Prescribed Automorphism Groups and Two Problems in Galois Geometries

by

Joanna Lynn Wallis

B.A., University of Windsor, 1998.

B.Sc., Brock University, 2002.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department of Mathematics

SIMON FRASER UNIVERSITY

Summer 2006

APPROVAL

Name:	Joanna Lynn Wallis
Degree:	Master of Science
Title of Thesis:	Prescribed Automorphism Groups and Two Problems in Galois Geometries
Examining Committee:	Dr. N. Bruin Chair
	Dr. P. Lisoněk Senior Supervisor
	Dr. J. Jedwab Supervisory Committee
	Dr. M. Mishna Examiner
Date of Defense:	July 25, 2006

Abstract

We study two combinatorial design problems: finding small t-fold blocking sets in $PG(2, q)$ for q prime, and finding large caps in $PG(n, q)$. These problems are combinatorially difficult, but their problem sizes can be reduced by prescribing a group of automorphisms in order to take advantage of the anticipated symmetry of the solution sets. Using this method, a new family of 2-fold blocking sets of size $3q + 1$ was discovered. These new sets have the additional property that their complement is also a 2-fold blocking set.

For caps, a backtracking algorithm with pruning was developed to solve the resulting 0-1 integer linear programming problem, and was implemented using C. This algorithm takes advantage of the small size of the right-hand side of the constraints in the LP problem.

Also presented is an introduction to finite projective geometries and group actions and the application of group actions to projective geometries.

Keywords: Prescribed Automorphisms, Finite Projective Geometries, Blocking Sets, Caps, Coding Theory.

Dedication

On the path of personal discovery and fulfillment all the energy of the universe strives to allow you to succeed. Just as you come close to actualizing these aspirations, life becomes more difficult than ever imagined, and all seems out of reach. The challenge is to find the strength and power to endure the final step to fulfilling your dreams.

For Don and Geo Anne Wallis, and Georgena Schisler, for all their support and encouragement in helping me to fulfill my dreams and to Kaylee, Jessica and Megan for always given me a reason to smile.

Acknowledgements

I would like to express my sincerest appreciation to my senior supervisor, Dr. Petr Lisoněk, whose dedication, commitment and encouragement kept me working long after I'd given up hope of ever completing this thesis.

I'd like to thank Dr. Alistair Lachlan for supporting me and giving me the opportunity to study at Simon Fraser University. I would like to also express my thanks to Dr. Sheridan Houghten, and Dr. Tom Jenkyns who supported me through my undergraduate degree and encouraged me to pursue a graduate degree.

I'd like to thank NSERC for the funding I received as a Reasearch Assistant, and I'd like to thank Simon Fraser University and the Simon Fraser University Mathematics Department for the funding I received through a Graduate Fellowship and many Teaching Assistant appointments. I'd also like to thank IRMACS and their friendly and helpful staff for providing a pleasant and functional environment in which I was able to work.

Finally, I would like to thank Mike Letourneau for going above and beyond the call of friendship, and my parents and family for always believing in me and supporting me.

Contents

Approval ii

Abstract iii

Dedication iv

Acknowledgements v

Contents vi

List of Tables viii

List of Figures ix

1 Galois Geometries 1

1.1 Introduction 1

1.2 Finite Projective Spaces 2

1.2.1 m-spaces in $PG(n,q)$ 3

1.2.2 Counting m-spaces in $PG(n,q)$ 5

1.2.3 The Dimension Theorem in $PG(n,q)$ 7

1.2.4 Collineations and Projectivities 8

1.2.5 Caps and Blocking Sets 11

1.3 Example: The Fano Plane 12

2 Prescribed Automorphisms and Linear Programming 15

2.1 Group Actions and Orbits 15

2.2 The Action of $PGL(n+1,q)$ on $PG(n,q)$ 20

2.3 The Method of Prescribed Automorphisms 21

2.4 The Linear Programming Problem 23

2.5 Choosing a Group 29

3 Blocking Sets in $PG(2, q)$ 33

3.1 1-fold Blocking Sets 33

3.2 Double Blocking Sets 36

3.2.1 A 38-Point Double Blocking Set in $PG(2, 13)$ 36

3.3 A Family of Proper Double Blocking Sets 37

3.3.1 Preparatory Facts 37

3.3.2 The Construction 40

4 Codes and Caps 46

4.1 Coding Theory Background 46

4.2 A Relationship Between Caps and Codes 50

4.3 Known Maximum Caps 51

4.4 Prescribed Automorphisms and Maximum Caps 54

4.5 Solving the Integer Linear Programming Problem 57

4.6 Results 63

5 Conclusion and Future Works 65

5.1 Conclusion 65

5.2 Future Works 66

A Magma Code 67

B Integer Programming Code 73

B.1 The C Code 73

B.2 Input File 89

Bibliography 91

List of Tables

3.1 Orbits On Lines and Repair Points 45

4.1 CPLEX and ipv6.c Timings (in seconds) 64

4.2 Large caps in small projective spaces with q prime 64

List of Figures

1.1 The Fano Plane - $PG(2,2)$ 12

1.2 A Relabeling of the Points in $PG(2,2)$ 12

1.3 An Alternative Labeling of the Points of $PG(2,2)$ 13

3.1 A Typical $\mathbb{Z}_4$-Orbit 41

4.1 Codes With Minimum Distance d. 48

4.2 An Example Search Tree for a Backtracking Algorithm. 58

4.3 Backtracking Algorithm with Pruning 59

4.4 A Linked List 60

Chapter 1

Galois Geometries

1.1 Introduction

Suppose in a given population there are two definable groups of people, a majority and a minority. Now, suppose the population has a government consisting of 57 people that resolves issues and decides policy for the entire population. The decision making is divided among committees; each committee consists of 8 people, and each person sits on 8 committees. Suppose that the committee votes on propositions, and any two votes can veto, or block a proposition. What is the minimum number of government members that must be from the minority group, and how should they be assigned to the committees in order to ensure that they can veto any proposition on any committee? This problem can be solved by means of a combinatorial design problem known as a 2-fold blocking set, or double blocking set in the finite projective plane $PG(2, q)$, where in this case, $q = 7$.

For prime q, the smallest known family of 2-fold blocking sets has size $3q$, but it requires that three of the committees are comprised entirely of minority members. In the case where q is prime and $q \equiv 3 \pmod 4$, using the method of prescribed automorphisms, we have discovered a 2-fold blocking set of size $3q + 1$ such that the complement of the 2-fold blocking set is also a 2-fold blocking set (see Theorem 3.11). Thus, by assigning one more seat to the minority, we can distribute the members in such a way that *both* groups have blocking power on every committee. These results along with others are presented in Chapter 3.

Coding theory, the study of encoding and transmitting information over noisy channels, is considered to have its origins in a 1948 paper by Claude Shannon, entitled "Mathematical Theory of Communication." Since then, scientists have attempted to find and construct better codes in an effort to improve the reliability and efficiency of transmission, as well as the amount of information that can be transmitted. In Chapter 4, a second combinatorial design problem, that of finding caps in $PG(n,q)$, is presented and discussed along with a relationship between caps and codes. As in the problem of finding small t-fold blocking sets, finding large caps in $PG(n,q)$ is combinatorially difficult for n and q large enough. Also presented in Chapter 4 is a program written by the author for solving the resulting zero-one integer programming problem.

In Chapter 2, the method of prescribed automorphisms, a technique used to reduce the size of the design problems mentioned above, is described in detail.

Chapter 1 introduces finite projective spaces, a type of Galois Geometry, along with the supporting definitions and notation used in this thesis.

1.2 Finite Projective Spaces

We follow much of the notation used in [Hir98] and direct the interested reader to that text for a more thorough introduction and study of Projective Geometries.

Throughout this thesis, q denotes a prime power, and all vectors are row vectors unless otherwise stated, or if it is clear from the context. For a prime power q, let $\mathbb{F}_q$ be the finite field of q elements and let $V = \mathbb{F}_q^{n+1}$ be the $(n+1)$-dimensional vector space over $\mathbb{F}_q$. For simplicity of notation, let $V_0 = V \setminus \{\mathbf{0}\}$ and let $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$. For $u, v \in V_0$, the relation R defined by $uRv \Leftrightarrow \exists\ t \in \mathbb{F}_q^*$ such that $tu = v$, is an equivalence relation on V_0. We denote the equivalence classes of this relation by $P(x)$ where $x \in V_0$. So, $P(x) = \{y \,|\, y \in V_0$ and $y = tx$ for some $t \in \mathbb{F}_q^*\}$. Let $P(V_0) = \{P(x) \,|\, x \in V_0\}$.

The elements of $P(V_0)$ are the **points** of a geometrical structure denoted by $PG(n,q)$, **the n-dimensional projective space over** $\mathbb{F}_q$. The projective and affine spaces over $\mathbb{F}_q$ are sometimes called Galois Geometries since $\mathbb{F}_q$ is also known as the Galois Field of q

elements. If $n = 2$ then $PG(2, q)$ may be referred to as the **projective plane over** $\mathbb{F}_q$. If $v = (v_0, v_1, \ldots, v_n) \in V_0$ then the points of $PG(n, q)$ are explicitly denoted by $P(v) = (tv_0 : tv_1 : \ldots : tv_n)$ for some $t \in \mathbb{F}_q^*$. Notice that the choice of t does not matter, and for any $r, s \in \mathbb{F}_q^*$, $P(rv) = P(sv)$. The point $P(v)$ is usually left normalized, that is t is chosen so that the leftmost nonzero entry is 1.

1.2.1 m-spaces in $PG(n, q)$

For $-1 \leq m \leq n$, an m-**subspace** of $PG(n, q)$ (also known as an m-**space** or m-**flat**) denotes a set of points in $PG(n, q)$ that, along with the zero vector, form an $(m + 1)$-dimensional subspace of V, with **projective dimension** m. Notice that by definition an m-space is a $PG(m, q)$. The 0-spaces, previously denoted by $P(V_0)$, are the **points** of $PG(n, q)$, the 1-spaces are called **lines**, the 2-spaces are called **planes** and the $(n - 1)$-spaces are called **hyperplanes**. The (-1)-space is the empty space.

Definition 1.1 *For $m_1 \leq m_2$, an m_1-space is said to be* **incident** *with an m_2-space if the m_1-space is completely contained within the m_2-space.*

The concept of incidence is very important in finite projective spaces, since it is the only geometric relationship defining the space. There is no concept of length or distance, for example, and so the only relationship between objects in the space is given by incidence. In particular, we look at the incidence between a given m-space, and the points, the smallest non-empty subspaces in $PG(n, q)$. The following proposition gives a mathematical relationship between these two objects.

Proposition 1.2 *An m-space M can be described by a nonzero, $(n - m) \times (n + 1)$ matrix A over $\mathbb{F}_q$ with rank $n - m$ where, for any point $P(x)$ incident with M, $xA^T = \mathbf{0}$.*

Note that if M in Proposition 1.2 is a hyperplane, then A is a vector in V_0. Let $u = (u_0, \ldots, u_n) \in \mathbb{F}_q^{n+1}$ be a non-zero vector. By $u^\perp$ we denote the hyperplane in $PG(n, q)$ consisting of all projective points $(v_0 : \ldots : v_n)$ such that $u_0v_0 + u_1v_1 + \ldots + u_nv_n = 0$. We denote $u^\perp$ explicitly by $[u_0 : u_1 : \ldots : u_n]$. Note that, as with projective points, $u^\perp = (tu)^\perp$ for $t \in \mathbb{F}_q^*$.

Definition 1.3 *If M_1 is an m_1-space and M_2 is an m_2-space in $PG(n,q)$, the* **join** *of M_1 and M_2, denoted by M_1M_2, is the smallest subspace containing both M_1 and M_2.*

Definition 1.4 *If M_1 is an m_1-space and M_2 is an m_2-space in $PG(n,q)$, the* **intersection** *of M_1 and M_2, denoted $M_1 \cap M_2$, is the set of points contained in both M_1 and M_2 and is also a subspace.*

Any m-space M in $PG(n,q)$ is a join of $m+1$ linearly independent points. Specifically, if M is a 1-space, or line, then it is the join of any two distinct points incident with M, and is often described in this way. That is, if r and s are points in $PG(n,q)$ then rs is the line that contains both r and s. (See Point 1. following Example 1.15.)

There are now two ways to describe an m-space M in $PG(n,q)$: by a representative matrix A such that $xA^T = \mathbf{0}$ for all $P(x) \in M$, as is often done with hyperplanes, or by the join of $m+1$ linearly independent points in $PG(n,q)$ incident with M, as is usually done with lines and planes. Finally, let $PG_m(n,q)$ denote the **set of all m-spaces** in $PG(n,q)$, and note that $P(V_0) = PG_0(n,q)$.

Example 1.5 The finite projective geometry $PG(2,3)$ has the following 13 points:

$$\begin{aligned} PG_0(2,3) \quad = \quad & \{(0:0:1),(0:1:0),(0:1:1),(0:1:2),(1:0:0),(1:0:1),(1:0:2), \\ & (1:1:0),(1:1:1),(1:1:2),(1:2:0),(1:2:1),(1:2:2)\}. \end{aligned}$$

In the next section, we count the number of points in a projective geometry and show that these are in fact all of the points of $PG(2,3)$.

The following lemma completes this section.

Lemma 1.6 *Points $P(a), P(b), P(c) \in PG(2,q)$ are collinear if and only if*

$$\begin{vmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{vmatrix} = 0,$$

where $P(a) = (a_0 : a_1 : a_2)$, $P(b) = (b_0 : b_1 : b_2)$, $P(c) = (c_0 : c_1 : c_2)$.

Proof: Points $P(a), P(b), P(c) \in PG(2,q)$ are collinear if and only if there exists $u^{\perp} = [u_0 : u_1 : u_2] \in PG(2,q)$ such that $au^T = 0$, $bu^T = 0$ and $cu^T = 0$ since a line in $PG(2,q)$ is also a hyperplane. This is true if and only if

$$\begin{pmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{pmatrix} \cdot \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \mathbf{0},$$

which has a nontrivial solution for u if and only if

$$\begin{vmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{vmatrix} = 0.$$

■

1.2.2 Counting m-spaces in $PG(n,q)$

Let $GL(n,q)$ denote the **general linear group of degree** n **over** $\mathbb{F}_q$, that is the set of $n \times n$, invertible matrices with entries from $\mathbb{F}_q$, together with the operation of ordinary matrix multiplication. Since the $n \times n$ identity matrix is invertible, the product of two $n \times n$ invertible matrices is an $n \times n$ invertible matrix, and the inverse of an $n \times n$ invertible matrix is also an $n \times n$ invertible matrix, and since matrix multiplication is associative, $GL(n,q)$ is indeed a group.

Theorem 1.7 *The number of matrices in $GL(n,q)$ is given by*

$$|GL(n,q)| = (q^n - 1)(q^n - q)(q^n - q^2)\dots(q^n - q^{n-1}) = \prod_{i=1}^{n}(q^n - q^{i-1}).$$

Proof: This can be shown by counting the possible columns of a matrix in $GL(n,q)$. The first column can be any column vector except $\mathbf{0}$ and so there are $q^n - 1$ possibilities. The second column can be any column vector except multiples of the first column. Since there are q multiples of the first column, including $\mathbf{0}$, there are $q^n - q$ possibilities for the second column. In general, for $m \geq 2$, the m^{th} column can be any column vector but a linear combination of the previous $m-1$ columns, so there are $q^n - q^{m-1}$ possibilities for the m^{th} column.

■

Theorem 1.8 *[VLW01, Chapter 24] The number of m-spaces in $PG(n,q)$ for $m \leq n$ is given by the Gaussian number $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix}_q$, where*

$$\begin{bmatrix} n+1 \\ m+1 \end{bmatrix}_q = \frac{(q^{n+1}-1)(q^n-1)...(q^{n-m+1}-1)}{(q^{m+1}-1)(q^m-1)...(q-1)}.$$

A Gaussian number is sometimes called a Gaussian coefficient in order to emphasize its analogous relationship to the binomial coefficient $\binom{n}{m}$ which counts the number of subsets of size m in a set of size n.

Proof: (Theorem 1.8) We start by counting all bases of $\mathbb{F}_q^{m+1}$ in $\mathbb{F}_q^{n+1}$, that is we count all sequences of linearly independent vectors of size $m+1$. Using a similar argument to that in the proof of Theorem 1.7, there are $(q^{n+1}-1)(q^{n+1}-q)(q^{n+1}-q^2)\dots(q^{n+1}-q^m)$ such sequences. However, two bases span the same $(m+1)$-dimensional vector space if and only if there is a matrix $M \in GL(m+1,q)$ that maps one basis to the other, so we have counted $|GL(m+1,q)|$ bases for each $(m+1)$-dimensional vector space. Therefore, the number of $(m+1)$-dimensional vector spaces in $\mathbb{F}_q^{n+1}$ is

$$\frac{(q^{n+1}-1)(q^{n+1}-q)(q^{n+1}-q^2)\dots(q^{n+1}-q^m)}{|GL(m+1,q)|} = \frac{(q^{n+1}-1)(q^n-1)...(q^{n-m+1}-1)}{(q^{m+1}-1)(q^m-1)...(q-1)}.$$

Recognize that the same argument holds for the number of m-spaces in $PG(n,q)$.

■

Corollary 1.9 *A projective geometry $PG(n,q)$ has $(q^{n+1}-1)/(q-1)$ points and each line is incident with $q+1$ points.*

Recall from Example 1.5, the set of 13 points in $PG(2,3)$ and note that since $(3^3-1)/2 = 13$, these are all of the points in $PG(2,3)$.

Example 1.10 Each line in $PG(3,5)$ has 6 points. For example, the line $(0:0:0:1)(0:0:1:0)$ contains the points

$$\{(0:0:0:1),(0:0:1:0),(0:0:1:1),(0:0:1:2),(0:0:1:3),(0:0:1:4)\},$$

while the line described by the matrix $A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 0 \end{pmatrix}$ contains the points

$$\{(1:0:2:4),(1:1:2:4),(1:2:2:4),(1:3:2:4),(1:4:2:4),(0:1:0:0)\}.$$

Example 1.11 The projective geometry $PG(3,7)$ contains $(7^4-1)/(7-1) = 400$ points, $(7^4-1)(7^3-1)/[(7^2-1)(7-1)] = 2850$ lines, and $(7^4-1)/(7-1) = 400$ hyperplanes. Each hyperplane contains $(7^3-1)/(7-1) = 57$ lines and $(7^3-1)/(7-1) = 57$ points. Each line contains $(7+1) = 8$ points.

1.2.3 The Dimension Theorem in $PG(n,q)$

Let $\dim(U)$ denote the projective dimension of U.

Theorem 1.12 *[VLW01, page 313] (The dimension theorem in $PG(n,q)$) Let U, W be subspaces in $PG(n,q)$, then*

$$\dim(UW) + \dim(U \cap W) = \dim(U) + \dim(W). \tag{1.1}$$

Example 1.13 To illustrate equation (1.1), consider two lines l_1, l_2 in $PG(3,q)$. In Theorem 1.12, let $U = l_1$, $W = l_2$. Since $\dim(U) = \dim(W) = 1$, the right-hand side of (1.1) is 2. Since $1 \leq \dim(UW) \leq 3$ and $\dim(U \cap W) \geq -1$, exactly three cases can occur: (i) $\dim(UW) = 3$ and $\dim(U \cap W) = -1$ (l_1 and l_2 are two skew lines), (ii) $\dim(UW) = 2$ and $\dim(U \cap W) = 0$ (l_1 and l_2 are two distinct lines in a plane, which intersect in a single point - see Example 1.15), or (iii) $\dim(UW) = 1$ and $\dim(U \cap W) = 1$ (l_1 and l_2 are equal).

Example 1.14 Consider two distinct points r, s in $PG(n,q)$. Let $U = r$, $W = s$ in Theorem 1.12. Since $\dim(U) = \dim(W) = 0$, the right-hand side of (1.1) is 0 and since r, s are distinct, their intersection is the empty set, so $\dim(U \cap W) = -1$. Thus, $\dim(UW) = 1$, and rs is a line.

Example 1.15 Let l_1, l_2 be two distinct lines in $PG(2,q)$, and let $U = l_1$, $W = l_2$ in Theorem 1.12. Again, the right-hand side of (1.1) is 2. Since U, W are distinct, $1 < \dim(UW) \leq 2$. Thus, $\dim(UW) = 2$ and $\dim(U \cap W) = 0$.

Examples 1.14 and 1.15 illustrate the following two important results for projective geometries:

1. Any two distinct points in $PG(n,q)$ uniquely determine a line, and
2. Any two distinct lines in $PG(2,q)$ intersect at a single point.

Corollary 1.16 *The number of lines through a point in $PG(2,q)$ is $q+1$.*

Proof: Let P be a point in $PG(2,q)$, and let l be a line in $PG(2,q)$ such that $P \notin l$. For each point Q on l, PQ is a distinct line through P, so there are at least $q+1$ lines through P. Consider a line l' through P. By Theorem 1.12 (and point 2. in the comments following Example 1.15) $|l \cap l'| \geq 1$, so there are at most $q+1$ lines through P.

■

1.2.4 Collineations and Projectivities

The following five definitions from [Hir98] form the basis of much of the work done in this thesis. The concepts are introduced here and are explored more fully in Chapter 2.

Definition 1.17 *The mapping $\Sigma : PG(n,q) \longrightarrow PG(n,q)$ is called a* **collineation** *if and only if it is a bijection and it preserves incidence; that is, if M_1 and M_2 are subspaces of $PG(n,q)$ such that $M_1 \subset M_2$ then $\Sigma(M_1) \subset \Sigma(M_2)$.*

In other words, a collineation preserves the incidence structure of subspaces in $PG(n,q)$ in much the same way that rigid motion preserves distance and angles in a Euclidean space. So if a set of m-spaces in $PG(n,q)$, $m \leq n$, satisfies a set of properties related to incidence, then these properties continue to be satisfied in the image of this set under a collineation. For example, if a set B of points is a cap (or t-fold blocking set) in $PG(n,q)$, then it remains a cap (or t-fold blocking set) under the collineation. (The definitions for caps and t-fold blocking sets can be found in Section 1.2.5.)

It is sufficient to show that a given mapping is a collineation by showing that it is a bijection that preserves the incidence between points and lines in a projective geometry, hence the name collineation.

Let the operation of multiplication between a matrix $M \in GL(n+1,q)$ and a point $P(x) \in PG_0(n,q)$ be defined by $MP(x) = P(Mx^T)$. Note that if $P(x) = P(y)$ then there exists $t \in \mathbb{F}_q^*$ such that $x = ty$, therefore $MP(x) = P(Mx^T) = P(M(ty)^T) = P(t(My^T)) = P(My^T) = MP(y)$, thus this multiplication is well-defined. Further, if K is a subspace of $PG(n,q)$ then we define $MK = \{MP(x) \mid P(x) \in K\}$.

Definition 1.18 *The mapping $\Sigma : PG(n,q) \longrightarrow PG(n,q)$ is called a* **projectivity** *if and only if it is a bijection given by a matrix $M \in GL(n+1,q)$ such that $\Sigma P(x) = MP(x)$ for all $P(x) \in PG(n,q)$.*

Note from definitions 1.17 and 1.18 that a projectivity is a collineation.

An automorphic collineation σ of $PG(n,q)$ is an extension of the automorphism $\overline{\sigma}$ of $\mathbb{F}_q$ given by $\sigma P(x) = P(\sigma(x)) = P((\overline{\sigma}(x_0), \overline{\sigma}(x_1), \ldots, \overline{\sigma}(x_n)))$. If q is prime then $\mathbb{F}_q$ has only one automorphism, the identity mapping, and thus there is only one automorphic collineation.

Theorem 1.19 [Hir98, Section 2.1.2] **The Fundamental Theorem of Projective Geometries** *Let $\Sigma' : PG(n,q) \longrightarrow PG(n,q)$ be a collineation, then $\Sigma' = \sigma\Sigma$ where σ is an automorphic collineation and Σ is a projectivity. Specifically, if q is a prime then there exists a matrix $M \in GL(n+1,q)$ such that $\Sigma P(x) = MP(x)$ for all $P(x) \in PG(n,q)$.*

The Fundamental Theorem of Projective Geometries tells us that if q is a prime, then the only collineations from $PG(n,q)$ to $PG(n,q)$ are projectivities. *In this thesis, only the cases when q is a prime are considered.*

Example 1.20 Let $\Sigma : PG(2,3) \longrightarrow PG(2,3)$ be given by the matrix $M = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{pmatrix}$. Note that the determinant of M, $\det(M) = 2 \neq 0$, so Σ is a projectivity. The projectivity Σ gives the following mapping of the points of $PG(2,3)$:

$$\begin{array}{lll}
\Sigma(0:0:1) = (1:0:2) & \Sigma(0:1:0) = (1:2:0) & \Sigma(0:1:1) = (0:1:2) \\
\Sigma(0:1:2) = (1:1:1) & \Sigma(1:0:0) = (1:1:2) & \Sigma(1:0:1) = (0:1:0) \\
\Sigma(1:0:2) = (1:2:2) & \Sigma(1:1:0) = (1:0:1) & \Sigma(1:1:1) = (1:0:0) \\
\Sigma(1:1:2) = (0:0:1) & \Sigma(1:2:0) = (0:1:1) & \Sigma(1:2:1) = (1:1:0) \\
\Sigma(1:2:2) = (1:2:1). & &
\end{array}$$

Notice that Σ does in fact preserve lines as expected. For example, the line

$$(0:1:1)(1:1:2) = \{(0:1:1), (1:1:2), (1:0:1), (1:2:0)\}$$

maps to the line

$$(0:1:2)(0:0:1) = \{(0:1:2), (0:0:1), (0:1:0), (0:1:1)\}.$$

The matrix M in Example 1.20 is not unique; the matrix $M' = \begin{pmatrix} 2 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 2 \end{pmatrix}$ defines the same projectivity. Define a relation R such that for $M, M' \in GL(n,q)$, MRM' if and only if $MP(x) = M'P(x)$ for all $P(x) \in PG_0(n-1,q)$. The relation R is an equivalence relation and thus partitions $GL(n,q)$ into classes. Let $[M] = \{M' \in GL(n,q) \mid M'P(x) = MP(x)$ for all $P(x) \in PG_0(n-1,q)\}$ and note that $M' \in [M]$ if and only if $M' = \lambda M$ for some $\lambda \in \mathbb{F}_q^*$. For simplicity of notation, let M denote the class containing the matrix M whenever the meaning is obvious from the context.

Let $PGL(n,q) = \{[M] \mid M \in GL(n,q)\}$, and define the operation of multiplication on this set by $[M][N] = [MN]$ for all $M, N \in GL(n,q)$. If $[M] = [M']$ then there exists $\lambda \in \mathbb{F}_q$ such that $M' = \lambda M$, similarly if $[N] = [N']$ then there exists $t \in \mathbb{F}_q$ such that $N' = tN$, thus, $[M'][N'] = [(\lambda M)(tN)] = [(\lambda t)(MN)] = [MN]$, so multiplication is well-defined. Under the operation of multiplication as defined here, $PGL(n,q)$ is known as the **projective general linear group** of degree n over $\mathbb{F}_q$. Note that for $q = 2$, $PGL(n,2) \simeq GL(n,2)$.

The **centre of a group** G, denoted $Z(G)$, is the set of those elements of G that commute with every element in G. That is, $Z(G) = \{g \in G \mid gx = xg$ for all $x \in G\}$. The centre of the group $GL(n,q)$ can be shown to be the set $Z(GL(n,q)) = \{\lambda I_n \mid \lambda \in \mathbb{F}_q^*\}$ where I_n is the $n \times n$ identity matrix. The factor group of $GL(n,q)$ by its centre $Z(GL(n,q))$ is given by $GL(n,q)/Z(GL(n,q)) = \{MZ(GL(n,q)) \mid M \in GL(n,q)\}$ and since $MZ(GL(n,q)) = \{\lambda M \mid \lambda \in \mathbb{F}_q^*\}$, it is clear that $PGL(n,q) \cong GL(n,q)/Z(GL(n,q))$.

For completeness, we note that the set of collineations form a group under composition known as the collineation group and often denoted by $P\Gamma L(n,q)$. By Theorem 1.19, for q a prime, $PGL(n,q) \simeq P\Gamma L(n,q)$.

1.2.5 Caps and Blocking Sets

The following sets are introduced here and discussed in more detail in the following chapters. Examples are given in the next section.

Definition 1.21 *A* **cap** *is a set of points S in $PG(n, q)$ such that no three points of S are collinear. A cap with m points is called an* **m-cap**.

By definition, every subset of a cap is again a cap. This reduces the problem of finding all possible caps of a given space to finding all maximal caps with respect to inclusion. This motivates the following definition.

Definition 1.22 *A* **complete cap** *or* **maximal cap** *is a cap which is not properly contained in any other cap in the same space.*

A **maximum cap** is a cap of maximum size in a given space. While not all complete caps are necessarily maximum caps, clearly all maximum caps are complete.

Definition 1.23 *A* **t-fold blocking set** *B is a set of points in $PG(2, q)$ such that every line in $PG(2, q)$ intersects B in at least t points. A* **t-fold blocking k-set** *is a t-fold blocking set with k points.*

Recall that in Section 1.1, we described the problem of assigning members to committees so that the minority group always has veto power over any committee. If the additional requirement that the majority group must also have veto power over any committee is added, then we are led to the following definition:

Definition 1.24 *A* **proper t-fold blocking set** *B is a t-fold blocking set such that the complement of B, $\overline{B} = PG_0(2, q) \setminus B$ is also a t-fold blocking set. That is,*

$$\forall r, s \in PG(2, q), r \neq s, \; t \leq |rs \cap B| \leq (q + 1) - t.$$

Analogous to the case of caps, every superset of a t-fold blocking set is again a t-fold blocking set, however, while every superset of a proper t-fold blocking set is a t-fold blocking set, it is not necessarily proper.

Definition 1.25 *A (proper) t-fold blocking B is said to be* **minimal** *if there is no point P in B such that $B \setminus \{P\}$ is a (proper) t-fold blocking set.*

1.3 Example: The Fano Plane

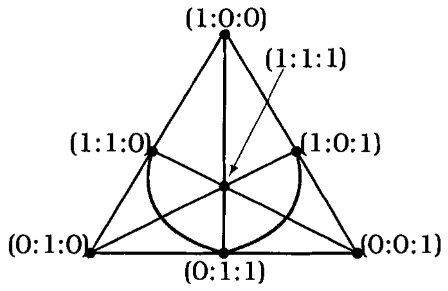


Figure 1.1: The Fano Plane - $PG(2,2)$

The Fano Plane $PG(2,2)$, named for Italian mathematician Gino Fano (1871-1952), a pioneer in the study of finite geometry, is the smallest non-trivial finite projective plane. The Fano Plane has 7 points and 7 lines. Each line has 3 points and each point is incident with precisely 3 lines. Thus, the Fano Plane is also a 2-(7,3,1) design (see [VLW01, Chapter 19]).

Example 1.26 Let $M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ be a matrix over $\mathbb{F}_2$. The determinant of M, $\det(M) = 1$ thus $M \in PGL(3,2)$. The matrix M defines in the following collineation;

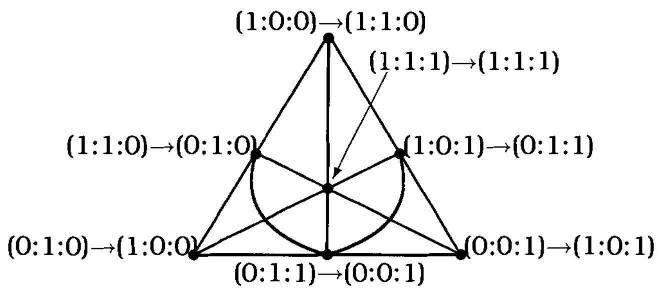


Figure 1.2: A Relabeling of the Points in $PG(2,2)$

The matrix M also gives the following mapping for the lines (not pictured): $[1:0:0] \to [0:1:0]$, $[0:1:0] \to [1:1:1]$, $[0:0:1] \to [0:0:1]$, $[1:1:0] \to [1:0:1]$, $[1:0:1] \to [0:1:1]$, $[0:1:1] \to [1:1:0]$, $[1:1:1] \to [1:0:0]$. Note that the application of the collineation to the points (lines) can be visualized as a rotation of the points (lines).

Example 1.27 Using the M in Example 1.26, the results can be rewritten as a permutation. Relabel the points in Figure 1.1 as in Figure 1.3, then M results in the following permutation: $\sigma = (132)(465)(7)$.

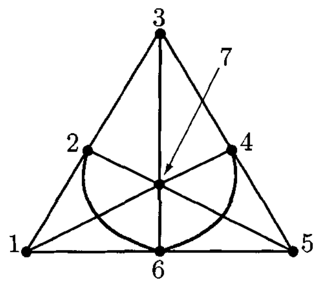


Figure 1.3: An Alternative Labeling of the Points of $PG(2,2)$

Example 1.28 Let Π be the set of points in $PG(2,2)$. By definition of a cap, the empty set is a 0-cap, every subset of Π of size 1 is a 1-cap and every subset of Π of size 2 is a 2-cap. Every subset of Π of size 3 that does not form a line in $PG(2,2)$ is a 3-cap. There are $\binom{7}{3} = 35$ distinct subsets of size 3 in Π, seven of which form lines in $PG(2,2)$. Thus there are $35 - 7 = 28$ 3-caps in Π. Every subset of Π of size 4 that does not contain a line in $PG(2,2)$ is a 4-cap. There are $\binom{7}{4} = 35$ distinct subsets of size 4 in Π and there are $4 \cdot 7$ subsets of size 4 which contain exactly one line since there are four points outside of every line. Thus, there are seven 4-caps. There are no k-caps in $PG(2,2)$ for $k > 4$. Consider $k = 5$. If π is a 5-cap in $PG(2,2)$ then for any $u, v, x, y \in \pi$ such that $u \neq v$ and $x \neq y$, $uv = xy$ if and only if $\{u, v\} = \{x, y\}$. Thus, π represents $\binom{5}{2} = 10$ lines in $PG(2,2)$ which has only seven lines. Since there are no 5-caps, there are clearly no 6-caps nor 7-caps in $PG(2,2)$. Thus, the total number of caps in $PG(2,2)$ is $\binom{7}{0} + \binom{7}{1} + \binom{7}{2} + 28 + 7 = 64$.

Example 1.29 The following is a case analysis of t-fold blocking sets in $PG(2,2)$. Note that we make no generalizations about these results.

Since there are 7 lines, and every point in the Fano Plane is incident with exactly three lines, no single point, nor any pair of points can form a 1-fold blocking set. However, since every line in the Fano Plane intersects every other line exactly once, every line forms a 1-fold blocking set of size 3. Two distinct points r, s are incident with exactly 5 lines, and the remaining two lines intersect at the point $r+s$ on rs so that there is no proper 1-fold blocking set of size 3. A set of four points such that no three are collinear, is incident with $4 \cdot 3 - \binom{4}{2} = 6$ lines, so there is no proper 1-fold blocking set of size 4. The analysis of caps in $PG(2,2)$ in Example 1.28 showed that any set of five or more points must contain a line, and so also cannot be a proper 1-fold blocking set. Thus, the Fano Plane contains no proper 1-fold blocking set and by extension, no proper t-fold blocking set for $t \geq 1$, and any 1-fold blocking set that is not a line cannot be minimal. For any set of 5 points, there must be a line that intersects the set at no more than 1 point, so there can be no 2-fold blocking set of size 5 or smaller. However, any set of six points is a 2-fold blocking set. Trivially, any 3-fold blocking set must contain all seven points.

Chapter 2

Prescribed Automorphisms and Linear Programming

In this chapter we introduce the topic of prescribed automorphisms. We begin the discussion with a brief introduction to group actions in Section 2.1, followed by the application of the definitions and theorems directly to $PGL(n+1, q)$ and $PG(n, q)$ in Section 2.2. In Section 2.3, the method of prescribed automorphisms is described as it pertains to $PG(n, q)$, and Section 2.4 shows how certain problems of finding sets with a specified structure relating to incidence can be translated into an integer linear programming problem (ILP problem) and how the method of prescribed automorphisms is applied to the ILP problem in an effort to make it solvable. Finally, in Section 2.5, we discuss methods for choosing the groups used to prescribe the automorphisms.

2.1 Group Actions and Orbits

Throughout this chapter, let G be a group and let X be a nonempty set. For groups G, H, the notation $H \leq G$ means that H is a subgroup of G. Most of the material covered in this section can be found in [Ker99].

Definition 2.1 *A* **group action** *of G on X is described by a mapping*

$$G \times X \to X : (g, x) \mapsto gx,$$

such that, for each $x \in X$ and any $g, g' \in G$, the following hold:

$$\begin{aligned} g(g'x) &= (gg')x, \text{ and} \\ 1x &= x. \end{aligned}$$

We may sometimes abbreviate this by saying that G **acts on** *X.*

Example 2.2 Let S_3 be the symmetric group on $\{1, 2, 3\}$ and let X be the set of all sequences of length 3 from the set $\{a, b, c, \ldots, z\}$. Define the action of S_3 on X by $\pi x = \pi(x_1, x_2, x_3) = (x_{\pi^{-1}(1)}, x_{\pi^{-1}(2)}, x_{\pi^{-1}(3)})$. So, for example, if $\pi = (123)$, then $\pi^{-1} = (132)$ and for $x = (a, b, r)$, $\pi x = (r, a, b)$.

Example 2.3 Let $G = \{1, g\}$, where $g^2 = 1$, and let X_N be the set of positive integer divisors of a positive integer N. Define the action of G on X_N by $1x = x$ and $gx = N/x$ for all $x \in X_N$. So, for example, if $N = 10$ and $a = 5$, then $1a = 5$ and $ga = 2$.

The action of G on X can be extended to all subsets of X. Let 2^X denote the **power set** of X, the set of all subsets of X.

Definition 2.4 *Let the* **induced action** *of G on 2^X be given by $gS := \{gs \mid s \in S\}$ for each $g \in G$ and each $S \subseteq X$.*

Notice that $1S = S$ and $g(g'S) = (gg')S$ and so this is indeed a group action.

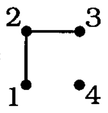
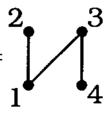
Lemma 2.5 *For all $g \in G$ and for all $S \subseteq X$, $|gS| = |S|$.*

Proof: Since $gS = \{gs \mid s \in S\}$, $|gS| \leq |S|$. For $s, t \in S$, if $gs = gt$ then $g^{-1}(gs) = g^{-1}(gt)$ and by Definition 2.1, $(g^{-1}g)s = (g^{-1}g)t$. Therefore, $s = t$ and $|gS| \geq |S|$.

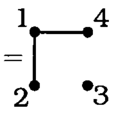
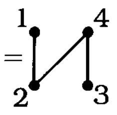
■

Example 2.6 Let G_4 be the set of labeled graphs on the vertex set $\{1, 2, 3, 4\}$ and let S_4 be the symmetric group on the set $\{1, 2, 3, 4\}$. Then S_4 acts on G_4 by permuting the vertices of $x \in G_4$ according to their labels. The induced action of S_4 on pairs of vertices gives the mapping for edges as stated in Definition 2.4.

For example, let $x_1, x_2 \in G_4$ be given by

$x_1 =$  $x_2 =$ 

and let $g = (12)(34) \in S_4$. Then,

$gx_1 =$  $gx_2 =$  .

In Example 2.6, the graphs x_1 and gx_1 are isomorphic, meaning that there is a bijection on the set of labels that maps one graph to the other. Similarly, x_2 and gx_2 are isomorphic. In fact, for any $g \in S_4$ and $x \in G_4$, x and gx will be isomorphic. Further, if $x, y \in G_4$ such that x and y are not isomorphic, then there is no $g \in S_4$ such that $y = gx$. This concept extends to all group actions and leads to the following lemma and definition.

Lemma 2.7 *The relation R given by*

$$xRy \Leftrightarrow \exists\, g \in G : y = gx$$

is an equivalence relation on X.

Proof: Let $x, y, z \in X$. The relation R is reflexive since $x = 1x$ where 1 is the identity of G. If xRy then there exists $g \in G$ such that $y = gx$ which implies by Definition 2.1 that $g^{-1}y = g^{-1}(gx) = (g^{-1}g)x = 1x = x$, so yRx and R is symmetric. If xRy and yRz then there exist $g, h \in G$ such that $y = gx$ and $z = hy$ which implies that $z = h(gx) = (hg)x$, so xRz and R is transitive.

■

Definition 2.8 *The equivalence class of R as described in Lemma 2.7, containing $x \in X$, is denoted by*

$$G(x) = \{gx \mid g \in G\}$$

*and is called the G-**orbit** of x, or the orbit of x when the meaning is clear.*

The set of all G-orbits on X will be denoted by

$$G \setminus\!\!\setminus X = \{G(x) \mid x \in X\}.$$

Notice that if G is the multiplicative group $\mathbb{F}_q^*$ and if $X = \mathbb{F}_q^{n+1}$, then the $\mathbb{F}_q^*$-orbit of $x \in \mathbb{F}_q^{n+1}$ is the projective point $P(x)$ and $\mathbb{F}_q^* \setminus\!\!\setminus \mathbb{F}_q^{n+1} = PG_0(n,q)$.

Now that the concept of an orbit has been established, it is natural to ask about the size of an orbit. Before this can be answered the following definition is needed.

Definition 2.9 *The stabilizer of $x \in X$, denoted by* $\text{Stab}_G(x)$, *is the set of elements of G which fix x; that is:*

$$\text{Stab}_G(x) = \{g \in G \mid gx = x\}.$$

The following definition is also included and will be of use in later sections.

Definition 2.10 *The stabilizer of a subset S of X, denoted by* $\text{Stab}_G(S)$, *is the set of elements of G which fix S setwise; that is:*

$$\text{Stab}_G(S) = \{g \in G \mid gS = S\}.$$

Note that for both the stabilizer of a set element and the stabilizer of a subset, the subscript G may be omitted if it is clear which group or subgroup is acting on X.

Theorem 2.11 *For $x \in X$,* $\text{Stab}_G(x)$ *is a subgroup of G.*

Proof: $\text{Stab}_G(x)$ is nonempty, since it contains the identity element of G. For $g_1, g_2 \in \text{Stab}_G(x)$, $(g_1g_2)x = g_1(g_2x) = g_1x = x$, therefore $g_1g_2 \in \text{Stab}_G(x)$. If $g \in \text{Stab}_G(x)$ then $gx = x$ implies that $x = g^{-1}x$, thus $g^{-1} \in \text{Stab}_G(x)$.

■

Theorem 2.12 *For $S \subseteq X$,* $\text{Stab}_G(S)$ *is a subgroup of G.*

The proof of Theorem 2.12 is similar to that for the Theorem 2.11.

Lemma 2.13 *There exists a bijection between the orbit of x, $G(x)$, and the set of left cosets $G/\mathrm{Stab}_G(x)$.*

Proof: Let $t : G(x) \mapsto G/\mathrm{Stab}_G(x)$ be given by $t(gx) = g\mathrm{Stab}_G(x)$. To show that t is well-defined and injective, note the following:

$$g_1x = g_2x \Leftrightarrow g_1^{-1}g_2x = x \Leftrightarrow g_1^{-1}g_2 \in \mathrm{Stab}_G(x) \Leftrightarrow g_1\mathrm{Stab}_G(x) = g_2\mathrm{Stab}_G(x)$$

Further, since every element of $G/\mathrm{Stab}_G(x)$ has the form $g\mathrm{Stab}_G(x)$ for some $g \in G$, t is surjective.

■

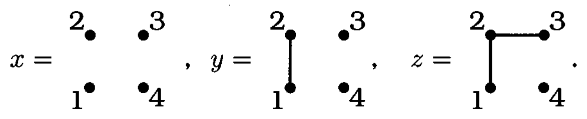
An immediate consequence of Lemma 2.13 is the following result:

Corollary 2.14 *The length of the orbit is the index of the stabilizer:*

$$|G(x)| = |G/\mathrm{Stab}_G(x)|$$

In particular, if $|G|$ is finite, then $|G(x)| = |G|/|\mathrm{Stab}_G(x)|$.

Example 2.15 As in Example 2.6, let G_4 be the set of labeled graphs on four vertices and let S_m be the symmetric group on the set $\{1, \dots, m\}$. Then S_4 acts on G_4 by permuting the vertices of $x \in G_4$ according to their labels. Note that $|S_4| = 24$. Let



The stabilizer of x, $\mathrm{Stab}_{S_4}(x) = S_4$, so $|S_4(x)| = |S_4|/|S_4| = 1$ by Corollary 2.14. The S_4-orbit $S_4(x) = \{x\}$. The stabilizer of y is the set of elements of S_4 that fix the sets $\{1,2\}$ and $\{3,4\}$, so $\mathrm{Stab}_{S_4}(y) \cong S_2 \times S_2$. Therefore $|\mathrm{Stab}_{S_4}(y)| = |S_2 \times S_2| = 4$, and $|S_4(y)| = 24/4 = 6 = \binom{4}{2}$ as expected. The S_4-orbit $S_4(y) = \{h \in G_4 \mid h \text{ has exactly one edge}\}$. The stabilizer of z is the set of elements of S_4 that fix the sets $\{1,3\}$, $\{2\}$ and $\{4\}$, so $\mathrm{Stab}_{S_4}(z) \cong S_2$. Therefore $|\mathrm{Stab}_{S_4}(z)| = |S_2| = 2$, and $|S_4(z)| = 24/2 = 12 = 4 \cdot 3$, as expected. The S_4-orbit $S_4(z) = \{h \in G_4 \mid h \text{ has exactly two edges which share a common vertex}\}$.

Notice that the set of all S_4-orbits of G_4, $S_4 \backslash\backslash G_4$, is the set of unlabeled graphs on four vertices.

2.2 The Action of $PGL(n+1,q)$ on $PG(n,q)$

In this section the definitions and properties of Section 2.1 are considered as they directly pertain to the group $PGL(n+1,q)$ and the set $PG(n,q)$.

Recall from Chapter 1 and the previous section that $[M] \in PGL(n+1,q)$ denotes the $\mathbb{F}_q^*$-orbit of $M \in GL(n+1,q)$ containing the matrix M, that $P(x) \in PG_0(n,q)$ denotes the $\mathbb{F}_q^*$-orbit of $x \in \mathbb{F}_q^{n+1}$ and that $[M]P(x) = P(Mx^T)$.

Proposition 2.16 *The mapping*

$$PGL(n+1,q) \times PG_0(n,q) \to PG_0(n,q) : ([M],P(x)) \mapsto P(Mx^T)$$

describes a group action of $PGL(n+1,q)$ on the set of points $PG_0(n,q)$.

Proof: Note that $1P(x) = [I_{n+1}]P(x) = P(x)$ and if $[g],[g\prime] \in PGL(n+1,q)$ then $[g]([g\prime]P(x)) = [g]P(g\prime x) = P(gg\prime x) = [gg\prime]P(x)$.

■

Now consider the induced action of $PGL(n+1,q)$ on $S \subseteq PG_0(n,q)$ and notice that since the elements of $PGL(n+1,q)$ are collineations, this induced action of $PGL(n+1,q)$ on S preserves all incidence relationships pertaining to S in $PG(n,q)$ (see Section 1.2.4). Specifically, if S is an m-space $PG(n,q)$, then gS is also an m-space in $PG(n,q)$, so for example g maps lines to lines, planes to planes and hyperplanes to hyperplanes.

Example 2.17 Let $M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in PGL(3,2)$, then $M^2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ and $M^3 = I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Let $G = \langle M \rangle$ and note that $G \cong \mathbb{Z}_3 = \{M, M^2, I\}$. Let X be the set of all points in $PG(2,2)$. Then $G \backslash\!\backslash X = \{\omega_1, \omega_2, \omega_3\}$ where

$$\begin{aligned} \omega_1 &= \{(0:0:1),(1:0:1),(0:1:1)\} \\ \omega_2 &= \{(1:0:0),(1:1:0),(0:1:0)\} \\ \omega_3 &= \{(1:1:1)\}. \end{aligned}$$

Using Definition 2.4 we can also calculate the orbits on the hyperplanes.

Let $a \in \mathbb{F}_2^{n+1}$ be the vector describing the hyperplane $a^\perp$ in $PG(2,2)$ and let $P(x)$ be a point incident with $a^\perp$. Recall from Proposition 1.2 and the comment following, that $xa^T = 0$. Let $N \in G$ such that $Na^\perp = b^\perp$. Note that since N is a collineation, $b^\perp$ must also be a hyperplane. By Definition 2.4, $NP(x) = P(y)$ where $P(y)$ is a point incident with $b^\perp$ which implies that $y = \lambda(Nx^T)$ for some $\lambda \in \mathbb{F}_2^*$, so $y^T = Nx^T$. Since $b^\perp$ is a hyperplane, $yb^T = 0$, thus $(xN^T)b^T = x(N^Tb^T) = x(bN)^T = 0$. Since this is true for any x incident with a it must be that $a = bN$. Multiplying on the left by N^{-1} we get $b = aN^{-1}$. So the orbits of G on the hyperplanes of $PG(2,2)$ are

$$\begin{aligned}
\Omega_1 &= \{[0:0:1]\}\\
\Omega_2 &= \{[1:0:1],[0:1:1],[1:1:0]\}\\
\Omega_3 &= \{[0:1:0],[1:1:1],[1:0:0]\}.
\end{aligned}$$

Remark 2.18 Example 2.17 shows that the action of $N \in PGL(n+1,q)$ on $a^\perp \in PG_{n-1}(n,q)$ where $a \in \mathbb{F}_q^{n+1}$ is given by $Na^\perp = (aN^{-1})^\perp$.

2.3 The Method of Prescribed Automorphisms

Much of the work completed in this thesis follows from the method of prescribed automorphisms, an approach that has previously been used successfully to find 7-designs and 8-designs [Ker99], linear codes [Bra05] and double blocking sets [BW05]. This method allows us to reduce the size of a problem's search space by considering an orbit of points as a single object rather than considering each point individually. We will see in Section 4.5 that this can significantly reduce the number of variables and constraints in the associated integer linear programming problem.

Throughout this section, we assume that the group G acts on the set X. This is a necessary condition of the method of prescribed automorphisms.

Definition 2.19 *An* **automorphism group** *of $S \subseteq X$ is any subgroup of* $\mathrm{Stab}_G(S)$. *An element of an automorphism group is called an* **automorphism**.

While the phrases automorphism group and subgroup of $\text{Stab}_G(S)$ can be used interchangeably, notice that the word automorphism implies a little more, in that it suggests not just the stabilizing of a set, but also the stabilizing of the set's shape. The fact that the groups being applied in this thesis are collineations is important, as has already been emphasized, because the preservation of the structure of the point-sets is important, and so there is a preference for calling these groups automorphism groups in order to futher emphasize this preservation of structure.

Also, note that an automorphism group of a set S is not necessarily maximal, meaning that it can be properly contained within another automorphism group. The stabilizer of S, $\text{Stab}_G(S)$ is maximal.

Recall the comments following Example 2.6 where we introduced the concept of isomorphisms of graphs. The following gives a formal definition for two isomorphic sets.

Definition 2.20 *Let S_1, S_2 be subsets of X. The set S_1 is said to be* **isomorphic** *to S_2, denoted $S_1 \cong S_2$, if and only there exists a $g \in G$ such that $gS_1 = gS_2$.*

Theorem 2.21 *Let S be a subset of X and let H be a subgroup of G. If H is an automorphism group of S then S is a union of H-orbits on X.*

Proof: If H is an automorphism group of S and $s \in S$, then $hs \in S$ for all $h \in H$, so the H-orbit of s, $H(s)$, is a subset of S. Therefore S is a union of H-orbits on X.

■

Theorem 2.22 *Let G be a subgroup of $PGL(n+1, q)$. Let ω be an arbitrary G-orbit on $PG_0(n, q)$, and let Ω be an arbitrary G-orbit on the set of m-spaces $PG_m(n, q)$. If $J, K \in \Omega$, then $|J \cap \omega| = |K \cap \omega|$.*

Proof: If $J, K \in \Omega$, then $K = gJ$ for some $g \in G$. Since ω is an orbit, $\omega = g\omega$. So, $K \cap \omega = gJ \cap \omega = g(J \cap \omega)$. Thus, by Lemma 2.5 $|J \cap \omega| = |K \cap \omega|$.

■

Please notice that, by Theorem 2.22, given a G-orbit on an m-space J, the size of the intersection between any m-space in $G(J)$ and the G-orbit of a point $P(x)$ is independent of the choice of m-space in $G(J)$. In other words, if J is an m-space and the intersection $|J \cap G(P(x))| = t$, then $|K \cap G(P(x))| = t$ for all $K \in G(J)$. As will be shown in the next section, this becomes very important when setting up the linear programming problem.

2.4 The Linear Programming Problem

Suppose we want to find a set $B \subseteq PG_0(n,q)$ such that for each m-space $l \in PG_m(n,q)$, for a fixed $m < n$, $|l \cap B| \leq t$ where t is an integer. That is, we want to find a set of points B, such that each m-space intersects B at no more than t points. Further, suppose we want to maximize the size of B. This geometric problem can be translated into an integer linear programming problem in the following way. If we consider that each point $P_i \in PG(n,q)$ is either in B or not and thus associate with each P_i an x_i, $1 \leq i \leq r = (q^{n+1}-1)/(q-1)$ where $x_i = 1$ if $P_i \in B$, and $x_i = 0$ otherwise, then $|B| = x_1 + x_2 + \ldots + x_r$. Also, for each $l_j \in PG_m(n,q)$, let $c_{ji} = 1$ if l_j is incident with P_i and $c_{ji} = 0$ otherwise. Then $|B \cap l_j| \leq t$ can be rewritten as the inequality $c_{j1}x_1 + c_{j2}x_2 + \ldots + c_{jr}x_r \leq t$. We then have the following 0-1 integer programming problem:

Maximize

$$|B| = x_1 + x_2 + \ldots + x_r, \tag{2.1}$$

subject to, for each $l_j \in PG_m(n,q)$

$$c_{j1}x_1 + c_{j2}x_2 + \ldots + c_{jr}x_r \leq t, \tag{2.2}$$

and $x_i \in \{0,1\}$, $1 \leq i \leq r$.

Similarly, suppose we want to find a smallest set $B \subseteq PG_0(n,q)$ such that for each m-space $l_j \in PG_m(n,q)$, $|l_j \cap B| \geq t$ where t is an integer. This can be rewritten as the following minimization problem.

Minimize

$$|B| = x_1 + x_2 + \ldots + x_r, \tag{2.3}$$

subject to, for each $l_j \in PG_m(n,q)$

$$c_{j1}x_1 + c_{j2}x_2 + \ldots + c_{jr}x_r \geq t, \tag{2.4}$$

and $x_i \in \{0,1\}$, $1 \leq i \leq r$.

Note that we can also associate more than one constraint with each m-space, as we do in the case of proper double blocking sets.

Before providing an example, the following definitions are given.

Definition 2.23 *The right-hand-sides of Equations (2.1) and (2.3) are called the* **objective function**. *Each inequality in (2.2) and (2.4) is known as a* **constraint**. *The requirement that $x_i \in \{0,1\}$ will be referred to as the* **0-1 constraints**. *A vector $X \in \{0,1\}^r$ is called a* **feasible solution**, *or simply feasible, if it satisfies all constraints, including the 0-1 constraints. A feasible solution is called an* **optimal solution** *if no other feasible solution returns a better objective function value.*

Example 2.24 Find a largest set of points B in $PG(2,3)$ such that every line in $PG(2,3)$ intersects B at no more than three points.

Let $P_1 = (1:0:0)$, $P_2 = (0:1:0)$, $P_3 = (0:0:1)$, $P_4 = (1:2:0)$, $P_5 = (0:1:2)$, $P_6 = (1:2:1)$, $P_7 = (1:1:1)$, $P_8 = (1:1:2)$, $P_9 = (1:0:1)$, $P_{10} = (1:1:0)$, $P_{11} = (0:1:1)$, $P_{12} = (1:2:2)$, $P_{13} = (1:0:2)$, and let $x_i = 1$ if $P_i \in B$ and $x_i = 0$ otherwise. Also, if $P_i = (a:b:c)$ then let $L_i = [a:b:c]$. The linear programming problem becomes:

Maximize

$$|B| = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13},$$

subject to

$$\begin{array}{llcl@{\qquad}llcl}
C_1: & x_2 + x_3 + x_5 + x_{11} & \leq & 3 & C_8: & x_4 + x_8 + x_9 + x_{11} & \leq & 3 \\
C_2: & x_1 + x_3 + x_9 + x_{13} & \leq & 3 & C_9: & x_2 + x_8 + x_{12} + x_{13} & \leq & 3 \\
C_3: & x_1 + x_2 + x_4 + x_{10} & \leq & 3 & C_{10}: & x_3 + x_4 + x_6 + x_{12} & \leq & 3 \\
C_4: & x_3 + x_7 + x_8 + x_{10} & \leq & 3 & C_{11}: & x_1 + x_5 + x_6 + x_8 & \leq & 3 \\
C_5: & x_1 + x_7 + x_{11} + x_{12} & \leq & 3 & C_{12}: & x_5 + x_9 + x_{10} + x_{12} & \leq & 3 \\
C_6: & x_6 + x_{10} + x_{11} + x_{13} & \leq & 3 & C_{13}: & x_2 + x_6 + x_7 + x_9 & \leq & 3, \\
C_7: & x_4 + x_5 + x_7 + x_{13} & \leq & 3 & & & &
\end{array}$$

where C_j is the constraint associated with line L_j and $x_i \in \{0,1\}$ for $1 \leq i \leq 13$. Note that the coefficient of x_i in C_j is 1 if P_i is incident with L_j and 0 if P_i is not incident with L_j.

An upper bound on $|B|$ can be found by considering that any solution must satisfy each of the 13 constraints and that each point is associated with exactly 4 constraints so that $|B| \leq \lfloor 13 \cdot 3/4 \rfloor = 9$. Since $PG(2,3)$ is a plane, every line intersects every other line at exactly one point, so if $B = PG_0(2,3) \setminus L_t$, for some t then $|B| = 13 - 4 = 9$ and B satisfies all constraints.

Solving the linear programming problem will give the above result, and further testing shows that the plane less a line is the only optimal solution to this problem.

It is known that the 0-1 integer programming problem is NP-complete [GJ79, pg. 245], and the problem of finding optimal solutions in $PG(n,q)$ as n and q get larger quickly becomes intractable. For example, recall from Theorem 1.8 that the number of points in $PG(5,5)$ is $\left[{6 \atop 1}\right]_5 = 3906$, and the number of lines is $\left[{6 \atop 2}\right]_5 = 508431$, so an associated linear programming problem would have 3906 variables and 508431 constraints.

By restricting our attention to sets that have a certain amount of symmetry, that is, by **prescribing an automorphism group** to the solution, we arrive at a **new, more constrained** problem of a **smaller size**. In this way, we are prescribing the symmetry or, in other words, requiring that a certain symmetry be contained within any solution considered. As such, we are assuming that some type of symmetry exists in the optimal solution S, or at least in some interesting solution, which is not unreasonable since the very nature of the objects within which we are working, the finite projective geometries, are filled with symmetries, as is much of mathematics, and indeed the world around us.

We first formulate the new problem and then comment on its relationship to the original problem.

Let G be a subgroup of $PGL(n+1,q)$ and let $\omega_1,\ldots,\omega_k$ denote the k distinct G-orbits on $PG_0(n,q)$, and let $\Omega_1,\ldots,\Omega_K$ denote the K distinct G-orbits on $PG_m(n,q)$. Let $x_i = 1$ if $\omega_i \subseteq B$ and $x_i = 0$ otherwise. Then, the new linear programming problem for the maximization problem defined by (2.1, 2.2) is given by:

Maximize

$$|B| = |\omega_1|x_1 + |\omega_2|x_2 + \ldots + |\omega_k|x_k, \tag{2.5}$$

subject to, for each Ω_j

$$c_{j1}x_1 + c_{j2}x_2 + \ldots + c_{jk}x_k \leq t, \tag{2.6}$$

and $x_i \in \{0, 1\}$, and $c_{ji} = |l \cap \omega_i|$ for any m-space $l \in \Omega_j$.

Note that the minimization problem originally introduced in (2.3, 2.4) is treated similarly.

Theorems 2.21 and 2.22 guarantee that any optimal solution to the problem given in (2.5) and (2.6) is a largest solution to the original problem defined by (2.1, 2.2) that also has the symmetry prescribed by the group G.

From Corollary 2.14, it follows that the length of any G-orbit could be as large as the order of G. Thus, in the best case, we can reduce the number of variables and the number of constraints from the original problem by a factor of $1/|G|$. An instance of such a case can be found in Example 4.24, where $|G| = 5$ and indeed the number of variables and constraints is exactly $1/5$ of the the number of variables and constraints in the original problem. Note that in the extreme case when G is the trivial group we see no change from the original problem.

Suppose a set $S \subseteq PG_0(n, q)$ is an optimal solution to the original ILP problem. If we have successfully chosen our group G such that the set S is in fact a union of G-orbits, then the new ILP problem given by (2.5) and (2.6) will have an equivalent solution in that it will return a set of size $|S|$. However, if G is not an automorphism group of any optimal solution to the original problem, then the new ILP problem cannot return an optimal solution to the original problem as is shown in Example 2.26. We may no longer be able to find an optimal solution to the original problem, or even to know that we have achieved an optimal solution to the original problem, but we may be able to increase our knowledge and find good sets in a reasonable amount of time by focusing our attention on the new, smaller problem.

Note that the following two examples are contrived in order to demonstrate the importance of finding the appropriate group. The groups applied were chosen with full knowledge of the optimal solution. For the problems in which the method of prescribed automorphisms is a useful strategy, the choice of G that will yield an optimal solution is largely unknown since the optimal solutions themselves are largely unknown, and at best educated guesses can be made about what group should be applied to the problem.

Example 2.25 Use the method of prescribed automorphisms to attempt to find an optimal solution to the problem described in Example 2.24.

Let $G \leq PGL(3,3)$ be the stabilizer of a set of points in $PG(2,3)$. In particular, let $L_1 = [1:0:0]$ and let G be $\text{Stab}_{PGL(3,3)}(L_1) \cap \text{Stab}_{PGL(3,3)}((1:1:1))$. Using the method of prescribed automorphisms, we can create a new, condensed linear programming problem by finding the G-orbits on the set of points and the set of lines in $PG(2,3)$. The group G has the following orbits on points:

$$\begin{aligned} \omega_1 &= \{(1:1:1)\} \\ \omega_2 &= \{P \mid P \in L_1\} \\ \omega_3 &= PG_0(2,3) \setminus (G_1 \cup G_2) \end{aligned}$$

and the following orbits on lines:

$$\begin{aligned} \Omega_1 &= \{[1:0:0]\} && = \{L_1\}, \\ \Omega_2 &= \{[0:1:2],[1:1:1],[1:0:2],[1:2:0]\} && = \{L \mid (1:1:1) \in L\} \\ \Omega_3 &= \{[0:1:0],[0:0:1],[1:2:1],[1:1:2],[1:0:1], \\ &\quad [1:1:0],[0:1:1],[1:2:2]\} \end{aligned}$$

Note that, as constructed, the line L_1 is fixed.

Let $x_i = 1$ if $\omega_i \subseteq B$ and 0 otherwise. We arrive at the following ILP problem:

Maximize

$$|B| = x_1 + 4x_2 + 8x_3$$

Subject to

$$\begin{aligned} C_1 :\ 0x_1 + 4x_2 + 0x_3 &\leq 3 \\ C_2 :\ 1x_1 + 1x_2 + 2x_3 &\leq 3 \\ C_3 :\ 0x_1 + 1x_2 + 3x_3 &\leq 3, \end{aligned}$$

$x_i \in \{0,1\}$.

An optimal solution is given by $X = (1,0,1)$ and the associated point-set is $B = \omega_1 \cup \omega_3$, which is the plane less a line (namely the line L_1), and has size equal to 9, as expected.

The linear programming problem in Example 2.25 is much smaller than the original problem in Example 2.24, only three variables and three constraints as compared with thirteen variables and thirteen constraints in the original problem, yet both problems yielded the same results. However, we were able to use the knowledge of the symmetry of the optimal solution to the problem being asked in order to find a subgroup of $PGL(3,3)$ that would act appropriately on our point set and yield the desired result. Without that knowledge, the method of prescribed automorphisms may not give the optimal solution as we show in the next example.

Example 2.26 We answer the question posed in Example 2.24 using the method of prescribed automorphisms, but with a group that will not yield an optimal solution.

Let $R = \{(1:0:0), (0:0:1), (1:2:0), (1:2:1), (1:1:1), (1:1:0), (1:0:2)\}$ and note that R is a feasible solution to the original problem (though it is not optimal). Let $G \leq PGL(3,3)$ be the stabilizer of R. It turns out that G is isomorphic to S_4 and gives the following orbits on the set of points of $PG(2,3)$:

$$\begin{aligned}
\omega_1 &= \{(1:0:0), (1:2:1), (1:1:1)\},\\
\omega_2 &= \{(0:0:1), (1:2:0), (1:1:0), (1:0:2)\},\\
\omega_3 &= \{(0:1:0), (0:1:2), (1:1:2), (1:0:1),\\
&\quad (0:1:1), (1:2:2)\},
\end{aligned}$$

and the following orbits on lines:

$$\begin{aligned}
\Omega_1 &= \{[0:1:2], [0:1:1], [1:0:2]\},\\
\Omega_2 &= \{[1:0:0], [1:1:2], [1:0:1], [1:2:2]\},\\
\Omega_3 &= \{[0:1:0], [0:0:1], [1:2:0], [1:2:1], [1:1:1], [1:1:0]\}.
\end{aligned}$$

The corresponding integer linear program is given by

Maximize

$$|B| = 3x_1 + 4x_2 + 6x_3$$

Subject to

$$\begin{aligned} C_1: \ 2x_1 + 0x_2 + 2x_3 &\leq 3 \\ C_2: \ 0x_1 + 1x_2 + 3x_3 &\leq 3 \\ C_3: \ 1x_1 + 2x_2 + 1x_3 &\leq 3 \end{aligned}$$

where $x_i = 1$ if $\omega_i \subseteq B$ and 0 otherwise.

An optimal solution is given by $X = (1, 1, 0)$ and the associated point-set is $B = \omega_1 \cup \omega_2 = R$, and has size equal to 7, which is clearly not an optimal solution to the original problem.

Example 2.27 We conclude this section with a practical example of the computational savings achieved using the method of prescribed automorphisms. We applied the method to the problem of finding a proper double blocking set in $PG(2, 19)$, of size less than or equal to $3 \cdot 19 + 1 = 58$, using the group $\mathbb{Z}_4$. This group is introduced in the proof of Theorem 3.11. The ILP solver, CPLEX, was able to find a solution of size 58 to the reduced problem in 0.82 seconds. After running the original problem for over 35 hours, no solution had yet been found.

2.5 Choosing a Group

As is implied by Theorem 2.21, if an optimal set X to the problem being considered is the union of G-orbits for some group G, then finding that group is key to finding the solution X. However, it is also necessary to find a group that will result in a small number of orbits, so that the related ILP problem can be solved. Note, that this is not a simple problem. Though there are some basic strategies for finding G (see [Bra05]), in general, it is problem dependent and difficult. In this thesis, we attempt two methods of finding groups.

The first strategy is to use the symmetric groups. This strategy is expected to produce successful results if the point-set in $PG(n, q)$ lies in a relatively small number of m-spaces. In the case of proper 2-fold blocking sets, the point-set found lies on 4 lines, and so success was achieved with a subgroup of the symmetric group S_4, specifically Z_4. This is presented more fully in Chapter 3.

The second strategy is the use of cyclic subgroups. The benefit of this strategy is that all groups have cyclic subgroups, and specifically, good groups, that is groups that are both an automorphism group of the set X and produce a relatively small number of orbits, have cyclic subgroups. So, if we cannot find all of the symmetry associated with a good group, we may still be able to find some of the symmetry, and hopefully enough of the symmetry to solve the problem. Also, as will be shown below, we have a sufficient condition (Theorem 2.30) that can be implemented practically in conjuction with Theorem 2.32 so that we need only test relatively few cyclic subgroups to ensure that all cyclic subgroups have been considered. The theory behind this is discussed here, and the technique is employed in Chapter 4 while searching for large caps.

Let G be a group. Let $g, h \in G$. We say that g is conjugate to h, denoted $g \sim h$, if and only if there exists $x \in G$ such that $h = xgx^{-1}$. Conjugacy is an equivalence relation on the elements of the group G. The concept of conjugacy can be extended to subgroups as follows:

Definition 2.28 *Let G_1, G_2 be subgroups of G, then G_1 is* **conjugate** *to G_2, denoted $G_1 \sim G_2$, if and only there exists $x \in G$ such that $G_2 = xG_1x^{-1}$, where $xG_1x^{-1} = \{xgx^{-1} \mid g \in G_1\}$.*

Theorem 2.29 *Conjugacy is an equivalence relation on the subgroups of the group G.*

Proof: Let G_1, G_2, G_3 be subgroups of G such that $G_1 \sim G_2$ and $G_2 \sim G_3$. Since $G_1 = 1G_11^{-1}$, conjugacy is reflexive. Since $G_1 \sim G_2$, there exists $x \in G$ such that $G_2 = xG_1x^{-1}$ which implies that $G_1 = x^{-1}G_2x$, so conjugacy is symmetric. Finally, since $G_2 \sim G_3$, there exists $y \in G$ such that $G_3 = yG_2y^{-1}$, then $G_3 = y(xG_1x^{-1})y^{-1} = (yx)G_1(yx)^{-1}$, so conjugacy is transitive.

■

Recall that for $g \in G$, $\langle g \rangle$ denotes the cyclic subgroup of G generated by g.

Theorem 2.30 *For $g_1, g_2 \in G$, if $g_1 \sim g_2$ then $\langle g_1 \rangle \sim \langle g_2 \rangle$.*

Proof: Suppose $g_1 \sim g_2$, then there exists $k \in G$ such that $g_2 = kg_1k^{-1}$. So ${g_2}^i = (kg_1k^{-1})^i = kg_1^ik^{-1}$. Now, $\langle g_2 \rangle = \{g_2^i \mid i \in \mathbb{Z}\} = \{kg_1^ik^{-1} \mid i \in \mathbb{Z}\} = k\langle g_1 \rangle k^{-1}$. Therefore, $\langle g_1 \rangle \sim \langle g_2 \rangle$.

■

Note that the converse of Theorem 2.30 is not true. For example, consider the group $\mathbb{Z}_n$ under addition. Since $\mathbb{Z}_n$ is abelian, $g_1 \sim g_2$ if and only if $g_1 = g_2$, but if $k \in \mathbb{Z}_n$ satisfies $\mathrm{GCD}(k, n) = 1$ then $\langle k \rangle = \mathbb{Z}_n$. The number of such $k \in \mathbb{Z}_n$ is known as the phi function $\phi(n)$ and for $n > 2$, $\phi(n) \geq 2$. In general, it is possible to have two or more non-conjugate elements generate cyclic subgroups in the same conjugacy class.

Lemma 2.31 *Let G_1, G_2 be subgroups of G such that $G_1 \sim G_2$, then $G_2(gx) = gG_1(x)$, where $g \in G$ satisfies $G_2 = gG_1g^{-1}$.*

Proof: Since $G_1 \sim G_2$, for every $h \in G_2$, $h = gkg^{-1}$ for some $k \in G_1$, so $G_2(gx) = \{gkg^{-1}(gx) \mid k \in G_1\} = \{gkx \mid k \in G_1\} = gG_1(x)$.

■

Theorem 2.32 *If $G_1 \sim G_2$ and $S_1 \subseteq X$ is a union of G_1-orbits, then there exists $S_2 \subseteq X$ isomorphic to S_1 such that S_2 is a union of G_2-orbits.*

Proof: By Lemma 2.31, $G_2(gx) = gG_1(x)$, where $g \in G$ satisfies $G_2 = gG_1g^{-1}$ so from $S_1 = \bigcup_i G_1(x_i)$, we get $gS_1 = g(\bigcup_i G_1(x_i)) = \bigcup_i G_2(gx_i) = S_2$.

■

In other words, Theorem 2.32 says that if G_1 is an automorphism group for some set X, then every group G_2 such that $G_1 \sim G_2$ is an automorphism group for some set isomorphic to X. So, only one representative from a conjugacy class of subgroups need be applied to the problem.

Thus, for small enough n and q all cyclic subgroups of $PGL(n+1, q)$ can be tested by generating a representative element from each conjugacy class and testing the group generated by that element on the set of points of $PG(n, q)$. This dramatically reduces the number of cyclic groups being tested. For example, $PGL(4, 2)$ has 14 conjugacy classes of elements with an average of 1440 elements in each class, while $PGL(5, 2)$ has 27 conjugacy classes of elements with an average of approximately 370347 elements per class. However, note that from the comments following Theorem 2.30, generating the conjugacy classes of the cyclic subgroups in this way will still result in some redundancy. For example, the number of conjugacy classes of cyclic subgroups in $PGL(4,2)$ and $PGL(5,2)$ is 12 and 18 respectively.

For the work done in this thesis, the mathematical software program Magma was used to generate the conjugacy classes of cyclic subgroups in $PGL(n+1,q)$, which attempts to do this by examining a random selection of group elements and their powers. However, conjugacy classes can be computed more efficiently by recognizing that two $(n+1)\times(n+1)$ matrices A and B are conjugate if and only if $xI-B$ and $xI-A$ have the same Smith normal form over the ring of polynomials over $\mathbb{F}_q$ [Rot02, Chapter 9.4].

Because of the algorithm used by Magma, for n and q large enough, producing all conjugacy classes is computationally difficult, and it becomes necessary to simply generate a subset of the conjugacy classes, though doing so reduces the chances of finding a good group.

Chapter 3

Blocking Sets in $PG(2, q)$

Recall that throughout this thesis, we are primarily interested in the case when q is a prime and there is no extra algebraic structure available on $\mathbb{F}_q$, therefore we exclusively study blocking sets in the case when q is odd. For the case where $q = 2$ see Example 1.29. Our analyses will be limited to 1-fold blocking sets and 2-fold blocking sets. For the remainder of this chapter, following the notation of [Hir98], a 1-fold blocking set will be referred to as a blocking set and a 2-fold blocking set will be referred to as a double blocking set. For all t-fold blocking sets, the problem being studied is to find minimal sets and to find sets of minimum size.

In Section 3.1, the discussion begins with known minimal 1-fold blocking sets in $PG(2, q)$. In Section 3.2 we look at double blocking sets and the discovery of a 38-point double blocking set in $PG(2, 13)$. In Section 3.3 we present two families of proper double blocking sets discovered during the completion of this thesis.

3.1 1-fold Blocking Sets

Recall from Definition 1.23 that a blocking set is a set of points in $PG(2, q)$ such that every line passes through the set at least once, and from Definition 1.24, a proper blocking set is a blocking set that contains no line completely. Recall from Definition 1.25 that a minimal blocking set is a blocking set which contains no proper subset which is also a blocking set.

Definition 3.1 *A projective triangle of side n in $PG(2,q)$ is a set B of $3(n-1)$ points such that*

(a) on each side of a triangle $P_0P_1P_2$ there are n points of B;

(b) the vertices P_0, P_1, P_2 are in B;

(c) If $Q_0 \in P_1P_2$ and $Q_1 \in P_2P_0$ are in B, $Q_0 \neq Q_1$, then so is $Q_2 = Q_0Q_1 \cap P_0P_1$.

Recall that $PG_0(2,q)$ denotes the set of 0-spaces, or the set of projective points in $PG(2,q)$. If $X \subseteq PG_0(2,q)$ then a ***k-secant*** of X is a line in $PG(2,q)$ that intersects the set X exactly k times. In [Hir98], it is proven that there exists a projective triangle of side $\frac{1}{2}(q+3)$ which is also a minimal proper double blocking set in $PG(2,q)$ of size $\frac{1}{2}(3q+2)$ and this is repeated here in Proposition 3.3. The following is a slightly stronger result; that is every projective triangle of side $\frac{1}{2}(q+3)$ in $PG(2,q)$ is a minimal proper blocking set.

Theorem 3.2 *If B is a projective triangle of side $\frac{1}{2}(q+3)$ in $PG(2,q)$ with q odd then B is a minimal proper blocking set of size $\frac{3}{2}(q+1)$.*

Proof: Let P_0, P_1, and P_2 denote the vertices of the projective triangle B, and let $\{i,j,k\} = \{0,1,2\}$.

Since P_0P_1, P_0P_2 and P_1P_2 each intersect B at $\frac{1}{2}(q+3)$ points, there are at least three $\left[\frac{1}{2}(q+3)\right]$-secants.

From the definition of a projective triangle, if an arbitrary line l intersects P_0P_1 and P_0P_2 at non-vertex points in B, then it intersects P_1P_2 at a non-vertex point in B and is therefore a 3-secant. Since any line is uniquely determined by two distinct points in $PG(2,q)$, we can choose a point r on P_0P_1 and a point s on P_0P_2 to determine l. There are $\frac{1}{2}(q+3)-2 = \frac{1}{2}(q-1)$ non-vertex points on each side of the projective triangle, so there are $\frac{1}{2}(q-1)$ choices for both r and s. Once r and s are chosen, $l \cap P_1P_2$ is uniquely determined, thus the number of such 3-secants is $\left[\frac{1}{2}(q-1)\right]^2$.

If l is incident with a vertex P_k and a non-vertex point on P_iP_j in B, then l is a 2-secant. There are three choices for P_k and $\frac{1}{2}(q-1)$ non-vertex points on P_iP_j which are in B, so the number of such 2-secants is $\frac{3}{2}(q-1)$.

If l is incident with the vertex P_k and a point on P_iP_j not in B, then l is a 1-secant. There are three choices for P_k and $(q+1) - \frac{1}{2}(q+3) = \frac{1}{2}(q-1)$ non-vertex points on P_iP_j which are not in B, so the number of such 1-secants is $\frac{3}{2}(q-1)$.

If l is incident with a non-vertex point on P_iP_j in B, and a point on P_iP_k not in B, then by the definition of a projective triangle, l is incident with a point on P_jP_k not in B and is thus a 1-secant. Since there are $\frac{1}{2}(q-1)$ non vertex points on P_iP_j which are in B, $\frac{1}{2}(q-1)$ points in P_iP_k which are not in B, and since there are three choices for the line P_iP_j, there are $3\left[\frac{1}{2}(q-1)\right]^2$ such 1-secants. Note also that the two types of 1-secants described are mutually exclusive since the 1-secants in the former set contain a vertex while those in the latter set do not.

Summing the number of 1-secants, 2-secants, 3-secants and $\left[\frac{1}{2}(q+3)\right]$-secants found above, we get

$$\begin{aligned} & 3\left(\frac{q-1}{2}\right)^2 + 3\left(\frac{q-1}{2}\right) + 3\left(\frac{q-1}{2}\right) + \left(\frac{q-1}{2}\right)^2 + 3 \\ = \ & q^2 + q + 1. \end{aligned}$$

But q^2+q+1 is the total number of lines in $PG(2,q)$, thus all lines in $PG(2,q)$ are incident with at least one point in B, and since we have counted all lines, no line intersects B at more than $\frac{1}{2}(q+3)$ points. Since $\frac{1}{2}(q+3) < q+1$ for $q > 2$ it follows that no line is completely contained within B, and therefore B is a proper blocking set.

Finally, note that if P is a vertex, then there exists line PP' such that P' is on the side of the projective triangle that does not contain P and $P' \notin B$ so P lies on a 1-secant and $B \setminus \{P\}$ is not a blocking set. If $P \in B$ is a non vertex, then again, there exists P' on a side of the projective triangle, not on the same side as P such that $P' \notin B$, so that PP' is a 1-secant and $B \setminus \{P\}$ is not a blocking set. Thus B is minimal.

■

Proposition 3.3 *There exists a projective triangle of side $\frac{1}{2}(q+3)$ in $PG(2,q)$, for each odd prime power q.*

Proof: Let $Q_0(a_0) = (0:1:a_0)$, $Q_1(a_1) = (1:0:a_1)$ and $Q_2(a_2) = (1:-a_2:0)$ be points in $PG(2,q)$. Let B consist of the vertices $(0:0:1)$, $(0:1:0)$, and $(1:0:0)$ and the points

$Q_i(a_i)$ such that a_i is a nonzero square. Since there are $\frac{1}{2}(q-1)$ nonzero squares in $\mathbb{F}_q$, $|B| = 3 + 3 \cdot \frac{1}{2}(q-1) = \frac{3}{2}(q+1)$. By Lemma 1.6, Q_0, Q_1 and Q_2 are collinear if and only if $a_0 = a_1 a_2$, so if 2 non-vertex points are in the set, then the third point must also be in the set. Thus B is a projective triangle of side $\frac{3}{2}(q+1)$.

■

As will be shown in Section 3.3, an analogous, though more complicated construction exists for proper double blocking sets in $PG(2,q)$.

3.2 Double Blocking Sets

Theorem 3.4 *There exists a double blocking set in $PG(2,q)$ of size $3q$.*

Proof: Let l_1, l_2, l_3 be three distinct, non-concurrent lines in $PG(2,q)$ and let $B = l_1 \cup l_2 \cup l_3$. The order of B is given by $|B| = 3 \cdot (q+1) - 3 = 3q$, so it only remains to show that B is a double blocking set. Since B lies in a plane, every line intersects every line at least once, so for an arbitrary line $l \in PG(2,q)$, l must intersect each of l_1, l_2, and l_3 so that $|l \cap B| = 2$, 3 or $q+1$.

■

3.2.1 A 38-Point Double Blocking Set in $PG(2,13)$

Until recently, for q a prime, the smallest known size of a double blocking set in $PG(2,q)$ was $3q$ (Theorem 3.4). By applying the method of prescribed automorphisms to a complementary problem in [BW05], the authors were able to find a double blocking set of size 38 for $PG(2,13)$, one point less than the known best of 39. Independently, while researching the contents of this thesis, we found an isomorphic set of size 38. Further, by studying the structure of this set, and the subgroup of $PGL(3,13)$ used to find it, we were able to discover some other interesting results (Theorem 3.11).

It had been suspected for a time before this set was discovered, that $3q$ was not optimal, but even with this knowledge, finding the set was difficult. Even if it were known that there existed a double blocking set of size 38 in $PG(2,13)$, there are $\binom{183}{38} \approx 2^{130}$ point sets of size 38 in $PG(2,13)$. However, $PGL(3,13)$ is still small enough that all of its subgroups can be generated and applied to $PG(2,13)$. The alternating group on 4 elements, A_4 produced the following 38-point double blocking set.

$B = \{(1:12:0),(0:1:12),(1:6:12),(1:4:2),(1:9:1),(1:8:1),(1:2:8),(1:11:11),(1:2:7),(1:0:2),(1:0:1),(1:5:7),(1:10:7),(1:1:0),(0:1:1),(1:4:10),(1:5:3),(1:10:4),(1:4:11),(0:1:7),(1:12:1),(1:9:5),(1:3:10),(1:7:9),(1:7:4),(1:4:9),(1:6:8),(1:10:12),(1:8:6),(1:3:5),(1:12:2),(1:0:8),(1:4:6),(1:1:1),(1:11:0),(1:12:3),(1:7:7),(1:0:7)\}$

Once B was found, a natural question to ask was "Does the set have any geometric properties that can be exploited in an attempt to find a double blocking set smaller than $3q$ for higher values of q?". It turns out that of the 38 points in B, 34 of them lie on 4 lines in general position, and the 6 intersecting points of these four lines are also in B. Using this information, we narrowed our focus to groups that permute four lines for $q > 13$ and used the ILP approach as outlined in Chapter 2. Though we were unable to find a double blocking set of size smaller than $3q$, we did consistently find a non-trivial proper double blocking set of size $3q+1$ for q prime and $q \equiv 3 \pmod 4$. Working with this knowledge, we were able to find a construction for a proper double blocking set for all such q which is presented in Section 3.3.2.

3.3 A Family of Proper Double Blocking Sets

Throughout this section, let q be an odd prime power.

3.3.1 Preparatory Facts

Definition 3.5 *For $x \in \mathbb{F}_q$ we say that x is* **square** *if $x = s^2$ for some $s \in \mathbb{F}_q$. Otherwise, x is* **nonsquare**.

Definition 3.6 *By $\square_q$ we denote the set of all nonzero squares of $\mathbb{F}_q$ and by $\not\square_q$ we denote the set of all nonsquares of $\mathbb{F}_q$. Note that 0 does not appear in either set.*

Lemma 3.7 *Let $a, b \in \Box_q$ and $c, d \in \not\Box_q$. Then $ab \in \Box_q$, $ac \in \not\Box_q$, $cd \in \Box_q$, $a^{-1} \in \Box_q$, $c^{-1} \in \not\Box_q$.*

Proof: Let α be a primitive element in $\mathbb{F}_q$. If $a, b \in \Box_q$, then there exist some $t, s \in \mathbb{Z}$ such that $a = \alpha^{2t}$ and $b = \alpha^{2s}$. If $c, d \in \not\Box_q$, then there exist some $p, r \in \mathbb{Z}$ such that $c = \alpha^{2p+1}$ and $d = \alpha^{2r+1}$. So $1 = \alpha^{q-1}$ where $q-1$ is even. Thus,

$$\begin{array}{ll}
ab = \alpha^{2t}\alpha^{2s} = \alpha^{2(t+s)} & \text{therefore } ab \in \Box_q. \\
ac = \alpha^{2t}\alpha^{2p+1} = \alpha^{2(t+p)+1} & \text{therefore } ac \in \not\Box_q. \\
cd = \alpha^{2p+1}\alpha^{2r+1} = \alpha^{2(p+r+1)} & \text{therefore } cd \in \Box_q. \\
a^{-1} = 1/a = \alpha^{q-1}/\alpha^{2t} = \alpha^{q-1-2t} & \text{therefore } a^{-1} \in \Box_q. \\
c^{-1} = 1/c = \alpha^{q-1}/\alpha^{2p+1} = \alpha^{q-2(p+1)} & \text{therefore } c^{-1} \in \not\Box_q.
\end{array}$$

Notice that, though not explicitly stated, all exponents are taken modulo $(q-1)$, but in the case where $(q-1)$ is even, $e \bmod (q-1)$, $e \in \mathbb{Z}$, is even if and only if e is even.

■

Lemma 3.8 *If $q \equiv 3 \pmod 4$, then $-1 \in \not\Box_q$.*

Proof: Let α be a primitive element in $\mathbb{F}_q$. Since $-1 = \alpha^{\frac{1}{2}(q-1)}$, it follows that -1 is square if and only if $\frac{1}{2}(q-1)$ is even. But, $q \equiv 3 \pmod 4$, thus $\frac{1}{2}(q-1) \equiv 1 \pmod 2$, so -1 is not square.

■

Corollary 3.9 *For $q \equiv 3 \pmod 4$, $a \in \Box_q$ if and only if $-a \in \not\Box_q$.*

Proof: The proof follows from Lemmas 3.7 and 3.8.

■

Proposition 3.10 *For any prime q with $q \equiv 3 \pmod 4$ the set*

$$S = \{x \in \mathbb{F}_q \mid x \in \Box_q \text{ or } x+1 \in \not\Box_q\}$$

has cardinality $\frac{1}{4}(3q-5)$.

Proof: Consider the set

$$S' = \{x \in \mathbb{F}_q \mid x \in \not\square_q \text{ and } x+1 \in \square_q\}$$

and note that $S' \cup \{0,-1\}$ is the complement of S. If $x \in S'$, then by Corollary 3.9, for some $s,t \in \mathbb{F}_q^*$, $x = -s^2$ and $x+1 = t^2$. Thus, $s^2+t^2 = 1$.

Let $C = \{(s,t) \in \mathbb{F}_q^2 \mid s^2+t^2 = 1\}$ and note that $(1,0) \in C$. For $(s,t) \in C$, $(s,t) \neq (1,0)$, define $c = t/(s-1)$, then $t = cs - c$ and

$$\begin{aligned} s^2+(cs-c)^2 &= 1 \\ s^2+c^2s^2-2c^2s+c^2 &= 1 \\ (1+c^2)s^2-2c^2s+c^2-1 &= 0 \\ (s-1)(s+c^2s-c^2+1) &= 0. \end{aligned}$$

Since $s \neq 1$, it must be that $s = \frac{c^2-1}{c^2+1}$ and thus $t = \frac{-2c}{c^2+1}$. So,

$$C = \left\{ \left(\frac{c^2-1}{c^2+1}, \frac{-2c}{c^2+1} \right), c \in \mathbb{F}_q \right\}.$$

We want to count the number of distinct values of $x = -s^2$ where $s = \frac{c^2-1}{c^2+1}$, and $x \in S'$. Note that $(s_0,t_0) \in C$ and $(s_1,t_1) \in C$ yield the same value for x if $s_0 = \pm s_1$. Now, if $c = 0$ then $x = -(-1)^2 = -1$ and $x \notin S'$, thus we need only consider what happens if $c \in \mathbb{F}_q^*$.

Suppose that for some $c_1, c_2 \in \mathbb{F}_q^*$, $\frac{c_1^2-1}{c_1^2+1}$ and $\frac{c_2^2-1}{c_2^2+1}$ yield the same value for $x = -s^2$. Then either

$$\begin{aligned} \frac{c_1^2-1}{c_1^2+1} &= \frac{c_2^2-1}{c_2^2+1} \\ (c_1^2-1)(c_2^2+1) &= (c_2^2-1)(c_1^2+1) \\ c_1^2c_2^2-c_2^2+c_1^2-1 &= c_2^2c_1^2-c_1^2+c_2^2-1 \\ -2c_2^2+2c_1^2 &= 0 \\ c_2^2 &= c_1^2 \\ c_2 &= \pm c_1, \end{aligned}$$

or

$$\begin{aligned}
\frac{c_1^2-1}{c_1^2+1} &= -\frac{c_2^2-1}{c_2^2+1} \\
(c_1^2-1)(c_2^2+1) &= (-c_2^2+1)(c_1^2+1) \\
c_1^2c_2^2 - c_2^2 - 1 &= -c_1^2 + c_1^2 - c_2^2 + 1 \\
c_1^2c_2^2 &= 1 \\
c_2 &= \pm\frac{1}{c_1}.
\end{aligned}$$

Thus for $c \in \mathbb{F}_q^*$, c, $-c$, c^{-1} and $-c^{-1}$ yield the same value for $x \in S'$. However, since $0 \notin S'$, we see that $c \neq \pm 1$ which implies that $c \neq \pm c^{-1}$ and since q is odd and $c \neq 0$, $c \neq -c$. Thus c, $-c$, c^{-1}, $-c^{-1}$ are all distinct. Since $c \in \mathbb{F}_q \setminus \{0, 1, -1\}$, there are $q-3$ choices for c and so $|S'| = \frac{1}{4}(q-3)$.

Therefore

$$\begin{aligned}
|S| &= |\mathbb{F}_q| - |S'| - |\{0,-1\}| \\
&= q - \frac{1}{4}(q-3) - 2 \\
&= \frac{1}{4}(3q-5).
\end{aligned}$$

■

3.3.2 The Construction

Theorem 3.11 *Let $q \geq 7$ such that $q \equiv 3 \pmod 4$. There is a proper double blocking set B in $PG(2,q)$ such that $|B| = 3q+1$ and each line of $PG(2,q)$ intersects B in at most $\frac{3}{4}(q+1)$ points.*

Our construction of the proper double blocking $(3q+1)$-set has some parallels to the construction of the proper blocking set discussed in Section 3.1. In the case of the proper blocking set, all points in the set lay on three lines in general position, while the proper double blocking set in our construction lies on four lines in general position. A second similarity lies in how the points are chosen for the sets. Though the selection is more complicated for double blocking sets, both constructions are based on properties of squares in $\mathbb{F}_q^*$. For these reasons, the construction of a proper blocking set presented below in the proof of Theorem 3.11 can be considered a generalization of the construction of a projective triangle.

Proof: (Theorem 3.11) Since we require four lines in general position, for the purpose of computation, it is natural to choose $l_0 = [1:0:0]$, $l_1 = [0:1:0]$, $l_2 = [0:0:1]$, and $l_3 = [1:1:1]$.

Let M be the projectivity of $PG(2,q)$ that maps l_0 to l_1, l_1 to l_2, l_2 to l_3, and l_3 to l_0. Then

$$M = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}, M^2 = \begin{pmatrix} 0 & 0 & 1 \\ -1 & -1 & -1 \\ 1 & 0 & 0 \end{pmatrix}, M^3 = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 1 & 1 \end{pmatrix}$$

and $M^4 = I_3$. The subgroup $G = \langle M \rangle$ of $PGL(3,q)$ is isomorphic to $\mathbb{Z}_4$. We have $l_1 = Ml_0$, $l_2 = M^2 l_0$, $l_3 = M^3 l_0$. The G-orbit of point $(0:1:a) \in l_0$ is

$$G((0:1:a)) = \{(0:1:a), (a+1:0:-1), (a:-(a+1):0), (-1:-a:a+1)\}. \tag{3.1}$$

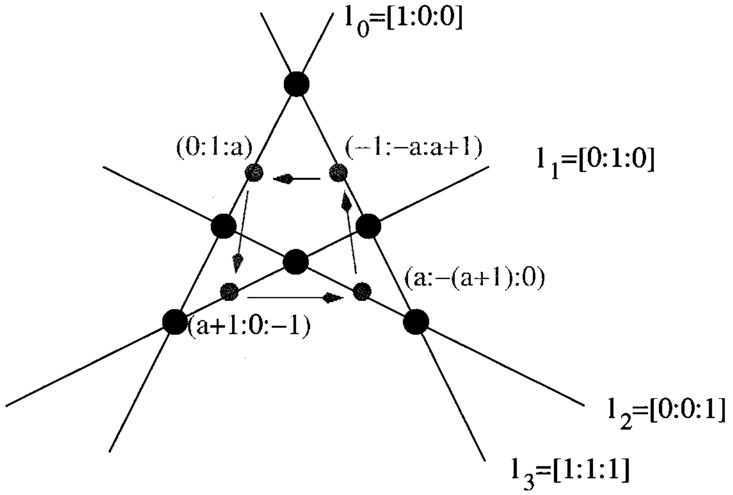


Figure 3.1: A Typical $\mathbb{Z}_4$-Orbit

Note that if $a \neq 0$ then $|G((0:1:a))| = 4$. If $a \notin \{-1,0\}$ then $p \in G((0:1:a))$ lies on exactly one of l_i, $0 \leq i \leq 3$, and if $a \in \{-1,0\}$, then $p = l_i \cap l_j$, $i \neq j$.

Let

$$B_I = \bigcup_{a \in S} G((0:1:a)),$$

where

$$S = \{x \in \mathbb{F}_q \mid x \in \square_q \text{ or } x+1 \in \not\square_q\},$$

as in Proposition 3.10, and let

$$\begin{aligned} B_V &= \{(0:1:0),(1:0:-1),(0:1:-1),(0:0:1),(1:0:0),(1:-1:0)\} \\ &= \{l_i \cap l_j \mid 1 \le i < j \le 4\}. \end{aligned}$$

Then, let

$$B = B_I \cup B_V.$$

We call B_V the vertices of B and B_I the internal points of B. Throughout this proof it will be useful to write points in the form implied by Equation (3.1), rather than in their left-normal form as is done elsewhere in the thesis. That is if a non-vertex point P lies on l_0 it will have the form $P = (0:1:a)$, $a \in \mathbb{F}_q^* \setminus \{-1\}$ and if P lies on l_1, it will have the form $P = (b+1:0:-1)$, $b \in \mathbb{F}_q^* \setminus \{-1\}$. Vertices will continue to be written in left-normalized form.

Note that if $a \neq 0$ then $|G((0:1:a))| = 4$. If $a \notin \{-1,0\}$ then $p \in G((0:1:a))$ lies on exactly one of l_i, $0 \le i \le 3$, and if $a \in \{-1,0\}$, then $p \in B_V$ and also notice that we have the following two orbits on vertices: $\{(0:1:-1),(0:0:1),(1:0:0),(1:-1:0)\}$ and $\{(0:1:0),(1:0:-1)\}$.

Clearly $|B_V| = 6$ and by Proposition 3.10 there are $\frac{1}{4}(3q-5)$ points of the form $(0:1:a)$, $a \in S$, thus $|B_I| = 4 \cdot \frac{1}{4}(3q-5) = 3q-5$. Since $-1, 0 \notin S$ the G-orbit of the point $(0:1:a)$ cannot contain a vertex and so $B_I \cap B_V = \emptyset$. Thus, $|B| = |B_V| + |B_I| = 6 + (3q-5) = 3q+1$.

Next we show that B is a proper double blocking set.

Consider a line $l \in PG_1(2,q)$. One of the following three cases must be true about l. Either l contains two vertices, l contains one vertex, or l contains no vertices.

If l contains two vertices and is not one of l_i, $i \in \{0,1,2,3\}$, then $|l \cap B| = 2$. If l contains two vertices and is one of l_i, then $2 \le |l \cap B| = \frac{1}{4}(3q-5)+2 = \frac{3}{4}(q+1) \le q-1$.

For the case where l contains a single vertex we note that by Theorem 2.22, we need only consider one vertex for each orbit on vertices.

Consider $l \in PG(2,q)$ such that $V = (0:1:0) = l \cap l_0 = l \cap l_2$, $P_1 = (b+1:0:-1) = l \cap l_1$, and $P_3 = (-1:-d:d+1) = l \cap l_3$ and $P_1, P_3 \notin B_V$. If $P_1 \in B$ then we are done. Otherwise, it must be that $b \notin S$, so $b \in \not\square_q$ and $b+1 \in \square_q$, since $b \notin \{-1,0\}$. Lemma 1.6 gives $d = -b/(b+1)$ so that $d \in \square_q$ by Lemmas 3.7 and 3.8. Thus $P_3 \in B$ and $2 \le |l \cap B| \le 3$.

Consider $l \in PG(2,q)$ such that $V = (0:1:-1) = l \cap l_0 = l \cap l_3$, $P_1 = (b+1:0:-1) = l \cap l_1$, and $P_2 = (c : -(c+1) : 0) = l \cap l_2$ and $P_1, P_2 \notin B_V$. If $P_1 \in B$ then we are done. Otherwise, Lemma 1.6 gives $c = -(b+1)/b$ so that $c \in \square_q$ by Lemmas 3.7 and 3.8. Thus $P_2 \in B$ and $2 \le |l \cap B| \le 3$.

Finally, we consider the case where l does not contain a vertex. Let l be a line not containing a vertex of B. Let $P = \{P_0, P_1, P_2, P_3\}$ where $P_i = l \cap l_i$ and $P_i \notin B_V$. We want to show that if two points in P are not in B then the other two points must be in B. Recall that $l_1 = Ml_0$, $l_2 = M^2 l_0$, $l_3 = M^3 l_0$, so $MP_0 \in l_1$, $M^2P_0 \in l_2$ and $M^3P_0 \in l_3$. Similarly, $MP_1 \in l_2$, $M^2P_1 \in l_3$ and $M^3P_1 \in l_0$.

Again, by Theorem 2.22 we need only consider the following two cases only: (i) P_0, P_1 are both outside B, (ii) P_0, P_2 are both outside B.

Case (i): Let $P_0 = (0:1:a)$ and $P_1 = (b+1:0:-1)$ where $a, b \notin S$. Thus $a \in \not\square_q$, $a+1 \in \square_q$, $b \in \not\square_q$, and $b+1 \in \square_q$. Let $P_2 = (c : -(c+1) : 0)$, $P_3 = (-1 : -d : d+1)$. Using Lemma 1.6 we find that $c = -a(b+1)/(ab+a+1)$, and note that $ab+a+1 \ne 0$ since $ab \in \square_q$ and $-(a+1) \in \not\square_q$. We want to show that $c \in S$ which is true if $c \in \square_q$ or $c+1 \in \not\square_q$. If $c \in \square_q$ then we are done. Otherwise, $c \in \not\square_q$, and $-c = a(b+1)/(ab+a+1) \in \square_q$. Since $a(b+1) \in \not\square_q$, by Lemma 3.7, $1/(ab+a+1) \in \not\square_q$. But $c+1 = -a(b+1)/(ab+a+1)+1 = 1/(ab+a+1)$. Therefore $c+1 \in \not\square_q$ and so $c \in S$ which implies $P_2 \in B$.

Using Lemma 1.6 again, we find $d = -b/[(b+1)(a+1)]$, but $b \in \not\square_q$ and $b+1, a+1 \in \square_q$ gives $b/[(b+1)(a+1)] \in \not\square_q$ by Lemma 3.7 and thus $d \in \square_q$. Therefore, $d \in S$ and $P_3 \in B$.

Case (ii): Let $P_0 = (0:1:a)$, $P_2 = (c : -(c+1) : 0)$ where $a, c \notin S$. Thus $a \in \not\square_q$, $a+1 \in \square_q$, $c \in \not\square_q$, and $c+1 \in \square_q$. Let $P_1 = (b+1 : 0 : -1)$, and $P_3 = (-1 : -d : d+1)$. We want to show that $P_1, P_3 \in B$. Using Lemma 1.6, we find $b = -(c+ac+a)/[a(c+1)]$ and thus $b+1 = -c/[a(c+1)]$ but $c/[a(c+1)] \in \square_q$ by Lemma 3.7, thus $b+1 \in \not\square_q$ by Lemma 3.8. Therefore, $b \in S$ and $P_2 \in B$.

Using Lemma 1.6 again, we find $d = -(c+ac+a)/[c(a+1)]$. Interchanging a and c, the argument is the same as in the previous paragraph.

■

With less effort we get the following slightly weaker result.

Theorem 3.12 *Let $q \geq 7$ and q prime. There is a proper double blocking set in $PG(2,q)$ of size $3q+2$.*

Proof: Let $l_0 = [1:0:0]$, $l_1 = [0:1:0]$, $l_2 = [0:0:1]$ and let $G \leq PGL(3,q)$ be the group of 3×3 permutation matrices; that is $G \cong S_3$. Also, refer to the points $(1:0:0)$, $(0:1:0)$, $(0:0:1)$ as vertices. Let T denote the set of $3q$ points on the lines l_0, l_1, l_2. Note that G is an automorphism group of T. Now take

$$B = (T \setminus U) \cup V$$

where $U = G((0:1:2))$ and $V = [G((1:2:3)) \cup G((1:-2:-2))] \setminus \{(-2:-2:1)\}$. Notice that $|U| = 6$ and $|V| = 6+3-1 = 8$ and $|B| = 3q+2$. We begin by considering the set $B' = B \cup \{(-2:-2:1)\}$, and later it will be shown that the point $(-2:-2:1)$ can be safely removed from the set. Notice that G is an automorphism group of B'.

By Theorem 3.4, T is a double blocking set, so we need only consider those points that pass through one or more of the points in U. For any line $l \in PG_1(2,q)$ such that $|l \cap U| > 0$, there are three cases to consider; the case where l passes through exactly one point in U and no vertex; the case where l passes through a vertex and a point in U; and the case where l passes through at least 2 points in U.

Case (i): If l passes through exactly one point P in U and no vertex, then $|l \cap B| = 2$ since it must intersect the remaining two lines (those lines that do not contain P) at two distinct points.

Case (ii): There are six lines that pass through one point in U and a vertex and these six lines all lie in the same G-orbit ($G([2:0:-1])$ in Table 3.1). Since B' is the union of G-orbits, by Theorem 2.22, we need only consider the size of the intersection of one line of the six lines. From Table 3.1 we see that $|l \cap B'| \geq 2$ for $l \in G([2:0:-1])$. Note also that $G((-2:-2:1)) \cap [2:0:-1] = \emptyset$ and $(1:2:3) \notin [2:0:1]$ so $|l \cap B'| \leq q-1$ since $q \geq 7$.

Case (iii): There are $\binom{6}{2} = 15$ lines that pass through at least two points in U. Three of these lines are the l_0, l_1 and l_2 and since $T \cap V = \emptyset$, we have that $|l_i \cap B| = q-1 \geq 2$ since $q \geq 7$, for $i \in \{0,1,2\}$. The remaining 12 lines lie in the three G-orbits shown in Table 3.1. From Table 3.1 we see that for l satisfying $|l \cap U| \geq 2$, $|l \cap B'| \geq 2$. Also note that $G((1:-2:$

		Line	Point 1	Point 2
Case (ii)	**G([2:0:-1])**	**(0:1:0)(1:0:2)**	**(1:3:2)**	**(0:1:0)**
		(0:0:1)(1:2:0)		
		(1:0:0)(0:1:2)		
		(1:0:0)(0:2:1)		
		(0:0:1)(2:1:0)		
		(0:1:0)(2:0:1)		
Case (iii)	**G([1:1:-2])**	**(0:2:1)(2:0:1)**	**(1:3:2)**	**(1:-1:0)**
		(1:0:2)(1:2:0)		
		(0:1:2)(2:1:0)		
	G([-2:1:-2])	**(0:2:1)(1:2:0)**	**(1:-2:-2) or (-2:-2:1)**	**(1:0:-1)**
		(0:1:2)(1:0:2)		
		(2:1:0)(2:0:1)		
	G([4:1:-2])	**(0:2:1)(1:0:2)**	**(1:2:3)**	**(1:-4:0)**
		(0:1:2)(1:2:0)		
		(2:0:1)(0:1:2)		
		(2:1:0)(0:2:1)		
		(1:0:2)(2:0:1)		
		(1:2:0)(2:0:1)		

Table 3.1: Orbits On Lines and Repair Points

$-2)) \cap [1:1:-2] = \emptyset$, $G((1:2:3)) \cap [-2:1:-2] = \emptyset$, and $G((1:-2:-2)) \cap [4:1:-2] = \emptyset$. So for $l \in PG_1(2,q)$ such that $|l \cap U| \geq 2$, $2 \leq |l \cap B'| \leq q-1$.

Finally, it is clear that all the points in $G((1:2:3))$ are require to repair the lines in $G([4:1:-2])$, but this is not true of the points of $G((1:-2:-2))$. It turns out that $|G((1:-2:-2)) \cap [-2:1:-2]| = 2$ and so only 2 points in the orbit need be chosen in order repair the lines in $G([-2:1:-2])$. Thus we remove the point $(-2:-2:1)$ from the set B' to obtain B.

■

Chapter 4

Codes and Caps

In the Introduction of Chapter 1.1, a relationship between caps and codes was suggested, motivating the desire to find caps in finite projective geometries. In Section 4.2, a known relationship between caps in $PG(r-1, q)$ and codes with distance at least 4 is explicitly shown. It turns out that the larger the cap is, the better the code, and so ultimately, we want to find maximum caps in a given projective geometry. In Section 4.3, a brief survey of known caps is given. In Section 4.4, the method of prescribed automorphisms is applied directly to the problem of finding large caps in $PG(r-1, q)$. Section 4.5 shows an algorithm written by the author to solve the resulting 0-1 integer linear programming (LP) problem. Finally, Section 4.6 gives the computational results found by the author.

We begin with the coding theory background presented in Section 4.1.

4.1 Coding Theory Background

The following required definitions can be found in any text that offers an introduction to coding theory including [MS77].

Definition 4.1 *A $[n, k]$* **linear code over** *$\mathbb{F}_q$, or $[n, k]_q$-code, C, is a k-dimensional linear subspace of $\mathbb{F}_q^n$. If $q = 2$ then C is called a binary linear code or an $[n, k]$*-**code**. *The elements of C are called* **codewords**, *and the* **block length** *of C is n. We call k the* **dimension** *of C,* *and note that such a code has q^k codewords.*

Throughout this chapter, let C denote an $[n,k]_q$-code. Note again that C is a linear code as some of the following definitions and theorems may not apply to nonlinear codes.

Definition 4.2 *The* **rate** *or* **efficiency** *of C is k/n and is denoted by $R(C)$.*

The larger the value of $R(C)$ is, the more "efficient" the code is considered to be. The efficiency of a code measures the relationship between the dimension of the code, and the length of a codeword. If the dimension k is considered to represent the amount of information that can be sent in a codeword, then the remaining $(n-k)$ elements can be considered to be the elements that provide the reliability, or error-correcting ability of the code. Clearly, these two desirable traits of a code are competing; that is, for a fixed code length n, one can only be increased at the expense of the other. So while a high efficiency is desirable, it comes at the cost of reliability; a code with efficiency equal to 1 has no error-correcting ability. The codes studied in this chapter have fixed error-correcting abilities, but attempt to improve efficiency by increasing both n and k. These codes are discussed further in Section 4.2.

Definition 4.3 *The* **Hamming weight** *of a vector $x = (x_1, x_2, \dots, x_n) \in \mathbb{F}_q^n$ is the number of nonzero x_i, $1 \le i \le n$, and is denoted by* $\mathrm{wt}(x)$.

Definition 4.4 *The* **Hamming distance** *between two vectors $x = (x_1, x_2, \dots, x_n)$ and $y = (y_1, y_2, \dots, y_n) \in \mathbb{F}_q^n$, denoted by* $\mathrm{dist}(x,y)$, *is the number of places in which x and y differ; that is,* $\mathrm{dist}(x,y) = \mathrm{wt}(x-y)$.

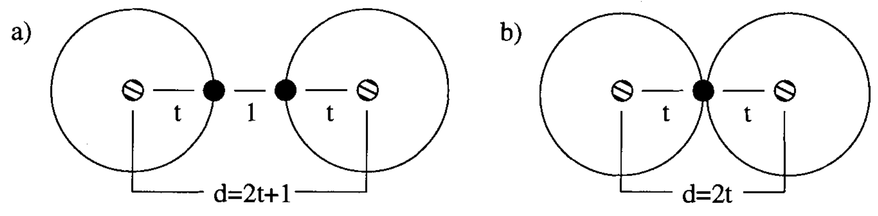
Definition 4.5 *The* **minimum distance** *of C is*

$$\min_{x,y\in C, x\neq y} \mathrm{dist}(x,y) = \min_{x,y\in C, x\neq y} \mathrm{wt}(x-y)$$

and is denoted by d.

Sometimes C is called an $[n,k,d]_q$-code.

The following theorem and proof is taken directly from [MS77, pg. 10].



⊘ Denotes a codeword.

● Denotes a noncodeword.

Figure 4.1: Codes With Minimum Distance d.

Theorem 4.6 *Let $t \in \mathbb{Z}$ be nonnegative. An $[n,k]_q$-code with minimum distance d can correct t errors if $d = 2t+1$ and $t-1$ errors if $d = 2t$.*

Proof: Suppose $d = 2t+1$ for some positive integer t. A **ball** of radius r and centre x is the set of vectors $y \in \mathbb{F}_q^n$ such that $\text{dist}(x,y) \leq r$. Consider the set of balls with centre in C and $r = t$. If $u, v \in C$ are the centres of two such balls and $x \in \mathbb{F}_q^n$ such that $\text{dist}(x,u) \leq t$ then by the triangle inequality, $\text{dist}(x,v) \geq t+1$, so that the balls are disjoint. So if codeword x is transmitted and vector y is received and $\text{dist}(x,y) \leq t$, y will be closest to x. If $\text{dist}(x,y) > t$ then y may not be closest to x. In other words, a code with minimum distance $d = 2t+1$ can detect and correct up to $t = (d-1)/2$ errors. Thus for d odd, the $[n,k]_q$-code can correct t errors. (See Figure 4.1a.)

If $d = 2t$ is even, by the same argument, at least one pair of balls of radius t centered at u and v, where $u, v \in C$ will overlap at one point, so that C can detect up to t errors, but can only correct $t-1$ errors. (See Figure 4.1b.)

■

Codes are often measured by their error-correcting ability; given a block length n, and a dimension k, what is the best error correcting ability a code can have. This is known as the packing problem; what is the largest possible radius t of disjoint balls centered at codewords. As shown in Theorem 4.6 this is the error-correcting ability of the code. There is a second common way of measuring codes that looks at the opposite problem.

Definition 4.7 *The* **covering radius** *of C, denoted by ρ, is the smallest integer r such that* *for every* $v \in \mathbb{F}_q^n$, $\text{dist}(x, v) \leq r$ *for some* $x \in C$.

Geometrically, the covering radius can be considered as the smallest integer r such that every vector in $\mathbb{F}_q^n$ lies in at least one ball of radius r centered at a codeword. In this way, the balls of radius r jointly cover the vector space $\mathbb{F}_q^n$. The covering radius is used to approach problems such as data compression and information hiding as the packing problem is used to approach error correcting. The following Proposition shows a relationship between the covering radius and the minimum distance of a code.

Proposition 4.8 *The covering radius* ρ *of a code with minimum distance* d *satisfies* $d \leq 2\rho + 1$.

Proof: Let $c_1, c_2 \in C$ such that $\text{dist}(c_1, c_2) = d$ and let B_c denote the ball of radius ρ around the codeword c. Consider $v \in \mathbb{F}_q^n$ such that $\text{dist}(c_1, v) + \text{dist}(v, c_2) = d$, and suppose v does not lie in either B_{c_1} nor in B_{c_2}, then there exists $c_3 \in C$ such that $\text{dist}(v, c_3) \leq \rho$. So, $\text{dist}(c_3, v) < \text{dist}(v, c_1)$, thus $\text{dist}(c_2, c_3) \leq \text{dist}(c_2, v) + \text{dist}(v, c_3) < \text{dist}(c_2, v) + \text{dist}(c_1, v) = d$, a contradiction. So, v lies in at least one of B_{c_1} or B_{c_2}. Now, suppose $d > 2\rho+1$, then there exists $u \in \mathbb{F}_q^n$ such that $\text{dist}(c_i, u) \geq \rho + 1$, for $i = 1, 2$, and $\text{dist}(c_1, u) + \text{dist}(u, c_2) = d$, but then u would lie outside of both B_{c_1} and B_{c_2}. So $d \leq 2\rho + 1$.

■

If $d = 2\rho + 1$ in Proposition 4.8, then C is called a **perfect code**. A perfect code has covering radius equal to its error-correcting ability, so it solves both problems optimally. While these are desirable codes, the number of such codes is extremely small; the family of Hamming codes $[\frac{q^r-1}{q-1}, \frac{q^r-1}{q-1} - r, 3]_q$ for integer $r \geq 2$, the binary Golay code $[23, 12, 7]_2$ and the ternary Golay code $[11, 6, 5]_3$ are the only non-trivial examples. Trivial perfect codes include $[n, n, 1]_q$ codes and the repetition code $[n, 1, n]_2$ when n is odd.

Definition 4.9 *A* **quasi-perfect code** C *is a code where* $d = 2\rho$.

In Section 4.2 it will be shown that the codes searched for in this chapter can be extended into either perfect codes, or more commonly, quasi-perfect codes.

Definition 4.10 *Let* G *be a* $k \times n$ *matrix over* $\mathbb{F}_q$, *then* G *is called a* **generator matrix** *for* C *if and only if the rows of* G *form a basis for* C.

Definition 4.11 *Let H be an $(n-k) \times n$ matrix over $\mathbb{F}_q$ with rank $n-k$, then H is a called a* **parity check matrix** *for C if and only if for every $c \in C$, $Hc^T = \mathbf{0}$.*

Definition 4.12 *Let C be a code with generator matrix G and parity check matrix H. The* **dual** *of C, denoted $C^{\perp}$, is the code with generator matrix H and parity check matrix G.*

Note that

$$C^{\perp} = \{u \in \mathbb{F}_q^n \mid u \cdot v = 0 \text{ for all } v \in C\},$$

and $C^{\perp}$ is an $[n, n-k]_q$-code.

The following theorem can be found in [MS77, pg. 33]. As an exercise we present a more explicit proof.

Theorem 4.13 *Let H be a parity check matrix for a linear code C. Then C has minimum distance d if and only if any set of $d-1$ columns of H is linearly independent, and there is at least one set of d columns that is linearly dependent.*

Proof: Let n be the block length of C. Suppose C has minimum distance d, then there exists some codeword $x = (x_1, x_2, \ldots, x_n) \in C$ such that $Hx = \mathbf{0}$ and $\text{wt}(x) = d$. Let $H = [h_1|h_2|\ldots|h_n]$ where $h_i \in \mathbb{F}_q^{n-k}$, then $Hx = h_1x_1 + h_2x_2 + \ldots + h_nx_n = \mathbf{0}$. Since x has weight d there are d nonzero entries in x, say $x_{i_1}, x_{i_2}, \ldots, x_{i_d}$ and the remaining entries are zero. Then $Hx = h_{i_1}x_{i_1} + h_{i_2}x_{i_2} + \ldots + h_{i_d}x_{i_d} = 0$, so that the set $\{h_{i_1}, h_{i_2}, \ldots, h_{i_d}\}$ is a set of d linearly dependent columns of H. Now, suppose there exists a set of $d-1$ linearly dependent columns in H, say $\{h_{j_1}, h_{j_2}, \ldots, h_{j_{(d-1)}}\}$, then for some $c_1, c_2, \ldots, c_{d-1}$ not all zero, we have $c_1h_{j_1} + c_2h_{j_2} + \ldots + c_{d-1}h_{j_{d-1}} = \mathbf{0}$, so that if $y = y_1y_2\ldots y_n$ with $y_{j_k} = c_k$ for $1 \le k \le d-1$, and all other entries 0, then y is a codeword with $\text{wt}(y) < d$, a contradiction.

■

4.2 A Relationship Between Caps and Codes

Recall from Definition 1.21 that a cap is a set of points B in $PG(n, q)$ such that no three points of B are collinear and an m-cap is a cap with m points. Also recall from Definition 1.22 that a complete cap is a cap which is not properly contained within any other cap in $PG(n, q)$, and a maximum cap is a cap of maximum size in $PG(n, q)$. The relationship between caps and linear codes is discussed in this section.

Let $B = \{P(h_1), P(h_2), \ldots, P(h_n)\}$ be an n-cap with $n \geq 4$ in $PG(r-1, q)$, then, by Theorem 4.13, the matrix H with columns $\{h_1, h_2, \ldots, h_n\}$ is a parity check matrix for an $[n, n-r]_q$ code C with distance $d \geq 4$. Such a code has q^{n-r} codewords and efficiency $R(C) = (n-r)/n$. Since r is fixed by the choice of $PG(r-1, q)$, increasing n results in a more efficient code with a larger number of codewords. So it is desirable to try to maximize the size of the cap B.

Theorem 4.14 *Let $H = \{h_1, h_2, \ldots, h_n\}$ be the columns of a parity check matrix of an $[n, n-r]_q$ code C with $d \geq 4$. Then $B = \{P(h_1), P(h_2), \ldots, P(h_n)\}$ is a complete cap in $PG(r-1, q)$ if and only if C has covering radius $\rho = 2$.*

Proof: Suppose $\rho = 2$. Let $y \in \mathbb{F}_q^r$. Since H has rank $n-k$, $y = Hx^T$ for some $x \in \mathbb{F}_q^n$. Now, since $\rho = 2$, there exists $z \in \mathbb{F}_q^n$ such that $(x - z) \in C$ and $\text{wt}(z) \leq 2$. So, $H(x-z)^T = 0$ which implies that $y = Hz^T$ and since $\text{wt}(z) \leq 2$ it follows that H is a complete cap. The reverse argument shows us that if H is a complete cap, then $\rho \leq 2$, so it only remains to show that equality holds. By Proposition 4.8, $d \leq 2\rho + 1$, thus $\rho \geq 2$.

■

By Proposition 4.8, it is clear that if $\rho = 2$ then H forms a parity check matrix of a perfect or quasi-perfect code.

It should be noted that any cap can be extended to a complete cap in polynomial time. The simplest algorithm to do so would be to find a point in $PG(n, q)$ that does not lie on a line that intersects the cap at exactly two points and add that point to the cap. If the new set is not a complete cap, then the process should be repeated.

4.3 Known Maximum Caps

In the previous section, it was shown that complete caps form the parity check matrix of perfect or quasi-perfect codes and that the larger the cap is, the better the code is considered to be, so it is desirable to find caps of maximum size. However, finding such maximum caps in $PG(n, q)$ is still an open problem for all but a very restricted number of finite projective geometries (see Theorem 4.15, Theorem 4.19, Theorem 4.21, and the comments following Theorem 4.21) . This section briefly summarizes the relevant information of two survey papers, one by J. Bierbrauer [Bie03] and the second by J.W.P. Hirschfeld and L. Storme [HS01], and provides some proofs and examples. For simplicity of notation, let $m_2(n, k)$ denote the maximum size of a complete cap in $PG(n, k)$.

Theorem 4.15 *[Bos47] The maximum size of a cap in $PG(N, 2)$ is $m_2(N, 2) = 2^N$.*

Proof: Let H be a set of points in $PG(N, 2)$ constructed by removing a hyperplane from the set of all points. By Theorem 1.12 a line in $PG(N, 2)$ is either contained completely in the hyperplane, or intersects the hyperplane at exactly one point. Since every line has three points, H is a cap. By Theorem 1.8, the number of points contained in a hyperplane in $PG(N, 2)$ is $\begin{bmatrix} (N-1)+1 \\ 0+1 \end{bmatrix}_q = (2^N - 1)$ points so that $|H| = [(2^{N+1} - 1) - (2^N - 1)] = 2^N$.

Now, suppose there exists a cap H' such that $|H'| \geq 2^N + 1$. Choose a point $y \in H'$, then for every $x \in H'$, there is a point $x + y \notin H'$ and if $x \neq x'$, then $x + y \neq x' + y$. Hence we can count $2(2^N) + 1 = 2^{N+1} + 1$ distinct points which is more than the total number of points in $PG(N, 2)$.

■

The proof of Theorem 4.15 suggests a construction for a complete cap in $PG(N, 2)$ and in fact, as Theorem 4.17 will show, the only complete caps in $PG(N, 2)$ are those constructed by removing a hyperplane from the set of points in $PG(N, 2)$.

Lemma 4.16 *If C is a cap of size 2^N in $PG(N, 2)$, then C has no tangent line. That is, no line intersects C exactly once.*

Proof: Suppose there exists a line l such that l intersect C at exactly one point, call that point y. By Corollary 1.9 we know that $PG(N, 2)$ has $2^{N+1} - 1$ points and each line has 3 points. For each $x \in C$ such that $x \neq y$, there exists a point $x + y$ not in C, since C is a cap, and not on l, since x is not on l. Counting, we get $2(2^N - 1) + 3 = 2^{N+1} + 1$ where the additional 3 points are the points on l. This is a contradiction, since there are only $2^{N+1} - 1$ points in $PG(N, 2)$.

■

Theorem 4.17 *[Seg59] If C is a cap of size 2^N in $PG(N,2)$, then the complement of C, $\overline{C}$, is a hyperplane.*

Proof: By Lemma 4.16, if $x, y \in \overline{C}$ then $x + y \in \overline{C}$. Also if $x \in \overline{C}$ then $cx \in \overline{C}$ or $cx = 0$ for $c \in \mathbb{F}_2$. Thus $\overline{C}$ forms a subspace of $PG(N,2)$ of size $2^{N+1} - 1 - 2^N = 2^N - 1$. Since the only subspaces of this size are hyperplanes, it must be that $\overline{C}$ is a hyperplane.

■

The following example shows that not all complete caps need be maximum.

Example 4.18 Let $H = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$ be a parity check matrix for a $[9,4]_2$ code and let H' be the set of columns of H. Since every nonzero vector in $\mathbb{F}_2^5$ which is not in H' can be written as a linear combination of exactly two elements of H', H' forms a complete cap in $PG(4,2)$. By Theorem 4.15, H' is not maximum.

The proof for the following theorem can be found in [Hir98, pg.177].

Theorem 4.19 *The maximum size of a cap in $PG(2,q)$ is $m_2(2,q) = q + 1$ if q is odd, and $m_2(2,q) = q + 2$ if q is even.*

Example 4.20 Let H be the set of points $(x_0 : x_1 : x_2)$ in $PG(2,q)$ that satisfy $x_0^2 - x_1x_2 = 0$. The set H has $q+1$ points, no three of which are collinear. If q is odd, then H is a complete cap. If q is even the set of tangents of H intersect at exactly one point N, thus $H \cup \{N\}$ is a complete cap.

The previous example shows a construction for maximum caps in $PG(2,q)$, and in fact the same construction can be made using the rational points of any conic. It happens that for q odd, any maximum cap must be the set of rational points of a conic [Seg54, Seg55].

For q even, however, it is known that not all maximum caps are the union of a conic and the intersection of the tangent lines of the conic.

Theorem 4.21 *[Bos47, q odd][Qvi52, q even] The maximum size of a complete cap in $PG(3,q)$ for $q > 3$ is $m_2(3,q) = q^2 + 1$.*

As in the case for $PG(2,q)$, there is a geometrical construction for maximum caps in $PG(3,q)$. The rational points of an elliptic quadric form a maximum cap in $PG(3,q)$. Like the case with $PG(2,q)$, not all maximum caps in $PG(3,q)$ are formed in this way. For example, in $PG(3,2^{2e+1})$, $e \geq 1$, there is another construction known as the Tits ovoid [Tit62].

G. Pellegrino offers a geometric proof for $m_2(4,3) = 20$ [Pel70]. The Hill cap, named after its discoverer R. Hill, is the largest cap in $PG(5,3)$ and has size 56 [Hil73], thus, $m_2(5,3) = 56$. J. Bierbrauer and Y. Edel were able to prove that $m_2(4,4) = 41$ with the aid of a computer program [EB99]. It is expected that any further discoveries will be largely based on computational effort.

Though no other values of $m_2(n,q)$ are known, their upper bounds are known. The following two theorems are found in [Bie03].

Theorem 4.22 *Let $n \geq 3$ and $q > 2$. Then*

$$m_2(n,q) \leq q^{n+1}\frac{n+2}{2(n+1)^2}.$$

Theorem 4.23 *For $n \geq 3$ and $q > 3$,*

$$m_2(n,q) \leq \left(\frac{n+1}{n^2} + \frac{3n}{2q(n-1)^2}\right) q^n.$$

For certain n and q, better upper bounds are known, and the interested reader is referred to [HS01] for a complete listing.

4.4 Prescribed Automorphisms and Maximum Caps

In Chapter 2, the foundations for using prescribed automorphisms to help find interesting point sets in $PG(n,q)$, were introduced. In this section, these foundations are applied directly to the problem of finding caps in $PG(n,q)$. It should be noted that a similar approach was used in [Bra05] to find the generator matrix for linear codes with a prescribed minimum distance.

Let $P_1,\ldots,P_t$ denote the $t=(q^{n+1}-1)/(q-1)$ distinct points in $PG(n,q)$. The problem of finding maximum caps in $PG(n,q)$ is equivalent to the following 0-1 integer LP problem:

Maximize

$$|B| = x_1 + x_2 + \ldots + x_t,$$

subject to, for each line l_j in $PG(n,q)$

$$c_{j1}x_1 + c_{j2}x_2 + \ldots + c_{jt}x_t \leq 2,$$

where $x_i \in \{0,1\}$ for all i and

$$c_{ji} = \begin{cases} 1 \text{ if line } l_j \text{ is incident with } P_i, \\ 0 \text{ otherwise.} \end{cases}$$

The point P_i is in the set B if and only if $x_i = 1$.

Such a problem becomes intractable very quickly. For example, $PG(5,3)$ has 364 points and 11011 lines, so the total number of possible solutions is 2^{364}, each of which must be tested against the 11011 constraints. The method of prescribed automorphisms can reduce the problem size dramatically; for example, the cyclic group generated by the element

$$g = \begin{pmatrix} 1 & 2 & 2 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 2 \\ 1 & 1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 2 & 0 & 1 & 2 & 2 \\ 2 & 1 & 2 & 2 & 0 & 1 \end{pmatrix}^T \in PGL(6,3)$$

reduces the original problem with 364 points and 11011 lines to 15 orbits on points and 431 orbits on lines, but we may lose the ability to find an optimal solution to the original problem, as was shown in Example 2.26, or even to know that an optimal solution has been found. Ideally, the goal is to find a group that will create a small number of orbits, but will also be an automorphism of a large cap, and thus the new 0-1 integer LP problem will still produce a good solution for the original problem.

In Section 2.5, strategies for choosing groups were discussed. Since little is known about the structure of maximum caps in $PG(n,q)$ the strategy used for this problem was to use cyclic subgroups of $PGL(n+1,q)$. Using cyclic subgroups can often be a good place to start since there are relatively few cyclic subgroups up to conjugacy, and since any 'good' group will have cyclic subgroups. As shown in Section 2.4, once the group is chosen and applied to the original 0-1 integer LP problem, a new 0-1 integer LP problem is found.

Let $\omega_1,\dots,\omega_k$ denote the k distinct G-orbits of $PG_0(n,q)$ for some $G \leq PGL(n+1,q)$, and let $\Omega_1,\dots,\Omega_K$ denote the K distinct G-orbits of $PG_1(n,q)$.

Maximize

$$|B| = |\omega_1|x_1 + |\omega_2|x_2 + \dots + |\omega_k|x_k,$$

subject to, for each Ω_j

$$c_{j1}x_1 + c_{j2}x_2 + \dots + c_{jk}x_k \leq 2,$$

where $x_i \in \{0,1\}$, and $c_{ji} = |l \cap \omega_i|$, for any line $l \in \Omega_j$.

Example 4.24 Theorem 4.15 states that $PG(3,2)$ has a maximum cap size of 8. However, consider $g = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$. The cyclic group generated by g is given by $\langle g \rangle =$
$\left\{ g, g^2 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, g^3 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}, g^4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, g^5 = I \right\}$. This group produces the reduced 0-1 Integer LP problem:

Maximize

$$|B| = 5x_1 + 5x_2 + 5x_3,$$

subject to

$$1x_1 + 1x_2 + 1x_3 \leq 2,$$

$$0x_1 + 1x_2 + 2x_3 \leq 2,$$

$$1x_1 + 0x_2 + 2x_3 \leq 2,$$

$$0x_1 + 2x_2 + 1x_3 \leq 2,$$

$$2x_1 + 0x_2 + 1x_3 \leq 2,$$

$$1x_1 + 1x_2 + 0x_3 \leq 2,$$

$$1x_1 + 2x_2 + 0x_3 \leq 2,$$

where $x_i \in \{0, 1\}$, for all i.

An optimal solution to this problem is associated with the cap

$$B = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\},$$

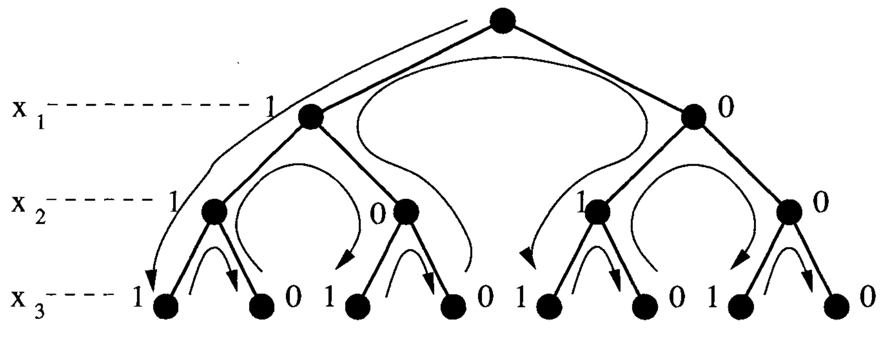
which is clearly not maximum, but is complete. Also, it should be noted that this is the only cyclic subgroup (up to conjugacy) that generates B.

4.5 Solving the Integer Linear Programming Problem

Though the method of prescribed automorphisms may significantly reduce the size of the original 0-1 integer LP problem, the new problem must still be solved. It is known that the 0-1 integer LP problem is NP-complete ([GJ79], Page 245), so there is no known polynomial-time algorithm to solve it. Instead, we look to an exponentially-timed algorithm and attempt to improve its efficiency by using knowledge of the current problem being solved and adapting the algorithm accordingly.

The exponential-time algorithm used here is a backtracking algorithm, and a method known as pruning is used in order to improve the efficiency of the backtracking algorithm. Before we describe what pruning is and how it is specifically applied to this problem, we give a brief overview of a general backtracking algorithm.

Let P be a 0-1 integer LP problem. Recall from Definition 2.23 that a feasible solution is a 0-1 vector such that all the constraints of P are satisfied. Let $X = (x_0, x_1, \ldots, x_n)$ be a not necessarily feasible solution of P. A backtracking algorithm is a recursive algorithm



Search Order

x_1	x_2	x_3
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

Figure 4.2: An Example Search Tree for a Backtracking Algorithm.

that performs a depth-first search of a binary tree where the i^{th} level of the tree represents the i^{th} variable x_i. Once a solution has been processed, the algorithm backtracks to find the next solution (Figure 4.2). Each recursive call of the backtracking algorithm takes the problem P, the current solution state X, and the current depth of the search tree l as parameters, and processes the solution X when $x_l = 0$ and when $x_l = 1$. If $l = n + 1$, processing X means computing the objective function value of X and comparing it to the best solution so far. If $l < n + 1$ then processing X means recursively calling the backtracking algorithm. The backtracking algorithm visits every possible solution, so if the problem has n variables and m constraints, the algorithm will generate 2^n solutions and will test every solution on m constraints and so has a complexity of order $m2^n$. Since every possible solution is visited, the algorithm is guaranteed to find an optimal solution if one exists. Recall however, that in the context of this thesis, this is a reduced problem, and the backtracking algorithm will not guarantee a solution to the original problem.

Very important to the 0-1 ILP problem and by extension, the backtracking algorithm, is the concept of feasibility. A standard backtracking algorithm tests the feasibility of the current solution when $x_l = 1$ at every level of recursion. If X is a feasible solution then the algorithm will continue normally, but if X is not a feasible solution, then the algorithm will need to act accordingly. In the case where the inequalities are $\leq$, as they are in the caps problem, the algorithm would stop searching that branch of the search tree; that is, if $X = [\overline{x}_1, \ldots, \overline{x}_l, 0, \ldots, 0]$ is not a feasible solution, the algorithm will not consider any solution where $X = [\overline{x}_1, \ldots, \overline{x}_l, x_{l+1}, \ldots, x_n]$. By testing for feasibilty early, that is before

```
Algorithm: Backtrack($P$, $X$, $l$)

global optP, optS

    if $l = n + 1$
        curP = $\sum_{i=1}^{n} w_i x_i$;
        if (curP>optP) then
            optP$\leftarrow$curP;
            optS$\leftarrow X$;
    else
        $x_l \leftarrow 1$
        $L$ = Feasibility($x_l$)
        if(Prune($x_l$)>optP) Backtrack($P, X, x_l.Right$)

        $x_l \leftarrow 0$
        DancingLinks($L$)
        if(Prune($x_l$)>optP) Backtrack($P, X, x_l.Right$)
```

Figure 4.3: Backtracking Algorithm with Pruning

$l = n$, the number of complete solutions visited may be reduced so that the algorithm is no longer visiting all 2^n solutions. The feasibility testing in the backtracking algorithm used in the algorithm in this thesis differs slightly from the standard method, in that rather than testing a current solution it takes advantage of an algorithm developed by Knuth [Knu00] called Dancing Links, and tests future solutions so that any solution visited is a feasible solution. This method for testing feasibility is discussed in further detail below.

Figure 4.3 gives a brief overview of the algorithm used in this thesis. It should be noted that initially, $X = \mathbf{0}$, optS $= \mathbf{0}$, and optP$= 0$.

There are three subprocesses being performed in the backtracking algorithm; they are Feasibility(x_l), DancingLinks(L) and Prune(x_l). Before discussing these three subprocesses in detail, we look at what happens when the terminal case $l = n + 1$ is reached. At this stage, the current solution X is compared to the best known solution so far. If the objective function value for the current solution *curP* is better than *optP*, the objective function value for the best solution so far, then the best known solution *optS* is replaced by the current solution X.

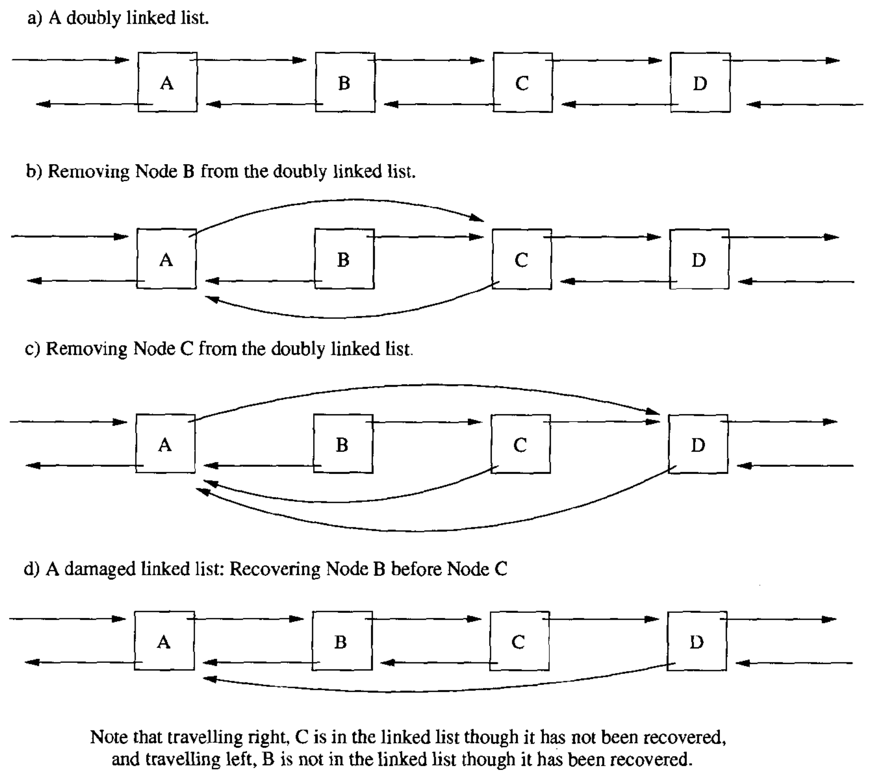

a) A doubly linked list.
A
B
C
D
b) Removing Node B from the doubly linked list.
c) Removing Node C from the doubly linked list.
d) A damaged linked list: Recovering Node B before Node C
Note that travelling right, C is in the linked list though it has not been recovered, and travelling left, B is not in the linked list though it has been recovered.

Figure 4.4: A Linked List

Before describing the remaining subprocesses, an overview of the Dancing Links Algorithm [Knu00] is given. The first part, Remove, is performed during the Feasibility(x_l) subprocess, and the second part, Recovery, is performed during the DancingLinks(L) subprocess. The Dancing Links Algorithm allows for an element stored in a doubly linked list to be removed from the list and later recovered in its original position. For this reason, each X-coordinate is stored in a structure known as a node. A node in a doubly linked list points to the node before it in the list (using the pointer Left), and the node after (using the pointer Right) (see Figure 4.4a). Removing a node from a doubly linked list is done in the following manner.

```
Remove(node b)
    b.Left.Right=b.Right
    b.Right.Left=b.Left
```

While the node has been removed from the doubly linked list, meaning that it cannot be reached from any node in the list, it still points to nodes in the list, indicating its original position. To recover the removed node, the following procedure is applied;

```
Recovery(node b)
    b.Left.Right=b
    b.Right.Left=b
```

Together, these two procedures, Remove and Recovery, form the Dancing Links Algorithm. It should be noted that if more than one node is removed before recovery takes place, then nodes should be recovered in the reverse order of removal, in order to ensure the integrity of the linked list (see Figure 4.4d).

The simple recovery process used in the Dancing Links Algorithm makes it ideal for testing the feasibility of all vectors below the current vector in the search tree, and not just the current vector as is normally done in a backtracking algorithm. If x_l is set to 1, any x_i, $i > l$ that can no longer be set to 1 and be part of a feasible solution is then considered to be 0 and is removed from the linked list so that it will not be visited by the backtracking algorithm. Any x_i considered for removal, but not removed is called **live**. Feasibility(x_l) returns a stacked list L which contains the location of all removed variables. When x_l is set to 0, the DancingLinks(L) subprocess recovers the removed x_i's using the Recovery algorithm. Since L is a stacked list, the x_i's are returned to linked list in the reverse order of removal, thus ensuring the integrity of the linked list. It is important to note that this method of feasibility testing means that any solution $X = [x_1, \dots, x_l, 0 \dots 0]$ visited is a feasible solution which is not necessarily true when standard feasibility testing is used.

The final subprocess is the bounding function, Prune(x_l). A bounding function assigns a value b to a partial solution $X_l = [a_1, \dots, a_l]$ such that the objective function value of the best solution containing X_l, $X_b = [a_1, \dots, a_l, x_{l+1}, \dots, x_n]$ is less than or equal to b. If b is less than or equal to the objective function value of the best known solution so far, then a better solution does not exist in the subtree rooted at the partial solution X_l and no further testing in that subtree need be done and so the search tree is 'pruned' of this subtree, reducing the number of complete solutions visited.

The algorithm in this thesis implements one of two bounding functions. The first is called the Standard Bounding Function. It takes a partial solution and sets all remaining live X-coordinates to 1 without any further feasibility testing. The objective function value of this solution becomes the bound b. It is not possible for any solution containing the current partial solution to be better than this bound.

The second bounding function, called the Modified Bounding Function, takes advantage of the structure of the problem for caps, using the fact that the right-hand-side of every constraint is less than or equal to 2 to store and easily retrieve information for computing the bound. The Modified Bounding Function chooses a single constraint and finds the best objective function value based on this constraint and the 0-1 constraints. This objective function value becomes the bound. For each constraint, three arrays are created C_2, C_1 and C_0. The array C_i, $i \in \{0, 1, 2\}$ contains the variables x_j such that the coefficient of x_j in the constraint is i. Each array is sorted in descending order according to the objective function coefficient of x_j so that a greedy algorithm can be applied efficiently. Let o_i be the objective function coefficient of the variable x_i. Let x_k be the first live variable in the array C_2 and let $b_2 = o_k$. Let x_{k1} and x_{k2} be the first two (respectively) live variables in C_1, and let $b_{21} = o_{k1} + o_{k2}$ and $b_{11} = o_{k1}$. Let b_0 be the sum of the objective function coefficients of all live variables in C_0. Finally, let $P = \sum_{t=1}^{l} o_t x_t$ be the objective function value of the current partial solution. In order to compute the bound b for the Modified Bounding Function given constraint L_j, we consider the slack $s_j = 2 - \sum_{t=1}^{n} c_{jt} x_t$ associated with L_j. If $s_l = 2$ then $b = P + \max(b_2 + b_0, b_{21} + b_0)$, if $s_l = 1$ then $b = P + b_{11} + b_0$, and if $s_l = 0$ then $b = P + b_0$.

Example 4.25 To demonstrate the two different bounding functions, consider the following linear programming problem.

Maximize

$$|B| = 4x_1 + 2x_2 + 2x_3 + x_4,$$

subject to

$$\begin{aligned} C_1 : \; 1x_1 + 1x_2 + 1x_3 + 1x_4 &\leq 2, \\ C_2 : \; 1x_1 + 1x_2 + 0x_3 + 0x_4 &\leq 2, \\ C_3 : \; 0x_1 + 1x_2 + 0x_3 + 2x_4 &\leq 2, \end{aligned}$$

$x_i \in \{0, 1\}$ for all i.

Suppose the partial solution $(x_1, x_2) = (1, 0)$ is being considered and that the current best known solution is $(1, 1, 0, 0)$ which has an objective function value of 6. Using the standard bounding function, the objective function value for the solution $(1, 0, 1, 1)$ is computed and has a value of 7. Using the modified bounding function and the constraint C_1, the objective function value for the solution $(1, 0, 1, 0)$ is computed and has a value of 6. The modified bounding function will prune this branch of the tree, and not look at any more solutions extending the partial solution $x_1 = 1$ and $x_2 = 0$ since the bounding function shows that no solution can have a better result than the current best. The standard bounding function will continue to the next level of the tree and test the partial solution $x_1 = 1$, $x_2 = 0$ and $x_3 = 1$.

Notice that the modified bounding function will never return a worse bound than the standard bounding function.

4.6 Results

The Algorithm described in Section 4.5 was implemented in the programming language *C*, in a program called ipv6.c, which can be found in Appendix B. There were two main reasons for writing a *C* program to solve the Integer Programming Problem associated with finding caps in $PG(n, q)$. The first was to take advantage of the fact that we are looking at a problem where the right-hand-side of the constraints is always 2, and thus implement the modified bounding function in an efficient manner. The second reason was to create a program that would not be constrained by licensing restrictions. That is, we wanted a program that was easily ported to any accessible computer.

The program ipv6.c has the added benefit of being able to consider numerous problem instances for a given $PG(n, q)$. That is, multiple problems can be submitted to the program and the best known solution will be used over all problems, not just the one it is found in. This ability also allows the program to run in parallel. By writing the best known solution found, as soon as it is found, to a file outside of the program, other programs can access the file, and update their best known solution accordingly.

Table 4.1 shows the running times for a set of integer programming problems for

$PG(3,2)$, $PG(3,3)$, $PG(4,2)$, and $PG(5,2)$. The problems were run twice using ipv6.c, once with the standard bound, and once with the modified bound, and again using the software package CPLEX.

Problem Instance	CPLEX Time	ipv6.c - Standard Bound	ipv6.c - Modified Bound	Number of Problem Instances
$PG(3,2)$	0.05s	0.01s	0.03s	13
$PG(3,3)$	4.52s	0.67s	0.71s	76
$PG(4,2)$	0.67s	0.09s	0.10s	26
$PG(5,2)$	9.79s	0.48s	0.46s	56

Table 4.1: CPLEX and ipv6.c Timings (in seconds)

n \ q	3	5	7	11	13
2	4*†	6*†	8*†	12*†	14*†
3	10*†	26*†	50*†	122†	170†
4	20*†	66*	132	316	388
5	56*†	186*	434		
6	112	675	2499		
7	248	1715	6472		

Table 4.2: Large caps in small projective spaces with q prime

Table 4.2 shows the size of largest known caps [HS01]. The method of prescribed automorphisms and ipv6.c were applied in attempt to find large caps in $PG(4,5)$, $PG(4,7)$, $PG(5,5)$, $PG(6,3)$ and $PG(6,5)$, running each for an average of approximately two weeks. The † entries indicate known maximum caps and the $*$ entries indicate instances where we were able to find caps of size equal to the largest known size.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

The method of prescribed automorphisms, discussed in Chapter 2, and applied to the problem of finding small double blocking sets in $PG(2,q)$ (Chapter 3), and large caps in $PG(n,q)$ (Chapter 4), has been used successfully in a number of problems including finding and classifying designs [Ker99], error-correcting codes [Bra05, BW04], and finding (m,r)-arcs in $PG(2,q)$ [BW05]. Note that a cap of size m in $PG(2,q)$ is an $(m,2)$-arc. For more information, a twenty-three page survey of the method and its applications, including the classification of Latin squares and 1-factorizations of complete graphs, can be found in [KÖ06, Chapter 9].

For both problems considered in this thesis, that of double blocking sets in $PG(2,q)$ and caps in $PG(n,q)$, the method of prescribed automorphisms was used in order to reduce their problem sizes and, consequently, significantly reduce the computing time as compared to solving the original problem, in an effort to find new results. In the case of double blocking sets in $PG(2,q)$, the method was used to successfully find and prove a new family of **proper** double blocking sets of size $3q+1$ for the case where q is prime and $q \equiv 3 \pmod 4$ (Theorem 3.11). Previously, no families of size less than or equal to $3q+1$ were known for proper double blocking sets. We were also able to use the method of prescribed automorphisms to prove the existence of a family of proper double blocking sets of size $3q+2$ for any prime $q \geq 7$ (Theorem 3.12). Though this result is slightly weaker, its proof and construction is less complicated than that for Theorem 3.11.

In the case of caps, we were able to apply the method of prescribed automorphisms to match the best known results for several large caps (Section 4.6). The implementation of an integer linear programming solver written by the author allowed us to take advantage of the specific structure of the ILP associated with finding large caps, and allowed us to run the problem on multiple computers without the need of a license as is required by software programs such as CPLEX (Section 4.5).

5.2 Future Works

In the future, we would like to extend the construction of Theorem 3.11 to the case of q prime satisfying $q \equiv 1 \pmod 4$. Though it is expected that the proper double blocking set will be larger than $3q+1$, we hope that the construction itself will be similar. We would also like to extend the results to finding proper t-fold blocking sets on $t+2$ lines for $t > 2$ by applying the method of prescribed automorphisms. For example, using the same strategy for constructing the ILP problem for proper double blocking sets, we can search for 3-fold blocking sets on 5 lines by considering subgroups of the symmetric group S_5.

We would like to continue the study of large caps in $PG(n,q)$ and reduce the amount of redundancy in testing cyclic subgroups by finding a way to recognize whether or not two cyclic subgroups in $PGL(n+1,q)$ are in the same conjugacy class much as we can recognize that two elements of $PGL(n+1,q)$ are in the same conjugacy class by considering their Smith Normal forms over the ring of polynomials over $\mathbb{F}_q$ as mentioned in Section 2.5.

Appendix A

Magma Code

The following is the Magma code and results for Examples 2.24, 2.25 and 2.26.

```
> /*the following is an example of a 0-1 linear programming problem
> this program finds the objective function and the constraint set for
> the problem of finding a maximum set of points in PG(2,q) such that
> every line passes through the set at most 3 times. */
>
> q:=3;
>
> /* Pl := the projective plane
>    P  := the point set - used to retrieve individual points.
>          eg.  the third point is given by P.3
>    L  := the line set - used to retrieve individual lines.
>           eg. the fourth line is given by L.4    */
>
> Pl, P, L := FiniteProjectivePlane(q);
>
> /* CG := PGL(3,3) returned as a permutation group.
>    Pa := the set of points on which G acts.
>    La := the set of lines on which G acts.    */
>
```

```
> CG, Pa, La := CollineationGroup(Pl) ;
>
>
> /* Print out the point set. */
> for i := 1 to #P do
> printf"P_{%o}=%o\n",i,P.i;
> end for;
P_{1}=( 1 : 0 : 0 )
P_{2}=( 0 : 1 : 0 )
P_{3}=( 0 : 0 : 1 )
P_{4}=( 1 : 2 : 0 )
P_{5}=( 0 : 1 : 2 )
P_{6}=( 1 : 2 : 1 )
P_{7}=( 1 : 1 : 1 )
P_{8}=( 1 : 1 : 2 )
P_{9}=( 1 : 0 : 1 )
P_{10}=( 1 : 1 : 0 )
P_{11}=( 0 : 1 : 1 )
P_{12}=( 1 : 2 : 2 )
P_{13}=( 1 : 0 : 2 )
>
> /* Print out the line set. */
> for i := 1 to #P do
> printf"L_{%o}=%o\n",i,L.i;
> end for;
L_{1}=< 1 : 0 : 0 >
L_{2}=< 0 : 1 : 0 >
L_{3}=< 0 : 0 : 1 >
L_{4}=< 1 : 2 : 0 >
L_{5}=< 0 : 1 : 2 >
L_{6}=< 1 : 2 : 1 >
L_{7}=< 1 : 1 : 1 >
L_{8}=< 1 : 1 : 2 >
L_{9}=< 1 : 0 : 1 >
L_{10}=< 1 : 1 : 0 >
L_{11}=< 0 : 1 : 1 >
```

```
L_{12}=< 1 : 2 : 2 >
L_{13}=< 1 : 0 : 2 >
>
> /* Create the incidence matrix between points and lines. */
>  IM:=IncidenceMatrix(Pl);
>
> /* Print out the incidence matrix in a LaTeX friendly format */
> for i := 1 to #P do
>     fst:=true;
>     for j := 1 to #P-1 do
>     if IM[i,j] ne 0 then
>       if not fst then
>         printf "+";
>       end if;
>       fst:=false;
>       printf"x_{%o}",j;
>
>     end if;
>   end for;
>   if IM[i,#P] ne 0 then
>     printf"x_{%o}",#P;
>
>   end if;
>   printf "&\leq 3& \\\\\n";
> end for;
x_{2}+x_{3}+x_{5}+x_{11}&leq 3& \\
x_{1}+x_{3}+x_{9}x_{13}+&leq 3& \\
x_{1}+x_{2}+x_{4}+x_{10}&leq 3& \\
x_{3}+x_{7}+x_{8}+x_{10}&leq 3& \\
x_{1}+x_{7}+x_{11}+x_{12}&leq 3& \\
x_{6}+x_{10}+x_{11}x_{13}&leq 3& \\
x_{4}+x_{5}+x_{7}x_{13}&leq 3& \\
x_{4}+x_{8}+x_{9}+x_{11}&leq 3& \\
x_{2}+x_{8}+x_{12}x_{13}&leq 3& \\
x_{3}+x_{4}+x_{6}+x_{12}&leq 3& \\
x_{1}+x_{5}+x_{6}+x_{8}&leq 3& \\
x_{5}+x_{9}+x_{10}+x_{12}&leq 3& \\
x_{2}+x_{6}+x_{7}+x_{9}&leq 3& \\
>
> /* The following code finds a subgroup H of PGL(3,3) that stabilizes
> both the line [1:0:0] and the point (1:1:1) and finds the
> H-orbits on the set of points and the set of lines. (Example 2.25) */
>
> G := Stabilizer(CG,La,L![1,0,0]);
> H := Stabilizer(G,Pa,P![1,1,1]);
>
> /* Find the H-Orbits of the point set. */
> Orbits(H,Pa);
[
    GSet{  ( 1 : 1 : 1 ) },
    GSet{  ( 0 : 1 : 0 ), ( 0 : 0 : 1 ), ( 0 : 1 : 2 ), ( 0 : 1 : 1 ) },
    GSet{  ( 1 : 0 : 0 ), ( 1 : 2 : 0 ), ( 1 : 2 : 1 ), ( 1 : 1 : 2 ), ( 1 : 0 : 1 ), ( 1 :
    2 : 2 ), ( 1 : 0 : 2 ) }
]
>
> /* Find the H-Orbits of the line set. */
> Orbits(H,La);
[
    GSet{
        < 1 : 0 : 0 >
    },
    GSet{
        < 1 : 2 : 0 >,
        < 0 : 1 : 2 >,
        < 1 : 1 : 1 >,
        < 1 : 0 : 2 >
    },
    GSet{
        < 0 : 1 : 0 >,
        < 0 : 0 : 1 >,
        < 1 : 2 : 1 >,
        < 1 : 1 : 2 >,
        < 1 : 0 : 1 >,
        < 1 : 1 : 0 >,
        < 0 : 1 : 1 >,
        < 1 : 2 : 2 >
    }
]
>
> /* The following code finds a subgroup G of PGL(3,3) that stabilizes
> the set R shown below the G-orbits on the set of points and the set of
> lines. (Example 2.26) */
>
> R:={P![1,0,0],P![0,0,1],P![1,2,0],P![1,2,1],P![1,1,1],P![1,1,0],P![1,0,2]};
> G:=Stabilizer(CG,Pa,R);
>
> /* Find the G-Orbits of the point set. */
> Orbits(G,Pa);
[
    GSet{  ( 1 : 0 : 0 ), ( 1 : 2 : 1 ), ( 1 : 1 : 1 ) },
    GSet{  ( 0 : 0 : 1 ), ( 1 : 2 : 0 ), ( 1 : 1 : 0 ), ( 1 : 0 : 2 ) },
    GSet{  ( 0 : 1 : 0 ), ( 0 : 1 : 2 ), ( 1 : 1 : 2 ), ( 1 : 0 : 1 ), ( 0 : 1 : 1 ), ( 1 :
]
>
> /* Find the G-Orbits of the line set. */
> Orbits(G,La);
[
    GSet{
        < 0 : 1 : 2 >,
        < 0 : 1 : 1 >,
        < 1 : 0 : 2 >
    },
    GSet{
        < 1 : 0 : 0 >,
        < 1 : 1 : 2 >,
        < 1 : 0 : 1 >,
        < 1 : 2 : 2 >
    },
    GSet{
        < 0 : 1 : 0 >,
        < 0 : 0 : 1 >,
        < 1 : 2 : 0 >,
        < 1 : 2 : 1 >,
        < 1 : 1 : 1 >,
        < 1 : 1 : 0 >
    }
]
>
> quit;
```

Appendix B

Integer Programming Code

B.1 The C Code

```
/*This program performs a backtracking algorithm on a 0-1 Integer
Programming Problem specifically designed to find caps in PG(n,q).

Author: Joanna Wallis
Date: July 2006
Version: 6

Command line arguments:
1) filename - the input file (read only)
2) filename - the output file (write only)
3) integer - bound type: 1 indicates the normal bound,
    2 indicates the modified bound
4) filename - a file containing the best known value so far.
    (read/write) */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

```
#define MAX_n 200+1
#define MAX_m 70000+1
#define BOUND2REP 1
#define MAXVAR 500

int backtrack(int l, int curP);
void terminate(int curP);
int bounding(int t, int curP, int l);
void dancinglink(int removedlist[MAX_n], int sizeofrl);
int modbound(int curP, int l);
int standbound(int curP, int l);
int feasibility (int removedlist[MAX_n], int l);
void readinput();
void initialize(void);

/* The following structure contains information about the ith variable.*/
struct xi
{
        int objcoeff;           /*coefficient of x in the objective
                                  function*/
        int conscoeff[MAX_m];   /*a single array containing the
                                  coefficients of x for each constraint*/
        int left;               /*left of x in a doubly linked list*/
        int right;              /*right of x in a doubly linked list*/
        int value;              /*value of x. Must be 0 or 1*/
        int sort;               /*x's location in the sorted list*/
        int loc;                /*x's original location*/
        int flag;               /*a control flag used for the modified
                                  bounding function.  If the flag is set to 0,
                                  then it cannot be considered for the bound*/
};

int N,Q;                 /*PG(N,Q)*/
int m,n;                 /*m is the number of rows in our matrix M,
                           n is the number of columns in our matrix M*/
```

```
int boundtype;            /*The type of bounding function.  0 - none,
                            1 - standard, 2 - modified*/
int nodestested;          /*the number of nodes tested.  Used for
                            efficiency and comparison testing*/
int leafnodestested;      /*the number of leaf nodes tested*/
struct xi x[MAX_n];       /*an array of xi structures.  This, along
                            with slack[MAX_m], is the 0-1 Integer
                            Programming Problem*/
int slack[MAX_m];         /*array to the matrix containing the RHS of
                            the constraints*/
int zeros[MAX_m][MAX_n]; /*zeros[i][j] stores the location of x
                            variables with coefficient 0 in constraint i*/
int ones[MAX_m][MAX_n];  /*ones[i][j] stores the location of x
                            variables with coefficient 1 in constraint i*/
int twos[MAX_m][MAX_n];  /*twos[i][j] stores the location of x
                            variables with coefficient 2 in constraint i*/
int optP;                 /*stores the objective function value of the
                            best solution so far*/
int optX[MAX_n];          /*stores the best solution so far*/
int optinstance;          /*the problem instance of the optimal solution*/
int optsolsize;           /*the number of variables in the optimum
                            solution*/
int curinstance;          /*the problem instance of the current solution*/
int magmaoptP;            /*the optimal solution given by Magma*/
char opfile[100];         /*the string name of the file containing the
                            optimal solution value*/
FILE *fpinput;            /*file pointer to the input file*/
FILE *fpoutput;           /*file pointer to the output file*/
FILE *fpdebug;            /*file pointer to the debug file*/
FILE *fperror;            /*file pointer to the error file*/
FILE *fpoptimal;          /*file pointer to a file containing the optimal
                            solution value*/

int main(int argc,char *argv[])
{
  int i,k;              /*standard loop counters*/
```

```
char str1[1000];      /*used for reading input data*/
int dummy[MAX_n];     /*this is an array that stores the location of x
                       variables that are not live (cannot be 1) in
                       the initial problem*/
clock_t start_clock; /*the start time for each instance*/

if(argc<4)
{
 printf("Error: incorrect line arguments.\n");
 return 0;
}

fpinput=fopen(argv[1],"r");
fpoutput=fopen(argv[2],"w");
strcpy(opfile, argv[4]);
boundtype = atoi(argv[3]);
fprintf(fpoutput,"Input File: %s\t\tBound type:%s\n\n",argv[1],argv[3]);
fprintf(fpoutput,"Problem\t\t#Point\t#Line\tClock\t(Nodes Tested)\n");
fprintf(fpoutput,"Instance\tOrbits\tOrbits\tTime\t\t Total\t
                      Leaf\tOptP\tMoptP\tOptS\n");

/*read "Problem Instance: ?" for some integer ?*/
fgets(str1,1000,fpinput);
k=0;

optinstance=0;
fpoptimal=fopen(opfile,"r");
fscanf(fpoptimal,"%d\n",&optP);
fclose(fpoptimal);
printf("Optval so far is %d\n",optP);

optsolsize=0;

while(!feof(fpinput))
{
 start_clock = clock();
```

```
  /* Initialize the  solution */

  nodestested=0;
  leafnodestested=0;

  k++;
  printf("PROBLEM INSTANCE: %d\t",k);

  curinstance=k;
  readinput();

  printf("n=%d\n",n);
  fflush(stdout);
  if(n<MAX_n && m<MAX_m)
  {
    initialize();
    /*perform the initial feasibility test. x variables that have a
      constraint coefficient greater than 2 are removed from the problem.
      The location of such x variables is stored in the array dummy.*/
    feasibility(dummy,0);

    backtrack(x[0].right,0);
    fprintf(fpoutput,"%d\t\t%d\t%d\t%4f\t  %d\t %d\t%d\t%d\t",k,n,m,
        (clock()-start_clock)/(double)CLOCKS_PER_SEC,nodestested,
        leafnodestested,optP,magmaoptP);
    for(i=1;i<=optsolsize;i++)  fprintf(fpoutput,"%d ",optX[i]);
    fprintf(fpoutput,"\n");
   }/*if condition*/
   fflush(fpoutput);
   /*read "Problem Instance: ?" for some integer ?*/
   fgets(str1,1000,fpinput);
 }/*while*/
   fprintf(fpoutput,"\n\nOptimal Solution Instance:%d\n",optinstance);

 fclose(fpinput);
 fclose(fpoutput);
```

```
  return 0;
}/*main*/

/* This function performs the recursive backtracking algorithm.  It first
tests for the terminal case, which is the case where we are considering
the x_{n+1} variable, a dummy variable.  If not it proceeds by setting
the current x variable to 1, testing feasibility, then testing the bound
and performing the recursion if appropriate. It then sets the current x
variable to 0 and retests the bound, performing the recursion if
appropriate.

  int l the location of the current x variable (node) being tested.
  int curP the value of the solution so far. */

int backtrack(int l, int curP)
{
  int sizeofrl;            /*number of x variables removed for
                             infeasibility.(The effective size of the
                             removed list)*/
  int removedlist[MAX_n]; /*a list of the locations of the x variables
                             removed due to infeasibility*/
  int i;                   /*standard loop counter*/
  int B;                   /*given the current state of the solution, B
                             denotes the best possible solution using the
                             bounding function*/

  i=x[0].right;

  x[l].flag=0;
  nodestested++;

  /* test for terminal case. */
  if(l==n+1)
  {
   terminate(curP);
   leafnodestested++;
  }
  else
  {
   x[l].value=1; /* set x_i=1. */
   /*adjust the slack (RHS) for each constraint*/
   for(i=1;i<=m;i++) slack[i]-=x[l].conscoeff[i];
   /*test for feasibility in the remaining x variables*/
   sizeofrl=feasibility(removedlist,l);

   curP+=x[l].objcoeff;          /*updated the current solution value. */
   /*perform the bounding function T.  If T is 1, modified bounding
      function.  If T is 2, use the standard bounding function.  If T is
      anything else, no bounding function is used. */
   B=bounding(boundtype,curP,l);

   /*if the bounded solution value is better than the current optimal
     value, optP, then continue testing */
   if(B>optP) backtrack(x[l].right, curP);

   /*undo the current node visit. */
   x[l].value=0;
   for(i=1;i<=m;i++) slack[i]+=x[l].conscoeff[i];
   curP-=x[l].objcoeff;
   dancinglink(removedlist,sizeofrl);

   /*given that x_i=0, test the bounding function.  */
   B=bounding(boundtype,curP,l);
   if(B>optP) backtrack(x[l].right, curP);
  }
  x[l].flag=1;
  return 0;
}/*backtrack*/

/*If the termination condition is met during backtracking, this function
is called to test if the current complete solution is better than the
current optimal solution. If it is, then the optimal solution is
```

```
replaced by the current solution.

  int curP the current solution value. */

void terminate(int curP)
{
  int i; /*standard loop counter*/

  if(curP>optP)
  {
   optP=curP;
   fpoptimal=fopen(opfile,"w");
   fprintf(fpoptimal,"%d\n",optP);
   fclose(fpoptimal);

   printf("***********New OptP = %d***********\n", optP);
   for(i=1;i<=n;i++) optX[i]=x[i].value;
   optinstance=curinstance;
   optsolsize=n;
  }
}/*terminate*/

/* The standard bounding function.  This function calculates the
upper bound on the current solution value by assuming all feasible
x variables following the current x variable being tested are 1.

  int curP the current solution value
  int c the location of the current x variable being tested.
  returns the upper bound given by the bounding function.*/

int standbound(int curP,int c)
{
  int i;      /*standard loop counter*/
  int value; /*upper bound given by the bounding function*/

  value=curP;
  i=x[c].right;

  /*the following while loop sums the objective function coefficient
  values for the feasible x variables following the current x variable*/
  while(i<=n)
  {
   value+=x[i].objcoeff;
   i=x[i].right;
  }/*whileloop*/

  return value;
}/*standbound*/

/* The modified bounding function (Problem specific) calculates the
upper bound by considering a single constraint choosing the maximum
possible number of x variables to set 1 based on the current solution
so far, and satisfying that single constraint.
  int curP the current value of the solution
  int l the current x variable (node) being tested.*/

int modbound(int curP, int l)
{
  int i,j,k; /*Standard loop variables*/
  int oneval; /*The sum of the objective function coefficients of the
 chosen (0, 1 or 2) x variables with a constraint
 coefficient value of 1.*/
  int twoval; /*The sum of the objective function coefficients of the
chosen (0, or 1) x variables with a constraint
coefficient value of 2.*/
  int zeroval; /*The sum of the objective function coefficients of
all (untested) x variables with a constraint coefficient
 value of 0.*/
  int minval; /*The minimum value of the upper bound.  This is used if
 testing more than one constraint.  CURRENTLY NOT
  IMPLEMENTED. */
  int curval; /*The upper bound given the current constraint.*/
int counter; /*A standard counter.*/

minval=999999;

for(i=1;i<=BOUND2REP;i++)
{
 oneval=0;
 twoval=0;
 zeroval=0;

 switch(slack[i])
 {
  case 2: j=1;
   k=twos[i][j];
   while(k)
   {
   if(x[k].flag)
   {
    twoval=x[k].objcoeff;
    break;
   }
   j++;
   k=twos[i][j];
  } /*FALL THROUGH*/
  case 1: j=1;
   k=ones[i][j];
   counter=0;
   while(k&&(counter<slack[i]))
   {
    if(x[k].flag)
    {
     oneval+=x[k].objcoeff;
     counter++;
    }
    j++;
    k=ones[i][j];
```

```
     }/*FALL THROUGH*/
    case 0: j=1;
     k=zeros[i][j];
     while(k)
     {
      if(x[k].flag) zeroval+=x[k].objcoeff;
      j++;
      k=zeros[i][j];
     }
     break;
    default: fprintf(stderr,"Illegal coefficient value\n");
   }/*switch*/

   if(twoval>=oneval) curval=twoval+zeroval+curP;
   else curval=oneval+zeroval+curP;
   if(curval<minval) minval=curval;
  }/*for loop*/
  return minval;
}/*modbound*/

int feasibility (int removedlist[MAX_n], int l)
{
  int sizeofrl; /*size of the list of infeasible(or removed)variables*/
  int i,j;      /*standard loop counters*/

  sizeofrl=0;

  /*for each constraint, test that each x variable is feasible.  If it
    is not, that is if the constraint coefficient for an x variable
    is greater than the slack, remove it from the linked list so that it
    is not considered. Note that this uses the method of the dancing
    links.  While the linked list no longer points to the removed x
    variable, the x variable still points to the linked list*/
  for (i=1;i<=m;i++)
  {
   j=x[l].right;
```

```
    while(j!=n+1)
    {
     if(x[j].conscoeff[i]>slack[i])
     {
      sizeofrl++;
      removedlist[sizeofrl]=j;
      x[x[j].left].right=x[j].right;
      x[x[j].right].left=x[j].left;
      x[j].flag=0;
     }
     j=x[j].right;
    }
  }
  return sizeofrl;
}/*feasibility*/



/* The following function directs the program to the appropriate
    bounding function.
    int t        bounding function flag.  2 for the modified function,
                 1 for the standard bounding function.
    int curP     the current solution value.
    int l        the current node being tested.

    returns the upper bound based on the bounding function.*/

int bounding(int t, int curP, int l)
{
  int bound;
  if(t==2) bound=modbound(curP,l);
  if(t==1) bound=standbound(curP,l);
  return bound;
}/*bounding*/

/* The following function performs the dancing link algorithm.  That is,
   it repairs the linked list, returning removed x variables to the list
   using the fact that the x-variables still point to the list even
   though the list does not point to them. */

void dancinglink(int removedlist[MAX_n], int sizeofrl)
{
  int i,k;   /*standard loop counters*/
  for(i=sizeofrl;i>=1;i--)
  {
   k=removedlist[i];
   x[x[k].left].right=k;
   x[x[k].right].left=k;
   x[k].flag=1;
  }/*for loop*/
}/*dancinglink*/

/*the following function reads an input file of appropriate form*/

void readinput()
{
  int i,j;          /*standard loop counters*/
  char str1[1000];  /*used for reading in data*/
  char *str2;       /*used for reading in data*/

  fgets(str1,1000,fpinput); /*reads "n=?" for some integer ?*/
  str2=strtok(str1,"=");
  str2=strtok(NULL,"=");
  N=atoi(str2);
  fgets(str1,1000,fpinput); /*reads "q=?" for some integer ?*/
  str2=strtok(str1,"=");
  str2=strtok(NULL,"=");
  Q=atoi(str2);

  fgets(str1,1000,fpinput); /*reads "The number of orbits on points is "*/
  fgets(str1,1000,fpinput); /*reads n = ?*/
  n=atoi(str1);
  fgets(str1,1000,fpinput); /*reads "The number of orbits on lines is "*/
```

```
 fgets(str1,1000,fpinput); /*reads m = ?*/
 m=atoi(str1);

if (m>MAX_m || n>MAX_n)
{
 printf("ERROR: Parameters are too large\t n=%d\t m=%d\n",n,m);
 fflush(stdout);
 while(strcmp(str1,"END\n")!=0) fgets(str1,1000,fpinput);
 return;
}

fgets(str1,1000,fpinput); /*reads "M is given by the following Matrix"*/
for(i=1;i<=m;i++) /*reads the next m lines which give the matrix M*/
{
 fgets(str1,1000,fpinput);
 str2=strtok(str1," []");
 x[1].conscoeff[i]=atoi(str2);
 for(j=2;j<=n;j++)
 {
  str2=strtok(NULL," []");
  x[j].conscoeff[i]=atoi(str2);
 }/* for loop*/
}/* for loop */

 fgets(str1,1000,fpinput); /*reads "The objective function is given by:"*/
 fgets(str1,1000,fpinput);
 str2=strtok(str1," []");

 x[1].objcoeff=atoi(str2);
 for(i=1;i<n;i++)
 {
  str2=strtok(NULL," []");
  x[i+1].objcoeff=atoi(str2);
 }/*for loop*/

 /*reads "An optimal solution reported by Magma is:"*/
```

```
  fgets(str1,1000,fpinput);
  /*reads optimal solution reported by Magma*/
  fgets(str1,1000,fpinput);
  /*reads "The value of this solution (size of optimal set for tr=1)
  is:"*/
  fgets(str1,1000,fpinput);
  /*reads the value of this solution*/
  fgets(str1,1000,fpinput);
  magmaoptP=atoi(str1);
  while(strcmp(str1,"END\n")!=0) fgets(str1,1000,fpinput);
}/*readinput*/

void initialize(void)
{
  int i,j;                /*standard loop counters*/
  int current;            /*a temporary variable used to sort and create
                          the doubly linked list of x variables*/
  int temp0,temp1,temp2; /*temporary variables used to track the number
                          of x variables in the array for a constraint
                          for zeros, ones and twos respectively.*/

  /*The following loop initializes the slack variables to two.
    NOTE THAT THIS IS MUST BE CHANGED FOR ANY PROBLEM WHERE THE RHS IS
    NOT TWO.*/
  for(i=1;i<=m;i++)        slack[i]=2;

  /*this loop initializes the values for x[i], setting each to 0*/
  for (i=1;i<=n;i++) x[i].value=0;

  /*set and link the tail and the head of the doubly linked list*/
  x[0].left=NULL;
  x[0].right=n+1;
  x[0].objcoeff=9999999;
  x[n+1].right=NULL;
  x[n+1].left=0;
  x[n+1].objcoeff=-1;
```

```
 /*this loop uses an insert sort to create the doubly linked list in
   descending order of the objective function coefficient value*/
 for (i=1;i<=n;i++)
 {
  current=0;
  /*The while loop finds the position to insert the ith x value.*/
  while(x[i].objcoeff<x[current].objcoeff) current=x[current].right;
  x[i].right=current;
  x[i].left=x[current].left;
  x[x[current].left].right = i;
  x[current].left=i;
  current=x[0].right;
 }/* forloop */

 /*this loop stores the initial position of each x[i]*/
 for (i=1;i<=n;i++) x[i].loc=i;

 /*the following for loop initializes the twos, ones and zeros arrays.
   These arrays contain the location of variables(x_i) with the
   appropriate constraint coefficient. For example, twos[3][4]=5 tells
   us that x_5 has coefficient 2 in the third constraint.  Further, it
   is the fourth variable (based on the sorting of the double linked
   list, to have a coefficient of 2 in that constraint. */
 for(i=1;i<=m;i++)
 {
  twos[i][1]=0;
  ones[i][1]=0;
  zeros[i][1]=0;
 } /*for loop*/


 /*the following for loop stores the location of x variables with
   appropriate coefficients in the arrays ones, twos and zeros, for each
constraint. An entry of 0 indicates there are no more coefficients with
this value in the constraint.*/
```

```
  for(j=1;j<=m;j++)
  {
   temp0=0;
   temp1=0;
   temp2=0;
   i=x[0].right;
   while(i!=n+1)
   {
    switch (x[i].conscoeff[j])
    {
     case 0: temp0++;
      zeros[j][temp0]=i;
      zeros[j][temp0+1]=0;
      break;
     case 1: temp1++;
      ones[j][temp1]=i;
      ones[j][temp1+1]=0;
      break;
     case 2: temp2++;
      twos[j][temp2]=i;
      twos[j][temp2+1]=0;
      break;
    }/*switch statement*/
    i=x[i].right;
   }/*while loop*/
  }/*for loop*/
}/* initialize */
```

B.2 Input File

The following is an example input file. Note that while an input file can contain multiple problem instances, each instance must be separated by the string END and a newline character. Also, any line beginning and ending with a * can contain any string with a newline character.

```
*Problem Instance: 1*
```

```
n=3
q=2
*The number of orbits on points is*
3
*The number of orbits on lines is*
5
*M is given by the following Matrix*
[1 2 0]
[1 1 1]
[1 2 0]
[1 2 0]
[3 0 0]
*The objective function is given by:*
[7 7 1]
*Any number of lines may follow until END*
END
```

Bibliography

[Bat97] L. Batten. *Combinatorics of Finite Geometries.* Cambridge University Press, Cambridge, second edition, 1997.

[Bie03] J. Bierbrauer. Large caps. *J. Geom.*, 76(1-2):16–51, 2003. Combinatorics, 2002 (Maratea).

[Bie05] J. Bierbrauer. *Introduction to Coding Theory.* Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2005.

[Bos47] R. C. Bose. Mathematical theory of the symmetrical factorial design. *Sankhyā*, 8:107–166, 1947.

[Bra05] M. Braun. Optimal linear codes from matrix groups. *IEEE Trans. Inform. Theory*, 51(12):4247–4251, 2005.

[BW04] M. Braun, A. Kohnert and A. Wasserman. Construction of linear codes with large minimum distance. *IEEE Trans. Inform. Theory*, 50(8):1687–1691, 2004.

[BW05] M. Braun, A. Kohnert and A. Wasserman. Construction of (n, r)-arcs in $\mathrm{PG}(2, q)$. *Innov. Incidence Geom.*, 1:133–141, 2005.

[EB99] Y. Edel and J. Bierbrauer. 41 is the largest size of a cap in $\mathrm{PG}(4, 4)$. *Des. Codes Cryptogr.*, 16(2):151–160, 1999.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability.* W. H. Freeman and Co., San Francisco, Calif., 1979.

[Hil73] R. Hill. On the largest size of cap in $S_{5,3}$. *Atti Accad. Naz. Lincei Rend. Cl. Sci. Fis. Mat. Natur. (8)*, 54:378–384 (1974), 1973.

[Hir85] J. Hirschfeld. *Finite Projective Spaces of Three Dimensions.* The Clarendon Press Oxford University Press, New York, 1985.

[Hir98] J. Hirschfeld. *Projective Geometries over Finite Fields.* Oxford Mathematical Monographs. The Clarendon Press Oxford University Press, New York, second edition, 1998.

[HS01] J. Hirschfeld and L. Storme. The packing problem in statistics, coding theory and finite projective spaces: update 2001. In *Finite geometries*, volume 3 of *Dev. Math.*, pages 201–246. Kluwer Acad. Publ., Dordrecht, 2001.

[Ker99] A. Kerber. *Applied Finite Group Actions*, volume 19 of *Algorithms and Combinatorics.* Springer-Verlag, Berlin, second edition, 1999.

[Knu00] D. Knuth. Dancing links. December 27 2000. Retrieved from http://www-cs-faculty.stanford.edu/~knuth/papers/dancing-color.ps.gz.

[KÖ06] P. Kaski and P. Östergård. *Classification Algorithms for Codes and Designs*, volume 15 of *Algorithms and Computation in Mathematics.* Springer-Verlag, Berlin, 2006. With 1 DVD-ROM (Windows, Macintosh and UNIX).

[KS99] D. Kreher and D. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search.* CRC Press, Boca Raton, FL, 1999.

[MS77] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes.* Elsevier Science Publishers B.V., Amsterdam, 1977. North-Holland Mathematical Library, Vol. 16.

[Pel70] G. Pellegrino. Sul massimo ordine delle calotte in $S_{4,3}$. *Matematiche (Catania)*, 25:149–157 (1971), 1970.

[Ple98] Vera Pless. *Introduction to the Theory of Error-Correcting Codes.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York, third edition, 1998. A Wiley-Interscience Publication.

[Qvi52] B. Qvist. Some remarks concerning curves of the second degree in a finite plane. *Ann. Acad. Sci. Fennicae. Ser. A. I. Math.-Phys.*, 1952(134):27, 1952.

[Rot02] J. Rotman. *Advanced Modern Algebra.* Prentice Hall Inc., Upper Saddle River, NJ, 2002.

[Seg54] B. Segre. Sulle ovali nei piani lineari finiti. *Atti Accad. Naz. Lincei. Rend. Cl. Sci. Fis. Mat. Nat. (8)*, 17:141–142, 1954.

[Seg55] B. Segre. Ovals in a finite projective plane. *Canad. J. Math.*, 7:414–416, 1955.

[Seg59] B. Segre. Le geometrie di Galois. *Ann. Mat. Pura Appl. (4)*, 48:1–96, 1959.

[Sti95] D. Stinson. *Cryptography: Theory and Practice.* CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL, 1995.

[Tit62] J. Tits. Ovoïdes et groupes de Suzuki. *Arch. Math.*, 13:187–198, 1962.

[VLW01] J. Van Lint and R. Wilson. *A Course in Combinatorics.* Cambridge University Press, Cambridge, second edition, 2001.