Upper bounds on Weight Hierarchies of Extremal Non-Chain Codes

A Study in a Projective Geometry

Thesis for the degree of
Candidatus Scientiarum

Industrial and Applied Mathematics and Informatics
Applicable Algebra and Coding Theory

Hans Georg Schaathun

20th December 1998

Department of Informatics
Universitas Bergensis

Book I: The book of results

Abstract

The weight hierarchy of a linear $[n, k; q]$ code $\mathcal{C}$ over $\mathsf{GF}(q)$ is the sequence $(d_1, d_2, \ldots, d_k)$ where d_r is the smallest support size of an r-dimensional subcode of $\mathcal{C}$. An $[n, k; q]$ code is extremal non-chain if for any r and s, where $1 \leq r < s \leq k$, there are no subspaces R and S, such that $R \subset S$, $\dim R = r$, $\dim S = s$, $w_S(R) = d_r$, and $w_S(S) = d_s$.

This thesis establish upper bounds for weight hierarchies of such codes, of arbitrary dimension. Constructions are provided to prove the bounds tight for five-dimensional codes.

Contents

Preface

This is my thesis for the Cand. Scient. degree. I have been working on it for about 18 months, so now it is time to submit it, even though the work is not quite finished. I have some results, that's true; but there are many more questions. I hope, however, that someone might enjoy the results I present here; and I look forward to returning to the remaining questions at some later time.

I wish to thank the people who have been working or studying at the institute the last couple of years. They have made it a very pleasant place to be. I would specially like to thank my supervisor, prof. Torleiv Kløve, for suggesting such a thrilling problem to work with, and for good help and advice on finding and verifying solutions.

This thesis is submitted for a degree in *Industrial and Applied Informatics and Mathematics*, and so it has got two parts. The first part includes the mathematical results which have been the object of the work. I have done my best to keep the mathematics pure. The other part is a set of computer programs, which have been applied to find and verify some of the mathematical results. The 'industry' I leave for someone else ...

Two more volumes accompany the thesis. Book II contains the source code for the computer programs, and Book III contains data listings from the programs. The data listings may be used to verify the constructions given in the thesis.

georg
Bergen, December 1998

Part I

The Mathematics

Chapter 1

Introduction

In this first chapter, we start by developing notations and terminology suitable for studying weight hierarchies of codes of arbitrary dimension. In Section 1.2 we have a look at previous works on the topic, before we define the purpose of the thesis in Section 1.3.

1.1 Notations and definitions

Throughout this paper $\mathcal{C}$ will denote a $[n, k+1; q]$ code, i.e. a linear code of length n and dimension $k+1$ over the Galois field, $\mathsf{GF}(q)$, with q elements. Codes of dimension $k+1$ will be studied in a projective space, $\mathsf{PG}(k, q)$, of dimension k and order q.

For any code $\mathcal{C}$ we define the support, $\chi(\mathcal{C})$, to be the set of positions where not all codewords of $\mathcal{C}$ are zero, i.e.

$$\chi(\mathcal{C}) \stackrel{\text{def}}{=} \{i \mid \exists (x_1, x_2, \dots x_{k+1}) \in \mathcal{C}, \ \text{ s.t. } x_i \neq 0\}.$$

The support weight of $\mathcal{C}$ is the size of $\chi(\mathcal{C})$, and we denote it $w_S(\mathcal{C})$, i.e.

$$w_S(\mathcal{C}) \stackrel{\text{def}}{=} \#\chi(\mathcal{C}).$$

For $0 \leq r \leq k+1$ *the r-th generalised Hamming weight* of $\mathcal{C}$ is defined by:

$$d_r(\mathcal{C}) \stackrel{\text{def}}{=} \min\{w_S(\mathcal{D}) \mid \mathcal{D} \text{ is an } [n, r; q] \text{ subcode of } \mathcal{C}\}.$$

The sequence $(d_1, d_2, \ldots d_{k+1})$ is called the weight hierarchy of $\mathcal{C}$. We note that the minimum weight of the code is $d = d_1$. For completeness $d_0 = 0$ is the support size of the code consisting only of the zero codeword, $\mathbf{0}$.

We note that by adding a zero-position to $\mathcal{C}$, we get an $[n+1, k+1; q]$ code

$$\mathcal{C}' = \{(\mathbf{c}|0) \mid \mathbf{c} \in \mathcal{C}\},$$

which has the same weight hierarchy as $\mathcal{C}$. Therefore, without loss of generality, we can restrict our study to codes with no zero-position; that is, we assume that $d_{k+1} = n$.

The difference sequence (DS), $(\delta_0, \delta_1, \ldots \delta_k)$, is defined by

$$\delta_j \stackrel{\text{def}}{=} d_{k+1-j} - d_{k-j}, \quad j = 0, 1, \ldots k.$$

We note that the difference sequence is easily computed from the weight hierarchy and vice versa. We say that the difference sequence, $(\delta_0, \delta_1, \ldots \delta_k)$, has dimension $k+1$. Due to the monotonicity of the generalised Hamming weights, the elements of the difference sequence are strictly positive.

We let G denote a generator matrix of $\mathcal{C}$. For any $\mathbf{x} \in \mathsf{GF}(q)^{k+1}$, $\nu(\mathbf{x})$, the *value* (or multiplicity) of $\mathbf{x}$, will be the number of occurrences of $\mathbf{x}$ as a column in G. We then extend the notion of value, so that the value of a set of vectors is the sum of the values of the vectors, i.e. define:

$$\nu(U) \stackrel{\text{def}}{=} \sum_{\mathbf{x} \in U} \nu(\mathbf{x}), \quad \forall\, U \subset \mathsf{GF}(q)^{k+1}.$$

1.1.1 Codes in projective geometry

It is obvious that if $\mathbf{y}$ is a column in G and $\mathbf{x} = \alpha\mathbf{y}$ for some non-zero $\alpha \in \mathsf{GF}(q)$, then we may replace $\mathbf{y}$ by $\mathbf{x}$ without affecting the weight hierarchy. Therefore, we will look at the vectors as projective points.

A *value assignment* is a function from a projective space to the set of natural numbers and zero, i.e.:

$$\nu : \mathsf{PG}(k, q) \longrightarrow \mathbb{N}_0 \stackrel{\text{def}}{=} \{0, 1, \ldots\}.$$

For any $p \in \mathsf{PG}(k, q)$, $\nu(p)$ will be called the *value* of p. The value assignment defines a generator matrix up to equivalence, and consequently it defines a code.

Let $\mathsf{PG}^{(m)}(k,q)$ be the set of *m-spaces* or m-dimensional subspaces of $\mathsf{PG}(k,q)$. Note that $\mathsf{PG}^{(0)}(k,q)$ is the collection of subsets of cardinality 1; both $P \in \mathsf{PG}(k,q)$ and $\{P\} \in \mathsf{PG}^{(0)}(k,q)$ will be called a point. The 1-, 2- and 3-spaces are called lines, planes and solids, respectively. A $(k-1)$-space is called a prime or synonymously a hyperplane. The term hyperplane will in this thesis be used for 3-spaces in $\mathsf{PG}(4,q)$.

In $\mathsf{PG}(4,q)$ we have that $\mathsf{PG}^{(1)}(4,q)$ is the set of lines, $\mathsf{PG}^{(2)}(4,q)$ is the set of planes, and $\mathsf{PG}^{(3)}(4,q)$ is the set of hyperplanes.

If Π_r and Π_s are subspaces of $\mathsf{PG}(k,q)$, we know that their intersection, $\Pi_r \cap \Pi_s$, is also a subspace. The join of Π_r and Π_s, denoted $\Pi_r\Pi_s$, is the intersection of all subspaces containing the union, $\Pi_r \cup \Pi_s$. Obviously $\Pi_r\Pi_s$ is a subspace, and we know that

$$\dim \Pi_r + \dim \Pi_s = \dim \Pi_r\Pi_s + \dim \Pi_r \cap \Pi_s,$$

where $\dim \Pi$ denotes the dimension of Π.

If $p_0, p_1, \dots p_m \in \mathsf{PG}(k,q)$ are projectively independent points, we write $\langle p_0, p_1, \dots p_m\rangle$ for their join. We say that the points, p_i, span the subspace. We reserve the notation for projectively independent points, so $\langle p_0, p_1, \dots p_m\rangle$ has dimension m.

We define the following shorthand notation:

$$Q(n) \stackrel{\text{def}}{=} \frac{q^{n+1}-1}{q-1} = \sum_{i=0}^{n} q^i,$$

and recall that $Q(k)$ is the cardinality of $\mathsf{PG}(k,q)$.

1.1.2 Subcodes and the value assignments

Helleseth et al [HKY92] showed that there is a one-to-one correspondence between subcodes $\mathcal{D}$ of $\mathcal{C}$ of dimension r and subspaces U of $\mathsf{GF}(q)^{k+1}$ of dimension $k+1-r$, such that $d_{k+1} - w_S(\mathcal{D}) = \nu(U)$.

Using our projective representation, we have equivalently a one-to-one correspondence between subcodes, $\mathcal{D}$, of dimension r and subspaces $\Pi_m \in \mathsf{PG}^{(m)}(k,q)$, where $m = k - r$, such that $d_{k+1} - w_S(\mathcal{D}) = \nu(\Pi_m)$.

This means that the existence of a weight hierarchy $(d_1, d_2, \dots, d_{k+1})$ is equivalent to the existence of a value assignment ν, such that:

$$\max\{\nu(\Pi_m) \mid \Pi_m \in \mathsf{PG}^{(m)}(k,q)\} = \sum_{i=0}^{m} \delta_i, \quad m = 0, 1, \dots k. \tag{1.1}$$

For a five-dimensional code, this is in particular:

$$\max\{\nu(p) \mid p \in \mathsf{PG}(k,q)\} = \delta_0 \tag{1.2}$$
$$\max\{\nu(\ell) \mid \ell \in \mathsf{PG}^{(1)}(k,q)\} = \delta_0 + \delta_1 \tag{1.3}$$
$$\max\{\nu(\mathcal{P}) \mid \mathcal{P} \in \mathsf{PG}^{(2)}(k,q)\} = \delta_0 + \delta_1 + \delta_2 \tag{1.4}$$
$$\max\{\nu(\mathcal{H}) \mid \mathcal{H} \in \mathsf{PG}^{(3)}(k,q)\} = \delta_0 + \delta_1 + \delta_2 + \delta_3 \tag{1.5}$$
$$\nu(\mathsf{PG}(4,q)) = \delta_0 + \delta_1 + \delta_2 + \delta_3 + \delta_4. \tag{1.6}$$

Given a value assignment, ν, we will also refer to the sequence $(\delta_0, \delta_1, \ldots \delta_k)$, as defined by equation (1.1) as the difference sequence of ν.

The set of m-spaces of maximum value is denoted by M_m:

$$M_m \stackrel{\text{def}}{=} \left\{ \Pi_m \;\middle|\; \Pi_m \in \mathsf{PG}^{(m)}(k,q) \;\wedge\; \nu(\Pi_m) = \sum_{i=0}^{m} \delta_i \right\}.$$

1.1.3 The Chain Condition

The chain condition was introduced by Wei and Yang [WY93], and it is defined as follows:

Definition 1 (The Chain Condition)
A code $\mathcal{C}$ is said to satisfy the chain condition if

$$\forall i = 0, 1, \ldots k-1, \; \exists \; \alpha_i \in M_i, \quad s.t. \; \alpha_0 \subset \alpha_1 \subset \ldots \subset \alpha_{k-1}.$$

We will refer to codes satisfying this condition as *chained codes.*

We define a number of subconditions, which all are implications of the chain condition. For any i and j such that $1 \leq i \leq j \leq k-1$, we have the condition:

$$(\mathrm{C}i.j) : \exists \alpha \in M_{i-1}, \beta \in M_j, \quad \text{s.t. } \alpha \subset \beta.$$

In particular, for a five-dimensional code, we have:

$$(\mathrm{C1.1}): \quad \exists \wp \in M_0 \wedge \exists \ell \in M_1, \quad \text{s.t. } \wp \subset \ell \tag{1.7}$$
$$(\mathrm{C1.2}): \quad \exists \wp \in M_0 \wedge \exists \mathcal{P} \in M_2, \quad \text{s.t. } \wp \subset \mathcal{P} \tag{1.8}$$
$$(\mathrm{C1.3}): \quad \exists \wp \in M_0 \wedge \exists \mathcal{H} \in M_3, \quad \text{s.t. } \wp \subset \mathcal{H} \tag{1.9}$$
$$(\mathrm{C2.2}): \quad \exists \ell \in M_1 \wedge \exists \mathcal{P} \in M_2, \quad \text{s.t. } \ell \subset \mathcal{P} \tag{1.10}$$
$$(\mathrm{C2.3}): \quad \exists \ell \in M_1 \wedge \exists \mathcal{H} \in M_3, \quad \text{s.t. } \ell \subset \mathcal{H} \tag{1.11}$$
$$(\mathrm{C3.3}): \quad \exists \mathcal{P} \in M_2 \wedge \exists \mathcal{H} \in M_3, \quad \text{s.t. } \mathcal{P} \subset H \tag{1.12}$$

The negations of these conditions, $(\mathrm{N}i.j) \stackrel{\mathrm{def}}{=} \neg(\mathrm{C}i.j)$, will be called the *non-chain conditions*. The chain condition will be denoted by (C0).

Analogous to the definition by Chen and Kløve [CK97a], we will define extremal non-chain codes of arbitrary dimension k, to be codes that satisfy all of the non-chain conditions, $(\mathrm{N}i.j)$. The difference sequence of an extremal non-chain code will be called an ENDS (*extremal non-chain difference sequence*).

1.2 Historical overview

The generalised Hamming weights were introduced as early as 1977 by Helleseth et al [HKM77], though they used the term 'minimum support weights'. Apparently the minimum support weights didn't come to much use until Wei [Wei91] reintroduced the parameters in 1991, and called them 'generalised Hamming weight'. He used the parameters to analyse codes applied on the wire-tap channel of type II, which is a cryptographic application of codes, introduced in 1984 by Ozarow and Wyner [OW84].

During the nineties, several researchers have studied the generalised Hamming weights of linear codes.

In 1993 Kløve [Klø93] studied weight hierarchies of binary codes of dimension 4, and all possible weight hierarchies of such codes were determined. Subsequent articles determine all possible weight hierarchies for three- and four-dimensional codes over several small fields (e.g. [CK96], [CK98b], [CS99]).

Also in 1993, Wei and Yang [WY93] introduced the chain condition. They showed that if a code, $\mathcal{C}$, satisfies the chain condition, then so does its dual, $\mathcal{C}^{\perp}$. Codes satisfying the chain condition have proved useful for several purposes, such as analysis of trellis complexity of linear codes (e.g. [KTFL93]) and analysis of weight hierarchies of product codes (e.g. [WY93]).

Weight hierarchies of chained codes of dimension 5 were studied by Encheva and Kløve [EK94], and Chen and Kløve have studied such codes of arbitrary dimension [CK98c].

Extremal non-chain codes were introduced by Chen and Kløve [CK97a], as the opposite extreme to chained codes. They found tight upper bounds for non-binary, four-dimensional codes. Later they have also found all possible ENDS for four-dimensional, binary codes [CK99].

Chen and Kløve [CK97b], [CK98a] have also divided four-dimensional codes into classes according to which of the conditions, (C0), (C1.1), (C1.2) and (C2.1) (my notation), they satisfy. For non-binary codes, they have given good upper bounds for each class; and for binary codes, they determine all possible difference sequences.

1.3 The problem

The aim of this thesis is to find good upper bounds on the difference sequences of extremal non-chain codes of dimension five and higher. We also want to prove the bounds good, by giving five-dimensional constructions meeting them with equality.

In the process several computer programs are written to aid the study of projective geometries and value assignments.

Chapter 2

Projective Geometry

Before we start examining value assignments, we will introduce some facts from the theory of projective geometries over finite fields.

2.1 Blocking sets

We know that the smallest subset of a projective space, intersecting any line in at least one point, is a prime. More generally, we can state the following lemma, which is proved in [Hir98].

Lemma 1
If $\mathcal{L}$ is a set of points in $\mathsf{PG}(n, q)$ such that, for any r-space Π_r, $\#(\mathcal{L} \cap \Pi_r) \geq Q(s)$, then $\#\mathcal{L} \geq Q(n - r + s)$. Also $\#\mathcal{L} = Q(n - r + s)$ if and only if $\mathcal{L}$ is a $(n - r + s)$-space.

It is also interesting to know the minimum size of a subset, $\mathcal{L}$, meeting any r-space in at least one point, without containing a $(k - r)$-space. We introduce the concept of t-blocking sets.

Definition 2 (t-blocking set)
A subset $\mathcal{L} \subset \mathsf{PG}(k, q)$, is called t-blocking set, *if it meets any t-space in $\mathsf{PG}(k, q)$, but no $(k - t)$-space is contained in $\mathcal{L}$.*

A lot of work is done on 1-blocking sets, which is also named *blocking sets*. Hirschfeld [Hir98] gives a summary for blocking sets in the projective plane, and Beutelspacher [Beu80] has treated t-blocking sets in projective geometries of arbitrary dimension.

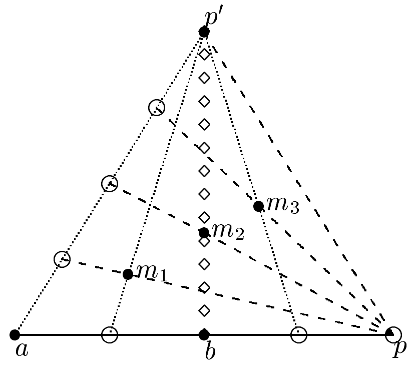

Figure 2.1: $\mathsf{PG}(2,4)$ as in the proof of Lemma 2. Black points are in $\mathcal{L}$, white points are not. $\mathcal{M} = \{p', m_1, m_2, m_3\}$

In the rest of this chapter we introduce a few weak bounds on the size of t-blocking sets in geometries of dimensions 2 and 3. Even though stronger bounds are known ([Beu80], [Hir98]), they are not needed for our purpose, and they would require proofs beyond the scope of this thesis.

2.1.1 The projective plane

Lemma 2
If $\mathcal{L} \subset \mathsf{PG}(2,q)$, such that for any line ℓ, $\emptyset \neq \mathcal{L} \cap \ell$, then there is a line ℓ', such that $\#(\mathcal{L} \cap \ell') \geq 3$.

Proof: By Lemma 1, there are more than two points in $\mathcal{L}$, so let ℓ be a line spanned by two points in $\mathcal{L}$. Obviously $\#(\mathcal{L} \cap \ell) \geq 2$.

Assume for a contradiction that no line has more than two points in $\mathcal{L}$. Then ℓ has exactly two points, a and b, in $\mathcal{L}$. Let $p \in \ell \backslash \mathcal{L}$ be another point on the line. There are q more lines through p, so let $\mathcal{M} \subset \mathcal{L}$ be a set including exactly one point on each of these lines, and fix one point $p' \in \mathcal{M}$. Since no line has more than two points in $\mathcal{L}$, there are $q-1$ distinct lines spanned by

p' and one more point in $\mathcal{M}$, intersecting ℓ in $q-1$ distinct points, different from p. Thus at least one line is incident with a or b and thus has at least three points in $\mathcal{L}$, proving the lemma by contradiction. □

Corollary 1
For any $\mathcal{L} \subset \mathsf{PG}(2,2)$ such that any line intersects $\mathcal{L}$ in at least one point, there is a line $\ell \subset \mathcal{L}$.

Lemma 3
If $\mathcal{L}$ is a blocking set in $\mathsf{PG}(2,q)$, then $\#\mathcal{L} \geq q+3$.

Proof: Lemma 2 says that there is a line, ℓ, such that $\#(\ell \cap \mathcal{L}) \geq 3$. Since $\ell \not\subset \mathcal{L}$, there is a point $p \in \ell \backslash \mathcal{L}$. There are q lines through p beside ℓ, intersecting $\mathcal{L}$ in q distinct points. Thus $\#(\mathcal{L} \cap \ell) \geq 3$ and $\#(\mathcal{L} \backslash \ell) \geq q$, and we have $\#\mathcal{L} \geq q+3$, as required. □

Hirschfeld [Hir98] states that the bound is really

$$\#\mathcal{L} \geq q + \sqrt{q} + 1,$$

which is tight for q square. We note that for $q \in \{3,4\}$, this this gives the same bound as our lemma, if we replace the square root with its ceiling:

$$q=3: \quad \#\mathcal{L} \geq 3+2+1=6 \tag{2.1}$$
$$q=4: \quad \#\mathcal{L} \geq 4+2+1=7. \tag{2.2}$$

To prove that the bound is tight for $q=3$, take two distinct lines, ℓ_1 and ℓ_2, and let $p_1 \in \ell_1 \backslash \ell_2$ and $p_2 \in \ell_2 \backslash \ell_1$ be two points. Let c be a point on $\langle p_1, p_2 \rangle$, different from p_1 and p_2. The set $\mathcal{L} \stackrel{\text{def}}{=} (\{c\} \cup \ell_1 \cup \ell_2) \backslash \{p_1, p_2\}$ is a blocking set of six points, meeting the bound with equality.

Table 2.1 gives a representation of $\mathsf{PG}(2,4)$. It is easily verified that

$$\mathcal{L} \stackrel{\text{def}}{=} \{A, B, C, F, G, J, K\},$$

satisfies the properties of a blocking set, and the cardinality is indeed 7.

2.1.2 The projective solid

Lemma 4
If $\mathcal{L} \subset \mathsf{PG}(3,q)$ so that any line, $\ell \in \mathsf{PG}^{(1)}(3,q)$, intersects $\mathcal{L}$ in at least two points, then $\#\mathcal{L} \geq 2q^2+2q+2$.

Lines in $\mathsf{PG}(2,4)$		
$\ell_0 = \{A, B, C, D, E\}$	$\ell_1 = \{A, F, G, H, I\}$	$\ell_2 = \{A, J, K, L, M\}$
$\ell_3 = \{A, N, O, P, Q\}$	$\ell_4 = \{A, R, S, T, U\}$	$\ell_5 = \{B, F, J, N, R\}$
$\ell_6 = \{B, G, K, O, S\}$	$\ell_7 = \{B, H, L, P, T\}$	$\ell_8 = \{B, I, M, Q, U\}$
$\ell_9 = \{C, F, K, P, U\}$	$\ell_{10} = \{C, G, J, Q, T\}$	$\ell_{11} = \{C, H, M, N, S\}$
$\ell_1 1 = \{C, I, L, O, R\}$	$\ell_{12} = \{D, F, L, Q, S\}$	$\ell_{14} = \{D, G, M, P, R\}$
$\ell_{15} = \{D, H, J, O, U\}$	$\ell_{16} = \{D, I, K, N, T\}$	$\ell_{17} = \{E, F, M, O, T\}$
$\ell_{18} = \{E, G, L, N, U\}$	$\ell_{19} = \{E, H, K, Q, R\}$	$\ell_{20} = \{E, I, J, P, S\}$

Table 2.1: A representation of $\mathsf{PG}(2,4)$.

Proof: If $\mathcal{L} = \mathsf{PG}(3,q)$, then $\#\mathcal{L} = q^3 + q^2 + q + 1 > 2q^2 + 2q + 2$.

If $\mathcal{L} \neq \mathsf{PG}(3,q)$ take $p \notin \mathcal{L}$. There are $q^2 + q + 1$ lines through p, each of which intersects $\mathcal{L}$ in at least two points, for a total of $\#\mathcal{L} \geq 2q^2 + 2q + 2$ points, as required. □

Lemma 5
If $\mathcal{L} \subset \mathsf{PG}(3,q)$ is a 2-blocking set, then $\#\mathcal{L} \geq q^2 + 3q$.

Before we start with the proof, recall that a t-arc is a set of t points in a plane, no three of which are collinear.

Proof: Suppose $\mathcal{L}$ is a 2-blocking set in $\mathsf{PG}(3,q)$, and $\#\mathcal{L} < q^2 + 3q$.

By Lemma 4, if all lines have at least two points in $\mathcal{L}$, then $\#\mathcal{L} \geq 2q^2 + 2q+2 \geq q^2+3q$, so there must be a line, ℓ, with only one point, p, in common with $\mathcal{L}$.

There are $q+1$ distinct planes $\Phi_i \supset \ell,\ i = 0, 1, \ldots q$. By Lemma 3, each plane, Φ_i, has either a line, ϕ_i, or at least $q+3$ points in $\mathcal{L}$. When ϕ_i exists, it lies in the same plane as ℓ, and so it intersects it. The point of intersection is p, as the only point in both ℓ and $\mathcal{L}$. This way each plane contributes with at least q points beside p, and if ϕ_i does not exist, Φ_i contributes with two points extra. Let r be the number of indices, $0 \leq i \leq q$, for which ϕ_i does not exist. Then

$$q^2 + 3q - 1 \geq \#\mathcal{L} \geq 1 + (q+1)q + r \cdot 2 = q^2 + q + 1 + 2r,$$

so $r \leq q-1$. Thus at least two of the planes, Φ_i, contains a line, ϕ_i, through p in $\mathcal{L}$. Let $\alpha \neq \beta$ be two of these lines. The plane spanned by α and β will be denoted by Δ. Obviously

$$\#(\mathcal{L} \cap \Delta) \geq 2q + 1. \tag{2.3}$$

For any plane, $\mathcal{P}$, there is a point $c \in \mathcal{P} \backslash \mathcal{L}$. There are q^2 lines containing c and not contained in $\mathcal{P}$, and consequently there are at least q^2 points in $\mathcal{L} \backslash \mathcal{P}$. In particular we have

$$\#(\mathcal{L} \backslash \Delta) \geq q^2. \tag{2.4}$$

Assume that any line in Δ has at least two points in $\mathcal{L}$. Then in particular any line through p must have at least one more point beside p in common with $\mathcal{L}$. There are $q-1$ such lines beside α and β. So $\#(\Delta \cap \mathcal{L}) \geq 2q+1+q-1 = 3q$. Thus $\#\mathcal{L} \geq q^2 + 3q$, contrary to assumption.

We can then conclude that there is a line, $\gamma \subset \Delta$, meeting $\mathcal{L}$ in exactly one point. All the lines γ, α and β are in Δ, so they must intersect. Both α and β are entirely in $\mathcal{L}$, so γ must intersect both of them in the very same point; that is $p \in \gamma$. There are q planes $\Gamma_i, i = 1, 2, \ldots q$, such that $\gamma \subset \Gamma_i \neq \Delta$. Now we will show that at least two of the planes Γ_i have a line each in $\mathcal{L}$, and we use the same kind of argument as we used for the planes Φ_i.

By Lemma 3, each plane, Γ_i, has either a line, γ_i, or at least $q+3$ points in $\mathcal{L}$. i.e. $\#((\Gamma_i \backslash \Delta) \cap \mathcal{L}) \geq q$ for all i. Let s be the number of indices, $1 \leq i \leq q$, for which γ_i does not exist. Then $\#(\mathcal{L} \backslash \Delta) \geq q^2 + 2s$. We use equation (2.3) to get

$$q^2 + 3q - 1 \geq \#\mathcal{L} \geq q^2 + 2s + \#(\mathcal{L} \cap \Delta) \geq q^2 + 2q + 1 + 2s.$$

We conclude that $s \leq q/2 - 1$, so at least two of the planes Γ_i have lines γ_i in $\mathcal{L}$. Let $\eta \neq \zeta$ be two of these lines. Both the lines η and ζ intersect γ in a point in $\mathcal{L}$, that is $\eta \cap \zeta = \{p\}$. This gives us a total of four lines through p, in $\mathcal{L}$: α, β, η and ζ.

Let a be a point on α. There are q distinct lines in Δ, incident with a but not p. Because they intersect α and β in distinct points, they have at least two points in $\mathcal{L}$. If all these lines have three points in $\mathcal{L}$, we have a total of $3q + 1$ points in $\Delta \cap \mathcal{L}$, which gives $\#\mathcal{L} \geq q^2 + 3q + 1$, contrary to assumption. Hence there is a line $\psi \ni a$, with exactly two points in $\mathcal{L}$.

Now consider q distinct planes, $\Psi_i,\ i = 1, 2, \ldots q$, such that $\psi \subset \Psi_i \neq \Delta$. The four lines, α, β, η and ζ, intersect Ψ_i in four distinct points. If three of these points are collinear, three of the lines are contained in the same plane Θ. Then there are at least $3q + 1$ points in $\Theta \cap \mathcal{L}$ and q^2 points in $\Theta \backslash \mathcal{L}$, for a total of $q^2 + 3q + 1$ in $\mathcal{L}$, contrary to assumption.

We conclude that each plane, Ψ_i, has a four-arc in $\mathcal{L}$. So it has at least $q+3$ points in $\mathcal{L}$, for even if it does have a line in $\mathcal{L}$, the four-arc guarantees two more points that are not on the line. The intersection of Ψ_i and Δ has exactly two points, a and b, in $\mathcal{L}$. This gives

$$\begin{aligned} \#(\mathcal{L} \cap (\Psi_i \backslash \Delta)) &\geq q+1, \quad 1 \leq i \leq q \\ \#(\mathcal{L} \cap \Delta) &\geq 2q+1; \end{aligned}$$

thus

$$\#\mathcal{L} \geq q(q+1) + 2q + 1 = q^2 + 3q + 1,$$

proving the lemma by contradiction. □

Lemma 6
If $\mathcal{L} \subset \mathsf{PG}(3,q)$ is a 1-blocking set, then $\#\mathcal{L} \geq q+3$.

Proof: Suppose that $\mathcal{L}$ is a blocking set with no more than $q+2$ points.

By Lemma 1, there are at least 3 points in $\mathcal{L}$. Any set of three points is contained in a plane, so obviously there is a plane, $\mathcal{P}$, meeting $\mathcal{L}$ in at least three points. There is a line, $\ell \subset \mathcal{P}$, which does not meet $\mathcal{L}$, because $q+2$ points cannot make a 1-blocking set in $\mathcal{P}$. There are q planes different from $\mathcal{P}$, through ℓ, and each of these planes has a point in $\mathcal{L}$. This gives at least three points in $\mathcal{P} \cap \mathcal{L}$ and q points in $\mathcal{L} \backslash \mathcal{P}$, for a total of $q+3$ points, proving the lemma by contradiction. □

Beutelspacher [Beu80] states that if $\mathcal{L}$ is t-blocking set in $\mathsf{PG}(t+1,q)$, then

$$\#\mathcal{L} \geq Q(t) + q^{t-1}\sqrt{q}.$$

For $q = 2$, the bound from Lemma 5 is tight:

Lemma 7
There is a 2-blocking set $\mathcal{L} \subset \mathsf{PG}(3,2)$, such that $\#\mathcal{L} = 10$.

Proof: Take two distinct planes $\mathcal{A}, \mathcal{B}$, and two points $a \in \mathcal{A} \backslash \mathcal{B}, b \in \mathcal{B} \backslash \mathcal{A}$. Let $c \in \langle a, b \rangle \backslash \{a, b\}$. Let $\mathcal{L} \stackrel{\text{def}}{=} (\mathcal{A} \cup \mathcal{B} \cup \{c\}) \backslash \{a, b\}$.

We have $\#(\mathcal{A} \backslash \{a\}) = \#(\mathcal{B} \backslash \{b\}) = 6$ and $\#(\mathcal{A} \cap \mathcal{B}) = 3$, so $\#\mathcal{L} = 2 \cdot 6 - 3 + 1 = 10$ as required.

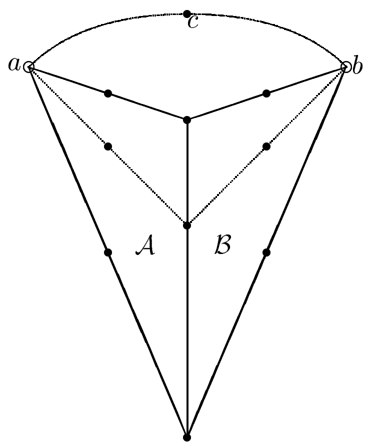

Figure 2.2: The construction for Lemma 7. Black points are in $\mathcal{L}$, white points are not.

Neither $\mathcal{A}$ nor $\mathcal{B}$ is in $\mathcal{L}$, and any other plane intersects $\mathcal{A} \cup \mathcal{B}$ in at most five points. Including c it can at most intersect $\mathcal{L}$ in six out of its seven points, so it is not entirely included in $\mathcal{L}$.

Any line in $\mathsf{PG}(3, 2)$ intersects $\mathcal{A}$ in at least one point, which is either a or in $\mathcal{L}$. Any line through a also intersects $\mathcal{B}$ in at least one point, which is either b or in $\mathcal{L}$. The line $\langle a, b \rangle$ intersects $\mathcal{L}$ in c. Hence $\mathcal{L}$ is a 2-blocking set of cardinality 10. $\square$

Chapter 3

Upper bounds on ENDS

Chen and Kløve [CK97a] give upper bounds for the difference sequence of extremal non-chain codes of dimension 4, and they show that the bounds are tight for non-binary codes. In this chapter, we will generalise their bounds to arbitrary dimension. The next chapter then provides a construction proving the bounds tight for five-dimensional, non-binary codes.

We start by proving upper bounds on the subspace values, i.e. on δ_1 through δ_{k-1}. In the second section we will prove certain properties for value assignments meeting the bounds with equality. We round off the chapter with upper bounds on the total value, $\nu(\mathsf{PG}(k,q))$.

3.1 Bounds on subspace values

We start by proving two lemmas, generalising Theorem 2 from [CK97a]. We will see that they make it possible to find a general statement on upper bounds for all differences except the last, δ_k, in Theorem 1.

Lemma 8
Given $(\delta_0, \delta_1, \ldots \delta_k)$ *satisfying* $(\mathrm{N}l.m)$, *where* $1 \leq l \leq m \leq k-1$. *If* $\Pi_m \in M_m$, *and*

$$\lambda \stackrel{\text{def}}{=} \max\{\nu(p) | p \in \Pi_m\},$$

then

$$\sum_{i=l}^{m} \delta_i \leq \lambda \sum_{i=l}^{m} q^i - 1.$$

Proof: Let Π_{l-1} be a $(l-1)$-space in Π_m. By condition (N$l.m$), we have

$$\nu(\Pi_{l-1}) \leq \sum_{i=0}^{l-1} \delta_i - 1,$$

and hence

$$\sum_{i=0}^{m} \delta_i = \nu(\Pi_m) = \nu(\Pi_{l-1}) + \nu(\Pi_m \backslash \Pi_{l-1}) \leq \sum_{i=0}^{l-1} \delta_i - 1 + \lambda \sum_{i=l}^{m} q^i.$$

□

For $l = m = 1$, we get the following corollary, by noting that $\lambda \leq \delta_0 - 1$ and that $\delta_1 \geq 1$.

Corollary 2
If $(\delta_0, \delta_1, \ldots \delta_k)$*,* $k \geq 2$*, satisfies* (N1.1)*, then* $\delta_1 \leq q\delta_0 - (q+1)$ *and* $\delta_0 \geq 2$*.*

Lemma 9
Given $(\delta_0, \delta_1, \ldots \delta_k)$ *satisfying* (Nl+2.m)*, where* $0 \leq l \leq m-2$*, and* $m \leq k-1$*. If* $\Pi_m \in M_m$ *and*

$$\lambda_l \stackrel{\text{def}}{=} \max\{\nu(\Pi_l) \mid \Pi_l \subset \Pi_m, \Pi_l \in \mathsf{PG}^{(l)}(k,q)\},$$

then

$$\delta_m \leq \left(\sum_{i=1}^{m-(l+1)} q^i \right) \cdot \left(\sum_{i=0}^{l+1} \delta_i - 1 - \lambda_l \right) - \sum_{i=l+2}^{m-1} \delta_i - 1.$$

Proof: Let Π_l be a l-space in Π_m, such that $\lambda_l = \nu(\Pi_l)$. For any $(l+1)$-space, Π_{l+1}, such that $\Pi_l \subset \Pi_{l+1} \subset \Pi_m$, we have by condition (N$l+2.m$) that

$$\nu(\Pi_{l+1} \backslash \Pi_l) \leq \sum_{i=0}^{l+1} \delta_i - 1 - \lambda_l.$$

Let S be the set of $(l+1)$-spaces containing Π_l in Π_m, i.e.

$$S \stackrel{\text{def}}{=} \{\Pi_{l+1} \in \mathsf{PG}^{(l+1)}(k,q) \mid \Pi_l \subset \Pi_{l+1} \subset \Pi_m\}.$$

There are $Q(m-(l+1))$ such subspaces. Thus we have the following expression for the value of Π_m:

$$\sum_{i=0}^{m} \delta_i = \nu(\Pi_l) + \sum_{\Pi_{l+1} \in S} \nu(\Pi_{l+1} \backslash \Pi_l) \leq \lambda_l + \left(\sum_{i=0}^{m-(l+1)} q^i \right) \cdot \left(\sum_{i=0}^{l+1} \delta_i - 1 - \lambda_l \right),$$

which gives the required expression:

$$\delta_m \leq \left(\sum_{i=1}^{m-(l+1)} q^i \right) \cdot \left(\sum_{i=0}^{l+1} \delta_i - 1 - \lambda_l \right) - \sum_{i=l+2}^{m-1} \delta_i - 1.$$

□

Corollary 3
If $(\delta_0, \delta_1, \ldots \delta_k)$ satisfies $(\mathrm{N}m.m)$, where $2 \leq m \leq k-1$, then

$$\delta_m \leq q \sum_{i=0}^{m-1} \delta_i - q\lambda - (q+1),$$

where

$$\lambda \stackrel{\text{def}}{=} \max\{\nu(\Pi_{m-2}) \mid \Pi_{m-2} \subset \Pi_m \in M_m,\ \Pi_{m-2} \in \mathsf{PG}^{(m-2)}(k,q)\}.$$

We now have the required tools to prove our first main result, which is a generalisation of Corollary 2 from [CK97a].

Theorem 1
If $(\delta_0, \delta_1, \ldots \delta_k)$ is an ENDS, $1 \leq m \leq k-1$, and

$$\delta_j = q^j \delta_0 - \sum_{i=0}^{j} q^i, \quad \forall\ j, \ \ s.t.\ 2 \leq j \leq m-1, \tag{3.1}$$

then

$$\delta_m \leq q^m \delta_0 - \sum_{i=0}^{m} q^i. \tag{3.2}$$

This gives unconditional bounds for δ_1 and δ_2. We have no bound on δ_i for $i \geq 3$, unless the bounds on δ_j, $2 \leq j \leq i-1$, are satisfied with equality.
Proof: The case with $m = 1$ is covered by Corollary 2. Assume for induction that it holds for any $m < \bar{m}$.

(The inductive hypothesis is not strictly necessary. The proof of the bound on δ_1, and the conditions for the values of $\delta_2, \ldots \delta_{\bar{m}-1}$ from equation (3.1) are be sufficient, but we use the inductive hypothesis to emphasise the point where condition (3.1) is necessary.)

Fix $\Pi_{\bar{m}} \in M_{\bar{m}}$, and define for $n = 0, 1, \ldots, \bar{m} - 2$,

$$\lambda_n \stackrel{\text{def}}{=} \max\{\nu(\Pi_n) \mid \Pi_n \subset \Pi_{\bar{m}}, \Pi_n \in \mathsf{PG}^{(n)}(k, q)\}.$$

Because the code is an ENDS, we have that

$$\lambda_n \leq \sum_{i=0}^{n} \delta_i - 1, \quad 0 \leq n \leq \bar{m} - 2. \tag{3.3}$$

If

$$\lambda_{\bar{m}-2} = \sum_{i=0}^{\bar{m}-2} \delta_i - 1,$$

the bound follows from Corollary 3 and the inductive hypothesis:

$$\delta_{\bar{m}} \leq q\delta_{\bar{m}-1} - 1 \leq q^{\bar{m}}\delta_0 - \sum_{i=0}^{\bar{m}} q^i.$$

In this case the bound may be satisfied with equality.

If $\lambda_0 \leq \delta_0 - 2$, then by Lemma 8, we have

$$\delta_{\bar{m}} \leq q^{\bar{m}}\delta_0 - 2q^{\bar{m}} - 1 \leq q^{\bar{m}}\delta_0 - q^{\bar{m}} - \frac{q^{\bar{m}} - 1}{q - 1} - 1 - 1 = q^{\bar{m}}\delta_0 - Q(\bar{m}) - 2,$$

and the bound holds with inequality. We note that the possibilities when $\bar{m} = 2$ are exhausted. For $\bar{m} \geq 3$, however, we have to consider the possibility that

$$\lambda_0 = \delta_0 - 1 \quad \bigwedge \quad \lambda_{\bar{m}-2} \leq \sum_{i=0}^{\bar{m}-2} \delta_i - 2.$$

This gives equality in equation (3.3) for $n = 0$, and inequality for $n = \bar{m} - 2$. Let l be the highest value of n for which equality holds, thus $0 \leq l \leq \bar{m} - 3$ and

$$\lambda_l = \sum_{i=0}^{l} \delta_i - 1 \quad \bigwedge \quad \lambda_{l+1} \leq \sum_{i=0}^{l+1} \delta_i - 2.$$

Let $\Pi_l \subset \Pi_{\bar{m}}$ be an l-space, such that $\nu(\Pi_l) = \lambda_l$. Let S be the set of $(l+1)$-spaces, Π_{l+1}, such that $\Pi_l \subset \Pi_{l+1} \subset \Pi_m$. There are $Q(\bar{m} - (l+1))$ such subspaces. Hence we have an expression for the value of $\Pi_{\bar{m}}$:

$$\begin{aligned}
\sum_{i=0}^{\bar{m}} \delta_i &= \sum_{\Pi_{l+1} \in S} \nu(\Pi_{l+1} \backslash \Pi_l) + \lambda_l \\
&\leq (\lambda_{l+1} - \lambda_l) Q(\bar{m} - (l+1)) + \lambda_l \\
&\leq (\delta_{l+1} - 1) \sum_{i=0}^{\bar{m}-(l+1)} q^i + \lambda_l.
\end{aligned}$$

We rearrange terms and simplify, using the inductive hypothesis, to get:

$$\begin{aligned}
\delta_{\bar{m}} \leq& \left(\sum_{i=0}^{\bar{m}-(l+1)} q^i \right) \cdot \left(\delta_{l+1} - 1 \right) - \sum_{i=0}^{\bar{m}-1} \delta_i + \sum_{i=0}^{l} \delta_i - 1 \\
\leq& \left(\sum_{i=1}^{\bar{m}-(l+1)} q^i \right) \cdot \left(q^{l+1} \delta_0 - Q(l+1) \right) - Q(\bar{m} - (l+1)) - \sum_{i=l+2}^{\bar{m}-1} \delta_i - 1.
\end{aligned}$$

Now we insert the assumed expressions for δ_i from equation (3.1) in the subtracted sum and rearrange the terms.

$$\begin{aligned}
\delta_{\bar{m}} \leq& \sum_{i=l+2}^{\bar{m}-1} Q(i) - \sum_{i=l+2}^{\bar{m}-1} q^i \delta_0 - 1 + \sum_{i=l+2}^{\bar{m}} q^i \delta_0 - \\
& \sum_{i=1}^{\bar{m}-(l+1)} q^i Q(l+1) - Q(\bar{m} - (l+1)) \\
=& \left\{ q^{\bar{m}} \delta_0 - q^{\bar{m}-l} Q(l) - Q(\bar{m} - (l+1)) \right\} + \\
& \left\{ q^{\bar{m}-l} Q(l) + \sum_{i=l+2}^{\bar{m}-1} Q(i) - \sum_{i=1}^{\bar{m}-(l+1)} q^i Q(l+1) - 1 \right\}
\end{aligned}$$

We note that the expression in the first pair of braces is the bound we want to prove, so we have to show that the second expression is negative.

$$
\begin{aligned}
& q^{\bar{m}-l}Q(l) + \sum_{i=l+2}^{\bar{m}-1} Q(i) - \sum_{i=1}^{\bar{m}-(l+1)} q^i Q(l+1) - 1 \\
= & \sum_{j=1}^{\bar{m}-(l+2)} \left[\sum_{i=0}^{l+1+j} q^i - \sum_{i=j}^{l+1} q^i \right] + q^{\bar{m}-l}Q(l) - q^{\bar{m}-(l+1)}Q(l+1) - 1 \\
= & \sum_{j=1}^{\bar{m}-(l+2)} \sum_{i=0}^{j-1} q^i - q^{\bar{m}-l-1} = \sum_{j=0}^{\bar{m}-(l+3)} \sum_{i=0}^{j} q^i - q^{\bar{m}-l-1} - 1 \\
= & \sum_{i=0}^{\bar{m}-(l+3)} (\bar{m} - (l+3) - i + 1) \cdot q^i - q^{\bar{m}-l-1} - 1 \\
= & -\theta(\hat{m}, q),
\end{aligned}
$$

where

$$
\theta(\hat{m}, q) \stackrel{\text{def}}{=} q^{\hat{m}} - \sum_{i=0}^{\hat{m}-2} (\hat{m} - 1 - i) \cdot q^i + 1,
$$

and

$$
\hat{m} \stackrel{\text{def}}{=} \bar{m} - l - 1.
$$

It remains to prove that $\theta(\hat{m}, q)$ is positive; in fact we will prove that $\theta(\hat{m}, q) \geq \hat{m} + q$, and we do this by induction on $\hat{m}$. First we note that $\theta(1, q) = q + 1$, then assume that $\theta(\hat{m}, q) \geq \hat{m} + q$, and we observe that:

$$
\begin{aligned}
\theta(\hat{m}+1, q) &= q^{\hat{m}+1} - \sum_{i=0}^{\hat{m}-1} (\hat{m} - i) \cdot q^i + 1 \\
&= q \left[q^{\hat{m}} - \sum_{i=1}^{\hat{m}-1} (\hat{m} - i) \cdot q^{i-1} + 1 \right] - q - \hat{m} + 1 \\
&= q \left[q^{\hat{m}} - \sum_{i=0}^{\hat{m}-2} (\hat{m} - 1 - i) \cdot q^i + 1 \right] - (\hat{m} + q) + 1 \\
&= q\theta(\hat{m}, q) - (\hat{m} + q) + 1 \\
&\geq (q-1)(\hat{m} + q) + 1 \geq \hat{m} + q + 1.
\end{aligned}
$$

Hence $\theta(\hat{m}, q)$ is positive and consequently the bound of the theorem holds with inequality. $\square$

3.2 Construction requirements

Ultimately we want to construct codes meeting the bounds with equality. In this section we prove some properties of such codes.

Lemma 10
If $(\delta_0, \delta_1, \ldots \delta_k)$, $k \geq 2$, is an ENDS, meeting the bounds on δ_1 from Theorem 1 with equality, and $\ell \in M_1$ is a line of maximum value, then

$$\nu(p) = \delta_0 - 1, \quad \forall p \in \ell.$$

Proof: If $\delta_1 = q\delta_0 - (q+1)$, then $\nu(\ell) = (q+1)(\delta_0 - 1)$. There are $q+1$ points on ℓ, none of which may have value greater than $\delta_0 - 1$. Hence all points on ℓ must have value exactly $\delta_0 - 1$. $\square$

Lemma 11
Let $(\delta_0, \delta_1, \ldots \delta_k)$, $k \geq 3$, be an ENDS, and let $\mathcal{P} \in M_2$ be a plane of maximum value. If the bound on δ_1 and δ_2 are tight, then there is $\ell \subset \mathcal{P}$, such that

$$\begin{aligned} \nu(p) &= \delta_0 - 2, \quad \forall p \in \ell, \\ \nu(p) &= \delta_0 - 1, \quad \forall p \in \mathcal{P} \backslash \ell. \end{aligned}$$

Proof: No point in $\mathcal{P}$ can have higher value than $\delta_0 - 1$, or it would be in M_0, so let $\delta_0 - 1$ be the default value for points in $\mathcal{P}$. Let $\mathcal{L}$ be the set of all points with value $\delta_0 - 2$ or less. The total value of $\mathcal{P}$ is

$$\begin{aligned} \delta_0 + \delta_1 + \delta_2 &= \delta_0 + q\delta_0 - (q+1) + q^2\delta_0 - (q^2 + q + 1) \\ &= q^2(\delta_0 - 1) + (q+1)(\delta_0 - 2), \end{aligned}$$

so $\#\mathcal{L} \leq q + 1$.

Any line, α, in $\mathcal{P}$ must meet $\mathcal{L}$ in at least one point, or otherwise $\alpha \in M_1$. Hence by Lemma 1, $\mathcal{L} = \ell$ is a line and thus there are exactly $q+1$ points in $\mathcal{L}$, each of which has value exactly $\delta_0 - 2$ as required. $\square$

Lemma 12
Let $(\delta_0, \delta_1, \dots \delta_k)$ be an ENDS for $q \geq 3$ and $k \geq 4$, and let $\mathcal{H} \in M_3$ be a hyperplane of maximum value. If the bounds on δ_1, δ_2 and δ_3 from Theorem 1 are satisfied with equality, then there are a line $\ell \subset \mathcal{H}$ and a plane $\mathcal{P} \subset \mathcal{H}$, such that

$$\begin{aligned}
\nu(p) &= \delta_0 - 3, \quad \forall p \in \ell \cap \mathcal{P}, \\
\nu(p) &= \delta_0 - 2, \quad \forall p \in (\ell \cup \mathcal{P}) \backslash (\ell \cap \mathcal{P}), \\
\nu(p) &= \delta_0 - 1, \quad \forall p \in \mathcal{H} \backslash (\mathcal{P} \cup \ell).
\end{aligned}$$

Proof: Clearly $\delta_0 - 1$ is the maximum value of any point in $\mathcal{H}$. We call it the default value. Let $\mathcal{L}$ be the set of points of reduced value, i.e. the points having value less than $\delta_0 - 1$.

The total value of $\mathcal{H}$ is

$$\delta_0 + \delta_1 + \delta_2 + \delta_3 = (q^3 + q^2 + q + 1)(\delta_0 - 1) - (q^2 + 2q + 2), \tag{3.4}$$

hence $\#\mathcal{L} \leq q^2 + 2q + 2$.

Because the code is an ENDS, any line in $\mathcal{H}$ has value at most $\delta_0 + \delta_1 - 1 = (q+1)(\delta_0 - 1) - 1$, and so has at least one point of reduced value. Hence $\mathcal{L}$ is a 2-blocking set or it contains a plane. If it is a 2-blocking set, $\#\mathcal{L} \geq q^2 + 3q$, by Lemma 5, which is impossible for $q \geq 3$. Thus there is a plane, $\mathcal{P}$, contained in $\mathcal{L}$.

Say that the default values of points in $\mathcal{P}$ is $\delta_0 - 2$. Let $\alpha \neq \mathcal{P}$ be another plane in $\mathcal{H}$. Obviously $\mathcal{P} \cap \alpha$ is a line, so assuming default values,

$$\nu(\alpha) = (q^2 + q + 1)\delta_0 - q^2 - 2q - 2 = \delta_2 + \delta_1 + \delta_0.$$

Thus $\alpha \in M_2$, which is impossible. Hence we have to reduce the value further for at least one point in α, i.e. in each plane different from $\mathcal{P}$.

We have established the default values of $\delta_0 - 2$ for points in $\mathcal{P}$, and $\delta_0 - 1$ for points in $\mathcal{H} \backslash \mathcal{P}$. We let $\mathcal{K}$ be the set of points, whose values we decrease further, i.e. such that points in $\mathcal{P} \cap \mathcal{K}$ have value at most $\delta_0 - 3$ and points in $\mathcal{K} \backslash \mathcal{P}$ have value at most $\delta_0 - 2$.

From equation (3.4) we get that

$$(q^2 + 2q + 2) \geq \#\mathcal{P} + \#\mathcal{K}.$$

Hence $\#\mathcal{K} \leq q + 1$.

We had to reduce the values of all planes except possibly $\mathcal{P}$, so $\mathcal{K}$ must meet all such planes. First we assume that $\mathcal{K}$ does not meet $\mathcal{P}$. Then we can add one point, $\wp$, from $\mathcal{P}$ to $\mathcal{K}$, to get $\mathcal{K}' = \wp \cup \mathcal{K}$ with at most $q+2$ points. By Lemma 6, $\mathcal{K}'$ cannot be a 1-blocking set, so it contains a line, ψ. There are q points in $\phi \cap \mathcal{K}$ and at most 1 point in $\mathcal{K} \backslash \phi$, and thus $\mathcal{K}$ is contained in some plane, Γ.

The intersection $\beta \stackrel{\text{def}}{=} \Gamma \cap \mathcal{P}$ has no point in $\mathcal{K}$. There are a total of $q+1 \geq 4$ planes through β, including $\mathcal{P}$ and Γ. Since $\mathcal{K} \subset \Gamma$, there are several planes which does not meet $\mathcal{K}$. The contradiction shows that $\mathcal{K}$ meets $\mathcal{P}$, and thus $\mathcal{K}$ is a line, ℓ, by Lemma 1.

There are exactly $q+1$ points in $\ell = \mathcal{K}$, so we cannot reduce the value of any point in $\mathcal{K}$ by more than one. This proves the lemma. $\square$

Corollary 4
If $(\delta_0, \delta_1, \ldots \delta_k)$ is a non-binary ENDS for $k \geq 4$, then $\delta_0 \geq 3$.

We will return to binary codes later, and prove that they cannot meet the bounds of Theorem 1 with equality.

3.3 Bounds on the total value

We have established bounds for all differences but the last. The following is an extension of Theorem 3 in [CK97a].

Theorem 2
If $(\delta_0, \delta_1, \ldots \delta_k)$, $k \geq 2$, satisfies $(\mathrm{N}m.m)$, then

$$\nu(\mathsf{PG}(k,q)) \leq \sum_{i=0}^{m-1} \delta_i + (\delta_m - 1) \sum_{i=0}^{k-m} q^i, \quad \forall 1 \leq m \leq k-1.$$

Proof: Let $\alpha \in M_{m-1}$. In $\mathsf{PG}(k,q)$, there are $Q(k-m)$ m-spaces containing α, and for any such m-space, $\beta \subset \alpha$, we know by condition $(\mathrm{N}m.m)$ that

$$\nu(\beta \backslash \alpha) \leq \delta_m - 1.$$

Thus $\nu(\mathsf{PG}(k,q) \backslash \alpha) \leq (\delta_m - 1)Q(k-m)$. By the definition of α, we know that

$$\nu(\alpha) = \sum_{i=0}^{m-1} \delta_i,$$

and the theorem follows. □

For an ENDS, several bounds may be derived from the above theorem. The bound in the following corollary is tight for the code constructed in Chapter 4. When we take up the study of binary codes, we will derive another bound from the theorem.

Corollary 5
For any $(\delta_0, \delta_1, \ldots \delta_k)$, $k \geq 2$, *if* (N1.1) *is satisfied, then*

$$\nu(\mathsf{PG}(k,q)) \leq \delta_0 + (\delta_1 - 1)\sum_{i=0}^{k-1} q^i \leq \left(\sum_{i=0}^{k} q^i\right)\delta_0 - (q+2)\sum_{i=0}^{k-1} q^i.$$

Chapter 4

Construction of a non-binary ENDS

We recall the upper bounds on δ_1 through δ_3 given by Theorem 1, and the bound on $\nu(\mathsf{PG}(4,q))$ from Corollary 5. Assuming that $q \geq 3$ and $\delta_0 \geq 3$, we want to construct a value assignment, $\nu : \mathsf{PG}(4,q) \to \mathbb{N}_0$, meeting these bounds with equality. That is

$$\begin{aligned} \delta_1 &= q\delta_0 - (q+1) \\ \delta_2 &= q^2\delta_0 - (q^2+q+1) \\ \delta_3 &= q^3\delta_0 - (q^3+q^2+q+1) \\ \nu(\mathsf{PG}(4,q)) &= (q^4+q^3+q^2+q+1)\delta_0 - (q^4+3q^3+3q^2+3q+2). \end{aligned}$$

This means that the maximum values of subspaces are:

$$\begin{aligned} \nu(\wp) &= \delta_0, \quad \forall \wp \in M_0 \\ \nu(\ell) &= (q+1)(\delta_0 - 1), \quad \forall \ell \in M_1 \\ \nu(\mathcal{P}) &= (q^2+q+1)\delta_0 - q^2 - 2q - 2, \quad \forall \mathcal{P} \in M_2 \\ \nu(\mathcal{H}) &= (q^3+q^2+q+1)\delta_0 - q^3 - 2q^2 - 3q - 3, \quad \forall \mathcal{H} \in M_3 \end{aligned}$$

The construction was first made in $\mathsf{PG}(4,3)$ and $\mathsf{PG}(4,4)$ using a computer program. This process is described in detail in Chapter 10.

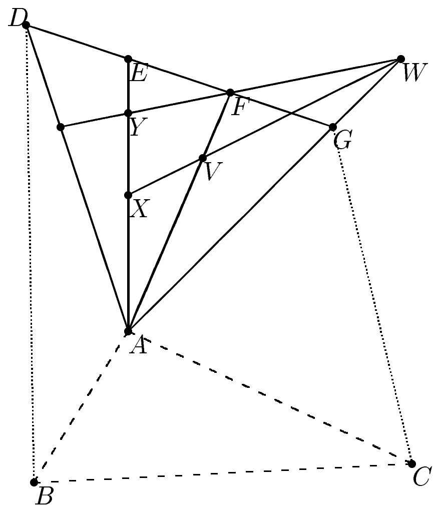

Figure 4.1: Parts of our optimal construction in $\mathsf{PG}(4,q)$, $q \geq 3$. Black lines are in $\langle A, D, G\rangle$, dotted and dashed lines are not.

4.1 The assignment

As reference points, we chose five points, A, B, C, D and E, spanning $\mathsf{PG}(4,q)$.

Consider the line $\langle D, E\rangle$. It has $q+1$ points, which we name $\rho_0 = D, \rho_1 = E, \rho_2, \rho_3, \ldots, \rho_q$. These points define $q+1$ hyperplanes with $\langle A, B, C\rangle$ as a subspace, and we call them $\mathcal{G}_i \stackrel{\text{def}}{=} \langle A, B, C, \rho_i\rangle$ for $i = 0, 1, \ldots, q$. We define $F \stackrel{\text{def}}{=} \rho_2$ and $G \stackrel{\text{def}}{=} \rho_3$. We also establish names for the affine spaces we get by removing $\langle A, B, C\rangle$ from $\mathcal{G}_i$:

$$\mathcal{A}_i \stackrel{\text{def}}{=} \mathcal{G}_i \backslash \langle A, B, C\rangle, \quad i = 0, 1, \ldots, q.$$

During the construction we will pay special attention to the points in $\mathcal{G}_0$, $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$. The remaining hyperplanes $\mathcal{G}_i$ will become isomorphic to $\mathcal{G}_0$ with respect to the value assignment.

4.1.1 Defining subspaces of maximum values

First we decide that $\mathcal{G}_0$ be in M_3, and assign values satisfying the requirements from Lemma 12:

$$\begin{aligned} \nu(p) &= \delta_0 - 3, \quad \forall p \in \langle A, B\rangle \\ \nu(p) &= \delta_0 - 2, \quad \forall p \in \langle A, B, C\rangle \backslash \langle A, B\rangle \\ \nu(p) &= \delta_0 - 1, \quad \forall p \in \langle A, B, C, D\rangle \backslash \langle A, B, C\rangle \end{aligned}$$

We go on by constructing a plane in M_2. For this, we choose $\mathcal{P} \stackrel{\text{def}}{=} \langle C, D, G\rangle$ and assign values such that it satisfies the requirements from Lemma 11:

$$\begin{aligned} \nu(p) &= \delta_0 - 2, \quad \forall p \in \langle C, G\rangle \\ \nu(p) &= \delta_0 - 1, \quad \forall p \in \langle C, D, G\rangle \backslash \langle C, G\rangle \end{aligned}$$

We note that $\mathcal{P}$ has but one point, C, in common with $\langle A, B, C\rangle$, which also means that it intersects each of the affine spaces, $\mathcal{A}_i$, in q points.

Now we choose a point for M_0. We want to place it in $\mathcal{G}_1$, particularly on the line $\langle A, E\rangle$. We will have to refer to two distinct points on this line, so let:

$$X, Y \in \langle A, E\rangle \backslash \{A, E\}, \quad X \neq Y$$

Let $\wp \stackrel{\text{def}}{=} \{X\} \in M_0$ and assign $\nu(X) = \delta_0$.

It remains to choose a line, ℓ, for M_1, so let $\ell \stackrel{\text{def}}{=} \langle F, Y\rangle$. We know that each point on ℓ must have value $\delta_0 - 1$.

Having established members of the sets M_0, M_1, M_2, M_3, we turn to the problem of giving the remaining points values just low enough to satisfy the non-chain conditions. We keep in mind that X will be the only point in M_0, so for the remainder of the chapter we will assign no value greater than $\delta_0 - 1$.

4.1.2 Lines through X

For any line, $\alpha \ni X$, we have

$$\nu(\alpha) \leq (q+1)(\delta_0 - 1) - 1 = \delta_0 + q(\delta_0 - 1) - 2,$$

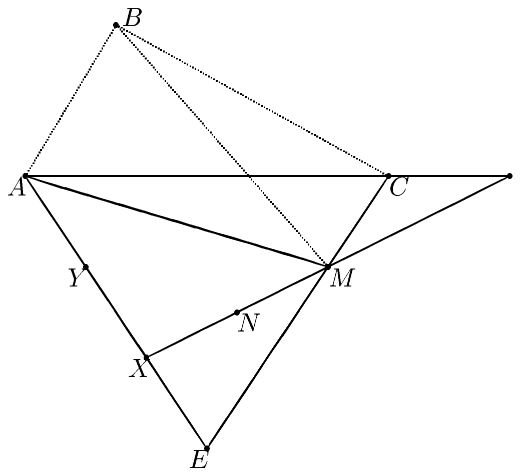

Figure 4.2: The hyperplane $\mathcal{G}_1$ in our optimal construction for $\mathsf{PG}(4,q)$. Black lines are in the plane $\langle A, C, E\rangle$, dotted lines are not.

because the DS is an ENDS. Hence α must have a point of value $\delta_0 - 3$ or less, or at least two points of value $\delta_0 - 2$.

First consider the lines through X, that does not intersect $\langle A, B, C\rangle$. We can handle a lot of these lines by assigning the value $\delta_0 - 3$ to all points in $\mathcal{A}_3$ that have not got any value yet:

$$\nu(p) = \delta_0 - 3, \quad \forall p \in \mathcal{A}_3 \backslash (\mathcal{P} \cup \ell).$$

Let α_i for $i = 1, 2, \ldots, q$ be the (distinct) lines through X meeting $\mathcal{A}_3$ in a point of value $\delta_0 - 2$, i.e. meeting $\langle C, G\rangle \backslash \{C\}$. The lines, α_i, meet $\mathcal{A}_2$ in distinct points. Let $\mathcal{T}$ be the set of these points:

$$\mathcal{T} \stackrel{\text{def}}{=} \bigcup_{i=1}^{q} (\alpha_i \cap \mathcal{A}_2) = \mathcal{A}_2 \cap \langle X, C, G\rangle.$$

Let $\mathcal{S} \subset \mathcal{A}_2$ be the set of points in $\mathcal{A}_2$, to which we have already assigned values. Thus

$$\mathcal{S} \stackrel{\text{def}}{=} \mathcal{A}_2 \cap (\mathcal{P} \cup \ell) = \mathcal{A}_2 \cap \langle C, D, G\rangle.$$

Obviously the plane $\langle X, C, G\rangle$ does not meet $\mathcal{S}$, and consequently $\mathcal{T}$ does not meet $\mathcal{S}$. We assign the value $\delta_0 - 2$ to each point in $\mathcal{T}$:

$$\nu(p) = \delta_0 - 2, \quad \forall p \in \mathcal{A}_2 \cap \langle X, C, G\rangle.$$

Now $\nu(\alpha_i) < \nu(\ell)$ for any i.

There is one point of value $\delta_0 - 1$ in $\mathcal{A}_3$, namely the point on ℓ. We call this point W, and let V be the point where the line $\langle W, X\rangle$ intersects $\mathcal{A}_2$. We refer to Figure 4.2, and note that $V \in \langle A, F\rangle \backslash \{F\}$, and consequently $V \notin \mathcal{T} \cup \mathcal{S}$. Hence V has no value, and we can assign a value of $\delta_0 - 3$, so that $\nu(\langle W, X\rangle) < \nu(\ell)$:

$$\begin{aligned} W &\in \ell \cap \mathcal{G}_3 \\ V &\in \langle W, X\rangle \cap \mathcal{G}_2 \\ \nu(V) &= \delta_0 - 3. \end{aligned}$$

This concludes the study of lines through X, that does not intersect $\langle A, B, C\rangle$. Now we turn to lines through X that does intersect $\langle A, B, C\rangle$. In other words, we are to consider the values of points in $\mathcal{A}_1$.

Let $M \neq E$ be a point in $\mathcal{A}_1 \cap \mathcal{P}$.

$$M \in \mathcal{P} \cap \mathcal{A}_1 \backslash \{E\}.$$

Define $\psi \stackrel{\text{def}}{=} \langle A, B, M\rangle$. We have assigned values to $\{X, Y\} \subset \langle A, E\rangle$ and to $\langle C, E\rangle \subset \mathcal{P}$. The remaining points in $\mathcal{A}_1$ have no value yet. We note that $\langle A, E\rangle \cap \psi = \{A\}$ and $\langle C, E\rangle \cap \psi = \{M\}$. Hence the only points in ψ with assigned values are M and the points on $\langle A, B\rangle$. We assign the value $\delta_0 - 2$ to the remaining points in ψ:

$$\nu(p) = \delta_0 - 2, \quad \forall p \in \langle A, B, M\rangle \backslash (\langle A, B\rangle \cup \{M\}).$$

Finally consider the line $\langle M, X\rangle$. We have assigned values to M and X, and also to $\langle M, X\rangle \cap \langle A, B, C\rangle$. The remaining points on $\langle M, X\rangle$ have no value, hence there is at least one point $N \in \langle M, X\rangle$ with no value yet. We give it the value $\delta_0 - 2$:

$$\begin{aligned} N &\in \langle M, X\rangle \backslash (\{M, X\} \cup \langle A, B, C\rangle) \\ \nu(N) &= \delta_0 - 2 \end{aligned}$$

value of p	p in	number of new points
δ_0	$\{X\}$	1
δ_0-3	$\langle A,B\rangle$	$q+1$
δ_0-2	$\langle A,B,C\rangle\backslash\langle A,B\rangle$	q^2
δ_0-1	$\mathcal{A}_0$	q^3
	$\cup_{i\geq 4}\mathcal{A}_i$	$(q-3)q^3$
	$\langle C,D,G\rangle\backslash\langle C,G\rangle$	$2q$
	$\langle F,Y\rangle$	2
	$\mathcal{A}_2\backslash(\{V\}\cup\langle X,C,G\rangle)$	q^3-2q-1
	$\mathcal{A}_1\backslash(\{N\}\cup\langle A,B,M\rangle$	q^3-q^2-q-1
δ_0-2	$\mathcal{A}_2\cap\langle X,C,G\rangle$	q
	$\langle C,G\rangle$	q
	$\langle A,B,M\rangle\backslash(\langle A,B\rangle\cup\{M\})$	q^2-1
	$\{N\}$	1
δ_0-3	$\mathcal{A}_3\backslash(\mathcal{P}\cup\ell)$	q^3-q-1
	$\{V\}$	1

Table 4.1: A general value assignment for $\mathsf{PG}(4,q)$, $q\geq 3$, $\delta_0\geq 3$.

Now consider any line α, such that $X\in\alpha\subset\mathcal{G}_1$. If α meets $\langle A,B\rangle$, it has a point of value δ_0-3, and so $\nu(\alpha)<\nu(\ell)$. Otherwise α meets $\langle A,B,C\rangle$ in a point of value δ_0-2, but then it also meets $\langle A,B,M\rangle$ in some distinct point, ρ. Either $\nu(\rho)=\delta_0-2$, or $\rho=M$, and $N\in\alpha$. In either case α has at least two points of value δ_0-2, and consequently $\nu(\alpha)<\nu(\ell)$.

The preceding argument proves that for any line, $\alpha\ni X$, $\nu(\alpha)<\nu(\ell)$.

4.1.3 Finishing touches

All remaining points get the value δ_0-1. We summarise the assignment in a table 4.1. The total value is easily verified by summing the values in the table.

4.2 Verifying the non-chain conditions

4.2.1 The point

The only point of value δ_0 or more is X, so

$$M_0 = \{\{X\}\}.$$

4.2.2 Lines

We know that $\nu(\ell) = (q+1)(\delta_0 - 1)$. Any line, $\alpha' \not\ni X$, consists of $q+1$ points of value at most $\delta_0 - 1$, so $\nu(\alpha') \leq (q+1)(\delta_0 - 1) = \nu(\ell)$, with equality if and only if any point on α' has value $\delta_0 - 1$. In Section 4.1.2, we proved that any line, α, containing X have value $\nu(\alpha) < \nu(\ell)$. This proves that $\ell \in M_1$, and (N1.1) follows.

We observe that any line, $\ell' \in M_1$, does not meet $\langle A, B, C\rangle$, because no point in $\langle A, B, C\rangle$ has value $\delta_0 - 1$. This implies that $\ell' \not\subset \mathcal{G}_i$ for any i.

There may be several lines of value $\delta_1 + \delta_0$, and we don't give any expression for number of such lines, except that at least one such line exists.

4.2.3 Planes

The study of planes is quite tedious. We will go as far as to prove that $\mathcal{P}$ is the only plane of maximum value. We suppose for a contradiction that $\gamma \neq \mathcal{P}$ is a plane, such that $\nu(\gamma) \geq \nu(\mathcal{P})$.

First assume that $X \in \gamma$. We have showed that any line, through X, has value at most $(q+1)\delta_0 - q - 2$; and there are $q+1$ such lines in γ. Hence

$$\begin{aligned}\nu(\gamma) &\leq (q+1)[(q+1)\delta_0 - q - 2 - \delta_0] + \delta_0 \\ &\leq (q^2+q+1)\delta_0 - q^2 - 3q - 2 < \nu(\mathcal{P}),\end{aligned}$$

contrary to assumption; hence $X \notin \gamma$. Condition (N1.2) follows immediately.

Suppose that $\gamma \subset \mathcal{G}_i$ for some i. The intersection $\gamma \cap \langle A, B, C\rangle$ is a line with one point of value $\delta_0 - 3$, and q points of value at most $\delta_0 - 2$. The remaining q^2 points have value at most $\delta_0 - 1$, since $X \notin \gamma$. Hence the total value is

$$\nu(\gamma) \leq (q^2+q+1)\delta_0 - q^2 - 2q - 3 < \nu(\mathcal{P}),$$

contrary to assumption. We conclude that $\gamma \not\subset \mathcal{G}_i$ for any i.

Consider the possibility that $\langle C, G\rangle \subset \gamma$. All points on $\langle C, G\rangle$ have value $\delta_0 - 2$, so $\nu(\langle C, G\rangle) = (q+1)\delta_0 - 2q - 2$. There is a point of intersection in $\gamma \cap \langle A, B, M\rangle$. If γ should meet $\langle A, B\rangle$, γ would be in $\mathcal{G}_3$, which it is not. If $M \in \gamma$, then $\gamma = \mathcal{P}$, contrary to assumption. Thus γ meets $\langle A, B, M\rangle$ in a point of value $\delta_0 - 2$. There are $q^2 - 1$ more points in $\gamma\backslash\langle C, G\rangle$, and since $X \notin \gamma$, no point has value greater than $\delta_0 - 1$. This gives a total value of $\nu(\gamma) \leq (q^2 + q + 1)\delta_0 - 2q - 3 < \nu(\mathcal{P})$, which is impossible.

As $\langle C, G\rangle \not\subset \gamma$, the intersection $\gamma \cap \mathcal{A}_3$ has at most two points of value $\delta_0 - 2$ or more. One of them may be W, with value $\delta_0 - 1$; and the other may be on the line $\langle C, G\rangle$ and have value $\delta_0 - 2$. Hence

$$\nu(\gamma \cap \mathcal{A}_3) \leq q\delta_0 - 3q + 3, \tag{4.1}$$

with equality if and only if $W \in \gamma$, and γ meets $\langle C, G\rangle$ in a point different from C.

Each of the other affine spaces, $\mathcal{A}_i$ for $i \neq 1$, meet γ in q points of value at most $\delta_0 - 1$. Hence

$$\nu(\gamma \cap \mathcal{A}_i) \leq q\delta_0 - q, \quad \forall i \neq 3, \tag{4.2}$$

with equality if and only if any point in $\gamma \cap \mathcal{A}_i$ has value $\delta_0 - 1$.

Finally γ meets $\langle A, B, C\rangle$ in a point, Ψ, of value at most $\delta_0 - 2$. Thus the total value

$$\nu(\gamma) \leq (q^2 + q + 1)\delta_0 - q^2 - 3q + 1. \tag{4.3}$$

For $q = 3$ this is exactly the value of $\mathcal{P}$, so equality is necessary. For $q \geq 4$, $\nu(\gamma) < \nu(\mathcal{P})$, contrary to assumption. Thus γ can only exist if $q = 3$.

To get equality in (4.3), $\nu(\Psi) = \delta_0 - 2$, so γ cannot meet $\langle A, B\rangle$. Also all points in $\gamma\backslash\mathcal{G}_3$ must have value $\delta_0 - 1$, thus the intersection of γ and $\langle A, B, M\rangle$ is M. We also need equality in (4.1), so $W \in \gamma$, and γ meets $\langle C, G\rangle\backslash\{C\}$. This shows that $\gamma \subset \langle W, M, C, G\rangle = \langle C, D, G, X\rangle$. Let $\Pi_3 \stackrel{\text{def}}{=} \langle C, D, G, X\rangle$, which is a three-space (see figure 4.3). The fact that $A \notin \gamma$ implies that $G \notin \gamma$, so there are but two possible choices for γ.

We know that $\langle C, G, X\rangle \cap \mathcal{G}_2$ is a line, β, of points of value $\delta_0 - 2$. Both β and γ are in Π_3, so they meet in some point of value $\delta_0 - 2$. However $C \notin \gamma$, and all points in $\gamma \cap \mathcal{A}_2$ have value $\delta_0 - 1$, and thus we have a contradiction.

This shows that no other plane has as high value as $\mathcal{P}$, so $M_2 = \{\mathcal{P}\}$. Any line in $\mathcal{P}$ contains a point of value $\delta_0 - 2$, so (N2.2) is satisfied.

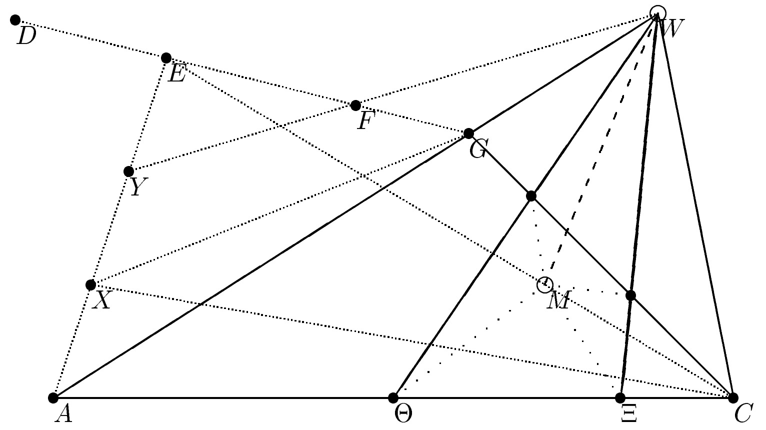

Figure 4.3: There are two choices for γ: $\langle M, W, \Xi\rangle$ and $\langle M, W, \Theta\rangle$. The planes $\langle C, G, X\rangle$ and γ meet in a line.

4.2.4 Hyperplanes

We have assigned the value $\delta_0 - 1$ to any point in $\mathcal{A}_0$ and in $\mathcal{A}_i$ for $i > 3$. As stated, $\mathcal{G}_0$ has the desired maximum value and so have $\mathcal{G}_i$ for $i > 3$. Each of $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$, have several points of value less than $\delta_0 - 1$ and at most one, namely X, of value higher than $\delta_0 - 1$. Hence $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3 \not\subset M_3$.

We will prove that the hyperplanes of maximum values are exactly $\mathcal{G}_0$ and $\mathcal{G}_i$ for $i > 3$. Suppose for a contradiction that there is a hyperplane, $\aleph \neq \mathcal{G}_i$ for all i, such that $\nu(\aleph) \geq \nu(\mathcal{G}_0)$.

First we assume that $X \in \aleph$. We have proved that any line through X has value at most $(q+1)(\delta_0 - 1) - 1$, and $\aleph$ contains $q^2 + q + 1$ such lines, and has the value

$$\begin{aligned}\nu(\aleph) &\leq (q^3 + q^2 + q)(\delta_0 - 1) - 2(q^2 + q + 1) + \delta_0 \\ &= (q^3 + q^2 + q + 1)\delta_0 - (q^3 + 3q^2 + 3q + 3) < \nu(\mathcal{G}_0),\end{aligned}$$

contrary to assumption. Thus we conclude that $X \notin \aleph$. Condition (N1.3) follows immediately.

Now $\aleph \cap \mathcal{G}_3$ is a plane, $\alpha \neq \langle A, B, C\rangle$, which intersects $\langle A, B, C\rangle$ in a line with one point of value $\delta_0 - 3$ and q points of value at most $\delta_0 - 2$, which gives

$$\nu(\aleph \cap \langle A, B, C\rangle) \leq (q+1)\delta_0 - 2q - 3. \tag{4.4}$$

Now consider values of points in $\mathcal{A}_3$. Most of the points in $\mathcal{A}_3$ have value $\delta_0 - 3$. Exceptions are the q points in $(\langle C, G\rangle \backslash \{C\}) \subset \mathcal{P}$ with value $\delta_0 - 2$, and the point $W \in \ell$ which has value $\delta_0 - 1$. Thus the intersection $\alpha \cap \mathcal{A}_3$ has at most $q+1$ points of value greater than or equal to $\delta_0 - 2$, out of which one, W, can have value $\delta_0 - 1$ and the remaining q points can have value $\delta_0 - 2$. This gives

$$\nu(\aleph \cap \mathcal{A}_3) \leq q^2\delta_0 - 3q^2 + q + 2. \tag{4.5}$$

There are q^3 points in $\aleph \backslash \mathcal{G}_3$, and since $X \notin \aleph$, all these points have value $\delta_0 - 1$ or less. Hence

$$\nu(\aleph \backslash \mathcal{G}_3) \leq q^3(\delta_0 - 1). \tag{4.6}$$

We sum up equations (4.4), (4.5), and (4.6), to get

$$\nu(\aleph) \leq Q(3)\delta_0 - q^3 - 3q^2 - q - 1 \leq Q(3)\delta_0 - q^3 - 2q^2 - 3q - 4 < \nu(\mathcal{G}_0),$$

contrary to assumption.

We conclude that

$$M_3 = \{\mathcal{G}_0, \mathcal{G}_i \mid i > 3\}.$$

We showed in Section 4.2.2, that for any $\ell' \in M_1$, $\ell' \not\subset \mathcal{G}_i$ for any i; hence (N2.3) follows. Obviously $\mathcal{P} \not\subset \mathcal{G}_i$ for any i, and there is no other plane in M_2; this proves (N3.3).

We have proved that all the non-chain conditions hold, so the construction is an ENDS.

4.3 The conclusion

Our construction proves the following theorem.

Theorem 3
If $q \geq 3$ and $\delta_0 \geq 3$, there exists a five-dimensional, extremal non-chain code with the difference sequence:

$$
\begin{aligned}
\delta_1 &= q\delta_0 - (q+1) \\
\delta_2 &= q^2\delta_0 - (q^2+q+1) \\
\delta_3 &= q^3\delta_0 - (q^3+q^2+q+1) \\
\delta_4 &= q^4\delta_0 - (q^4+2q^3+q^2)
\end{aligned}
$$

Chapter 5

Binary ENDS

We remember from Chapter 3, several bounds on the differences. We know from the work of Chen and Kløve [CK99], that binary four-dimensional codes must meet stronger bounds. In this chapter we will prove such upper bounds for arbitrary dimension. A five-dimensional code meeting the bounds with equality, will be provided in the next chapter.

We start with the general bounds. In Section 5.2 we will prove some special properties for codes meeting the bounds with equality. Finally we establish a lower bound on δ_0 in Section 5.3.

5.1 General bounds

The following is a generalisation of Lemma 1 part ii) from [CK99], and it gives a stronger bound for the second to the last difference, δ_{k-1}.

Theorem 4
If $(\delta_0, \delta_1, \dots \delta_k)$, $k \geq 3$, is a binary ENDS, then

$$\delta_{k-1} \leq 2^{k-2}\delta_1 - 2 - 2^{k-2}.$$

Proof: Take $\Pi_{k-1} \in M_{k-1}$ and $\Pi_{k-2} \in M_{k-2}$, and let $\Pi_{k-3} \stackrel{\text{def}}{=} \Pi_{k-1} \cap \Pi_{k-2}$. Because the code is extremal non-chain, Π_{k-3} is a $(k-3)$-space. Also let $\{P\} \in M_0$.

Define

$$\mathcal{S} \stackrel{\text{def}}{=} \Pi_{k-2} \backslash \Pi_{k-3} = \{S_i \mid i = 1, 2, \dots, 2^{k-2}\}, \tag{5.1}$$

$$\ell_i \stackrel{\text{def}}{=} \langle P, S_i \rangle = \{P, S_i, T_i\}, \quad i = 1, 2, \dots, 2^{k-2}. \tag{5.2}$$

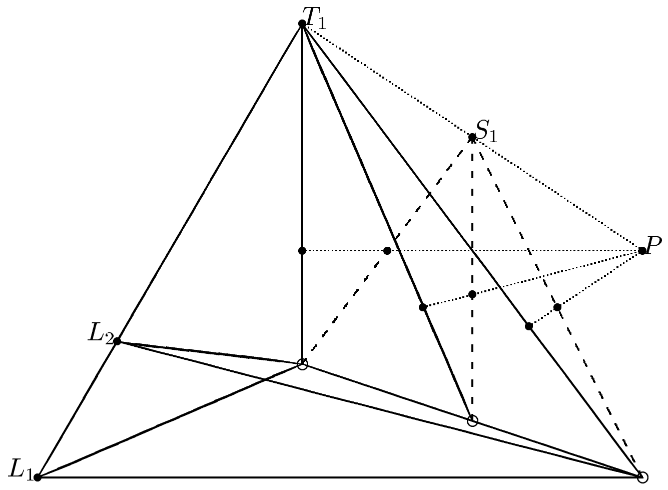

Figure 5.1: Representation of $\mathsf{PG}(4,2)$ for Theorem 4. Black lines are in Π_3, dashed lines in Π_2, and dotted lines are in neither. White points are in Π_1. The points L_1 and L_2, with Π_1, span $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively.

Any line through P meets Π_{k-1}, so the points T_i are in Π_{k-1}. Define the set

$$\mathcal{T} \stackrel{\text{def}}{=} \{T_i \mid i = 1, 2, \ldots, 2^{k-2}\}. \tag{5.3}$$

Because the code is an ENDS, $\nu(\ell_i) \leq \delta_0 + \delta_1 - 1$, for all i; hence

$$\nu(T_i) \leq \delta_1 - \nu(S_i) - 1, \quad i = 1, 2, \ldots 2^{k-2},$$

and

$$\nu(\mathcal{T}) \leq 2^{k-2}\delta_1 - \nu(\mathcal{S}) - 2^{k-2}.$$

We know that

$$\nu(\Pi_{k-2}) = \nu(\mathcal{S}) + \nu(\Pi_{k-3}) = \sum_{i=0}^{k-2} \delta_i,$$

so

$$\nu(\mathcal{T}) - \nu(\Pi_{k-3}) \leq 2^{k-2}\delta_1 - 2^{k-2} - \sum_{i=0}^{k-2} \delta_i.$$

The join of $\{P\}$ and Π_{k-2} is a $(k-1)$-space, intersecting Π_{k-1} in a $(k-2)$-space, namely $\mathcal{T} \cup \Pi_{k-3}$. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the other two (distinct) $(k-2)$-spaces, such that $\Pi_{k-3} \subset \mathcal{L}_i \subset \Pi_{k-1}$, for $i = 1, 2$.

Now we have

$$\begin{aligned} \sum_{i=0}^{k-1} \delta_i = \nu(\Pi_{k-1}) &= \nu(\mathcal{L}_1) + \nu(\mathcal{L}_2) - \nu(\Pi_{k-3}) + \nu(\mathcal{T}) \\ &\leq 2\left(\sum_{i=0}^{k-2} \delta_i - 1\right) + 2^{k-2}\delta_1 - 2^{k-2} - \sum_{i=0}^{k-2} \delta_i. \end{aligned}$$

This is simplified to

$$\delta_{k-1} \leq 2^{k-2}\delta_1 - 2^{k-2} - 2,$$

and the lemma is proved. □

Corollary 6
No extremal non-chain, binary code of dimension four or more, can meet the bound on δ_{k-1} from Theorem 1 with equality.

Proof: Theorem 4, combined with the bound on δ_1 from Theorem 1, gives

$$\begin{aligned} \delta_{k-1} &\leq 2^{k-2}\delta_1 - 2 - 2^{k-2} \\ &\leq 2^{k-1}\delta_0 - 2^{k-2} \cdot 3 - 2 - 2^{k-2} = 2^{k-1}\delta_0 - q^k - 2 \\ &= 2^{k-1}\delta_0 - (q^k - 1) - 3 = [2^{k-1}\delta_0 - Q(k-1)] - 3, \end{aligned}$$

which is a stronger bound than the bound from Theorem 1. □

For the last difference, δ_k, we use Theorem 2, to get a bound expressed in terms of δ_{k-1}. This bound coincides with the one given in [CK99].

Theorem 5
If $(\delta_0, \delta_1, \ldots \delta_k)$, $k \geq 2$, is a binary ENDS, then

$$\delta_k \leq 2\delta_{k-1} - 3.$$

Proof: From Theorem 2, we have for $m = k-1$,

$$\sum_{i=0}^{k} \delta_i \leq \sum_{i=0}^{k-2} \delta_i + \delta_{k-1}(q+1) - (q+1) = \sum_{i=0}^{k-1} \delta_i + 2\delta_{k-1} - 3,$$

and the lemma follows immediately. □

For the remaining differences we use the bounds known from Theorem 1. We will see that for five-dimensional codes they are tight.

For a five-dimensional ENDS, we get the following bounds:

$$\delta_1 \leq 2\delta_0 - 3 \tag{5.4}$$
$$\delta_2 \leq 4\delta_0 - 7 \tag{5.5}$$
$$\delta_3 \leq 4\delta_1 - 6 \tag{5.6}$$
$$\delta_4 \leq 2\delta_3 - 3 \tag{5.7}$$

The two first bounds, (5.4) and (5.5), follow directly from Theorem 1, the third, (5.6), comes from Theorem 4, and the fourth one, (5.7), comes from Theorem 5.

5.2 Construction requirements

Lemma 13

If $(\delta_0, \delta_1, \delta_2, \delta_3, \delta_4)$ is a binary ENDS, and $\delta_1 = 2\delta_0 - 3$ and $\delta_2 = 4\delta_0 - 7$, then for any $\wp \in M_0$, any $\ell \in M_1$, and any $\mathcal{P} \in M_2$, we have:

$$\ell \cap \mathcal{P} = (\ell\wp) \cap \mathcal{P} = \{X\} \in \mathsf{PG}^{(0)}(4,2).$$

Proof: We will first prove that that $\ell \cap \mathcal{P}$ is a point. Because the code is an ENDS, $\ell \not\subset \mathcal{P}$, so $\ell \cap \mathcal{P}$ is either a point or the empty set. Suppose for a contradiction that $\ell \cap \mathcal{P} = \emptyset$.

We let $\wp = \{P\}$ be a point of maximum value, and $\alpha \stackrel{\text{def}}{=} \ell\wp$. In $\mathsf{PG}(4,2)$ any two planes must meet, and $\ell \subset \alpha$ does not meet $\mathcal{P}$, so α and $\mathcal{P}$ meet in a point, O. Let $\langle O, P\rangle = \{N, O, P\}$. Since ℓ doesn't meet $\mathcal{P}$, and $P \notin \ell$, $N \in \ell$.

By Lemma 10, $\nu(N) = \delta_0 - 1$, and

$$m(\{N, O, P\}) \leq \delta_1 + \delta_0 - 1 = 3\delta_0 - 4,$$

so $\nu(O) \leq \delta_0 - 3$. Hence $\delta_2 < 4\delta_0 - 7$ by Lemma 11. It follows by contradiction that $\ell \cap \mathcal{P}$ is a point, X.

It remains to prove that $(\wp\ell) \cap \mathcal{P}$ is a point. We have showed that there is a point $X \in \ell \cap \mathcal{P} \subset (\wp\ell) \cap \mathcal{P}$. Suppose for a contradiction that $(\wp\ell) \cap \mathcal{P}$ is a line. Then $\aleph \stackrel{\text{def}}{=} (\wp\ell)\mathcal{P}$ is a 3-space. We consider the code defined by the value assignment restricted to $\aleph$, i.e. $\nu|_\aleph$. Obviously $\wp$, ℓ and $\mathcal{P}$ are contained in $\aleph$, and having maximum values in the original geometry, they have in particular maximum value in $\aleph$. Hence the difference sequence of $\nu|_\aleph$ is $(\delta_0, \delta_1, \delta_2, \delta_3')$, for some δ_3'.

There are no subspaces satisfying either of the conditions, (C$i.j$), in $\mathsf{PG}(k, 2)$, so in particular no such subspaces can exist in $\aleph$. Hence $\nu|_\aleph$ define a four-dimensional ENDS, and by Theorems 4 and 1,

$$\delta_2 \leq 2\delta_1 - 4 \leq 4\delta_0 - 10,$$

and the lemma follows by contradiction. $\square$

5.3 Lower bound on δ_0

In practice our construction of a binary, five-dimensional ENDS requires $\delta_0 \geq 4$, so it is interesting to know if any smaller value for δ_0 is possible. We remember that Chen and Kløve [CK99] showed that $\delta_0 \geq 5$ for four-dimensional codes, and they did so by first proving lower bounds on the other differences. A complete study of lower bounds is beyond the scope of this thesis, but we introduce two lemmas which will be used to prove that $\delta_0 \geq 4$ for any binary, five-dimensional ENDS in the theorem at the end of this section.

Lemma 14
If $(\delta_0, \delta_1, \delta_2, \delta_3, \delta_4)$ is a binary ENDS, and

$$\lambda = \min\{\nu(p) \mid p \in \ell\},$$

for some $\ell \in M_1$, then

$$\delta_4 \geq \delta_2 + \delta_0 + 1 + \lambda.$$

Proof: Let $\wp \in M_0$, $\ell \in M_1$, $\mathcal{P} \in M_2$, and $\mathcal{H} \in M_3$ be subspaces of maximum value. We know that $\mathcal{H}$ has value $\delta_0 + \delta_1 + \delta_2 + \delta_3$, and its complement, $\mathcal{H}^{\mathrm{C}}$, has value δ_4.

The intersection $\mathcal{P} \cap \mathcal{H}$ is a line of value at most $\delta_1 + \delta_0 - 1$, so $\nu(\mathcal{P} \cap \mathcal{H}^{\mathrm{C}}) \geq \delta_2 + 1$. We know that ℓ has two points in $\mathcal{H}^{\mathrm{C}}$, at most one of which may coincide with a point in $\mathcal{P}$, and the other has value at least λ. Finally $\wp \subset \mathcal{H}^{\mathrm{C}}$, so $\delta_4 \geq \delta_2 + \delta_0 + 1 + \lambda$, as required. $\square$

Lemma 15
If $(\delta_0, \delta_1, \dots \delta_k)$, $k \geq 3$, *is a binary ENDS, then*

$$\delta_1 \geq 2 \tag{5.8}$$
$$\delta_2 \geq \delta_0 + 1. \tag{5.9}$$

Proof: Assume $\delta_1 = 1$. Let $\wp \in M_0$ be a point of maximum value. Obviously $\wp$ cannot be the only point of positive value, so let $\rho \neq \wp$ be another such point. The line $\wp\rho$ has value at least $\delta_0 + 1 = \delta_1 + \delta_0$, so it is a line of maximum value, and the code is no ENDS. We conclude that $\delta_1 \geq 2$.

Let $\ell \in M_1$ and $\wp \in M_0$ be a line and a point of maximum value. If $\delta_2 \leq \delta_0$, then

$$\nu(\ell\wp) \geq 2\delta_0 + \delta_1 \geq \delta_2 + \delta_1 + \delta_0,$$

and the code is no ENDS, contrary to assumption. We conclude that $\delta_2 \geq \delta_0 + 1$. □

Theorem 6
If $(\delta_0, \delta_1, \delta_2, \delta_3, \delta_4)$ *is a binary ENDS, then* $\delta_0 \geq 4$.

Proof: We let ℓ and λ be defined as in Lemma 14.

We have from Theorem 5 that

$$\delta_4 \leq 16\delta_0 - 39,$$

and from Lemma 14 that

$$\delta_4 \geq \delta_2 + \delta_0 + 1 + \lambda.$$

Combining these inequalities, we have

$$15\delta_0 \geq 40 + \delta_2 + \lambda.$$

This gives

$$\delta_0 \geq 2 + \frac{10 + \delta_2 + \lambda}{15}.$$

So either $\delta_0 \geq 4$, or $\delta_0 = 3$ and $\delta_2 + \lambda \leq 5$.

We consider $\delta_0 = 3$. Then $\delta_2 \geq \delta_0 + 1 = 4$, so that $\lambda \leq 1$. We have upper and lower bounds for the value of ℓ:

$$5 \leq \delta_0 + \delta_1 = \nu(\ell) \leq 2(\delta_0 - 1) + \lambda = 4 + \lambda \leq 5.$$

It follows that $\delta_1 = 2$ and $\lambda = 1$, and $4 \leq \delta_2 \leq 5 - \lambda = 4$, so $\delta_2 = 4$.

If $\mathcal{P} \in M_2$ is a plane of maximum value, then $\nu(\mathcal{P}) = \delta_2 + \delta_1 + \delta_0 \geq 4+2+3 = 9$. There are seven points in $\mathcal{P}$, so at least one point has value at least $\lceil 9/7 \rceil = 2$, and no point has value more that $\delta_0 - 1 = 2$. We let $P \in \mathcal{P}$ be a point of value 2. There are three lines through P in $\mathcal{P}$, each of which has value at most 4, or otherwise it would be in M_1. Thus the value of $\mathcal{P}$ is at most $3 \cdot 2 + 2 = 8$, which is impossible. Hence $\delta_0 \geq 4$, as required. $\square$

Chapter 6

Binary Construction

In this chapter we construct a value assignment for a five-dimensional, binary code, with the difference sequence $(4, 5, 9, 14, 25)$, which meets the bounds from Chapter 5 with equality. It is easily verified that by uniformly increasing the values of all points, we can get any value for $\delta_0 \geq 4$, and equality is maintained.

6.1 The assignment

There are 31 points in $\mathsf{PG}(4, 2)$, and we name them

$$A, B, C, \ldots, Y, Z, \Gamma, \Delta, \Theta, \Lambda, \Xi.$$

The geometry is defined by the list of lines in Table 6.1. All points are shown in Figure 6.1. Figure 6.2 is a simplified version, showing the points treated specially in the text. We note that the five points A, C, G, K and W span $\mathsf{PG}(4, 2)$.

We arbitrarily choose $\mathcal{H} \stackrel{\text{def}}{=} \langle A, C, G, W \rangle$ as a hyperplane of maximum value, and $\mathcal{P} \stackrel{\text{def}}{=} \langle A, C, K \rangle$ as a plane of maximum value. We define the hyperplanes $\mathcal{G} \stackrel{\text{def}}{=} \langle A, C, G, K \rangle$ and $\mathcal{I} \stackrel{\text{def}}{=} \langle A, C, G, \Delta \rangle$. Thus $\mathcal{P} \subset \mathcal{G}$ and the intersection of the three hyperplanes $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{I}$, is the plane $\langle A, C, G \rangle$.

We let the line of maximum value, ℓ, intersect each of the hyperplanes $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{I}$ in one point. We make sure that it intersects $\mathcal{P}$, and let $\ell \stackrel{\text{def}}{=} \ell_{95} = \{H, T, \Delta\}$. As a point of maximum value, we choose X which is in $\mathcal{I}$.

ℓ_i	$\nu(\ell_i)$	the points
ℓ_0	8	$A(3), B(3), C(2)$
ℓ_1	7	$A(3), D(1), E(3)$
ℓ_2	8	$A(3), F(3), G(2)$
ℓ_3	8	$A(3), H(3), I(2)$
ℓ_4	8	$A(3), J(2), K(3)$
ℓ_5	5	$A(3), L(1), M(1)$
ℓ_6	5	$A(3), N(1), O(1)$
ℓ_7	6	$A(3), P(1), Q(2)$
ℓ_8	6	$A(3), R(2), S(1)$
ℓ_9	8	$A(3), T(3), U(2)$
ℓ_{10}	7	$A(3), V(2), W(2)$
ℓ_{11}	8	$A(3), X(4), Y(1)$
ℓ_{12}	5	$A(3), Z(1), \Gamma(1)$
ℓ_{13}	7	$A(3), \Delta(3), \Theta(1)$
ℓ_{14}	3	$A(3), \Lambda(0), \Xi(0)$
ℓ_{15}	7	$B(3), D(1), F(3)$
ℓ_{16}	8	$B(3), E(3), G(2)$
ℓ_{17}	8	$B(3), H(3), J(2)$
ℓ_{18}	8	$B(3), I(2), K(3)$
ℓ_{19}	5	$B(3), L(1), N(1)$
ℓ_{20}	5	$B(3), M(1), O(1)$
ℓ_{21}	6	$B(3), P(1), R(2)$
ℓ_{22}	6	$B(3), Q(2), S(1)$
ℓ_{23}	8	$B(3), T(3), V(2)$
ℓ_{24}	7	$B(3), U(2), W(2)$
ℓ_{25}	8	$B(3), X(4), Z(1)$
ℓ_{26}	5	$B(3), Y(1), \Gamma(1)$
ℓ_{27}	6	$B(3), \Delta(3), \Lambda(0)$
ℓ_{28}	4	$B(3), \Theta(1), \Xi(0)$
ℓ_{29}	5	$C(2), D(1), G(2)$
ℓ_{30}	8	$C(2), E(3), F(3)$
ℓ_{31}	8	$C(2), H(3), K(3)$
ℓ_{32}	6	$C(2), I(2), J(2)$
ℓ_{33}	4	$C(2), L(1), O(1)$
ℓ_{34}	4	$C(2), M(1), N(1)$
ℓ_{35}	4	$C(2), P(1), S(1)$
ℓ_{36}	6	$C(2), Q(2), R(2)$
ℓ_{37}	7	$C(2), T(3), W(2)$
ℓ_{38}	6	$C(2), U(2), V(2)$
ℓ_{39}	7	$C(2), X(4), \Gamma(1)$
ℓ_{40}	4	$C(2), Y(1), Z(1)$
ℓ_{41}	5	$C(2), \Delta(3), \Xi(0)$
ℓ_{42}	3	$C(2), \Theta(1), \Lambda(0)$
ℓ_{43}	5	$D(1), H(3), L(1)$
ℓ_{44}	4	$D(1), I(2), M(1)$
ℓ_{45}	4	$D(1), J(2), N(1)$
ℓ_{46}	5	$D(1), K(3), O(1)$
ℓ_{47}	5	$D(1), P(1), T(3)$
ℓ_{48}	5	$D(1), Q(2), U(2)$
ℓ_{49}	5	$D(1), R(2), V(2)$
ℓ_{50}	4	$D(1), S(1), W(2)$
ℓ_{51}	8	$D(1), X(4), \Delta(3)$
ℓ_{52}	3	$D(1), Y(1), \Theta(1)$
ℓ_{53}	2	$D(1), Z(1), \Lambda(0)$
ℓ_{54}	2	$D(1), \Gamma(1), \Xi(0)$
ℓ_{55}	7	$E(3), H(3), M(1)$
ℓ_{56}	6	$E(3), I(2), L(1)$
ℓ_{57}	6	$E(3), J(2), O(1)$
ℓ_{58}	7	$E(3), K(3), N(1)$
ℓ_{59}	6	$E(3), P(1), U(2)$
ℓ_{60}	8	$E(3), Q(2), T(3)$
ℓ_{61}	7	$E(3), R(2), W(2)$
ℓ_{62}	6	$E(3), S(1), V(2)$
ℓ_{63}	8	$E(3), X(4), \Theta(1)$
ℓ_{64}	7	$E(3), Y(1), \Delta(3)$
ℓ_{65}	4	$E(3), Z(1), \Xi(0)$
ℓ_{66}	4	$E(3), \Gamma(1), \Lambda(0)$
ℓ_{67}	7	$F(3), H(3), N(1)$
ℓ_{68}	6	$F(3), I(2), O(1)$
ℓ_{69}	6	$F(3), J(2), L(1)$
ℓ_{70}	7	$F(3), K(3), M(1)$
ℓ_{71}	6	$F(3), P(1), V(2)$
ℓ_{72}	7	$F(3), Q(2), W(2)$
ℓ_{73}	8	$F(3), R(2), T(3)$
ℓ_{74}	6	$F(3), S(1), U(2)$
ℓ_{75}	7	$F(3), X(4), \Lambda(0)$
ℓ_{76}	4	$F(3), Y(1), \Xi(0)$
ℓ_{77}	7	$F(3), Z(1), \Delta(3)$
ℓ_{78}	5	$F(3), \Gamma(1), \Theta(1)$
ℓ_{79}	6	$G(2), H(3), O(1)$
ℓ_{80}	5	$G(2), I(2), N(1)$
ℓ_{81}	5	$G(2), J(2), M(1)$
ℓ_{82}	6	$G(2), K(3), L(1)$
ℓ_{83}	5	$G(2), P(1), W(2)$
ℓ_{84}	6	$G(2), Q(2), V(2)$
ℓ_{85}	6	$G(2), R(2), U(2)$
ℓ_{86}	6	$G(2), S(1), T(3)$
ℓ_{87}	6	$G(2), X(4), \Xi(0)$
ℓ_{88}	3	$G(2), Y(1), \Lambda(0)$
ℓ_{89}	4	$G(2), Z(1), \Theta(1)$
ℓ_{90}	6	$G(2), \Gamma(1), \Delta(3)$
ℓ_{91}	8	$H(3), P(1), X(4)$
ℓ_{92}	6	$H(3), Q(2), Y(1)$
ℓ_{93}	6	$H(3), R(2), Z(1)$
ℓ_{94}	5	$H(3), S(1), \Gamma(1)$
ℓ_{95}	9	$H(3), T(3), \Delta(3)$
ℓ_{96}	6	$H(3), U(2), \Theta(1)$
ℓ_{97}	5	$H(3), V(2), \Lambda(0)$
ℓ_{98}	5	$H(3), W(2), \Xi(0)$
ℓ_{99}	4	$I(2), P(1), Y(1)$
ℓ_{100}	8	$I(2), Q(2), X(4)$
ℓ_{101}	5	$I(2), R(2), \Gamma(1)$
ℓ_{102}	4	$I(2), S(1), Z(1)$
ℓ_{103}	6	$I(2), T(3), \Theta(1)$
ℓ_{104}	7	$I(2), U(2), \Delta(3)$
ℓ_{105}	4	$I(2), V(2), \Xi(0)$
ℓ_{106}	4	$I(2), W(2), \Lambda(0)$
ℓ_{107}	4	$J(2), P(1), Z(1)$
ℓ_{108}	5	$J(2), Q(2), \Gamma(1)$
ℓ_{109}	8	$J(2), R(2), X(4)$
ℓ_{110}	4	$J(2), S(1), Y(1)$
ℓ_{111}	5	$J(2), T(3), \Lambda(0)$
ℓ_{112}	4	$J(2), U(2), \Xi(0)$
ℓ_{113}	7	$J(2), V(2), \Delta(3)$
ℓ_{114}	5	$J(2), W(2), \Theta(1)$
ℓ_{115}	5	$K(3), P(1), \Gamma(1)$
ℓ_{116}	6	$K(3), Q(2), Z(1)$
ℓ_{117}	6	$K(3), R(2), Y(1)$
ℓ_{118}	8	$K(3), S(1), X(4)$
ℓ_{119}	6	$K(3), T(3), \Xi(0)$
ℓ_{120}	5	$K(3), U(2), \Lambda(0)$
ℓ_{121}	6	$K(3), V(2), \Theta(1)$
ℓ_{122}	8	$K(3), W(2), \Delta(3)$
ℓ_{123}	5	$L(1), P(1), \Delta(3)$
ℓ_{124}	4	$L(1), Q(2), \Theta(1)$
ℓ_{125}	3	$L(1), R(2), \Lambda(0)$
ℓ_{126}	2	$L(1), S(1), \Xi(0)$
ℓ_{127}	8	$L(1), T(3), X(4)$
ℓ_{128}	4	$L(1), U(2), Y(1)$
ℓ_{129}	4	$L(1), V(2), Z(1)$
ℓ_{130}	4	$L(1), W(2), \Gamma(1)$
ℓ_{131}	3	$M(1), P(1), \Theta(1)$
ℓ_{132}	6	$M(1), Q(2), \Delta(3)$
ℓ_{133}	3	$M(1), R(2), \Xi(0)$
ℓ_{134}	2	$M(1), S(1), \Lambda(0)$
ℓ_{135}	5	$M(1), T(3), Y(1)$
ℓ_{136}	7	$M(1), U(2), X(4)$
ℓ_{137}	4	$M(1), V(2), \Gamma(1)$
ℓ_{138}	4	$M(1), W(2), Z(1)$
ℓ_{139}	2	$N(1), P(1), \Lambda(0)$
ℓ_{140}	3	$N(1), Q(2), \Xi(0)$
ℓ_{141}	6	$N(1), R(2), \Delta(3)$
ℓ_{142}	3	$N(1), S(1), \Theta(1)$
ℓ_{143}	5	$N(1), T(3), Z(1)$
ℓ_{144}	4	$N(1), U(2), \Gamma(1)$
ℓ_{145}	7	$N(1), V(2), X(4)$
ℓ_{146}	4	$N(1), W(2), Y(1)$
ℓ_{147}	2	$O(1), P(1), \Xi(0)$
ℓ_{148}	3	$O(1), Q(2), \Lambda(0)$
ℓ_{149}	4	$O(1), R(2), \Theta(1)$
ℓ_{150}	5	$O(1), S(1), \Delta(3)$
ℓ_{151}	5	$O(1), T(3), \Gamma(1)$
ℓ_{152}	4	$O(1), U(2), Z(1)$
ℓ_{153}	4	$O(1), V(2), Y(1)$
ℓ_{154}	7	$O(1), W(2), X(4)$

Table 6.1: The lines in $\mathsf{PG}(4,2)$, with the value of each point in parenthesis after the point.

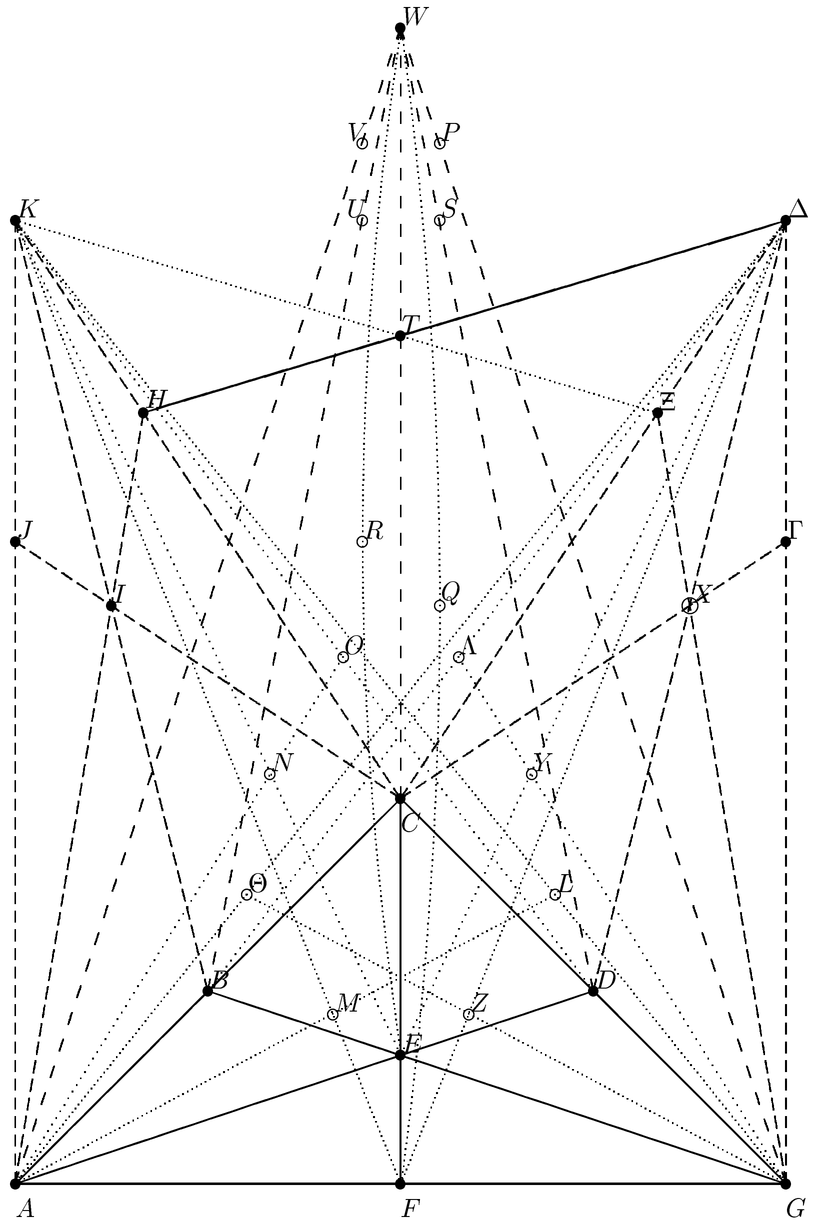

Figure 6.1: Representation of $\mathsf{PG}(4,2)$ for construction.

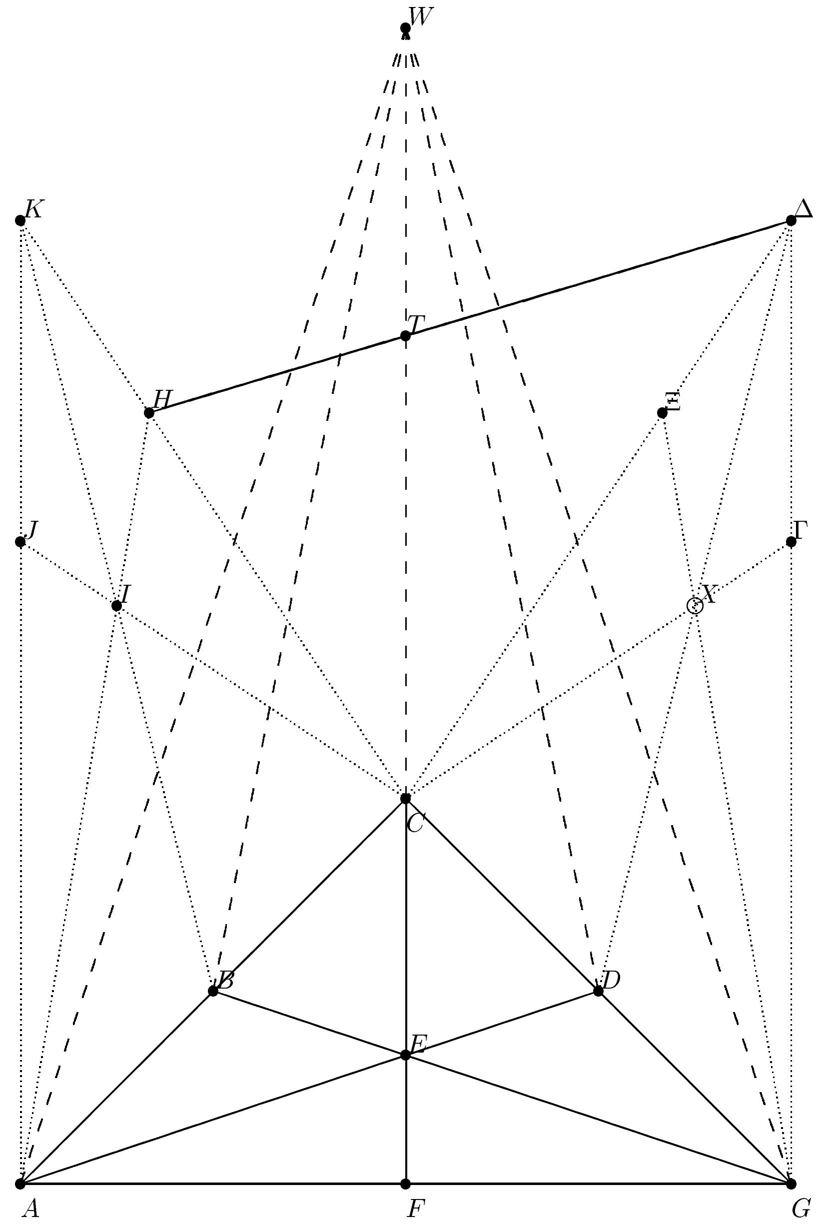

Figure 6.2: Simplified sketch of $\mathsf{PG}(4,2)$ for construction.

$\nu(A) = 3$	$\nu(B) = 3$	$\nu(C) = 2$	$\nu(D) = 1$	$\nu(E) = 3$	$\nu(F) = 3$
$\nu(G) = 2$	$\nu(H) = 3$	$\nu(I) = 2$	$\nu(J) = 2$	$\nu(K) = 3$	$\nu(L) = 1$
$\nu(M) = 1$	$\nu(N) = 1$	$\nu(O) = 1$	$\nu(P) = 1$	$\nu(Q) = 2$	$\nu(R) = 2$
$\nu(S) = 1$	$\nu(T) = 3$	$\nu(U) = 2$	$\nu(V) = 2$	$\nu(W) = 2$	$\nu(X) = 4$
$\nu(Y) = 1$	$\nu(Z) = 1$	$\nu(\Gamma) = 1$	$\nu(\Delta) = 3$	$\nu(\Theta) = 1$	$\nu(\Lambda) = 0$
$\nu(\Xi) = 0$					

Table 6.2: A value assignment in $\mathsf{PG}(4, 2)$.

Now we assign the following values:

$$\nu(X) = 4 \tag{6.1}$$
$$\nu(p) = 3, \quad \forall p \in \ell \tag{6.2}$$
$$\nu(p) = 2, \quad \forall p \in \langle C, I \rangle \tag{6.3}$$
$$\nu(p) = 3, \quad \forall p \in \mathcal{P} \backslash \langle C, I \rangle \tag{6.4}$$

This gives us sufficient information to limit our search, so that it may feasibly be done by trial and error on a computer. The value assignment shown in Table 6.2, has been found by a computer program as described in Section 10.2. It can be verified that it is extremal non-chain, and that its difference sequence is indeed $(4, 5, 9, 14, 25)$. We summarise the result in the following theorem:

Theorem 7
If $\delta_0 \geq 3$, there exists a binary, five-dimensional, extremal non-chain code with the difference sequence:

$$\begin{aligned}
\delta_1 &= 2\delta_0 - 3 \\
\delta_2 &= 4\delta_0 - 7 \\
\delta_3 &= 4\delta_1 - 6 \\
\delta_4 &= 2\delta_3 - 3
\end{aligned}$$

Part II

The Computer

Chapter 7

Programs

This and the following chapters describe a set of computer programs, which have been used to validate the results and create sample constructions for the thesis.

We start in this chapter by giving an overview, and describing some general modules. In the Chapters 8 and 9, we will look at modules related to projective geometries and value assignments, respectively. Chapter 10 provides an example of how to use **vsh**(1). Unix manual pages for the programs and file formats can be found in Chapter 11.

The header files provides a comment for every function, and is suggested for an overview of functions.

A key task of the system is to determine the difference sequence of a value assignment, as well as which of the conditions (C$i.j$) and (C0) are satisfied. We call this to *classify* the value assignment.

7.1 Program overview

- **vsh**(1) is an interactive program to inspect value assignments and their properties. The program is command driven, and it can also run in batch. It has been the most useful tool so far.

- **pgc**(1) creates the projective geometry and sets up the data structure for **gen**(1).

- **gen**(1) searches a large set of possible value assignments and classifies them.

- **vdg**(1) translates the binary output from **gen**(1) to readable output. It can also classify value assignments given on file.
- **pidcheck**(1) is yet another companion of **gen**(1), written to restart **gen**(1) automatically when it is accidentally killed. It is supposed to be run by **cron**(8).
- **pgtest**(1) is not a very refined tool. It was first created to test functions and debug the data structure, but it can indeed do some computations in projective geometry, and it can translate between different file formats.

7.2 Design and implementation

The original idea was to search a large number of value assignments for 'good' difference sequences. Efficiency was the main goal in the early stages of programming. The data structure is large, because recomputation of data is costly. The choice of C as the programming language was also made to get efficiency. Despite all efforts, feasible applications have not been found for codes of dimension 5 or higher.

When the idea of extensive searches was abandoned, a program to classify a given value assignment was desired. The first implementation was **vdg**(1), which takes the value assignments from file. A much better solution for our purposes has been to use an interactive program, so that values may quickly be changed and the assignment reclassified. This idea is implemented in **vsh**(1).

When **vsh**(1) was written, a lot of C code existed, and it was a natural choice to use it to test the new ideas. Significant extensions became necessary, however, and the system has grown a bit too large to be easily maintained in C. To work in large geometries, it is no longer practical to look up incidences in arrays, because the data structure grows too large. For instance, it takes **vsh**(1) about five minutes on a 300MHz Ultra Sparc to build the data structure for $\mathsf{PG}(5,3)$, using about 34Mb of memory.

If a similar project is undertaken in the future, **vsh**(1) should probably be reimplemented from scratch. With a reasonable representation of points and subspaces, any incidence relations can be computed at need. Efficiency is not likely to be a problem for any interactive program. Another suggested feature is some way to extract subspaces from a construction, and classify the subcodes they define; and conversely to take a construction and use it as a subspace in a new construction. Better object-orientation is expected to make both features possible.

The system is divided into a number of modules, but the modularisation is only partial. To keep things as simple as possible, principles of encapsulation are frequently violated. Some modules fail to deallocate memory. This has been done deliberately when the consequences are believed to be negligible.

The system is not robust, and hardly any error checking is made. In **vsh**(1) one can easily experience segmentation violation by giving invalid arguments to commands.

7.3 Description of components

7.3.1 Miscellaneous modules

aux (`aux.h,aux.c`) contains functions to write standardised messages on stdout, sending e-mail reports, and reading and writing short integers on file in hardware independent binary format.

prep (`prep.h,prep.c`) sets up the global data structure used by **gen**(1) and **vdg**(1).

vsh (`vsh.c,vsh.h`) is the main function for **vsh**(1).

lex (`lex.l,vsh.h`) provides the command parser for **vsh**(1), implemented in flex(1).

pgc (`pgc.c`) is the main function for **pgc**(1).

gen (`gen.c`) is the main function for **gen**(1).

signal (`signal.h,signal.c`) controls the signal handling for **gen**(1).

vdg (`vdg.c`) is the main function for **vdg**(1).

pgtest (`pgtest.c`) is the main function for **pgtest**(1).

7.3.2 Common header files

- `common.h` contains common macros and definitions, and it includes other common header files.
- `errors.h` defines error messages and exit codes.
- `gen.h` contains configuration settings intended to be specified at compile time.

7.3.3 Implementation of pidcheck

This program is intended to be run by crontab(1), and automatically restart programs when they accidentally get killed. Simple scripts have been used to install a cron job before the main process (i.e. **gen**(1)) was started. The main process (e.g. **gen**(1)) creates a file named 'active', with all data needed by **pidcheck**(1).

To check if the process is still running, pidcheck uses sigsend(2) with the null signal.

The **pidcheck**(1) program does not share any modules with the other programs. The following modules are used by **pidcheck**(1):

logenv (`logenv.h`,`logenv.c`) reads the active file and exports the data to the environment for **pidcheck**(1). It can also read the same data from the environment.

pidcheck (`pidcheck.c`) contains the main function.

Chapter 8

Projective Geometry Modules

8.1 Geometrical representations

A *coordinate representation* of a point in $\mathsf{PG}(k,q)$, is a vector in a $(k+1)$-dimensional vector field over $\mathsf{GF}(q)$. Each projective point has $q-1$ representatives. When a unique representative is needed, we use the vector with one as the first non-zero coordinate. We call this the *normal* representative.

We represent the elements of a Galois field by numbers of type *char*. A vector over a small field, is represented by an array of $k+1$ *char*s.

Occasionally an ordering of the points is necessary. We order the points according to their coordinates, i.e. $(x_0,\ldots x_k)$ comes before $(y_0,\ldots y_k)$ if for some i, $0 \le i \le k$, $x_i < y_i$ and $x_j = y_j$ for all $j < i$; according to the numeric values of the coordinates. This ordering induces a numbering, where the first point in the order is point no. 0. This provides us with an alternate and more compact representation of the points, called the *number representation*.

8.2 Overview

8.2.1 The simple representation

When **gen**(1) does an exhaustive search, we want it to look up subspace relations as fast as possible. Therefore we use the number representation of points, and keep all relations between subspaces and superspaces in arrays. This is handled by the module *pg*.

Module: *pg*
Files: `pg.h,pg.c`
This module defines the *t_pg* struct, which contains the tables of subspace relations. To avoid repeating expensive exponentiations, there are also tables for the number of points in any subspace, the number of subspaces of any given dimensions, and the number of m-spaces containing a given n-space.

The tables of subspace relations are loaded from file, while the others are computed using functions in the module. All functions to create the files are in the module *pginit*. Points are represented by the data type *t_ps*, using the number representation described above.

The subspace relations are stored in the member `ps` of *t_pg*. This member is a four-dimensional array of type *t_ps*. If `ps[i][j][k][l]` is equal to m, it means that j-space no. m is a superspace of i-space no. k. The different superspaces of the same i-space are sorted, so that `ps[i][j][k][l]` is less than `ps[i][j][k][l+1]`. This member must in most cases be directly referenced, while the other members should be accessed using the macros defined in `pg.h`.

8.2.2 The coordinate representation

Using the coordinate representation of points, all other properties of the geometry can be computed. The following modules take care of the basic operations required.

Module: *gf*
Files: `gf.h,gf.c`
This module handles multiplication and addition in small Galois Fields. The multiplication and addition tables are hard coded in the module, as we don't expect to need fields of order greater than five.

The field elements are represented as *chars*, $0, 1, \ldots, q-1$, where q is the order of the field. When q is prime, arithmetic operations modulo q are used. Otherwise the different symbols are arbitrarily chosen, except that 1 and 0 are the multiplicative and additive identities, respectively.

Module: *points*
Files: `points.h,points.c`
This module provides data types for projective points, represented by numbers or by coordinates.

There are two functions to translate a normal representative to a number. They work recursively, so that pg_getno() translate the first coordinates up to the first non-zero entry, and then pt_vno() take the last coordinates, as if it were an affine coordinate vector. In the same way, pt_fromno() and pt_nov() translate from numbers to coordinates.

The module provides functions for addition and scalar multiplication of vectors, and the function pt_normalize(), which is used to replace an arbitrary representative vector by the normal representative.

Finally there is a printing routine, writing the coordinates in ASCII.

Module: *base*
Files: `base.h,base.c`
This module defines the struct *t_bas* for a basis of points. The functions include bas_comb(bas,d,k,q) to form a linear combination of the points in a *t_bas*. The integer d is written in radix q, so that $d = \sum_{i=0}^{m} d_i q^i$ and the point returned is found by adding d_0 times the last point in the basis, d_1 times the second last, and so on, up to d_m times the first point in the basis, where m is the dimension of the basis. The normal representative of the new point is returned.

8.2.3 The complete incidence tables

These modules find subspaces by use of the coordinate representation of points. They are required to create the arrays used in the simple representation.

Module: *subspace*
Files: `subspace.h,subspace.c`
The struct *t_msp* defines a subspace, represented by a basis (of type *t_bas*) and by a unique number assigned when the space is created.

The member `pt` is a character array with an entry for each point, marking it either TRUE, for a member, or FALSE for a non-member. Thus we can quickly determine whether a given point is contained in the subspace or not. The member `bas` is a basis of type *t_bas*.

There is a function to determine if a given space is a subspace of another. This works by testing if each point in the basis of the first space is contained in the other one.

Module: *pgmsp*
Files: `pgmsp.h,pgmsp.c`
This module defines the *t_s* struct, containing pointers to *t_msp* structs for every subspace in the geometry. The module also provides different printing routines for **vsh**(1), depending on a global variable of type *t_s*. These functions should be able to produce lists of superspaces containing a given subspace and vice versa.

Module: *pginit*
Files: `pginit.h,pginit.c`
This module contains the functions to create subspace tables and store them on file. Refer to Section 8.3.1 for details on the implementation.

Module: *pgaux*
Files: `pginit.h,pgaux.c`
This module defines a quasi-queue of points, *t_ptq*, which is used by *pginit*. It has an array with an entry for each point, marking it either TRUE for a member or FALSE for a non-member. It has functions to return and remove the next point, and to simply remove a set of points.

8.3 More details on the implementation

8.3.1 Creating the geometry

The projective geometry is created by using the pg_create() function. The main part of the algorithm is as follows:

1. $s :=$ new instance of *t_s*.
2. $ptno :=$ the number of points in $\mathsf{PG}(k, q)$.
3. $j := 0$
4. for $p := 0 \ldots ptno - 1$:
    (a) Create a *t_bas*, *bas*, of dimension 0 with p as the only point.
    (b) $s.s[0][p] :=$ a new instance of *t_msp*, spanned by *bas*.
5. for $m := 0 \ldots k - 2$:
    (a) $elno :=$ number of m-spaces in $\mathsf{PG}(k, q)$.
    (b) For $i := 0 \ldots elno - 1$.

i. $Q :=$ a queue containing all points.
ii. Remove from Q, all points in $s.s[m][i]$.
iii. Remove from Q, all points contained in some previously created $(m+1)$-space, $\Pi \supset s.s[m][i]$.
iv. For each point $p \in Q$:
   A. Let $s.s[m+1][j]$ be the subspace spanned by the points in the basis of $s.s[m][i]$ and the point p.
   B. Remove the points in $s.s[m+1][j]$ from Q.
   C. For $e := i+1 \ldots elno - 1$:
      - If $s.s[m+1][j] \supset s.s[m][e]$, insert $s.s[m+1][j]$ in the array, $pg.ps$, as a superspace of $s.s[m][e]$.
   D. Insert $s.s[m+1][j]$ in the array in pg as a superspace of $s.s[m][i]$.
   E. $++j$

Now we have recorded for each m-space, Π, the $(m+1)$-spaces containing Π. It remains to record the m'-spaces containing Π for $m' > m+1$. We do this the trivial way, by checking for every possible combination of an m-space and an m'-space.

Chapter 9

Value assignment modules

The simplest data structure for a value assignment, is a struct, t_v, with a pointer to an array with an element storing the value for each point. However, to classify the value assignment, we need a more complicated structure, t_val, storing also the values for each subspace.

9.1 Value assignments

The discussion of the value assignment modules is split into two subsections. The so-called basic modules are the ones used by **gen**(1), and they do not use global variables. The so-called complex modules depend on global variables, especially a t_s variable.

9.1.1 Basic modules

Module: *val*
Files: `val.h,val.c`
This module defines the t_val struct which stores the value assignment, including the values of subspaces. There are functions for file handling and printing of ASCII representations. Values of individual points may be set, subspace values may be recomputed; the value assignment can be classified, and one can step through all possible value assignments.

The t_val struct has a pointer to a set of t_mval structs, storing the maximum values of subspaces, and references to subspaces attaining the maximum value. The `v` member is a pointer to t_v, storing the value assignment and its classification. The member `val` is a two-dimensional array of values for all points and subspaces, such that `val[0]` points to the value assignment stored as a member of `v`. There is also a boolean array marking whether the value of each point is *fixed* or not. Most functions changing values, will not change fixed values.

The module contains several I/O functions, to read and write binary files and to print ASCII output, even though these mainly use the corresponding functions related to the *t_v* structure, using the member `v`.

Module: *valv*
Files: `val.h,valv.c`
The *t_v* struct defined in this module, contains the data we want to store about each value assignment, that is the class number and difference sequence as well as the value of each point. Functions for file handling are included, as well as a comparison function, v_compare(), for use with the binary search trees. The value assignment and the classification can also be printed nicely in ASCII.

The constructor, v_alloc(), allocates memory in such a way that memory cannot be deallocated. It will also fail if several *t_pg* instances are used simultaneously. If one switches between two geometries, v_alloc() must be called with a *t_pg* NULL pointers to reinitialise itself. However, we don't really want to do that, due to the memory leak.

Module: *valaux*
Files: `valaux.h,valaux.c`
This module contains two auxiliary data structures: *t_ds*, which stores a difference sequence, and *t_mval*, which stores the maximum value as well as a stack (of type *t_stack*) of all subspaces attaining this value. There are functions to produce ASCII prints of the data.

Module: *vtree*
Files: `vtree.h,vtree.c`
This module defines the *t_vt* struct, which uses a binary search tree (tsearch(3C)) to store value assignments of type *t_v*. The ordering in the search tree is induced by the function v_compare() from the *valv* module.

A value assignment is inserted by the vt_insert() function, taking a pointer to *t_val* as an argument. If there already is a value assignment with the same difference sequence of the same class, nothing is done, otherwise the *t_v* struct is extracted from *t_val* and inserted in the tree. Then new memory is allocated for a *t_v* instance, and the pointer from *t_val* is set to point to the new memory. This is handled by the val_freev() function from the *val* module.

There are also I/O functions for binary data files and ASCII prints.

Module: *stack*
Files: `stack.c,stack.h`
This is a simple array implementation of a stack of subspaces, *t_stack*, with the additional possibility of traversing the stack without popping, and to discard all elements and replace them with a single new one.

Module: *valex*
Files: `valex.l,val.h`
This is a parser to read value assignments from file. It is implemented in **flex**(1). The file format is described in the man page for **valex**(4) (see Chapter 11).

9.1.2 More complex modules

Module: *pgval*
Files: `pgval.h,pgval.c`
This module provides **vsh**(1) with functions to look up in the geometry and print points and values for subspaces. The implementation resemble that of *pgmsp*, and the two modules depend on the same global variables. This module includes a function to reinitialise the global variables for **vsh**(1).

Module: *valmsp*
Files: `valmsp.h,valmsp.c`
This module depends on a global *t_s* variable, `s`, and it provides **vsh**(1) with functions to set and fix values of all points in a given subspace. It also provides the val_incprobe() function used by the 'probe' command in **vsh**(1).

Module: *nametab*
Files: `nametab.h,nametab.c`
The symbolic names used in **vsh**(1) are defined by this module. Lowercase letters represents a pair (m, s) identifying m-space number s, while uppercase letters represent a number p, identifying point number p. There are functions to assign values to these names, and to take a name and fill the data it represents into an argument vector as used by **vsh**(1).

9.2 Details on the implementation

9.2.1 Exhaustive search

When **gen**(1) searches for value assignments, it initialises the data structure with the val_stepinit() function, and traverses the possible value assignments step by step with the val_stepval() function.

During this search, we try to exclude some assignments which are equivalent to assignments we do check. We do this by choosing a set of projectively independent 'control points' spanning the space, in such a way that the space spanned by the first m control points include exactly the points up to, but not including the next control point. We assume that each control point has at least as high value as any of the succeeding points. We also assume that the first $q+1$ points are sorted by decreasing value.

Since differences must be positive, we initialise all control points, but the first one with 1. The value of the first control point, point no. 0, is set to δ_0.

When we step through the assignment, we start at the last point and increase its value by one. The maximum allowed value of a point is the value of the last control point in front of it, and when this maximum is exceeded, the value is reset to 0 (or 1 for a control point), and the value of the predecessor is increased instead.

If only the value of the last point is changed, we only add one to the value of each subspace containing it. Otherwise we recompute all subspace values using val_sumval().

9.2.2 Allocating memory

Because we keep value assignments in a search tree, a lot of memory is allocated for t_v structs. To reduce the number of calls to **malloc**(3), the v_alloc() function allocates a large amount of memory when it is first called. On later calls, it returns blocks from this memory. When needed, more memory is allocated with **malloc**(3).

No function to deallocate the memory used by t_v is provided, but for the original application, that would not be necessary. New applications may suffer from memory leak.

The number of `size_t` units required for a t_v struct, is kept in the variable `v_size`. To provide aligned pointers, we assume that by adding **sizeof**(**void***) to a pointer aligned for any use, we get a pointer which is also aligned for any use. Therefore we make sure that `v_size` is a multiple of **sizeof**(**void***).

9.2.3 Classification

The *class number* of a value assignment is an integer with one bit for each of the conditions, (C$i.j$), which is defined, including the chain condition, (C0). We associate with each bit, starting with the least significant one, a condition in the following order:

$$(\mathrm{C}1.1), (\mathrm{C}1.2), \ldots (\mathrm{C}1.k-1), (\mathrm{C}2.1), \ldots (\mathrm{C}k-1.1), (\mathrm{C}0).$$

The most significant bit is related to the chain condition. If (C0) is true, the corresponding bit is 1, while all other bits are 0. Hence a five-dimensional, chained code is class no. 64.

If (C0) is false, the corresponding bit is 0, and any other bit is 1 if and only if the corresponding condition is true. Thus extremal non-chain codes are always class no. 0.

The *classification* algorithm (val_getclass()), initialises the class number to 0, and then starts searching for subspaces satisfying the first condition. As soon as a pair of subspaces satisfying the condition is found, the corresponding bit is turned on, and the algorithm proceeds with the next condition.

If all the conditions (C$i.j$) are true, the algorithm checks separately for the chain condition with the function val_chained().

Classification also requires us to find the difference sequence, but that's a straight forward calculation, as long as we continuously keep track of maximum subspace values.

Chapter 10

Using vsh

This chapter shows how **vsh**(1) was used to create the construction of Chapters 4 and 6. It is hoped that it can serve two purposes, by being a realistic example of how to use **vsh**(1), and by outlining the line of thought behind the constructions. Keep in mind, though, that the sessions described are quite experimental in nature, and thus educational clarity may have suffered to the benefit of realism.

10.1 A non-binary construction

10.1.1 Description

For easy reference, the points have the same names in this chapter, as they have in Chapter 4. We use $\delta_0 = 3$ as the maximum value for any point.

We start by loading the geometry with 'pg' and clearing the assignment with 'clear'. The latter also defines a maximal set of projectively independent points, A, B, C, D and E.

```
vsh:1> pg 4 3
vsh:2> clear
```

Our first task is to define the values of points in a hyperplane of maximum value. The command 'spdef' defines a subspace by spanning a set of points, thus we let $a = \langle A, B \rangle$, $b = \langle A, B, C \rangle$, and $h = \langle A, B, C, D \rangle$.

```
vsh:3> spdef a A B
vsh:4> spdef b A B C
vsh:5> spdef h A B C D
```

We give the value 0 to points in a, 1 to points in b and 2 to points in h. We assign values to the smaller sets first, because 'spset' don't change the values of points that already have a value.

```
vsh:6> spset a 0
vsh:7> spset b 1
vsh:8> spset h 2
```

The hyperplane h meets the requirements of Lemma 12.

Now we turn to a line of maximum weight. The 'line' command assigns names the points on line spanned by two given points. The commands below will name the points on $\langle D, E\rangle$, starting with F, and the points on $\langle A, E\rangle$, starting with X.

```
vsh:9> line F D E
vsh:10> line X A E
```

Now D, E, F, G are distinct points on the same line, and A, E, X, Y are distinct points on another line. We let l be the line spanned by the points Y and F and give all its points the value 2.

```
vsh:11> spdef l Y F
vsh:12> spset l 2
```

Thus l meets the requirements of Lemma 11.

We let X be the point of maximum value, and assign a fixed value to it:

```
vsh:13> setfix X 3
```

We need the numbers of the points on some of the lines through X. This can be found with the 'span' command, listing the points with values in the subspace spanned by the given points.

```
vsh:14> span X F
1-space[ 536] (0):   15:2*   41:3*   67:2*   96:-
vsh:15> span X G
1-space[ 509] (0):   14:2*   41:3*   69:-    94:-
```

The total value of the subspace is printed in parenthesis. It is zero because we haven't asked to get it computed yet, neither can it be computed as long as several points (e.g. no. 69 and 96 above) have no value.

Information about a particular subspace is printed with the 'space' command:

```
vsh:16> space l
Points in 1-space #510:
14 42 67 95
vsh:17> span X 95
1-space[ 482] (0):   13:2*   41:3*   68:-    95:2*
```

The output tells us that point no. 95 is the last point on l, that is the intersection of l and $\langle A, B, C, G\rangle$. The line spanned by X and point no. 95 (W) contains point no. 68 (V), to which we will assign a value of 0.

```
vsh:18> setfix 68 0
```

We define a plane $x = \langle A, B, X\rangle$ and assign the value 2 to all its points, which have no value yet.

```
vsh:19> spdef x A B X
vsh:20> spset x 2
```

Before we can run 'probe' with success, we have to establish a plane of maximum value. We define the line $m = \langle C, G\rangle$ and the plane $p = \langle C, D, G\rangle$. The points on m get the value 1 and the remaining points in p get the value 2. Then p meets the requirements of Lemma 11.

```
vsh:21> spdef m C G
vsh:22> spdef p C D G
vsh:23> spset m 1
vsh:24> spset p 2
```

We define the hyperplane g, and assign all its points on g get the fixed value 0.

```
vsh:25> spdef g A B C G
vsh:26> spset g 0
```

We can inspect the entire value assignment with 'pvp':

```
vsh:27> pvp
;
   0:0*    1:0*    2:0*    3:0*    4:1*    5:1*    6:1*    7:1*    8:1*    9:1*
  10:1*   11:1*   12:1*   13:2*   14:2*   15:2*   16:2*   17:2*   18:2*   19:2*
  20:2*   21:2*   22:2*   23:2*   24:2*   25:2*   26:2*   27:2*   28:2*   29:2*
  30:2*   31:2*   32:2*   33:2*   34:2*   35:2*   36:2*   37:2*   38:2*   39:2*
  40:2*   41:3*   42:2*   43:2*   44:2*   45:2*   46:2*   47:2*   48:2*   49:2*
  50:-    51:-    52:-    53:-    54:-    55:-    56:-    57:-    58:2*   59:-
  60:-    61:-    62:-    63:-    64:-    65:-    66:-    67:2*   68:0*   69:-
  70:-    71:-    72:-    73:-    74:-    75:-    76:2*   77:-    78:-    79:-
  80:-    81:-    82:-    83:-    84:-    85:2*   86:-    87:-    88:-    89:-
  90:-    91:-    92:-    93:-    94:1*   95:2*   96:0*   97:0*   98:0*   99:0*
 100:0*  101:0*  102:0*  103:1*  104:0*  105:0*  106:0*  107:0*  108:0*  109:0*
 110:0*  111:0*  112:1*  113:0*  114:0*  115:0*  116:0*  117:0*  118:0*  119:0*
 120:0*
```

A lot of points have no value yet, so we set a default of 1 value with 'cset', but these values are not fixed, as a second run of 'pvp' shows.

```
vsh:28> cset 4 0 1 0
vsh:29> pvp
;
   0:0*    1:0*    2:0*    3:0*    4:1*    5:1*    6:1*    7:1*    8:1*    9:1*
  10:1*   11:1*   12:1*   13:2*   14:2*   15:2*   16:2*   17:2*   18:2*   19:2*
  20:2*   21:2*   22:2*   23:2*   24:2*   25:2*   26:2*   27:2*   28:2*   29:2*
  30:2*   31:2*   32:2*   33:2*   34:2*   35:2*   36:2*   37:2*   38:2*   39:2*
  40:2*   41:3*   42:2*   43:2*   44:2*   45:2*   46:2*   47:2*   48:2*   49:2*
  50:1    51:1    52:1    53:1    54:1    55:1    56:1    57:1    58:2*   59:1
  60:1    61:1    62:1    63:1    64:1    65:1    66:1    67:2*   68:0*   69:1
  70:1    71:1    72:1    73:1    74:1    75:1    76:2*   77:1    78:1    79:1
  80:1    81:1    82:1    83:1    84:1    85:2*   86:1    87:1    88:1    89:1
  90:1    91:1    92:1    93:1    94:1*   95:2*   96:0*   97:0*   98:0*   99:0*
 100:0*  101:0*  102:0*  103:1*  104:0*  105:0*  106:0*  107:0*  108:0*  109:0*
 110:0*  111:0*  112:1*  113:0*  114:0*  115:0*  116:0*  117:0*  118:0*  119:0*
 120:0*
```

Now we count the points and find that we have fixed the values of 82 points.

```
vsh:30> count
82 fixed values.
```

As all points have got a value, it is possible to classify the assignment:

```
vsh:31> veval
#0 ( 3, 5, 14, 41, 73 )
```

It tells us that the assignment has class no. 0 (ENDS), as required. The difference sequence is as desired, except for the last element, δ_4.

We use probe to increase the values. Each point is examined, and the value is increased by one if it won't make the assignment change class. Fixed values are not increased. The output lists a lot of points getting an increased value.

```
vsh:32> probe
Increasing value of point #50.
Increasing value of point #52.
[...]
Increasing value of point #93.
```

A second classification confirms that we have the desired difference sequence.

```
vsh:33> veval
#0 ( 3, 5, 14, 41, 100 )
```

Finally we print the assignment, and we list the subspaces of the different dimensions using ‘max’.

```
vsh:34> pvp
;
   0:0*    1:0*    2:0*    3:0*    4:1*    5:1*    6:1*    7:1*    8:1*    9:1*
  10:1*   11:1*   12:1*   13:2*   14:2*   15:2*   16:2*   17:2*   18:2*   19:2*
  20:2*   21:2*   22:2*   23:2*   24:2*   25:2*   26:2*   27:2*   28:2*   29:2*
  30:2*   31:2*   32:2*   33:2*   34:2*   35:2*   36:2*   37:2*   38:2*   39:2*
  40:2*   41:3*   42:2*   43:2*   44:2*   45:2*   46:2*   47:2*   48:2*   49:2*
  50:2    51:1*   52:2    53:2    54:2    55:2    56:2    57:2    58:2*   59:1*
  60:1*   61:1*   62:1*   63:1*   64:1*   65:1*   66:1*   67:2*   68:0*   69:1*
  70:2    71:2    72:2    73:2    74:2    75:2    76:2*   77:2    78:1*   79:2
  80:2    81:2    82:2    83:2    84:2    85:2*   86:2    87:1*   88:2    89:2
  90:2    91:2    92:2    93:2    94:1*   95:2*   96:0*   97:0*   98:0*   99:0*
 100:0*  101:0*  102:0*  103:1*  104:0*  105:0*  106:0*  107:0*  108:0*  109:0*
 110:0*  111:0*  112:1*  113:0*  114:0*  115:0*  116:0*  117:0*  118:0*  119:0*
 120:0*
vsh:35> max 0
[3]: 41
vsh:36> max 1
[8]: 955 930 902 871 846 818 734 703
678 650 619 594 566 510
vsh:37> max 2
[22]: 481
vsh:38> max 3
[63]: 0
vsh:39>
```

We note that there are several lines of maximum values, but only one point, one plane and one hyperplane, consistent with the analysis of Chapter 4. In the supplements of the thesis, a complete listing of subspaces is provided, as it is created by the ‘printall’ command in **vsh**(1).

10.2 A binary construction

We will try to construct an optimal, binary, five-dimensional ENDS, for $\delta_0 = 4$; i.e. we want the difference sequence $(4, 5, 9, 14, 25)$.

We recall the argument in Chapter 6. The names of the points correspond to numbers in the computer representation as follows:

$$A \sim 0, B \sim 1, \dots Y \sim 24, Z \sim 25, \Gamma \sim 26, \Delta \sim 27, \Theta \sim 28, \Lambda \sim 29, \Xi \sim 30.$$

We write a *valex* file with the following definition:

```
; Lemma 14: probe to get ENDS ( 4, 5, 9, 14, 25 )
  @fix=1
  2:2 8:2 9:2
  0:3 1:3 7:3 10:3
  5:2 6:2
  23:4
  @fix=0
  19:3 27:3
  15:1 16:2 17:2 18:1 20:2 21:2 22:2
  3:1 4:3
```

The first line assigns values to the points in ℓ, the second and third lines give values to the remaining points in $\mathcal{P}$. The fourth line gives a value to the point X. These values are completely determined by the theoretical analysis in Chapter 6, so they are fixed.

The last three lines give values to points in $\mathcal{H}$, and make sure that the value of $\mathcal{H}$ is higher than the value of any other hyperplane. The remaining points will be initialised with value of zero by command no. 3 below. Now we can run the following:

```
vsh:1> open sk1
vsh:2> next
New value assignment:
; Lemma 14: probe to get ENDS ( 4, 5, 9, 14, 25 )
End of assignment.
vsh:3> cset 4 0 0 0
vsh:4> veval
#0 ( 4, 5, 9, 13, 17 )
vsh:5> qprobe
vsh:6> veval
#0 ( 4, 5, 9, 14, 25 )
vsh:7> pvp
;
   0:3*    1:3*    2:2*    3:1*    4:3*    5:3     6:2*    7:3*    8:2*    9:2*
  10:3*   11:1    12:1    13:1    14:1    15:1*   16:2*   17:2*   18:1*   19:3*
  20:2*   21:2*   22:2*   23:4*   24:1    25:1    26:1    27:3*   28:1    29:0*
  30:0*
```

We get the desired difference sequence, and the value assignment given in Chapter 6. The supplements include a complete listing of subspaces, also for this binary construction.

Chapter 11

Man pages

NAME

vsh − interactive study of projective geometries and value assignments

SYNOPSIS

vsh [-q *q* **][-k** *k* **]** [**-e**|**-n**] [**-i**|**-b**]
vsh -v
vsh -h

DESCRIPTION

The intention of *vsh* is the study of weight hierarchies and chain properties of linear codes. We consider the columns of the generator matrix as projective points. The code is completely characterised when the number of occurrences of each projective point as a column is known. This number is known as the *value* of the point, and the code is characterised by the *value assignment,* giving a value to each point. (See REFERENCES below.)

The main task of this program is to *classify* value assignments, i.e. to compute the *difference sequence* and to determine the *class* according to the non-chain conditions.

vsh is a command driven tool, where values may be assigned and changed, and the assignment classified. The program can run either in batch or interactively on a tty. The commands are described below. Values may also be *fixed*. Several commands change the values of several points (e.g. **spset**) simultaneously. Unless explicitly stated, these commands will not change *fixed* values.

The geometry may be specified either with the **-k** and **-q** options, or with the **pg** command.

OPTIONS

−b Pretend to run in batch, i.e. do not print the prompt.

−e Echo commands.

−h Print a help message and exit.

−i Pretend to run interactively, i.e. print the prompt for each command expected.

−k *k* Specify the dimension of the geometry.

−n Do not echo commands.

−q *q* Specify the order of the geometry.

−v Print version information and exit.

COMMENTS

Comments start with a hash sign (#) and lasts until the end of the line. They can be written on separate lines or after the last argument of a command.

Verbose comments start with :, and are output verbatim. Verbose comments trailing a command, are echoed before the command is executed.

A file name or other string argument must either start with a letter and contain only letters, numbers and dashes (-); or it should be quoted in double quotes ("). No escape character is recognised.

Several commands requires an integer, *p*, representing a point, or a pair, *m p*, representing *m*-space number *p*. In either case the number(s) may be replaced by symbolic names as defined by **pdef**, **sdef**, **spdef**, or **line**.

class Print the class number and difference sequence, as previously computed.

clear Clear the value assignment and define a maximal set of projectively independent points, named *A, B,...* .

close Close the file used for reading value assignments.

count Count the number of unfixed points.

cset *m p v* **[** *c* **[** *f* **]]**
Give each point in *m*-space no. *p* the value *v*. Previously assigned values are overridden, unless *c*

is specified and equal to zero. If *f* is 1, assigned values are immediately fixed. If *f* is 0, they are unfixed. The default is not to change the fix tag of any point.

echo Turn on echoing of commands.

eval Recompute and print the class number and difference sequence. The format is like this:
#0 (4, 5, 9, 14, 25)
where #0 means class no. 0 (ENDS), and the difference sequence follows in parenthesis.

get *p_0 p_1* [.. *p_m*]
Show the space spanned by the given points, *p_0, p_1,...*, which are assumed to be projectively independent.

help Get help. (Not usefully implemented yet.)

log *file* Send output (appending) to *file* rather than to stdout. Error messages still goes to stderr.

max *m* Print the maximum value and the subspaces having that value, for dimension *m*.

next Read the next value assignment from a **valex**(4) file. The file must be opened with **open**, or otherwise stdin will be used instead. If EOF is reached, the file is closed and an error note is produced. The class number and difference sequence is recomputed,

noecho Turn off echoing of commands.

open *file* Open *file* for reading value assignments. The file format is described in **valex**(4).

pdef *name p*
Assign symbolic *name* to point no. *p*. The name must be a capital letter

pg *k q* Load PG(*k,q*). Be careful when switching between geometries. It may cause a memory leak.

printall *m*
Print all *m*-spaces with points and their values.

probe [*i*] Try to increase the value of each unfixed point by *i* if it does not affect the class. If *i* is 0 or not given, the increment is one, and any value which cannot be increased, is fixed.

pvp Print all points with their values, in the **valex**(4) file format.

qprobe *[i]*
'Quiet Probe' is a less verbose version of **probe** (see above).

quit Quit the program. Ctrl-D may be used as well.

sdef *name m p*
Assign symbolic *name* to *m*-space no. *p*. The name must be a lowercase letter

set *p v* Give point no. *p* value *v*. (Fixed values are changed as well.)

setfix *p v* Give point no. *p* value *v* and fix it. (Fixed values are changed as well.)

sh *command* **[***args***]**
Run shell command. The command is passed to the *system*(3) library function.

source *file*
Read commands from the given *file.*

space *m s*
Print *m*-space no. *s*.

span *p_0 p_1* [.. *p_m*]
Show the space spanned by the given points, *p_0, p_1,...*, which are assumed to be linearly independent, and print the value for each point.

spdef *name p_0 p_1 ...*
Assign symbolic name to the subspace spanned by the given points *p_0, p_1, ...* .

spfix *m p [f]*
If *f* is 1 or not given, fix all points in *m*-space no. *p*. If *f* is 0, the points are unfixed.

spset *m p v*
Give each point in *m*-space no. *p* the value *v,* if it currently has no value. Assigned values are immediately fixed.

sub *d_1 d_2 s*
Print all *d_1*-spaces being a sub- or superset of *d_2*-space no. *s.*

subval *m p*
Print the points in *m*-space no. *p* with values.

unlog Send output to stdout. This is of course, the opposite of the **log** command.

val *p* Print the point, *p,* with its value.

ver Print version information.

veval Recompute and print the class number and difference sequence. Unlike **eval**, **veval** also prints any relation which causes non-chain conditions to fail.

OUTPUT

Points

When a point is printed with its value, it takes the form *p*:*v*, where *p* is the number of the point and *v* is the value. A value may be succeeded by an asterisk (∗) to indicate that it is fixed. If dash (-) is given in place of the value, it means that the point has no value yet.

Subspaces

When a subspace is printed with values, the output is on this form:

1-space[536] (0): 15:2∗ 41:3∗ 67:2∗ 96:-

First the kind of subspace is stated (a 1-space in the example), then follows the number of the subspace in brackets and the value in parenthesis. The value is zero in the example, because it has not been computed. After the first colon, all the points in the subspace are listed with their values.

SEE ALSO

valex(4), *gen*(1), *vdg*(1)

BUGS

The error checking is poor. Be sure to pass valid arguments to the commands.

Apart from that, please enjoy the features...

REFERENCES

Hans Georg Schaathun: *Upper bounds on Weight Hierarchies of Extremal Non-Chain Codes* (Graduate thesis from the University of Bergen, Department of Informatics)

VERSION

gen-vsh version 5.0

AUTHOR

(C) Hans Georg Schaathun <georg@ii.uib.no>

NAME

gen − exhaustive search for linear codes with good weight hierarchies

SYNOPSIS

gen [-efstx] [**-k***k*] [**-q***q*] [**-a** *min*] [**-b** *n*] [**-c** *class*] [**−d** *d*] [**-m** *address*] [**-n** *nice*] [**-w** *nice*] [**−B** *file*] [**−O** *file*] [**−p** *file*] [**−S** *file*] [**−T** *file*] [*p:v . . .*]
gen -v
gen -h

DESCRIPTION

gen does an exhaustive search over all possible value assignments, and analyses them with respect to the *chain condition* and the *difference sequence.* One representative value assignment is recorded for each difference sequence and class. Given the **-f** option, *gen* accepts a sequence of arguments, *p*:*v,* assigning a fixed value, *v,* to the point, *p.*

The value assignments are analysed in a projective space and **gen** depends on an output file from **pgc**(1), which can be specified with the **−p** option. If no such file is given, **gen** will try and deduce the filename from the parameters *k* and *q*, defining the dimension and order of the geometry, respectively.

A status report can be requested at any time during an exhaustive search by sending SIGUSR1. This status report is the similar to the report generated regularly by the **−s** option, and it is written to the same file.

OPTIONS

−a *min* Create output and terminate after *min* minutes.

−B *file* Specify the name of the backup file.

−b *n* Make a backup for every *n* value assignments. *n* = 0 give no backups.

−c *class* Discard all value assignments except those of the given class.

−d *d* Set the maximum value of any point equal to *d*.

−e Discard all chained value assignments.

−f Accept arguments, predefining fixed values for some points.

−h Print a help message and exit.

−k *k* Specify the dimension of the geometry.

−m *address*
Set the e-mail address for output reports. (Default is username as specified by the $USER environment.)

−n *nice* Set nice value. (Default is 19.)

−O *filename*
Specify the basename for the binary output file.

−p *filename*
Specify the name of the file with the projective geometry.

−q *q* Specify the order of the geometry.

−S Specify a file name for the status reports.

−s Create a status report each time backup is taken.

−t Ignored.

−T *file* Read an initial collection of value assignments from *file*.

−v Print version information and exit.

−w *nice* Set nice value increment for child processes. (Default is 6.)

−x Don't report by e-mail when finished.

EXAMPLES

gen -eT *backup.*

Continue the work from the file named *backup.*

gen -e -d *2* **-k** *4* **-q** *3*

Find non-chain value assignments over PG(4,3) with delta_0 = 2.

SIGNALS

gen traps a lot of signals in an attempt to create a backup file before termination. These signals includes SIGINT, SIGTERM, SIGQUIT and SIGALRM. SIGSEGV, SIGBUS, SIGILL, SIGFPE and SIGSYS, which usually give a core dump are trapped in the same way, to create a backup and then call **abort**(2) to get the core dump.

SIGHUP and SIGCHLD are always ignored.

SIGUSR1 generates a status report.

ENVIRONMENT

USER The presumed user name and default e-mail address for output reports.

HOST The presumed host name for reports and log files.

SEE ALSO

vdg(1), *pgc*(1), *pgtest*(1), *vsh*(1)

REFERENCES

Hans Georg Schaathun: *Upper bounds on Weight Hierarchies of Extremal Non-Chain Codes* (Graduate thesis from the University of Bergen, Department of Informatics)

VERSION

gen-vsh version 5.0

AUTHOR

(C) Hans Georg Schaathun <georg@ii.uib.no>

NAME

pgc − generating incidence tables for a projective geometry

SYNOPSIS

pgc -q *q* **-k** *k* [**-o***file*] [**-d**]
pgc -v
pgc -h

DESCRIPTION

pgc constructs a projective geometry of dimension *k* over GF(*q*) and creates tables of all incidence relations between subspaces of different dimensions. The tables are written to a binary file for use with other programmes, e.g. gen**(1)**

OPTIONS

−d Print the number of subspaces of each dimension for the given parameters to stdout. Do not create the geometry.

−h Print a help message and exit.

−k *k* Specify the dimension of the geometry.

−o *file* Specify the name of the output file.

−q *q* Specify the order of the geometry.

−v Print version information and exit.

SEE ALSO

gen(1), *pgtest*(1)

VERSION

gen-vsh version 5.0

AUTHOR

(C) 1998: Hans Georg Schaathun <georg@ii.uib.no>

NAME

pgtest – computations for projective geometry

SYNOPSIS

pgtest -q *q* **-k** *k* [-c *file* | -n *n* | -C *n* | -g | -t]
pgtest -g|-t -q *q* **-k** *k a b*
pgtest -x | -y | -z | -w *infile outfile*
pgtest -q *q* **-k** *k* ®.B -b *n* ®.B -s ®.I basis . . .
pgtest -v
pgtest -h

DESCRIPTION

pgtest has different modes of operation to make different computations in a projective geometry. See OPTIONS below.

OPTIONS

–b *n* Forms a linear combination from the basis given in coordinate representation.

–c *str* Take *str* as point coordinates and get the point number.

–C *n* Take *n* as a point number and get the point coordinates.

–g Compute *a*∗*b* and the inverses of *a* and *b* over GF(*q*).

–h Print a help message and exit.

–k *k* Specify the dimension of the geometry.

–s Print the subspace spanned by the basis given in coordinate representation.

–q *q* Specify the order of the geometry.

-t Compute the theta-function for *a* and *b*.

-n *n* Compute the phi-function of *n*.

–w Copy the file describing the projective geometry to an ASCII file listing the points in each subspace.

–x Copy the file describing the projective geometry to a user readable print.

–y Copy the file describing the projective geometry.

–z Copy the file describing the projective geometry to an ASCII file listing for each *l*-space, the *(l-1)*-spaces contained in it.

–v Print version information and exit.

SEE ALSO

gen(1), *pgc*(1), *vsh*(1)

AUTHOR

(C) 1998: Hans Georg Schaathun <georg@ii.uib.no>

VERSION

gen-vsh version 5.0

NAME

pidcheck − check if a program is running, and restart if necessary.

SYNOPSIS

pidcheck [-v] *dir*

DESCRIPTION

The object of *pidcheck* is to regularly check if a certain process is running, and if necessary, to restart it. To use *pidcheck,* the main process (e.g. *gen*(1)) has to create an *active* file when it is started, to record the PID, the program name and command line options. It is suggested to run *pidcheck* as a cron job.

pidcheck does a *chdir (2)* to the given directory, and reads the *active* file. It uses *sigsend (3)* to check if the main process is running. If the main process has been killed, *pidcheck* looks for its backup file, as specified in the *active* file, and renames it before the main process is restarted.

OPTIONS

−v Verbose option. Print some diagnostic messages on stdout and stderr.

SEE ALSO

gen(1), *crontab*(1)

COMMENTS

The program is part of my graduate thesis. Technical documentation is to be found in this thesis.

AUTHOR

(C) 1998: Hans Georg Schaathun <georg@ii.uib.no>

VERSION

pidcheck version 1.0

NAME

vdg – handle output from *gen*(1)

SYNOPSIS

vdg -g -k *k* -q *q* [*infile*]
vdg -P *infile*
vdg -C *infile* [*outfile*]
vdg -v
vdg -h

DESCRIPTION

When **–g** is given, *vdg* takes a value assignment from *infile* and classifies it, printing the result on stdout.

With the **–C** option *vdg* translates *infile*, as created by *gen*(1), to a readable output consisting of one file per class found, listing the difference sequences and one example of a value assignment.

OPTIONS

–C Create an ASCII print from a binary output file from *gen*(1).

–g Read a value assignment from file and classify it. Default file is stdin, another file can be specified with **–i**. The file format is described in *valex*(4).

–h Print a help message and exit.

–P Print the order and dimension of the projective geometry used in the given file.

–v Print version information and exit.

SEE ALSO

gen(1), *valex*(4), *vsh*(1), *pgc*(1)

AUTHOR

(C) Hans Georg Schaathun <georg@ii.uib.no>

VERSION

gen-vsh version 5.0

NAME

valex − file format for value assignments

DESCRIPTION

The valex file defines one or more value assignments for programs like *vdg*(1) and *vsh*(1). The file must be pure 7bit ASCII.

Each assignment consists of a header line and a sequence of entries, separated by whitespace. The header line is a line starting with a semicolon (;) in the first position. The rest of the header line may be a descriptive comment for human readers.

There are two type of entries: value entries and keywords. No whitespace is allowed inside an entry. Each value entry assigns a value to one point. Keywords are described below.

The value entry may be a single number *v*, in which case the value *v* is assigned to the next point (or point no. 0, if it is the first value of the assignment), or it may be a point/value pair: *p*:*v*, assigning the value *v* to point no. *p*. Any value entry (in either format) may be suffixed with an asterisk to indicate that the value should be fixed.

Value assignments may be terminated with the **@end** keyword, but this is not necessary. If the assignment is not explicitly terminated, the value assignment ends at the end of file, or when the header line of new value assignment is encountered. After the **@end** keyword, only comments (see below) and certain keywords are allowed, until the header of a new value assignment is encountered or the file ends.

COMMENTS

Lines starting with a hash (#) mark may occur between value assignments and before the first value assignment. They are considered to be comments and ignored by the parser. To insert comments between two value assignments, the first one must be terminated with the **@end** keyword.

KEYWORDS

Keywords ending with an equal sign (=) expect an integer argument, which should follow with no intervening whitespace.

@end Mark the end of the assignment. (Allowed only within an assignment.)

@endfile
Mark the end of the file. Anything after this keyword is ignored.

@fix= Setting this to 1 causes all values of the current value assignment to be fixed when they are set. If it is set to 0, values are fixed if succeeded by an asterisk. (Allowed only within an assignment.)

@verbose=
The parser's verbose flag is turned on if this is set to 1. If it is set to 0, the verbose flag is turned off. The parser prints some diagnostic info to stdout/stderr when the verbose flag is turned on.

SEE ALSO

vdg(1),*vsh*(1)

AUTHOR

(C) Hans Georg Schaathun <georg@ii.uib.no>

VERSION

gen-vsh version 5.0

Part III

The Appendices

Appendix A

Index of Notations

- $\mathsf{GF}(q)$ – the Galois Field of order q.
- $\mathsf{PG}(n,q)$ – the Projective space of dimension n over $\mathsf{GF}(q)$.
- $\mathsf{PG}^{(m)}(k,q)$ – the set of m-spaces in $\mathsf{PG}(n,q)$ (page 8).
- $\nu(\mathbf{x})$ – the value assignment function (page 8).
- d_r – the rth generalized Hamming weight (page 7).
- $(\delta_0, \delta_1, \ldots \delta_k)$ – the difference sequence. (page 8).
- $\wp$, ℓ, $\mathcal{P}$ – a point, a line and a plane.
- $\mathcal{H}$ – a hyperplane in $\mathsf{PG}(4,q)$.
- Π_m – an m-space.
- $\langle \mathbf{x}_1, \mathbf{x}_2, \ldots \rangle$ – the space spanned by the vectors or points $\mathbf{x}_1, \mathbf{x}_2, \ldots$ (page 9).
- $\mathcal{C}$ – a $[n, k+1; q]$ code.
- $\chi(\mathcal{C})$ – the support of $\mathcal{C}$ (page 7).
- G – a generator matrix for $\mathcal{C}$.
- (C0) – the chain condition (page 10).
- (C$i.j$) – the subconditions of the chain condition (page 10).

- (N$i.j$) – the non-chain conditions (page 11).
- M_i – the set of i-spaces of maximum value (page 10).
- λ, λ_i – maximum values, i.e. $\lambda, \lambda_i \in \mathbb{N}_0$.
- k – the dimension of a projective geometry.
- $k+1$ – the dimension of a code.
- q – the order of a field.
- $\mathcal{L}, \mathcal{K}$ – sets of projective points.

Bibliography

[Beu80] A. Beutelspacher. Blocking sets and partial spreads in finite projective spaces. *Geometrica Dedicata*, 9:425–449, 1980.

[CK96] Wende Chen and Torleiv Kløve. The weight hierarchies of q-ary codes of dimension 4. *IEEE Trans. Inform. Theory*, 42(6):2265–2272, November 1996.

[CK97a] Wende Chen and Torleiv Kløve. Bounds on the weight hierarchies of extremal non-chain codes of dimension 4. *Applicable Algebra in Engineering, Communication and Computing*, 8:379–386, 1997.

[CK97b] Wende Chen and Torleiv Kløve. Bounds on the weight hierarchies of linear codes of dimension 4. *IEEE Trans. Inform. Theory*, 43:2047–2054, 1997.

[CK98a] Wende Chen and Torleiv Kløve. Classification of the weight hierarchies of binary linear codes of dimension 4. Technical Report 147, Department of Informatics, University of Bergen, March 1998.

[CK98b] Wende Chen and Torleiv Kløve. Weight hierarchies of linear codes of dimension 3. *Journal of Statistical Planning and Inference*, 1998.

[CK98c] Wende Chen and Torleiv Kløve. Weight hierarchies of linear codes satisfying the chain condition. *Designs, Codes and Cryptography*, 11, 1998.

[CK99] Wende Chen and Torleiv Kløve. Weight hierarchies of extremal non-chain binary codes of dimension 4. *IEEE Trans. Inform. Theory*, January 1999.

[Cla91] W. E. Clark. Blocking sets in finite projective spaces and uneven binary-codes. *Discrete Mathematics*, 94(1):65–68, 1991.

[CS99] Wende Chen and Xushun Sun. Weight hierarchies of 7-ary codes of dimension 3 and improved genetic algorithm. Third Shanghai Conference on the Sciences, Codes and Finite Geometry, May 1999. In Chinese, abstract in English.

[EK94] Sylvia Encheva and Torleiv Kløve. Codes satisfying the chain condition. *IEEE Trans. Inform. Theory*, 40:175–180, 1994.

[Hei97] U. Heim. Proper blocking sets in projective spaces. *Discrete Mathematics*, 174(1-3):167–176, 1997.

[Hir84] J. W. P. Hirschfeld. Linear codes and algebraic curves. *Research notes in mathematics*, 114:35–53, 1984.

[Hir98] J. W. P. Hirschfeld. *Projective Geometries over Finite Fields*. Oxford University Press, second edition, 1998.

[HKM77] T. Helleseth, T. Kløve, and J. Mykkeltveit. The weight distribution of irreducible cyclic codes with block lengths $n_1((q^l - 1/n)$. *Discrete Mathematics*, 18:179–211, 1977.

[HKY92] Tor Helleseth, Torleiv Kløve, and Øyvind Ytrehus. Generalized Hamming weights of linear codes. *IEEE Trans. Inform. Theory*, 38:1133–1140, 1992.

[HKYL95] Tor Helleseth, Torleiv Kløve, Øyvind Ytrehus, and Vladimir I. Levenshtein. Bounds on the minimum support weights. *IEEE Trans. Inform. Theory*, 41(2):432–440, March 1995.

[HS91] J. W. P. Hirschfeld and T. Szönyi. Construction of large arcs and blocking sets in finite planes. *European Journal of Combinatorics*, 12:499–511, 1991.

[Klø92] Torleiv Kløve. Support weight distribution of linear codes. *Discrete Mathematics*, 106/107:311–316, 1992.

[Klø93] Torleiv Kløve. Minimum support weights of binary codes. *IEEE Trans. Inform. Theory*, 39(2), March 1993.

[KTFL93] T Kasami, T Takata, T Fujiwara, and S Lin. On the optimum bit orders with respect to the state complexity of trellis diagrams for binary linear codes. *IEEE Trans. Inform. Theory*, 39(1):242–245, 1993.

[OW84] L. H. Ozarow and A. D. Wyner. Wire-tap channel II. *AT&T Bell Laboratories Technical Journal*, 63(10):2135–2157, December 1984.

[Wei91] Victor K. Wei. Generalized Hamming weights for linear codes. *IEEE Trans. Inform. Theory*, 37:1412–1418, 1991.

[WY93] Victor K. Wei and Kyeongcheol Yang. On the generalized Hamming weights of product codes. *IEEE Trans. Inform. Theory*, 39:1709–1713, September 1993.