Almost Bent (AB) functions. These functions have been studied extensively because they play a role in several topics of information theory such as 'm-sequences' or 'sequences of low correlation' (cf.[21]), 'resilient functions' (cf.[7]) and 'permutations suitable for block ciphers' (cf.[10, 29]). However, we are interested in AB functions because this class of APN functions yields an interesting class of semibiplanes which will be explored in the later chapters following the direction of van Dam and Fon-Der-Flaass [36].

Section 1.5 includes a collection of all known examples of binary APN functions. There are several infinite families of APN power maps which have been discovered (and, indeed, theoretically proven to be APN functions) by many researchers, including Nyberg [30], Beth and Ding [3], Carlet, Charpin, and Zinoviev [9], Dobbertin [14, 17, 16, 15], Helleseth, Rong and Sandberg [22], etc. We will summarize their findings and then classify the known binary APN functions up to equivalence. Our computer search revealed no APN power maps over $\mathbb{F}_{2^n}$ which could not be explained by the theoretical results of the above authors. We include a list of all of the binary APN power maps for $n \leq 13$. Earlier Helleseth, Rong, and Sandberg [22] collected examples of all APN power maps up to $n = 11$. Our list confirms that the list in [22] for $n \leq 11$ is complete.

Section 1.6 focusses on the construction of infinite families of differentially k-uniform power

maps for small values of k and includes a classification of all differentially k-uniform power maps over $\mathbb{F}_2^n$ up to $n = 13$.

0.2 Characterization of associated combinatorial structures

APN functions have also been studied because of their relationship to many interesting combinatorial structures. We will discuss some of these structures in Chapter 2. Coulter and Henderson [11] used APN functions to construct semi-biplanes. Carlet et al. [9] constructed a double-error-correcting binary linear uniformly packed code of length $2^n - 1$ and dimension $2^n - 1 - 2n$ over the binary n-dimensional vector space from a certain AB-function. Using a certain type of APN functions, van Dam and Fon-Der-Flaass [35] were able to generalize the 5-class association schemes constructed by de-Caen and van Dam in [12]. They, van Dam and Fon-Der-Flass, also constructed a distance-regular graph of diameter 3 from an AB function in [36] and Xiang [38] constructed an elementary Hadamard difference set from an AB function. These examples indicate that there is a lot more research to do in this direction. Our aim is to collect, study, and characterize (if possible) new combinatorial structures associated with APN functions.

Given an APN function, we can obtain an incidence structure $S = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ as follows. Let $\mathbb{V} = \mathbb{F}_2^n$, with $n \geq 2$ and let $f : \mathbb{V} \to \mathbb{V}$ be an APN function. Define the incidence structure $S = S(n; f) = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ by

Points: $P = (x, a) \in \mathcal{P} = \mathbb{V} \times \mathbb{V}$

Blocks: $B = [y, b] \in \mathcal{B} = \mathbb{V} \times \mathbb{V}$

Incidence: $((x, a), [y, b]) \in \mathcal{I} \Leftrightarrow a + b = f(x + y)$.

The number of points and blocks of this incidence structure S is $|\mathcal{P}| = |\mathcal{B}| = |\mathbb{V} \times \mathbb{V}| = 2^{2n}$. Coulter and Henderson [11] observed the following properties of $S(n; f)$:

(a) Each block is incident with 2^n points.

(b) Each point lies in 2^n blocks.

(c) It is a self-dual structure.

(d) Every pair of points occurs in 0 or 2 blocks.

(e) Every pair of blocks intersect in 0 or 2 points.

(f) For every point $P \in \mathcal{P}$ there are exactly $2^n(2^n-1)/2$ other points defined by the blocks incident with P.

The properties (a)-(e) make $S(n;f)$ a semi-biplane with parameters $(2^{2n}, 2^n)$ if the incidence structure is connected; otherwise, it consists of two disjoint semi-biplanes with parameters $(2^{2n-1}, 2^n)$. It is known that S is connected if n is odd or if f is a function defined by $f(x) = x^{2^\alpha+1}$ for $(\alpha, n) = 1$ [11].

Suppose $S(n;f) = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ is the semi-biplane with parameters $(2^{2n}, 2^n)$. Then by defining binary relations on $\mathcal{P}$ by

$$(P_1, P_2) \in R_1 \quad \textit{iff } P_1 \text{ and } P_2 \text{ are incident with two common blocks;}$$
$$(P_1, P_2) \in R_2 \quad \textit{iff } P_1 \text{ and } P_2 \text{ are not incident with any common blocks}$$

we can construct the two-class association scheme $\mathcal{X} = \mathcal{X}(n;f) = (\mathcal{P}, \{R_0, R_1, R_2\})$. The first relation graph $\Gamma = \Gamma(n;f) = (\mathcal{P}, R_1)$ and the second relation graph $(\mathcal{P}, R_2)$, the complement Γ^C of Γ, are strongly regular graphs with parameters

$$\Gamma = \mathrm{SRG}(2^{2n},\ 2^{n-1}(2^n-1),\ 2^{n-1}(2^{n-1}-1),\ 2^{n-1}(2^{n-1}-1))$$

and

$$\Gamma^C = \mathrm{SRG}(2^{2n},\ (2^{n-1}+1)(2^n-1),\ (2^{n-2}+1)(2^{n-1}-1),\ 2^{n-1}(2^{n-1}+1)),$$

respectively.

In Chapters 2 and 4 we investigate the incidence structures and association schemes obtained from APN functions. In the latter, we construct three infinite classes of association schemes from a class of APN functions and describe the associated relation graphs. We then characterize the structure of strongly regular graphs of the form $\Gamma(n; x^{2^\alpha+1})$, for $n \geq 3$, by in-

vestigating their local structures. In particular, we will see that $\Gamma(3; x^3) = \text{SRG}(64, 28, 12, 12)$, which is co-spectral with $\Gamma_{OA(4,8)}$ and $\Gamma_{H(8,2)}$, is isomorphic to $\Gamma_{H(8,2)}$ but not $\Gamma_{OA(4,8)}$, where $\Gamma_{OA(4,8)}$ is the graph obtained from the orthogonal array $OA(4, 8)$ and $\Gamma_{H(8,2)}$ is the halved-folded graph of Hamming cube $H(8, 2)$. We will also characterize the structure of the strongly regular graphs $\Gamma(q; x^3) = \text{SRG}(q^2, \frac{q(q-1)}{2}, \frac{q(q-2)}{4}, \frac{q(q-2)}{4})$ for some odd q. Chapter 3 serves as an introduction to distance-regular graphs and association schemes and provides the necessary foundation for the material discussed in Chapter 4.

CHAPTER 1. CLASSIFICATION OF NONLINEAR FUNCTIONS

1.1 Nonlinear functions over $\mathbb{F}_q$

We will begin with an analysis of nonlinear functions. Let $\mathbb{F}_q$ be the finite field with q elements where q is a power of a prime number and consider functions $f : \mathbb{F}_q \to \mathbb{F}_q$. For $a, b \in \mathbb{F}_q$, let $N_f(a,b) = |\{x \in \mathbb{F}_q : f(x+a) - f(x) = b\}|$. Then, we say that a function is *nonlinear* if $\nabla_f(q) = \max\{N_f(a,b) : a, b \in \mathbb{F}_q, a \neq 0\}$ is smaller than q. Let's now consider some properties of this quantity we have just defined.

Proposition 1.1.1. $\sum_{a,b \in \mathbb{F}_q} N_f(a,b) = q^2$

Proof.

$$\sum_{a,b \in \mathbb{F}_q} N_f(a,b) = \sum_{a \in \mathbb{F}_q} \sum_{b \in \mathbb{F}_q} N_f(a,b)$$

$$= \sum_{a \in \mathbb{F}_q} \sum_{b \in \mathbb{F}_q} |\{x \in \mathbb{F}_q : f(x+a) - f(x) = b\}|$$

but $\sum_{b \in \mathbb{F}_q} |\{x \in \mathbb{F}_q : f(x+a) - f(x) = b\}|$ counts all of the elements of $\mathbb{F}_q$

exactly once since every $x \in \mathbb{F}_q$ produces exactly one b, so

$$= \sum_{a \in \mathbb{F}_q} q$$

$$= q^2 \text{ since } |\mathbb{F}_q| = q$$

□

Proposition 1.1.2. $\sum_{a \in \mathbb{F}_q^*, b \in \mathbb{F}_q} N_f(a,b) = q^2 - q$

Proof. Let $a = 0$, then $N_f(a,b) = N_f(0,b) = \begin{cases} q & \text{if } b = 0 \\ 0 & \text{otherwise} \end{cases}$ since $|\mathbb{F}_q| = q$, so we must subtract off the q solutions corresponding to $a = 0$. □

Proposition 1.1.3. If f is linear, then $\nabla_f(q) = q$.

Proof. f is linear implies that $f(x) = \alpha x + \beta$. Then,

$$\begin{aligned} |\{x \in \mathbb{F}_q : f(x+a) - f(x) = b\}| &= |\{x \in \mathbb{F}_q : \alpha(x+a) + \beta - (\alpha x + \beta) = b\}| \\ &= |\{x \in \mathbb{F}_q : \alpha a = b\}| \\ &\leq q, \text{ with equality when } \alpha a = b \end{aligned}$$

□

Definition 1.1.4. For q odd, a function f with $N_f(a,b) = 1$ for all $a \in \mathbb{F}_q^*, b \in \mathbb{F}_q$, is called a perfect nonlinear function.

In a sense, these functions are as nonlinear as possible since $\nabla_f(q) = 1$, whereas for linear functions, $\nabla_f(q) = q$.

Definition 1.1.5. For q even, $\nabla_f(q) \geq 2$ and those functions for which $\nabla_f(q) = 2$ are called almost perfect nonlinear.

Let us pause briefly to analyze why we call functions that have this property Almost Perfect Nonlinear. Notice that if x is a solution of $f(x+a) + f(x) = b$, then so is $x + a$. This is where the almost perfect part of the definition comes in to play. We would consider the function to be perfect if the maximum number of solutions was one, but because of this pairing of the solutions, having a maximum of 2 solutions is the best that we can do over fields with even characteristic. Now for the nonlinear part. Notice, if f were linear, then $f(x+a) = f(x)+f(a)$, so $f(x) + f(x+a) = f(x) + f(x) + f(a) = f(a)$ so for $b = f(a)$, every vector x would satisfy the equation. Thus, by our definition, f must be nonlinear.

As previously mentioned, we will restrict ourselves to $q = 2$ and focus our study on the almost perfect nonlinear functions that exist in this case.

1.2 Binary APN functions

We are most interested in binary Almost Perfect Nonlinear functions. We are motivated to consider APN functions on the vector space $\mathbb{F}_2^n$ because of the resistance that they offer against certain types of crypt attacks. As we will see, however, these APN functions over the vector space $\mathbb{F}_2^n$ become very complex as n increases. Therefore, we will begin our treatment of APN functions by studying those expressed over the finite field $\mathbb{F}_{2^n}$. We will then define a correspondence between APN functions on the finite field $\mathbb{F}_{2^n}$ and the vector space $\mathbb{F}_2^n$ and perform some analysis of the functions on the vector space.

1.2.1 APN functions over $\mathbb{F}_{2^n}$

Definition 1.2.1. Let $\mathbb{F}_{2^n}$ with $q = 2^n = |\mathbb{F}_{2^n}|$ and let $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be any function. For $0 \neq a \in \mathbb{F}_{2^n}$, let

$$H_a = H_a(f) = \{f(x) + f(x+a) : x \in \mathbb{F}_{2^n}\}.$$

Then, f is APN (almost perfect nonlinear) if $|H_a(f)| = \frac{1}{2}q$ for all $0 \neq a \in \mathbb{F}_{2^n}$.

An alternate definition of APN functions is:

Definition 1.2.2. [36] A function f is APN if and only if the system of equations

$$\begin{cases} x + y &= a \\ f(x) + f(y) &= b \end{cases} \tag{1.1}$$

has 0 or 2 solutions (x, y) for every $(a, b) \neq (0, 0)$.

The equivalence of these definitions is given in the following Corollary.

Corollary 1.2.3. If (1.1) has 0 or 2 solutions, (x, y), for all $(a, b) \neq (0, 0)$, then the system has 2 solutions precisely when $b \in H_a(f)$.

Proof. ($\Rightarrow$): Assume $|H_a(f)| = \frac{1}{2}q$. Show the system (1.1) has 0 or 2 solutions.

$$|H_a(f)| = \frac{1}{2}q \Rightarrow |\{f(x) + f(x+a) : x \in \mathbb{F}_{2^n}\}| = \frac{1}{2}|\mathbb{F}_{2^n}|$$

Since the pair x and $x+a$ gives the same value for $f(x)+f(x+a)$, for all x with $f(x)+f(x+a) = b$, there exists an $x' = x+a$ such that $f(x')+f(x'+a) = b$. But $x+x' = a$ and $f(x)+f(x') = b$ so the pair x and x' solves the system of equations. Therefore, the system of equations can have 2 solutions, but since $|H_a(f)| = \frac{1}{2}|\mathbb{F}_{2^n}|$ the system cannot have more than 2 solutions. Thus, since solutions must occur in pairs, the system (1.1) has 0 or 2 solutions.

($\Leftarrow$): Assume the system (1.1) has 0 or 2 solutions. Show that $|H_a(f)| = \frac{1}{2}q$.

The system (1.1) has 0 or 2 solutions

$$\Rightarrow \text{ for all } b \in \mathbb{F}_{2^n}, \text{ either } f(x) + f(x+a) = b \text{ for two } x \in \mathbb{F}_{2^n} \text{ or for no } x \in \mathbb{F}_{2^n}.$$

$$\Rightarrow |\{f(x) + f(x+a) : x \in \mathbb{F}_{2^n}\}| \leq \frac{1}{2}|\mathbb{F}_{2^n}|.$$

But $b \in \mathbb{V}$ and $f(x) + f(x+a) = b$

$$\Rightarrow |\{f(x) + f(x+a) : x \in \mathbb{F}_{2^n}\}| \geq \frac{1}{2}|\mathbb{F}_{2^n}|$$

$$\Rightarrow |\{f(x) + f(x+a) : x \in \mathbb{F}_{2^n}\}| = \frac{1}{2}|\mathbb{F}_{2^n}|.$$

$\therefore |H_a(f)| = \frac{1}{2}q.$

□

As we have noted, it is easiest to characterize APN functions over the finite field $\mathbb{F}_{2^n}$, but because of the cryptographic implications, is useful to work with these functions on the corresponding vector space $\mathbb{F}_2^n$. So, we will now demonstrate the correspondence between functions in the finite field and functions in the vector space.

Theorem 1.2.4. Let $\mathbb{V}$ be the standard vector space of row vectors $(x_1, \ldots, x_n)$, $x_i \in \mathbb{F}_2$. Any function $f : \mathbb{V} \to \mathbb{V}$ can be represented as a polynomial in the variables $x_1, \ldots, x_n$ with coefficients in $\mathbb{V}$, and since $x^2 = x$ in $\mathbb{F}_2$, all monomials can be chosen to have degree at most 1 in each variable. So, the polynomial representation of f is unique and can be found by

expanding the representation

$$f(x_1,\ldots,x_n) = \sum_{(a_1,\ldots,a_n)\in\mathbb{V}} f(a_1,\ldots,a_n)\cdot(x_1+a_1+1)\cdots(x_n+a_n+1). \tag{1.2}$$

Proof. The reason why this relationship holds is due to the fact that in the binary space $\mathbb{V}$,

$$\prod_{i=1}^{n}(x_i+a_i+1) = \begin{cases} 1 & \text{if } x_i = a_i \text{ for all } i \\ 0 & \text{if } x_i \neq a_i \text{ for any } i \in \{1,2,3,\ldots,n\} \end{cases}$$

□

Let us consider an example.

Example 1.2.5. Let $F(x) = x^5$ over $\mathbb{F}_{2^3}$ and let α be a primitive element in $\mathbb{F}_{2^3}$. Then, we have

$$\mathbb{F}_{2^3} = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$$

which can be represented, using the irreducible polynomial, $\alpha^3 + \alpha + 1 = 0$, as

$$\mathbb{F}_{2^3} = \{0, 1, \alpha, \alpha^2, \alpha+1, \alpha^2+\alpha, \alpha^2+\alpha+1, \alpha^2+1\}.$$

Then the function $F(x) = x^5$ gives the permutation of the elements of $\mathbb{F}_{2^3}$ shown in Table 1.1.

Table 1.1 Permutation for $F(x) = x^5$

x	$F(x) = x^5$
0	0
1	1
α	$\alpha^2+\alpha+1$
α^2	$\alpha+1$
$\alpha^3 = \alpha+1$	α
$\alpha^4 = \alpha^2+\alpha$	α^2+1
$\alpha^5 = \alpha^2+\alpha+1$	$\alpha^2+\alpha$
$\alpha^6 = \alpha^2+1$	α^2

Now, since for all $x \in \mathbb{F}_{2^3}$, we have $x = \sum_{i=1}^{3} a_i \alpha^{3-i}$, we define $a \in \mathbb{F}_2^3$ as $a = (a_1, a_2, a_3)$. Using this correspondence, then, we obtain the mapping of the elements of $\mathbb{F}_2^3$ shown in Table 1.2.

Table 1.2 Mapping of $F(x) = x^5$ over $\mathbb{F}_2^3$

$a = (a_1, a_2, a_3)$	$f(a)$
$(0,0,0)$	$(0,0,0)$
$(0,0,1)$	$(0,0,1)$
$(0,1,0)$	$(1,1,1)$
$(0,1,1)$	$(0,1,0)$
$(1,0,0)$	$(0,1,1)$
$(1,0,1)$	$(1,0,0)$
$(1,1,0)$	$(1,0,1)$
$(1,1,1)$	$(1,1,0)$

We can then use these to find the explicit formula for f over $\mathbb{F}_2^3$. So, we now have:

$$
\begin{aligned}
f(x_1,x_2,x_3) &= (0,\ 0,\ 0)(x_1+1)(x_2+1)(x_3+1) + (0,\ 0,\ 1)(x_1+1)(x_2+1)x_3 \\
&\quad + (1,\ 1,\ 1)(x_1+1)(x_2)(x_3+1) + (0,\ 1,\ 0)(x_1+1)x_2x_3 \\
&\quad + (0,\ 1,\ 1)x_1(x_2+1)(x_3+1) + (1,\ 0,\ 0)x_1(x_2+1)x_3 \\
&\quad + (1,\ 0,\ 1)x_1x_2(x_3+1) + (1,\ 1,\ 0)x_1x_2x_3 \\
&= (0,\ 0,\ 0) + (0,\ 0,\ x_1x_2x_3 + x_1x_3 + x_2x_3 + x_3) \\
&\quad + (x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2,\ x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2, \\
&\qquad\qquad x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2) \\
&\quad + (0,\ x_1x_2x_3 + x_2x_3,\ 0) \\
&\quad + (0,\ x_1x_2x_3 + x_1x_2 + x_1x_3 + x_1,\ x_1x_2x_3 + x_1x_2 + x_1x_3 + x_1) \\
&\quad + (x_1x_2x_3 + x_1x_3,\ 0,\ 0) \\
&\quad + (x_1x_2x_3 + x_1x_2,\ 0,\ x_1x_2x_3 + x_1x_2) \\
&\quad + (x_1x_2x_3,\ x_1x_2x_3,\ 0) \\
&= (x_2 + x_1x_3 + x_2x_3,\ x_1 + x_2 + x_1x_3,\ x_1 + x_2 + x_3 + x_1x_2)
\end{aligned}
$$

So, $f(x_1, x_2, x_3) = (f_1, f_2, f_3)$ given by:

$$f_1(x_1, x_2, x_3) = x_2 + x_1x_3 + x_2x_3$$

$$f_2(x_1, x_2, x_3) = x_1 + x_2 + x_1x_3$$

$$f_3(x_1, x_2, x_3) = x_1 + x_2 + x_3 + x_1x_2$$

is the function over $\mathbb{F}_2^3$ which corresponds to $F(x) = x^5$ over $\mathbb{F}_{2^3}$.

A special case of these functions would be APN permutations. It is important to note that such permutations can only occur for $\mathbb{F}_{2^n}$ where n is odd. In the case where n is even, we can easily observe that function f maps three-to-one with the exception of 0 which maps to itself. This follows since $f(x) = x^3$ is the only APN power function on $\mathbb{F}_{2^2}$ and it maps onto $\mathbb{F}_2$ (i.e. 0 maps to 0 and everything else maps to 1). Extending this to larger fields, we have that $n = 2m$ but our exponents are determined $mod(2^n - 1) \equiv mod(4^m - 1)$ so clearly the functions will map three-to-one. We will leave the study of APN permutations to the next section where we will consider APN functions over the vector space $\mathbb{F}_2^n$.

1.2.2 APN functions over $\mathbb{F}_2^n$

As we have seen using Theorem 1.2.4, given a function over the finite field $\mathbb{F}_{2^n}$, we can find a corresponding function over the vector space $\mathbb{F}_2^n$. Our definitions for APN functions over the vector space will likewise be analogous to those over the finite field, and we have:

Definition 1.2.6. [36] Let $\mathbb{V}$ be an n-dimensional vector space over the field $\mathbb{F}_2$ with $q = 2^n = |\mathbb{V}|$ and let $f : \mathbb{V} \to \mathbb{V}$ be any function. For $0 \neq a \in \mathbb{V}$, let

$$H_a = H_a(f) = \{f(x) + f(x + a) : x \in \mathbb{V}\}.$$

Then, f is APN (almost perfect nonlinear) if $|H_a(f)| = \frac{1}{2}q$ for all $0 \neq a \in \mathbb{V}$.

as well as:

Definition 1.2.7. [36] A function f is APN if and only if the system of equations

$$\begin{cases} x+y &= a \\ f(x)+f(y) &= b \end{cases} \tag{1.3}$$

has 0 or 2 solutions (x,y) for every $(a,b) \neq (0,0)$.

What we would most like to study on the vector space, however, are those functions which correspond to APN permutations.

Theorem 1.2.8. [3] Let $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))$ be an APN permutation on $\mathbb{V}$, $x = (x_1, x_2, \ldots, x_n) \in \mathbb{V}$. Then none of $f_1, f_2, \ldots, f_n$ is affine.

Proof. Suppose that one of the component functions, *wlog*, f_1 is affine. Then, $f_1(x)$ can be written as a linear combination of the x_i's plus a constant term. In other words, $f_1(x) = b_{1n}x_n + \cdots + b_{11}x_1 + b_0$. Then

$$\begin{aligned} f_1(x) + f_1(x+c) &= b_{1n}x_n + \cdots + b_{11}x_1 + b_0 + b_{1n}(x_n + c_n) + \cdots + b_{11}(x_1 + c_1) + b_0 \\ &= b_{1n}x_n + b_{1n}x_n + \cdots + b_{11}x_1 + b_{11}x_1 + b_0 + b_0 + b_{1n}c_n + \cdots + b_{11}c_1 \\ &= b_{1n}c_n + \cdots + b_{11}c_1 \\ &= \sum_{i=1}^{n} b_{1i}c_i \end{aligned}$$

So we can find a vector $c \neq 0$, such that $f_1(x) + f_1(x+c) = 0$. But then, $f(x) + f(x+c) = (0, f_2(x)+f_2(x+c), \ldots, f_n(x)+f_n(x+c))$. Now, in order for f to be APN, we must have that $f(x)+f(x+c)$ takes on 2^{n-1} (i.e. $\frac{1}{2}|\mathbb{V}|$) vectors of $\mathbb{F}_2^n$, but since the first component is now fixed at 0, in order for this to hold, there must exist a vector x such that $f(x)+f(x+c) = (0, \ldots, 0)$ which contradicts the one-to-one property of $f(x)$. Therefore, each component function of an APN permutation cannot be affine. □

Theorem 1.2.9. [3] Let $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))$ be an APN permutation on $\mathbb{V}$, $x = (x_1, x_2, \ldots, x_n) \in \mathbb{V}$. Then every quadratic term $x_i x_j$ $(i \neq j)$ must appear in at least one of the component functions $f_1, f_2, \ldots, f_n$.

Proof. If $f : \mathbb{V} \to \mathbb{V}$ is an APN function, then in the representation

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) &= (f_1(x_1, \ldots, x_n), f_2(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)) \\ &= \sum_{(a_1, a_2, \ldots, a_n) \in \mathbb{V}} \left(\prod_{i=1}^{n} (x_i + a_i + 1) \right) f(a_1, a_2, \ldots, a_n) \end{aligned}$$

for each $x_i x_j$, there exists a $k \in \{1, 2, \ldots, n\}$ such that $f_k(x_1, x_2, \ldots, x_n)$ contains the term $x_i x_j$. To see this, notice that for any i, j the $x_i x_j$ term appears in the expansion of $\prod_{l=1}^{n} (x_l + a_l + 1)$ only when $a_l = 0$ for all l except i and j. In other words, if we consider the expansion of $f(x_1, x_2, \ldots, x_n) = \sum_{(a_1, a_2, \ldots, a_n) \in \mathbb{V}} \left(\prod_{l=1}^{n} (x_l + a_l + 1) \right) f(a_1, a_2, \ldots, a_n)$, the only components which have an $x_i x_j$ term are those whose corresponding components in the following sum are nonzero. However,

$$\begin{aligned} & f(0, \ldots, 0, \quad 0, \quad 0, \ldots, 0, \quad 0, \quad 0, \ldots, 0) \\ + \; & f(0, \ldots, 0, \quad 1, \quad 0, \ldots, 0, \quad 0, \quad 0, \ldots, 0) \\ + \; & f(0, \ldots, 0, \quad 0, \quad 0, \ldots, 0, \quad 1, \quad 0, \ldots, 0) \\ + \; & f(0, \ldots, 0, \quad \underset{\substack{\uparrow \\ i^{th}}}{1}, \quad 0, \ldots, 0, \quad \underset{\substack{\uparrow \\ j^{th}}}{1}, \quad 0, \ldots, 0) \end{aligned}$$

$\neq 0$ by the definition of APN permutations.

□

Theorem 1.2.10. [3] Let $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))$ be an APN permutation on $\mathbb{V}$, $x = (x_1, x_2, \ldots, x_n) \in \mathbb{V}$. Then the term $\prod_{i=1}^{n} x_i$ never appears in any component function.

Proof. Consider our conversion formula (1.2). Each component of the expansion of $\prod_{i=1}^{n} (x_i + a_i + 1)$ contains the term $\prod_{i=1}^{n} x_i$. But f is a permutation of the elements of $\mathbb{F}_2^n$ so for each f_k in $f(x_1, \ldots, x_n) = (f_1, \ldots, f_n)$ it will take on the value 1 an even number of times. Thus, for each f_k, the term $\prod_{i=1}^{n} x_i$ is added an even number of times which is equivalent to zero. □

Let us now consider some examples of APN functions over both the finite field $\mathbb{F}_{2^n}$ and the corresponding vector space $\mathbb{F}_2^n$.

1.3 Examples of APN functions

Example 1.3.1. The function $F(x) = x^5$ over $\mathbb{F}_{2^3}$ (Welch's function) and the corresponding function $f(x_1, x_2, x_3) = (f_1, f_2, f_3)$ over $\mathbb{F}_2^3$ given by:

$$f_1(x_1, x_2, x_3) = x_2 + x_1x_3 + x_2x_3$$

$$f_2(x_1, x_2, x_3) = x_1 + x_2 + x_1x_3$$

$$f_3(x_1, x_2, x_3) = x_1 + x_2 + x_3 + x_1x_2$$

is APN.

We can see that this function is APN since, for all $a \in \mathbb{F}_2^3$ such that $a \neq 0$, we can find $H_a(f)$ and see that $|H_a(f)| = \frac{1}{2}q$ so the function is APN. (Note: This is done using the Matlab program (B.1) which, given a function f defined on $\mathbb{F}_2^n$, finds $H_a(f)$ and $|H_a(f)|$ for all $a \in \mathbb{F}_2^n$).

Table 1.3 $F(x) = x^5$ over $\mathbb{F}_{2^3}$

a	$H_a(f)$	$\lvert H_a(f)\rvert$
$(0,0,1)$	$\{(0,0,1),(0,1,1),(1,0,1),(1,1,1)\}$	4
$(0,1,0)$	$\{(0,1,0),(0,1,1),(1,1,0),(1,1,1)\}$	4
$(0,1,1)$	$\{(0,0,1),(0,1,0),(1,0,1),(1,1,0)\}$	4
$(1,0,0)$	$\{(0,1,0),(0,1,1),(1,0,0),(1,0,1)\}$	4
$(1,0,1)$	$\{(0,0,1),(0,1,0),(1,0,0),(1,1,1)\}$	4
$(1,1,0)$	$\{(1,0,0),(1,0,1),(1,1,0),(1,1,1)\}$	4
$(1,1,1)$	$\{(0,0,1),(0,1,1),(1,0,0),(1,1,0)\}$	4

Example 1.3.2. $F(x) = x^3$ over $\mathbb{F}_{2^4}$ is APN.

Using the irreducible polynomial $\alpha^4 + \alpha + 1 = 0$ and the given relationship, we have that $F(x) = x^3$ over $\mathbb{F}_{2^4}$ corresponds to the function $f(x_1, x_2, x_3, x_4) = (f_1, f_2, f_3, f_4)$ over $\mathbb{F}_2^4$ given

by:

$$f_1(x_1,x_2,x_3,x_4) = x_1 + x_2 + x_3 + x_1x_2 + x_1x_3$$

$$f_2(x_1,x_2,x_3,x_4) = x_2 + x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4$$

$$f_3(x_1,x_2,x_3,x_4) = x_1 + x_1x_2 + x_2x_4 + x_3x_4$$

$$f_4(x_1,x_2,x_3,x_4) = x_4 + x_1x_3 + x_2x_3 + x_2x_4$$

And thus, for all $a \in \mathbb{F}_2^4$, such that $a \neq 0$, we can find $H_a(f)$ and see that $|H_a(f)| = \frac{1}{2}q$. So the function must be APN.

Example 1.3.3. $F(x) = x^{29}$ over $\mathbb{F}_{2^5}$ (Dobbertin's function) is APN. Using the irreducible polynomial $\alpha^5 + \alpha^2 + 1 = 0$ and the given relationship, we have that $F(x) = x^{29}$ over $\mathbb{F}_{2^5}$ corresponds to the function $f(x_1,x_2,x_3,x_4,x_5) = (f_1,f_2,f_3,f_4,f_5)$ over $\mathbb{F}_2^5$ given by:

$$\begin{aligned}
f_1(x_1,x_2,x_3,x_4,x_5) &= x_2 + x_1x_2 + x_1x_3 + x_2x_4 + x_3x_4 + x_3x_5 + x_4x_5 \\
&\quad + x_1x_2x_3 + x_2x_3x_5 + x_2x_4x_5 + x_1x_2x_3x_4 + x_1x_3x_4x_5 \\
f_2(x_1,x_2,x_3,x_4,x_5) &= x_1 + x_2 + x_3 + x_4 + x_1x_2 + x_1x_5 + x_2x_5 + x_3x_4 + x_4x_5 \\
&\quad + x_1x_2x_4 + x_1x_3x_4 + x_1x_3x_5 + x_2x_4x_5 + x_3x_4x_5 \\
&\quad + x_1x_2x_3x_4 + x_1x_2x_4x_5 + x_1x_3x_4x_5 + x_2x_3x_4x_5 \\
f_3(x_1,x_2,x_3,x_4,x_5) &= x_1 + x_1x_3 + x_2x_3 + x_2x_4 + x_3x_4 + x_4x_5 + x_1x_2x_4 \\
&\quad + x_1x_3x_4 + x_1x_4x_5 + x_2x_4x_5 + x_1x_2x_3x_5 + x_2x_3x_4x_5 \\
f_4(x_1,x_2,x_3,x_4,x_5) &= x_1 + x_3 + x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_5 \\
&\quad + x_4x_5 + x_1x_2x_3 + x_1x_2x_5 + x_1x_3x_4 + x_1x_3x_5 + x_2x_3x_4 \\
&\quad + x_1x_2x_3x_5 + x_1x_2x_4x_5 + x_2x_3x_4x_5 \\
f_5(x_1,x_2,x_3,x_4,x_5) &= x_1 + x_2 + x_3 + x_4 + x_5 + x_1x_3 + x_1x_5 + x_2x_3 + x_4x_5 + x_1x_2x_4 \\
&\quad + x_1x_4x_5 + x_2x_3x_5 + x_2x_4x_5 + x_1x_2x_4x_5 + x_1x_3x_4x_5 + x_2x_3x_4x_5
\end{aligned}$$

As we noticed earlier, when we consider APN functions over the vector space $\mathbb{F}_2^n$, the com-

Table 1.4 $F(x) = x^3$ over $\mathbb{F}_{2^4}$

a	$H_a(f)$	$\lvert H_a(f)\rvert$
$(0,0,0,1)$	$\{(0,0,0,0),(0,0,0,1),(0,0,1,0),(0,0,1,1)$ $(0,1,0,0),(0,1,0,1),(0,1,1,0),(0,1,1,1)\}$	8
$(0,0,1,0)$	$\{(0,0,0,0),(0,0,1,1),(0,1,0,1),(0,1,1,0)$ $(1,0,0,0),(1,0,1,1),(1,1,0,1),(1,1,1,0)\}$	8
$(0,0,1,1)$	$\{(0,0,0,0),(0,0,1,0),(0,1,0,0),(0,1,1,0)$ $(1,0,0,1),(1,0,1,1),(1,1,0,1),(1,1,1,1)\}$	8
$(0,1,0,0)$	$\{(0,0,0,0),(0,0,1,0),(0,1,0,1),(0,1,1,1)$ $(1,0,0,1),(1,0,1,1),(1,1,0,0),(1,1,1,0)\}$	8
$(0,1,0,1)$	$\{(0,0,0,0),(0,0,1,1),(0,1,0,0),(0,1,1,1)$ $(1,0,0,1),(1,0,1,0),(1,1,0,1),(1,1,1,0)\}$	8
$(0,1,1,0)$	$\{(0,0,0,0),(0,0,0,1),(0,0,1,0),(0,0,1,1)$ $(0,1,0,0),(0,1,0,1),(0,1,1,0),(0,1,1,1)\}$	8
$(0,1,1,1)$	$\{(0,0,0,0),(0,0,0,1),(0,0,1,0),(0,0,1,1)$ $(0,1,0,0),(0,1,0,1),(0,1,1,0),(0,1,1,1)\}$	8
$(1,0,0,0)$	$\{(0,0,0,0),(0,0,1,1),(0,1,0,0),(0,1,1,1)$ $(1,0,0,1),(1,0,1,0),(1,1,0,1),(1,1,1,0)\}$	8
$(1,0,0,1)$	$\{(0,0,0,0),(0,0,1,0),(0,1,0,0),(0,1,1,0)$ $(1,0,0,1),(1,0,1,1),(1,1,0,1),(1,1,1,1)\}$	8
$(1,0,1,0)$	$\{(0,0,0,0),(0,0,1,0),(0,1,0,0),(0,1,1,0)$ $(1,0,0,1),(1,0,1,1),(1,1,0,1),(1,1,1,1)\}$	8
$(1,0,1,1)$	$\{(0,0,0,0),(0,0,1,0),(0,1,0,1),(0,1,1,1)$ $(1,0,0,1),(1,0,1,1),(1,1,0,0),(1,1,1,0)\}$	8
$(1,1,0,0)$	$\{(0,0,0,0),(0,0,1,1),(0,1,0,1),(0,1,1,0)$ $(1,0,0,0),(1,0,1,1),(1,1,0,1),(1,1,1,0)\}$	8
$(1,1,0,1)$	$\{(0,0,0,0),(0,0,1,1),(0,1,0,0),(0,1,1,1)$ $(1,0,0,1),(1,0,1,0),(1,1,0,1),(1,1,1,0)\}$	8
$(1,1,1,0)$	$\{(0,0,0,0),(0,0,1,1),(0,1,0,1),(0,1,1,0)$ $(1,0,0,0),(1,0,1,1),(1,1,0,1),(1,1,1,0)\}$	8
$(1,1,1,1)$	$\{(0,0,0,0),(0,0,1,0),(0,1,0,1),(0,1,1,1)$ $(1,0,0,1),(1,0,1,1),(1,1,0,0),(1,1,1,0)\}$	8

ponent functions quickly grow in complexity as the length of the vectors increases. Because of this, it is generally easier to consider the corresponding functions over the finite field $\mathbb{F}_{2^n}$, so from this point forward, we will just consider APN functions over the finite field $\mathbb{F}_{2^n}$, remembering that we can always use the aforementioned relationship (1.2) to find the corresponding function over the related vector space. Before we move on to a characterization of all of the known APN functions, we want to take a brief detour to discuss a special class of APN functions.

1.4 Almost bent functions

1.4.1 Definitions and theorems

A special class of APN functions are the Almost Bent (AB) functions.

Definition 1.4.1. [36] Let $\mathbb{V}$ be an n-dimensional vector space over the field $\mathbb{F}_2$ with $q = 2^n = |\mathbb{V}|$ and let $f : \mathbb{V} \to \mathbb{V}$ be any function. The Fourier transform $\mu_f : \mathbb{V} \times \mathbb{V} \to \mathbb{R}$ of f is defined by the formula

$$\mu_f(a,b) = \sum_{x \in \mathbb{V}} (-1)^{\langle a,x \rangle} (-1)^{\langle b,f(x) \rangle},$$

and f is AB (almost bent) if $\mu_f(a,b) = 0, \pm\sqrt{2q}$ for all $(a,b) \neq (0,0)$ where $\langle , \rangle$ denotes the standard inner product on $\mathbb{V}$.

Proposition 1.4.2. AB functions exist only for n odd.

Proof. Suppose n is even. Then $n = 2k$ for some k. But if f is AB, then

$$\mu_f(a,b) = 0, \pm\sqrt{2q} \text{ for all } (a,b) \neq (0,0) \quad \Rightarrow \quad \mu_f(a,b) = 0, \pm\sqrt{2 \cdot 2^{2k}} = \sqrt{2^{2k+1}} \notin \mathbb{Z}.$$

But $\mu_f(a,b) = \sum_{x \in \mathbb{V}} (-1)^{\langle a,x \rangle + \langle b,f(x) \rangle}$ which must be an integer since $(-1)^{\langle a,x \rangle + \langle b,f(x) \rangle}$ is equal to ± 1 for all (a,b). □

Theorem 1.4.3. [36] A function f is AB if and only if the system of equations

$$\begin{cases} x+y+z &= a \\ f(x)+f(y)+f(z) &= b \end{cases} \tag{1.4}$$

has $q-2$ or $3q-2$ solutions (x,y,z) for every (a,b). If (1.4) has $q-2$ or $3q-2$ solutions, (x,y,z), for all (a,b), then the system has $3q-2$ solutions if $b=f(a)$, and $q-2$ solutions otherwise.

A sketch of the proof of this theorem is given by van Dam and Fon-Der-Flass in [36]. For completeness, we will provide a more detailed proof here.

Proof. The proof will be given in terms of matrices. We begin by defining several $q \times q$ matrices having real entries whose rows and columns are indexed by vectors from $\mathbb{V}$. Let I be the identity matrix, J the all-one matrix, E the matrix with a single non-zero entry, $E_{00}=1$, $E_{ij}=0$ for $(i,j) \neq (0,0)$. We also define the matrices X, M, $M^{(3)}$, F, and S whose entries are defined as follows:

$$X_{ab} = (-1)^{\langle a,b \rangle}$$

$$M_{ab} = \mu_f(a,b)$$

$$M^{(3)}_{ab} = \mu_f(a,b)^3$$

$$S_{ab} = |\{(x,y,z) : x+y+z=a; f(x)+f(y)+f(z)=b\}|$$

$$F_{ab} = \begin{cases} 1 & \text{if } b=f(a) \\ 0 & \text{otherwise} \end{cases}$$

Then we have the following equalities: $X^2 = qI$; $M = XFX$; $XJX = q^2E$.

Verify that $X^2 = qI$:

$$\begin{aligned} X^2_{ab} &= \sum_{x \in \mathbb{V}} X_{ax}X_{xb} \\ &= \sum_{x \in \mathbb{V}} (-1)^{\langle a,x \rangle + \langle x,b \rangle} \end{aligned}$$

$$= \sum_{x \in \mathbb{V}} (-1)^{\langle a,x \rangle + \langle b,x \rangle}$$

$$= \sum_{x \in \mathbb{V}} (-1)^{\langle a+b,x \rangle}$$

$$= \begin{cases} q & \text{if } a+b = \bar{0} \text{ so that } \langle a+b, x \rangle = 0 \text{ for all } x \in \mathbb{V} \\ 0 & \text{if } a+b \neq \bar{0} \text{ so that } \langle a+b, x \rangle = 0 \text{ for exactly half of the vectors } x \in \mathbb{V} \text{ and} \\ & \langle a+b, x \rangle = 1 \text{ for the rest of the vectors in } \mathbb{V}. \end{cases}$$

But $a+b = \bar{0}$ if and only if $a = b$ so $X^2_{ab} = \begin{cases} q & a = b \\ 0 & otherwise \end{cases}$

$\therefore X^2 = qI \Rightarrow X$ is nonsingular.

Verify that $M = XFX$: Since $F_{uv} = 1$ only when $v = f(u)$, row a of XF is a permutation of row a of X. So,

$$\begin{aligned} XFX_{ab} &= \sum_{x \in \mathbf{V}} (-1)^{\langle a, f^{-1}(x) \rangle} \cdot (-1)^{\langle x,b \rangle} \\ &= \sum_{x \in \mathbf{V}} (-1)^{\langle a, f^{-1}(x) \rangle + \langle x,b \rangle} \\ &= \sum_{y \in \mathbf{V}} (-1)^{\langle a,y \rangle + \langle f(y),b \rangle} \\ &= \sum_{y \in \mathbf{V}} (-1)^{\langle a,y \rangle + \langle b,f(y) \rangle} \\ &= \mu_f(a,b) \\ &= M_{ab} \end{aligned}$$

$\therefore M = XFX$

Verify that $XJX = q^2E$:

$$XJX_{ab} = \begin{bmatrix} \sum_{x\in\mathbb{V}}(-1)^{\langle a,x\rangle} & \sum_{x\in\mathbb{V}}(-1)^{\langle a,x\rangle} & \cdots \end{bmatrix} \begin{bmatrix} (-1)^{\langle x_1,b\rangle} \\ (-1)^{\langle x_2,b\rangle} \\ \vdots \\ (-1)^{\langle x_N,b\rangle} \end{bmatrix}$$

$$= \left(\sum_{x\in\mathbb{V}}(-1)^{\langle a,x\rangle}\right)\left(\sum_{x\in\mathbb{V}}(-1)^{\langle x,b\rangle}\right) \text{ but } \sum_{x\in\mathbb{V}}(-1)^{\langle a,x\rangle} = \begin{cases} q & a=0 \\ 0 & a\neq 0 \end{cases}$$

$$= \begin{cases} q\sum_{x\in\mathbb{V}}(-1)^{\langle x,b\rangle} & a=0 \\ 0 & a\neq 0 \end{cases}$$

$$= \begin{cases} q^2 & a,b=0 \\ 0 & otherwise \end{cases}$$

$$= (q^2E)_{ab}$$

$$\therefore XJX = q^2E$$

The condition that the system (1.4) has $q-2$ or $3q-2$ solutions (x,y,z) for every (a,b) is equivalent to the identity $S=(q-2)J+2qF$ since

$$[(q-2)J+2qF]_{ab} = \begin{cases} 3q-2 & b=f(a) \\ q-2 & otherwise \end{cases} = S_{ab}$$

The property that f is AB can be stated in terms of matrices as the identity:

$$M^{(3)} - 2qM = (q^3-2q^2)E$$

To see that this is true, consider the following:

$$f \text{ is } AB \Rightarrow \mu_f(a,b) \in \{0, \pm\sqrt{2q}\} \text{ if } (a,b) \neq (0,0)$$

$$\Leftrightarrow \mu_f(a,b) = \begin{cases} q & (a,b) = (0,0) \\ \text{roots of cubic equation: } x^3 - 2qx = 0 & \textit{otherwise} \end{cases}$$

So,

$$\begin{aligned}(M^{(3)} - 2qM)_{ab} &= \mu_f(a,b)^3 - 2q\mu_f(a,b) \\ &= \begin{cases} q^3 - 2q^2 & a = b = 0 \\ 0 & \textit{otherwise} \end{cases} \\ &= [(q^3 - 2q^2)E]_{ab}\end{aligned}$$

$\therefore M^{(3)} - 2qM = (q^3 - 2q^2)E$

Finally, consider the identity: $M^{(3)} = XSX$.

$$\begin{aligned}M^{(3)}_{ab} &= \mu_f(a,b)^3 \\ &= \sum_{x\in\mathbb{V}} (-1)^{\langle a,x\rangle}(-1)^{\langle b,f(x)\rangle} \sum_{y\in\mathbb{V}} (-1)^{\langle a,y\rangle}(-1)^{\langle b,f(y)\rangle} \sum_{z\in\mathbb{V}} (-1)^{\langle a,z\rangle}(-1)^{\langle b,f(z)\rangle} \\ &= \sum_{x,y,z\in V} (-1)^{\langle a,x+y+z\rangle}(-1)^{\langle b,f(x)+f(y)+f(z)\rangle} \\ &= \sum_{p\in V} (-1)^{\langle a,p\rangle} \sum_{x+y+z=p} (-1)^{\langle b,f(x)+f(y)+f(z)\rangle}\end{aligned}$$

For all r such that $r = f(x) + f(y) + f(z)$, there will be S_{pr} terms having this sum. So,

$$\begin{aligned}&= \sum_{p\in\mathbb{V}} (-1)^{\langle a,p\rangle} \sum_{r\in\mathbb{V}} S_{pr}(-1)^{\langle b,p\rangle} \\ &= \sum_{p,r\in\mathbb{V}} X_{ap}S_{pr}X_{rb} \\ &= (XSX)_{ab}\end{aligned}$$

$\therefore M^{(3)} = XSX.$

Then using these identities, we obtain:

$$\begin{aligned} M^{(3)} - 2qM - (q^3 - 2q^2)E &= XSX - 2qXFX - (q-2)q^2E \\ &= XSX - 2qXFX - (n-2)XJX \\ &= X(S - 2qF - (q-2)J)X \end{aligned}$$

Which implies that, if $M^{(3)} - 2qM = (q^3 - 2q^2)E$, then $S = 2qF + (q-2)J$. So f is AB is equivalent to the system (1.4) having $q-2$ or $3q-2$ solutions (x, y, z) for every (a, b).
$\therefore$ the definitions are equivalent. □

1.4.2 Relationship between AB and APN functions

Lemma 1.4.4. Every AB function is APN.

Proof. Assume that f is not APN, and show that f is not AB.
f is not APN implies that for some $r \neq 0, a \neq p \neq a + r$, the equality $f(p) + f(p+r) = f(a) + f(a+r)$ holds and the system (1.4) in addition to the "trivial" solutions has the solution, $x = p, y = p+r, z = a+r$. The system has $3q-2$ "trivial" solutions with one variable equal to a and the other two variables equal to each other. (i.e. There are $q-1$ solutions of the form (a, c, c) and three possible ways that a, c, c can be arranged which yields $3q-3$ solutions plus the solution (a, a, a), for a total of $3q-2$ "trivial" solutions). So f is not AB. □

1.5 Known APN functions

All known binary APN functions are equivalent (see Defn. 1.5.4) to certain power functions $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ of the form $f(x) = x^k$. The first table considers the case when n is odd, $n = 2m+1$, and lists the values of the exponent k and the type of function obtained. The second table considers the case for n even, $n = 2m$, and lists the values for the exponent k for which the function is APN. We should also note that the inverse of an APN or AB function is also APN or AB, respectively.

Table 1.5 Known APN functions x^k on $\mathbb{F}_{2^n}$, $n = 2m + 1$

Name	Exponent k	ref.
Gold's functions*	$2^i + 1$ with $(i, n) = 1$, $1 \leq i \leq m$	[20], [2]
Kasami's functions*	$2^{2i} - 2^i + 1$ with $(i, n) = 1$, $2 \leq i \leq m$	[26]
Field inverse	$2^n - 2$	[30]
Welch's function*	$2^m + 3$	[8], [24]
Niho's function*	$2^m + 2^{m/2} - 1$ (even m) $2^m + 2^{(3m+1)/2} - 1$ (odd m)	[24]
Dobbertin's function	$2^{4i} + 2^{3i} + 2^{2i} + 2^i - 1$ if $n = 5i$	[15]

* AB functions

Table 1.6 Known APN functions x^k on $\mathbb{F}_{2^n}$, $n = 2m$

Name	Exponent k	ref.
Gold's functions	$2^i + 1$ with $(i, n) = 1$, $1 \leq i \leq m$	[20]
Kasami's functions	$2^{2i} - 2^i + 1$ with $(i, n) = 1$, $2 \leq i \leq m$	[26]
Dobbertin's function	$2^{4i} + 2^{3i} + 2^{2i} + 2^i - 1$ if $n = 5i$	[15]

Theorem 1.5.1. Given an APN function f, for any invertible linear transformation A and a vector b in $\mathbb{V}$, the function $g : \mathbb{V} \to \mathbb{V}$ defined by $g(x) = f(Ax + b)$ is APN.

Proof. Prove $g(x) = f(Ax + b)$ is APN.

f is $APN \to |H_a(f)| = |\{f(x + a) + f(x) : x \in \mathbb{V}\}| = \frac{1}{2}|\mathbb{V}|$.

$$
\begin{aligned}
|\{g(x + a) + g(x) : x \in \mathbb{V}\}| &= |\{f(A(x + a) + b) + f(Ax + b) : x \in \mathbb{V}\}| \\
&= |\{f(Ax + b + Aa) + f(Ax + b) : x \in \mathbb{V}\}| \\
&\text{let } y = Ax + b \text{ and } c = Aa, \text{ then } y = Ax + b \text{ varies as } x \text{ varies over} \\
&\mathbb{V} \text{ since the transformation } Ax + b \text{ is bijective} \\
&= |\{f(y + c) + f(y) : y \in V\}| \\
&= |H_c(f)| \\
&= \frac{1}{2}|\mathbb{V}|
\end{aligned}
$$

$\therefore g(x) = f(Ax + b)$ is APN. □

Theorem 1.5.2. If f is APN permutation, then f^{-1} is APN.

Proof. Prove f^{-1} is APN where f is an APN permutation. Since f is a permutation, f is bijective and since f is APN, if $b \in H_a(f)$, then $f(x+a)+f(x) = b$ has exactly 2 solutions. Let $y = f(x)$ and $y' = f(x+a)$, then $y' = y+b$. So, for given a and b, $f(x+a)+f(x) = b$ has exactly 0 or 2 solutions. But, $x+a = f^{-1}(y+b)$ and $x = f^{-1}(y)$, so $f^{-1}(y+b)+f^{-1}(y) = a$ which has exactly 0 or 2 solutions since $f(x+a)+f(x) = b$ has exactly 0 or 2 solutions.
$\therefore f^{-1}$ is APN. □

Theorem 1.5.3. Given an APN function f, for any linear function g, $f+g$ is APN.

Proof. Prove $f+g$ is APN.
Let $h(x) = f(x)+g(x)$.

$$
\begin{aligned}
|H_a(h)| = |\{h(x+a)+h(x) : x \in \mathbb{V}\}| &= |\{f(x+a)+g(x+a)+f(x)+g(x) : x \in \mathbb{V}\}| \\
&= |\{f(x+a)+g(x)+g(a)+f(x)+g(x) : x \in \mathbb{V}\}| \\
&\quad \text{since } g \text{ is linear} \\
&= |\{f(x+a)+f(x)+g(a) : x \in \mathbb{V}\}| \\
&= |\{f(x+a)+f(x) : x \in \mathbb{V}\}| \text{ since } g(a) \text{ is a constant} \\
&= |H_a(f)| \\
&= \frac{1}{2}|\mathbb{V}|
\end{aligned}
$$

$\therefore f+g$ is APN. □

Since every known APN function is equivalent to one of the power functions listed in Tables 1.5 and 1.6, we want to further study this notion of equivalence and the relationships between APN functions.

Definition 1.5.4. [18] We say that two functions f and g are *equivalent* if the lists of values $N_f(a,b)$ and $N_g(a,b)$ with $a,b \in \mathbb{F}_q$ are equal. (i.e. $\{N_f(a,b) : a,b \in \mathbb{F}_q\} = \{N_g(a,b) : a,b \in \mathbb{F}_q\}$).

For power functions, we have that functions whose exponents come from the same cyclotomic coset are equivalent.

Theorem 1.5.5. $N_d(a,b)$, where $f(x) = x^d$, is constant on the cyclotomic coset

$$\{dp^i : i = 0, 1, \ldots, n-1\} \text{ (i.e. } N_{dp^i}(a,b) = N_d(a,b) \text{ for } i = 0, 1, \ldots, n-1)$$

Proof.

$$\begin{aligned} |\{x : (x+a)^{dp^i} - x^{dp^i} = b\}| &= |\{x : (x^{p^i} + a)^d - (x^{p^i})^d = b\}| \\ &= |\{y : (y+a)^d - y^d = b\}| \text{ with } y = x^{p^i} \end{aligned}$$

□

Because of this, we can classify all APN power functions according the the cyclotomic coset from which the exponent came. We present in Table 1.7 a listing of all APN power functions organized by the exponent d for the binary fields up to $\mathbb{F}_{2^{13}}$. These are organized according to class and have been verified through the use of the computer programs C.2.1 and C.2.2 which can be found in Appendix C.

This notion of equivalence can be extended to polynomial functions using the following theorem. In order to well state this relation, we first need to explore the notion of p-polynomials.

Definition 1.5.6. Let $L \in \mathbb{F}_{p^n}[x]$ be a polynomial. L is called a p-polynomial if

$$\begin{cases} L(x+y) = L(x) + L(y) \\ L(\lambda x) = \lambda L(x), \ \lambda \in \mathbb{F}_p \end{cases}.$$

(i.e. The map $L : \mathbb{F}_q \to \mathbb{F}_q$ is a linear $\mathbb{F}_p$-mapping.)

Proposition 1.5.7. [18] $L \in \mathbb{F}_{p^n}[x]$ is a p-polynomial *iff*

$$L = \sum_{i=0}^{n-1} a_i x^{p^i} = a_0 x + a_1 x^p + a_2 x^{p^2} + \cdots + a_{n-1} x^{p^{n-1}}$$

Proposition 1.5.8. [18] Suppose L, M are p-polynomials and $a \in \mathbb{F}_p^n$. Then we have the following equivalence.

1. $f(x) \sim f(L(x)) + M(x) + a$
2. $f(x) \sim L(f(x)) + M(x) + a$

Because of this, we can assume *wlog* that $f(0) = 0$ and $f(1) = 1$.

As was previously mentioned, whether or not the list of binary APN functions given in Tables 1.5 and 1.6 is complete (up to equivalence) remains an open question. We will now turn our focus to a relaxation of our definition and consider other functions with low uniformity.

1.6 Functions with low uniformity

Definition 1.6.1. Let $\Delta_f := \max\{N_f(a,b) : a, b \in \mathbb{F}_q,\ a \neq 0\}$.

Then, perfect nonlinear functions are those for which $\Delta_f = 1$ and APN functions are those for which $\Delta_f = 2$. We are interested in functions with low uniformity, that is, functions for which Δ_f is small. We have previously characterized PN and APN functions, so we will now turn our focus to characterizing other functions with low Δ_f's. In order to classify them, we will again need to use the notions of equivalence in nonlinearity and p-polynomials from the previous section.

Theorem 1.6.2. For $f(x) = x^d$ with $\gcd(d, q-1) = 1$, $f(x) = x^d$ and $g(x) = x^{d^{-1}}$ are equivalent, where d^{-1} is calculated modulo $q-1$. (i.e. $N_d(a,b) = N_{d^{-1}}(b,a) \Rightarrow \Delta_d = \Delta_{d^{-1}}$)

Proof.

$$\begin{aligned} N_d(a,b) &= |\{x : f(x+a) - f(x) = b\}| \\ &= |\{x : (x+a)^d - x^d = b\}| \\ &= |\{x : x + a = (x^d + b)^{d^{-1}}\}| \\ &= |\{x : a = (x^d + b)^{d^{-1}} - (x^d)^{d^{-1}}\}| \end{aligned}$$

$$= |\{y : a = (y+b)^{d^{-1}} - y^{d^{-1}}\}| \text{ with } y = x^d$$
$$= |\{y : g(y+b) - g(y)y = a\}|$$
$$= N_{d^{-1}}(b, a)$$

□

Theorem 1.6.3. $N_d(a,b) = N_d(1, ba^{-d})$ for $a \neq 0$

Proof.

$$\begin{aligned} N_d(1, ba^{-d}) &= |\{x : f(x+1) - f(x) = ba^{-d}\}| \\ &= |\{x : (x+1)^d - x^d = ba^{-d}\}| \\ &= |\{x : a^d(x+1)^d - a^d x^d = b\}| \\ &= |\{x : (ax+a)^d - (ax)^d = b\}| \\ &= |\{y : (y+a)^d - y^d = b\}| \text{ where } y = ax \\ &= |\{y : f(y+a) - f(y) = b\}| \\ &= N_d(a,b) \end{aligned}$$

□

My analysis of the results for Δ_d over the various fields has led me to pursue the possibility of classifying the power functions $f(x) = x^d$ over $\mathbb{F}_{2^n}$ for which $\Delta_d = 4$. A classification of the power functions $f(x) = x^d$ by Δ_d over $\mathbb{F}_{2^n}$ for n up to 13 is given in Appendix A. The following is a proof for one of the subclasses of these functions for which $\Delta_d = 4$.

Proposition 1.6.4. Let $f(x) = x^d$ be a mapping $\mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$, for n even. Then $\Delta_d = 4$ if $d = 2^{n-1} - 1$.

Proof. Since $N_d(a,b) = N_d(1, ba^{-d})$, we can just consider $N_d(1,b)$. Now,

$$N_d(1,b) = |\{x : f(x+1) + f(x) = b\}|$$

$$= |\{x : (x+1)^{2^{n-1}-1} + x^{2^{n-1}-1} = b\}|$$

So, if $b = 1$, then $x = 0$ and $x = 1$ are roots of $f(x+1) + f(x) = b$. We will now show that $f(x+1) + f(x) = b$ has exactly two more roots. If $x \neq 0$ and $x \neq 1$, then

$$\begin{aligned}(x+1)^{2^{n-1}-1} + x^{2^{n-1}-1} = 1 &\Leftrightarrow (x+1)^{2^{n-1}}(x+1)^{-1} + x^{2^{n-1}}x^{-1} = 1 \\ &\Leftrightarrow (x+1)^{-1} + x^{-1} = 1 \\ &\Leftrightarrow (x+1)x[(x+1)^{-1} + x^{-1}] = (x+1)x \\ &\Leftrightarrow x + (x+1) = x^2 + x \\ &\Leftrightarrow x^2 + x + 1 = 0\end{aligned}$$

which has exactly two solutions in $\mathbb{F}_{2^n}$ since n is even. So $N_d(1,1) = 4$. If $b \neq 1$, then $x+1 \neq 0$ and $x \neq 0$ so

$$\begin{aligned}N_d(1,b) &= |\{x : f(x+1) + f(x) = b\}| \\ &= |\{x : (x+1)^{2^{n-1}-1} + x^{2^{n-1}-1} = b\}| \\ &= |\{x : (x+1)^{-1} + x^{-1} = b\}| \\ &= |\{x : x^2 + x + 1 = 0\}| \\ &\leq 2\end{aligned}$$

$\therefore \Delta_d = 4$ □

Table 1.7 APN power functions, x^d, over the field $\mathbb{F}_{2^n}$

$\mathbb{F}_{2^n}$	d	Ref.
$\mathbb{F}_{2^2}$	3	Gold
$\mathbb{F}_{2^3}$	3 6 5	Gold, Welch, Niho, Field Inverse
$\mathbb{F}_{2^4}$	3 6 12 9	Gold
$\mathbb{F}_{2^5}$	3 6 12 24 17	Gold
	5 10 20 9 18	Gold, Niho
	7 14 28 25 19	Welch
	11 22 13 26 21	Kasami
	15 30 29 27 23	Dobbertin, Field Inverse
$\mathbb{F}_{2^6}$	3 6 12 24 48 33	Gold
$\mathbb{F}_{2^7}$	3 6 12 24 48 96 65	Gold
	5 10 20 40 80 33 66	Gold
	9 18 36 72 17 34 68	Gold
	11 22 44 88 49 98 69	Welch
	13 26 52 104 81 35 70	Kasami
	15 30 60 120 113 99 71	Gold
	23 46 92 57 114 101 75	Kasami
	27 54 108 89 51 102 77	Gold
	29 58 116 105 83 39 78	Niho
	43 86 45 90 53 106 85	Gold
	63 126 125 123 119 111 95	Field Inverse
$\mathbb{F}_{2^8}$	3 6 12 24 48 96 192 129	Gold
	9 18 36 72 144 33 66 132	Gold
	39 78 156 57 114 228 201 147	Kasami
$\mathbb{F}_{2^9}$	3 6 12 24 48 96 192 384 257	Gold
	5 10 20 40 80 160 320 129 258	Gold
	13 26 52 104 208 416 321 131 262	Kasami
	17 34 68 136 272 33 66 132 264	Gold
	19 38 76 152 304 97 194 388 265	Niho, Welch
	27 54 108 216 432 353 195 390 269	Niho, Welch
	31 62 124 248 496 481 451 391 271	Gold
	47 94 188 376 241 482 453 395 279	Kasami
	59 118 236 472 433 355 199 398 285	Kasami
	87 174 348 185 370 229 458 405 299	Kasami
	103 206 412 313 115 230 460 409 307	Gold
	171 342 173 346 181 362 213 426 341	Gold
	255 510 509 507 503 495 479 447 383	Field Inverse

Table 1.7 (Continued)

$\mathbb{F}_{2^n}$	d	Ref.
$\mathbb{F}_{2^{10}}$	3 6 12 24 48 96 192 384 768 513	Gold
	9 18 36 72 144 288 576 129 258 516	Gold
	57 114 228 456 912 801 579 135 270 540	Kasami
	213 426 852 681 339 678 333 666 309 618	Dobbertin
$\mathbb{F}_{2^{11}}$	3 6 12 24 48 96 192 384 768 1536 1025	Gold
	5 10 20 40 80 160 320 640 1280 513 1026	Gold
	9 18 36 72 144 288 576 1152 257 514 1028	Gold
	13 26 52 104 208 416 832 1664 1281 515 1030	Kasami
	17 34 68 136 272 544 1088 129 258 516 1032	Gold
	33 66 132 264 528 1056 65 130 260 520 1040	Gold
	35 70 140 280 560 1120 193 386 772 1544 1041	Welch
	43 86 172 344 688 1376 705 1410 773 1546 1045	Kasami
	57 114 228 456 912 1824 1601 1155 263 526 1052	Kasami
	63 126 252 504 1008 2016 1985 1923 1799 1551 1055	Gold
	95 190 380 760 1520 993 1986 1925 1803 1559 1071	Kasami
	107 214 428 856 1712 1377 707 1414 781 1562 1077	Niho
	117 234 468 936 1872 1697 1347 647 1294 541 1082	Welch
	143 286 572 1144 241 482 964 1928 1809 1571 1095	Kasami
	151 302 604 1208 369 738 1476 905 1810 1573 1099	Kasami
	231 462 924 1848 1649 1251 455 910 1820 1593 1139	Gold
	249 498 996 1992 1937 1827 1607 1167 287 574 1148	Niho
	315 630 1260 473 946 1892 1737 1427 807 1614 1181	Kasami
	365 730 1460 873 1746 1445 843 1686 1325 603 1206	Gold
	411 822 1644 1241 435 870 1740 1433 819 1638 1229	Gold
	413 826 1652 1257 467 934 1868 1689 1331 615 1230	Kasami
	683 1366 685 1370 693 1386 725 1450 853 1706 1365	Gold
	1023 2046 2045 2043 2039 2031 2015 1983 1919 1791 1535	Field Inverse
$\mathbb{F}_{2^{12}}$	3 6 12 24 48 96 192 384 768 1536	Gold
	33 66 132 264 528 1056 2112 129 258 516 1032 2064	Gold
	159 318 636 1272 2544 993 1986 3972 3849 3603 3111 2127	Kasami

Table 1.7 (Continued)

$\mathbb{F}_{2^n}$	d	Ref.
$\mathbb{F}_{2^{13}}$	3 6 12 24 48 96 192 384 768 1536 3072 6144 4097	Gold
	5 10 20 40 80 160 320 640 1280 2560 5120 2049 4098	Gold
	9 18 36 72 144 288 576 1152 2304 4608 1025 2050 4100	Gold
	13 26 52 104 208 416 832 1664 3328 6656 5121 2051 4102	Kasami
	17 34 68 136 272 544 1088 2176 4352 513 1026 2052 4101	Gold
	33 66 132 264 528 1056 2112 4224 257 514 1028 2056 4112	Gold
	57 114 228 456 912 1824 3648 7296 6401 4611 1031 2062 4124	Kasami
	65 130 260 520 1040 2080 4160 129 258 516 1032 2064 4128	Gold
	67 134 268 536 1072 2144 4288 385 770 1540 3080 6160 4129	Welch
	71 142 284 568 1136 2272 4544 897 1794 3588 7176 6161 4131	Niho
	127 254 508 1016 2032 4064 8128 8065 7939 7687 7183 6175 4159	Gold
	171 342 684 1368 2736 5472 2753 5506 2821 5642 3093 6186 4181	Kasami
	191 382 764 1528 3056 6112 4033 8066 7941 7691 7191 6191 4191	Kasami
	241 482 964 1928 3856 7712 7233 6275 4359 527 1054 2108 4216	Kasami
	287 574 1148 2296 4592 993 1986 3972 7944 7697 7203 6215 4239	Kasami
	347 694 1388 2776 5552 2913 5826 33461 6922 5653 3115 6230 4269	Niho
	367 734 1468 2936 5872 3553 7106 6021 3851 7702 7213 6235 4279	Welch
	635 1270 2540 5080 1969 3938 7876 7561 6931 5671 3151 6302 4413	Kasami
	723 1446 2892 5784 3377 6754 5317 2443 4886 1581 3162 6324 4457	Kasami
	911 1822 3644 7288 6385 4579 967 1934 3868 7736 7281 6371 4551	Gold
	1243 2486 4972 1753 3506 7012 5833 3475 6950 5709 3227 6454 4717	Gold
	1245 2490 4980 1769 3538 7076 5961 3731 7462 6733 5275 2359 4718	Kasami
	1453 2906 5812 3433 6866 5541 2891 5782 3373 6746 5301 2411 4822	Gold
	1639 3278 6556 4921 1651 3302 6604 5017 1843 3686 7372 6553 4915	Gold
	1691 3382 6764 5337 2483 4966 1741 3482 6964 5737 3283 6566 4941	Kasami
	2731 5462 2733 5466 2741 5482 2773 5546 2901 5802 3413 6826 5461	Gold
	4095 8190 8189 8187 8183 8175 8159 8127 8063 7935 7679 7167 6143	Field Inverse

CHAPTER 2. SEMI-BIPLANES AND APN FUNCTIONS

In the rest of this dissertation, we study several combinatorial structures, such as semi-biplanes, distance-regular graphs, and association schemes, which are derived from APN functions. We will begin by considering known results regarding combinatorial structures which have been constructed from APN functions. In addition to a rich internal structure, the combinatorial structures obtained from APN functions share various interesting properties with other structures. We will investigate the structure of combinatorial objects obtained from APN functions, examine their connection to known combinatorial objects such as, distance-regular graphs and symmetric association schemes, and characterize some of these combinatorial structures in Chapter 4.

2.1 Finite incidence structures and semi-biplanes

Definition 2.1.1. An incidence structure is a triple $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ where

1. $\mathcal{P}$ is a finite set, the elements of which are called points,
2. $\mathcal{B}$ is a finite set, the elements of which are called blocks (or lines),
3. $\mathcal{I}$ is an incidence relation between $\mathcal{P}$ and $\mathcal{B}$, i.e., $\mathcal{I}$ is a subset of $P \times \mathcal{B}$.

The elements of $\mathcal{I}$ are called *flags*, and, if $(P, B) \in \mathcal{I}$, then we say that the point P and block B are incident. Let

$$(P) = \{B \in \mathcal{B} : (P, B) \in \mathcal{I}\} \text{ for } P \in \mathcal{P}$$

$$(B) = \{P \in \mathcal{P} : (P, B) \in \mathcal{I}\} \text{ for } B \in \mathcal{B},$$

and let $r_P = |(P)|$ and $k_B = |(B)|$; the cardinality of the two sets (P) and (B), respectively. If $r_P = r$ and $k_B = k$ are constant, then the incidence structure $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ is called a *tactical configuration.* The constants r and k together with $v = |\mathcal{P}|$ and $b = |\mathcal{B}|$ are called the parameters of the incidence structure.

Proposition 2.1.2. In a tactical configuration, $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$, with the parameters v, b, r, and k, $vr = bk$.

Proof. This can be shown by counting the total number of flags in $\mathcal{I}$ in two ways. There are v ways to choose $P \in \mathcal{P}$. For each such P, there are r blocks B such that (P, B) belongs to $\mathcal{I}$. Hence $|\mathcal{I}| = vr$. On the other hand, there are b ways to choose a block $B \in \mathcal{B}$. For each such B, there are k ways to choose $P \in \mathcal{P}$ such that $(P, B) \in \mathcal{I}$. Hence $|\mathcal{I}| = bk$, and we have $vr = bk$ as desired. □

It is often convenient to represent an incidence structure by means of a matrix.

Let $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ be an incidence structure with $\mathcal{P} = \{P_1, P_2, ..., P_v\}$ and $\mathcal{B} = \{B_1, B_2, ..., B_b\}$. The *incidence matrix* of $\mathcal{S}$ is the $v \times b$-matrix $M = [m_{ij}]$, with entries 0 and 1, which is defined by the rule $m_{ij} = \begin{cases} 1 & \text{if } (P_i, B_j) \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$.

An incidence structure is called *simple* if there are no repeated blocks; i.e., $(B_i) \neq (B_j)$ *iff* $i \neq j$. In this case, a block B is identified with (B) as a subset of $\mathcal{P}$. In what follows only simple incidence structures will be discussed.

Let $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ and $\mathcal{S}' = (\mathcal{P}', \mathcal{B}', \mathcal{I}')$ be two incidence structures. $\mathcal{S}$ and $\mathcal{S}'$ are said to be *isomorphic* if there exists a bijection $f : \mathcal{P} \to \mathcal{P}'$ such that $\{f((B)) : B \in \mathcal{B}\} = \{(B') : B' \in \mathcal{B}'\}$. In general, determining whether or not two incidence structures are isomorphic is a difficult computational problem.

We can also describe the isomorphism of incidence structures in terms of incidence matrices.

Proposition 2.1.3. Suppose $M = [m_{ij}]$ and $N = [n_{ij}]$ are the incidence matrices of two incidence structures $\mathcal{S}$ and $\mathcal{S}'$, respectively. Then the two incidence structures are isomorphic *iff* there exist permutations π of $I_v = \{1, 2, ..., v\}$ and σ of $I_b = \{1, 2, ..., b\}$ such that $m_{ij} = n_{\pi(i)\sigma(j)}$ for all $i \in I_v$ and $j \in I_b$.

Proof. Straightforward. □

There are many other structures that can be derived from a given incidence structure. We describe three examples of such structures which will be used in our discussion.

1. (*Dual structures.*) $\overline{\mathcal{S}} = (\overline{\mathcal{P}}, \overline{\mathcal{B}}, \overline{\mathcal{I}})$ is called the dual structure of $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ if $\overline{\mathcal{P}} = \mathcal{B}$, $\overline{\mathcal{B}} = \mathcal{P}$ and $(P, B) \in \mathcal{I}$ *iff* $(B, P) \in \overline{\mathcal{I}}$. We note that $\overline{\overline{\mathcal{S}}} = \mathcal{S}$.

2. (*Complementary structures.*) $\mathcal{T} = (\mathcal{P}, \mathcal{B}, \mathcal{J})$ where $\mathcal{J} = \mathcal{P} \times \mathcal{B} - \mathcal{I}$ is called the complementary structure of $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$; i.e., $(P, B) \in \mathcal{J}$ in $\mathcal{T}$ *iff* $(P, B) \notin \mathcal{I}$ in $\mathcal{S}$.

3. (*Subincidence structures.*) $\mathcal{T} = (\mathcal{Q}, \mathcal{L}, \mathcal{I})$ is called a sub(incidence)-structure of a given incidence structure $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ if $\mathcal{Q} \subseteq \mathcal{P}$, $\mathcal{L} \subseteq \mathcal{B}$, and $\mathcal{J} = \mathcal{I} \cap (\mathcal{Q} \times \mathcal{L})$.

Semi-biplanes were introduced by D. R. Hughes [25] and are studied by many people, most notably, P. Wild [37]. Together with biplanes they exhibit interesting geometric and combinatorial properties.

Definition 2.1.4. A semi-biplane is a connected finite incidence structure satisfying

1. every pair of points is incident with 0 or 2 blocks, and

2. every pair of blocks is incident with 0 or 2 points.

Proposition 2.1.5. A semi-biplane has the following properties.

1. There is a positive integer k such that every block is incident with k points and every point is incident with k blocks.

2. The number of points equals the number of blocks.

3. The dual structure of a semi-biplane is isomorphic to itself; that is, a semi-biplane is self-dual.

Proof. It is straightforward from the definition (cf. [25]). □

A semi-biplane has two parameters v and k where v is the number of points. A semi-biplane on v points where each block is incident with k points is denoted $sbp(v,k)$.

Let $(\mathcal{P},\mathcal{B},\mathcal{I})$ be a $sbp(v,k)$ with $\mathcal{P} = \{P_1, P_2, \ldots, P_v\}$. The *collineation* between points can be represented conveniently by a $v \times v$ matrix C whose rows and columns are indexed by points, the (i,j)-entry, C_{ij}, of which is defined by

$$C_{ij} = \begin{cases} 1 & \text{if there is a block incident with both } P_i \text{ and } P_j \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 2.1.6. If C is the collineation matrix of $sbp(v,k)$, then

$$CJ = JC = \frac{k(k-1)}{2}J$$

where J is the all-1 matrix.

Proof. There are k blocks incident with a fixed point $P \in \mathcal{P}$. Each of these k blocks is incident with k points including the fixed point P. So the total number of points collinear with P is $k(k-1)/2$. The denominator 2 comes from the fact that each pair $\{P,Q\}$ is incident with exactly two blocks for any point Q which is incident with the above k blocks. □

Proposition 2.1.7. If M is an incidence matrix of $sbp(v,k)$, then

$$MJ = JM = kJ$$

$$MM^t = M^tM = kI + 2M$$

Proof. Straightforward from 2.1.5 and the definition of the incidence matrix. □

Some known constructions of semi-biplanes are iterative and some require the input semi-biplanes to have special algebraic and geometric properties, such as, divisibility, regular automorphism groups, or polarities. There are two well-known constructions which supply infinite families of semi-biplanes with the properties we require. The one is based on involutions in

projective planes, and the other uses point and block sets corresponding to vectors from the n-dimensional binary vector space (cf. [37]). The families using this latter method can be constructed from APN functions.

2.2 Semi-biplanes obtained from APN functions

Dembowski and Ostrom [13] introduced a functionally dependent incidence structure. They showed that the existence of a perfect nonlinear function was equivalent to the corresponding incidence structure representing an affine plane with particular properties. Motivated by their results, Coulter and Henderson [11] have used APN functions to construct semi-biplanes as follows.

Definition 2.2.1. Let $V = \mathbb{F}_2^n$ be the affine space over the finite field $\mathbb{F}_2$ of order 2, and let $f : V \to V$ be a map. Define the incidence structure $S(n; f) = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ by

Points: $P = (x, a) \in \mathcal{P} = V \times V$

Blocks: $B = [y, b] \in \mathcal{B} = V \times V$

Incidence: $((x, a), [y, b]) \in \mathcal{I} \Leftrightarrow a + b = f(x + y)$.

The number of points and blocks of the incidence structure $S(n; f)$ is the same with $|\mathcal{P}| = |\mathcal{B}| = |V \times V| = 2^{2n}$. It has the following properties.

Proposition 2.2.2. Let $S(n; f)$ be an incidence structure as defined above. Then, it will satisfy the following properties.

(a) Each block incident with 2^n points and each point lies on 2^n blocks.

(b) It is self-dual.

(c) If $([y_1, b_1]) \cap ([y_2, b_2]) \neq \emptyset$, then $y_1 \neq y_2$. Also, if $((x_1, a_1)) \cap ((x_2, a_2)) \neq \emptyset$, then $x_1 \neq x_2$.

(d) For each point (x, a), there are exactly $2^{n-1}(2^n - 1)$ other points defined by the lines through it.

Proof. In $S(n;f) = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ where $\mathcal{P} = \{(x,a) : x, a \in V\}$ and $\mathcal{B} = \{[y,b] : y, b \in V\}$, a point (x,a) is incident with the block $[y,b]$ *iff* $f(x+y) = a+b$.

(a) follows from the following simple counting argument:

$$\begin{aligned} k = |([y,b])| &= |\{(x,a) \in \mathcal{P} : ((x,a),[y,b]) \in \mathcal{I}\}| \\ &= |\{(x,a) \in V \times V : a+b = f(x+y)\}| \\ &= 2^n \text{ since for all } x \in V, \exists!\ a \in V \text{ such that } a = f(x+y) - b. \end{aligned}$$

This number k does not depend on the choice of y and b, so, every block is incident with 2^n points. Similarly, for $(x,a) \in \mathcal{P}$, we can solve for $b \in V$ in the equation $a+b = f(x+y)$ for each $y \in V$; so, every point is incident with 2^n blocks.

(b) Observe that point (x,a) is incident with block $[y,b]$ *iff* the point (y,b) is incident with the block $[x,a]$, as $a+b = f(x+y)$ *iff* $b+a = f(y+x)$. So it is self-dual.

(c) Suppose $[y_1,b_1]$ and $[y_2,b_2]$ have a common point, (x,a), and suppose that $y_1 = y_2$. Then $a+b_1 = a+b_2$ since $f(x+y_1) = f(x+y_2)$, which implies that the two lines coincide. The second statement follows from the duality.

(d) By (a), there are 2^n blocks which are incident with the given point P, and each of these blocks contains $2^n - 1$ points besides P. Since every pair of blocks containing P also contains a second point in common, we have that half of the $2^n(2^n-1)$ points lie in the blocks containing P. This completes the proof. □

Proposition 2.2.3. [11] Let $V = \mathbb{F}_2^n$, $n \geq 2$, and let $f : V \to V$ be a bijective APN function. Then $S(n;f) = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ is a semi-biplane.

Proof. For any $(x_1,a_1),(x_2,a_2) \in \mathcal{P}$, we need to show that either no block or two blocks contain both points. Suppose there is a block $[y,b] \in \mathcal{B}$ that contains both points. Then $f(x_1+y) = a_1+b$ and $f(x_2+y) = a_2+b$. Let $A = (x_1+y)+(x_2+y) = x_1+x_2$ and let $B = f(x_1+y)+f(x_2+y) = (a_1+b)+(a_2+b) = a_1+a_2$.

Then x_1+y and x_2+y satisfy the system of equations $\begin{cases} X+Y &= A \\ f(X)+f(Y) &= B \end{cases}$. But, f is APN,

so this system must have no solution or exactly two solutions. Therefore, the points (x_1, a_1) and (x_2, a_2) are either contained together in no blocks or in two blocks. By the duality, each block is incident with 0 or 2 points as well. □

The incidence structure $S(n; f)$ given by an APN function f yields either $sbp(2^{2n}, 2^n)$ or $sbp(2^{2n-1}, 2^n)$.

Proposition 2.2.4. [11] The incidence structure $S(n; f)$ is a semi-biplane with parameters $(2^{2n}, 2^n)$ if the incidence structure is connected. If $S(n; f)$ is not connected, then it consists of two disjoint semi-biplanes with parameters $(2^{2n-1}, 2^n)$.

Proof. Suppose $S(n; f)$ is not connected. Let $q = 2^n$. Then it can have at most two connected components, since otherwise, a point P would be collinear with $1 + q(q-1)/2$ points which exceeds the bound of $q^2/3$ from part (d) of 2.2.2. For each $y \in V$, define the parallel class of lines for y to be the set $\{[y, b] : b \in V\}$. It is clear that all lines in a parallel class are parallel since for a given y and point (x, a), $f(x + y) = a + b$ for only one b. Let t be the number of lines from a parallel class in one of the components. Then, each of these t lines contains q points. Hence

$$\frac{q(q-1)}{2} + 1 \leq tq \leq q^2 - \frac{q(q-1)}{2} - 1$$

and dividing through by q we have

$$\frac{q-1}{2} + \frac{1}{q} \leq t \leq \frac{q+1}{2} - \frac{1}{q}.$$

So $t = \frac{q}{2}$, which is an integer, and thus each component contains exactly one half of the lines from each parallel class. □

Proposition 2.2.5. If $n > 1$ and f is a bijective APN function, then $S(n; f)$ is connected.

Proof. Suppose $S(n; f)$ is not connected. Then for distinct $y, z \in V$, the lines $[y, b]$ and $[z, b]$ cannot intersect. Consider one of the connected components. In this component, the sets $P_y = \{[y, b] : b \in V\}$ form the parallel classes of lines. Every line in P_y intersects every line not in P_y, and never intersects any other line in P_y. Let $V_y = \{b \in V : [y, b] \in P_y\}$. Then clearly,

$|V_y| = |P_y| = 2^{n-1}$. For distinct $y, z \in V$, every line in P_y intersects every line in P_z and, since f is bijective, $V_y \cap V_z = \varnothing$ and $V_y \cup V_z = V$. If there exists a third parallel line class, we have a contradiction since $|\bigcup_{y \in V} V_y| > 2^n$. □

Theorem 2.2.6. [11] Let $f(x) = x^{2^\alpha+1}$, let $n \geq 4$ be even, and $(n, \alpha) = 1$. Then $S(n; f)$ is connected.

Proof. Fix $b \in V$. Consider a line $[y, b]$. Every point (x, a) incident with the line $[y, b]$ is incident with a line $[z, b]$ with $z \neq y$, *iff* $f(x + y) = f(x + z)$. For fixed $x \in V$, $x \neq y$, there will be $\gcd(2^\alpha + 1, 2^n - 1) = 3$ choices of $z \in V$ for which this equation holds (one of which will be $z = y$). So for each point (x, a) incident with $[y, b]$, $x \neq y$, there are two distinct lines of the form $[z, b]$ with $z \neq y$ incident with (x, a). Each such line $[z, b]$, $z \neq y$, intersecting $[y, b]$ does so twice. Thus all $2^n - 1$ lines $[z, b]$ with $z \neq y$ intersect $[y, b]$. This means that the equation $f(u + v) + f(u) = 0$ is solvable for all non-zero $v \in V$ since $f(u + v) - f(u) = u \cdot v^{2^\alpha} + v \cdot u^{2^\alpha} + v^{2^\alpha+1} = v^{2^\alpha+1}(uv^{-1} + (uv^{-1})^{2^\alpha} + 1)$. By dividing by $f(v)$, we can see that this is, in turn, equivalent to the solvability of $X^{2^\alpha} + X = 1$. The polynomial $X^{2^\alpha} + X$ has 2^{n-1} distinct images and there are 2^{n-1} elements $x \in V$ which satisfy $Tr(x) = 0$. Since $Tr(x^{2^\alpha} + x) = 0$ for all $x \in V$, it is clear that $X^{2^\alpha} + X = 1$ is solvable *iff* $Tr(1) = 0$. This holds since n is even.

Suppose the structure $S(n; f)$ splits into two semi-biplanes. We use the notation from the previous proof. Consider any $b \in V_y$. From an argument above, all q lines $[z, b]$ must be in this connected component. By considering all $b \in V_y$, we can account for $q^2/2$ lines in this way. Since we only have $q^2/2$ possible lines in this component, we have $V_y = V_z$ for all $y, z \in V$. Therefore, for every point (x, a) in our component and every $b \in V_y$, $b \neq a$, there are 3 lines of the form $[z, b]$ which contain (x, a). In addition, the line $[x, a]$ contains (x, a), so since we know that there are 2^n lines through any point, we have that $2^n = 3(2^{n-1} - 1) + 1$ whereby $n = 2$. □

CHAPTER 3. GRAPHS AND ASSOCIATION SCHEMES

Many combinatorial structures are obtained from semi-biplanes. In order to discuss such combinatorial structures, we will first review some preliminary facts about these structures. For material not covered in this chapter, as well as more detailed information about distance-regular graphs and association schemes, refer to [6], [1], or [31].

3.1 Graphs and their spectra

Definition 3.1.1. A graph is an incidence structure $\Gamma = (X, E, \mathcal{I})$ where each edge $e \in E$ can be written as an unordered pair of vertices from X. The sets X and E are referred to as the vertex set and the edge set, respectively. If $e = \{x, x\}$, for some $x \in X$, i.e, e is incident with only one vertex, then e is called a loop; otherwise, the edge e is called a link.

In a graph $\Gamma = (X, E, \mathcal{I})$, if $(x, e) \in \mathcal{I}$, then we also say that x and e are *coincident* with each other, and that x is an *end vertex* of e. For $x, y \in X$, if there is an $e \in E$ such that $e = \{x, y\}$, then we say that x and y are *adjacent*. If a graph has no loops, then we say that the graph is *simple*. In this case, we simply denote the graph Γ as a pair (X, E) as the adjacency between the vertices determines the graph. All graphs considered in what follows will be simple graphs unless otherwise stated.

Given a graph Γ, sometimes, we will use the notation $V(\Gamma)$ and $E(\Gamma)$ for the vertex set and the edge set of Γ, respectively. Let us now fix the number of vertices of Γ as $|X| = |V(\Gamma)| = v \in \mathbb{Z}^+$. Let K be a field. Most often, K will be $\mathbb{C}$ or $\mathbb{R}$. Let $Mat_X(K)$ denote the set of all $v \times v$ matrices over K. Note that $Mat_X(K)$ is *the full matrix algebra*; i.e., $Mat_X(K)$ is vector space over K that also has a ring structure with the usual operations on matrices.

Definition 3.1.2. The matrix $A \in Mat_X(K)$ is called the *adjacency matrix* of Γ if

$$A_{ij} = (A)_{ij} = \begin{cases} 1 & \text{if } \{x_i, x_j\} \in E(\Gamma) \\ 0 & \text{otherwise} \end{cases}$$

Definition 3.1.3. The algebra, as a subalgebra of $Mat_X(K)$, generated by A is called the *adjacency algebra*, denoted $\mathcal{A} = \mathcal{A}(\Gamma)$, of Γ. *i.e.*,

$$\begin{aligned} \mathcal{A} &:= \{\text{the set of all polynomials of } I, A, A^2, \ldots, \text{ with their coefficients in } K\} \\ &= \{a_0 I + a_1 A + a_2 A^2 + \cdots : a_i \in K\}. \end{aligned}$$

Note that $dim\mathcal{A}$ is finite because of the Cayley-Hamilton Theorem. Let

$$V = K^v = \left\{ \begin{bmatrix} a_1 \\ \vdots \\ a_v \end{bmatrix} : a_i \in K \right\}$$

and let θ_i denote the eigenvalues of A with corresponding eigenspace $V_i = \{\mathbf{x} \in V : A\mathbf{x} = \theta_i \mathbf{x}\}$.

Definition 3.1.4. The *spectrum* of $A(\Gamma)$ (or of Γ) is the array consisting of two rows, whose entries are the eigenvalues, θ_i, of $A(\Gamma)$ and their (algebraic) multiplicities m_i, as in

$$Spec(\Gamma) := \begin{pmatrix} \theta_0 & \theta_1 & \cdots & \theta_d \\ m_0 & m_1 & \cdots & m_d \end{pmatrix}$$

where $\theta_0 > \theta_1 > \cdots > \theta_d$.

The spectrum and adjacency algebra of a graph help us to understand the combinatorial structure and properties of the given graph. We will now recall some related known results in algebraic graph theory.

Let $V(\Gamma) = \{x_1, x_2, \ldots, x_v\}$. A *path* (*walk*) of length ℓ in Γ, from x to y, is a sequence of $(\ell + 1)$ vertices $x = x_0, x_1, \cdots, x_l = y$ such that $\{x_{i-1}, x_i\} \in E(\Gamma)$ for all $i \in \{1, 2, \ldots, \ell\}$. The *distance*, $\partial(x, y)$, between two vertices x and y is the length of a shortest path connecting x

and y. A *connected* graph is a graph having at least one path between any two vertices of Γ, and the *diameter* $d = d(\Gamma)$ of a graph Γ is $max\{\partial(x,y) : x,y \in V(\Gamma)\}$.

Let $V = K^v$, the set of column vectors whose coordinates are indexed by $X = \{x_1, ...x_v\}$.

$\mathcal{A} \subset Mat_X(K)$ acts on $V = \langle \hat{x_1}, \hat{x_2}, \cdots, \hat{x_v} \rangle$ where $\hat{x_i} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \leftarrow i^{th}$.

Let $\langle , \rangle$ denote the Hermitian inner product $\langle a, b \rangle = a^t \bar{b}$, $a, b \in V$. Given a graph $\Gamma = (X, E)$, and $x \in X$, let $\Gamma(x) = \{y \in X : \{x, y\} \in E\}$, the set of vertices adjacent to x. $\mathcal{A} = \mathcal{A}(\Gamma)$ acts on V by $A\hat{x} = \sum_{y \in \Gamma(x)} \hat{y}$ for all $x \in X$. Since A is a real symmetric matrix, V can be decomposed as $V = V_1 + \cdots + V_d$ for some positive integer d.

Let $E_i \in Mat_X(K)$ be the orthogonal projection map $E_i : V \to V_i$. Then $E_i E_j = \delta_{ij} E_i$ where $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$ and $\mathcal{A} = Span_K\{E_0, E_1, \cdots, E_d\}$.

Lemma 3.1.5. Let A be the adjacency matrix of Γ. Then $(A^l)_{ij}$ indicates the number of paths of length l connecting x_i and x_j.

Proof. (We will use induction on l)

Since $A^0 = I$ and $A^1 = A$, this clearly holds for $l = 0, 1$. Assume the result is true for $l \leq m$. We will show that it works for $l = m + 1$. The number of paths of length $m + 1$ connecting x_i and x_j is equal to the number of paths of length m connecting x_i and any vertex in $\Gamma(x_j)$. Thus, this number equals $\sum_{h=1}^{v} (A^m)_{ih} A_{hj} = (A^{m+1})_{ij}$. This completes the proof. □

Lemma 3.1.6. If Γ is a connected graph with diameter d, then dim $\mathcal{A}(\Gamma) \geq d + 1$.

Proof. Since the diameter of Γ is d, there exist $x, y \in V(\Gamma)$ such that $\partial(x, y) = d$. Let $x = x_0, x_1, ..., x_d = y$ be a shortest path connecting x and y. For all $i = 1, 2, ..., d$, $\partial(x_0, x_i) = i$, and there exists no path of length less than i connecting x_0 to x_i. That is, $(A^i)_{0i} \neq 0$, and $(A^h)_{0i} = 0$ for all $h = 1, 2, ..., (i - 1)$. This means that A^i and $\{A^0, A^1, ..., A^{i-1}\}$ are linearly

independent. Consequently, we have a linearly independent set $\{A^0, ..., A^d\}$, and thus the $\dim \mathcal{A}(\Gamma) \geq d+1$. □

Corollary 3.1.7. If Γ is a connected graph, and $A = A(\Gamma)$, then the number of distinct eigenvalues of $A \geq d+1$.

As we previously mentioned, the Spec(Γ) provides a lot of useful information about Γ itself. We will now consider some of the information that it gives us.

Definition 3.1.8. Let $k(x) := |\Gamma(x)| = |\{y \in V(\Gamma) : \{x, y\} \in E(\Gamma)\}|$; the number of neighbors of x, which is called the *degree* or *valency* of x.

Definition 3.1.9. A graph is *regular* with degree k if $k(x) = k$ for all $x \in V(\Gamma)$.

Theorem 3.1.10. With the above notation, we have

(a) $\theta_0 \leq max\{k(x) : x \in V(\Gamma)\}$,

(b) If Γ is a regular graph with degree k, then $\theta_0 = k$.

Proof. Consider $V = K^v$. For each $a \in V$, $a = \sum a_x \hat{x} \in V$. Since θ_0 is an eigenvalue of A, there exists $a \in V$ such that $Aa = \theta_0 a$. Now, $Aa = A(\sum_{x \in X} a_x \hat{x}) = \sum_{x \in X} a_x A\hat{x} = \sum_{x \in X} a_x \sum_{y \in \Gamma(x)} \hat{y} = \sum_{x \in X} (\sum_{y \in \Gamma(x)} a_y)\hat{x}$ (by switching the order of the sum). On the other hand, $\theta_0 a = \theta_0 (\sum_{x \in X} a_x \hat{x}) = \sum_{x \in X} (\theta_0 a_x)\hat{x}$. Therefore, $Aa = \theta_0 a$ implies that $\sum_{x \in X} (\sum_{y \in \Gamma(x)} a_y)\hat{x} = \sum_{x \in X} (\theta_0 a_x)\hat{x}$, and thus, $\theta_0 a_x = \sum_{y \in \Gamma(x)} a_y$ which gives, $|\theta_0 a_x| \leq k(x) \cdot |a_x|$ where $|a_x|$ is the largest among all x. Hence, we have (a) $\theta_0 \leq k(x)$. For (b), we observe that the all-1 vector $a = \sum_{x \in X} \hat{x}$ satisfies $Aa = ka$ if Γ is regular with degree k. Thus, $k \leq \theta_0 \leq max\{k(x)\} = k$. This completes the proof. □

Definition 3.1.11. A symmetric $v \times v$ matrix C is *bipartite* if there exists a partition $X^+ \dot{\bigcup} X^- = \{1, 2, ..., v\}$ such that $C_{ij} = 0$ for all $i, j \in X^+$ and for all $i, j \in X^-$. A graph Γ is *bipartite* if there exists a partition $X^+ \dot{\bigcup} X^- = V(\Gamma)$ such that each edge of Γ consists of one vertex from X^+ and one vertex from X^-. A graph Γ is called *complete bipartite* if there exists a partition $X^+ \dot{\bigcup} X^- = V(\Gamma)$ such that edge set of Γ consists of all two element subsets consisting of a vertex in X^+ and a vertex in X^-.

Definition 3.1.12. Let $\Gamma = (X, E)$ be a graph. The *subgraph* of $\Gamma = (X, E)$ induced on X' is the graph with vertex set X' whose edge set consists of the edges of Γ that are contained in X'.

Given a graph Γ, define a relation $\sim$ on $V(\Gamma)$ by the existence of path connecting two vertices; i.e. $x \sim y$ *iff* there exists a path connecting x and y. Then $\sim$ is an equivalence relation on $V(\Gamma)$ and the equivalence classes of $\sim$ are called *connected components* of Γ. A graph Γ is said to be connected if Γ has one component; i.e., any two vertices in Γ are connected by a path.

Proposition 3.1.13. (Perron-Frobenius Theorem) If Γ is a connected graph with $Spec(\Gamma) = \begin{pmatrix} \theta_0 & \theta_1 & \cdots & \theta_d \\ m_0 & m_1 & \cdots & m_d \end{pmatrix}$, then

(a) For any vertex $a \in V_0$, the coordinates of a are all positive or all negative.

(b) $dim V_0 = 1$

(c) For all $i = 0, 1, ..., d$, $|\theta_i| \leq \theta_0$.

(d) Γ is bipartite if and only if $\theta_d = -\theta_0$.

Proof. This is a corollary of the well-known Perron-Frobenius Theorem proved in Theorem 3.1.1 of [6]. □

Definition 3.1.14. A path joining x and y is said to be *closed* whenever the initial vertex x and the terminal vertex y are identical.

Corollary 3.1.15. Let Γ be a graph with $Spec(\Gamma) = \begin{pmatrix} \theta_0 & \theta_1 & \cdots & \theta_d \\ m_0 & m_1 & \cdots & m_d \end{pmatrix}$. Then $\sum_{i=0}^{d} m_i \theta_i^l$ is the total number of closed path in Γ of length l where $l \in \mathbb{Z}^+ \cup \{0\}$.

In particular,

1. If $l = 0$, then $\sum_{i=0}^{d} m_i = |V(\Gamma)|$

2. If $l = 1$, then $\sum_{i=0}^{d} m_i\theta_i = 0$

3. If $l = 2$, then $\sum_{i=0}^{d} m_i\theta_i^2 = 2|E(\Gamma)|$

Definition 3.1.16. A graph is *complete* if, for each (distinct) pair of vertices, there exists an edge connecting them. That is, each vertex is adjacent to every other vertex in the graph.

Complete graphs are also referred to as *cliques* and the complete graph on v vertices is denoted by K_v. The spectrum of the complete graph on v vertices, then, is given by $Spec(K_v) = \begin{pmatrix} v-1 & -1 \\ 1 & v-1 \end{pmatrix}$.

Lemma 3.1.17. Let $\Gamma = (X, E)$ be a graph. Then Γ has minimum eigenvalue ≥ -1 (*i.e.* $\theta_i \geq -1$ for all i) *iff* Γ is a disjoint union of complete graphs.

Proof. ($\Leftarrow$) Straightforward.

($\Rightarrow$) Define a relation $\sim$ on X by $x \sim y$ *iff* $x = y$ or $\{x, y\} \in E$. Let $A = A(\Gamma)$. Then $A + I$ has nonnegative eigenvalues $1 + \theta_i$ as $\theta_i \geq -1$. So $A + I$ is positive semidefinite and thus $(A + I)_{ij} = \langle u_i, u_j \rangle$ for some vectors u_i and u_j. Note that u_i is a unit vector because the diagonal entry of $A + I$ is 1. Furthermore, we can show that $x \sim y \Leftrightarrow u_x = u_y$. This guarantees that $\sim$ is an equivalence relation on $V(\Gamma)$. Hence the vertex set X is partitioned by the equivalence classes and each class is a clique (i.e. a complete subgraph) as we desired. We now verify that $x \sim y \Leftrightarrow u_x = u_y$ as follows: If $x \sim y$ then $(A + I)_{xy} = \langle u_x, u_y \rangle = 1$, which implies that $u_x = u_y$ since $\langle u_x, u_y \rangle = |u_x||u_y|$. On the other hand, if x is not equivalent to y, then $(A + I)_{xy} = \langle u_x, u_y \rangle = 0$ which implies that $u_x \neq u_y$. This completes the proof. □

Definition 3.1.18. Let $\Gamma = (X, E)$ be a graph. A line graph $L(\Gamma)$ of Γ is a graph whose vertex set is the set of edges in Γ and whose edge set consists of the pairs of edges of Γ which have a common vertex.

We note that if Γ is a regular graph with degree $k \geq 2$ then $L(\Gamma)$ is also regular with degree $2(k - 1)$. Due to the following well-known lemma, we can determine the spectrum of a line graph in terms of the spectrum of the original graph.

Lemma 3.1.19. Let Γ be a regular graph on v vertices with degree k.

If $Spec(\Gamma) = \begin{pmatrix} k & \theta_1 & \dots & \theta_{s-1} \\ 1 & m_1 & \dots & m_{s-1} \end{pmatrix}$, then

$$Spec(L(\Gamma)) = \begin{pmatrix} 2k-2 & k-2+\theta_1 & \dots & k-2+\theta_{s-1} & -2 \\ 1 & m_1 & \dots & m_{s-1} & m-v \end{pmatrix}, \text{ where } m = \tfrac{1}{2}vk.$$

Let Γ^c be the complement of Γ. That is, let Γ^c be the graph which has the same vertex set as Γ, but the complementary edge set. In other words, two vertices are adjacent in Γ^c *iff* they are not adjacent in Γ. The spectrum of the complement graph is easily obtained from the following theorem by Cretkovic.

Theorem 3.1.20. If θ is an eigenvalue of Γ with multiplicity $m \geq 2$, then Γ^c has an eigenvalue $-1-\theta$ with multiplicity m_c, where $m-1 \leq m_c \leq m+1$

An induced subgraph Λ on V_1 of $\Gamma = (V, E)$ is the graph with $V(\Gamma) = V_1 \subseteq V$, and all edges between pair of vertices of V_1 in Γ from the edge set of Γ; i.e., $E(\Lambda) = E(\Gamma) \cap \mathcal{P}_2(V_1)$.

3.2 Distance-regular graphs

Definition 3.2.1. A *strongly regular graph* with parameters (v, k, λ, μ) is a graph, Γ, with v vertices, which is neither complete nor the complement of a complete graph, in which

$$|\Gamma(x) \cap \Gamma(y)| = \begin{cases} k & \text{if } x = y \\ \lambda & \text{if } \{x, y\} \in E(\Gamma) \\ \mu & \text{if } \{x, y\} \notin E(\Gamma) \end{cases}$$

Such a graph will be denoted by $\mathrm{SRG}(v, k, \lambda, \mu)$.

Lemma 3.2.2. If Γ is a $\mathrm{SRG}(v, k, \lambda, \mu)$, then $k(k-\lambda-1) = (v-k-1)\mu$.

Proof. Count $|\{\{y, z\} \in E(\Gamma) : y \in \Gamma(x) \text{ and } z \notin \Gamma(x)\}|$ in two ways. □

Lemma 3.2.3. The complement of a $\mathrm{SRG}(v, k, \lambda, \mu)$ is also a strongly regular graph with parameters

$$(\overline{v}, \overline{k}, \overline{\lambda}, \overline{\mu}) = (v, v-1-k, v-2-2k+\mu, v-2k+\lambda).$$

Note 3.2.4. $\overline{\lambda}$ and $\overline{\mu}$ should be nonnegative integers. Thus, if there exists SRG(v,k,λ,μ), then $\begin{cases} v-2k+\mu-2 \geq 0 \\ v-2k+\lambda \geq 0 \end{cases}$

Let A be the adjacency matrix of the SRG(v,k,λ,μ) Γ. Then,

1. the all-1 vector $\mathbf{1}$ is an eigenvector of A and J.

2. Since rank$(J)=1$, all eigenvectors of J corresponding to the eigenvalue 0 are orthogonal to $\mathbf{1}$.

Eigenvalues, $\rho \neq k$ of A must satisfy the relation $A^2 = kI + \lambda A + \mu(J-I-A)$. So $\rho^2 = (k-\mu)+(\lambda-\mu)\rho$. Hence if $\rho = r, s$ $(r>s)$ then $\rho = \frac{1}{2}\left\{(\lambda-\mu) \pm \sqrt{(\lambda-\mu)^2+4(k-\mu)}\right\}$. To find the corresponding multiplicities $m_1 = f$ and $m_2 = g$ for $\theta_1 = r$ and $\theta_2 = s$, respectively, use the identities

$$\begin{cases} v = f+g+1 \\ 0 = Tr(A) = fr+gs+k \end{cases}.$$

Since $k = \lambda = \mu$ is impossible,

$$f,g = \frac{1}{2}\left\{(v-1) \pm \frac{(v-1)(\mu-\lambda)-2k}{\sqrt{(\mu-\lambda)^2+4(k-\mu)}}\right\}.$$

Since f and g must be integers, the right-hand side of the equation must also correspond to an integer value.

Definition 3.2.5. Let $\Gamma(V,E)$ be a simple connected regular graph. Γ is called a distance-regular graph (DRG) with diameter d if it satisfies the following distance-regularity condition: For all $h,i,j \in \{0,1,...,d\}$, and x,y with $\partial(x,y)=h$, the number

$$p_{ij}^h = |\{z \in V : \partial(x,z)=i,\ \partial(z,y)=j\}|$$

is constant in that it depends only on h,i,j but does not depend on the choice of x and y. If $\Gamma_i(x) = \{y \in V : \partial(x,y)=i\}$, then $p_{ij}^h := |\Gamma_i(x) \cap \Gamma_j(y)|$ for all $x,y \in V(\Gamma)$ with $\partial(x,y)=h$. This number is called the *intersection number*.

Let $\partial(x,y) = i$ and define

$$a_i := p^i_{1,i} = |\Gamma_1(x) \cap \Gamma_i(y)|$$

$$b_i := p^i_{1,i+1} = |\Gamma_1(x) \cap \Gamma_{i+1}(y)|$$

$$c_i := p^i_{1,i-1} = |\Gamma_1(x) \cap \Gamma_{i-1}(y)|$$

Then $b_0 = p^0_{1,1} = |\Gamma(x)| = k$; the degree of Γ, and we will define $c_0 := 0$, $b_d := 0$, and $a_0 := 0$. The following matrix is called the *intersection array* (matrix)

$$B\ (= B_1) = \begin{pmatrix} a_0 & c_1 & & & \\ b_0 & a_1 & c_0 & & \\ & \ddots & \ddots & \ddots & \\ & & & & c_d \\ & & & b_{d-1} & a_d \end{pmatrix}.$$

$$a_0 = 0,\ c_1 = 1,\ b_0 = k$$

$$c_0 \leq c_1 \leq \ldots \leq c_d$$

$$b_0 \geq b_1 \geq \ldots \geq b_d$$

$$c_i + a_i + b_i = k$$

The $d+1$ intersection matrices $B_0, B_1, B_2, \ldots, B_d$ of Γ are defined as follows:

$$B_i = ((B_i)_{jh}) \quad \text{where} \quad (B_i)_{jh} = p^h_{ij}$$

Given a DRG we also have $d+1$ matrices $A_0,\ A_1,\ \ldots,\ A_d$, called the *adjacency matrices* of Γ. Let $A_1 := A$, the (usual) adjacency matrix of the graph Γ, $A_0 := I$, and define A_i for $i = 2, \ldots, d$ by

$$(A_i)_{xy} = \begin{cases} 1 & \text{if } \partial(x,y) = i \\ 0 & \text{otherwise} \end{cases}.$$

Theorem 3.2.6. $A_iA_j = \sum_{h=0}^{d} p_{i,j}^h A_h$ for all i, j, $1 \leq i, j \leq d$.

Proof. By comparing the (x, y)-entry of both sides of the equation, we can count the coordinates. $(A_iA_j) = p_{i,j}^0 A_0 + p_{i,j}^1 A_1 + \ldots + p_{i,j}^d A_d$. □

The adjacency matrices of Γ, $A_0, A_1, \ldots, A_d$, yield a basis for the adjacency algebra $A(\Gamma) = \langle A_0, A_1, ..., A_d \rangle$ of Γ. $\{A_0, A_1, \ldots, A_d\}$ and $\{B_0, B_1, \ldots, B_d\}$ are known to be algebraically anti-isomorphic.

Remark 3.2.7. All DRGs with diameter 2 are SRGs. Conversely, all connected SRGs are DRGs with diameter 2.

There are many interesting distance-regular graphs, but the distance-regular graphs we are particularly interested in are the Hamming cubes.

Definition 3.2.8. Let F be a set of q-elements ($q \geq 2$), and let $X = F^n$; the set of ordered n-tuples of elements of F. A Hamming graph, $\Gamma = H(n, q)$, is a graph whose vertex set is given by $V(\Gamma) = X = \{x = (x_1, x_2, \ldots, x_n) \in F^n\}$, and whose edge set, $E(\Gamma)$, is defined by

$$(x, y) \in E(\Gamma) \quad \textit{iff} \quad x_i = y_i \text{ for all but one } i$$

where $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n) \in V(\Gamma)$. The Hamming graph is a DRG with diameter $d = n$.

In $\Gamma = H(n, q)$, the degree of Γ is $k = |\Gamma(x)| = b_0 = n(q-1)$, and the intersection numbers and intersection matrix of $H(n, q)$ are given by

$c_i = i$,

$b_i = (n-i)(q-1)$,

$a_i = k - b_i - c_i = n(q-1) - i - (n-i)(q-1) = i(q-2)$, and

$$B = \begin{pmatrix} a_0 & 1 & & & \\ n(q-1) & a_1 & 2 & & \\ & \ddots & \ddots & \ddots & \\ & & & & n-1 \\ & & & (q-1) & a_n \end{pmatrix}.$$

That is,

$$p_{i,j}^h = \sum_{l=0}^{n} (q-2)^{j+2l-i-h}(q-1)^{i-l} \begin{pmatrix} l \\ j+2l-i-h \end{pmatrix} \begin{pmatrix} h \\ l \end{pmatrix} \begin{pmatrix} n-h \\ i-l \end{pmatrix}.$$

When $q = 2$ and $F = \mathbb{F}_2 = \{0, 1\}$, the Hamming graphs are known as Hamming cubes.

We include the definition of 'cospectral graphs' here for later use in characterizing graphs.

Definition 3.2.9. Cospectral graphs are the graphs with the same spectrum.

There are many nonisomorphic cospectral distance-regular graphs. Recently, van Dam and Haemers [34] found 100 cospectral graphs on 35 vertices, and no fewer than 33525 cospectral graphs on 56 vertices. Among the smallest distance-regular graphs, the Hamming $H(2,4)$ graph, which is also known as the 'lattice graph, $L_2(4)$,' and the line graph $L(K_{4,4})$ of the complete bipartite graph $K_{4,4}$, have a cospectral graph known as the 'Shrikhande graph.'(cf. p.105 in [6].) Doob graphs which are obtained by taking the direct product of copies of $H(2,4)$ and the Shrikhande graph are all cospectral as long as the number of factors is the same.

3.3 Halved-folded hamming cubes

Definition 3.3.1. Suppose Γ is a bipartite distance-regular graph with bipartition $V(\Gamma) = X \cup Y$. Let $V(\Gamma') = X$. For $x, y \in X$, x and y are adjacent in Γ' *iff* $\partial(x,y) = 2$ in Γ. Γ' is distance-regular and is called the *halved graph* of Γ.

It might also be interesting to note that Γ has another halved graph with vertex set Y which need not be isomorphic to Γ', however, this fact has no bearing on what follows. Let

the parameters of a bipartite distance-regular graph Γ be designated as a_i, b_i, c_i. Those of its halved graph Γ' will be designated a'_i, b'_i, c'_i. Since Γ is a bipartite graph, it has no odd cycles, which forces $a_i = 0$, for all i. As a result, $b_i + c_i = k$, the degree of Γ. Due to Gardiner [19], we get: If $c_3 = c_2$, then $c_3 = c_2 = 1$ and by Hemmeter [23], we also have:

Lemma 3.3.2. $b'_i = \frac{b_{2i}b_{2i+1}}{c_2}$ and $c'_j = \frac{c_{2j}c_{2j-1}}{c_2}$.

Proof. Let x and y be vertices of Γ' such that $x \in \Gamma_{2i}(y)$. Count the number of pairs (w, z) with $w \in \Gamma_1(y) \cap \Gamma_{2i+1}(x)$ and $z \in \Gamma_1(w) \cap \Gamma_{2i+2}(x)$. There are b_{2i} such w, each of which has b_{2i+1} z. Counting z's first gives $b'_i c_2$. This implies the former. The latter can be proved in the same manner, with x, y being as above, by counting the pairs (w, z) with $w \in \Gamma_1(y) \cap \Gamma_{2j-1}(x)$, $z \in \Gamma_1(w) \cap \Gamma_{2j-2}(x)$. □

Lemma 3.3.3. Suppose Γ' is not a complete graph. Then for every $y \in Y$, $\Gamma(y)$ is a maximal clique in Γ'. Furthermore, if $y_1 \neq y_2$, then $\Gamma(y_1) \neq \Gamma(y_2)$.

We now introduce another class of distance-regular graphs that are obtained from 'antipodal distance-regular graphs.'

Definition 3.3.4. Let Γ be a distance-regular graph with diameter d. For x and y in $V(\Gamma)$, we say that x is *opposite* y if $\partial(x, y) = d$, or $x = y$. The graph Γ is called *antipodal* if being opposite is an equivalence relation on Γ; i.e., $x \sim y$ in Γ *iff* x and y are opposite. For an antipodal graph Γ, we can define a graph $\overline{\Gamma}$ whose vertices are the equivalence classes of Γ. If $\bar{x}$ and $\bar{y}$ are in $V(\overline{\Gamma})$, we say that $\bar{x}$ is in $\overline{\Gamma}(\bar{y})$ if for some $x \in \bar{x}$ and $y \in \bar{y}$, $x \in \Gamma(y)$. $\overline{\Gamma}$ is distance regular, and is called the *folded graph*, or *antipodal quotient* of Γ.

Let Γ be an antipodal distance-regular graph, and let $\overline{\Gamma}$ be its folded graph. Then we have the canonical surjective map ϕ defined by $x \mapsto \bar{x}$.

Lemma 3.3.5. Let $x \in V(\Gamma)$ and Υ be the subgraph induced by $\bigcup_{j=0}^{[\frac{d}{2}]-1} \Gamma_j(x)$. Then ϕ restricted to Υ is an isomorphism onto $\phi(\Upsilon)$. If $d \geq 4$, $\phi(\Upsilon)$ contains all of the neighbors of $\bar{x}$.

Corollary 3.3.6. If $d \geq 4$, then the structure of the cliques which contain a vertex $\bar{x} \in V(\overline{\Gamma})$ is isomorphic to the structure of the cliques containing the vertex $x \in V(\Gamma)$.

Example 3.3.7. Halved n-cubes. The halved n-cube H_n is constructed from a binary code consisting of all even weight words of length n. Words are defined to be adjacent if they differ in two coordinate entries; i.e., if their Hamming distance is 2. H_n is a distance-regular graph which has 2^{n-1} vertices with degree $k = \binom{n}{2}$ and diameter $d = \lfloor \frac{n}{2} \rfloor$. For $n \geq 3$,

$$b_j = \frac{1}{2}(n-2j)(n-2j-1)$$

$$c_j = j(2j-1)$$

$$\theta_j = \frac{1}{2}(n-2j)^2 - \frac{1}{2}n$$

$$m_j = \binom{n}{j}, \quad (2j < n)$$

(If n is even, then $m_d = \frac{1}{2}\binom{n}{j}$). By Neumaier [28] and Terwilliger [32], the halved n-cubes are characterized by their intersection arrays. That is, all graphs which are cospectral with the halved n-cubes are isomorphic.

Example 3.3.8. Folded n-cubes. Since the Hamming n-cubes $H(n,2)$ are antipodal distance-regular graphs, they yield folded graphs, which are called 'folded n-cubes.' For $n \geq 3$, the intersection array of the folded n-cube is given by $d = \lfloor \frac{n}{2} \rfloor$, $b_j = n - j$, $c_j = j$ $(2j < n)$, and, if n is even, $c_d = n$. The eigenvalues θ_j and multiplicities m_j are $\theta_j = n - 4j$ and $m_j = \binom{n}{2j}$.

Proposition 3.3.9. For $n \neq 6$, the folded n-cube is uniquely characterized by its intersection array and for the folded 6-cube, there are precisely three nonisomorphic cospectral graphs.

Proof. cf. [5] or [33]. □

Example 3.3.10. Halved and folded $2l$-cubes. The *halved folded $2l$-cube*, $l \geq 3$ an integer, is a distance-regular graph with diameter $d := \lfloor \frac{l}{2} \rfloor$ and parameters

$$b_i = (l-i)(2l-2i-1), \qquad c_i = i(2i-1), \qquad (0 \leq i \leq d-1)$$

$$c_d = \begin{cases} d(2d-1), & \text{if } l \text{ is odd;} \\ 2d(2d-1), & \text{if } l \text{ is even.} \end{cases}$$

In particular, the halved and folded 8-cube and 10-cube are strongly regular graphs with parameters $(v, k, \lambda, \mu) = (64, 28, 12, 12)$ and $(256, 45, 16, 6)$, respectively. These graphs are not uniquely determined by their parameters. For example, the block graph of a 'transversal design' $TD[4; 8]$ (the graph obtained from the orthogonal array $OA[4, 8]$) or $TD[3; 16]$ is cospectral but it is not isomorphic to folded halved 8-cube or 10-cube, respectively. However, it is known that folded halved n-cubes are characterized by their parameters if n is sufficiently large.

3.4 Association schemes

Definition 3.4.1. Let X be a finite set and $R_0, R_1, \ldots, R_d$ be a (non-empty) binary relation of X; i.e., $R_i \subseteq X \times X = \{(x, y) : x, y \in X\}$. The configuration $(X, \{R_i\}_{0 \le i \le d})$ is called an association scheme if it satisfies the following axioms:

1. $R_0 = \{(x, x) : x \in X\}$, the diagonal relation
2. $R_0 \dot{\cup} R_1 \dot{\cup} \ldots \dot{\cup} R_d = X \times X$
3. $R_i^T = \{(y, x) : (x, y) \in R_i\}$ must be a member of $\{R_0, R_1, \ldots, R_d\}$. Denote $R_i^T = R_{i'}$ for some $i' \in \{1, 2, \ldots, d\}$
4. For any $h, i, j \in \{1, 2, \ldots, d\}$, the number $p_{ij}^h(x, y) = |\{z \in X : (x, z) \in R_i, (z, y) \in R_j\}|$ does not depend on the choice of x and y, but depends only on h, where $(x, y) \in R_h$.

Furthermore, if $\mathcal{X} = (X, \{R_i\}_{0 \le i \le d})$ is an association scheme that satisfies $p_{ij}^h = p_{ji}^h$ for all $h, i, j = 1, 2, \ldots, d$, then $\mathcal{X}$ is called commutative, and if $R_i^T = R_i$, then $\mathcal{X}$ is called symmetric.

Definition 3.4.2. If $\mathcal{X} = (X, \{R_I\}_{0 \le i \le d})$ is a commutative association scheme of class d, and the matrices $A_0, A_1, \ldots, A_d$ are adjacency matrices of $\mathcal{X}$ such that $(A_i)_{xy} = \begin{cases} 1 & \text{if } (x, y) \in R_i \\ 0 & \text{otherwise} \end{cases}$, then

1. $A_0 = I$
2. $A_0 + A_1 + \cdots + A_d = J$ and $(A_i)_{xy} = 1$ *iff* $(A_j)_{xy} = 0$ for all $j \neq i$

3. $A_i^T = A_{i'}$ for some $i' \in \{0, 1, \ldots, d\}$

4. $A_i A_j = \sum_{h=0}^{d} p_{ij}^h A_h$

5. $A_i A_j = A_j A_i$

6. $\mathcal{X}$ is symmetric $\Leftrightarrow A_i^T = A_i$ for all i.

A symmetric association scheme $\mathcal{X}$ of class d can be viewed as a color partition of a complete graph on X satisfying certain properties and each non-diagonal (symmetric) relation R_i of $\mathcal{X}$ can be thought of as the graph (X, R_i) on X.

Lemma 3.4.3. Let $\mathcal{X} = (X, \{R_i\}_{0 \le i \le d})$ be a d-class commutative association scheme and let $A_0, A_1, \ldots, A_d$ be adjacency matrices, (for convenience, let $A_{-1} = A_{d+1} = O$). Then we have the following:

1. $A_0, A_1, \ldots, A_d$ linearly independent

2. For each $x \in X$, and each $i = 0, 1, \ldots, d$, $A_i \hat{x} = \sum_{y \in X,\ (y,x) \in R_i} \hat{y}$

3. Let $k_i = p_{ii'}^0 \quad 0 \le i \le d$.

   (a) For each $i \in \{0, 1, \ldots, d\}$, and each $x \in X$, $k_i = |\{z \in X : (x, z) \in R_i\}|$

   (b) $k_0 = 1, \quad k_i > 0$

   (c) $k_i = k_{i'}$

   (d) $|X| = \sum_{i=0}^{d} k_i$

4. The intersection numbers have the following identities:

   (a) $p_{0j}^h = \delta_{hj}, \quad p_{i0}^h = \delta_{hi}$

   (b) $p_{ij}^0 = \delta_{ij'} k_i$

   (c) $p_{ij}^h = p_{i'j'}^{h'}$

   (d) $\sum_{i=0}^{d} p_{ij}^h = k_j$

(e) $k_h p_{ij}^h = k_j p_{i'h}^j = k_i p_{hj'}^i$

(f) $\sum_{\alpha=0}^{d} p_{\alpha r}^h p_{ij}^\alpha = \sum_{\beta=0}^{d} p_{i\beta}^h p_{jr}^\beta$

5. Using facts from linear algebra about the positive definite hermitian bilinear form $\langle B, C\rangle := trace(B\overline{C}^T)$, where $\overline{C}$ is the complex conjugate of $C \in M_n(\mathbb{C})$, we also have:

   (a) $\langle A_i, A_j\rangle = \delta_{ij} k_i |X|$

   (b) $p_{ij}^h = |X|^{-1} k_j^{-1} \langle A_i A_j, A_j\rangle = |X|^{-1} k_h^{-1} \langle A_h, A_i A_j\rangle$

   (c) $A_i J = k_i J$ for all $i \in \{0, 1, \ldots, d\}$

Proof. (5a) can be proved as follows: $\langle A_h, I\rangle = trace(A_h \bar{I}^T) = trace(A_h) = \delta_{0h}|X|$.
$\langle A_i, A_j\rangle = \left\langle A_i \bar{A_j}^T, I\right\rangle = \langle A_i A_{j'}, I\rangle = \sum_h p_{ij'}^h \langle A_h, I\rangle = \sum_h p_{ij'}^h \delta_{0h}|X| = p_{ij'}^0 |X| = \delta_{ij} k_i |X|$.
For proofs of the rest of the parts, refer to Bannai and Ito [1] or Brouwer, Cohen, and Neumaier [6]. □

Theorem 3.4.4. Let $\mathcal{X} = (X, \{R_i\}_{0\le i\le d})$ be a commutative association scheme and let $\mathcal{A} = \langle A_0, A_1, \ldots, A_d\rangle_{\mathbb{C}}$ be its Bose-Mesner algebra. Then there exists another basis $E_0, E_1, \ldots, E_d$ for $\mathcal{A}(\mathcal{X})$ such that

1. $E_0 = |X|^{-1} J$

2. $I = E_0 + E_1 + \cdots + E_d$

3. $E_i E_j = \delta_{ij} E_i$ $0 \le i \le d$

Moreover, $\{E_0, E_1, \ldots, E_d\} = \{E \in \mathcal{A} : E^2 = E,\ dim(\mathcal{A}E) = 1\}$. We refer to E_i as the i^{th} primitive idempotent of $\mathcal{X}$. E_0 is called the trivial idempotent.

Proof. Let L be an ideal in $\mathcal{A}$, that is, let $L \subseteq \mathcal{A}$ be a subspace such that $\mathcal{A}L \subseteq L$. For example, $L_0 := Span\{J\}$ is an ideal of $\mathcal{A}$ where $A_0, A_1, \ldots, A_d$ is a basis for $\mathcal{A}$. Let L denote any ideal in $\mathcal{A}$. Then L is said to be minimal whenever $L \neq O$, and L contains no ideal of $\mathcal{A}$ other than O and L. Observe that L_0 has dimension 1, so L_0 is minimal. Let L denote

any ideal in $\mathcal{A}$. Since $\mathcal{A}$ is closed under the conjugate transpose, one can easily show that the orthogonal complement of L in $\mathcal{A}$, $L^{\perp} := \{B \in \mathcal{A} : \ \langle B, C\rangle = 0 \text{ for all } C \in L\}$ is again an ideal in $\mathcal{A}$. Beginning with L_0, taking orthogonal complements, and arguing by induction, we find that $\mathcal{A}$ can be expressed as an orthogonal direct sum $L_0 + L_1 \cdots + L_r$ of minimal ideals for some r. Let L denote any minimal ideal of $\mathcal{A}$. We will show that L has dimension 1. To see this, we pick any $B \in \mathcal{A}$ and claim that there exists a $\theta \in \mathbb{C}$ such that

$$(B - \theta I)L = O. \tag{3.1}$$

To obtain (3.1), let $\theta_1, \theta_2, \ldots, \theta_v$, where $v = |X|$, denote the eigenvalues of B (including multiplicities). By the Cayley-Hamilton theorem from elementary linear algebra

$$\prod_{i=1}^{v}(B - \theta_i I) = O. \tag{3.2}$$

Suppose (3.1) fails. Then $(B - \theta_i I)L \neq O$ for $1 \leq i \leq v$. Since $\mathcal{A}$ is commutative, each $(B - \theta_i I)L$ is an ideal of $\mathcal{A}$ contained in L, so $(B - \theta_i I)L = L$ $(1 \leq i \leq v)$ by the minimality of L. Now $\prod_{i=1}^{v}(B - \theta_i I)L = L$ contradicting (3.2). So we now have $(B - \theta I)L = O$. But this implies that each element of $\mathcal{A}$ acts as a scalar multiple of the identity on L, so any 1-dimensional subspace of L is an ideal of $\mathcal{A}$. It follows from the minimality of L that L has dimension 1. So, since $\mathcal{A}$ has dimension $(d+1)$,

$$\mathcal{A} = L_0 + L_1 + \cdots + L_d. \tag{3.3}$$

Now, by (3.3) and since $I \in \mathcal{A}$, there exists an $E_i \in L_i$ $(0 \leq i \leq d)$ such that $I = E_0 + E_2 + \cdots + E_d$ and since $L_0, L_1, \ldots, L_d$ are ideals in $\mathcal{A}$, $L_i L_j \subseteq L_i \cap L_j = O$ if $i \neq j$. In particular, $E_i E_j = O$ if $i \neq j$. For all i $(0 \leq i \leq d)$ and for all $B \in L_i$, $B = BI = B(E_0 + E_2 + \cdots + E_d) = BE_i$. Picking $B \neq O$, we find $E_i \neq O$ $(0 \leq i \leq d)$. Picking $B = E_i$, we find $E_i^2 = E_i$ $(0 \leq i \leq d)$. So by (3.3) and the above observation, we see that $E_0, \ldots, E_d$ makes a basis for $\mathcal{A}$. Using this construction, then, we can see that $E_0 = \alpha J$ for some $\alpha \in \mathbb{C}$. Since $J^2 = |X|J$ and $E_0^2 = E_0$,

we find that $\alpha = \frac{1}{|X|}$. To verify that $\{E_0, E_1, \ldots, E_d\} = \{E \in \mathcal{A} \ : \ E^2 = E, \ dim(\mathcal{A}E) = 1\}$, we need to check that the left hand side contains the right hand side. Pick any $E \in \mathcal{A}$ such that $E^2 = E$ and $\mathcal{A}E$ has dimension 1. We will show that $E = E_i$, for some i $(0 \leq i \leq d)$. Since $E_0, E_1, \ldots, E_d$ is a basis for $\mathcal{A}$, $E = \sum_{i=0}^{d} \alpha_i E_i$ for some $\alpha_i \in \mathbb{C}$. Since $E_i E_j = \delta_{ij} E_i$ and $E^2 = E$, we have that $\alpha \in \{0, 1\}$ $(0 \leq i \leq d)$. In fact, $\alpha_i = 1$ for a unique $i \in \{0, 1, \ldots, d\}$. To see this, observe that $E = \sum_{i=0}^{d} \alpha_i E_i$ and $E_i E_j = \delta_{ij} E_i$. $\mathcal{A}E = Span\{E_i E : 0 \leq i \leq d\} = Span\{\alpha_i E_i : 0 \leq i \leq d\} = Span\{E_i : 0 \leq i \leq d, \ \alpha_i \neq 0\}$ and the result follows since $\mathcal{A}E$ has dimension 1. Hence we have the equality and the theorem is proved. □

Lemma 3.4.5. Let $\mathcal{X} = (X, \{R_i\}_{0 \leq i \leq d})$ denote a scheme with primitive idempotents $E_0, E_1, \ldots, E_d$, and $V = \mathbb{C}^X$.

1. $V = E_0 V + E_1 V + \cdots + E_d V$ (direct sum)
2. For all integers i $(0 \leq i \leq d)$, E_i acts as the identity on $E_i V$, and vanishes on $E_j V$ for $j \neq i$. In other words, $E_i : V \to E_i V$ is an (orthogonal) projection map.
3. $E_0 V, E_1 V, \ldots, E_d V$ are precisely the maximal common eigenspaces of $\mathcal{A}$ acting on V.

Proof. It is enough to show that $E_0 V, E_1 V, \ldots, E_d V$ are mutually orthogonal, for all $i, j \in \{0, 1, \ldots, d\}$ and $u, v \in V$. $\langle E_i u, E_j v \rangle = \left\langle u, \overline{E_i}^T E_j v \right\rangle = \langle u, E_i E_j v \rangle = 0$ if $i \neq j$. □

Lemma 3.4.6. Let $\mathcal{X} = (X, \{R_i\}_{0 \leq i \leq d})$ be a scheme with primitive idempotents $E_0, E_1, \ldots, E_d$. Define $m_i := rank(E_i), 0 \leq i \leq d$. Then

1. $m_i = tr(E_i)$
2. $m_0 = 1$
3. $m_i > 0$
4. $m_i^{\vee} = m_i$
5. $|X| = \sum_{i=0}^{d} m_i$

Proof. E_i is a projection map. (1) The eigenvalues of an idempotent ($E^2 = E$) matrix are 0 or 1 and $tr(E_i) = \sum(eigenvalues)$= the number of 1's = $rank(E_i)$.

(2) $E_0 = \frac{1}{|X|}J$ and $m_0 = rank(E_0)$, so $m_0 = 1$. □

Lemma 3.4.7. $\langle E_i, E_j \rangle = \delta_{ij} m_i$

Proof. $\langle E_i, E_j \rangle = \left\langle E_i \overline{E_j}^T, I \right\rangle = \langle E_i E_j, I \rangle = \delta_{ij} \langle E_i, I \rangle = \delta_{ij} tr(E_i)$ □

Lemma 3.4.8. The following are equivalent.

1. $\mathcal{X}$ is symmetric.
2. all elements of $\mathcal{A}$ are symmetric.
3. $E_0, E_1, \ldots, E_d$ are symmetric.
4. $\hat{i} = i$, $\forall i$.
5. The entries of $E_0, E_1, \ldots, E_d$ are real.

Theorem 3.4.9. There exist $p_j(i)$ and $q_j(i) \in \mathbb{C}$ such that $A_j = \sum_{i=0}^{d} p_j(i) E_i$: eigenvalues of $\mathcal{X}$ and $E_j = \frac{1}{|X|} \sum_{i=0}^{d} q_j(i) A_i$: dual eigenvalues of $\mathcal{X}$

Definition 3.4.10. For $i, j \in \{0, 1, \ldots, d\}$, let P and Q be defined as shown.

$$P = P(\mathcal{X}) = \begin{array}{c} \\ i \end{array} \begin{array}{c} \overset{j}{} \\ \begin{pmatrix} & & \\ & p_j(i) & \\ & & \end{pmatrix} \end{array} = \begin{pmatrix} 1 & k_1 & k_2 & \cdots & k_d \\ 1 & & & & \\ 1 & & & p_j(i) & \\ 1 & & & & \\ 1 & & & & \end{pmatrix}$$

$$Q = Q(\mathcal{X}) = \begin{array}{c} \\ i \end{array} \begin{array}{c} \begin{array}{c} j \end{array} \\ \begin{pmatrix} \\ \\ q_j(i) \\ \\ \\ \end{pmatrix} \end{array} = \begin{pmatrix} 1 & m_1 & m_2 & \cdots & m_d \\ 1 & & & & \\ 1 & & & q_j(i) & \\ 1 & & & & \\ 1 & & & & \end{pmatrix}$$

Then P is called the first eigenmatrix (or "character table") of $\mathcal{X}$ and Q is called the second eigenmatrix of $\mathcal{X}$.

Lemma 3.4.11. For P and Q as defined above, we have

1. P and $\frac{1}{|X|}Q$ are inverses ($PQ = |X|I$).
2. $\sum_{k=0}^{d} p_k(i)q_j(k) = \delta_{ij}|X|$.
3. $\sum_{k=0}^{d} q_k(i)p_j(k) = \delta_{ij}|X|$. $0 \leq i, j \leq d$.

Proof. Immediate from the previous lemma and the definition of Q and P. □

Lemma 3.4.12. For A_i and E_j, as defined in Theorem 3.4.9, we have

1. $\langle A_i, E_j \rangle = p_i(j)m_j$
2. $\langle A_i, E_j \rangle = \overline{q_j(i)}k_i$

Proof. For (1), replace A_i by $\sum_{h=0}^{d} p_i(h)E_h$ in $\langle A_i, E_j \rangle = \left\langle \sum_{h=0}^{d} p_i(h)E_h, E_j \right\rangle$.
Since $\langle E_h, E_j \rangle = \delta_{hj}m_j$, we get (1).
For (2), use $\langle A_i, A_j \rangle = \delta_{ij}|X|k_i$, after replacing E_j by $\sum_{h=0}^{d} q_j(h)A_h$ in $\langle A_i, E_j \rangle$. □

Lemma 3.4.13.

1. $p_{i'}(j) = \overline{p_i(j)}$.

2. $p_i(\hat{j}) = \overline{p_i(j)}$.

3. $q_i(j') = \overline{q_i(j)}$.

4. $q_{\hat{i}}(j) = \overline{q_i(j)}$.

Proof. (1) Using $A_{i'} = \sum_{h=0}^{d} p_i(h)E_h$, and the conjugate transpose of $A_i = \sum_{h=0}^{d} p_i(h)E_h$, we obtain

$$\sum_{h=0}^{d} p_{i'}(h)E_h = (\sum_{h=0}^{d} p_i(h)E_h)^T$$

which implies that

$$A_{i'} = \sum_{h=0}^{d} \overline{p_i(h)}E_h.$$

The proofs for the rest are similar. □

Now we are ready for the following series of conversion formulas.

Theorem 3.4.14.

1. $\sum_{h=0}^{d} p_h(i)\overline{p_h(j)}k_h^{-1} = \delta_{ij}m_j^{-1}|X|$

2. $\sum_{h=0}^{d} p_i(h)\overline{p_j(h)}m_h = \delta_{ij}k_i|X|$

3. $\sum_{h=0}^{d} q_h(i)\overline{q_h(j)}m_h^{-1} = \delta_{ij}k_i^{-1}|X|$

4. $\sum_{h=0}^{d} q_i(h)\overline{p_j(h)}k_h = \delta_{ij}m_i|X|$

Lemma 3.4.15.

1. $\sum_{h=0}^{d} p_h(i) = \delta_{i0}|X|$ (row sum of $P = 0$ except the $0th$ row) (i.e. $\sum_{i=0}^{d} k_i = |X| - \sum_{i=0}^{d} p_i(0)$)

2. $\sum_{h=0}^{d} q_h(i) = \delta_{i0}|X|$ (i.e. $\sum_{h=0}^{d} m_i = |X| - \sum_{i=0}^{d} q_i(0)$)

3. $\sum_{h=0}^{d} p_i(h)m_h = \delta_{i0}|X|$ (weighted column sum)

4. $\sum_{h=0}^{d} q_i(h)k_h = \delta_{i0}|X|$

Theorem 3.4.16. (Conversion between $\{p_{ij}^k\}$ and $\{p_j(i)\}$ and $\{q_j(i)\}$)

1. $p_i(r)q_j(r) = \sum_{h=0}^{d} p_{ij}^h p_h(r)$ $(0 \leq i, j, r \leq d)$
2. $p_{ij}^h = \frac{1}{|X|k_h} \sum_{r=0}^{d} p_i(r)p_j(r)\overline{p_h(r)}m_r$
3. $p_{ij}^h = \frac{k_i k_j}{|X|} \sum_{r=0}^{d} q_r(i)q_r(j)\overline{q_r(h)}/m_r^2$

Due to the conversion formulas, having one of the sets $\{p_{ij}^k\}$, $\{p_j(i)\}$, and $\{q_j(i)\}$ is equivalent to having them all. That is, association schemes characterized by their parameters are also characterized by their eigenmatrices.

Some of the most important classes of symmetric association schemes are the P- and Q-polynomial association schemes (also called the metric schemes). These association schemes come from distance-regular graphs. Let $\Gamma = (X, R)$ be a distance-regular graph with diameter d. Then we get a d-class association scheme on X by defining the relations $R_1, R_2, \ldots, R_d$ by

$$(x, y) \in R_i \quad \Leftrightarrow \quad \partial(x, y) = i.$$

We now recall how to calculate the eigenvalues of the adjacency matrix of a DRG, and how to find a set of parameters $p_{i,j}^h$ that can define a DRG and its corresponding P-polynomial association scheme.

Lemma 3.4.17. Let $\Gamma = (X, R)$ be a DRG with diameter d. Then in terms of adjacency matrices (distance matrices) $A_0, A_1, \ldots, A_d$, we have $A_1 A_i = c_{i+1}A_{i+1} + a_i A_i + b_{i-1}A_{i-1}$ $(0 \leq i \leq d)$, and *wlog*, we may assume $A_{d+1} = 0, b_{-1} = a_0 = 0$, and $c_{d+1} = 1$.

Proof. (Indirect) From the definition of DRG, $A_i A_j = \sum_{h=0}^{d} p_{i,j}^h A_h$.
(Direct) Pick $x, y \in X$. By matrix multiplication, the (x, y)-entry of the right-hand side $|\{z \in X : \partial(x, y) = 1, \partial(x, y) = i\}|$ is equal to the (x, y)-entry of the left-hand side. (Consider $c_i = p_{1,i+1}^i, a_i = p_{1,i}^i, b_i = p_{1,i-1}^i$) □

Lemma 3.4.18. With the same assumptions as the previous lemma, let $\mathcal{A}$ be the Bose-Mesner algebra $\mathcal{A}(\Gamma)$ over $\mathbb{R}$ or $\mathbb{C}$.

1. The distance matrices $A_0, A_1, \cdots, A_d$ form a basis of $\mathcal{A}$.

2. $dim\mathcal{A}$ as a $\mathbb{C}$-vector space is $d+1$.

3. Γ has $d+1$ distinct eigenvalues.

Proof. (1) $A_0, A_1, \ldots, A_d$ are linearly independent since the (x,y) -entry of A_i is 1 *iff* $\partial(x,y) = i$. It suffices to show that $A_0, A_1, \ldots, A_d$ open $\mathcal{A}$. First, $A_0 = I \in Span(A_0, A_1, \ldots, A_d)$. By the identity $AA_i = c_{i+1}A_{i+1} + a_iA_i + b_{i-1}A_{i-1}$, the $Span(A_0, A_1, \ldots, A_d)$ is closed under multiplication by A. Therefore $\mathcal{A} \subseteq Span(A_0, A_1, \ldots, A_d)$. But since the dimension of $\mathcal{A} \geq d+1$; the equality holds. (2) and (3) follow from this. □

Definition 3.4.19. Let $\Gamma = (X, R)$ be a DRG with diameter d. We define the polynomials $f_0, f_1, \ldots, f_d \in \mathbb{C}[X]$, by $f_0 = 1, f_1 = \lambda$, and $\lambda f_i = c_{i+1}f_{i+1} + a_if_i + b_{i-1}f_{i-1} (0 \leq i \leq d)$, where for convenience, we write $c_{d+1} = 1$.

Lemma 3.4.20. With the above polynomials $f_0, f_1, \ldots, f_{d+1}$,

1. $deg f_i = i (0 \leq i \leq d+1)$

2. The coefficient of λ^i in f_i is $(c_1c_2\cdots c_i)^{-1}$.

3. $f_i(A) = A_i (0 \leq i \leq d)$. In particular, $f_{d+1}(A) = 0$.

4. The distinct eigenvalues of Γ are precisely the zeros of f_{d+1}.

Proof. For (1), (2), and (3), refer to Section 3.1 of [1]. (4) Let $m \in \mathbb{C}[\lambda]$ denote the polynomial of A, *i.e.* the monic nonzero polynomial of least degree such that $m(A) = 0$. Since A is diagonalizable, the distinct eigenvalues of A are the zeros of m. Since $f_{d+1}(A) = 0$, $f_{d+1}(A)$ can be divided by m and thus f_{d+1} is a scalar multiple of m. Since $deg(m) = k+1$, the proof follows. □

Let $\Gamma(X,R)$ be DRG with diameter d, and let $A_0, A_1, \ldots, A_d$ be distance matrices of Γ. Then $\langle A_0, A_1, \ldots, A_d \rangle_{\mathbb{C}} = \mathcal{A}(\Gamma)$ and there exist $p_{i,j}^h$ such that $A_i A_j = A_j A_i = \sum_{h=0}^{d} p_{i,j}^h A_j$

Lemma 3.4.21. If $\Gamma = (X,R)$ is a DRG with diameter d, then

1. $k_h p_{i,j}^h = k_i p_{h,j}^i = k_j p_{i,h}^j (0 \leq h,i,j \leq d)$ where $k_i = |z \in x : \partial(x,z) = i|$ for some $x \in X$ and $k_0 = 1$, $k_1 (= a_i + b_i + c_i) = k$, $k_i \neq 0$ $(0 \leq i \leq d)$ for some $x \in X$.

2. $k_{i-1} b_{i-1} = k_i c_i$ $(1 \leq i \leq d)$.

3. $k_i = \frac{b_0 b_1 \ldots b_{i-1}}{c_1 c_2 \ldots c_i}$

Proof. (1) Count $|\{(x,y,z) \in X^3 : \partial(x,y) = h, \partial(x,z) = i, \partial(y,z) = j\}|$ and obtain

$|X| k_h p_{i,j}^h = |X| k_i p_{h,j}^i = |X| k_j p_{i,h}^j$

(2) $k_{i-1} b_{i-1} = k_{i-1} p_{1,j}^{i-1} = k_h p_{j,k}^i = k_i p_{1,i-1}^i = k_i c_i$ (consider $h = i-1, j = 1$)

(3) $k_i = \frac{b_{i-1}}{c_i} k_{i-1} = \frac{b_{i-1} b_{i-2}}{c_i c_{i-1}} k_{i-2} = \ldots = \frac{b_{i-1} b_{i-2} \ldots b_0}{c_i c_{i-1} \ldots c_1}$. □

CHAPTER 4. ASSOCIATION SCHEMES AND APN FUNCTIONS

Given an APN function f on $\mathbb{F}_{2^n}$, we can obtain an incidence structure $S = S(n; f) = (\mathcal{P}, \mathcal{B}, \mathcal{I})$. S is connected if n is odd or if $n \geq 4$ is even and f is a Gold function defined by $f(x) = x^{2^\alpha+1}$ where $(\alpha, n) = 1$ (Thm. 2.2.6). Suppose $S(n; f) = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ is the semi-biplane with parameters $(2^{2n}, 2^n)$. Then, defining binary relations on $\mathcal{P}$ by

$((x, a), (y, b)) \in R_1$ *iff* (x, a) and (y, b) are incident with two common blocks; and

$((x, a), (y, b)) \in R_2$ *iff* (x, a) and (y, b) are not incident with any common blocks,

we have that $\mathcal{X} = \mathcal{X}(n; f) = (\mathcal{P}, \{R_0, R_1, R_2\})$ is a two-class association scheme [11].

4.1 The association schemes $\mathcal{X}(n; f)$

In this section, we first describe the association scheme $\mathcal{X}(n; f)$ with $f(x) = x^{2^\alpha+1}$ and n odd, and then we construct two three-class association schemes which are fission schemes of $\mathcal{X}(n; f)$.

Theorem 4.1.1. Let $f(x) = x^{2^\alpha+1}$ be an APN function over $\mathbb{F}_q$ where $q = 2^n$ with n odd. Then the relations R_1 and R_2 define a two-class association scheme $\mathcal{X}$ on the points $\mathcal{P}$. The parameters and eigenmatrices of $\mathcal{X}$ are

$$B_1 = \begin{bmatrix} p_{10}^0 & p_{10}^1 & p_{10}^2 \\ p_{11}^0 & p_{11}^1 & p_{11}^2 \\ p_{12}^0 & p_{12}^1 & p_{12}^2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ q(q-1)/2 & q(q-2)/4 & q(q-2)/4 \\ 0 & (q^2-4)/4 & q^2/4 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} p_{20}^0 & p_{20}^1 & p_{20}^2 \\ p_{21}^0 & p_{21}^1 & p_{21}^2 \\ p_{22}^0 & p_{22}^1 & p_{22}^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & (q^2-4)/4 & q^2/4 \\ (q+2)(q-1)/2 & q(q+2)/4 & (q+4)(q-2)/4 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & q(q-1)/2 & (q+2)(q-1)/2 \\ 1 & q/2 & -(q+2)/2 \\ 1 & -q/2 & (q-2)/2 \end{bmatrix}.$$

Proof. Since n is odd, $(2^\alpha+1, 2^n-1) = 1$ and the Gold function $f(x) = x^{2^\alpha+1}$ is a permutation over $\mathbb{F}_q$, the relations R_1 and R_2 defined above are mutually disjoint and, along with the diagonal relation $R_0 = \{((x,a),(x,a)) : (x,a) \in \mathcal{P}\}$, form a partition of $\mathcal{P}$. It is also clear that $R_1^T = R_1$ and $R_2^T = R_2$ so the relations are symmetric.

In 2.2.2, we have already seen that $k_1 = p_{11}^0 = q(q-1)/2$ is constant. In order to calculate the intersection numbers for a two class association scheme, it is enough to compute $\lambda = p_{11}^1$ and $\mu = p_{11}^2$.

Without loss of generality, consider the point $(0,0)$ and the line $[0,0]$ which passes through the point $(0,0)$. Let (x,a) be an arbitrary point which is different from $(0,0)$. Then, the set of lines through the point (x,a) is given by $\{[y,b] : y \in \mathbb{F}_q,\ b = f(x+y)+a\}$. The line $[0,e]$, where $e = f(x)+a$, is in this set, and is parallel to the line $[0,0]$, so we assume $y \in \mathbb{F}_q^*$. Any intersection point (z,c) of the lines $[0,0]$ and $[y,\ f(x+y)+a]$ must satisfy

$$\begin{cases} c &= f(z) = z^{2^\alpha+1} \\ c &= f(y+z)+f(x+y)+a \end{cases}.$$

Solving this system, we obtain

$$\begin{aligned} 0 &= f(y+z) - f(z) + f(x+y) + a \\ &= y(z+x)^{2^\alpha} + y^{2^\alpha}(z+x) + f(x) + a \\ &= yw^{2^\alpha} + y^{2^\alpha}w + e \end{aligned}$$

where $w = z + x$ and $e = f(x) + a$, which is a constant.

Since we are counting the common intersection points of $(0,0)$ and (x,a), we may assume that $w = z + x \neq 0$ and $w \neq x$. If $e = 0$, then $a = f(x)$ which implies that (x,a) lies on $[0,0]$. Thus the line $[0,0]$ intersects every line through (x,a) and will define $q-2$ points of intersection in addition to $(0,0)$ and (x,a).

Now suppose $e \neq 0$. We want to count the number of y's for which

$$yw^{2^\alpha} + y^{2^\alpha} w = e$$

is solvable in w. This equation can be converted into

$$f(w+y) - f(w) = e + f(y)$$

which is equivalent to

$$f(1 + w/y) - f(w/y) = e/f(y) + 1.$$

Since f is a permutation with $f(0) = 0$, we obtain distinct values for $e/f(y)$ as y varies on $\mathbb{F}_q^*$. So we can reduce our problem to that of determining the number of β's, $\beta \neq 1$, for which the equation

$$f(1 + w/y) - f(w/y) = \beta$$

is solvable in w and how many distinct solutions there are in these cases. For $\beta \in \mathbb{F}_q$ there are $q/2$ choices of β for which this equation has two solutions and $q/2$ for which it has none. For the case when $\beta = 1$, $w/y = 0, 1$ satisfy the equation. So for $\beta \in \mathbb{F}_q - \{1\}$ there are $(q-2)/2$ choices for which the equation will have the solution. When $(0,0)$ and (x,a) are first associates $w = x$ will be a solution. In this case we must remove the β for which $w = x$ is a solution (this will not be $\beta = 1$). This leaves $(q-4)/2$ possible choices for β. For each of the possible choices of β there will be two solutions; i.e., two lines that are intersected by $[0,0]$ which gives a total of $q-4$ intersection points if $((0,0),(x,a)) \in R_1$, and $q-2$ otherwise. If $((0,0),(x,a)) \in R_1$, then there will be two lines through (x,a) intersecting $[0,0]$ at $(0,0)$. Each

will intersect $[0,0]$ at a second distinct point. In the previous argument, these two points have been removed, so we now add them back in. So, overall any line through $(0,0)$ defines $q-2$ intersection points with the line through (x,a) if $(0,0)$ and (x,a) are in the first relation and no intersection points if $(0,0)$ and (x,a) are not in the first relation.

Now we need to determine the number of times we count each intersection point. For an arbitrary chosen point (x,a), each of the q lines through $(0,0)$ defines $q-2$ intersection points with lines through (x,a), giving a count of $q(q-2)$ points. Since every pair of collinear points has exactly two lines which pass through both, each of these points would have been counted four times giving $q(q-2)/4$ common first associates of $(0,0)$ and (x,a) regardless of the relation between $(0,0)$ and (x,a). This gives $p_{11}^1 = p_{11}^2 = q(q-2)/4$. All of the rest of the parameters and the character table follow from the conversion formulas given in Section 3.4. This completes the proof. □

We now show that the association scheme $\mathcal{X}(n;f)$ has two symmetric fission schemes of class three. These fission schemes are obtained when we split the first relation R_1 of $\mathcal{X}(n;f)$ into two classes.

Theorem 4.1.2. Let $f(x) = x^{2^\alpha+1}$, n be odd, and $S(n;f) = (\mathcal{P},\mathcal{B},\mathcal{I})$ be the semi-biplane with parameters $(2^{2n},2^n)$ as before. Then, defining binary relations on $\mathcal{P}$ by

$$\begin{aligned}
((x,a),(y,b)) \in \tilde{R}_1 &\Leftrightarrow f(x+y) = a+b \quad ((x,a) \neq (y,b));\\
((x,a),(y,b)) \in \tilde{R}_2 &\Leftrightarrow f(x+y) \neq a+b, \text{ but}\\
&\qquad (x,a) \text{ and } (y,b) \text{ are incident with two common blocks;}\\
((x,a),(y,b)) \in \tilde{R}_3 &\Leftrightarrow (x,a) \text{ and } (y,b) \text{ are not incident with any common blocks}
\end{aligned}$$

we obtain a three class association scheme $\tilde{\mathcal{X}} = \tilde{\mathcal{X}}(n;f) = (\mathcal{P},\{\tilde{R}_0,\tilde{R}_1,\tilde{R}_2,\tilde{R}_3\})$ having the following parameters and character table.

$$\tilde{B}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ q-1 & 0 & 2 & 0 \\ 0 & q-2 & (q-8)/2 & (q-2)/2 \\ 0 & 0 & (q+2)/2 & q/2 \end{bmatrix}$$

$$\tilde{B}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & q-2 & (q-8)/2 & (q-2)/2 \\ (q-2)(q-1)/2 & (q-2)(q-8)/4 & (q^2-6q+24)/4 & (q-2)(q-4)/4 \\ 0 & (q^2-4)/4 & (q+2)(q-4)/4 & q(q-2)/4 \end{bmatrix}$$

$$\tilde{B}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & (q+2)/2 & q/2 \\ 0 & (q^2-4)/4 & (q+2)(q-4)/4 & q(q-2)/4 \\ (q+2)(q-1)/2 & q(q+2)/4 & q(q+2)/4 & (q+4)(q-2)/4 \end{bmatrix}$$

$$\tilde{P} = \begin{bmatrix} 1 & q-1 & (q/2-1)(q-1) & (q+2)(q-1)/2 \\ 1 & -1-\sqrt{2q} & q/2+1+\sqrt{2q} & -(q+2)/2 \\ 1 & -1+\sqrt{2q} & -q/2+1-\sqrt{2q} & -(q+2)/2 \\ 1 & -1 & -q/2+1 & (q-2)/2 \end{bmatrix}.$$

Proof. It is clear that if two distinct points (x, a) and (y, b) satisfy the equation $f(x+y) = a+b$, then these points are coincident with two lines $[x, a]$ and $[y, b]$. From the incidence counts given in Prop. 2.2.2, for a given point (x, a), the number of points (y, b) for which $f(x+y) = a+b$ is $q-1$, so $k_1 = p_{11}^0 = q-1$. The rest of the parameters are determined in a manner similar to that in the previous theorem. We omit the details. □

Remark 4.1.3. (1) From the definition of the relations for the association schemes $\mathcal{X}$ and $\tilde{\mathcal{X}}$, it is clear that $\tilde{R}_1 \cup \tilde{R}_2 = R_1$ and $\tilde{R}_3 = R_2$; that is, $\mathcal{X}$ is a fusion scheme of $\tilde{\mathcal{X}}$.

(2) The first intersection matrix of $\tilde{\mathcal{X}}(n; f) = (\mathcal{P}, \{\tilde{R}_0, \tilde{R}_1, \tilde{R}_2, \tilde{R}_3\})$ is a tridiagonal matrix.

This means that the first relation graph is a distance-regular graph and the association scheme is a P-polynomial association scheme. This first relation graph is a cospectral graph of the distance-regular graph corresponding to the coset graph of the uniformly packed Kasami code which was introduced by Delsarte in Chapter 11 of [6]. An alternate description of this coset graph is also found in [12] and [36].

Theorem 4.1.4. The association scheme $\mathcal{X}(n;f) = (\mathcal{P}, \{R_0, R_1, R_2\})$ where $f(x) = x^{2^a+1}$ and n is odd, also has the three-class fission scheme $\breve{\mathcal{X}}(n;f) = (\mathcal{P}, \{\breve{R}_0, \breve{R}_1, \breve{R}_2, \breve{R}_3\})$ with the following parameters and character table

$$\breve{B}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ (q-2)(q-1)/2 & (q^2-6q+24)/4 & (q-2)(q-4)/4 & (q-2)(q-8)/4 \\ 0 & (q+2)(q-4)/4 & q(q-2)/4 & (q^2-4)/4 \\ 0 & (q-8)/2 & (q-2)/2 & q-2 \end{bmatrix}$$

$$\breve{B}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & (q+2)(q-4)/4 & q(q-2)/4 & (q^2-4)/4 \\ (q+2)(q-1)/2 & q(q+2)/4 & (q+4)(q-2)/4 & q(q+2)/4 \\ 0 & (q+2)/2 & q/2 & 0 \end{bmatrix}$$

$$\breve{B}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & (q-8)/2 & (q-2)/2 & q-2 \\ 0 & (q+2)/2 & q/2 & 0 \\ q-1 & 2 & 0 & 0 \end{bmatrix}$$

$$\breve{P} = \begin{bmatrix} 1 & (q/2-1)(q-1) & (q+2)(q-1)/2 & q-1 \\ 1 & q/2+1-\sqrt{2q} & -(q+2)/2 & -1+\sqrt{2q} \\ 1 & -q/2+1 & (q-2)/2 & -1 \\ 1 & q/2+1+\sqrt{2q} & -(q+2)/2 & -1-\sqrt{2q} \end{bmatrix}.$$

Proof. This fission scheme of $\mathcal{X}$ can be obtained by reordering the relations of $\tilde{\mathcal{X}}$. □

Remark 4.1.5. While the fission scheme $\tilde{\mathcal{X}}(n;f)$ is a P-polynomial for every odd $n \geq 3$, the fission scheme $\breve{\mathcal{X}}(n;f)$, in general, is not a P-polynomial since the first intersection matrix of $\breve{\mathcal{X}}(n;f) = (\mathcal{P}, \{\breve{R}_0, \breve{R}_1, \breve{R}_2, \breve{R}_3\})$ is not tridiagonal. The first intersection matrix only becomes tridiagonal when $q = 8$. In this case, the first relation graph is a distance-regular graph. We will treat the association scheme for $q = 8$ as a special case in the following section.

4.2 Characterization of $\tilde{\mathcal{X}}(3;x^3)$ and $\breve{\mathcal{X}}(3;x^3)$

The intersection matrices and the character table of the association scheme $\mathcal{X}(3;x^3)$ are given by

$$B_1 = \begin{bmatrix} 0 & 1 & 0 \\ 28 & 12 & 12 \\ 0 & 15 & 16 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 15 & 16 \\ 35 & 20 & 18 \end{bmatrix}, \quad P = \begin{bmatrix} 1 & 28 & 35 \\ 1 & 4 & -5 \\ 1 & -4 & 3 \end{bmatrix}.$$

This association scheme has two three-class symmetric fission schemes by 4.1.2 and 4.1.4 which have the following parameters and character tables.

4.2.1 The scheme $\tilde{\mathcal{X}}(3;x^3)$

$$B_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 7 & 0 & 2 & 0 \\ 0 & 6 & 0 & 3 \\ 0 & 0 & 5 & 4 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 6 & 0 & 3 \\ 21 & 0 & 10 & 6 \\ 0 & 15 & 10 & 12 \end{bmatrix},$$

$$B_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 4 \\ 0 & 15 & 10 & 12 \\ 35 & 20 & 20 & 18 \end{bmatrix}, \quad P = \begin{bmatrix} 1 & 7 & 21 & 35 \\ 1 & -5 & 9 & -5 \\ 1 & 3 & 1 & -5 \\ 1 & -1 & -3 & 3 \end{bmatrix}.$$

This association scheme is isomorphic to the P-polynomial association scheme that is obtained from the folded 7-cube [6, p.426]. The folded 7-cube has the spectrum

$$\begin{pmatrix} 7 & 3 & -1 & -5 \\ 1 & 21 & 35 & 7 \end{pmatrix}.$$

4.2.2 The scheme $\breve{\mathcal{X}}(3; x^3)$

$$B_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 21 & 10 & 6 & 0 \\ 0 & 10 & 12 & 15 \\ 0 & 0 & 3 & 6 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 10 & 12 & 15 \\ 35 & 20 & 18 & 20 \\ 0 & 5 & 4 & 0 \end{bmatrix},$$

$$B_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 3 & 6 \\ 0 & 5 & 4 & 0 \\ 7 & 2 & 0 & 0 \end{bmatrix}, \quad P = \begin{bmatrix} 1 & 21 & 35 & 7 \\ 1 & 1 & -5 & 3 \\ 1 & -3 & 3 & -1 \\ 1 & 9 & -5 & -5 \end{bmatrix}.$$

This association scheme is isomorphic to the P-polynomial association scheme that is obtained from the halved 7-cube [6, p.427]. The halved 7-cube has the spectrum

$$\begin{pmatrix} 21 & 9 & 1 & -3 \\ 1 & 7 & 21 & 35 \end{pmatrix}.$$

4.3 Characterization of the relation graphs of $\mathcal{X}(n; f)$

We now return to the original two class association schemes $\mathcal{X}(n; f)$ and study the structure of some of their relation graphs. Both the first and second relation graphs of $\mathcal{X}(n; f)$ are strongly regular graphs, and the first relation graph $\Gamma = (\mathcal{P}, R_1)$ is the complement of the second relation graph. The parameters of these strongly regular graphs are

$$\Gamma = (\mathcal{P}, R_1) = \mathrm{SRG}(2^{2n},\ 2^{n-1}(2^n - 1),\ 2^{n-1}(2^{n-1} - 1),\ 2^{n-1}(2^{n-1} - 1)), \text{ and}$$

$$\Gamma^C = (\mathcal{P}, R_2) = \mathrm{SRG}(2^{2n},\ (2^{n-1}+1)(2^n-1),\ (2^{n-2}+1)(2^{n-1}-1),\ 2^{n-1}(2^{n-1}+1)),$$

respectively. This yields an infinite class of strongly regular graphs with such parameters.

For $q = 2^n$ with an odd integer $n \geq 3$ and the Gold function $f(x) = x^3$ (with $\alpha = 1$), we have the following structure theorem for the first relation graph

$$\Gamma = \mathrm{SRG}(q^2,\ q(q-1)/2,\ q(q-2)/4,\ q(q-2)/4) \text{ of } \mathcal{X}(n; x^3).$$

Theorem 4.3.1. Let Γ be the first relation graph of $\mathcal{X}(n; x^3)$ with n odd and let u_0 be a vertex of Γ. Then,

1. For any vertex $w_0 \in \Gamma_1(u_0)$, the vertices u_0 and w_0 appear together in exactly two copies of K_q, and the set $(\Gamma_1(u_0) \cap \Gamma_1(w_0)) \cup \{u_0, w_0\}$ of $2q-2$ vertices forms the configuration in Figure 4.1.

2. The induced subgraph H consisting of the vertex u_0 and its neighborhood $\Gamma_1(u_0)$ contains q induced subgraphs that are isomorphic to K_q.

3. Any adjacent pair of vertices lie together in two cliques of order q, and $(q-2)^2$ tetrahedra.

Proof. (1) The pair of collinear vertices u_0 and w_0 is incident with exactly two lines, and each of these lines makes a clique of size q in the induced subgraph H on the vertex set $\Gamma(u_0) \cup \{u_0\}$. Without loss of generality, let $u_0 = (0,0)$ and $w_0 = (1,1)$. Then the two lines that are incident with u_0 and w_0 will be $[0,0]$ and $[1,1]$. It is clear that the vertices incident with the line $[0,0]$ are precisely (x, x^3) for $x \in \mathbb{F}_q$. Also, the points incident with the line $[1,1]$ are those of the form $(x, 1+(1+x)^3)$ for $x \in \mathbb{F}_q$. In order to show that all $2q-2$ vertices lying on these two lines $[0,0]$ and $[1,1]$ together form the configuration depicted in Figure 1, it suffices to show that

(i) For each pair of vertices on $[0,0]$, the vertices in that pair are first associates; i.e., they are co-incident with another line.

(ii) For each pair of vertices on $[1,1]$, the vertices in that pair are first associates.

(iii) Every vertex (x, x^3) on $[0,0]$ except for u_0 and w_0, is a first associate of exactly one vertex on the line $[1,1]$ (from the set $\{(y, 1+f(1+y)) : y \in \mathbb{F}_q^*\} - \{(0,0),(1,1)\}$).

(i) and (ii) follow from the fact that the vertices (x, x^3) and (y, y^3) are both incident with the line $[x+y, (x+y)^3]$ as well as $[0,0]$ or $[1,1]$. For (iii), it is straightforward to verify that the vertex (x, x^3) on the line $[0,0]$ is a first associate of the vertex $(1+x, 1+x^3)$ on the line $[1,1]$ since both vertices are collinear with the line $[x, x^3]$. This completes the proof of (1).

(2) Since there are exactly q lines passing through the vertex u_0, each of the lines is incident with q vertices including u_0. The q vertices incident with each of the lines form a clique K_q, so the proof of (2) follows.

(3) The first half of the statement is clear from (2). The second part of (3) follows from the fact that every time we choose four vertices including u_0 and w_0 from a clique, we have a tetrahedron. So, from each clique, we have a total of $(q-2)(q-3)$ tetrahedra which contain u_0 and w_0. There are also $q-2$ tetrahedra of type $\{u_0 = (0,0), w_0 = (1,1), (x, x^3), (1+x, 1+x^3)\}$. So, for any pair of adjacent vertices in Γ, there are $(q-2)^2$ tetrahedra containing the pair. □

This local structure characterizes the strongly regular graphs $\Gamma(n; x^3)$ for small n. In particular, we will see that $\Gamma(3; x^3) = \mathrm{SRG}(64, 28, 12, 12)$ is isomorphic to the halved-folded 8-cube.

4.3.1 The structure of the subconstituent of $\Gamma(3; x^3)$

The first relation graph Γ of $\mathcal{X}(3; x^3)$ is the $\mathrm{SRG}(64, 28, 12, 12)$ and it has the following structural properties.

Corollary 4.3.2. Let Γ be the first relation graph of $\mathcal{X}(3; x^3)$. Let u_0 be a vertex of Γ.

1. For any vertex $w_0 \in \Gamma_1(u_0)$, the vertices u_0 and w_0 appear together in exactly two copies of K_8, and the set $(\Gamma_1(u_0) \cap \Gamma_1(w_0)) \cup \{u_0, w_0\}$ of fourteen vertices forms the configuration in Figure 4.2.

2. The induced subgraph H consisting of the vertex u_0 and its neighborhood $\Gamma_1(u_0)$ contains eight induced subgraphs that are isomorphic to K_8.

3. For every pair of adjacent vertices, the vertices lie together in two cliques of order 8, and 36 tetrahedra.

Proof. This is a particular case of 4.3.1 with $n = 3$. So we omit the proof. □

Theorem 4.3.3. The relation graph $\Gamma(3; x^3) = \mathrm{SRG}(64, 28, 12, 12)$ is isomorphic to the halved-folded 8-cube.

Proof. The properties described in Cor. 4.3.2 are equivalent to the requirements of the characterization of the halved-folded 8-cube described in [27]. □

4.3.2 The structure of the subconstituent of $\Gamma(5; x^3)$

We now consider $\mathcal{X} = \mathcal{X}(5; x^3)$, another two class association scheme with the following parameters and character table

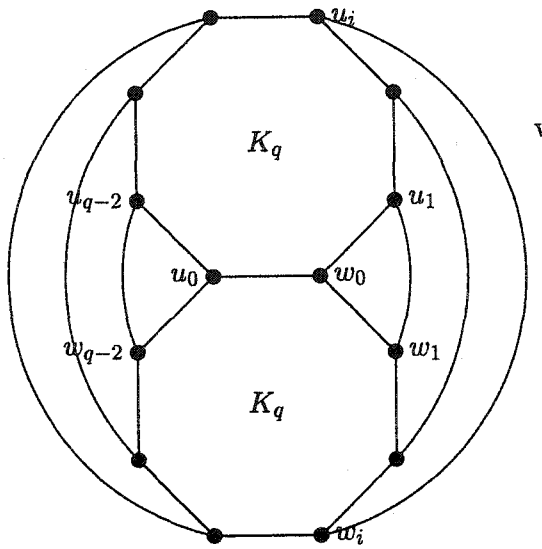
$$B_1 = \begin{bmatrix} 0 & 1 & 0 \\ 496 & 240 & 240 \\ 0 & 255 & 256 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 255 & 256 \\ 527 & 272 & 270 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 496 & 527 \\ 1 & 16 & -17 \\ 1 & -16 & 15 \end{bmatrix}.$$

The first and the second relation graphs of $\mathcal{X}(5; x^3)$ are the strongly regular graphs

$$\Gamma_1 = \mathrm{SRG}(1024, 496, 240, 240) \text{ and } \Gamma_2 = \mathrm{SRG}(1024, 527, 270, 272).$$

Remark 4.3.4. The above graph $\mathrm{SRG}(1024, 496, 240, 240)$ is the first known strongly regular graph with the given parameters.



where $u_0 = (0,0)$

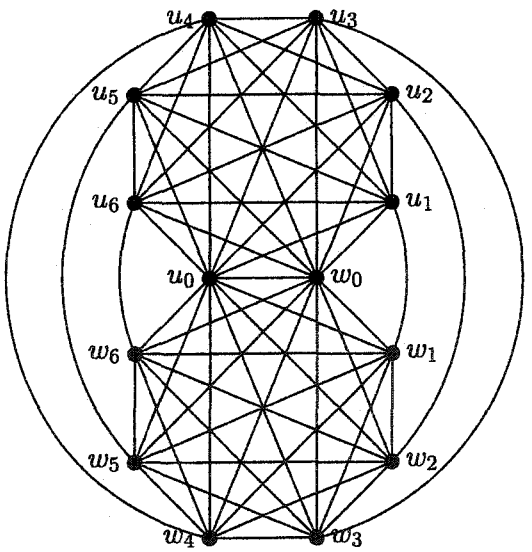
$w_0 = (1,1)$

$u_i = (x_i, x_i^3),\ i = 1,2,\ldots,q-2$

$w_i = (1+x_i, 1+x_i^3),\ i = 1,2,\ldots,q-2$

$x_i \in \mathbb{F}_q$

Figure 4.1 The configuration of $(\Gamma_1(u_0) \bigcap \Gamma_1(w_0)) \bigcup \{u_0, w_0\}$ in $\Gamma(n; x^3)$.



where $u_0 = (0,0)$

$w_0 = (1,1)$

$u_i = (x_i, x_i^3),\ i = 1,2,\ldots,6$

$w_i = (1+x_i, 1+x_i^3),\ i = 1,2,\ldots,6$

$x_i \in \mathbb{F}_8$

Figure 4.2 The configuration of $(\Gamma_1(u_0) \bigcap \Gamma_1(w_0)) \bigcup \{u_0, w_0\}$ in $\Gamma(3; x^3)$.

APPENDIX A. MAXIMUM NUMBER OF SOLUTIONS TABLES

Table A.1 Maximum number of solutions to $f(x+a)+f(x)=b$ for power functions, x^d, over the field $\mathbb{F}_{2^n}$, for $n=2,\ldots,6$

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^2}$	3	2	3	Gold
	1	4	(1 2)	
$\mathbb{F}_{2^3}$	3	2	(3 6 5)	Gold, Welch, Niho, Field Inverse
	1	8	(1 2 4)	
$\mathbb{F}_{2^4}$	3	2	(3 6 12 9)	Gold
	5	4	(5 10)	
	7	4	(7 14 13 11)	
	1	16	(1 2 4 8)	
$\mathbb{F}_{2^5}$	3	2	(3 6 12 24 17)	Gold
	5	2	(5 10 20 9 18)	Gold, Niho
	7	2	(7 14 28 25 19)	Welch
	11	2	(11 22 13 26 21)	Kasami
	15	2	(15 30 29 27 23)	Dobbertin, Field Inverse
	1	32	(1 2 4 8 16)	
$\mathbb{F}_{2^6}$	3	2	(3 6 12 24 48 33)	Gold
	5	4	(5 10 20 40 17 34)	
	13	4	(13 26 52 41 19 38)	
	31	4	(31 62 61 59 55 47)	
	7	6	(7 14 28 56 49 35)	
	9	8	(9 18 36)	
	15	8	(15 30 60 57 51 39)	
	11	10	(11 22 44 25 50 37)	
	23	10	(23 46 29 58 53 43)	
	27	12	(27 54 45)	
	21	20	(21 42)	
	1	64	(1 2 4 8 16 32)	

Table A.2 Maximum number of solutions to $f(x+a)+f(x)=b$ for power functions, x^d, over the field $\mathbb{F}_{2^7}$

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^7}$	3	2	(3 6 12 24 48 96 65)	Gold
	5	2	(5 10 20 40 80 33 66)	Gold
	9	2	(9 18 36 72 17 34 68)	Gold
	11	2	(11 22 44 88 49 98 69)	Welch
	13	2	(13 26 52 104 81 35 70)	Kasami
	15	2	(15 30 60 120 113 99 71)	Gold
	23	2	(23 46 92 57 114 101 75)	Kasami
	27	2	(27 54 108 89 51 102 77)	Gold
	29	2	(29 58 116 105 83 39 78)	Niho
	43	2	(43 86 45 90 53 106 85)	Gold
	63	2	(63 126 125 123 119 111 95)	Field Inverse
	19	4	(19 38 76 25 50 100 73)	
	47	4	(47 94 61 122 117 107 87)	
	7	6	(7 14 28 56 67 97 112)	
	21	6	(21 42 84 41 82 37 74)	
	31	6	(31 62 124 121 115 103 79)	
	55	6	(55 110 93 59 118 109 91)	
	1	128	(1 2 4 8 16 32 64)	

Table A.3 Maximum number of solutions to $f(x+a)+f(x)=b$ for power functions, x^d, over the field $\mathbb{F}_{2^8}$

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^8}$	3	2	(3 6 12 24 48 96 192 129)	Gold
	9	2	(9 18 36 72 144 33 66 132)	Gold
	39	2	(39 78 156 57 114 228 201 147)	Kasami
	5	4	(5 10 20 40 80 160 65 130)	
	21	4	(21 42 84 168 81 162 69 138)	
	95	4	(95 190 125 250 245 235 215 175)	
	111	4	(111 222 189 123 246 237 219 183)	
	127	4	(127 254 253 251 247 239 223 191)	
	7	6	(7 14 28 56 112 224 193 131)	
	25	6	(25 50 100 200 145 35 70 140)	
	37	6	(37 74 148 41 82 164 73 146)	
	63	6	(63 126 252 249 243 231 207 159)	
	11	10	(11 22 44 88 176 97 194 133)	
	29	10	(29 58 116 232 209 163 71 142)	
	13	12	(13 26 52 104 208 161 67 134)	
	55	12	(55 110 220 185 115 230 205 155)	
	59	12	(59 118 236 217 179 103 206 157)	
	15	14	(15 30 60 120 240 225 195 135)	
	45	14	(45 90 180 105 210 165 75 150)	
	17	16	(17 34 68 136)	
	19	16	(19 38 76 152 49 98 196 137)	
	23	16	(23 46 92 184 113 226 197 139)	
	31	16	(31 62 124 248 241 227 199 143)	
	47	16	(47 94 188 121 242 229 203 151)	
	53	16	(53 106 212 169 83 166 77 154)	
	61	16	(61 122 244 233 211 167 79 158)	
	91	16	(91 182 109 218 181 107 214 173)	
	119	22	(119 238 221 187)	
	27	26	(27 54 108 216 177 99 198 141)	
	43	30	(43 86 172 89 178 101 202 149)	
	87	30	(87 174 93 186 117 234 213 171)	
	51	50	(51 102 204 153)	
	85	84	(85 170)	
	1	256	(1 2 4 8 16 32 64 128)	

Table A.4 Maximum number of solutions to $f(x+a)+f(x)=b$ for power functions, x^d, over the field $\mathbb{F}_{2^9}$

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^9}$	3	2	(3 6 12 24 48 96 192 384 257)	Gold
	5	2	(5 10 20 40 80 160 320 129 258)	Gold
	13	2	(13 26 52 104 208 416 321 131 262)	Kasami
	17	2	(17 34 68 136 272 33 66 132 264)	Gold
	19	2	(19 38 76 152 304 97 194 388 265)	Niho, Welch
	27	2	(27 54 108 216 432 353 195 390 269)	Niho, Welch
	31	2	(31 62 124 248 496 481 451 391 271)	Gold
	47	2	(47 94 188 376 241 482 453 395 279)	Kasami
	59	2	(59 118 236 472 433 355 199 398 285)	Kasami
	87	2	(87 174 348 185 370 229 458 405 299)	Kasami
	103	2	(103 206 412 313 115 230 460 409 307)	Gold
	171	2	(171 342 173 346 181 362 213 426 341)	Gold
	255	2	(255 510 509 507 503 495 479 447 383)	Field Inverse
	45	4	(45 90 180 360 209 418 325 139 278)	
	125	4	(125 250 500 489 467 423 335 159 318)	
	7	6	(7 14 28 56 112 224 448 385 259)	
	21	6	(21 42 84 168 336 161 322 133 266)	
	35	6	(35 70 140 280 49 98 196 392 273)	
	61	6	(61 122 244 488 465 419 327 143 286)	
	63	6	(63 126 252 504 497 483 455 399 287)	
	83	6	(83 166 332 153 306 101 202 404 297)	
	91	6	(91 182 364 217 434 357 203 406 301)	
	111	6	(111 222 444 377 243 486 461 411 311)	
	117	6	(117 234 468 425 339 167 334 157 314)	
	119	6	(119 238 476 441 371 231 462 413 315)	
	175	6	(175 350 189 378 245 490 469 427 343)	
	9	8	(9 18 36 72 144 288 65 130 260)	
	11	8	(11 22 44 88 176 352 193 386 261)	
	15	8	(15 30 60 120 240 480 449 387 263)	
	23	8	(23 46 92 184 368 225 450 389 267)	
	25	8	(25 50 100 200 400 289 67 134 268)	
	29	8	(29 58 116 232 464 417 323 135 270)	
	39	8	(39 78 156 312 113 226 452 393 275)	
	41	8	(41 82 164 328 145 290 69 138 276)	
	43	8	(43 86 172 344 177 354 197 394 277)	
	51	8	(51 102 204 408 305 99 198 396 281)	
	53	8	(53 106 212 424 337 163 326 141 282)	

Table A.4 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^9}$	57	8	(57 114 228 456 401 291 71 142 284)	
	79	8	(79 158 316 121 242 484 457 403 295)	
	85	8	(85 170 340 169 338 165 330 149 298)	
	93	8	(93 186 372 233 466 421 331 151 302)	
	95	8	(95 190 380 249 498 485 459 407 303)	
	107	8	(107 214 428 345 179 358 205 410 309)	
	109	8	(109 218 436 361 211 422 333 155 310)	
	123	8	(123 246 492 473 435 359 207 414 317)	
	127	8	(127 254 508 505 499 487 463 415 319)	
	187	8	(187 374 237 474 437 363 215 430 349)	
	191	8	(191 382 253 506 501 491 471 431 351)	
	239	8	(239 478 445 379 247 494 477 443 375)	
	55	20	(55 110 220 440 369 227 454 397 283)	
	75	20	(75 150 300 89 178 356 201 402 293)	
	77	20	(77 154 308 105 210 420 329 147 294)	
	223	20	(223 446 381 251 502 493 475 439 367)	
	37	26	(37 74 148 296 81 162 324 137 274)	
	183	26	(183 366 221 442 373 235 470 429 347)	
	73	72	(73 146 292)	
	219	72	(219 438 365)	
	1	512	(1 2 4 8 16 32 64 128 256)	

Table A.5 Maximum number of solutions to $f(x+a)+f(x)=b$ for power functions, x^d, over the field $\mathbb{F}_{2^{10}}$

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{10}}$	3	2	(3 6 12 24 48 96 192 384 768 513)	Gold
	9	2	(9 18 36 72 144 288 576 129 258 516)	Gold
	57	2	(57 114 228 456 912 801 579 135 270 540)	Kasami
	213	2	(213 426 852 681 339 678 333 666 309 618)	Dobbertin
	5	4	(5 10 20 40 80 160 320 640 257 514)	
	13	4	(13 26 52 104 208 416 832 641 259 518)	
	17	4	(17 34 68 136 272 544 65 130 260 520)	
	21	4	(21 42 84 168 336 672 321 642 261 522)	
	29	4	(29 58 116 232 464 928 833 643 263 526)	
	69	4	(69 138 276 552 81 162 324 648 273 546)	
	79	4	(79 158 316 632 241 482 964 905 787 551)	
	87	4	(87 174 348 696 369 738 453 906 789 555)	
	103	4	(103 206 412 824 625 227 454 908 793 563)	
	149	4	(149 298 596 169 338 676 329 658 293 586)	
	181	4	(181 362 724 425 850 677 331 662 301 602)	
	205	4	(205 410 820 617 211 422 844 665 307 614)	
	223	4	(223 446 892 761 499 998 973 923 823 623)	
	237	4	(237 474 948 873 723 423 846 669 315 630)	
	247	4	(247 494 988 953 883 743 463 926 829 635)	
	367	4	(367 734 445 890 757 491 982 941 859 695)	
	375	4	(375 750 477 954 885 747 471 942 861 699)	
	511	4	(511 1022 1021 1019 1015 1007 991 959 895 767)	
	7	6	(7 14 28 56 112 224 448 896 769 515)	
	15	6	(15 30 60 120 240 480 960 897 771 519)	
	19	6	(19 38 76 152 304 608 193 386 772 521)	
	23	6	(23 46 92 184 368 736 449 898 773 523)	
	27	6	(27 54 108 216 432 864 705 387 774 525)	
	43	6	(43 86 172 344 688 353 706 389 778 533)	
	45	6	(45 90 180 360 720 417 834 645 267 534)	
	59	6	(59 118 236 472 944 865 707 391 782 541)	
	61	6	(61 122 244 488 976 929 835 647 271 542)	
	71	6	(71 142 284 568 113 226 452 904 785 547)	
	73	6	(73 146 292 584 145 290 580 137 274 548)	
	75	6	(75 150 300 600 177 354 708 393 786 549)	
	89	6	(89 178 356 712 401 802 581 139 278 556)	
	91	6	(91 182 364 728 433 866 709 395 790 557)	
	111	6	(111 222 444 888 753 483 966 909 795 567)	
	115	6	(115 230 460 920 817 611 199 398 796 569)	

Table A.5 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{10}}$	117	6	(117 234 468 936 849 675 327 654 285 570)	
	119	6	(119 238 476 952 881 739 455 910 797 571)	
	127	6	(127 254 508 1016 1009 995 967 911 799 575)	
	147	6	(147 294 588 153 306 612 201 402 804 585)	
	151	6	(151 302 604 185 370 740 457 914 805 587)	
	167	6	(167 334 668 313 626 229 458 916 809 595)	
	173	6	(173 346 692 361 722 421 842 661 299 598)	
	175	6	(175 350 700 377 754 485 970 917 811 599)	
	189	6	(189 378 756 489 978 933 843 663 303 606)	
	191	6	(191 382 764 505 1010 997 971 919 815 607)	
	207	6	(207 414 828 633 243 486 972 921 819 615)	
	215	6	(215 430 860 697 371 742 461 922 821 619)	
	235	6	(235 470 940 857 691 359 718 413 826 629)	
	239	6	(239 478 956 889 755 487 974 925 827 631)	
	245	6	(245 490 980 937 851 679 335 670 317 634)	
	255	6	(255 510 1020 1017 1011 999 975 927 831 639)	
	347	6	(347 694 365 730 437 874 725 427 854 685)	
	379	6	(379 758 493 986 949 875 727 431 862 701)	
	439	6	(439 878 733 443 886 749 475 950 877 731)	
	479	6	(479 958 893 763 503 1006 989 955 887 751)	
	25	8	(25 50 100 200 400 800 577 131 262 524)	
	37	8	(37 74 148 296 592 161 322 644 265 530)	
	41	8	(41 82 164 328 656 289 578 133 266 532)	
	49	8	(49 98 196 392 784 545 67 134 268 536)	
	51	8	(51 102 204 408 816 609 195 390 780 537)	
	83	8	(83 166 332 664 305 610 197 394 788 553)	
	107	8	(107 214 428 856 689 355 710 397 794 565)	
	11	10	(11 22 44 88 176 352 704 385 770 517)	
	77	10	(77 154 308 616 209 418 836 649 275 550)	
	85	10	(85 170 340 680 337 674 325 650 277 554)	
	105	10	(105 210 420 840 657 291 582 141 282 564)	
	121	10	(121 242 484 968 913 803 583 143 286 572)	
	179	10	(179 358 716 409 818 613 203 406 812 601)	
	253	10	(253 506 1012 1001 979 935 847 671 319 638)	
	351	10	(351 702 381 762 501 1002 981 939 855 687)	
	383	10	(383 766 509 1018 1013 1003 983 943 863 703)	
	55	12	(55 110 220 440 880 737 451 902 781 539)	
	447	12	(447 894 765 507 1014 1005 987 951 879 735)	

Table A.5 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{10}}$	31	30	(31 62 124 248 496 992 961 899 775 527)	
	155	30	(155 310 620 217 434 868 713 403 806 589)	
	33	32	(33 66 132 264 528)	
	39	32	(39 78 156 312 624 225 450 900 777 531)	
	63	32	(63 126 252 504 1008 993 963 903 783 543)	
	123	32	(123 246 492 984 945 867 711 399 798 573)	
	159	32	(159 318 636 249 498 996 969 915 807 591)	
	183	32	(183 366 732 441 882 741 459 918 813 603)	
	219	32	(219 438 876 729 435 870 717 411 822 621)	
	35	34	(35 70 140 280 560 97 194 388 776 529)	
	47	34	(47 94 188 376 752 481 962 901 779 535)	
	53	34	(53 106 212 424 848 673 323 646 269 538)	
	95	34	(95 190 380 760 497 994 965 907 791 559)	
	101	34	(101 202 404 808 593 163 326 652 281 562)	
	109	34	(109 218 436 872 721 419 838 653 283 566)	
	125	34	(125 250 500 1000 977 931 839 655 287 574)	
	157	34	(157 314 628 233 466 932 841 659 295 590)	
	187	34	(187 374 748 473 946 869 715 407 814 605)	
	221	34	(221 442 884 745 467 934 845 667 311 622)	
	251	34	(251 502 1004 985 947 871 719 415 830 637)	
	231	42	(231 462 924 825 627)	
	363	42	(363 726 429 858 693)	
	495	42	(495 990 957 891 759)	
	99	62	(99 198 396 792 561)	
	165	62	(165 330 660 297 594)	
	93	92	(93 186 372 744 465 930 837 651 279 558)	
	171	122	(171 342 684 345 690 357 714 405 810 597)	
	343	124	(343 686 349 698 373 746 469 938 853 683)	
	341	340	(341 682)	
	1	1024	(1 2 4 8 16 32 64 428 256 512)	

Table A.6 Maximum number of solutions to $f(x+a)+f(x)=b$ for power functions, x^d, over the field $\mathbb{F}_{2^{11}}$

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{11}}$	3	2	(3 6 12 24 48 96 192 384 768 1536 1025)	Gold
	5	2	(5 10 20 40 80 160 320 640 1280 513 1026)	Gold
	9	2	(9 18 36 72 144 288 576 1152 257 514 1028)	Gold
	13	2	(13 26 52 104 208 416 832 1664 1281 515 1030)	Kasami
	17	2	(17 34 68 136 272 544 1088 129 258 516 1032)	Gold
	33	2	(33 66 132 264 528 1056 65 130 260 520 1040)	Gold
	35	2	(35 70 140 280 560 1120 193 386 772 1544 1041)	Welch
	43	2	(43 86 172 344 688 1376 705 1410 773 1546 1045)	Kasami
	57	2	(57 114 228 456 912 1824 1601 1155 263 526 1052)	Kasami
	63	2	(63 126 252 504 1008 2016 1985 1923 1799 1551 1055)	Gold
	95	2	(95 190 380 760 1520 993 1986 1925 1803 1559 1071)	Kasami
	107	2	(107 214 428 856 1712 1377 707 1414 781 1562 1077)	Niho
	117	2	(117 234 468 936 1872 1697 1347 647 1294 541 1082)	Welch
	143	2	(143 286 572 1144 241 482 964 1928 1809 1571 1095)	Kasami
	151	2	(151 302 604 1208 369 738 1476 905 1810 1573 1099)	Kasami
	231	2	(231 462 924 1848 1649 1251 455 910 1820 1593 1139)	Gold
	249	2	(249 498 996 1992 1937 1827 1607 1167 287 574 1148)	Niho
	315	2	(315 630 1260 473 946 1892 1737 1427 807 1614 1181)	Kasami
	365	2	(365 730 1460 873 1746 1445 843 1686 1325 603 1206)	Gold
	411	2	(411 822 1644 1241 435 870 1740 1433 819 1638 1229)	Gold
	413	2	(413 826 1652 1257 467 934 1868 1689 1331 615 1230)	Kasami
	683	2	(683 1366 685 1370 693 1386 725 1450 853 1706 1365)	Gold
	1023	2	(1023 2046 2045 2043 2039 2031 2015 1983 1919 1791 1535)	Field Inverse
	79	4	(79 158 316 632 1264 481 962 1924 1801 1555 1063)	
	109	4	(109 218 436 872 1744 1441 835 1670 1293 539 1078)	
	183	4	(183 366 732 1464 881 1762 1477 907 1814 1581 1115)	
	251	4	(251 502 1004 2008 1969 1891 1735 1423 799 1598 1149)	
	367	4	(367 734 1468 889 1778 1509 971 1942 1837 1627 1207)	
	463	4	(463 926 1852 1657 1267 487 974 1948 1849 1651 1255)	
	695	4	(695 1390 733 1466 885 1770 1493 939 1878 1709 1371)	
	703	4	(703 1406 765 1530 1013 2026 2005 1963 1879 1711 1375)	
	7	6	(7 14 28 56 112 224 448 896 1792 1537 1027)	
	11	6	(11 22 44 88 176 352 704 1408 769 1538 1029)	
	15	6	(15 30 60 120 240 480 960 1920 1793 1539 1031)	
	21	6	(21 42 84 168 336 672 1344 641 1282 517 1034)	
	29	6	(29 58 116 232 464 928 1856 1665 1283 519 1038)	
	31	6	(31 62 124 248 496 992 1984 1921 1795 1543 1039)	
	37	6	(37 74 148 296 592 1184 321 642 1284 521 1042)	

Table A.6 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{11}}$	47	6	(47 94 188 376 752 1504 961 1922 1797 1547 1047)	
	49	6	(49 98 196 392 784 1568 1089 131 262 524 1048)	
	51	6	(51 102 204 408 816 1632 1217 387 774 1548 1049)	
	53	6	(53 106 212 424 848 1696 1345 643 1286 525 1050)	
	55	6	(55 110 220 440 880 1760 1473 899 1798 1549 1051)	
	67	6	(67 134 268 536 1072 97 194 388 776 1552 1057)	
	71	6	(71 142 284 568 1136 225 450 900 1800 1553 1059)	
	73	6	(73 146 292 584 1168 289 578 1156 265 530 1060)	
	75	6	(75 150 300 600 1200 353 706 1412 777 1554 1061)	
	81	6	(81 162 324 648 1296 545 1090 133 266 532 1064)	
	83	6	(83 166 332 664 1328 609 1218 389 778 1556 1065)	
	85	6	(85 170 340 680 1360 673 1346 645 1290 533 1066)	
	99	6	(99 198 396 792 1584 1121 195 390 780 1560 1073)	
	101	6	(101 202 404 808 1616 1185 323 646 1292 537 1074)	
	103	6	(103 206 412 824 1648 1249 451 902 1804 1561 1075)	
	111	6	(111 222 444 888 1776 1505 963 1926 1805 1563 1079)	
	113	6	(113 226 452 904 1808 1569 1091 135 270 540 1080)	
	121	6	(121 242 484 968 1936 1825 1603 1159 271 542 1084)	
	125	6	(125 250 500 1000 2000 1953 1859 1671 1295 543 1086)	
	127	6	(127 254 508 1016 2032 2017 1987 1927 1807 1567 1087)	
	137	6	(137 274 548 1096 145 290 580 1160 273 546 1092)	
	139	6	(139 278 556 1112 177 354 708 1416 785 1570 1093)	
	149	6	(149 298 596 1192 337 674 1348 649 1298 549 1098)	
	153	6	(153 306 612 1224 401 802 1604 1161 275 550 1100)	
	155	6	(155 310 620 1240 433 866 1732 1417 787 1574 1101)	
	157	6	(157 314 628 1256 465 930 1860 1673 1299 551 1102)	
	159	6	(159 318 636 1272 497 994 1988 1929 1811 1575 1103)	
	167	6	(167 334 668 1336 625 1250 453 906 1812 1577 1107)	
	171	6	(171 342 684 1368 689 1378 709 1418 789 1578 1109)	
	173	6	(173 346 692 1384 721 1442 837 1674 1301 555 1110)	
	179	6	(179 358 716 1432 817 1634 1221 395 790 1580 1113)	
	181	6	(181 362 724 1448 849 1698 1349 651 1302 557 1114)	
	185	6	(185 370 740 1480 913 1826 1605 1163 279 558 1116)	
	187	6	(187 374 748 1496 945 1890 1733 1419 791 1582 1117)	
	189	6	(189 378 756 1512 977 1954 1861 1675 1303 559 1118)	
	191	6	(191 382 764 1528 1009 2018 1989 1931 1815 1583 1119)	
	201	6	(201 402 804 1608 1169 291 582 1164 281 564 1124)	
	203	6	(203 406 812 1624 1201 355 710 1420 793 1586 1125)	

Table A.6 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{11}}$	205	6	(205 410 820 1640 1233 419 838 1676 1305 563 1126)	
	213	6	(213 426 852 1704 1361 675 1350 653 1306 565 1130)	
	215	6	(215 430 860 1720 1393 739 1478 909 1818 1589 1131)	
	217	6	(217 434 868 1736 1425 803 1606 1165 283 566 1132)	
	219	6	(219 438 876 1752 1457 867 1734 1421 795 1590 1133)	
	221	6	(221 442 884 1768 1489 931 1862 1677 1307 567 1134)	
	223	6	(223 446 892 1784 1521 995 1990 1933 1819 1591 1135)	
	229	6	(229 458 916 1832 1617 1187 327 654 1308 569 1138)	
	247	6	(247 494 988 1976 1905 1763 1479 911 1822 1597 1147)	
	255	6	(255 510 1020 2040 2033 2019 1991 1935 1823 1599 1151)	
	293	6	(293 586 1172 297 594 1188 329 658 1316 585 1170)	
	295	6	(295 590 1180 313 626 1252 457 914 1828 1609 1171)	
	301	6	(301 602 1204 361 722 1444 841 1682 1317 587 1174)	
	307	6	(307 614 1228 409 818 1636 1225 403 806 1612 1177)	
	309	6	(309 618 1236 425 850 1700 1353 659 1318 589 1178)	
	311	6	(311 622 1244 441 882 1764 1481 915 1830 1613 1179)	
	317	6	(317 634 1268 489 978 1956 1865 1683 1319 591 1182)	
	319	6	(319 638 1276 505 1010 2020 1993 1939 1831 1615 1183)	
	331	6	(331 662 1324 601 1202 357 714 1428 809 1618 1189)	
	333	6	(333 666 1332 617 1234 421 842 1684 1321 595 1190)	
	335	6	(335 670 1340 633 1266 485 970 1940 1833 1619 1191)	
	339	6	(339 678 1356 665 1330 613 1226 405 810 1620 1193)	
	341	6	(341 682 1364 681 1362 677 1354 661 1322 597 1194)	
	343	6	(343 686 1372 697 1394 741 1482 917 1834 1621 1195)	
	347	6	(347 694 1388 729 1458 869 1738 1429 811 1622 1197)	
	351	6	(351 702 1404 761 1522 997 1994 1941 1835 1623 1199)	
	359	6	(359 718 1436 825 1650 1253 459 918 1836 1625 1203)	
	371	6	(371 742 1784 921 1842 1637 1227 407 814 1628 1209)	
	373	6	(373 746 1492 937 1874 1701 1355 663 1326 605 1210)	
	375	6	(375 750 1500 953 1906 1765 1483 919 1838 1629 1211)	
	379	6	(379 758 1516 985 1970 1893 1739 1431 815 1630 1213)	
	381	6	(381 762 1524 1001 2002 1957 1867 1687 1327 607 1214)	
	383	6	(383 766 1532 1017 2034 2021 1995 1943 1839 1631 1215)	
	423	6	(423 846 1692 1337 627 1254 461 922 1844 1641 1235)	
	427	6	(427 854 1708 1369 691 1382 717 1434 821 1642 1237)	
	443	6	(443 886 1772 1497 947 1894 1741 1435 823 1646 1245)	
	469	6	(469 938 1876 1705 1363 679 1358 669 1338 629 1258)	
	471	6	(471 942 1884 1721 1395 743 1486 925 1850 1653 1259)	

Table A.6 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{11}}$	475	6	(475 950 1900 1753 1459 871 1742 1437 827 1654 1261)	
	477	6	(477 954 1908 1769 1491 935 1870 1693 1339 631 1262)	
	479	6	(479 958 1916 1785 1523 999 1998 1949 1851 1655 1263)	
	491	6	(491 982 1964 1881 1715 1383 719 1438 829 1658 1269)	
	493	6	(493 986 1972 1897 1747 1447 847 1694 1341 635 1270)	
	495	6	(495 990 1980 1913 1779 1511 975 1950 1853 1659 1271)	
	507	6	(507 1014 2028 2009 1971 1895 1743 1439 831 1662 1277)	
	511	6	(511 1022 2044 2041 2035 2023 1999 1951 1855 1663 1279)	
	687	6	(687 1374 701 1402 757 1514 981 1962 1877 1707 1367)	
	727	6	(727 1454 861 1722 1397 747 1494 941 1882 1717 1387)	
	731	6	(731 1462 877 1754 1461 875 1750 1453 859 1718 1389)	
	735	6	(735 1470 893 1786 1525 1003 2006 1965 1883 1719 1391)	
	751	6	(751 1502 957 1914 1781 1515 983 1966 1885 1723 1399)	
	763	6	(763 1526 1005 2010 1973 1899 1751 1455 863 1726 1405)	
	767	6	(767 1534 1021 2042 2037 2027 2007 1967 1887 1727 1407)	
	879	6	(879 1758 1469 891 1782 1517 987 1974 1901 1755 1463)	
	887	6	(887 1774 1501 955 1910 1773 1499 951 1902 1757 1467)	
	959	6	(959 1918 1789 1531 1015 2030 2013 1979 1911 1775 1503)	
	991	6	(991 1982 1917 1787 1527 1007 2014 1981 1915 1783 1519)	
	19	8	(19 38 76 152 304 608 1216 385 770 1540 1033)	
	25	8	(25 50 100 200 400 800 1600 1153 259 518 1036)	
	27	8	(27 54 108 216 432 864 1728 1409 771 1542 1037)	
	39	8	(39 78 156 312 624 1248 449 898 1796 1545 1043)	
	41	8	(41 82 164 328 656 1312 577 1154 261 522 1044)	
	45	8	(45 90 180 360 720 1440 833 1666 1285 523 1046)	
	61	8	(61 122 244 488 976 1952 1857 1667 1287 527 1054)	
	77	8	(77 154 308 616 1232 417 834 1668 1289 531 1062)	
	87	8	(87 174 348 696 1392 737 1474 901 1802 1557 1067)	
	91	8	(91 182 364 728 1456 865 1730 1413 779 1558 1069)	
	105	8	(105 210 420 840 1680 1313 579 1158 269 538 1076)	
	119	8	(119 238 476 952 1904 1761 1475 903 1806 1565 1083)	
	123	8	(123 246 492 984 1968 1889 1731 1415 783 1566 1085)	
	141	8	(141 282 564 1128 209 418 836 1672 1297 547 1094)	
	147	8	(147 294 588 1176 305 610 1220 393 786 1572 1097)	
	163	8	(163 326 652 1304 561 1122 197 394 788 1576 1105)	
	165	8	(165 330 660 1320 593 1186 325 650 1300 553 1106)	
	175	8	(175 350 700 1400 753 1506 965 1930 1813 1579 1111)	
	199	8	(199 398 796 1592 1137 227 454 908 1816 1585 1123)	

Table A.6 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{11}}$	211	8	(211 422 844 1688 1329 611 1222 397 794 1588 1129)	
	233	8	(233 466 932 1864 1681 1315 583 1166 285 570 1140)	
	235	8	(235 470 940 1880 1713 1379 711 1422 797 1594 1141)	
	237	8	(237 474 948 1896 1745 1443 839 1678 1309 571 1142)	
	239	8	(239 478 956 1912 1777 1507 967 1934 1821 1595 1143)	
	349	8	(349 698 1396 745 1490 933 1866 1685 1323 599 1198)	
	363	8	(363 726 1452 857 1714 1381 715 1430 813 1626 1205)	
	415	8	(415 830 1660 1273 499 998 1996 1945 1843 1639 1231)	
	429	8	(429 858 1716 1385 723 1446 845 1690 1333 619 1238)	
	431	8	(431 862 1724 1401 755 1510 973 1946 1845 1643 1239)	
	439	8	(439 878 1756 1465 883 1766 1485 923 1846 1645 1243)	
	501	8	(501 1002 2004 1961 1875 1703 1359 671 1342 637 1274)	
	503	8	(503 1006 2012 1977 1907 1767 1487 927 1854 1661 1275)	
	699	8	(699 1398 749 1498 949 1898 1749 1451 855 1710 1373)	
	895	8	(895 1790 1533 1019 2038 2029 2011 1975 1903 1759 1471)	
	59	10	(59 118 236 472 944 1888 1729 1411 775 1550 1053)	
	93	10	(93 186 372 744 1488 929 1858 1669 1291 535 1070)	
	169	10	(169 338 676 1352 657 1314 581 1162 277 554 1108)	
	243	10	(243 486 972 1944 1841 1635 1223 399 798 1596 1145)	
	303	10	(303 606 1212 377 754 1508 969 1938 1829 1611 1175)	
	509	10	(509 1018 2036 2025 2003 1959 1871 1695 1343 639 1278)	
	245	16	(245 490 980 1960 1873 1699 1351 655 1310 573 1146)	
	447	16	(447 894 1788 1529 1011 2022 1997 1947 1847 1647 1247)	
	23	22	(23 46 92 184 368 736 1472 897 1794 1541 1035)	
	69	22	(69 138 276 552 1104 161 322 644 1288 529 1058)	
	115	22	(115 230 460 920 1840 1633 1219 391 782 1564 1081)	
	207	22	(207 414 828 1656 1265 483 966 1932 1817 1587 1127)	
	253	22	(253 506 1012 2024 2001 1955 1863 1679 1311 575 1150)	
	299	22	(299 598 1196 345 690 1380 713 1426 805 1610 1173)	
	437	22	(437 874 1748 1449 851 1702 1357 667 1334 621 1242)	
	759	22	(759 1518 989 1978 1909 1771 1495 943 1886 1725 1403)	
	89	88	(89 178 356 712 1424 801 1602 1157 267 534 1068)	
	445	88	(445 890 1780 1513 979 1958 1869 1691 1335 623 1246)	
	1	2048	(1 2 4 8 16 32 64 128 256 512 1024)	

Table A.7 Maximum number of solutions to $f(x+a)+f(x)=b$ for power functions, x^d, over the field $\mathbb{F}_{2^{12}}$

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	3	2	(3 6 12 24 48 96 192 384 768 1536)	Gold
	33	2	(33 66 132 264 528 1056 2112 129 258 516 1032 2064)	Gold
	159	2	(159 318 636 1272 2544 993 1986 3972 3849 3603 3111 2127)	Kasami
	5	4	(5 10 20 40 80 160 320 640 1280 2560 1025 2050)	
	73	4	(73 146 292 584 1168 2336 577 1154 2308 521 1042 2084)	
	731	4	(731 1462 2924 1753 3506 2917 1739 3478 2861 1627 3254 2413)	
	2047	4	(2047 4094 4093 4091 4087 4079 4063 4031 3967 3839 3583 3071)	
	7	6	(7 14 28 56 112 224 448 896 1792 3584 3073 2051)	
	145	6	(145 290 580 1160 2320 545 1090 2180 265 530 1060 2120)	
	187	6	(187 374 748 1496 2992 1889 3778 3461 2827 1559 3118 2141)	
	327	6	(327 654 1308 2616 1137 2274 453 906 1812 3624 3153 2211)	
	341	6	(341 682 1364 2728 1361 2722 1349 2698 1301 2602 1109 2218)	
	343	6	(343 686 1372 2744 1393 2786 1477 2954 1813 3626 3157 2219)	
	363	6	(363 726 1452 2904 1713 3426 2757 1419 2838 1581 3162 2229)	
	425	6	(425 850 1700 3400 2705 1315 2630 1165 2330 565 1130 2260)	
	447	6	(447 894 1788 3576 3057 2019 4038 3981 3867 3639 3183 2271)	
	489	6	(489 978 1956 3912 3729 3363 2631 1167 2334 573 1146 2292)	
	699	6	(699 1398 2796 1497 2994 1893 3786 3477 2859 1623 3246 2397)	
	749	6	(749 1498 2996 1897 3794 3493 2891 1687 3374 2653 1211 2422)	
	763	6	(763 1526 3052 2009 4018 3941 3787 3479 2863 1631 3262 2429)	
	847	6	(847 1694 3388 2681 1267 2534 973 1946 3892 3689 3283 2471)	
	853	6	(853 1706 3412 2729 1363 2726 1357 2714 1333 2666 1237 2474)	
	943	6	(943 1886 3772 3449 2803 1511 3022 1949 3898 3701 3307 2519)	
	957	6	(957 1914 3828 3561 3027 1959 3918 3741 3387 2679 1263 2526)	
	959	6	(959 1918 3836 3577 3059 2023 4046 3997 3899 3703 3311 2527)	
	1371	6	(1371 2742 1389 2778 1461 2922 1749 3498 2901 1707 3414 2733)	
	9	8	(9 18 36 72 144 288 576 1152 2304 513 1026 2052)	
	51	8	(51 102 204 408 816 1632 3264 2433 771 1542 3084 2073)	
	55	8	(55 110 220 440 880 1760 3520 2945 1795 3590 3085 2075)	
	87	8	(87 174 348 696 1392 2784 1473 2946 1797 3594 3093 2091)	
	93	8	(93 186 372 744 1488 2976 1857 3714 3333 2571 1047 2094)	
	97	8	(97 194 388 776 1552 3104 2113 131 262 524 1048 2096)	
	141	8	(141 282 564 1128 2256 417 834 1668 3336 2577 1059 2118)	
	177	8	(177 354 708 1416 2832 1569 3138 2181 267 534 1068 2136)	
	207	8	(207 414 828 1656 3312 2529 963 1926 3852 3609 3123 2151)	
	209	8	(209 418 836 1672 3344 2593 1091 2182 269 538 1076 2152)	
	213	8	(213 426 852 1704 3408 2721 1347 2694 1293 2586 1077 2154)	
	217	8	(217 434 868 1736 3472 2849 1603 3206 2317 539 1078 2156)	
	237	8	(237 474 948 1896 3792 3489 2883 1671 3342 2589 1083 2166)	

Table A.7 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	249	8	(249 498 996 1992 3984 3873 3651 3207 2319 543 1086 2172)	
	251	8	(251 502 1004 2008 4016 3937 3779 3463 2831 1567 3134 2173)	
	303	8	(303 606 1212 2424 753 1506 3012 1929 3858 3621 3147 2199)	
	311	8	(311 622 1244 2488 881 1762 33524 2953 1811 3622 3149 2203)	
	329	8	(329 658 1316 2632 1169 2338 581 1162 2324 553 1106 2212)	
	333	8	(333 666 1332 2664 1233 2466 837 1674 3348 2601 1107 2214)	
	335	8	(335 670 1340 2680 1265 2530 965 1930 3860 3625 3155 2215)	
	339	8	(339 678 1356 2712 1329 2658 1221 2442 789 1578 3156 2217)	
	381	8	(381 762 1524 3048 2001 4002 3909 3723 3351 2607 1119 2238)	
	427	8	(427 854 1708 3416 2737 1379 2758 1421 2842 1589 3178 2261)	
	477	8	(477 954 1908 3816 3537 2979 1863 3726 3357 2619 1143 2286)	
	591	8	(591 1182 2364 633 1266 2532 969 1938 3876 3657 3219 2343)	
	597	8	(597 1194 2388 681 1362 2724 1353 2706 1317 2634 1173 2346)	
	603	8	(603 1206 2412 729 1458 2916 1737 3474 2853 1611 3222 2349)	
	671	8	(671 1342 2684 1273 2546 997 1994 3988 3881 3667 3239 2383)	
	719	8	(719 1438 2876 1657 3314 2533 971 1942 3884 3673 3251 2407)	
	723	8	(723 1446 2892 1689 3378 2661 1227 2454 813 1626 3252 2409)	
	727	8	(727 1454 2908 1721 3442 2789 1483 2966 1837 3674 3253 2411)	
	755	8	(755 1510 3020 1945 3890 3685 3275 2455 815 1630 3260 2425)	
	759	8	(759 1518 3036 1977 3954 3813 3531 2967 1839 3678 3261 2427)	
	879	8	(879 1758 3516 2937 1779 3558 3021 1947 3894 3693 3291 2487)	
	891	8	(891 1782 3564 3033 1971 3942 3789 3483 2871 1647 3294 2493)	
	895	8	(895 1790 3580 3065 2035 4070 4045 3995 3895 3695 3295 2495)	
	981	8	(981 1962 3924 3753 3411 2727 1359 2718 1341 2682 1269 2538)	
	1023	8	(1023 2046 4092 4089 4083 4071 4047 3999 3903 3711 3327 2559)	
	1467	8	(1467 2934 1773 3546 2997 1899 3798 3501 2907 1719 3438 2781)	
	1527	8	(1527 3054 2013 4026 3957 3819 3543 2991 1887 3774 3453 2811)	
	1783	8	(1783 3566 3037 1979 3958 3821 3547 2999 1903 3806 3517 2939)	
	1983	8	(1983 3966 3837 3579 3063 2031 4062 4029 3963 3831 3567 3039)	
	11	10	(11 22 44 88 176 352 704 1408 2816 1537 3074 2053)	
	37	10	(37 74 148 296 592 1184 2368 641 1282 2564 1033 2066)	
	43	10	(43 86 172 344 688 1376 2752 1409 2818 1541 3082 2069)	
	59	10	(59 118 236 472 944 1888 3776 3457 2819 1543 3086 2077)	
	69	10	(69 138 276 552 1104 2208 321 642 1284 2568 1041 2082)	
	81	10	(81 162 324 648 1296 2592 1089 2178 261 522 1044 2088)	
	85	10	(85 170 340 680 1360 2720 1345 2690 1285 2570 1045 2090)	
	99	10	(99 198 396 792 1584 3168 2241 387 774 1548 3096 2097)	
	119	10	(119 238 476 952 1904 3808 3521 2947 1799 3598 3101 2107)	
	123	10	(123 246 492 984 1968 3936 3777 3459 2823 1551 3102 2109)	

Table A.7 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	133	10	(133 266 532 1064 2128 161 322 644 1288 2576 1057 2114)	
	149	10	(149 298 596 1192 2384 673 1346 2692 1289 2578 1061 2122)	
	155	10	(155 310 620 1240 2480 865 1730 3460 2825 1555 3110 2125)	
	157	10	(157 314 628 1256 2512 929 1858 3716 3337 2579 1063 2126)	
	163	10	(163 326 652 1304 2608 1121 2242 389 778 1556 3112 2129)	
	179	10	(179 358 716 1432 2864 1633 3266 2437 779 1558 3116 2137)	
	183	10	(183 366 732 1464 2928 1761 3522 2949 1803 3606 3117 2139)	
	201	10	(201 402 804 1608 3216 2337 579 1158 2316 537 1074 2148)	
	283	10	(283 566 1132 2264 433 866 1732 3464 2833 1571 3142 2189)	
	291	10	(291 582 1164 2328 561 1122 2244 393 786 1572 3144 2193)	
	295	10	(295 590 1180 2360 625 1250 2500 905 1810 3620 3145 2495)	
	307	10	(307 614 1228 2456 817 1634 3268 2441 787 1574 3148 2201)	
	309	10	(309 618 1236 2472 849 1698 3396 2697 1299 2598 1101 2202)	
	313	10	(313 626 1252 2504 913 1826 3652 3209 2323 551 1102 2204)	
	359	10	(359 718 1436 2872 1649 3298 2501 907 1814 3628 3161 2227)	
	365	10	(365 730 1460 2920 1745 3490 2885 1675 3350 2605 1115 2230)	
	371	10	(371 742 1484 2968 1841 3682 3269 2443 791 1582 3164 2233)	
	373	10	(373 746 1492 2984 1873 3746 3397 2699 1303 2606 1117 2234)	
	431	10	(431 862 1724 3448 2801 1507 3014 1933 3866 3637 33179 2263)	
	485	10	(485 970 1940 3880 3665 3235 2375 655 1310 2620 1145 2290)	
	491	10	(491 982 1964 3928 3761 3427 2759 1423 2846 1597 3194 2293)	
	613	10	(613 1226 2452 809 1618 3236 2377 659 1318 2636 1177 2354)	
	629	10	(629 1258 2516 937 1874 3748 3401 2707 1319 2638 1181 2362)	
	667	10	(667 1334 2668 1241 2482 869 1738 3476 2857 1619 3238 2381)	
	717	10	(717 1434 2868 1641 3282 2469 843 1686 3372 2649 1203 2406)	
	831	10	(831 1662 3324 2553 1011 2022 4044 3993 3891 3687 3279 2463)	
	877	10	(877 1754 3508 2921 1747 3494 2893 1691 3382 2669 1243 2486)	
	925	10	(925 1850 3700 3305 2515 935 1870 3740 3385 2675 1255 2510)	
	941	10	(941 1882 3764 3433 2771 1447 2894 1693 3386 2677 1259 2518)	
	955	10	(955 1910 3820 3545 2995 1895 3790 3485 2875 1655 3310 2525)	
	1003	10	(1003 2006 4012 3929 3763 3431 2767 1439 2878 1661 3322 2549)	
	1019	10	(1019 2038 4076 4057 4019 3943 3791 3487 2879 1663 3326 2557)	
	1499	10	(1499 2998 1901 3802 3509 2923 1751 3502 2909 1723 3446 2797)	
	1535	10	(1535 3070 2045 4090 4085 4075 4055 4015 3935 3775 3455 2815)	
	13	12	(13 26 52 104 208 416 832 1664 3328 2561 1027 2054)	
	27	12	(27 54 108 216 432 864 1728 3456 2817 1539 3078 2061)	
	101	12	(101 202 404 808 1616 3232 2369 643 1286 2572 1049 2098)	
	111	12	(111 222 444 888 1776 3552 3009 1923 3846 3597 3099 2103)	
	115	12	(115 230 460 920 1840 3680 3265 2435 775 1550 3100 2105)	

Table A.7 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	153	12	(153 306 612 1224 2448 801 1602 3204 2313 531 1062 2124)	
	171	12	(171 342 684 1368 2736 1377 2754 1413 2826 1557 3114 2133)	
	215	12	(215 430 860 1720 3440 2785 1475 2950 1805 3610 3125 2155)	
	223	12	(223 446 892 1784 3568 3041 1987 3974 3853 3611 3127 2159)	
	247	12	(247 494 988 1976 3952 3809 3523 2951 1807 3614 3133 2171)	
	279	12	(279 558 1116 2232 369 738 1476 2952 1809 3618 3141 2187)	
	297	12	(297 594 1188 2376 657 1314 2628 1161 2322 549 1098 2196)	
	299	12	(299 598 1196 2392 689 1378 2756 1417 2834 1573 3146 2197)	
	367	12	(367 734 1468 2936 1777 3554 3013 1931 3862 3629 3163 2231)	
	403	12	(403 806 1612 3224 2353 611 1222 2444 793 1586 3172 2249)	
	423	12	(423 846 1692 3384 2673 1251 2502 909 1818 3636 3177 2259)	
	461	12	(461 922 1844 3688 3281 2467 839 1678 3356 2617 1139 2278)	
	479	12	(479 958 1916 3832 3569 3043 1991 3982 3869 3643 3191 2287)	
	493	12	(493 986 1972 3944 3793 3491 2887 1679 3358 2621 1147 2294)	
	607	12	(607 1241 2428 761 1522 3044 1993 3986 3877 3659 3223 2351)	
	621	12	(621 1242 2484 873 1746 3492 2889 1683 3366 2637 1179 2358)	
	627	12	(627 1254 2508 921 1842 3684 3273 2451 807 1614 3228 2361)	
	635	12	(635 1270 2540 985 1970 3940 3785 3475 2855 1615 3230 2365)	
	703	12	(703 1406 2812 1529 3058 2021 4042 3989 3883 3671 3247 2399)	
	747	12	(747 1494 2988 1881 3762 3429 2763 1431 2862 1629 3258 2421)	
	927	12	(927 1854 3708 3321 2547 999 1998 3996 3897 3699 3303 2511)	
	1375	12	(1375 2750 1405 2810 1525 3050 2005 4010 3925 3755 3415 2735)	
	1391	12	(1391 2782 1469 2938 1781 3562 3029 1963 3926 3757 3419 2743)	
	1503	12	(1503 3006 1917 3834 3573 3051 2007 4014 3933 3771 3447 2799)	
	1791	12	(1791 3582 3069 2043 4086 4077 4059 4023 3951 3807 3519 2943)	
	15	14	(15 30 60 120 240 480 960 1920 3840 3585 3075 2055)	
	75	14	(75 150 300 600 1200 2400 705 1410 2820 1545 3090 2085)	
	89	14	(89 178 356 712 1424 2848 1601 3202 2309 523 1046 2092)	
	125	14	(125 250 500 1000 2000 4000 3905 3715 3335 2575 1055 2110)	
	165	14	(165 330 660 1320 2640 1185 2370 645 1290 2580 1065 2130)	
	243	14	(243 486 972 1944 3888 3681 3267 2439 783 1566 3132 2169)	
	255	14	(255 510 1020 2040 4080 4065 4035 3975 3855 3615 3135 2175)	
	285	14	(285 570 1140 2280 465 930 1860 3720 3345 2595 1095 2190)	
	345	14	(345 690 1380 2760 1425 2850 1605 3210 2325 555 1110 2220)	
	375	14	(375 750 1500 3000 1905 3810 3525 2955 1815 3630 3165 2235)	
	415	14	(415 830 1660 3320 2545 995 1990 3980 3865 3635 3175 2255)	
	435	14	(435 870 1740 3480 2865 1635 3270 2445 795 1590 3180 2265)	
	501	14	(501 1002 2004 4008 3921 3747 3399 2703 1311 2622 1149 2298)	
	509	14	(509 1018 2036 4072 4049 4003 3911 3727 3359 2623 1151 2302)	

Table A.7 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	615	14	(615 1230 2460 825 1650 3300 2505 915 1830 3660 3225 2355)	
	669	14	(669 1338 2676 1257 2514 933 1866 3732 3369 2643 1191 2382)	
	687	14	(687 1374 2748 1401 2802 1509 3018 1941 3882 3669 3243 2391)	
	885	14	(885 1770 3540 2985 1875 3750 3405 2715 1335 2670 1245 2490)	
	951	14	(951 1902 3804 3513 2931 1767 3534 2973 1851 3702 3309 2523)	
	1005	14	(1005 2010 4020 3945 3795 3495 2895 1695 3390 2685 1275 2550)	
	1455	14	(1455 2910 1725 3450 2805 1515 3030 1965 3930 3765 3435 2775)	
	17	16	(17 34 68 136 272 544 1088 2176 257 514 1028 2056)	
	19	16	(19 38 76 152 304 608 1216 2432 769 1538 3076 2057)	
	25	16	(25 50 100 200 400 800 1600 3200 2305 515 1030 2060)	
	31	16	(31 62 124 248 496 992 1984 3968 3841 3587 3079 2063)	
	47	16	(47 94 188 376 752 1504 3008 1921 3842 3589 3083 2071)	
	49	16	(49 98 196 392 784 1568 3136 2177 259 518 1036 2072)	
	61	16	(61 122 244 488 976 1952 3904 3713 3331 2567 1039 2078)	
	77	16	(77 154 308 616 1232 2464 833 1666 3332 2569 1043 2086)	
	83	16	(83 166 332 664 1328 2656 1217 2434 773 1546 3092 2089)	
	139	16	(139 278 556 1112 2224 353 706 1412 2824 1553 3106 2117)	
	143	16	(143 286 572 1144 2288 481 962 1924 3848 3601 3107 2119)	
	167	16	(167 334 668 1336 2672 1249 2498 901 1802 3604 3113 2131)	
	173	16	(173 346 692 1384 2768 1441 2882 1669 3338 2581 1067 2134)	
	181	16	(181 362 724 1448 2896 1697 3394 2693 1291 2582 1069 2138)	
	199	16	(199 398 796 1592 3184 2273 451 902 1804 3608 3121 2147)	
	203	16	(203 406 812 1624 3248 2401 707 1414 2828 1561 3122 2149)	
	227	16	(227 454 908 1816 3632 3169 2243 391 782 1564 3128 2161)	
	229	16	(229 458 916 1832 3664 3233 2371 647 1294 2588 1081 2162)	
	241	16	(241 482 964 1928 3856 3617 3139 2183 271 542 1084 2168)	
	287	16	(287 574 1148 2296 497 994 1988 3976 3857 3619 3143 2191)	
	301	16	(301 602 1204 2408 721 1442 2884 1673 3346 2597 1099 2198)	
	349	16	(349 698 1396 2792 1489 2978 1861 3722 3349 2603 1111 2222)	
	355	16	(355 710 1420 2840 1585 3170 2245 395 790 1580 3160 2225)	
	377	16	(377 754 1508 3016 1937 3874 3653 3211 2327 559 1118 2236)	
	383	16	(383 766 1532 3064 2033 4066 4037 3979 3863 3631 3167 2239)	
	409	16	(409 818 1636 3272 2449 803 1606 3212 2329 563 1126 2252)	
	413	16	(413 826 1652 3304 2513 931 1862 3724 3353 2611 1127 2254)	
	437	16	(437 874 1748 3496 2897 1699 3398 2701 1307 2614 1133 2266)	
	467	16	(467 934 1868 3736 3377 2659 1223 2446 797 1594 3188 2281)	
	475	16	(475 950 1900 3800 3505 2915 1735 3470 2845 1595 3190 2285)	
	503	16	(503 1006 2012 4024 3953 3811 3527 2959 1823 3646 3197 2299)	
	511	16	(511 1022 2044 4088 4081 4067 4039 3983 3871 3647 3199 2303)	

Table A.7 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	619	16	(619 1238 2476 857 1714 3428 2761 1427 2854 1613 3226 2357)	
	623	16	(623 1246 2492 889 1778 3556 3017 1939 3878 3661 3227 2359)	
	661	16	(661 1322 2644 1193 2386 677 1354 2708 1321 2642 1189 2378)	
	679	16	(679 1358 2716 1337 2674 1253 2506 917 1834 3668 3241 2387)	
	691	16	(691 1382 2764 1433 2866 1637 3274 2453 811 1622 3244 2393)	
	829	16	(829 1658 3316 2537 979 1958 3916 3737 3379 2663 1231 2462)	
	859	16	(859 1718 3436 2777 1459 2918 1741 3482 2869 1643 3286 2477)	
	871	16	(871 1742 3484 2873 1651 3302 2509 923 1846 3692 3289 2483)	
	887	16	(887 1774 3548 3001 1907 3814 3533 2971 1847 3694 3293 2491)	
	983	16	(983 1966 3932 3769 3443 2791 1487 2974 1853 3706 3317 2539)	
	1007	16	(1007 2014 4028 3961 3827 3559 3023 1951 3902 3709 3323 2551)	
	1013	16	(1013 2026 4052 4009 3923 3751 3407 2719 1343 2686 1277 2554)	
	1021	16	(1021 2042 4084 4073 4051 4007 3919 3743 3391 2687 1279 2558)	
	1399	16	(1399 2798 1501 3002 1909 3818 3541 2987 1879 3758 3421 2747)	
	1403	16	(1403 2806 1517 3034 1973 3946 3797 3499 2903 1711 3422 2749)	
	1519	16	(1519 3038 1981 3962 3829 3563 3031 1967 3934 3773 3451 2807)	
	1531	16	(1531 3062 2029 4058 4021 3947 3799 3503 2911 1727 3454 2813)	
	639	18	(639 1278 2556 1017 2034 4068 4041 3987 3879 3663 3231 2367)	
	685	18	(685 1370 2740 1385 2770 1445 2890 1685 3370 2645 1195 2390)	
	21	20	(21 42 84 168 336 672 1344 2688 1281 2562 1029 2058)	
	29	20	(29 58 116 232 464 928 1856 3712 3329 2563 1031 2062)	
	147	20	(147 294 588 1176 2352 609 1218 2436 777 1554 3108 2121)	
	231	20	(231 462 924 1848 3696 3297 2499 903 1806 3612 3129 2163)	
	357	20	(357 714 1428 2856 1617 3234 2373 651 1302 2604 1113 2226)	
	399	20	(399 798 1596 3192 2289 483 966 1932 3864 3633 3171 2247)	
	861	20	(861 1722 3444 2793 1491 2982 1869 3738 3381 2667 1239 2478)	
	987	20	(987 1974 3948 3801 3507 2919 1743 3486 2877 1659 3318 2541)	
	989	20	(989 1978 3956 3817 3539 2983 1871 3742 3389 2683 1271 2542)	
	1407	20	(1407 2814 1533 3066 2037 4074 4053 4011 3927 3759 3423 2751)	
	23	22	(23 46 92 184 368 736 1472 2944 1793 3586 3077 2059)	
	53	22	(53 106 212 424 848 1696 3392 2689 1283 2566 1037 2074)	
	107	22	(107 214 428 856 1712 3424 2753 1411 2822 1549 3098 2101)	
	109	22	(109 218 436 872 1744 3488 2881 1667 3334 2573 1051 2102)	
	113	22	(113 226 452 904 1808 3616 3137 2179 263 526 1052 2104)	
	121	22	(121 242 484 968 1936 3872 3649 3203 2311 527 1054 2108)	
	151	22	(151 302 604 1208 2416 737 1474 2948 1801 3602 3109 2123)	
	169	22	(169 338 676 1352 2704 1313 2626 1157 2314 533 1066 2132)	
	211	22	(211 422 844 1688 3376 2657 1219 2438 781 1562 3124 2153)	
	233	22	(233 466 932 1864 3728 3361 2627 1159 2318 541 1082 2164)	

Table A.7 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	361	22	(361 722 1444 2888 1681 3362 2629 1163 2326 557 1114 2228)	
	407	22	(407 814 1628 3256 2417 739 1478 2956 1817 3634 3173 2251)	
	421	22	(421 842 1684 3368 2641 1187 2374 653 1306 2612 1129 2258)	
	743	22	(743 1486 2972 1849 3698 3301 2507 919 1838 3676 3257 2419)	
	767	22	(767 1534 3068 2041 4082 4069 4043 3991 3887 3679 3263 2431)	
	863	22	(863 1726 3452 2809 1523 3046 1997 3994 3893 3691 3287 2479)	
	893	22	(893 1786 3572 3049 2003 4006 3917 3739 3383 2671 1247 2494)	
	991	22	(991 1982 3964 3833 3571 3047 1999 3998 3901 3707 3319 2543)	
	35	34	(35 70 140 280 560 1120 2240 385 770 1540 3080 2065)	
	175	34	(175 350 700 1400 2800 1505 3010 1925 3850 3605 3115 2135)	
	239	34	(239 478 956 1912 3824 3553 3011 1927 3854 3613 3131 2167)	
	245	34	(245 490 980 1960 3920 3745 3395 2695 1295 2590 1085 2170)	
	275	34	(275 550 1100 2200 305 610 1220 2440 785 1570 3140 2185)	
	277	34	(277 554 1108 2216 337 674 1348 2696 1297 2594 1093 2186)	
	281	34	(281 562 1124 2248 401 802 1604 3208 2321 547 1094 2188)	
	595	34	(595 1190 2380 665 1330 2660 1225 2450 805 1610 3220 2345)	
	875	34	(875 1750 3500 2905 1715 3430 2765 1435 2870 1645 3290 2485)	
	1015	34	(1015 2030 4060 4025 3955 3815 3535 2975 1855 3710 3325 2555)	
	1775	34	(1775 3550 3005 1915 3830 3565 3035 1975 3950 3805 3515 2935)	
	1919	34	(1919 3838 3581 3067 2039 4078 4061 4027 3959 3823 3551 3007)	
	39	38	(39 78 156 312 624 1248 2496 897 1794 3588 3081 2067)	
	411	38	(411 822 1644 3288 2481 867 1734 3468 2841 1587 3174 2253)	
	429	38	(429 858 1716 3432 2769 1443 2886 1677 3354 2613 1131 2262)	
	507	38	(507 1014 2028 4056 4017 3939 3783 3471 2847 1599 3198 2301)	
	663	38	(663 1326 2652 1209 2418 741 1482 2964 1833 3666 3237 2379)	
	41	40	(41 82 164 328 656 1312 2624 1153 2306 517 1034 2068)	
	103	40	(103 206 412 824 1648 3296 2497 899 1798 3596 3097 2099)	
	185	40	(185 370 740 1480 2960 1825 3650 3205 2315 535 1070 2140)	
	397	40	(397 794 1588 3176 2257 419 838 1676 3352 2609 1123 2246)	
	605	40	(605 1210 2420 745 1490 2980 1865 3730 3365 2635 1175 2350)	
	733	40	(733 1466 2932 1759 3538 2981 1867 3734 3373 2651 1207 2114)	
	45	44	(45 90 180 360 720 1440 2880 1665 3330 2565 1035 2070)	
	135	44	(135 270 540 1080 2160 225 450 900 1800 3600 3105 2115)	
	219	44	(219 438 876 1752 3504 2913 1731 3462 2829 1563 3126 2157)	
	405	44	(405 810 1620 3240 2385 675 1350 2700 1305 2610 1125 2250)	
	495	44	(495 990 1980 3960 3825 3555 3015 1935 3870 3645 3195 2295)	
	765	44	(765 1530 3060 2025 4050 4005 3915 3735 3375 2655 1215 2430)	
	855	44	(855 1710 3420 2745 1395 2790 1485 2970 1845 3690 3285 2475)	
	939	44	(939 1878 3756 3417 2739 1383 2766 1437 2874 1653 3306 2517)	

Table A.7 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	137	46	(137 274 548 1096 2192 289 578 1156 2312 529 1058 2116)	
	235	46	(235 470 940 1880 3760 3425 2755 1415 2830 1565 3130 2165)	
	751	46	(751 1502 3004 1913 3826 3557 3019 1943 3886 3677 3259 2423)	
	57	56	(57 114 228 456 912 1824 3648 3201 2307 519 1038 2076)	
	459	56	(459 918 1836 3672 3249 2403 711 1422 2844 1593 3186 2277)	
	471	56	(471 942 1884 3768 3441 2787 1479 2958 1821 3642 3189 2283)	
	499	58	(499 998 1996 3992 3889 3683 33271 2447 799 1598 3196 2297)	
	1471	58	(1471 2942 1789 3578 3061 2027 4054 4013 3931 3767 3439 2783)	
	63	62	(63 126 252 504 1008 2016 4032 3969 3843 3591 3087 2079)	
	189	62	(189 378 756 1512 3024 1953 3906 3717 3339 2583 1071 2142)	
	441	62	(441 882 1764 3528 2961 1827 3654 3213 2331 567 1134 2268)	
	463	62	(463 926 1852 3704 3313 2531 967 1934 3868 3641 3187 2279)	
	487	62	(487 974 1948 3896 3697 3299 2503 911 1822 3644 3193 2291)	
	693	62	(693 1386 2772 1449 2898 1701 33402 2709 1323 2646 1197 2394)	
	65	64	(65 130 260 520 1040 2080)	
	67	64	(67 134 268 536 1072 2144 193 386 772 1544 3088 2081)	
	71	64	(71 142 284 568 1136 2272 449 898 1795 3592 3089 2083)	
	95	64	(95 190 380 760 1520 3040 1985 3970 3845 3595 3095 2095)	
	127	64	(127 254 508 1016 2032 4064 4033 3971 3847 3599 3103 2111)	
	191	64	(191 382 764 1528 3056 2017 4034 3973 3851 3607 3119 2143)	
	195	64	(195 390 780 1560 3120 2145)	
	221	64	(221 442 884 1768 3536 2977 1859 3718 3341 2587 1079 2158)	
	253	64	(253 506 1012 2024 4048 4001 3907 3719 3343 2591 1087 2174)	
	445	64	(445 890 1780 3560 3025 1955 3910 3725 3355 2615 1135 2270)	
	457	64	(457 914 1828 3656 3217 2339 583 1166 2332 569 1138 2276)	
	505	64	(505 1010 2020 4040 3985 3875 3655 3215 2335 575 1150 2300)	
	599	64	(599 1198 2396 697 1394 2788 1481 2962 1829 3658 3221 2347)	
	695	64	(695 1390 2780 1465 2930 1765 3530 2965 1835 3670 3245 2395)	
	701	64	(701 1402 2804 1513 3026 1957 3914 3733 3371 2647 1199 2398)	
	725	64	(725 1450 2900 1705 3410 2725 1355 2710 1325 2650 1205 2410)	
	757	64	(757 1514 3028 1961 3922 3749 3403 2711 1327 2654 1213 2426)	
	1387	64	(1387 2774 1453 2906 1717 3434 2773 1451 2902 1709 3418 2741)	
	79	76	(79 158 316 632 1264 2528 961 1922 3844 3593 3091 2087)	
	197	76	(197 394 788 1576 3152 2209 323 646 1292 2584 1073 2146)	
	317	76	(317 634 1268 2536 977 1954 3908 3721 3347 2599 1103 2206)	
	319	76	(319 638 1276 2552 1009 2018 4036 3977 3859 3623 3151 2207)	
	331	76	(331 662 1324 2648 1201 2402 709 1418 2836 1577 3154 2213)	
	347	76	(347 694 1388 2776 1457 2914 1733 3466 2837 1579 3158 2221)	
	379	76	(379 758 1516 3032 1969 3938 3781 3467 2839 1583 3166 2237)	

Table A.7 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{12}}$	443	76	(443 886 1772 3544 2993 1891 3782 3469 2843 1591 3182 2269)	
	473	76	(473 946 1892 3784 3473 2851 1607 3214 2333 571 1142 2284)	
	631	76	(631 1262 2524 953 1906 3812 3529 2963 1831 3662 3229 2363)	
	949	76	(949 1898 3796 3497 2899 1703 3406 2717 1339 2678 1261 2522)	
	2015	76	(2015 4030 3965 3835 3575 3055)	
	715	82	(715 1430 2860 1625 3250 2405)	
	1495	82	(1495 2990 1885 3770 3445 2795)	
	845	84	(845 1690 3380 2665 1235 2470)	
	293	88	(293 586 1172 2344 593 1186 2372 649 1298 2596 1097 2194)	
	325	88	(325 650 1300 2600 1105 2210)	
	439	88	(439 878 1756 3512 2929 1763 3526 2957 1819 3638 3181 2267)	
	587	88	(587 1174 2348 601 1202 2404 713 1426 2852 1609 2318 2341)	
	1463	88	(1463 2926 1757 3514 2933 1771 3542 2989 1883 3766 3437 2779)	
	91	90	(91 182 364 728 1456 2912 1729 3458 2821 1547 3094 2093)	
	637	90	(637 1274 2548 1001 2002 4004 3913 3731 3367 2639 1183 2366)	
	589	94	(589 1178 2356 617 1234 2468 841 1682 3364 2633 1171 2342)	
	1759	94	(1759 3518 2941 1787 3574 3053 2011 4022 3949 3803 3511 2927)	
	105	104	(105 210 420 840 1680 3360 2625 1155 2310 525 1050 2100)	
	735	104	(735 1470 2940 1785 3570 3045 1995 3990 3885 3675 3255 2415)	
	117	116	(117 234 468 936 1872 3744 3393 2691 1287 2574 1053 2106)	
	351	116	(351 702 1404 2808 1521 3042 1989 3978 3861 3627 3159 2223)	
	975	194	(975 1950 3900 3705 3315 2535)	
	205	196	(205 410 820 1640 3280 2465 835 1670 3340 2585 1075 2150)	
	821	196	(821 1642 3284 2473 851 1702 3404 2713 1331 2662 1229 2458)	
	823	196	(823 1646 3292 2489 883 1766 3532 2969 1843 3686 3277 2459)	
	827	208	(827 1654 3308 2521 947 1894 3788 3481 2867 1639 3278 2461)	
	273	272	(273 546 1092 2184)	
	1911	308	(1911 3822 3549 3003)	
	315	314	(315 630 1260 2520 945 1890 3780 3465 2835 1575 3150 2205)	
	455	454	(455 910 1820 3640 3185 2275)	
	1755	602	(1755 3510 2925)	
	683	462	(683 1366 2732 1369 2738 1381 2762 1429 2858 1621 3242 2389)	
	1367	462	(1367 2734 1373 2746 1397 2794 1493 2986 1877 3754 3413 2731)	
	585	584	(585 1170 2340)	
	819	818	(819 1638 3276 2457)	
	1365	1364	(1365 2730)	
	1	4096	(1 2 4 8 16 32 64 128 256 512 1024 2048)	

Table A.8 Maximum number of solutions to $f(x+a)+f(x)=b$ for power functions, x^d, over the field $\mathbb{F}_{2^{13}}$

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	3	2	(3 6 12 24 48 96 192 384 768 1536 3072 6144 4097)	Gold
	5	2	(5 10 20 40 80 160 320 640 1280 2560 5120 2049 4098)	Gold
	9	2	(9 18 36 72 144 288 576 1152 2304 4608 1025 2050 4100)	Gold
	13	2	(13 26 52 104 208 416 832 1664 3328 6656 5121 2051 4102)	Kasami
	17	2	(17 34 68 136 272 544 1088 2176 4352 513 1026 2052 4101)	Gold
	33	2	(33 66 132 264 528 1056 2112 4224 257 514 1028 2056 4112)	Gold
	57	2	(57 114 228 456 912 1824 3648 7296 6401 4611 1031 2062 4124)	Kasami
	65	2	(65 130 260 520 1040 2080 4160 129 258 516 1032 2064 4128)	Gold
	67	2	(67 134 268 536 1072 2144 4288 385 770 1540 3080 6160 4129)	Welch
	71	2	(71 142 284 568 1136 2272 4544 897 1794 3588 7176 6161 4131)	Niho
	127	2	(127 254 508 1016 2032 4064 8128 8065 7939 7687 7183 6175 4159)	Gold
	171	2	(171 342 684 1368 2736 5472 2753 5506 2821 5642 3093 6186 4181)	Kasami
	191	2	(191 382 764 1528 3056 6112 4033 8066 7941 7691 7191 6191 4191)	Kasami
	241	2	(241 482 964 1928 3856 7712 7233 6275 4359 527 1054 2108 4216)	Kasami
	287	2	(287 574 1148 2296 4592 993 1986 3972 7944 7697 7203 6215 4239)	Kasami
	347	2	(347 694 1388 2776 5552 2913 5826 33461 6922 5653 3115 6230 4269)	Niho
	367	2	(367 734 1468 2936 5872 3553 7106 6021 3851 7702 7213 6235 4279)	Welch
	635	2	(635 1270 2540 5080 1969 3938 7876 7561 6931 5671 3151 6302 4413)	Kasami
	723	2	(723 1446 2892 5784 3377 6754 5317 2443 4886 1581 3162 6324 4457)	Kasami
	911	2	(911 1822 3644 7288 6385 4579 967 1934 3868 7736 7281 6371 4551)	Gold
	1243	2	(1243 2486 4972 1753 3506 7012 5833 3475 6950 5709 3227 6454 4717)	Gold
	1245	2	(1245 2490 4980 1769 3538 7076 5961 3731 7462 6733 5275 2359 4718)	Kasami
	1453	2	(1453 2906 5812 3433 6866 5541 2891 5782 3373 6746 5301 2411 4822)	Gold
	1639	2	(1639 3278 6556 4921 1651 3302 6604 5017 1843 3686 7372 6553 4915)	Gold
	1691	2	(1691 3382 6764 5337 2483 4966 1741 3482 6964 5737 3283 6566 4941)	Kasami
	2731	2	(2731 5462 2733 5466 2741 5482 2773 5546 2901 5802 3413 6826 5461)	Gold
	4095	2	(4095 8190 8189 8187 8183 8175 8159 8127 8063 7935 7679 7167 6143)	Field Inverse
	303	4	(303 606 1212 2424 4848 1505 3010 6020 3849 7698 7205 6219 4247)	
	947	4	(947 1894 3788 7576 6961 5731 3271 6542 4893 1595 3190 6380 4569)	
	7	6	(7 14 28 56 112 224 448 896 1792 3584 7168 6145 4099)	
	15	6	(15 30 60 120 240 480 960 1920 3840 7680 7169 6147 4103)	
	21	6	(21 42 84 168 336 672 1344 2688 5376 2561 5122 2053 4106)	
	23	6	(23 46 92 184 368 736 1472 2944 5888 3585 7170 6149 4107)	
	31	6	(31 62 124 248 496 992 1984 3968 7936 7681 7171 6151 4111)	
	35	6	(35 70 140 280 560 1120 2240 4480 769 1538 3076 6152 4113)	
	53	6	(53 106 212 424 848 1696 3392 6784 5377 2563 5126 2061 4122)	
	61	6	(61 122 244 488 976 1952 3904 7808 7425 6659 5127 2063 4126)	
	63	6	(63 126 252 504 1008 2016 4032 8064 7937 7683 7175 6159 4127)	
	73	6	(73 146 292 584 1168 2336 4672 1153 2306 4612 1033 2066 4132)	
	75	6	(75 150 300 600 1200 2400 4800 1409 2818 5636 3081 6162 4133)	
	81	6	(81 162 324 648 1296 2592 5184 2177 4354 517 1034 2068 4136)	
	87	6	(87 174 348 696 1392 2784 5568 2945 5890 3589 7178 6165 4139)	
	93	6	(93 186 372 744 1488 2976 5952 3713 7426 6661 5131 2071 4142)	
	101	6	(101 202 404 808 1616 3232 6464 4737 1283 2566 5132 2073 4146)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	107	6	(107 214 428 856 1712 3424 6848 5505 2819 5638 3085 6170 4149)	
	111	6	(111 222 444 888 1776 3552 7104 6017 3843 7686 7181 6171 4151)	
	117	6	(117 234 468 936 1872 3744 7488 6785 5379 2567 5134 2077 4154)	
	121	6	(121 242 484 968 1936 3872 7744 7297 6403 4615 1039 2078 4156)	
	125	6	(125 250 500 1000 2000 4000 8000 7809 7427 6663 5135 2079 4158)	
	131	6	(131 262 524 1048 2096 4192 193 386 772 1544 3088 6176 4161)	
	133	6	(133 266 532 1064 2128 4256 321 642 1284 2568 5136 2081 4162)	
	137	6	(137 274 548 1096 2192 4384 577 1154 2308 4616 1041 2082 4164)	
	147	6	(147 294 588 1176 2352 4704 1217 2434 4868 1545 3090 6180 4169)	
	151	6	(151 302 604 1208 2416 4832 1473 2946 5892 3593 7186 6181 4171)	
	155	6	(155 310 620 1240 2480 4960 1729 3458 6916 5641 3091 6182 4173)	
	163	6	(163 326 652 1304 2608 5216 2241 4482 773 1546 3092 6184 4177)	
	167	6	(167 334 668 1336 2672 5344 2497 4994 1797 3594 7188 6185 4179)	
	181	6	(181 362 724 1448 2896 5792 3393 6786 5381 2571 5142 2093 4186)	
	183	6	(183 366 732 1464 2928 5856 3521 7042 5893 3595 7190 6189 4187)	
	185	6	(185 370 740 1480 2960 5920 3649 7298 6405 4619 1047 2094 4188)	
	189	6	(189 378 756 1512 3024 6048 3905 7810 7429 6667 5143 2095 4190)	
	197	6	(197 394 788 1576 3152 6304 4417 643 1286 2572 5144 2097 4194)	
	203	6	(203 406 812 1624 3248 6496 4801 1411 2822 5644 3097 6194 4197)	
	205	6	(205 410 820 1640 3280 6560 4929 1667 3334 6668 5145 2099 4198)	
	207	6	(207 414 828 1656 3312 6624 5057 1923 3846 7692 7193 6195 4199)	
	211	6	(211 422 844 1688 3376 6752 5313 2435 4870 1549 3098 6196 4201)	
	217	6	(217 434 868 1736 3472 6944 5697 3203 6406 4621 1051 2102 4204)	
	221	6	(221 442 884 1768 3536 7072 5953 3715 7430 6669 5147 2103 4206)	
	225	6	(225 450 900 1800 3600 7200 6209 4227 263 526 1052 2104 4208)	
	227	6	(227 454 908 1816 3632 7264 6337 4483 775 1550 3100 6200 4209)	
	229	6	(229 458 916 1832 3664 7328 6465 4739 1287 2574 5148 2105 4210)	
	231	6	(231 462 924 1848 3696 7392 6593 4995 1799 3598 7196 6201 4211)	
	233	6	(233 466 932 1864 3728 7456 6721 5251 2311 4622 1053 2106 4212)	
	235	6	(235 470 940 1880 3760 7520 6849 5507 2823 5646 3101 6202 4213)	
	239	6	(239 478 956 1912 3824 7648 7105 6019 3847 7694 7197 6203 4215)	
	243	6	(243 486 972 1944 3888 7776 7361 6531 4871 1551 3102 6204 4217)	
	255	6	(255 510 1020 2040 4080 8160 8129 8067 7943 7695 7199 6207 4223)	
	269	6	(269 538 1076 2152 4304 417 834 1668 3336 6672 5153 2115 4230)	
	271	6	(271 542 1084 2168 4336 481 962 1924 3848 7696 7201 6211 4231)	
	273	6	(273 546 1092 2184 4368 545 1090 2180 4360 529 1058 2116 4232)	
	275	6	(275 550 1100 2200 4400 609 1218 2436 4872 1553 3106 6212 4233)	
	277	6	(277 554 1108 2216 4432 673 1346 2692 5384 2577 5154 2117 4234)	
	281	6	(281 562 1124 2248 4496 801 1602 3204 6408 4625 1059 2118 4236)	
	285	6	(285 570 1140 2280 4560 929 1858 3716 7432 6673 5155 2119 4238)	
	291	6	(291 582 1164 2328 4656 1121 2242 4484 777 1554 3108 6216 4241)	
	295	6	(295 590 1180 2360 4720 1249 2498 4996 1801 3602 7204 6217 4243)	
	299	6	(299 598 1196 2392 4784 1377 2754 5508 2825 5650 3109 6218 4245)	
	301	6	(301 602 1204 2408 4816 1441 2882 5764 3337 6674 5157 2123 4246)	
	305	6	(305 610 1220 2440 4880 1569 3138 6276 4361 531 1062 2124 4248)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	307	6	(307 614 1228 2456 4912 1633 3266 6532 4873 1555 3110 6220 4249)	
	309	6	(309 618 1236 2472 4944 1697 3394 6788 5385 2579 5158 2125 4250)	
	313	6	(313 626 1252 2504 5008 1825 3650 7300 6409 4627 1063 2126 4252)	
	327	6	(327 654 1308 2616 5232 2273 4546 901 1802 3604 7208 6225 4259)	
	331	6	(331 662 1324 2648 5296 2401 4802 1413 2826 5652 3113 6226 4261)	
	335	6	(335 670 1340 2680 5360 2529 5058 1925 3850 7700 7209 6227 4263)	
	339	6	(339 678 1356 2712 5424 2657 5314 2437 4874 1557 3114 6228 4265)	
	355	6	(355 710 1420 2840 5680 3169 6338 4485 779 1558 3116 6232 4273)	
	357	6	(357 714 1428 2856 5712 3233 6466 4741 1291 2582 5164 2137 4274)	
	365	6	(365 730 1460 2920 5840 3489 6978 5765 3339 6678 5165 2139 4278)	
	369	6	(369 738 1476 2952 5904 3617 7234 6277 4363 535 1070 2140 4280)	
	395	6	(395 790 1580 3160 6320 4449 707 1414 2828 5656 3121 6242 4293)	
	397	6	(397 794 1588 3176 6352 4513 835 1670 3340 6680 5169 2147 4294)	
	401	6	(401 802 1604 3208 6416 4641 1091 2182 4364 537 1074 2148 4296)	
	405	6	(405 810 1620 3240 6480 4769 1347 2694 5388 2585 5170 2149 4298)	
	421	6	(421 842 1684 3368 6736 5281 2371 4742 1293 2586 5172 2153 4306)	
	433	6	(433 866 1732 3464 6928 5665 3139 6278 4365 539 1078 2156 4312)	
	439	6	(439 878 1756 3512 7024 5857 3523 7046 5901 3611 7222 6253 4315)	
	441	6	(441 882 1764 3528 7056 5921 3651 7302 6413 4635 1079 2158 4316)	
	453	6	(453 906 1812 3624 7248 6305 4419 647 1294 2588 5176 2161 4322)	
	461	6	(461 922 1844 3688 7376 6561 4931 1671 3342 6684 5177 2163 4326)	
	465	6	(465 930 1860 3720 7440 6689 5187 2183 4366 541 1082 2164 4328)	
	467	6	(467 934 1868 3736 7472 6753 6315 2439 4878 1565 3130 6260 4329)	
	477	6	(477 954 1908 3816 7632 7073 5955 3719 7438 6685 5179 2167 4334)	
	491	6	(491 982 1964 3928 7856 7521 6851 5511 2831 5662 3133 6266 4341)	
	495	6	(495 990 1980 3960 7920 7649 7107 6023 3855 7710 7229 6267 4343)	
	499	6	(499 998 1996 3992 7984 7777 7363 6535 4879 1567 3134 6268 4345)	
	501	6	(501 1002 2004 4008 8016 7841 7491 6791 5391 2591 5182 2173 4346)	
	503	6	(503 1006 2012 4024 8048 7905 7619 7047 5903 3615 7230 6269 4347)	
	505	6	(505 1010 2020 4040 8080 7969 7747 7303 6415 4639 1087 2174 4348)	
	509	6	(509 1018 2036 4072 8144 8097 8003 7815 7439 6687 5183 2175 4350)	
	511	6	(511 1022 2044 4088 8176 8161 8131 8071 7951 7711 7231 6271 4351)	
	547	6	(547 1094 2188 4376 561 1122 2244 4488 785 1570 3140 6280 4369)	
	549	6	(549 1098 2196 4392 593 1186 2372 4744 1297 2594 5188 2185 4370)	
	553	6	(553 1106 2212 4424 657 1314 2628 5256 2321 4642 1093 2186 4372)	
	555	6	(555 1110 2220 4440 689 1378 2756 5512 2833 5666 3141 6282 4373)	
	557	6	(557 1114 2228 4456 721 1442 2884 5768 3345 6690 5189 2187 4374)	
	567	6	(567 1134 2268 4536 881 1762 3524 7048 5905 3619 7238 6285 4379)	
	575	6	(575 1150 2300 4600 1009 2018 4036 8072 7953 7715 7239 6287 4383)	
	581	6	(581 1162 2324 4648 1105 2210 4420 649 1298 2596 5192 2193 4386)	
	585	6	(585 1170 2340 4680 1169 2338 4676 1161 2322 4644 1097 2194 4388)	
	595	6	(595 1190 2380 4760 1329 2658 5316 2441 4882 1573 3146 6292 4393)	
	603	6	(603 1206 2412 4824 1457 2914 5828 3465 6930 5669 3147 6294 4397)	
	611	6	(611 1222 2444 4888 1585 3170 6340 4489 787 1574 3148 6296 4401)	
	613	6	(613 1226 2452 4904 1617 3234 6468 4745 1299 2598 5196 2201 4402)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	617	6	(617 1234 2468 4936 1681 3362 6724 5257 2323 4646 1101 2202 4404)	
	623	6	(623 1246 2492 4984 1777 3554 7108 6025 3859 7718 7245 6299 4407)	
	627	6	(627 1254 2508 5016 1841 3682 7364 6537 4883 1575 3150 6300 4409)	
	631	6	(631 1262 2524 5048 1905 3810 7620 7049 5907 3623 7246 6301 4411)	
	633	6	(633 1266 2532 5064 1937 3874 7748 7305 6419 4647 1103 2206 4412)	
	639	6	(639 1278 2556 5112 2033 4066 8132 8073 7955 7719 7247 6303 4415)	
	653	6	(653 1306 2612 5224 2257 4514 837 1674 3348 6696 5201 2211 4422)	
	655	6	(655 1310 2620 5240 2289 4578 965 1930 3860 7720 7249 6307 4423)	
	663	6	(663 1326 2652 5304 2417 4834 1477 2954 5908 3625 7250 6309 4427)	
	667	6	(667 1334 2668 5336 2481 4962 1733 3466 6932 5673 3155 6310 4429)	
	669	6	(669 1338 2676 5352 2513 5026 1861 3722 7444 6697 5203 2215 4430)	
	671	6	(671 1342 2684 5368 2545 5090 1989 3978 7956 7721 7251 6311 4431)	
	679	6	(679 1358 2716 5432 2673 5346 2501 5002 1813 3626 7252 6313 4435)	
	681	6	(681 1362 2724 5448 2705 5410 2629 5258 2325 4650 1109 2218 4436)	
	683	6	(683 1366 2732 5464 2737 5474 2757 5514 2837 5674 3157 6314 4437)	
	687	6	(687 1374 2748 5496 2801 5602 3013 6026 3861 7722 7253 6315 4439)	
	693	6	(693 1386 2772 5544 2897 5794 3397 6794 5397 2603 5206 2221 4442)	
	703	6	(703 1406 2812 5624 3057 6114 4037 8074 7957 7723 7255 6319 4447)	
	711	6	(711 1422 2844 5688 3185 6370 4549 907 1814 3628 7256 6321 4451)	
	713	6	(713 1426 2852 5704 3217 6434 4677 1163 2326 4652 1113 2226 4452)	
	715	6	(715 1430 2860 5720 3249 6498 4805 1419 2838 5676 3161 6322 4453)	
	717	6	(717 1434 2868 5736 3281 6562 4933 1675 3350 6700 5209 2227 4454)	
	719	6	(719 1438 2876 5752 3313 6626 5061 1931 3862 7724 7257 6323 4455)	
	725	6	(725 1450 2900 5800 3409 6818 5445 2699 5398 2605 5210 2229 4458)	
	729	6	(729 1458 2916 5832 3473 6946 5701 3211 6422 4653 1115 2230 4460)	
	739	6	(739 1478 2956 5912 3633 7266 6341 4491 791 1582 3164 6328 4465)	
	743	6	(743 1486 2972 5944 3697 7394 6597 5003 1815 3630 7260 6329 4467)	
	747	6	(747 1494 2988 5976 3761 7522 6853 5515 2839 5678 3165 6330 4469)	
	749	6	(749 1498 2996 5992 3793 7586 6981 5771 3351 6702 5213 2235 4470)	
	755	6	(755 1510 3020 6040 3889 7778 7365 6539 4887 1583 3166 6332 4473)	
	761	6	(761 1522 3044 6088 3985 7970 7749 7307 6423 4655 1119 2238 4476)	
	763	6	(763 1526 3052 6104 4017 8034 7877 7563 6935 5679 3167 6334 4477)	
	765	6	(765 1530 3060 6120 4049 8098 8005 7819 7447 6703 5215 2239 4478)	
	767	6	(767 1534 3068 6136 4081 8162 8133 8075 7959 7727 7263 6335 4479)	
	797	6	(797 1594 3188 6376 4561 931 1862 3724 7448 6705 5219 2247 4494)	
	807	6	(807 1614 3228 6456 4721 1251 2502 5004 1817 3634 7268 6345 4499)	
	809	6	(809 1618 3236 6472 4753 1315 2630 5260 2329 4658 1125 2250 4500)	
	811	6	(811 1622 3244 6488 4785 1379 2758 5516 2841 5682 3173 6346 4501)	
	819	6	(819 1638 3276 6552 4913 1635 3270 6540 4889 1587 3174 6348 4505)	
	821	6	(821 1642 3284 6568 4945 1699 3398 6796 5401 2611 5222 2253 4506)	
	829	6	(829 1658 3316 6632 5073 1955 3910 7820 7449 6707 5223 2255 4510)	
	839	6	(839 1678 3356 6712 5233 2275 4550 909 1818 3636 7272 6353 4515)	
	841	6	(841 1682 3364 6728 5265 2339 4678 1165 2330 4660 1129 2258 4516)	
	855	6	(855 1710 3420 6840 5489 2787 5574 2957 5914 3637 7274 6357 4523)	
	857	6	(857 1714 3428 6856 5521 2851 5702 3213 6426 4661 1131 2262 4524)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	861	6	(861 1722 3444 6888 5585 2979 5958 3725 7450 6709 5227 2263 4526)	
	871	6	(871 1742 3484 6968 5745 3299 6598 5005 1819 3638 7276 6361 4531)	
	873	6	(873 1746 3492 6984 5777 3363 6726 5261 2331 4662 1133 2266 4532)	
	875	6	(875 1750 3500 7000 5809 3427 6854 5517 2843 5686 3181 6362 4533)	
	877	6	(877 1754 3508 7016 5841 3491 6982 5773 3355 6710 5229 2267 4534)	
	879	6	(879 1758 3516 7032 5873 3555 7110 6029 3867 7734 7277 6363 4535)	
	883	6	(883 1766 3532 7064 5937 3683 7366 6541 4891 1591 3182 6364 4537)	
	915	6	(915 1830 3660 7320 6449 4707 1223 2446 4892 1593 3186 6372 4553)	
	919	6	(919 1838 3676 7352 6513 4835 1479 2958 5916 3641 7282 6373 4555)	
	927	6	(927 1854 3708 7416 6641 5091 1991 3982 7964 7737 7283 6375 4559)	
	935	6	(935 1870 3740 7480 6769 5347 2503 5006 1821 3642 7284 6377 4563)	
	941	6	(941 1882 3764 7528 6865 5539 2887 5774 3357 6714 5237 2283 4566)	
	949	6	(949 1898 3796 7592 6993 5795 3399 6798 5405 2619 5238 2285 4570)	
	951	6	(951 1902 3804 7608 7025 5859 3527 7054 5917 3643 7286 6381 4571)	
	957	6	(957 1914 3828 7656 7121 6051 3911 7822 7453 6715 5239 2287 4574)	
	959	6	(959 1918 3836 7672 7153 6115 4039 8078 7965 7739 7287 6383 4575)	
	969	6	(969 1938 3876 7752 7313 6435 4679 1167 2334 4668 1145 2290 4580)	
	1001	6	(1001 2002 4004 8008 7825 7459 6727 5263 2335 4670 1149 2298 4596)	
	1005	6	(1005 2010 4020 8040 7889 7587 6983 5775 3359 6718 5245 2299 4598)	
	1011	6	(1011 2022 4044 8088 7985 7779 7367 6543 4895 1599 3198 6396 4601)	
	1015	6	(1015 2030 4060 8120 8049 7907 7623 7055 5919 3647 7294 6397 4603)	
	1017	6	(1017 2034 4068 8136 8081 7971 7751 7311 6431 4671 1151 2302 4604)	
	1023	6	(1023 2046 4092 8184 8177 8163 8135 8079 7967 7743 7295 6399 4607)	
	1171	6	(1171 2342 4684 1177 2354 4708 1225 2450 4900 1609 3218 6436 4681)	
	1173	6	(1173 2346 4692 1193 2386 4772 1353 2706 5412 2633 5266 2341 4682)	
	1175	6	(1175 2350 4700 1209 2418 4836 1481 2962 5924 3657 7314 6437 4683)	
	1189	6	(1189 2378 4756 1321 2642 5284 2377 4754 1317 2634 5268 2345 4690)	
	1195	6	(1195 2390 4780 1369 2738 5476 2761 5522 2853 5706 3221 6442 4693)	
	1197	6	(1197 2394 4788 1385 2770 5540 2889 5778 3365 6730 5269 2347 4694)	
	1205	6	(1205 2410 4820 1449 2898 5796 3401 6802 5413 2635 5270 2349 4698)	
	1211	6	(1211 2422 4844 1497 2994 5988 3785 7570 6949 5707 3223 6446 4701)	
	1215	6	(1215 2430 4860 1529 3058 6116 4041 8082 7973 7755 7319 6447 4703)	
	1229	6	(1229 2458 4916 1641 3282 6564 4937 1683 3366 6732 5273 2355 4710)	
	1239	6	(1239 2478 4956 1721 3442 6884 5577 2963 5926 3661 7322 6453 4715)	
	1247	6	(1247 2494 4988 1785 3570 7140 6089 3987 7974 7757 7323 6455 4719)	
	1253	6	(1253 2506 5012 1833 3666 7332 6473 4755 1319 2638 5276 2361 4722)	
	1255	6	(1255 2510 5020 1849 3698 7396 6601 5011 1831 3662 7324 6457 4723)	
	1261	6	(1261 2522 5044 1897 3794 7588 6985 5779 3367 6734 5277 2363 4726)	
	1263	6	(1263 2526 5052 1913 3826 7652 7113 6035 3879 7758 7325 6459 4727)	
	1269	6	(1269 2538 5076 1961 3922 7844 7497 6803 5415 2639 5278 2365 4730)	
	1271	6	(1271 2542 5084 1977 3954 7908 7625 7059 5927 3663 7326 6461 4731)	
	1279	6	(1279 2558 5116 2041 4082 8164 8137 8083 7975 7759 7327 6463 4735)	
	1323	6	(1323 2646 5292 2393 4786 1381 2762 5524 2857 5714 3237 6474 4757)	
	1327	6	(1327 2654 5308 2425 4850 1509 3018 6036 3881 7762 7333 6475 4759)	
	1335	6	(1335 2670 5340 2489 4978 1765 3530 7060 5929 3667 7334 6477 4763)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	1341	6	(1341 2682 5364 2537 5074 1957 3914 7828 7465 6739 5287 2383 4766)	
	1355	6	(1355 2710 5420 2649 5298 2405 4810 1429 2858 5716 3241 6482 4773)	
	1357	6	(1357 2714 5428 2665 5330 2469 4938 1685 3370 6740 5289 2387 4774)	
	1359	6	(1359 2718 5436 2681 5362 2533 5066 1941 3882 7764 7337 6483 4775)	
	1365	6	(1365 2730 5460 2729 5458 2725 5450 2709 5418 2645 5290 2389 4778)	
	1367	6	(1367 2734 5468 2745 5490 2789 5578 2965 5930 3669 7338 6485 4779)	
	1373	6	(1373 2746 5492 2793 5586 2981 5962 3733 7466 6741 5291 2391 4782)	
	1387	6	(1387 2774 5548 2905 5810 3429 6858 5525 2859 5718 3245 6490 4789)	
	1389	6	(1389 2778 5556 2921 5842 3493 6986 5781 3371 6742 5293 2395 4790)	
	1391	6	(1391 2782 5564 2937 5874 3557 7114 6037 3883 7766 7341 6491 4791)	
	1399	6	(1399 2798 5596 3001 6002 3813 7626 7061 5931 3671 7342 6493 4795)	
	1405	6	(1405 2810 5620 3049 6098 4005 8010 7829 7467 6743 5295 2399 4798)	
	1435	6	(1435 2870 5740 3289 6578 4965 1739 3478 6956 5721 3251 6502 4813)	
	1455	6	(1455 2910 5820 3449 6898 5605 3019 6038 3885 7770 7349 6507 4823)	
	1463	6	(1463 2926 5852 3513 7026 5861 3531 7062 5933 3675 7350 6509 4827)	
	1493	6	(1493 2986 5972 3753 7506 6821 5451 2711 5422 2653 5306 2421 4842)	
	1519	6	(1519 3038 6076 3961 7922 7653 7115 6039 3887 7774 7357 6523 4855)	
	1523	6	(1523 3046 6092 3993 7986 7781 7371 6551 4911 1631 3262 6524 4857)	
	1525	6	(1525 3050 6100 4009 8018 7845 7499 6807 5423 2655 5310 2429 4858)	
	1527	6	(1527 3054 6108 4025 8050 7909 7627 7063 5935 3679 7358 6525 4859)	
	1643	6	(1643 3286 6572 4953 1715 3430 6860 5529 2867 5734 3277 6554 4917)	
	1647	6	(1647 3294 6588 4985 1779 3558 7116 6041 3891 7782 7373 6555 4919)	
	1653	6	(1653 3306 6612 5033 1875 3750 7500 6809 5427 2663 5326 2461 4922)	
	1661	6	(1661 3322 6644 5097 2003 4006 8012 7833 7475 6759 5327 2463 4926)	
	1707	6	(1707 3414 6828 5465 2739 5478 2765 5530 2869 5738 3285 6570 4949)	
	1709	6	(1709 3418 6836 5481 2771 5542 2893 5786 3381 6762 5333 2475 4950)	
	1719	6	(1719 3438 6876 5561 2931 5862 3533 7066 5941 3691 7382 6573 4955)	
	1751	6	(1751 3502 7004 5817 3443 6886 5581 2971 5942 3693 7386 6581 4971)	
	1759	6	(1759 3518 7036 5881 3571 7142 6093 3995 7990 7789 7387 6583 4975)	
	1771	6	(1771 3542 7084 5977 3763 7526 6861 5531 2871 5742 3293 6586 4981)	
	1773	6	(1773 3546 7092 5993 3795 7590 6989 5787 3383 6766 5341 2491 4982)	
	1775	6	(1775 3550 7100 6009 3827 7654 7117 6043 3895 7790 7389 6587 4983)	
	1781	6	(1781 3562 7124 6057 3923 7846 7501 6811 5431 2671 5342 2493 4986)	
	1789	6	(1789 3578 7156 6121 4051 8102 8013 7835 7479 6767 5343 2495 4990)	
	1791	6	(1791 3582 7164 6137 4083 8166 8141 8091 7991 7791 7391 6591 4991)	
	1883	6	(1883 3766 7532 6873 5555 2919 5838 3485 6970 5749 3307 6614 5037)	
	1899	6	(1899 3798 7596 7001 5811 3431 6862 5533 2875 5750 3309 6618 5045)	
	1911	6	(1911 3822 7644 7097 6003 3815 7630 7069 5947 3703 7406 6621 5051)	
	1915	6	(1915 3830 7660 7129 6067 3943 7886 7581 6971 5751 3311 6622 5053)	
	1917	6	(1917 3834 7668 7145 6099 4007 8014 7837 7483 6775 5359 2527 5054)	
	1919	6	(1919 3838 7676 7161 6131 4071 8142 8093 7995 7799 7407 6623 5055)	
	1951	6	(1951 3902 7804 7417 6643 5095 1999 3998 7996 7801 7411 6631 5071)	
	1973	6	(1973 3946 7892 7593 6995 5799 3407 6814 5437 2683 5366 2541 5082)	
	1979	6	(1979 3958 7916 7641 7091 5991 3791 7582 6973 5755 3319 6638 5085)	
	1981	6	(1981 3962 7924 7657 7123 6055 3919 7838 7485 6779 5367 2543 5086)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	2013	6	(2013 4026 8052 7913 7635 7079 5967 3743 7486 6781 5371 2551 5102)	
	2027	6	(2027 4054 8108 8025 7859 7527 6863 5535 2879 5758 3325 6650 5109)	
	2037	6	(2037 4074 8148 8105 8019 7847 7503 6815 5439 2687 5374 2557 5114)	
	2039	6	(2039 4078 8156 8121 8051 7911 7631 7071 5951 3711 7422 6653 5115)	
	2047	6	(2047 4094 8188 8185 8179 8167 8143 8095 7999 7807 7423 6655 5119)	
	2735	6	(2735 5470 2749 5498 2805 5610 3029 6058 3925 7850 7509 6827 5463)	
	2743	6	(2743 5486 2781 5562 2933 5866 3541 7082 5973 3755 7510 6829 5467)	
	2775	6	(2775 5550 2909 5818 3445 6890 5589 2987 5974 3757 7514 6837 5483)	
	2795	6	(2795 5590 2989 5978 3765 7530 6869 5547 2903 5806 3421 6842 5493)	
	2799	6	(2799 5598 3005 6010 3829 7658 7125 6059 3927 7854 7517 6843 5495)	
	2807	6	(2807 5614 3037 6074 3957 7914 7637 7083 5975 3759 7518 6845 5499)	
	2811	6	(2811 5622 3053 6106 4021 8042 7893 7595 6999 5807 3423 6846 5501)	
	2907	6	(2907 5814 3437 6874 5557 2923 5846 3501 7002 5813 3435 6870 5549)	
	2939	6	(2939 5878 3565 7130 6069 3947 7894 7597 7003 5815 3439 6878 5565)	
	2991	6	(2991 5982 3773 7546 6901 5611 3031 6062 3933 7866 7541 6891 5591)	
	2999	6	(2999 5998 3805 7610 7029 5867 3543 7086 5981 3771 7542 6893 5595)	
	3007	6	(3007 6014 3837 7674 7157 6123 4055 8110 8029 7867 7543 6895 5599)	
	3067	6	(3067 6134 4077 8154 8117 8043 7895 7599 7007 5823 3455 6910 5629)	
	3071	6	(3071 6142 4093 8186 8181 8171 8151 8111 8031 7871 7551 6911 5631)	
	3511	6	(3511 7022 5853 3515 7030 5869 3547 7094 5997 3803 7606 7021 5851)	
	3519	6	(3519 7038 5885 3579 7158 6125 4059 8118 8045 7899 7607 7023 5855)	
	3551	6	(3551 7102 6013 3835 7670 7149 6107 4023 8046 7901 7611 7031 5871)	
	3575	6	(3575 7150 6109 4027 8054 7917 7643 7095 5999 3807 7614 7037 5883)	
	3823	6	(3823 7646 7101 6011 3831 7662 7133 6075 3959 7918 7645 7099 6007)	
	4031	6	(4031 8062 7933 7675 7159 6127 4063 8126 8061 7931 7671 7151 6111)	
	11	8	(11 22 44 88 176 352 704 1408 2816 5632 3073 6146 4101)	
	19	8	(19 38 76 152 304 608 1216 2432 4864 1537 3074 6148 4105)	
	27	8	(27 54 108 216 432 864 1728 3456 6912 5633 3075 6150 4109)	
	29	8	(29 58 116 232 464 928 1856 3712 7424 6657 5123 2055 4110)	
	39	8	(39 78 156 312 624 1248 2496 4992 1793 3586 7172 6153 4115)	
	41	8	(41 82 164 328 656 1312 2624 5248 2305 4610 1029 2058 4116)	
	55	8	(55 110 220 440 880 1760 3520 7040 5889 3587 7174 6157 4123)	
	69	8	(69 138 276 552 1104 2208 4416 641 1282 2564 5128 2065 4130)	
	77	8	(77 154 308 616 1232 2464 4928 1665 3330 6660 5129 2067 4134)	
	79	8	(79 158 316 632 1264 2528 5056 1921 3842 7684 7177 6163 4135)	
	83	8	(83 166 332 664 1328 2656 5312 2433 4866 1541 3082 6164 4137)	
	85	8	(85 170 340 680 1360 2720 5440 2689 5378 2565 5130 2069 4138)	
	89	8	(89 178 356 712 1424 2848 5696 3201 6402 4613 1035 2070 4140)	
	91	8	(91 182 364 728 1456 2912 5824 3457 6914 5637 3083 6166 4141)	
	95	8	(95 190 380 760 1520 3040 6080 3969 7938 7685 7179 6167 4143)	
	97	8	(97 194 388 776 1552 3104 6208 4225 259 518 1036 2072 4144)	
	99	8	(99 198 396 792 1584 3168 6336 4481 771 1542 3084 6168 4145)	
	109	8	(109 218 436 872 1744 3488 6976 5761 3331 6662 5133 2075 4150)	
	115	8	(115 230 460 920 1840 3680 7360 6529 4867 1543 3086 6172 4153)	
	119	8	(119 238 476 952 1904 3808 7616 7041 5891 3591 7182 6173 4155)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	123	8	(123 246 492 984 1968 3936 7872 7553 6915 5639 3087 6174 4157)	
	135	8	(135 270 540 1080 2160 4320 449 898 1796 3592 7184 6177 4163)	
	139	8	(139 278 556 1112 2224 4448 705 1410 2820 5640 3089 6178 4165)	
	143	8	(143 286 572 1144 2288 4576 961 1922 3844 7688 7185 6179 4167)	
	145	8	(145 290 580 1160 2320 4640 1089 2178 4356 521 1042 2084 4168)	
	149	8	(149 298 596 1192 2384 4768 1345 2690 5380 2569 5138 2085 4170)	
	157	8	(157 314 628 1256 2512 5024 1857 3714 7428 6665 5139 2087 4174)	
	161	8	(161 322 644 1288 2576 5152 2113 4226 261 522 1044 2088 4176)	
	165	8	(165 330 660 1320 2640 5280 2369 4738 1285 2570 5140 2089 4178)	
	173	8	(173 346 692 1384 2768 5536 2881 5762 3333 6666 5141 2091 4182)	
	175	8	(175 350 700 1400 2800 5600 3009 6018 3845 7690 7189 6187 4183)	
	179	8	(179 358 716 1432 2864 5728 3265 6530 4869 1547 3094 6188 4185)	
	187	8	(187 374 748 1496 2992 5984 3777 7554 6917 5643 3095 6196 4189)	
	195	8	(195 390 780 1560 3120 6240 4289 387 774 1548 3096 6192 4193)	
	199	8	(199 398 796 1592 3184 6368 4545 899 1798 3596 7192 6193 4195)	
	201	8	(201 402 804 1608 3216 6432 4673 1155 2310 4620 1049 2098 4196)	
	209	8	(209 418 836 1672 3344 6688 5185 2179 4358 525 1050 2100 4200)	
	215	8	(215 430 860 1720 3440 6880 5569 2947 5894 3597 7194 6197 4203)	
	219	8	(219 438 876 1752 3504 7008 5825 3459 6918 5645 3099 6198 4205)	
	223	8	(223 446 892 1784 3568 7136 6081 3971 7942 7693 7195 6199 4207)	
	247	8	(247 494 988 1976 3952 7904 7617 7043 5895 3599 7198 6205 4219)	
	249	8	(249 498 996 1992 3984 7968 7745 7299 6407 4623 1055 2110 4220)	
	251	8	(251 502 1004 2008 4016 8032 7873 7555 6919 5647 3103 6206 4221)	
	253	8	(253 506 1012 2024 4048 8096 8001 7811 7431 6671 5151 2111 4222)	
	267	8	(267 534 1068 2136 4272 353 706 1412 2824 5648 3105 6210 4229)	
	279	8	(279 558 1116 2232 4464 737 1474 2948 5896 3601 7202 6213 4235)	
	283	8	(283 566 1132 2264 4528 865 1730 3460 6920 5649 3107 6214 4237)	
	293	8	(293 586 1172 2344 4688 1185 2370 4740 1289 2578 5156 2121 4242)	
	311	8	(311 622 1244 2488 4976 1761 3522 7044 5897 3603 7206 6221 4251)	
	315	8	(315 630 1260 2520 5040 1889 3778 7556 6921 5651 3111 6222 4253)	
	317	8	(317 634 1268 2536 5072 1953 3906 7812 7433 6675 5159 2127 4254)	
	319	8	(319 638 1276 2552 5104 2017 4034 8068 7945 7699 7207 6223 4255)	
	323	8	(323 646 1292 2584 5168 2145 4290 389 778 1556 3112 6224 4257)	
	325	8	(325 650 1300 2600 5200 2209 4418 645 1290 2580 5160 2129 4258)	
	329	8	(329 658 1316 2632 5264 2337 4674 1157 2314 4628 1065 2130 4260)	
	333	8	(333 666 1332 2664 5328 2465 4930 1669 3338 6676 5161 2131 4262)	
	337	8	(337 674 1348 2696 5392 2593 5186 2181 4362 533 1066 2132 4264)	
	341	8	(341 682 1364 2728 5456 2721 5442 2693 5386 2581 5162 2133 4266)	
	343	8	(343 686 1372 2744 5488 2785 5570 2949 5898 3605 7210 6229 4267)	
	345	8	(345 690 1380 2760 5520 2849 5698 3205 6410 4629 1067 2134 4268)	
	349	8	(349 698 1396 2792 5584 2977 5954 3717 7434 6677 5163 2135 4270)	
	359	8	(359 718 1436 2872 5744 3297 6594 4997 1803 3606 7212 6233 4275)	
	361	8	(361 722 1444 2888 5776 3361 6722 5253 2315 4630 1069 2138 4276)	
	371	8	(371 742 1484 2968 5936 3681 7362 6533 4875 1559 3118 6236 4281)	
	391	8	(391 782 1564 3128 6256 4321 451 902 1804 3608 7216 6241 4291)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	393	8	(393 786 1572 3144 6288 4385 579 1158 2316 4632 1073 2146 4292)	
	399	8	(399 798 1596 3192 6384 4577 963 1926 3852 7704 7217 6243 4295)	
	403	8	(403 806 1612 3224 6448 4705 1219 2438 4876 1561 3122 6244 4297)	
	407	8	(407 814 1628 3256 6512 4833 1475 2950 5900 3609 7218 6245 4299)	
	409	8	(409 818 1636 3272 6544 4897 1603 3206 6412 4633 1075 2150 4300)	
	411	8	(411 822 1644 3288 6576 4961 1731 3462 6924 5657 3123 6246 4301)	
	413	8	(413 826 1652 3304 6608 5025 1859 3718 7436 6681 5171 2151 4302)	
	415	8	(415 830 1660 3320 6640 5089 1987 3974 7948 7705 7219 6247 4303)	
	425	8	(425 850 1700 3400 6800 5409 2627 5254 2317 4634 1077 2154 4308)	
	427	8	(427 854 1708 3416 6832 5473 2755 5510 2829 5658 3125 6250 4309)	
	431	8	(431 862 1724 3448 6896 5601 3011 6022 3853 7706 7221 6251 4311)	
	435	8	(435 870 1740 3480 6960 5729 3267 6534 4877 1563 3126 6252 4313)	
	443	8	(443 886 1772 3544 7088 5985 3779 7558 6925 5659 3127 6254 4317)	
	445	8	(445 890 1780 3560 7120 6049 3907 7814 7437 6683 5175 2159 4318)	
	447	8	(447 894 1788 3576 7152 6113 4035 8070 7949 7707 7223 6255 4319)	
	455	8	(455 910 1820 3640 7280 6369 4547 903 1806 3612 7224 6257 4323)	
	459	8	(459 918 1836 3672 7344 6497 4803 1415 2830 5660 3129 6258 4325)	
	463	8	(463 926 1852 3704 7408 6625 5059 1927 3854 7708 7225 6259 4327)	
	473	8	(473 946 1892 3784 7568 6945 5699 3207 6414 4637 1083 2166 4332)	
	475	8	(475 950 1900 3800 7600 7009 5827 3463 6926 5661 3131 6262 4333)	
	479	8	(479 958 1916 3832 7664 7137 6083 3975 7950 7709 7227 6263 4335)	
	485	8	(485 970 1940 3880 7760 7329 6467 4743 1295 2590 5180 2169 4338)	
	489	8	(489 978 1956 3912 7824 7457 6723 5255 2319 4638 1085 2170 4340)	
	493	8	(493 986 1972 3944 7888 7585 6979 5767 3343 6686 5181 2171 4342)	
	497	8	(497 994 1988 3976 7952 7713 7235 6279 4367 543 1086 2172 4344)	
	507	8	(507 1014 2028 4056 8112 8033 7875 7559 6927 5663 3135 6270 4349)	
	551	8	(551 1102 2204 4408 625 1250 2500 5000 1809 3618 7236 6281 4371)	
	559	8	(559 1118 2236 4472 753 1506 3012 6024 3857 7714 7237 6283 4375)	
	569	8	(569 1138 2276 4552 913 1826 3652 7304 6417 4643 1095 2190 4380)	
	571	8	(571 1142 2284 4568 945 1890 3780 7560 6929 5667 3143 6286 4381)	
	573	8	(573 1146 2292 4584 977 1954 3908 7816 7441 6691 5191 2191 4382)	
	583	8	(583 1166 2332 4664 1137 2274 4548 905 1810 3620 7240 6289 4387)	
	591	8	(591 1182 2364 4728 1265 2530 5060 1929 3858 7716 7241 6291 4391)	
	599	8	(599 1198 2396 4792 1393 2786 5572 2953 5906 3621 7242 6293 4395)	
	601	8	(601 1202 2404 4808 1425 2850 5700 3209 6418 4645 1099 2198 4396)	
	605	8	(605 1210 2420 4840 1489 2978 5956 3721 7442 6693 5195 2199 4398)	
	615	8	(615 1230 2460 4920 1649 3298 6596 5001 1811 3622 7244 6297 4403)	
	621	8	(621 1242 2484 4968 1745 3490 6980 5769 3347 6694 5197 2203 4406)	
	629	8	(629 1258 2516 5032 1873 3746 7492 6793 5395 2599 5198 2205 4410)	
	659	8	(659 1318 2636 5272 2353 4706 1221 2442 4884 1577 3154 6308 4425)	
	661	8	(661 1322 2644 5288 2385 4770 1349 2698 5396 2601 5202 2213 4426)	
	637	8	(637 1274 2548 5096 2001 4002 8004 7817 7443 6695 5199 2207 4414)	
	675	8	(675 1350 2700 5400 2609 5218 2245 4490 789 1578 3156 6312 4433)	
	677	8	(677 1354 2708 5416 2641 5282 2373 4746 1301 2602 5204 2217 4434)	
	691	8	(691 1382 2764 5528 2865 5730 3269 6538 4885 1579 3158 6316 4441)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	695	8	(695 1390 2780 5560 2929 5858 3525 7050 5909 3627 7254 6317 4443)	
	697	8	(697 1394 2788 5576 2961 5922 3653 7306 6421 4651 1111 2222 4444)	
	699	8	(699 1398 2796 5592 2993 5986 3781 7562 6933 5675 3159 6318 4445)	
	701	8	(701 1402 2804 5608 3025 6050 3909 7818 7445 6699 5207 2223 4446)	
	727	8	(727 1454 2908 5816 3441 6882 5573 2955 5910 3629 7258 6325 4459)	
	731	8	(731 1462 2924 5848 3505 7010 5829 3467 6934 5677 3163 6326 4461)	
	733	8	(733 1466 2932 5864 3537 7074 5957 3723 7446 6701 5211 2231 4462)	
	741	8	(741 1482 2964 5928 3665 7330 6469 4747 1303 2606 5212 2233 4466)	
	745	8	(745 1490 2980 5960 3729 7458 6725 5259 2327 4654 1117 2234 4468)	
	751	8	(751 1502 3004 6008 3825 7650 7109 6027 3863 7726 7261 6331 4471)	
	759	8	(759 1518 3036 6072 3953 7906 7621 7051 5911 3631 7262 6333 4475)	
	795	8	(795 1590 3180 6360 4529 867 1734 3468 6936 5681 3171 6342 4493)	
	799	8	(799 1598 3196 6392 4593 995 1990 3980 7960 7729 7267 6343 4495)	
	805	8	(805 1610 3220 6440 4689 1187 2374 4748 1305 2610 5220 2249 4498)	
	813	8	(813 1626 3252 6504 4817 1443 2886 5772 3353 6706 5221 2251 4502)	
	815	8	(815 1630 3260 6520 4849 1507 3014 6028 3865 7730 7269 6347 4503)	
	823	8	(823 1646 3292 6584 4977 1763 3526 7052 5913 3635 7270 6349 4507)	
	825	8	(825 1650 3300 6600 5009 1827 3654 7308 6425 4659 1127 2254 4508)	
	827	8	(827 1654 3308 6616 5041 1891 3782 7564 6937 5683 3175 6350 4509)	
	831	8	(831 1662 3324 6648 5105 2019 4038 8076 7961 7731 7271 6351 4511)	
	845	8	(845 1690 3380 6760 5329 2467 4934 1677 3354 6708 5225 2259 4518)	
	847	8	(847 1694 3388 6776 5361 2531 5062 1933 3866 7732 7273 6355 4519)	
	853	8	(853 1706 3412 6824 5457 2723 5446 2701 5402 2613 5226 2261 4522)	
	859	8	(859 1718 3436 6872 5553 2915 5830 3469 6938 5685 3179 6358 4525)	
	863	8	(863 1726 3452 6904 5617 3043 6086 3981 7962 7733 7275 6359 4527)	
	869	8	(869 1738 3476 6952 5713 3235 6470 4749 1307 2614 5228 2265 4530)	
	889	8	(889 1778 3556 7112 6033 3875 7750 7309 6427 4663 1135 2270 4540)	
	891	8	(891 1782 3564 7128 6065 3939 7878 7565 6939 5687 3183 6366 4541)	
	893	8	(893 1786 3572 7144 6097 4003 8006 7821 7451 6711 5231 2271 4542)	
	895	8	(895 1790 3580 7160 6129 4067 8134 8077 7963 7735 7279 6367 4543)	
	917	8	(917 1834 3668 7336 6481 4771 1351 2702 5404 2617 5234 2277 4554)	
	921	8	(921 1842 3684 7368 6545 4899 1607 3214 6428 4665 1139 2278 4556)	
	923	8	(923 1846 3692 7384 6577 4963 1735 3470 6940 5689 3187 6374 4557)	
	925	8	(925 1850 3700 7400 6609 5027 1863 3726 7452 6713 5235 2279 4558)	
	939	8	(939 1878 3756 7512 6833 5475 2759 5518 2845 5690 3189 6378 4565)	
	943	8	(943 1886 3772 7544 6897 5603 3015 6030 3869 7738 7285 6379 4567)	
	953	8	(953 1906 3812 7624 7057 5923 3655 7310 6429 4667 1143 2286 4572)	
	955	8	(955 1910 3820 7640 7089 5987 3783 7566 6941 5691 3191 6382 4573)	
	973	8	(973 1946 3892 7784 7377 6563 4935 1679 3358 6716 5241 2291 4582)	
	985	8	(985 1970 3940 7880 7569 6947 5703 3215 6430 4669 1147 2294 4588)	
	987	8	(987 1974 3948 7896 7601 7011 5831 3471 6942 5693 3195 6390 4589)	
	989	8	(989 1978 3956 7912 7633 7075 5959 3727 7454 6717 5243 2295 4590)	
	991	8	(991 1982 3964 7928 7665 7139 6087 3983 7966 7741 7291 6391 4591)	
	999	8	(999 1998 3996 7992 7793 7395 6599 5007 1823 3646 7292 6393 4595)	
	1007	8	(1007 2014 4028 8056 7921 7651 7111 6031 3871 7742 7293 6395 4599)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	1013	8	(1013 2026 4052 8104 8017 7843 7495 6799 5407 2623 5246 2301 4602)	
	1021	8	(1021 2042 4084 8168 8145 8099 8007 7823 7455 6719 5247 2303 4606)	
	1179	8	(1179 2358 4716 1241 2482 4964 1737 3474 6948 5705 3219 6438 4685)	
	1183	8	(1183 2366 4732 1273 2546 5092 1993 3986 7972 7753 7315 6439 4687)	
	1191	8	(1191 2382 4764 1337 2674 5348 2505 5010 1829 3658 7316 6441 4691)	
	1199	8	(1199 2398 4796 1401 2802 5604 3017 6034 3877 7754 7317 6443 4695)	
	1203	8	(1203 2406 4812 1433 2866 5732 3273 6546 4901 1611 3222 6444 4697)	
	1207	8	(1207 2414 4828 1465 2930 5860 3529 7058 5925 3659 7318 6445 4699)	
	1213	8	(1213 2426 4852 1513 3026 6052 3913 7826 7461 6731 5271 2351 4702)	
	1227	8	(1227 2454 4908 1625 3250 6500 4809 1427 2854 5708 3225 6450 4709)	
	1235	8	(1235 2470 4940 1689 3378 6756 5321 2451 4902 1613 3226 6452 4713)	
	1259	8	(1259 2518 5036 1881 3762 7524 6857 5523 2855 5710 3229 6458 4725)	
	1267	8	(1267 2534 5068 1945 3890 7780 7369 6547 4903 1615 3230 6460 4729)	
	1331	8	(1331 2662 5324 2457 4914 1637 3274 6548 4905 1619 3238 6476 4761)	
	1333	8	(1333 2666 5332 2473 4946 1701 3402 6804 5417 2643 5286 2381 4762)	
	1339	8	(1339 2678 5356 2521 5042 1893 3786 7572 6953 5715 3239 6478 4765)	
	1343	8	(1343 2686 5372 2553 5106 2021 4042 8084 7977 7763 7335 6479 4767)	
	1363	8	(1363 2726 5452 2713 5426 2661 5322 2453 4906 1621 3242 6484 4777)	
	1383	8	(1383 2766 5532 2873 5746 3301 6602 5013 1835 3670 7340 6489 4787)	
	1397	8	(1397 2794 5588 2985 5970 3749 7498 6805 5419 2647 5294 2397 4794)	
	1403	8	(1403 2806 5612 3033 6066 3941 7882 7573 6955 5719 3247 6494 4797)	
	1407	8	(1407 2814 5628 3065 6130 4069 8138 8085 7979 7767 7343 6495 4799)	
	1431	8	(1431 2862 5724 3257 6514 4837 1483 2966 5932 3673 7346 6501 4811)	
	1437	8	(1437 2874 5748 3305 6610 5029 1867 3734 7468 6745 5299 2407 4814)	
	1439	8	(1439 2878 5756 3321 6642 5093 1995 3990 7980 7769 7347 6503 4815)	
	1451	8	(1451 2902 5804 3417 6834 5477 2763 5526 2861 5722 3253 6506 4821)	
	1459	8	(1459 2918 5836 3481 6962 5733 3275 6550 4909 1627 3254 6508 4825)	
	1461	8	(1461 2922 5844 3497 6994 5797 3403 6806 5421 2651 5302 2413 4826)	
	1467	8	(1467 2934 5868 3545 7090 5989 3787 7574 6957 5723 3255 6510 4829)	
	1469	8	(1469 2938 5876 3561 7122 6053 3915 7830 7469 6747 5303 2415 4830)	
	1471	8	(1471 2942 5884 3577 7154 6117 4043 8086 7981 7771 7351 6511 4831)	
	1485	8	(1485 2970 5940 3689 7378 6565 4939 1687 3374 6748 5305 2419 4838)	
	1487	8	(1487 2974 5948 3705 7410 6629 5067 1943 3886 7772 7353 6515 4839)	
	1495	8	(1495 2990 5980 3769 7538 6885 5579 2967 5934 3677 7354 6517 4843)	
	1499	8	(1499 2998 5996 3801 7602 7013 5835 3479 6958 5725 3259 6518 4845)	
	1501	8	(1501 3002 6004 3817 7634 7077 5963 3735 7470 6749 5307 2423 4846)	
	1503	8	(1503 3006 6012 3833 7666 7141 6091 3991 7982 7773 7355 6519 4847)	
	1515	8	(1515 3030 6060 3929 7858 7525 6859 5527 2863 5726 3261 6522 4853)	
	1517	8	(1517 3034 6068 3945 7890 7589 6987 5783 3375 6750 5309 2427 4854)	
	1533	8	(1533 3066 6132 4073 8146 8101 8011 7831 7471 6751 5311 2431 4862)	
	1535	8	(1535 3070 6140 4089 8178 8165 8139 8087 7983 7775 7359 6527 4863)	
	1655	8	(1655 3310 6620 5049 1907 3814 7628 7065 5939 3687 7374 6557 4923)	
	1659	8	(1659 3318 6636 5081 1971 3942 7884 7577 6963 5735 3279 6558 4925)	
	1693	8	(1693 3386 6772 5353 2515 5030 1869 3738 7476 6761 5331 2471 4942)	
	1695	8	(1695 3390 6780 5369 2547 5094 1997 3994 7988 7785 7379 6567 4943)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	1703	8	(1703 3406 6812 5433 2675 5350 2509 5018 1845 3690 7380 6569 4947)	
	1711	8	(1711 3422 6844 5497 2803 5606 3021 6042 3893 7786 7381 6471 4951)	
	1717	8	(1717 3434 6868 5545 2899 5798 3405 6810 5429 2667 5334 2477 4954)	
	1723	8	(1723 3446 6892 5593 2995 5990 3789 7578 6965 5739 3287 6574 4957)	
	1727	8	(1727 3454 6908 5625 3059 6118 4045 8090 7989 7787 7383 6575 4959)	
	1743	8	(1743 3486 6972 5753 3315 6630 5069 1947 3894 7788 7385 6579 4967)	
	1749	8	(1749 3498 6996 5801 3411 6822 5453 2715 5430 2669 5338 2485 4970)	
	1755	8	(1755 3510 7020 5849 3507 7014 5837 3483 6966 5741 3291 6582 4973)	
	1783	8	(1783 3566 7132 6073 3955 7910 7629 7067 5943 3695 7390 6589 4987)	
	1787	8	(1787 3574 7148 6105 4019 8038 7885 7579 6967 5743 3295 6590 4989)	
	1851	8	(1851 3702 7404 6617 5043 1895 3790 7580 6969 5747 3303 6606 5021)	
	1853	8	(1853 3706 7412 6633 5075 1959 3918 7836 7481 6771 5351 2511 5022)	
	1855	8	(1855 3710 7420 6649 5107 2023 4046 8092 7993 7795 7399 6607 5023)	
	1871	8	(1871 3742 7484 6777 5363 2535 5070 1949 3898 7796 7401 6611 5031)	
	1877	8	(1877 3754 7508 6825 5459 2727 5454 2717 5434 2677 5354 2517 5034)	
	1879	8	(1879 3758 7516 6841 5491 2791 5582 2973 5946 3701 7402 6613 5035)	
	1887	8	(1887 3774 7548 6905 5619 3047 6094 3997 7994 7797 7403 6615 5039)	
	1903	8	(1903 3806 7612 7033 5875 3559 7118 6045 3899 7798 7405 6619 5047)	
	1909	8	(1909 3818 7636 7081 5971 3751 7502 6813 5435 2679 5358 2525 5050)	
	1963	8	(1963 3926 7852 7513 6835 5479 2767 5534 2877 5754 3317 6634 5077)	
	1965	8	(1965 3930 7860 7529 6867 5543 2895 5790 3389 6778 5365 2539 5078)	
	1967	8	(1967 3934 7868 7545 6899 5607 3023 6046 3901 7802 7413 6635 5079)	
	1975	8	(1975 3950 7900 7609 7027 5863 3535 7070 5949 3707 7414 6637 5083)	
	1983	8	(1983 3966 7932 7673 7155 6119 4047 8094 7997 7803 7415 6639 5087)	
	2005	8	(2005 4010 8020 7849 7507 6823 5455 2719 5438 2685 5370 2549 5098)	
	2007	8	(2007 4014 8028 7865 7539 6887 5583 2975 5950 3709 7418 6645 5099)	
	2011	8	(2011 4022 8044 7897 7603 7015 5839 3487 6974 5757 3323 6646 5101)	
	2029	8	(2029 4058 8116 8041 7891 7591 6991 5791 3391 6782 5373 2555 5110)	
	2031	8	(2031 4062 8124 8057 7923 7655 7119 6047 3903 7806 7421 6651 5111)	
	2043	8	(2043 4086 8172 8153 8115 8039 7887 7583 6975 5759 3327 6654 5117)	
	2045	8	(2045 4090 8180 8169 8147 8103 8015 7839 7487 6783 5375 2559 5118)	
	2747	8	(2747 5494 2797 5594 2997 5994 3797 7594 6997 5803 3415 6830 5469)	
	2779	8	(2779 5558 2925 5850 3509 7018 5845 3499 6998 5805 3419 6838 5485)	
	2783	8	(2783 5566 2941 5882 3573 7146 6101 4011 8022 7853 7515 6839 5487)	
	2815	8	(2815 5630 3069 6138 4085 8170 8149 8107 8023 7855 7519 6847 5503)	
	2927	8	(2927 5854 3517 7034 5877 3563 7126 6061 3931 7862 7533 6875 5559)	
	2935	8	(2935 5870 3549 7098 6005 3819 7638 7085 5979 3767 7534 6877 5563)	
	2943	8	(2943 5886 3581 7162 6133 4075 8150 8109 8027 7863 7535 6879 5567)	
	3003	8	(3003 6006 3821 7642 7093 5995 3799 7598 7005 5819 3447 6894 5597)	
	3035	8	(3035 6070 3949 7898 7605 7019 5847 3503 7006 5821 3451 6902 5613)	
	3039	8	(3039 6078 3965 7930 7669 7147 6103 4015 8030 7869 7547 6903 5615)	
	3055	8	(3055 6110 4029 8058 7925 7659 7127 6063 3935 7870 7549 6907 5623)	
	3063	8	(3063 6126 4061 8122 8053 7915 7639 7087 5983 3775 7550 6909 5627)	
	3567	8	(3567 7134 6077 3963 7926 7661 7131 6071 3951 7902 7613 7035 5879)	
	3583	8	(3583 7166 6141 4091 8182 8173 8155 8119 8047 7903 7615 7039 5887)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	3967	8	(3967 7934 7677 7163 6135 4079 8158 8125 8059 7927 7663 7135 6079)	
	37	10	(37 74 148 296 592 1184 2368 4736 1281 2562 5124 2057 4114)	
	43	10	(43 86 172 344 688 1376 2752 5504 2817 5634 3077 6154 4117)	
	45	10	(45 90 180 360 720 1440 2880 5760 3329 6658 5125 2059 4118)	
	47	10	(47 94 188 376 752 1504 3008 6016 3841 7682 7173 6155 4119)	
	49	10	(49 98 196 392 784 1568 3136 6272 4353 515 1030 2060 4120)	
	51	10	(51 102 204 408 816 1632 3264 6528 4865 1539 3078 6156 4121)	
	59	10	(59 118 236 472 944 1888 3776 7552 6913 5635 3079 6158 4125)	
	103	10	(103 206 412 824 1648 3296 6592 4993 1795 3590 7180 6169 4147)	
	105	10	(105 210 420 840 1680 3360 6720 5249 2307 4614 1037 2074 4148)	
	113	10	(113 226 452 904 1808 3616 7232 6273 4355 519 1038 2076 4152)	
	141	10	(141 282 564 1128 2256 4512 833 1666 3332 6664 5137 2083 4166)	
	153	10	(153 306 612 1224 2448 4896 1601 3202 6404 4617 1043 2086 4172)	
	169	10	(169 338 676 1352 2704 5408 2625 5250 2309 4618 1045 2090 4180)	
	177	10	(177 354 708 1416 2832 5664 3137 6274 4357 523 1046 2092 4184)	
	237	10	(237 474 948 1896 3792 7584 6977 5763 3335 6670 5149 2107 4214)	
	245	10	(245 490 980 1960 3920 7840 7489 6787 5383 2575 5150 2109 4218)	
	265	10	(265 530 1060 2120 4240 289 578 1156 2312 4624 1057 2114 4228)	
	297	10	(297 594 1188 2376 4752 1313 2626 5252 2313 4626 1061 2122 4244)	
	351	10	(351 702 1404 2808 5616 3041 6082 3973 7946 7701 7211 6231 4271)	
	363	10	(363 726 1452 2904 5808 3425 6850 5509 2827 5654 3117 6234 4277)	
	381	10	(381 762 1524 3048 6096 4001 8002 7813 7435 6679 5167 2143 4286)	
	429	10	(429 858 1716 3432 6864 5537 2883 5766 3341 6682 5173 2155 4310)	
	437	10	(437 874 1748 3496 6992 5793 3395 6790 5389 2587 5174 2157 4314)	
	469	10	(469 938 1876 3752 7504 6817 5443 2695 5390 2589 5178 2165 4330)	
	471	10	(471 942 1884 3768 7536 6881 5571 2951 5902 3613 7226 6261 4331)	
	483	10	(483 966 1932 3864 7728 7265 6339 4487 783 1566 3132 6264 4337)	
	563	10	(563 1126 2252 4504 817 1634 3268 6536 4881 1571 3142 6284 4377)	
	587	10	(587 1174 2348 4696 1201 2402 4804 1417 2834 5668 3145 6290 4389)	
	589	10	(589 1178 2356 4712 1233 2466 4932 1673 3346 6692 5193 2195 4390)	
	607	10	(607 1214 2428 4856 1521 3042 6084 3977 7954 7717 7243 6295 4399)	
	619	10	(619 1238 2476 4952 1713 3426 6852 5513 2835 5670 3149 6298 4405)	
	665	10	(665 1330 2660 5320 2449 4898 1605 3210 6420 4649 1107 2214 4428)	
	793	10	(793 1586 3172 6344 4497 803 1606 3212 6424 4657 1123 2246 4492)	
	843	10	(843 1686 3372 6744 5297 2403 4806 1421 2842 5684 3177 6354 4517)	
	851	10	(851 1702 3404 6808 5425 2659 5318 2445 4890 1589 3178 6356 4521)	
	885	10	(885 1770 3540 7080 5969 3747 7494 6797 5403 2615 5230 2269 4538)	
	887	10	(887 1774 3548 7096 6001 3811 7622 7053 5915 3639 7278 6365 4539)	
	933	10	(933 1866 3732 7464 6737 5283 2375 4750 1309 2618 5236 2281 4562)	
	937	10	(937 1874 3748 7496 6801 5411 2631 5262 2333 4666 1141 2282 4564)	
	975	10	(975 1950 3900 7800 7409 6627 5063 1935 3870 7740 7289 6387 4583)	
	1003	10	(1003 2006 4012 8024 7857 7523 6855 5519 2847 5694 3197 6394 4597)	
	1019	10	(1019 2038 4076 8152 8113 8035 7879 7567 6943 5695 3199 6398 4605)	
	1181	10	(1181 2362 4724 1257 2514 5028 1865 3730 7460 6729 5267 2343 4686)	
	1231	10	(1231 2462 4924 1657 3314 6628 5065 1939 3878 7756 7321 6451 4711)	

Table A.8 (Continued)

$\mathbb{F}_{2^n}$	d	Δ_f	Cyclotomic Cosets	References
$\mathbb{F}_{2^{13}}$	1237	10	(1237 2474 4948 1705 3410 6820 5449 2707 5414 2637 5274 2357 4714)	
	1275	10	(1275 2550 5100 2009 4018 8036 7881 7571 6951 5711 3231 6462 4733)	
	1277	10	(1277 2554 5108 2025 4050 8100 8009 7827 7463 6735 5279 2367 4734)	
	1325	10	(1325 2650 5300 2409 4818 1445 2890 5780 3369 6738 5285 2379 4758)	
	1371	10	(1371 2742 5484 2777 5554 2917 5834 3477 6954 5717 3243 6486 4781)	
	1395	10	(1395 2790 5580 2969 5938 3685 7370 6549 4907 1623 3246 6492 4793)	
	1491	10	(1491 2982 5964 3737 7474 6757 5323 2455 4910 1629 3258 6516 4841)	
	1511	10	(1511 3022 6044 3897 7794 7397 6603 5015 1839 3678 7356 6521 4851)	
	1531	10	(1531 3062 6124 4057 8114 8037 7883 7575 6959 5727 3263 6526 4861)	
	1645	10	(1645 3290 6580 4969 1747 3494 6988 5785 3379 6758 5325 2459 4918)	
	1663	10	(1663 3326 6652 5113 2035 4070 8140 8089 7987 7783 7375 6559 4927)	
	1725	10	(1725 3450 6900 5609 3027 6054 3917 7834 7477 6763 5335 2479 4958)	
	1757	10	(1757 3514 7028 5865 3539 7078 5965 3739 7478 6765 5339 2487 4974)	
	1767	10	(1767 3534 7068 5945 3699 7398 6605 5019 1847 3694 7388 6585 4979)	
	1885	10	(1885 3770 7540 6889 5587 2983 5966 3741 7482 6773 5355 2519 5038)	
	1901	10	(1901 3802 7604 7017 5843 3495 6990 5789 3387 6774 5357 2523 5046)	
	2015	10	(2015 4030 8060 7929 7667 7143 6095 3999 7998 7805 7419 6647 5103)	
	2751	10	(2751 5502 2813 5626 3061 6122 4053 8106 8021 7851 7511 6831 5471)	
	2911	10	(2911 5822 3453 6906 5621 3051 6102 4013 8026 7861 7531 6871 5551)	
	3839	10	(3839 7678 7165 6139 4087 8174 8157 8123 8055 7919 7647 7103 6015)	
	25	12	(25 50 100 200 400 800 1600 3200 6400 4609 1027 2054 4108)	
	159	12	(159 318 636 1272 2544 5088 1985 3970 7940 7689 7187 6183 4175)	
	213	12	(213 426 852 1704 3408 6816 5441 2691 5382 2573 5146 2101 4202)	
	419	12	(419 838 1676 3352 6704 5217 2243 4486 781 1562 3124 6248 4305)	
	423	12	(423 846 1692 3384 6768 5345 2499 4998 1805 3610 7220 6249 4307)	
	457	12	(457 914 1828 3656 7312 6433 4675 1159 2318 4636 1081 2162 4324)	
	487	12	(487 974 1948 3896 7795 7393 6595 4999 1807 3614 7228 6265 4339)	
	565	12	(565 1130 2260 4520 849 1698 3396 6792 5393 2595 5190 2189 4378)	
	597	12	(597 1194 2388 4776 1361 2722 5444 2697 5394 2597 5194 2197 4394)	
	651	12	(651 1302 2604 5208 2225 4450 709 1418 2836 5672 3153 6306 4421)	
	685	12	(685 1370 2740 5480 2769 5538 2885 5770 3349 6698 5205 2219 4438)	
	735	12	(735 1470 2940 5880 3569 7138 6085 3979 7958 7725 7259 6327 4463)	
	757	12	(757 1514 3028 6056 3921 7842 7493 6795 5399 2607 5214 2237 4474)	
	971	12	(971 1942 3884 7768 7345 6499 4807 1423 2846 5692 3193 6386 4581)	
	979	12	(979 1958 3916 7832 7473 6755 5319 2447 4894 1597 3194 6388 4585)	
	981	12	(981 1962 3924 7848 7505 6819 5447 2703 5406 2621 5242 2293 4586)	
	983	12	(983 1966 3932 7864 7537 6883 5575 2959 5918 3645 7290 6389 4587)	
	997	12	(997 1994 3988 7976 7761 7331 6471 4751 1311 2622 5244 2297 4594)	
	1375	12	(1375 2750 5500 2809 5618 3045 6090 3989 7978 7765 7339 6487 4783)	
	1447	12	(1447 2894 5788 3385 6770 5349 2507 5014 1837 3674 7348 6505 4819)	
	1	8192	(1 2 4 8 16 32 64 128 256 512 1024 2048 4096)	

APPENDIX B. MATLAB PROGRAMS

Included here are two Matlab programs which were of use in the study of APN functions. The first, FcnConverter.m is the precursor to the C++ program PolyFcnConverter.cpp. A change from using Matlab to using C++ was necessary in this case because Matlab could not support the recursive function calls required to determine the terms in the component functions of f over $\mathbb{F}_2^n$. The Matlab program given here only serves to define the corresponding map, but is incapable of defining the corresponding functions. The second program given here, CyclotomicCoset.m is just a short function that generates the cyclotomic coset for any given power d. This was especially useful in classifying all maps with regards to their uniformity.

B.1 FcnConverter.m

```
% Program FcnConverter.m
% the purpose of this program is to take a function F in GF(2^n) and find
% the corrsponding function in GF(2)^n

%Let F be the function F(x)=x^k over GF(2^n);  0 < k < 2^n - 1

function FcnConverter
n=4; %need to specify each time program is run
k=4; %need to specify each time program is run
primitive=zeros(1,n); %vector rep of primitive poly
Table=zeros(k*(2^n-2),n); %def. of table containing reps of powers on alpha
temp=zeros(1,n); % 1xn row vector
```

```
fMat=zeros(2^n,n); %matrix rep of the map f; rows are f(row-1)
notfound = 1; %boolean variable; 1 means true.
alpha = 1; %initializing the power on alpha to 1

% define the row vector corresponding to the primitive polynomial; powers
% are descending currently it is defined for n=4
primitive(1,3)=1;
primitive(1,4)=1;

% define the vector rep of alpha to the zero power
for i = 1:n-1
    alphazero(1,i) = 0;
end
alphazero(1,n) = 1;

% create the Table of reps of powers on alpha
% multiplying by alpha is the same as shifting every entry 1 unit to the
% left and adding the primitive in at the end if a 1 "falls off" the left
% side so that is how the entries of the table are generated
for pwr = 1:k*(2^n-2)
    if pwr == 1
        temp = alphazero;
    else temp = Table(pwr-1,:);
    end %end if-then-else
    for col = 1:n-1
        Table(pwr, col) = temp(1,col+1);
        Table(pwr,n) = 0;
        if temp(1,1) == 1
```

```
            Table(pwr,:) = mod(Table(pwr,:) + primitive(1,:),2);
        end %end if
    end %end for col
end %end for pwr

% define a matrix for function f (which is over the vector space GF(2)^n)
% such that the entry correpsonds to the vector that the binary rep of (row-1)
% maps to. need to do this so that row one corresponds to the all 0 vector

% initialize 1st 2 rows.
for i = 1:n
    fMat(1,i) = 0;
    if i < n
        fMat(2,i) = 0;
    else
        fMat(2,i) = 1;
    end
end

% main definition
for row = 3:2^n
    % find binary rep of row-1
    temp1 = row-1;
    for index = 1:n
        power = n-index;
        if temp1 >= 2^power
            binaryrep(1,index)=1;
            temp1 = temp1 - 2^power;
```

```
        else
            binaryrep(1,index)=0;
        end % end if
    end % end for index

    % search the table for this binary rep
    exponent = 1;
    notfound = 1;
    while notfound == 1
        if Table(exponent,:) == binaryrep(1,:)
            notfound = 0;
            alpha = exponent;
        else
        exponent = exponent + 1;
        end %end if-else
    end %end while
    % once the binary rep has been found we want to assign the value that it
    % maps to under the map F to our map f
    fMat(row,:) = Table(alpha*k,:);
end % end for row

fMat
end
```

B.2 CyclotomicCoset.m

```
% Program CyclotomicCoset.m
% Purpose: to find the cyclotomic coset of a user specified value
```

```
function CyclotomicCosets
n=13;
coset=zeros(1,n);
quit = 0;

while (quit < 1)

    % get input from user
    rep = input('Enter an element of the desired Cyclotomic Coset: ');
    coset(1,1)=rep;

    % find the cyclotomic coset determined by this value
    for i = 2:n
        coset(1,i) = mod(2*coset(1,i-1),2^n-1);
    end %end for

    % display the coset
    disp('The Cyclotomic coset is: ');
    disp(coset);
    quit = input('Enter 0 to continue: ');

end %end while
```

APPENDIX C. VISUAL C++ PROGRAMS

Although we initially sought to utilize Mathlab for most of our computer programming needs, we eventually were forced to switch gears and use C++ because of Matlab's inability to support our need for recursion in the PolyFcnConverter.cpp program. We will here give a brief overview of the programs utilized as well as some instruction on how to best use them. It should be noted that we have just included the program code from the main files of our various projects. In order to run any of the programs, the user should create a Visual C++ Project with the appropriate name, copy the code into the main file, and use the standard header files that Visual Studio generates.

Although the only known binary APN functions are power maps we have included the capability to handle other polynomial functions in many of our programs. PolyFcnConverter.cpp is the extension of the original Matlab program designed to take power maps over the finite field $\mathbb{F}_{2^n}$ and generate the corresponding function over the vector space $\mathbb{F}_2^n$. This program works well for small values of n and d. Both PolyFunctionSolCtr.cpp and SolutionCountervfile.cpp are programs which are designed to, given a function, determine the maximum number of solutions to the equation $f(x+a)+f(x) = b$ over the finite field $\mathbb{F}_{2^n}$. The programs differ in that PolyFunctionSolCtr.cpp is designed to test user specified polynomial functions, whereas SolutionCountervfile.cpp is designed to run independent of user input by testing multiple power functions whose powers are read-in from a user created file. Both of these programs can be useful in testing various functions to determine whether or not they are APN.

The programs AssociationSchemes.cpp, SearchScheme.cpp, and MultiPointStructFinder.cpp should be used in concert to determine the local structure of the graphs generated using the defined incidence structure for APN functions. AssociationSchemes.cpp finds the Point-Block

pairs that are in Relation 1 according to our defined incidence structure for a given power map $f(x) = x^d$ over $\mathbb{F}_{2^n}$ and writes these pairs to a file which can then be used for input into the SearchScheme.cpp program. The purpose of this program is to find the "common neighbors" of user specified points. In order to efficiently run the MulitPointStructFinder.cpp program, which writes all maximal cliques containing the user specified points to a file, we must run the SearchScheme.cpp program numerous times. Generally, we run the SearchScheme.cpp program for one point to find its neighborhood, then choose one of its neighbors and find the common neighborhood of both and continue in this manner until the common neighborhood is of a manageable size. Usually, a size in the lower twenties will yield a reasonable running time for the MultiPointStructFinder.cpp program. As the length of the "initial segment" (i.e. number of common neighbors specified) increases, the running time for the MultiPointStructFinder.cpp program decreases. However, the tradeoff is an exponential increase in the amount of work that must be done by the user before implementing the program. So there is a rather significant balancing act taking place. On one hand, you would like to speed up the running time, but on the other, you would like to minimize the amount of work that needs to be done. Because of these competing goals, some significant study into how the algorithm could be improved is definitely warranted.

C.1 Conversion from functions over $\mathbb{F}_{2^n}$ to functions over $\mathbb{F}_2^n$

```
// This is the main project file for a VC++ application project
/*****************************************************************************
Program: PolyFcnConverter.cpp
Author:  Mandi Maxwell
Purpose: To take a function F in GF(2^n) and find the corresponding function
 in GF(2)^n
Input from:  Keyboard
Output to:   Screen
Functions used: create_alpha_table, create_f_map, create_label_matrix,
```

```
create_X_matrix, create_f_matrix
*****************************************************************************/

// preprocessor directives:
#include "stdafx.h"
#include <iostream>
#include <string>
#include <iomanip>
#include <math.h>
#include <cmath>
#include <functional>

using namespace std;

// Named constant declaration:
const short int n = 5;
const short int numterms = 3; //number of terms in polynomial F(x)
const short int degree = 12;  //degree of the polynomial F(x)
//const int size = (int) degree*(pow(2,n)-2);
const int size = 360; //for n=5, degree=12
//const int 2toN = (int) pow(2,n);
const int twotoN = 32;

// Let F be the function F(x)=x^d1+x^d2+...+x^dk over GF(2^n);  0 < di < 2^n - 1

void main()
{
cout << "size " << size << endl;
```

```
cout << "2 raised to the n is " << twotoN << endl;

// Local variable declarations:
short int d[numterms];       // vector of exponents of terms in poly F(x)
short int primitive[n];  // vector rep of primitive poly
short int Table[size][n];  // def. of table containing reps of powers on alpha
short int fMap[twotoN][n];  // matrix rep of the map f; rows are f(row)
short int alphazero[n];  // vector rep of alpha^0
short int VLM[n][twotoN-1];  // matrix of vertex labels
short int XMatrix[twotoN][twotoN]; // matrix for X
short int fMatrix[n][twotoN]; // matrix for function f
short int i,j;

char rtn;

// function prototypes:
void create_alpha_table(short int alphazero[], short int Table[][n],
short int primitive[]);
void create_f_map(short int fMap[][n], short int Table[][n], short int d[]);
void create_label_matrix(short int VLM[][twotoN-1]);
void create_X_matrix(short int VLM[][twotoN-1], short int XMatrix[twotoN][twotoN]);
void create_f_matrix(short int fmap[][n], short int XMatrix[][twotoN],
 short int fMatrix[][twotoN], short int VLM[][twotoN-1]);

// zeroing out matrices
for (i = 0; i < n; i++)
{
for (j = 0; j < size; j++)
```

```
{
Table[j][i] = 0;
}// end for j
primitive[i] = 0;
} //end for i

for (j = 0; j < twotoN; j++)
for (i = 0; i < n; i++)
{
fMap[j][i] = 0;
fMatrix[j][i] = 0;
if (j < twotoN-1)
VLM[i][j] = 0;
}
// end for
//end for

// zero out XMatrix
for (i = 0; i < twotoN; i++)
for (j = 0; j < twotoN; j++)
XMatrix[i][j] = 0;

// define the row vector for the exponents of the terms in poly F(x)
// currently F(x) = x^12+x^6+x^3
d[0] = 12;
d[1] = 6;
d[2] = 3;
```

```
// define the row vector of the primitive polynomial; powers
// are descending currently it is defined for n=5; x^5+x^2+1=0
primitive[2] = 1;
primitive[4] = 1;

// define the vector rep of alpha to the zero power
for (i = 0; i < n-1; i++)
    alphazero[i] = 0;
alphazero[n-1] = 1;

// call function to create Table
create_alpha_table(alphazero, Table, primitive);

/*// display Table
for (i = 0; i < size; i++)
{
for (j = 0; j < n; j++)
cout << setw(1) << Table[i][j] << "  ";
cout << endl;
}*/

// call a function to create the map for function f
create_f_map(fMap, Table, d);

// call a function to create the variable label matrix (VLM)
create_label_matrix(VLM);

// call a function to create the matrix
```

```
create_X_matrix(VLM, XMatrix);

// call a function to create a matrix corresponding to function f
create_f_matrix(fMap, XMatrix, fMatrix, VLM);

cout << "Hit c to continue " << endl;
cin >> rtn;

} // end function main

/*****************************************************************************
Function Name: create_alpha_table
Purpose: This function creates the table of reps of powers on alpha.
Note: multiplying by alpha is the same as shifting every entry 1 unit to
the left and adding the primitive in at the end if a 1 "falls off" the left
side so that is how the entries of the table are generated
Called by: main
Functions Called: none
*****************************************************************************/
void create_alpha_table(short int alphazero[], short int Table[][n],
short int primitive[])
{
// local variables
short int temp[n];
short int pwr, col, i, j, l;

// initialize variables
for (i = 0; i < n; i++)
```

```
temp[i] = 0;

for (pwr = 0; pwr < size; pwr++)
{
if (pwr == 0)
for (i = 0; i < n; i++)
temp[i] = alphazero[i];
else
for (j = 0; j < n; j++)
temp[j]  = Table[pwr-1][j];
// end if-then-else
for (col = 0; col < n; col++)
{
Table[pwr][col] = temp[col+1];
} // end for col
Table[pwr][n-1] = 0;
if (temp[0] == 1)
for (l = 0; l < n; l++)
Table[pwr][l] = (Table[pwr][l] + primitive[l]) % 2;

} // end for pwr

} // end function create_alpha_table

/*************************************************************************
Function Name: create_f_map
Purpose: Defines a matrix for function f (which is over the vector space
 GF(2)^n) such that each row corresponds to the vector that the binary
```

```
 representation of this row maps to under map f.
Called by: main
Functions Called: Binary_Rep
*****************************************************************************/
void create_f_map(short int fMap[][n], short int Table[][n], short int d[])
{
// function prototypes
void Binary_Rep(short int binaryrep[], short int num);

// local variables
short int exponent, alpha, i, r, j, c;
short int value, t, tempexp;
short int binaryrep[n];
bool notfound;

// initialize variables
alpha = 1;
notfound = true;
exponent = 1;

// initialize 1st 2 rows of fMap and binaryrep
for (i = 0; i < n; i++)
{
binaryrep[i] = 0;
fMap[0][i] = 0;
if (i < n-1)
fMap[1][i] = 0;
else
```

```
fMap[1][i] = 1;
} // end for

// main definition
for (r = 2; r < twotoN; r++)
{
    // find binary rep of row r
    Binary_Rep(binaryrep, r);

cout << "binary rep for " << r << " is ";
for (j = 0; j < n; j++)
cout << binaryrep[j];
cout << endl;

    // search the table for this binary rep
    exponent = 1;
    notfound = true;
    while (notfound)
{
        for (c = 0; c < n; c++)
{
if (Table[exponent-1][c] != binaryrep[c])
{
notfound = true;
exponent++;
break;
} // end
} // end for c
```

```
if (c == n)
{
notfound = false;
alpha = exponent;
}
} // end while

    // once the binary rep has been found we want to assign the value that it
    // maps to under the map F to our map f
for (c = 0; c < n; c++)
{
value = 0;
for (t=0; t < numterms; t++)
{
tempexp = alpha*d[t]-1;
value = value + Table[tempexp][c];
} // end for t

fMap[r][c] = value % 2;
} // end for c

} // end for r

// display fMap
cout << "fMap " << endl;
for (r = 0; r < twotoN; r++)
{
for (c = 0; c < n; c++)
```

```
cout << fMap[r][c] << " ";
cout << endl;
}

} // end function create_f_map

/*****************************************************************************
Function Name: create_X_matrix
Purpose: Create a 2^n by 2^n matrix where rows correspond to the elements of
 GF(2)^n and columns correspond to the terms involving the xi's in the
 prescribed order.  (Prescribed order consists of the single terms first
 followed by the (n choose 2) terms involving a product of 2 xi's, etc.
 with the 2nd to last column containing the term containing all n xi's
 and the last column representing the constant 1).
Called by: main
Functions Called: Binary_Rep, term_search
*****************************************************************************/
void create_X_matrix(short int VLM[][twotoN-1], short int XMatrix[twotoN][twotoN])
{
// function prototypes
void Binary_Rep(short int binaryrep[], short int num);
void term_search(short int VLM[][twotoN-1], short int termlist[],
 short int XMatrix[][twotoN], short int num_nonzero,
 short int row);

// local variables
short int termlist[n]; // list of terms in function
short int X[n]; // binary rep of element of GF(2)^n
```

```
short int row, i, ptr, j, num_nonzero;

num_nonzero = 0;

// zero out X
for (i = 0; i < n; i++)
X[i] = 0;

// fill in all 1's in row 0 of XMatrix
for (i = 0; i < twotoN; i++)
XMatrix[0][i] = 1;

for (row = 1; row < twotoN; row++)
{
// zero out termlist
for (i = 0; i < n; i++)
termlist[i] = 0;

// get binary rep of current row
Binary_Rep(X,row);

// determine what terms are needed
ptr = 0;
for (i = 0; i < n; i++)
{
// if X[i] is not zero include term
if (X[i] != 0)
{
```

```
termlist[ptr] = i+1;
ptr++;
} // end if
} // end for

cout << "termlist for " << row << " is ";
for (j = 0; j < n; j++)
cout << termlist[j];
cout << endl;

// call function term_search to search VLM for all terms containing
// the components of termlist (size of termlist = ptr)
term_search(VLM, termlist, XMatrix, ptr, row);

} // end for row

// display XMatrix
for (row = 0; row < twotoN; row++)
{
cout << "Row " << row << " of XMatrix is ";
for (j = 0; j < twotoN; j++)
cout << XMatrix[row][j];
cout << endl;
} // end for row

} // end function create_X_matrix

/*****************************************************************************
```

```
Function Name: create_f_matrix
Purpose: Create an n by 2^n matrix for the function f where the rows correspond
 to the fi's and the columns correspond to the terms involving the xi's
 in the prescribed order.  (Prescribed order consists of the single terms first
 followed by the (n choose 2) terms involving a product of 2 xi's, etc.
 with the 2nd to last column containing the term containing all n xi's
 and the last column representing the constant 1).
Called by: main
Functions Called: none
*******************************************************************************/
void create_f_matrix(short int fmap[][n], short int XMatrix[][twotoN],
 short int fMatrix[][twotoN], short int VLM[][twotoN-1])
{
//local variables
short int tempMatrix[twotoN][twotoN];
short int col, row, term, c, r, i, j, total;
bool flag;

for (col = 0; col < n; col++)
{
// create (2^n)x(2^n) matrix of col of fmap times row of XMatrix
for (row = 0; row < twotoN; row++)
for (term = 0; term < twotoN; term++)
tempMatrix[row][term] = (fmap[row][col])*(XMatrix[row][term]);

// add columns of tempMatrix mod 2
for (c = 0; c < twotoN; c++)
{
```

```
total = 0;
for (r = 0; r < twotoN; r++)
total = total + tempMatrix[r][c];

fMatrix[col][c] =  total % 2;
} // end for c
} // end for col

// display fMatrix
cout << "The vector representation of function f is: " << endl;
for (i = 0; i < n; i++)
{
cout << "f" << i+1 << " is ";
for (j = 0; j < twotoN; j++)
cout << fMatrix[i][j];
cout << endl;
} // end for i

cout << endl;

//display the fi's in a more user-friendly form
for (i = 0; i < n; i++)
{
cout << "f" << i+1 << " is ";
flag = false;
for (j = 0; j < twotoN-1; j++)
{
if (fMatrix[i][j] == 1)
```

```
{
term = 0;
if (flag)
cout << "+"; //print a + sign if it's not the 1st term
while (VLM[term][j] != 0)
{
cout << "X" << VLM[term][j];
term ++;
}
flag = true;
} //end if
} // end for j
if (fMatrix[i][twotoN-1] == 1)
cout << "+1";
cout << endl;
} // end for i
cout << endl;

} // end function create_f_matrix

/*************************************************************************
Function Name: create_label_matrix
Purpose: To create a matrix containing the labels of all possible
 terms repsesenting products of Xi's for 0 < i <=n
Called by: main
Functions Called: Binomial_Coeff, AssignValues
*************************************************************************/
void create_label_matrix(short int VLM[][twotoN-1])
```

```
{
// function prototypes
short int Binomial_Coeff(int k, int t);
int AssignValues(short int VLM[][twotoN-1], int k, int i, int col, int row);

// Local variables
long int start;
short int row, col, t, r, c;

// intialize variables
start = -1;
row = 0;
col = 0;

// Creation of the VarLabelMatrix (VLM) which has n rows and 2^n-1 columns
for (t = 1; t <= n; t++)
{
start = start + Binomial_Coeff(n,t-1);
    col = AssignValues(VLM,n,t,start,row);
} //end for

// display VLM
cout << "VLM " << endl;
for (r = 0; r < n; r++)
{
for (c = 0; c < twotoN-1; c++)
cout << VLM[r][c] << " ";
cout << endl;
```

```
}

} // end function create_label_matrix

/*************************************************************************
Function Name: AssignValues
Purpose: This function is recursive in that given a binomial coeff
 corresponding to a group of terms it will fill in the first row
 corresponding to that grouping and then decompose it into its
 component parts and recursively call itself with these new binomial
 coeff. until all rows 'beneath' a particular grouping are filled.
Called by: create_label_matrix
Functions Called: Binomial_Coeff
Input: VLM,k,i,col,row where k and i are the components of a binomial coeff.
corresponding to a group of terms of size k choose i. row-col is the
position where the first value will be assigned.
Output: col which is the next column in which a value needs to be assigned.
*************************************************************************/
int AssignValues(short int VLM[][twotoN-1], int k, int i, int col, int row)
{
// function prototypes
short int Binomial_Coeff(int k, int t);

// local variables
long int uplim;
int j,l;

for (j = 1; j <= k-i+1; j++)
```

```
{
if (i == 1)
VLM[row][col+j-1] = n-k+j;
else
{
uplim = Binomial_Coeff(k-j,i-1);
for (l = 1; l <= uplim; l++)
VLM[row][col+l-1] = n-k+j;

col = AssignValues(VLM,k-j,i-1,col,row+1);
} // end else
} // end for j

if (i == 1)
return col+k;
else
return col;

} // end function AssignValues

/*****************************************************************************
Function Name: term_search
Purpose: To search through the matrix VLM to find all terms that contain the
 components of termlist and to change the values in the
 corresponding positions of XMatrix to 1's.
Called by: create_X_matrix
Functions Called: Binomial_Coeff
*****************************************************************************/
```

```
void term_search(short int VLM[][twotoN-1], short int termlist[],
 short int XMatrix[][twotoN], short int num_nonzero, short int row)
{
// function prototypes
short int Binomial_Coeff(int k, int t);

// local variables
short int startcol, c, term, coord, j;
bool match;

startcol = 0;

for (j = 1; j < num_nonzero; j++)
startcol = startcol + Binomial_Coeff(n,j);

// loop for search
for (c = startcol; c < twotoN-1; c++)
{
match = true;
term = 0; // position in termlist
coord = 0; // position in a column c of VLM

while (term < num_nonzero && match)
{
// search in VLM
if ((VLM[coord][c] == 0) || (coord >= n))
match = false;
else
```

```
if (termlist[term] == VLM[coord][c])
{
term++;
coord++;
}
else
coord++;
// end if-else search
} // end while

// if term was found in this column, need to change the 0 to a 1
// in this position of XMatrix this should be in column c

if (match)
XMatrix[row][c] = 1;

} // end for c

} // end function term_search

/*************************************************************************
Function Name: Binary_Rep
Purpose: Finds the binary representation of a number and returns it.
Called by: create_f_map, create_X_matrix
Functions Called: none
*************************************************************************/
void Binary_Rep(short int binaryrep[], short int num)
{
```

```
// local variables
short int j, power, temp1;

temp1 = num;
for (j = 0; j < n; j++)
{
        power = n-j-1;
        if (temp1 >= (int) pow(2,power))
{
            binaryrep[j] = 1;
            temp1 = temp1 - (int) pow(2,power);
} // end if
        else
            binaryrep[j] = 0;
} // end for j

} // end function Binary_Rep

/*************************************************************************
Function Name: Binomial_Coeff
Purpose: Calculates the binomial coeff (k choose t) and returns it.
Called by: Create_Label_Matrix, AssignValues, term_search
Functions Called: Factorial
*************************************************************************/
short int Binomial_Coeff(int k, int t)
{
// function prototypes
long int Factorial(int num);
```

```
// local variables
long int numerator, denom1, denom2;
short int value;

numerator = Factorial(k);
denom1 = Factorial(t);
denom2 = Factorial(k-t);

value = numerator/(denom1*denom2);

return value;

}// end function Binomial_Coeff

/*************************************************************************
Function Name: Factorial
Purpose: Calculates the factorial of a given number and returns it.
Called by: Binomial_Coeff
Functions Called: none
*************************************************************************/
long int Factorial(int num)
{
// local variables
int kount;
long int product;

product = 1;
```

```
for (kount = 1; kount <= num; kount++)
product = product * kount;
//end for

return product;

}// end function Factorial
```

C.2 Solution counting programs

C.2.1 PolyFunctionSolCtr.cpp

```
// This is the main project file for a VC++ application project
/*****************************************************************************
Program: PolyFunctionSolCtr.cpp
Author:  Mandi Maxwell
Purpose: To find the number of solutions of the equation f(x+a)+f(x)=b over
the field GF(2^n) where f(x) is a polynomial function.
Input from:  Keyboard
Output to:   Screen
Functions used: create_alpha_table, create_f_map, Find_Haf

Directions: You must hard code the values for the following constants
            before you run the program.

            n: as in GF(2^n)
            numterms: the number of terms in the polynomial function
            twotoN: which is 2^n
```

```
            You must also appropriately define the primitive vector to
            correspond to the primitive polynomial for the field GF(2^n).
*******************************************************************************/

// preprocessor directives:
#include "stdafx.h"
#include <iostream>
#include <string>
#include <iomanip>
#include <math.h>
#include <cmath>
#include <functional>

using namespace std;

// Named constant declaration:
const short int n = 5;
const short int numterms = 3; //number of terms in polynomial F(x)
const long int twotoN = 32;

// Let F(x)=x^d1+x^d2+...+x^dk over GF(2^n);  0 < di < 2^n - 1

void main()
{
    cout << "2 raised to the n is " << twotoN << endl;

    // Local variable declarations:
    short int degs[numterms];  // vector of exponents of terms in poly F(x)
```

```
short int primitive[n];       // vector rep of primitive poly
short int Table[twotoN-1][n];  // table of reps of powers on alpha
short int fMap[twotoN][n];    // matrix rep of the map f; rows are f(row)
short int alphazero[n];       // vector rep of alpha^0
short int alphaToPwr[n];      // vector rep of alpha^pwr
long int i,j;

short int rtn;

// function prototypes:
void create_alpha_table(short int alphazero[], short int Table[][n],
                        short int primitive[]);
void create_f_map(short int alphazero[], short int primitive[],
 short int fMap[][n], short int Table[][n], short int degs[]);
void Find_Haf(short int fMap[][n]);

rtn = 1;
// zeroing out matrices
for (i = 0; i < n; i++)
{
    for (j = 0; j < twotoN; j++)
        fMap[j][i] = 0;
    for (j = 0; j < twotoN-1; j++)
        Table[j][i] = 0;
    primitive[i] = 0;
    alphaToPwr[i] = 0;
} //end for i
```

```
// define the row vector for the exponents of the terms in the poly F(x)
// currently F(x)= x^15+x^14+x^7
degs[0]=15;
degs[1]=14;
degs[2]=7;

// define the row vector of the primitive polynomial; powers
// are descending currently it is defined for n=5; x^5+x^2+1=0
primitive[2] = 1;
primitive[4] = 1;

// define the vector rep of alpha to the zero power
for (i = 0; i < n-1; i++)
    alphazero[i] = 0;
alphazero[n-1] = 1;

// call function to create Table
create_alpha_table(alphazero, Table, primitive);

/* // display Table
for (i = 0; i < twotoN-1; i++)
{
    for (j = 0; j < n; j++)
        cout << setw(1) << Table[i][j] << "  ";
    cout << endl;
} */

while (rtn != 0)
```

```
{
// call a function to create the map for function f
create_f_map(alphazero, primitive, fMap, Table, degs);

// call a function to create and display Ha(f) for all a in the vector space
Find_Haf(fMap);

cout << "Hit 0 to end " << endl;
cin >> rtn;
} // end while

} // end function main

/*****************************************************************************
Function Name: create_alpha_table
Purpose: This function creates the table of reps of powers on alpha.
         Note: multiplying by alpha is the same as shifting every entry 1 unit
         to the left and adding the primitive in at the end if a 1 "falls off"
         the left side so that is how the entries of the table are generated
Called by: main
Functions Called: none
*****************************************************************************/
void create_alpha_table(short int alphazero[], short int Table[][n],
                        short int primitive[])
{
    // local variables
    short int temp[n];
    short int col, i, j, l;
```

```
    long int pwr;

    // initialize variables
    for (i = 0; i < n; i++)
    {
        temp[i] = 0;
        Table[0][i] = alphazero[i];
    }

    // Note: Table[pwr][] is actually alpha to the pwr+1

    for (pwr = 1; pwr < twotoN-1; pwr++)
    {
        for (j = 0; j < n; j++)
            temp[j]  = Table[pwr-1][j];

        for (col = 0; col < n; col++)
            Table[pwr][col] = temp[col+1];

        Table[pwr][n-1] = 0;
        if (temp[0] == 1)
            for (l = 0; l < n; l++)
                Table[pwr][l] = (Table[pwr][l] + primitive[l]) % 2;

    } // end for pwr

} // end function create_alpha_table
```

```
/*****************************************************************************
Function Name: create_f_map
Purpose: Defines a matrix for function f (which is over the vector space
         GF(2)^n) such that each row corresponds to the vector that the binary
         representation of this row maps to under map f.
Called by: main
Functions Called: Binary_Rep, power_raiser
*****************************************************************************/
void create_f_map(short int alphazero[], short int primitive[],
short int fMap[][n], short int Table[][n], short int degs[])
{
    // function prototypes
    void Binary_Rep(short int binaryrep[], long int num);
    void power_raiser(short int primitive[], short int alphaToPwr[],
     long int alphapwr);

    // local variables
    short int i, j, c, t;
    long int exponent, alpha, alphapwr;
    long int r;
    short int binaryrep[n], alphaToPwr[n], Temp[n];
    bool notfound;

    // initialize variables
    alpha = 1;
    notfound = true;
    exponent = 1;
```

```
    // initialize 1st 2 rows of fMap and binaryrep
    for (i = 0; i < n; i++)
    {
        binaryrep[i] = 0;
        fMap[0][i] = 0;
        if (i < n-1)
            fMap[1][i] = 0;
        else
        { // if numterms is odd then assign 1
            if ((numterms % 2) == 1)
                fMap[1][n-1] = 1;
            else
                fMap[1][n-1] = 0;
        } // end else
    } // end for

// main definition
for (r = 2; r < twotoN; r++)
{
    // find binary rep of row r
    Binary_Rep(binaryrep, r);

    cout << "binary rep for " << r << " is ";
    for (j = 0; j < n; j++)
        cout << binaryrep[j];
    cout << endl;

    // search the table for this binary rep
```

```
exponent = 0;
notfound = true;
while (notfound)
{
    for (c = 0; c < n; c++)
    {
        if (Table[exponent][c] != binaryrep[c])
        {
            notfound = true;
            exponent++;
            break;
        } // end
    } // end for c
    if (c == n)
    {
        notfound = false;
        alpha = exponent;
    }
} // end while

// once the binary rep has been found we want to assign the value that it
// maps to under the map F to our map f
for (i = 0; i < n; i++)
    Temp[i] = 0;

for(t = 0; t < numterms; t++)
{
    for (i = 0; i < n; i++)
```

```
            alphaToPwr[i] = alphazero[i];

        alphapwr = alpha*degs[t];
        power_raiser(primitive, alphaToPwr, alphapwr);

        for (c = 0; c < n; c++)
            Temp[c] = (Temp[c] + alphaToPwr[c]) % 2;
    } // end for t

    for (c = 0; c < n; c++)
        fMap[r][c] = Temp[c];
} // end for r

// display fMap
cout << "fMap " << endl;
for (r = 0; r < twotoN; r++)
{
    for (c = 0; c < n; c++)
        cout << fMap[r][c] << " ";
    cout << endl;
}

} // end function create_f_map

/*********************************************************************
Function Name: power_raiser
Purpose: This function returns alpha raised to a given power.
Called by: create_f_map
```

```
Functions Called: none
*****************************************************************/
void power_raiser(short int primitive[], short int alphaToPwr[],
long int alphapwr)
{
    // local variables
    short int temp[n];
    short int col, i, j;
    long int pwr;

    // Note: end result of the loop is that we get alpha to the pwr+1
    for (pwr = 0; pwr < alphapwr; pwr++)
    {
        for (i = 0; i < n; i++)
            temp[i] = alphaToPwr[i]; // stores alpha to pwr in temp

        for (j = 0; j < n-1; j++)
            alphaToPwr[j]  = temp[j+1];
        alphaToPwr[n-1] = 0;

        if (temp[0] == 1)
            for (col = 0; col < n; col++)
                alphaToPwr[col] = (alphaToPwr[col] + primitive[col]) % 2;
    } // end for pwr

} // end function power_raiser

/*****************************************************************************
```

```
Function Name: Find_Haf
Purpose: To find and calculate the size of Ha(f) for all a in the vector space.
Called by: main
Functions Called: Value_from_Binary, Binary_Rep
*****************************************************************************/
void Find_Haf(short int fMap[][n])
{
    // function prototypes
    long int Value_from_Binary(short int vector[]);
    void Binary_Rep(short int binaryrep[], long int num);

    // local variables
    short int a[n], x[n], temp[n], Hf[twotoN][n];
    short int Hfsize, digit, p;
    long int i, j, ctr, r, value;
    short int spot, c;
    long int c1, c2, c3, Sctr, maxsolutions;
    bool flag, flag2;
    char rtn;

    // initialize variables
    Hfsize = 0;
    value = 0;

    for (i = 0; i < twotoN; i++)
    {
        Binary_Rep(a,i); // finds binary rep of i and stores in vector a
        Hfsize = 0;
```

```
for (j = 0; j < twotoN; j++)
{
    Binary_Rep(x,j); // finds binary rep of j and stores in vector x
    for (spot = 0; spot < n; spot++)
        temp[spot] = (x[spot] + a[spot]) % 2; //calculates x+a
    // end for spot
    value = Value_from_Binary(temp);
    for (c = 0; c < n; c++)  //find f(x+a)+f(x)
        Hf[j][c] = (fMap[value][c] + fMap[j][c]) % 2;

    // loop to calculate the number of different values in H(f)
    ctr = 0;
    while (ctr < j)
    {
        // loop to compare digits in the vectors
        digit = 0;
        flag = true;
        while (flag)
        {
            if (Hf[ctr][digit] != Hf[j][digit])
            {
                // not equal so increment counter and keep searching
                flag = false;
                ctr++;
            } //end if
            else digit++;
            if (digit == n)
            {
```

```
                    // equal so it's not a new vector; quit searching
                    flag = false;
                    ctr = j + 100;
                } // end if
            } // end while flag
        } // end while ctr
        if (ctr == j)
            Hfsize = Hfsize++; // none match so j is a new vector

    } // end for j

    // Display a and Ha(f) on the screen
    cout << "a = ";
    for (r = 0; r < n; r++)
        cout << a[r] << " ";
    cout << endl;
    /* cout << "H = " << endl;
    for (r = 0; r < twotoN; r++)
    {
        for (p = 0; p < n; p++)
            cout << Hf[r][p] << " ";
        cout << endl;
    } // end for r */
    cout << "Size of Ha(f) = " << Hfsize << endl;

/*  //pause
    cout << "Hit c to continue " << endl;
    cin >> rtn;  */
```

```
// create a loop to determine the max # of
// solutions to f(x+a)+f(x)=b for the current value of a
maxsolutions = 0;
for (c1 = 0; c1 < twotoN; c1++) //c1 gives row of Hf
{
    c2 = c1 + 1;
    Sctr = 1;
    while (c2 < twotoN)
    {
        // loop to compare digits in the vectors
        c3 = 0;
        flag2 = true;
        while (flag2)
        {
            if (Hf[c1][c3] != Hf[c2][c3])
                flag2 = false; // vectors are not equal; move to next row
            else c3++;
            if (c3 == n)
            {
                // vectors are equal; increment Sctr and move to next row
                flag2 = false;
                Sctr = Sctr++;
            } // end if
        } // end while flag2
        c2++;
    } // end while c2
    if (Sctr > maxsolutions)
```

```
            maxsolutions = Sctr;
    } // end for c1

    cout << "Max # solutions = " << maxsolutions << endl;

    } // end for i

} // end function Find_Haf

/*****************************************************************************
Function Name: Binary_Rep
Purpose: Finds the binary representation of a number and returns it.
Called by: create_f_map, Find_Haf
Functions Called: none
*****************************************************************************/
void Binary_Rep(short int binaryrep[], long int num)
{
    // local variables
    short int j, power;
    long int temp1;

    temp1 = num;
    for (j = 0; j < n; j++)
    {
        power = n-j-1;
            if (temp1 >= (int) pow(2,power))
    {
            binaryrep[j] = 1;
```

```
            temp1 = temp1 - (int) pow(2,power);
    } // end if
        else
            binaryrep[j] = 0;
    } // end for j

} // end function Binary_Rep

/*****************************************************************************
Function Name: Value_from_Binary
Purpose: Takes a binary vector and returns the value that it represents.
Called by: Find_Haf
Functions Called: none
*****************************************************************************/
long int Value_from_Binary(short int vector[])
{
    // local variables
    short int j;
    long int value;

    value = 0;
    for (j = 1; j <= n; j++)
        value = value + (int) pow(2,(n-j))*vector[j-1];
    // end for j
    return value;

} // end function Value_from_Binary
```

C.2.2 SolutionCountervfile.cpp

This program is essentially the same as that in C.2.1, only it has been modified to run for multiple power functions $f(x) = x^d$ and to read the potential values for d in from a file.

```
// This is the main project file for a VC++ application project
/*****************************************************************************
Program: SolutionCountervFile.cpp
Author:  Mandi Maxwell
Purpose: To find the number of solutions of the equation f(x+a)+f(x)=b over
the field GF(2^n).
Input from:  File
Output to:   File
Functions used: create_alpha_table, create_f_map, Find_Haf

Directions: You must hard code in the input and output file names and the
values for the following constants before you run the program.

            n: as in GF(2^n)
            twotoN: which is 2^n

            You must also appropriately define the primitive vector to
            correspond to the primitive polynomial for the given field GF(2^n).
*****************************************************************************/

// preprocessor directives:
#include "stdafx.h"
#include <iostream>
#include <string>
#include <iomanip>
```

```
#include <math.h>
#include <cmath>
#include <functional>
#include <fstream>

using namespace std;

// Named constant declaration:
const short int n = 13;
const long int twotoN = 8192;

// Let F be the function F(x)=x^d over GF(2^n);  0 < d < 2^n - 1

void main()
{
    ifstream in;
    ofstream out;

    // open the input file
    in.open("input_data.txt");
    // open the output file
    out.open("results.txt");

    cout << "2 raised to the n is " << twotoN << endl;

    // Local variable declarations:
    short int primitive[n];      // vector rep of primitive poly
    short int Table[twotoN-1][n];    // table of reps of powers on alpha
```

```
    short int fMap[twotoN][n];    // matrix rep of the map f; rows are f(row)
    short int alphazero[n];       // vector rep of alpha^0
    short int alphaToPwr[n];      // vector rep of alpha^pwr
    long int i,j;
    long int d, maxsolutions;

    // function prototypes:
    void create_alpha_table(short int alphazero[], short int Table[][n],
                            short int primitive[]);
    void create_f_map(short int alphazero[], short int primitive[],
     short int fMap[][n], short int Table[][n], long int d);
    long int Find_Haf(short int fMap[][n], long int d);


    // zeroing out matrices
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < twotoN; j++)
            fMap[j][i] = 0;
        for (j = 0; j < twotoN-1; j++)
            Table[j][i] = 0;
        primitive[i] = 0;
        alphaToPwr[i] = 0;
    } //end for i

// define the row vector of the primitive polynomial; powers
// are descending currently it is defined for n=13; x^13+x^4+x^3+x+1=0
primitive[8] = 1;
```

```
primitive[9] = 1;
primitive[11] = 1;
primitive[12] = 1;

// define the vector rep of alpha to the zero power
for (i = 0; i < n-1; i++)
    alphazero[i] = 0;
alphazero[n-1] = 1;

// call function to create Table
create_alpha_table(alphazero, Table, primitive);

/*// display Table
for (i = 0; i < twotoN-1; i++)
{
    for (j = 0; j < n; j++)
        cout << setw(1) << Table[i][j] << "  ";
    cout << endl;
} */

while (!in.eof())
{
    // read the input
    in >> d;

// call a function to create the map for function f
create_f_map(alphazero, primitive, fMap, Table, d);
```

```
// call a function to create and display Ha(f) for all a in the vector space
maxsolutions = Find_Haf(fMap, d);

// write out something to the output file
out << "Number of solutions for d = " << d << " is: " << maxsolutions << endl;
}
in.close();
out.close();

} // end function main

/*****************************************************************************
Function Name: create_alpha_table
Purpose: This function creates the table of reps of powers on alpha.
         Note: multiplying by alpha is the same as shifting every entry 1 unit
         to the left and adding the primitive in at the end if a 1 "falls off"
         the left side so that is how the entries of the table are generated
Called by: main
Functions Called: none
*****************************************************************************/
void create_alpha_table(short int alphazero[], short int Table[][n],
                        short int primitive[])
{
    // local variables
    short int temp[n];
    short int col, i, j, l;
    long int pwr;
```

```
    // initialize variables
    for (i = 0; i < n; i++)
    {
        temp[i] = 0;
        Table[0][i] = alphazero[i];
    }

    // Note: Table[pwr][] is actually alpha to the pwr+1

    for (pwr = 1; pwr < twotoN-1; pwr++)
    {
        for (j = 0; j < n; j++)
            temp[j]  = Table[pwr-1][j];

        for (col = 0; col < n; col++)
        {
            Table[pwr][col] = temp[col+1];
        } // end for col
        Table[pwr][n-1] = 0;
        if (temp[0] == 1)
            for (l = 0; l < n; l++)
                Table[pwr][l] = (Table[pwr][l] + primitive[l]) % 2;

    } // end for pwr

} // end function create_alpha_table

/*****************************************************************************
```

```
Function Name: create_f_map
Purpose: Defines a matrix for function f (which is over the vector space
         GF(2)^n) such that each row corresponds to the vector that the binary
         representation of this row maps to under map f.
Called by: main
Functions Called: Binary_Rep, power_raiser
*****************************************************************************/
void create_f_map(short int alphazero[], short int primitive[],
short int fMap[][n], short int Table[][n], long int d)
{
    // function prototypes
    void Binary_Rep(short int binaryrep[], long int num);
    void power_raiser(short int primitive[], short int alphaToPwr[],
     long int alphapwr);

    // local variables
    short int i, j, c;
    long int exponent, alpha, temp, alphapwr;
    long int r;
    short int binaryrep[n], alphaToPwr[n];
    bool notfound;

    // initialize variables
    alpha = 1;
    notfound = true;
    exponent = 1;

    // initialize 1st 2 rows of fMap and binaryrep
```

```
    for (i = 0; i < n; i++)
    {
        binaryrep[i] = 0;
        fMap[0][i] = 0;
        if (i < n-1)
            fMap[1][i] = 0;
        else
            fMap[1][i] = 1;
    } // end for

// main definition
for (r = 2; r < twotoN; r++)
{
    // find binary rep of row r
    Binary_Rep(binaryrep, r);

    /*cout << "binary rep for " << r << " is ";
    for (j = 0; j < n; j++)
        cout << binaryrep[j];
    cout << endl; */

    // search the table for this binary rep
    exponent = 0;
    notfound = true;
    while (notfound)
    {
        for (c = 0; c < n; c++)
        {
```

```
            if (Table[exponent][c] != binaryrep[c])
            {
                notfound = true;
                exponent++;
                break;
            } // end
        } // end for c
        if (c == n)
        {
            notfound = false;
            alpha = exponent;
        }
    } // end while

    // once the binary rep has been found we want to assign the value that it
    // maps to under the map F to our map f
    for (i = 0; i < n; i++)
        alphaToPwr[i] = alphazero[i];

    alphapwr = alpha*d;
    power_raiser(primitive, alphaToPwr, alphapwr);

    for (c = 0; c < n; c++)
        fMap[r][c] = alphaToPwr[c];

} // end for r

// display fMap
```

```
/* cout << "fMap " << endl;
for (r = 0; r < twotoN; r++)
{
    for (c = 0; c < n; c++)
        cout << fMap[r][c] << " ";
    cout << endl;
} */

} // end function create_f_map

/*********************************************************************
Function Name: power_raiser
Purpose: This function returns alpha raised to a given power.
Called by: create_f_map
Functions Called: none
*********************************************************************/
void power_raiser(short int primitive[], short int alphaToPwr[],
long int alphapwr)
{
    // local variables
    short int temp[n];
    short int col, i, j;
    long int pwr;

    // Note: end result of the loop is that we get alpha to the pwr+1
    for (pwr = 0; pwr < alphapwr; pwr++)
    {
        for (i = 0; i < n; i++)
```

```
            temp[i] = alphaToPwr[i]; // stores alpha to pwr in temp

        for (j = 0; j < n-1; j++)
            alphaToPwr[j]  = temp[j+1];
        alphaToPwr[n-1] = 0;

        if (temp[0] == 1)
            for (col = 0; col < n; col++)
                alphaToPwr[col] = (alphaToPwr[col] + primitive[col]) % 2;
    } // end for pwr

} // end function power_raiser

/*****************************************************************************
Function Name: Find_Haf
Purpose: To find and calculate the size of Ha(f) for all a in the vector space.
Called by: main
Functions Called: Value_from_Binary, Binary_Rep
*****************************************************************************/
long int Find_Haf(short int fMap[][n], long int d)
{
    // function prototypes
    long int Value_from_Binary(short int vector[]);
    void Binary_Rep(short int binaryrep[], long int num);

    // local variables
    short int a[n], x[n], temp[n], Hf[twotoN][n];
    short int Hfsize, digit, p;
```

```
long int i, j, ctr, r, value;
short int spot, c;
long int c1, c2, c3, Sctr, maxsolutions;
bool flag, flag2;
char rtn;

// initialize variables
Hfsize = 0;
value = 0;

for (i = 0; i < twotoN; i++)
{
    Binary_Rep(a,i); //finds binary rep of i and stores in vector a
    Hfsize = 0;
    for (j = 0; j < twotoN; j++)
    {
        Binary_Rep(x,j); //finds binary rep of j and stores in vector x
        for (spot = 0; spot < n; spot++)
            temp[spot] = (x[spot] + a[spot]) % 2; //calculates x+a
        // end for spot
        value = Value_from_Binary(temp);
        for (c = 0; c < n; c++)    // find f(x+a)+f(x)
            Hf[j][c] = (fMap[value][c] + fMap[j][c]) % 2;

        // loop to calculate the number of different values in H(f)
        ctr = 0;
        while (ctr < j)
        {
```

```
        // loop to compare digits in the vectors
        digit = 0;
        flag = true;
        while (flag)
        {
            if (Hf[ctr][digit] != Hf[j][digit])
            {
                // not equal so increment counter and keep searching
                flag = false;
                ctr++;
            } //end if
            else digit++;
            if (digit == n)
            {
                // equal so it's not a new vector; quit searching
                flag = false;
                ctr = j + 100;
            } // end if
        } // end while flag
    } // end while ctr
    if (ctr == j)
        Hfsize = Hfsize++; // none match; j is a new vector

} // end for j

// Display a and Ha(f) on the screen
/* cout << "a = ";
for (r = 0; r < n; r++)
```

```
        cout << a[r] << " ";
    cout << endl; */
/*  cout << "H = " << endl;
    for (r = 0; r < twotoN; r++)
    {
        for (p = 0; p < n; p++)
            cout << Hf[r][p] << " ";
        cout << endl;
    } // end for r */
    // cout << "Size of Ha(f) = " << Hfsize << endl;

/*  //pause
    cout << "Hit c to continue " << endl;
    cin >> rtn;  */
} // end for i

// create a loop to determine the max # of
// solutions to f(x+a)+f(x)=b for a given a
maxsolutions = 0;
for (c1 = 0; c1 < twotoN; c1++) //c1 gives row of Hf
{
    c2 = c1 + 1;
    Sctr = 1;
    while (c2 < twotoN)
    {
        // loop to compare digits in the vectors
        c3 = 0;
        flag2 = true;
```

```
            while (flag2)
            {
                if (Hf[c1][c3] != Hf[c2][c3])
                    flag2 = false; // vectors are not equal; move to next row
                else c3++;
                if (c3 == n)
                {
                    // vectors are equal; increment Sctr and move to next row
                    flag2 = false;
                    Sctr = Sctr++;
                } // end if
            } // end while flag2
            c2++;
        } // end while c2
        if (Sctr > maxsolutions)
            maxsolutions = Sctr;
    } // end for c1

    cout << "d = " << d << endl;
    cout << "Max # solutions = " << maxsolutions << endl;

    return maxsolutions;

} // end function Find_Haf

/*************************************************************************
Function Name: Binary_Rep
Purpose: Finds the binary representation of a number and returns it.
```

```
Called by: create_f_map, Find_Haf
Functions Called: none
**************************************************************************/
void Binary_Rep(short int binaryrep[], long int num)
{
    // local variables
    short int j, power;
    long int temp1;

    temp1 = num;
    for (j = 0; j < n; j++)
    {
        power = n-j-1;
        if (temp1 >= (int) pow(2,power))
    {
            binaryrep[j] = 1;
            temp1 = temp1 - (int) pow(2,power);
    } // end if
        else
            binaryrep[j] = 0;
    } // end for j

} // end function Binary_Rep

/**************************************************************************
Function Name: Value_from_Binary
Purpose: Takes a binary vector and returns the value that it represents.
Called by: Find_Haf
```

```
Functions Called: none
*************************************************************************/
long int Value_from_Binary(short int vector[])
{
    // local variables
    short int j;
    long int value;

    value = 0;
    for (j = 1; j <= n; j++)
        value = value + (int) pow(2,(n-j))*vector[j-1];
    // end for j
    return value;

} // end function Value_from_Binary
```

C.3 Determining the incidence structure

```
// This is the main project file for a VC++ application project
/*************************************************************************
Program: AssociationSchemes.cpp
Author:  Mandi Maxwell

Purpose: To determine the association scheme for a given power map over the
         field GF(2^n).  Points (x,a) and Blocks (y,b) are in relation 1 if
         f(x+y) = a+b.  x is defined to be the 1st n digits in the binary
         representation of Point, a is given by the last n digits in the
         binary representation of Point.  Likewise, y is the first n digits in
         the binary rep of Block, b is the last n digits in the bianry rep of
```

```
        block.  The output of the program is written to a user specified
        output file in the form (Point, Block) for those Point-Block pairs
        satisfying the above equation.  (Note: if the output from this
        program will be used as input to the Search Scheme program,
        the output should instead be written to the user specified output
        file in the alternate (commented out) form which creates a single
        column of numbers with no formatting characters.

Input from:  Keyboard
Output to:   File  *** Note: the file name for this file must be hard coded
                       into the program before it is run.

Functions used: create_alpha_table, create_f_map, power_raiser

Directions: You must hard code in the output file name (within the code for
the create_AS_Matrix function) and the values for the following
constants before you run the program.

            n: as in GF(2^n)
            d: as in f(x)= x^d
            twotoN: which is 2^n
            size: which is 2^(2n)

            You must also appropriately define the primitive vector to
            correspond to the primitive polynomial for the given field GF(2^n).
*************************************************************************/

#include "stdafx.h"
```

```
#using <mscorlib.dll>
using namespace System;

// preprocessor directives:
#include <iostream>
#include <string>
#include <iomanip>
#include <math.h>
#include <cmath>
#include <functional>
#include <fstream>

using namespace std;

// Named constant declaration:
const short int n = 4;
const long int twotoN = 16;
const long int size = 256;

// Let F be the function F(x)=x^d over GF(2^n);  0 < d < 2^n - 1
const short int d = 3;

void main()
{
    cout << "2 raised to the n is " << twotoN << endl;

    // Local variable declarations:
    short int primitive[n];      // vector rep of primitive poly
```

```
    short int Table[twotoN-1][n];    // table of reps of powers on alpha
    short int fMap[twotoN][n];   // matrix rep of the map f; rows are f(row)
    short int alphazero[n];      // vector rep of alpha^0
    short int alphaToPwr[n];     // vector rep of alpha^pwr
    long int i,j;

    // function prototypes:
    void create_alpha_table(short int alphazero[], short int Table[][n],
                            short int primitive[]);
    void create_f_map(short int alphazero[], short int primitive[],
     short int fMap[][n], short int Table[][n], long int d);
    void create_AS_matrix(short int fMap[][n]);


    // zeroing out matrices
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < twotoN; j++)
            fMap[j][i] = 0;
        for (j = 0; j < twotoN-1; j++)
            Table[j][i] = 0;
        primitive[i] = 0;
        alphaToPwr[i] = 0;
    } //end for i

/**************DEFINE PRIMITIVE POLYNOMIAL VECTOR*****************
*****************************************************************/
// define the row vector of the primitive polynomial; powers
```

```
// are descending currently it is defined for n=5; x^5+x^2+1=0
primitive[2] = 1;  //n=5
primitive[4] = 1;
    //primitive[1]=1; n=3
    //primitive[2]=1;
    //primitive[2]=1;  n=4
    //primitive[3]=1;
/*********************************************************************
*********************************************************************/

// define the vector rep of alpha to the zero power
for (i = 0; i < n-1; i++)
    alphazero[i] = 0;
alphazero[n-1] = 1;

// call function to create Table
create_alpha_table(alphazero, Table, primitive);

/*// display Table
for (i = 0; i < twotoN-1; i++)
{
    for (j = 0; j < n; j++)
        cout << setw(1) << Table[i][j] << "  ";
    cout << endl;
} */

// call a function to create the map for function f
create_f_map(alphazero, primitive, fMap, Table, d);
```

```
// call a function to generate the Association Scheme Matrix
create_AS_matrix(fMap);

} // end function main

/*****************************************************************************
Function Name: create_alpha_table
Purpose: This function creates the table of reps of powers on alpha.
         Note: multiplying by alpha is the same as shifting every entry 1 unit
         to the left and adding the primitive in at the end if a 1 "falls off"
         the left side so that is how the entries of the table are generated
Called by: main
Functions Called: none
*****************************************************************************/
void create_alpha_table(short int alphazero[], short int Table[][n],
                        short int primitive[])
{
    // local variables
    short int temp[n];
    short int col, i, j, l;
    long int pwr;

    // initialize variables
    for (i = 0; i < n; i++)
    {
        temp[i] = 0;
        Table[0][i] = alphazero[i];
```

```
    }

    // Note: Table[pwr][] is actually alpha to the pwr+1

    for (pwr = 1; pwr < twotoN-1; pwr++)
    {
        for (j = 0; j < n; j++)
            temp[j]  = Table[pwr-1][j];

        for (col = 0; col < n; col++)
        {
            Table[pwr][col] = temp[col+1];
        } // end for col
        Table[pwr][n-1] = 0;
        if (temp[0] == 1)
            for (l = 0; l < n; l++)
                Table[pwr][l] = (Table[pwr][l] + primitive[l]) % 2;

    } // end for pwr

} // end function create_alpha_table

/*****************************************************************************
Function Name: create_f_map
Purpose: Defines a matrix for function f (which is over the vector space
         GF(2)^n) such that each row corresponds to the vector that the binary
         representation of this row maps to under map f.
Called by: main
```

```
Functions Called: Binary_Rep, power_raiser
*****************************************************************************/
void create_f_map(short int alphazero[], short int primitive[],
short int fMap[][n], short int Table[][n], long int d)
{
    // function prototypes
    void Binary_Rep(short int binaryrep[], long int num, short int length);
    void power_raiser(short int primitive[], short int alphaToPwr[],
     long int alphapwr);


    // local variables
    short int i, j, c;
    long int exponent, alpha, temp, alphapwr;
    long int r;
    short int binaryrep[n], alphaToPwr[n];
    bool notfound;

    // initialize variables
    alpha = 1;
    notfound = true;
    exponent = 1;

    // initialize 1st 2 rows of fMap and binaryrep
    for (i = 0; i < n; i++)
    {
        binaryrep[i] = 0;
        fMap[0][i] = 0;
```

```
        if (i < n-1)
            fMap[1][i] = 0;
        else
            fMap[1][i] = 1;
    } // end for

// main definition
for (r = 2; r < twotoN; r++)
{
    // find binary rep of row r
    Binary_Rep(binaryrep, r, n);

    /*cout << "binary rep for " << r << " is ";
    for (j = 0; j < n; j++)
        cout << binaryrep[j];
    cout << endl; */

    // search the table for this binary rep
    exponent = 0;
    notfound = true;
    while (notfound)
    {
        for (c = 0; c < n; c++)
        {
            if (Table[exponent][c] != binaryrep[c])
            {
                notfound = true;
                exponent++;
```

```
                break;
            } // end
        } // end for c
        if (c == n)
        {
            notfound = false;
            alpha = exponent;
        }
    } // end while

    // once the binary rep has been found we want to assign the value that it
    // maps to under the map F to our map f
    for (i = 0; i < n; i++)
        alphaToPwr[i] = alphazero[i];

    alphapwr = alpha*d;
    power_raiser(primitive, alphaToPwr, alphapwr);

    for (c = 0; c < n; c++)
        fMap[r][c] = alphaToPwr[c];

} // end for r

// display fMap
 cout << "fMap " << endl;
for (r = 0; r < twotoN; r++)
{
    for (c = 0; c < n; c++)
```

```
        cout << fMap[r][c] << " ";
    cout << endl;
}

} // end function create_f_map

/*****************************************************************************
Function Name: create_AS_matrix
Purpose: Creates a 0-1 matrix defining the association scheme and prints to a
file the row and column coords of associated points and blocks.
Called by: main
Functions Called: Binary_Rep, Value_from_Binary
*****************************************************************************/
void create_AS_matrix(short int fMap[][n])
{
    ofstream out;

    /***************SPECIFY OUTPUT FILE*********************************
    *******************************************************************/
    // open the output file
    out.open("inputGF(16)d3.txt");
    /*****************************************************************
    *****************************************************************/

    // function prototypes
    void Binary_Rep(short int binaryrep[], long int num, short int length);
    long int Value_from_Binary(short int vector[]);
```

```
// local variables
short int i, j;
long int vertex, block, xyrow;
short int binrepVert[2*n], binrepBlok[2*n], sumxy[n], sumab[n], fofxy[n];
//short int ASMatrix[size][size];
bool match;
char rtn;

// initialize variables
match = true;

// initialize vectors
for (i = 0; i < 2*n; i++)
{
    binrepVert[i] = 0;
    binrepBlok[i] = 0;
} // end for
for (i = 0; i < n; i++)
{
    sumxy[i] = 0;
    sumab[i] = 0;
} // end for

/*// zero out ASMatrix
for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
        ASMatrix[i][j] = 0;
// end for */
```

```
    // write function and format to output file
    //out << "Power function f(x) = x^" << d << endl;
    //out << "Output is of the form (Vertex, Block)" << endl;

// main definition
for (vertex = 0; vertex < size; vertex++)
{
    for (block = 0; block < size; block++)
    {
        // find binary reps of vertex and block
        Binary_Rep(binrepVert, vertex, 2*n);
        Binary_Rep(binrepBlok, block, 2*n);

    cout << "binary rep for " << vertex << " is ";
    for (j = 0; j < 2*n; j++)
        cout << binrepVert[j];
    cout << endl;

    cout << "binary rep for " << block << " is ";
    for (j = 0; j < 2*n; j++)
        cout << binrepBlok[j];
    cout << endl;

        // calculate x+y and a+b for point: (x,a); block: (y,b)
        // x corresponds to the 1st n digits of binrepVert, a the last n
        // y corresponds to the 1st n digits of binrepBlok, b the last n
        for (i = 0; i < n; i++)
```

```
{
    sumxy[i] = (binrepVert[i] + binrepBlok[i]) % 2;
    sumab[i] = (binrepVert[i+n] + binrepBlok[i+n]) % 2;
} // end for i

// find f(x+y); row in fMap corresponding to the binary rep of x+y
xyrow = Value_from_Binary(sumxy);
for (i = 0; i < n; i++)
    fofxy[i] = fMap[xyrow][i];

// display f(x+y) and a+b
cout << "f(x+y) is ";
for (i = 0; i < n; i++)
    cout << fofxy[i];
cout << " and a+b is ";
for (i = 0; i < n; i++)
    cout << sumab[i];
cout << endl;

//pause
//cout << "hit c to continue " << endl;
//cin >> rtn;

// check to see if f(x+y) = a+b
match = true;
j = 0;
while (match && j < n)
{
```

```
            if (fofxy[j] == sumab[j])
                j++;
            else
                match = false;
        } // end while

        /*WRITING TO OUTPUT FILE - MAKE SURE THE CORRECT FORMAT IS BEING USED*
        ************************************************************************/
        if (j == n)
        {
            // the point and block are incident
            //ASMatrix[vertex][block] = 1;
            //out << "(" << vertex << "," << block << ")" << endl;
            out << vertex << endl; // format required when the data will
            out << block << endl; // be input for a SearchScheme program
        }
        //else
        //  ASMatrix[vertex][block] = 0;
        // end if-else

    } // end for block
}// end for vertex

out.close();

} // end function create_AS_matrix

/*********************************************************************
```

```
Function Name: power_raiser
Purpose: This function returns alpha raised to a given power.
Called by: create_f_map
Functions Called: none
**********************************************************************/
void power_raiser(short int primitive[], short int alphaToPwr[],
long int alphapwr)
{
    // local variables
    short int temp[n];
    short int col, i, j;
    long int pwr;

    // Note: end result of the loop is that we get alpha to the pwr+1
    for (pwr = 0; pwr < alphapwr; pwr++)
    {
        for (i = 0; i < n; i++)
            temp[i] = alphaToPwr[i]; // stores alpha to pwr in temp

        for (j = 0; j < n-1; j++)
            alphaToPwr[j]  = temp[j+1];
        alphaToPwr[n-1] = 0;

        if (temp[0] == 1)
            for (col = 0; col < n; col++)
                alphaToPwr[col] = (alphaToPwr[col] + primitive[col]) % 2;
    } // end for pwr
```

```
} // end function power_raiser

/*************************************************************************
Function Name: Binary_Rep
Purpose: Finds the binary representation of a number and returns it.
Called by: create_f_map, create_AS_matrix
Functions Called: none
*************************************************************************/
void Binary_Rep(short int binaryrep[], long int num, short int length)
{
    // local variables
    short int j, power;
    long int temp1;

    temp1 = num;
    for (j = 0; j < length; j++)
    {
        power = length-j-1;
        if (temp1 >= (int) pow(2,power))
        {
            binaryrep[j] = 1;
            temp1 = temp1 - (int) pow(2,power);
        } // end if
        else
            binaryrep[j] = 0;
    } // end for j

} // end function Binary_Rep
```

```
/*************************************************************************
Function Name: Value_from_Binary
Purpose: Takes a binary vector and returns the value that it represents.
Called by: create_AS_matrix
Functions Called: none
*************************************************************************/
long int Value_from_Binary(short int vector[])
{
    // local variables
    short int j;
    long int value;

    value = 0;
    for (j = 1; j <= n; j++)
        value = value + (int) pow(2,(n-j))*vector[j-1];
    // end for j
    return value;

} // end function Value_from_Binary
```

C.4 Finding common neighborhoods

```
// This is the main project file for a VC++ application project
/*************************************************************************
Program: SearchScheme.cpp
Author:  Mandi Maxwell
Purpose: To find the "common neighborhood" of k points in the Association
Scheme corresponding to a given power function over a given field
```

GF(2^n). By commom neighborhood, we mean all points that are collinear with the k user specified points.

Input from: File *** Note: the format for this file is a single column of whole numbers. The binary representation of the 1st number gives the coords of the Point with x = 1st n binary digits and a = 2nd n binary digits. The binary representation of the number on the 2nd line gives the corresponding Block in the Association Scheme with y = 1st n binary digits and b = 2nd n binary digits. This file name also must be hard coded into the program before it is run. Generally, this file is created as the (appropriately formatted) output of the Association Scheme.cpp program.

Output to: File *** Note: the file name for this file must be hard coded into the program before it is run.

Functions used: create_PBP_Matrix, find_neighborhood, get_points, nbr_search

Directions: You must hard code in the file names and the values for the following constants before you run the program.

n: as in GF(2^n)
twotoN: which is 2^n
maxnbrhood: which is 2^(2n)
size: which is 2^(3n)
maxNumPoints: max # points to find the neighborhood of
maxNumNbrs: max size of a common neighborhood of points

```
**************************************************************************/

#include "stdafx.h"
#using <mscorlib.dll>
using namespace System;

// preprocessor directives:
#include <iostream>
#include <string>
#include <iomanip>
#include <math.h>
#include <cmath>
#include <functional>
#include <fstream>

using namespace std;

// Named constant declaration:
const short int n = 5;
const long int twotoN = 32; // 2toN = (int) pow(2,n)
const long int maxnbrhood = 1024; // maxnbrhood = (int) pow(twotoN,2)
const long int size = 32768; // size = (int) pow(twotoN,3)
const short int maxNumPoints = 16; //max # points to find the neighborhood of
const short int maxNumNbrs = 130;  // max size of the common neighborhood

// Let F be the function F(x)=x^d over GF(2^n);  0 < d < 2^n - 1

void main()
```

```
{
    // Local variable declarations:
    short int pointVector[maxNumPoints]; // vector of user determined points
                                         // whose common neighborhood will be found
    short int PointBlockpairs[size][2]; // Matrix of point-block pairs
    short int pointnbrs[maxnbrhood][maxNumPoints]; // matrix in which each
    // column is the sorted vector of neighbors for the
    //corresponding user specified point
    short int neighbors[maxnbrhood];  // vector of neighborhood of point
    short int commonnbrs[maxnbrhood]; // common neighbors of k pts vector
    short int d; // the degree of the power function in question
    long int i, j, k, m;
    short int element;
    int numnbrs[maxNumPoints]; // vector of # neighbors for points
    int numcnbrs;
    short int numpoints; // # of points used to determine common neighborhood
    char pbfilename[25]; // input file containing point-block pairs
    char outputfile[65]; // output file name
    bool found;
    short int rtn;

    // function prototypes:
    void create_PBP_Matrix(short int PointBlockpairs[][2], char filename[]);
    short int get_points(short int pointVector[]);
    int find_neighborhood(short int PointBlockpairs[][2],
     short int neighbors[], short int point);
    bool nbr_search(int nbrhoodsize, short int element,
     short int neighbors[]);
```

```
rtn = 1;
while (rtn != 0)
{
    // zeroing out matrices
    for (i = 0; i < size; i++)
    {
        PointBlockpairs[i][0] = 0;
        PointBlockpairs[i][1] = 0;
    }

    for (i = 0; i < maxnbrhood; i++)
    {
        neighbors[i] = -1;
        commonnbrs[i] = -1;
        for (j = 0; j < maxNumPoints; j++)
            pointnbrs[i][j] = -1;
    } // end for i

    for (i = 0; i < maxNumPoints; i++)
    {
        pointVector[i] = -1;
        numnbrs[i] = -1;
    } // end for i

    ofstream out;

    // get the exponent, d, for the power function from the user
```

```
cout << "For the power function f(x)=x^d over GF(2^" << n;
cout << "), enter the value of d: ";
cin >> d;

// get name of file containing point block pairs for this function
cout << "Enter the name of Association Scheme file for this function: ";
cin.width(25);
cin >> pbfilename;
cout << pbfilename << endl;

// call a function to open this file and read its contents into
// the PointBlockpairs matrix
create_PBP_Matrix(PointBlockpairs, pbfilename);

// get the name of the output file
cout << "Enter the name of the output file: ";
cin.width(65);
cin >> outputfile;

// open the output file
out.open(outputfile);

// get the points whose common neighborhood will be analyzed
numpoints = get_points(pointVector);

// loop to find the neighborhood of the user specified points
for (k = 0; k < numpoints; k++)
{
```

```
    // call a function to return the neighborhood of the kth point
    numnbrs[k] = find_neighborhood(PointBlockpairs, neighbors,
     pointVector[k]);

    //write this to the kth column of the pointnbrs matrix
    for (m = 0; m < numnbrs[k]; m++)
        pointnbrs[m][k] = neighbors[m];

    // write neighborhood of kth point to screen and to a file
    cout << endl;
    cout << "The neighborhood for the point " ;
    cout << pointVector[k] << " is: " << endl;
    cout << "{" << pointnbrs[0][k];
    //out << endl;
    //out << "The neighborhood for the point ";
    //out << pointVector[k] << " is: " << endl;
    //out << "{" << pointnbrs[0][k];
    for (i = 1; i < numnbrs[k]; i++)
    {
        cout << ", " << pointnbrs[i][k];
    //  out << ", " << pointnbrs[i][k];
    } // end for
    cout << "}" << endl;
    //out << "}" << endl;
    //out << "The neighborhood of " << pointVector[k] << " contains ";
    //out << numnbrs[k] << " points." << endl;

} // end for k
```

```
// find the intersection of all of these neighborhoods
numcnbrs = 0;
for (i = 0; i < numnbrs[0]; i++)
{
    element = pointnbrs[i][0];

    // check to see if this is a neighbor of the other points
    for (k = 1; k < numpoints; k++)
    {
        found = false;

        for (j = 0; j < numnbrs[k]; j++)
        {
            // read the kth column of pointnbrs into neighbor vector
            neighbors[j]=pointnbrs[j][k];
        } // end for j

        // call function to determine whether element is nbr of pt k
        found = nbr_search(numnbrs[k], element, neighbors);

        // if element is not a neighbor of point k, break
        if (found == false)
            break;
    } // end for k
    if (found)
    {
        // element is a common neighbor to all of the points
```

```
        commonnbrs[numcnbrs] = element;
        numcnbrs++;
    } // end if found

} // end for i

// write common neighborhood of k points to screen and to a file
cout << endl;
cout << "The common neighborhood of the " << numpoints << " points: ";
for (i = 0; i < numpoints - 1; i++)
    cout << pointVector[i]  << ", ";
cout << pointVector[numpoints-1] << " is: " << endl;
cout << "{" << commonnbrs[0];
//out << endl;
//out << "The common neighborhood of the " << numpoints << " points: ";
//for (i = 0; i < numpoints - 1; i++)
//  out << pointVector[i]  << ", ";
//out << pointVector[numpoints-1] << " is: " << endl;
out << commonnbrs[0] << endl;
for (i = 1; i < numcnbrs; i++)
{
    cout << ", " << commonnbrs[i];
//  out << ", " << commonnbrs[i];
    out << commonnbrs[i] << endl;
} // end for
cout << "}" << endl;
cout << endl;
//out << "}" << endl;
```

```
    cout << "It contains " << numcnbrs << " points." << endl;

out.close();

cout << "Hit 0 to end " << endl;
cin >> rtn;
} // end while

} // end function main

/*****************************************************************************
Function Name: create_PBP_Matrix
Purpose:    To read the contents of the user specified Association Scheme
matrix into the PointBlockpairs matrix which will be used extensively
in this program.
Called by: main
Functions Called: none
*****************************************************************************/
void create_PBP_Matrix(short int PointBlockpairs[][2], char filename[])
{
    ifstream in;

    // local variables
    int row;
    short int pt, bk;

    // open the input file
    in.open(filename);
```

```
    row = 0;
    do
    {
        // read the input into the PointBlockpairs matrix
        in >> pt;
        in >> bk;
        PointBlockpairs[row][0] = pt;
        PointBlockpairs[row][1] = bk;

        //cout << "(" << PointBlockpairs[row][0] << ",";
        //cout << PointBlockpairs[row][1] << ")" << endl;
        row++;
    } while (!in.eof());

    in.close();

} // end function create_PBP_Matrix

/*****************************************************************************
Function Name: get_points
Purpose:    To get from the user the points whose common neighborhood is
being analyzed and put them into a vector.  It returns the number of
        points used to define the common neighborhood.
Called by: main
Functions Called: none
*****************************************************************************/
short int get_points(short int pointVector[])
```

```
{
    // local variables
    short int i, point, numpoints;

    cout << "Enter the number of points used to determine the common ";
    cout << "neighborhood: ";
    cin >> numpoints;
    cout << endl;

for (i = 0; i < numpoints; i++)
{
    cout << "Enter point #" << i+1 << ": ";
    cin >> point;
    pointVector[i] = point;
} // end for

return numpoints;

} // end function get_ points

/*****************************************************************************
Function Name: find_neighborhood
Purpose:    To query the user for a point, then generate the neighborhood
for that point and return the vector of the point's neighborhood
along with the size of the neighborhood. (i.e. this finds all points
 that are collinear with the user specified point and returns the
 total number of these points along with their names.)
Called by: main
```

```
Functions Called: find_point, sort_vector
*************************************************************************/
int find_neighborhood(short int PointBlockpairs[][2], short int neighbors[],
                      short int point)
{
    // function prototypes
    short int find_point(short int PointBlockpairs[][2], short int point);
    void sort_vector(short int array_to_sort[], int listsize);

    // local variables
    int i, j, k, nindex;
    short int row, block, bkrow, nbr;
    bool newnbr;

    // initialize variables
    nindex = 0;
    block = -1;
    row = -1;
    bkrow = -1;
    nbr = -1;
    for (i = 0; i < maxnbrhood; i++)
        neighbors[i] = -1;

    // search the PointBlockpairs matrix for this point
    row = find_point(PointBlockpairs, point);

    // for each block incident with this point, find the other points
    // incident with this block
```

```
// loop to read in the blocks associated with this point
for (i = 0; i < twotoN; i++)
{
    block = PointBlockpairs[row+i][1];
    if (block != point)
    {
        // consider this block as a point and find the blocks incident on
        // this "point"; search the PointBlockpairs matrix for this "point"
        bkrow = find_point(PointBlockpairs, block);
        // add the blocks incident with "point" to the neighbors vector
        for (j = 0; j < twotoN; j++)
        {
            nbr = PointBlockpairs[bkrow+j][1];
            //check to make sure nbr != point
            if (nbr != point)
            {
                // check to see if it is already included in nbr vector
                newnbr = true;
                for (k = 0; k < nindex; k++)
                {
                    if (nbr == neighbors[k])
                        newnbr = false;
                } // end for k

                if (newnbr == true)
                {
                    // add nbr to the neighbors vector
                    neighbors[nindex] = nbr;
```

```
                        nindex++;
                    } // end if newnbr
                } // end if nbr != point
            } // end for j
        } // end if block != point
    } // end for i

    // call function to sort the neighbors vector
    sort_vector(neighbors, nindex);

    return nindex;

} // end function find_neighborhood */

/*****************************************************************************
Function Name: nbr_search
Purpose:    Searches a vector to for a specific value.
Called by: main
Functions Called: none
*****************************************************************************/
bool nbr_search(int nbrhoodsize, short int element, short int neighbors[])
{
    // local variables
    short int i;
    bool found;

    found = false;
```

```
    for (i = 0; i < nbrhoodsize; i++)
        if (neighbors[i] == element)
        {
            found = true;
            break;
        } // end if

    return found;

} // end nbr_search

/*****************************************************************************
Function Name: find_point
Purpose:    To find the first row in the PointBlockpairs matrix where this
point occurs.  Takes as input the PointBlockpairs matrix and a point
and returns the row in which that point first appears.
Called by: find_neighborhood
Functions Called: none
*****************************************************************************/
short int find_point(short int PointBlockpairs[][2], short int point)
{
    // local variables
    short int row;
    bool notfound;

    // initialize variables
    notfound = true;
    row = 0;
```

```
    // search the PointBlockpairs matrix for the point
    while(notfound)
    {
        if (PointBlockpairs[row][0] == point)
            notfound = false;
        else
            row++;
        //end if-else
    } // end while

    return row;

} // end function find_point

/*************************************************************************
Function Name: sort_vector
Purpose:    To take an unsorted vector and return it with the entries in
            ascending numerical order.
Called by: find_neighborhood
Functions Called: none
*************************************************************************/
void sort_vector(short int array_to_sort[], int listsize)
{
    // local variables
    short int temp;
    int pass, i;
    bool exchanges;
```

```
    // initialize variables
    pass = 1;
    exchanges = true;

    while (pass < listsize && exchanges)
    {
        exchanges = false;
        for (i = 0; i < listsize - pass; i++)
        {
            if (array_to_sort[i] > array_to_sort[i+1])
            {
                temp = array_to_sort[i];
                array_to_sort[i] = array_to_sort[i+1];
                array_to_sort[i+1] = temp;
                exchanges = true;
            } // end if
        } // end for
        pass++;
    } // end while

} // end function sort_vector
```

C.5 Determining neighborhood structures

```
// This is the main project file for a VC++ application project
/*****************************************************************************
Program: MultiPointStructFinder.cpp
Author:  Mandi Maxwell
```

```
Purpose:
Given a user specified number of neighboring points, to find the
        structure determined by the points in their common neighborhood.  This
        program requires input from a file containing incident point-block
        for the user specified power function f(x)=x^d.  This input generally
        comes from a file output from the program AssociationSchemes.cpp.  The
        program writes all maximal cliques containing the user specified points
        to a user specified output file.
Input from:  File and Keyboard
Output to:   File
Functions used: create_Point_Vector, create_PBP_Matrix, get_points,
find_initial_neighborhood, find_neighborhood, nbr_search

Directions: You must hard code the values for the following constants before
you run the program.

            n: as in GF(2^n)
            twotoN: which is 2^n
            maxnbrhood: which is 2^(2n)
            size: which is 2^(3n)
            maxNumPoints: max # points to find the neighborhood of
            maxNumNbrs: max size of a common neighborhood of points
*****************************************************************************/

#include "stdafx.h"
#using <mscorlib.dll>
using namespace System;
```

```
// preprocessor directives:
#include <iostream>
#include <string>
#include <iomanip>
#include <math.h>
#include <cmath>
#include <functional>
#include <fstream>

using namespace std;

// Named constant declaration:
const short int n = 5;
const long int twotoN = 32; // 2toN = (int) pow(2,n);
const long int maxnbrhood = 1024; // maxnbrhood = (int) pow(twotoN,2);
const long int size = 32768; // size = (int) pow(twotoN,3);

const short int maxNumPoints = 20; //max # points to find the neighborhood of
const short int maxNumNbrs = 130;  // max size of the common neighborhood

// global variables
short int initPtNum;
short int NbrhoodSize;
short int initPV[maxNumPoints]; // vector of user determined points whose
                                // common neighborhood will be found
short int PointBlockpairs[size][2]; // Matrix of point-block pairs from file
char outputfile[65]; // output file name
```

```
void main()
{
    // Local variable declarations:
    short int nbrVector[maxnbrhood];    // neighborhood of user specified points
    short int pointVector[maxNumPoints]; // vector of points whose common
                                         // neighborhood will be found
    short int d; // the degree of the power function in question
    short int pt, start, end;
    long int i;
    short int numPoints; // # of points used to determine common neighborhood
    char pbfilename[35]; // input file containing point-block pairs

    char rtn;

    // function prototypes:
    void create_Point_Vector(short int pointVector[], short int nbrVector[],
     short int pointNum, short int start, short int end,
short int numPoints);
    void create_PBP_Matrix(char filename[]);
    short int find_initial_neighborhood(short int nbrVector[]);
    int find_neighborhood(short int neighbors[], short int point);
    bool nbr_search(int nbrhoodsize, short int element, short int neighbors[]);
    short int get_points();

    // zeroing out matrices
    for (i = 0; i < size; i++)
    {
        PointBlockpairs[i][0] = 0;
```

```
    PointBlockpairs[i][1] = 0;
}

for (i = 0; i < maxNumPoints; i++)
{
    initPV[i] = -1;
    pointVector[i] = -1;
    nbrVector[i] = -1;
} // end for i

// get the exponent, d, for the power function from the user
cout << "For the power function f(x)=x^d over GF(2^" << n;
cout << "), enter the value of d: ";
cin >> d;

// get name of file containing point block pairs for this function
cout << "Enter the name of Association Scheme file for this function: ";
cin.width(25);
cin >> pbfilename;
cout << pbfilename << endl;

// call function to read contents of this file into PointBlockpairs matrix
create_PBP_Matrix(pbfilename);

// get the name of the output file
cout << "Enter the name of the output file: ";
cin.width(65);
cin >> outputfile;
```

```
// call function to get from user vertices whose nbrhood will be studied
initPtNum = get_points();

// call function to return common nbrhood of user specified points
NbrhoodSize = find_initial_neighborhood(nbrVector);

// write the common neighborhood of the user specified points to the screen
/*cout << endl;
cout << "The neighborhood for the points is: " << endl;
cout << "{" << nbrVector[0];
for (i = 1; i < NbrhoodSize; i++)
    cout << ", " << nbrVector[i];
cout << "}" << endl;
cin >> rtn;*/

// loop on the possible value for numPoints
for (numPoints = 1; numPoints <= maxNumPoints; numPoints++)
{
    pt = 1;
    start = 0;
    end = NbrhoodSize - numPoints + 1;

    // create all possible point vectors of length numPoints and
    // analyze their structures
    create_Point_Vector(pointVector, nbrVector, pt, start, end, numPoints);

} // end loop on numPoints
```

```
} // end main

/*****************************************************************************
Function Name: create_Point_Vector
Purpose:    To recursively generate all point vectors of a certain length and
to send them into the Analyze_Structures function so the the structure
of the common neighborhoods of these points can be studied.
Called by: main
Functions Called: Analyze_Structures
*****************************************************************************/
void create_Point_Vector(short int pointVector[], short int nbrVector[],
short int ptNum, short int start, short int end,
short int numPoints)
{
    //local variables
    short int ctr, i;
    char rtn;

    //function prototypes
    void Analyze_Structures(short int pointVector[], short int numPoints,
     short int nbrVector[]);

    for (ctr = start; ctr < end; ctr++)
    {
        pointVector[ptNum-1] = nbrVector[ctr];

        if (ptNum < numPoints)
```

```
            create_Point_Vector(pointVector, nbrVector, ptNum+1, ctr+1, end+1,
             numPoints);
        else
        {
            // do analysis of common neighborhood of elements in pointVector
            Analyze_Structures(pointVector, numPoints, nbrVector);

            // for debugging purposes, print out the contents of the point vector
            /*cout << "{" << pointVector[0];
            for (i = 1; i < numPoints; i++)
                cout << ", " << pointVector[i];
            cout << "}" << endl;

            cout << "hit c to continue: ";
            cin >> rtn; */

        } // end else

    } // end for ctr

} // end function create_Point_Vector

/*****************************************************************************
Function Name: create_PBP_Matrix
Purpose:    To read the contents of the user specified Association Scheme
matrix into the PointBlockpairs matrix which will be used
extensively in this program.
Called by: main
```

```
Functions Called: none
*****************************************************************************/
void create_PBP_Matrix(char filename[])
{
    ifstream in;

    // local variables
    int row;
    short int pt, bk;

    // open the input file
    in.open(filename);

    row = 0;
    do
    {
        // read the input into the PointBlockpairs matrix
        in >> pt;
        in >> bk;
        PointBlockpairs[row][0] = pt;
        PointBlockpairs[row][1] = bk;

        //cout << "(" << PointBlockpairs[row][0] << ",";
        //cout << PointBlockpairs[row][1] << ")" << endl;
        row++;
    } while (!in.eof());

    in.close();
```

```
} // end function create_PBP_Matrix

/*****************************************************************************
Function Name: find_initial_neighborhood
Purpose:    To find the common neighborhood of the user specified points.
Called by: main
Functions Called: find_neighborhood, nbr_search
*****************************************************************************/
short int find_initial_neighborhood(short int nbrVector[])
{
    // local variables
    short int neighbors[maxnbrhood];  // vector of neighborhood of a point
    //short int commonnbrs[maxnbrhood]; // vector of nbrs common to all k pts
    int numnbrs[maxNumPoints]; // vector of # neighbors for cor. pt
    short int pointnbrs[maxnbrhood][maxNumPoints]; // matrix in which each
     // column is the sorted vector of neighbors for the
// corresponding user specified point
    int numcnbrs;
    long int i, j, k, m;
    short int element;
    bool found;
    char rtn;

    // function prototypes
    int find_neighborhood(short int neighbors[], short int point);
    bool nbr_search(int nbrhoodsize, short int element, short int neighbors[]);
```

```
// initialize arrays
for (i = 0; i < maxnbrhood; i++)
{
    neighbors[i] = -1;
    //commonnbrs[i] = -1;
    for (j = 0; j < maxNumPoints; j++)
    {
        pointnbrs[i][j] = -1;
        numnbrs[j] = -1;
    } // end for j
} // end for i

// loop to find the neighborhood of the points in the set
for (k = 0; k < initPtNum; k++)
{
    // call a function to return the neighborhood of the kth point
    numnbrs[k] = find_neighborhood(neighbors, initPV[k]);

    //write this to the kth column of the pointnbrs matrix
    for (m = 0; m < maxnbrhood; m++)
        pointnbrs[m][k] = neighbors[m];

} // end for k

// find the intersection of all of these neighborhoods
numcnbrs = 0;
for (i = 0; i < numnbrs[0]; i++)
{
```

```
    element = pointnbrs[i][0];

    // check to see if this is a neighbor of the other points
    for (k = 1; k < initPtNum; k++)
    {
        found = false;

        for (j = 0; j < numnbrs[k]; j++)
        {
            // read the kth column of pointnbrs into neighbor vector
            neighbors[j]=pointnbrs[j][k];
        } // end for j

        // call function to determine whether element is a nbr of pt k
        found = nbr_search(numnbrs[k], element, neighbors);

        // if element is not a neighbor of point k, break
        if (found == false)
            break;
    } // end for k
    if (found)
    {
        // element is a common neighbor to all of the points
        nbrVector[numcnbrs] = element;
        numcnbrs++;
    } // end if found

} // end for i
```

```
    // write the common neighborhood of the k points to the screen
    cout << endl;
    cout << "The common neighborhood of the " << initPtNum << " points: ";
    for (i = 0; i < initPtNum-1; i++)
        cout << initPV[i]  << ", ";
    cout << initPV[initPtNum-1] << " is: " << endl;
    cout << "{" << nbrVector[0];
    for (i = 1; i < numcnbrs; i++)
        cout << ", " << nbrVector[i];
    cout << "}" << endl;
    cout << "It contains " << numcnbrs << " points." << endl;
    cout << "hit c to continue: ";
    cin >> rtn;

    return numcnbrs;

} // end function find_inital_neighborhood

/*****************************************************************************
Function Name: Analyze_Structures
Purpose:    To find the common neighborhood of the points in the PointVector
and analyze the structure of the common neighborhood.
Called by: create_Point_Vector
Functions Called: find_neighborhood, nbr_search, Common_Nbrs,
validate_point_list
*****************************************************************************/
void Analyze_Structures(short int pointVector[], short int numPoints,
```

```
short int nbrVector[])
{
    // local variables
    short int neighbors[maxnbrhood];  // vector of neighborhood of a point
    short int commonnbrs[maxnbrhood]; // vector of nbrs common to all k pts
    int numnbrs[maxNumPoints]; // vector of # neighbors for the cor. pt
    short int pointnbrs[maxnbrhood][maxNumPoints]; // matrix in which each column
     // is the sorted vector of neighbors for the
// corresponding user specified point
    short int vectorA[maxnbrhood], vectorB[maxnbrhood], CNbrsVector[maxnbrhood];
    int numcnbrs;
    long int i, j, k, m, c;
    short int element, sizeCNbrhood;
    bool validpts, found;
    char rtn;

    ofstream out;

    // function prototypes
    int find_neighborhood(short int neighbors[], short int point);
    bool nbr_search(int nbrhoodsize, short int element, short int neighbors[]);
    short int Common_Nbrs(short int vectorA[], short int lengthA,
     short int vectorB[], short int lengthB, short int tempCNbrs[]);
    bool validate_point_list(short int pointVector[], int numnbrs[],
     short int pointnbrs[][maxNumPoints], short int numPoints);

    // initialize arrays
    for (i = 0; i < maxnbrhood; i++)
```

```
{
    neighbors[i] = -1;
    commonnbrs[i] = -1;
    vectorA[i] = -1;
    vectorB[i] = -1;
    CNbrsVector[i] = -1;
    for (j = 0; j < maxNumPoints; j++)
    {
        pointnbrs[i][j] = -1;
        numnbrs[j] = -1;
    } // end for j
} // end for i

// open the output file
out.open(outputfile, ios::app);

// loop to find the neighborhood of the points in the set
for (k = 0; k < numPoints; k++)
{
    // call a function to return the neighborhood of the kth point
    numnbrs[k] = find_neighborhood(neighbors, pointVector[k]);

    //write this to the kth column of the pointnbrs matrix
    for (m = 0; m < maxnbrhood; m++)
        pointnbrs[m][k] = neighbors[m];

} // end for k
```

```
//cout << "Done finding neighbors of the members of pointVector " << endl;
validpts = false;

// check to see if all of the members of pointVector are neighbors
validpts = validate_point_list(pointVector, numnbrs, pointnbrs, numPoints);

if (validpts)
{
    // find the intersection of all of these neighborhoods
    numcnbrs = 0;
    for (i = 0; i < numnbrs[0]; i++)
    {
        element = pointnbrs[i][0];

        // check to see if this is a neighbor of the other points
        for (k = 1; k < numPoints; k++)
        {
            found = false;

            for (j = 0; j < numnbrs[k]; j++)
            {
                // read the kth column of pointnbrs into neighbor vector
                neighbors[j]=pointnbrs[j][k];
            } // end for j

            // call function to determine whether element is a nbr of pt k
            found = nbr_search(numnbrs[k], element, neighbors);
```

```
        // if element is not a neighbor of point k, break
        if (found == false)
            break;
    } // end for k
    if (found)
    {
        // element is a common neighbor to all of the points
        commonnbrs[numcnbrs] = element;
        numcnbrs++;
    } // end if found

} // end for i

// intersect this common neighbor vector with the neighborhood
// of the initial points
sizeCNbrhood = Common_Nbrs(nbrVector, NbrhoodSize, commonnbrs,
 numcnbrs, CNbrsVector);

// is this a new & complete structure; if so, write to file
//cout << "sizeCNbrhood: " << sizeCNbrhood << " common neighbor: ";
//cout << CNbrsVector[0] << " last element of pointVector: ";
//cout << pointVector[numPoints-1] << endl;
//cin >> rtn;

if (CNbrsVector[0] == -1)
{
    // write this set of points to the file
    out << "The set: {";
```

```
        for (c = 0; c < initPtNum; c++)
            out << initPV[c] << ", ";
        for (c = 0; c < numPoints-1; c++)
            out << pointVector[c] << ", ";
        out << pointVector[numPoints-1] << "} forms a complete graph on ";
        out << numPoints+initPtNum << " vertices" << endl;
    } // end if

/*  // find the intersection of all of these neighborhoods
    numcnbrs = 0;
    for (j=0; j < maxnbrhood; j++)
        vectorA[j] = pointnbrs[j][0];
    lengthA = numnbrs[0];

    for (i=1; i < numPoints; i++)
    {
        // read in vectorB
        for (j=0; j < maxnbrhood; j++)
            vectorB[j] = pointnbrs[j][i];

        // find the common neighborhoods of two vectors
        numcnbrs = Common_Nbrs(vectorA, lengthA, vectorB, numnbrs[i],
         tempCNbrs);

        // read in vectorA
        for (j=0; j < maxnbrhood; j++)
            vectorA[j] = tempCNbrs[j];
        lengthA = numcnbrs;
```

```
} // end for i

for (j=0; j < maxnbrhood; j++)
    commonnbrs[j] = tempCNbrs[j];

// write common nbrhood of k points to screen & to a file
cout << endl;
cout << "The common neighborhood of the " << numPoints+1 << " points: ";
for (i = 0; i < numPoints; i++)
    cout << pointVector[i]  << ", ";
cout << pointVector[numPoints] << " is: " << endl;
cout << "{" << commonnbrs[0];
//out << endl;
out << "The common neighborhood of the " << numPoints+1 << " points: ";
for (i = 0; i < numPoints; i++)
    out << pointVector[i]  << ", ";
out << pointVector[numPoints] << " is: " << endl;
out << commonnbrs[0] << endl;
for (i = 1; i < numcnbrs; i++)
{
    cout << ", " << commonnbrs[i];
    out << ", " << commonnbrs[i];
    out << commonnbrs[i] << endl;
} // end for
cout << "}" << endl;
cout << "hit c to continue: ";
cin >> rtn;
```

```
        out << "}" << endl;
        out << "It contains " << numcnbrs << " points." << endl;*/

    } // end if validpts

    out.close();

} // end function Analyze_Structures

/*****************************************************************************
Function Name: find_neighborhood
Purpose:    To query the user for a point, then generate the neighborhood for
            that point and return the vector of the point's neighborhood along
            with the size of the neighborhood. (i.e. this finds all points that
            are collinear with the user specified point and returns the total
            number of these points along with their names.)
Called by: main, Analyze_Structures, find_initial_neighborhood
Functions Called: find_point, sort_vector
*****************************************************************************/
int find_neighborhood(short int neighbors[], short int point)
{
    // function prototypes
    short int find_point(short int point);
    void sort_vector(short int array_to_sort[], int listsize);

    // local variables
    int i, j, k, m, nindex, ctr;
    short int row, block, bkrow, nbr;
```

```
short int tempnbrs[maxnbrhood];
bool newnbr, nomatch;

char rtn;

// initialize variables
nindex = 0;
block = -1;
row = -1;
bkrow = -1;
nbr = -1;
for (i = 0; i < maxnbrhood; i++)
{
    tempnbrs[i] = -1;
    neighbors[i] = -1;
} // end for i

// search the PointBlockpairs matrix for this point
row = find_point(point);

// for each block incident with this point,
// find the other points incident with this block
// loop to read in the blocks associated with this point
for (i = 0; i < twotoN; i++)
{
    block = PointBlockpairs[row+i][1];

    if (block != point)
```

```
{
    // consider this block as a point & find blocks incident on it
    // search the PointBlockpairs matrix for this "point"
    bkrow = find_point(block);
    // add the blocks incident with "point" to the neighbors vector
    for (j = 0; j < twotoN; j++)
    {
        nbr = PointBlockpairs[bkrow+j][1];
        //check to make sure nbr != point
        if (nbr != point)
        {
            // see if it is already included in neighbor vector
            newnbr = true;
            for (k = 0; k < nindex; k++)
            {
                if (nbr == tempnbrs[k])
                    newnbr = false;
            } // end for k

            if (newnbr == true)
            {
                // add nbr to the neighbors vector
                tempnbrs[nindex] = nbr;
                nindex++;
            } // end if newnbr
        } // end if nbr != point
    } // end for j
} // end if block != point
```

```
} // end for i

// remove any elements from initial point list from neighbors vector
ctr = 0;
for (k = 0; k < nindex; k++)
{
    nomatch = true;
    for (m = 0; m < initPtNum; m++)
    {
        if (tempnbrs[k] == initPV[m])
        {
            nomatch = false;
            break;
        } // end if
    } // end for m

    if (nomatch)
    {
        neighbors[ctr] = tempnbrs[k];
        ctr++;
    } // end if

} // end for k

// call function to sort the neighbors vector
sort_vector(neighbors, ctr);
//sort_vector(tempnbrs, nindex);
```

```
/*  cout << "The temp vector contains: {";
    for (k=0; k < nindex-1; k++)
        cout << tempnbrs[k] << ", ";
    cout << tempnbrs[nindex-1] << "}" << endl;

    cout << "The neighbor vector contains: {";
    for (k=0; k < ctr-1; k++)
        cout << neighbors[k] << ", ";
    cout << neighbors[ctr-1] << "}" << endl;
    cin >> rtn; */

    return ctr;

} // end function find_neighborhood */

/*****************************************************************************
Function Name: validate_point_list
Purpose:    Determines whether all the points in the list are neighbors or not.
Called by: Analyze_Structures
Functions Called: nbr_search
*****************************************************************************/
bool validate_point_list(short int pointVector[], int numnbrs[],
                         short int pointnbrs[][maxNumPoints], short int numPoints)
{
    // local variables
    short int i, j, m, c;
    short int element;
    short int nbrs2[maxnbrhood];  // vector of the neighborhood of a point
```

```
bool found2, validlist;
char rtn;

// function prototypes
bool nbr_search(int nbrhoodsize, short int element, short int nbrs2[]);

for (i=0; i < maxnbrhood; i++)
    nbrs2[i] = -1;

validlist = false;
found2 = false;

//cout << "numPoints: " << numPoints << endl;

for (i=0; i < numPoints-1; i++)
{
    element = pointVector[i];

    //cout << "element: " << element << endl;

    for (j=i+1; j < numPoints; j++)
    {
        found2 = false;

        for (m = 0; m < maxnbrhood; m++)
        {
            // read the jth column of pointnbrs into neighbor vector
            nbrs2[m]=pointnbrs[m][j];
```

```
        } // end for m

        // call function to determine if element is a nbr of point j
        found2 = nbr_search(numnbrs[j], element, nbrs2);

        // if element is not a neighbor of point j, break
        if (found2 == false)
            break;
    } // end for j
    if (found2 == false)
        break;
} // end for i
if (found2)
{
    // all of the points in pointVector are neighbors
    validlist = true;
} // end if found

cout << "{";
for (c=0; c < numPoints-1; c++)
    cout << pointVector[c] << ", ";
cout << pointVector[numPoints-1] << "}" << endl;

/*if (validlist)
{
    cout << "list is valid " << endl;
    cin >> rtn;
}*/
```

```
    return validlist;

} // end function validate_point_list

/*****************************************************************************
Function Name: Common_Nbrs
Purpose:   Takes 2 neighborhood vectors and finds their common members.
           Returns the number of common neighbors and creates a vector
           containing the common membership.
Called by: Analyze_Structures
Functions Called: nbr_search
*****************************************************************************/
short int Common_Nbrs(short int vectorA[], short int lengthA, short int vectorB[],
                      short int lengthB, short int tempCNbrs[])
{
    // local variables
    short int numcnbrs, element, i;
    bool found;

    // function prototypes
    bool nbr_search(int nbrhoodsize, short int element, short int neighbors[]);

    // initialize vector
    for (i = 0; i < maxnbrhood; i++)
        tempCNbrs[i] = -1;

    numcnbrs = 0;
```

```
    for(i=0; i < lengthA; i++)
    {
        element = vectorA[i];
        found = false;

        // call a function to see if this is a neighbor of point B
        found = nbr_search(lengthB, element, vectorB);

        if (found)
        {
            // element is a common neighbor
            tempCNbrs[numcnbrs] = element;
            numcnbrs++;
        } // end if found
    } // end for i

    return numcnbrs;

} // end function Common_Nbrs

/*************************************************************************
Function Name: nbr_search
Purpose:   Searches a vector to determine if it contains a specific value.
Called by: Common_Nbrs, validate_point_list, find_initial_neighborhood
           Analyze_Structures
Functions Called: none
*************************************************************************/
bool nbr_search(int nbrhoodsize, short int element, short int neighbors[])
```

```
{
    // local variables
    short int i;
    bool found;
    char rtn;

    found = false;

    // write the neighborhood to the screen
    /*cout << endl;
    cout << "The neighborhood is: ";
    cout << "{" << neighbors[0];
    for (i = 1; i < nbrhoodsize; i++)
        cout << ", " << neighbors[i];
    cout << "}" << endl;*/

    for (i = 0; i < nbrhoodsize; i++)
    {
    //cout << "element: " << element << " compared with: ";
    //cout << neighbors[i] << endl;
    /*if (found)
        cout << " found ";
    cout << "hit c to continue ";
    cin >> rtn; */

        if (neighbors[i] == element)
        {
            found = true;
```

```
            break;
        } // end if
    }
    return found;

} // end nbr_search

/*****************************************************************************
Function Name: find_point
Purpose:    To find the first row in the PointBlockpairs matrix where this
point occurs.  Takes as input the PointBlockpairs matrix and a
point and returns the row in which that point first appears.
Called by: find_neighborhood
Functions Called: none
*****************************************************************************/
short int find_point(short int point)
{
    // local variables
    short int row;
    bool notfound;

    // initialize variables
    notfound = true;
    row = 0;

    // search the PointBlockpairs matrix for the point
    while(notfound)
    {
```

```
        if (PointBlockpairs[row][0] == point)
            notfound = false;
        else
            row++;
        //end if-else
    } // end while

    return row;

} // end function find_point

/*****************************************************************************
Function Name: sort_vector
Purpose:    To take an unsorted vector and return it with the entries in
            ascending numerical order.
Called by: find_neighborhood
Functions Called: none
*****************************************************************************/
void sort_vector(short int array_to_sort[], int listsize)
{
    // local variables
    short int temp;
    int pass, i;
    bool exchanges;

    // initialize variables
    pass = 1;
    exchanges = true;
```

```
    while (pass < listsize && exchanges)
    {
        exchanges = false;
        for (i = 0; i < listsize - pass; i++)
        {
            if (array_to_sort[i] > array_to_sort[i+1])
            {
                temp = array_to_sort[i];
                array_to_sort[i] = array_to_sort[i+1];
                array_to_sort[i+1] = temp;
                exchanges = true;
            } // end if
        } // end for
        pass++;
    } // end while

} // end function sort_vector

/*****************************************************************************
Function Name: get_points
Purpose:    To get from the user the points whose common neighborhood is being
            analyzed and put them into a vector.  It returns the number of
            points used to define the common neighborhood.
Called by: main
Functions Called: none
*****************************************************************************/
short int get_points()
```

```
{
    // local variables
    short int i, point, initPtNum;

    cout << "Enter the number of points used to determine the common ";
    cout << "neighborhood:  ";
    cin >> initPtNum;
    cout << endl;

for (i = 0; i < initPtNum; i++)
{
    cout << "Enter point #" << i+1 << ": ";
    cin >> point;
    initPV[i] = point;
} // end for

return initPtNum;

} // end function get_ points
```

Bibliography

[1] E. Bannai and T. Ito. *Algebraic Combinatorics I: Association Schemes.* Benjamin Cummings, Menlo Park, CA, 1984.

[2] T. Bending and D. Fon-Der-Flaass. Crooked functions, bent functions, and distance regular graphs. *Electronic Journal of Combinatorics*, 5:R34, 14, 1998.

[3] T. Beth and C. Ding. On almost perfect nonlinear permutations. In *Advances in Cryptography, EUROCRYPT '93, Lecture Notes in Computer Science*, volume 765, pages 65–76. Springer-Verlag, 1994.

[4] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4:3–72, 1991.

[5] A.E. Brouwer. On the uniqueness of a certain thin near octagon (or partial 2-geometry, or parallelism) derived from the binary Golay code. *IEEE Transactions on Information Theory*, 29:370–371, 1983.

[6] A.E. Brouwer, A.M. Cohen, and A. Neumaier. *Distance-Regular Graphs.* Springer-Verlag, 1989.

[7] P. Camion and A. Canteaut. Construction of t-resilient functions over a finite alphabet. In *Advances in Cryptography, EUROCRYPT '96, Lecture Notes in Computer Science*, volume 1070, pages 283–293. Springer-Verlag, 1996.

[8] A. Canteaut, P. Charpin, and H. Dobbertin. Binary m-sequences with three-valued cross-correlation: a proof of Welch's conjecture. *IEEE Transactions on Information Theory*, 46:4–8, 2000.

[9] C. Carlet, P. Charpin, and V. Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes, and Cryptography*, 15:125–156, 1998.

[10] F. Chabaud and S. Vaudenay. Links between differential and linear cryptanalysis. In *Advances in Cryptology, EUROCRYPT '94, Lecture Notes in Computer Science*, volume 950, pages 356–365. Springer-Verlag, 1995.

[11] R.S. Coulter and M. Henderson. A class of functions and their application in constructing semi-biplanes and association schemes. *Discrete Mathematics*, 202:21–31, 1999.

[12] D. de Caen and E.R. van Dam. Association schemes related to Kasami codes and Kerdock sets. *Designs, Codes, and Cryptography*, 18:89–102, 1999.

[13] P. Dembowski and T. Ostrom. Planes of order n with collineation groups for order n^2. *Mathematische Zeitschrift*, 103:239–258, 1968.

[14] H. Dobbertin. One to one highly nonlinear power functions on $GF(2^n)$. *Applicable Algebra in Engineering, Communication, and Computing*, 9:139–152, 1998.

[15] H. Dobbertin. Almost perfect nonlinear power functions on $GF(2^n)$: a new case for n divisible by 5. In *Finite Fields and Applications*, pages 113–121. Springer, 1999.

[16] H. Dobbertin. Almost perfect nonlinear power functions on $GF(2^n)$: the Welch case. *IEEE Transactions on Information Theory*, 45:1271–1275, 1999.

[17] H. Dobbertin. Almost perfect nonlinear power functions on$GF(2^n)$: the Niho case. *Information and Computation*, 151:57–72, 1999.

[18] H. Dobbertin, D. Mills, E. Müller, A. Pott, and W. Willems. APN functions in odd characteristic. *Discrete Mathematics*, 267:95–112, 2003.

[19] A. Gardiner. Antipodal covering graphs. *Journal of Combinatorial Theory*, 16:255–273, 1974.

[20] R. Gold. Maximal recursive sequences with 3-valued recursive crosscorrelation functions. *IEEE Transactions on Information Theory*, 14:154–156, 1968.

[21] T. Helleseth and P.V. Kumar. Sequences with low correlation. In W.C. Huffman V.S. Pless, editor, *Handbook of Coding Theory*, pages 1765–1853. Elsevier, Amsterdam, 1998.

[22] T. Helleseth, C. Rong, and D. Sandberg. New families of almost perfect nonlinear power mappings. *IEEE Transactions on Information Theory*, 45:475–485, 1999.

[23] J. Hemmeter. Halved graphs, Johnson and Hamming graphs. *Utilitas Mathematica*, 25:115–118, 1984.

[24] H.D.L. Hollman and Q. Xiang. A proof of the Welch and Niho conjectures on crosscorrelations of binary m-sequences. *Finite Fields and their Applications*, 7:253–286, 2001.

[25] D.R. Hughes. Biplanes and semibiplanes. In *Lecture Notes in Mathematics*, volume 686, pages 55–58. Springer-Verlag, 1978.

[26] T. Kasami. The weight enumerators for several classes of subcodes of the second order binary Reed-Muller codes. *Information and Control*, 18:369–394, 1971.

[27] J. Kim and S. Y. Song. Classification of small class association schemes of order 64. preprint, 2005.

[28] A. Neumaier. Characterization of a class of distance-regular graphs. *Journal für die Reine und Angewandte Mathematik*, 357:182–192, 1985.

[29] K. Nyberg. Perfect nonlinear S-boxes. In *Advances in Cryptology, EUROCRYPT '91, Lecture Notes in Computer Science*, volume 547, pages 378–386. Springer-Verlag, 1992.

[30] K. Nyberg. Differentially uniform mappings for cryptography. In *Advances in Cryptology, EUROCRYPT '93, Lecture Notes in Computer Science*, volume 765, pages 55–64. Springer-Verlag, 1994.

[31] S. Y. Song. Association schemes and design theory. Course lecture notes at Iowa State University, 2000.

[32] P. Terwilliger. Root systems and the Johnson and Hamming graphs. *European Journal of Combinatorics*, 8:73–102, 1987.

[33] P. Terwilliger. The classification of distance-regular graphs of type IIB. *Combinatorica*, 8:125–132, 1988.

[34] E. R. van Dam, W. H. Haemers, J. H. Koolen, and E. Spence. Characterizing distance-regularity of graphs by the spectrum. preprint, 2005.

[35] E.R. van Dam and D. Fon-Der-Flaass. Uniformly packed codes and more distance regular graphs from crooked functions. *Journal of Algebraic Combinatorics*, 12:115–121, 2000.

[36] E.R. van Dam and D. Fon-Der-Flaass. Codes, graphs, and schemes from nonlinear functions. *European Journal of Combinatorics*, 24:85–98, 2003.

[37] P. Wild. Some families of semibiplanes. *Discrete Mathematics*, 138:397–403, 1995.

[38] Q. Xiang. Maximally nonlinear functions and bent functions. *Designs, Codes, and Cryptography*, 17:211–218, 1999.

ACKNOWLEDGMENTS

"... *we know that suffering produces perseverance; perseverance, character; and character, hope. And hope does not disappoint us* ... - Romans 5:3-5.

Thanks be to God for His indescribable gift. Thank you Lord, for the precious gift of your son Jesus whose spilt blood pardoned my sin and whose resurrection conquered death allowing me to have a personal relationship with you. Thank you for being my strength when I was weak and your faithfulness in sustaining me along a path which I never envisioned I would travel. Thank you for the passion you have given me to teach and your provision in helping me to finish my degree so that I can more fully be who you created me to be.

Thanks also to my family. Mom, Dad, Cortney, and Brock, without your prayers and constant prodding and encouragement I never would have achieved this goal. Thanks also to Dr. Mary Wiedenhoeft for modeling for me what a Christian professor looks like. Thanks for meeting with me on a weekly basis these last five years to listen to my complaints, share in my joys, and most of all to pray with me as I have travelled this difficult road. Thanks also to Rachel and the rest of the international fellowship. It has been so exciting to worship our great God alongside you these past two years. Thanks also to the many people who have been "praying me through", especially Mark and Peg, Sue, Heather, and my Bible Study group. To the members of First Evangelical Free Church, thanks for being a reflection of the "Body of Christ" and for becoming my family. I will miss you all very much.

Thanks also to my new colleagues as Trinity Christian College. Thanks for taking a chance on me and for all of the encouragement, excitement, and acceptance you have shown me as I have fulfilled my final requirements at ISU and prepared to join you.

Finally, thanks to my advisor and committee. I know that I have not always been the easiest person to work with, but thanks, Dr. Song, for being patient and encouraging through all of my frustrations. Thank you all for seeing me through to the completion of this degree.