Fifteenth International Workshop on Algebraic and Combinatorial Coding Theory
June 18-24, 2016, Albena, Bulgaria *pp. 319–324*

Completeness of the 95256-cap in $PG(12, 4)$ [1]

Daniele Bartoli, Stefano Marcugini, Alfredo Milani, Fernanda Pambianco
{daniele.bartoli, stefano.marcugini, alfredo.milani, fernanda.pambianco}@unipg.it
Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Via Vanvitelli 1, Perugia, 06123, Italy

Abstract. We describe an algorithm for testing the completeness of caps in $\mathrm{PG}(r, q)$, q even. It allowed us to check that the 95256-cap in $\mathrm{PG}(12, 4)$ recently found by Fu el al. (see [7]) is complete.

1 Introduction

Let $\mathrm{PG}(r, q)$ be the r-dimensional projective space over the Galois field $\mathbb{F}_q$. An n-cap in $\mathrm{PG}(r, q)$ is a set of points no three of which are collinear. An n-cap in $\mathrm{PG}(r, q)$ is called complete if it is not contained in an $(n + 1)$-cap in $\mathrm{PG}(r, q)$; see [8].

The points of a complete n-cap in $\mathrm{PG}(r-1, q)$ can be treated as columns of a parity check matrix of an $[n, n - r, 4]_q$ linear code with the exceptions of the complete 5-cap in $\mathrm{PG}(3, 2)$ and the complete 11-cap in $\mathrm{PG}(4, 3)$ corresponding to the binary $[5, 1, 5]_2$ code and to the Golay $[11, 6, 5]_3$ code respectively.

An n-cap in $\mathrm{PG}(r, q)$ of maximal size is called a maximal cap in $\mathrm{PG}(r, q)$. A classical problem on caps is to determine the maximal size of complete caps in $\mathrm{PG}(r, q)$. This is also known as the packing problem; see [9]. Denote the size of a maximal cap in $\mathrm{PG}(r, q)$ as $m_2(r, q)$, and the largest size of a known complete cap as $\overline{m}_2(r, q)$.

Of particular interest is the case $q = 4$, due the connection with quantum error correction established in [6], where a class of quantum codes, the quantum stabilizer codes, is described in terms of certain additive quaternary codes.

Additive quaternary codes are defined over $\mathbb{F}_4$ but are linear over $\mathbb{F}_2$. If we restrict considering quaternary quantum codes that are indeed $\mathbb{F}_4$-linear then we have the following definition; see [1, 2]:

[1] This research was supported in part by Ministry for Education, University and Research of Italy (MIUR) and by the Italian National Group for Algebraic and Geometric Structures and their Applications (GNSAGA - INDAM).

Definition 1. *A linear quaternary quantum stabilizer code is a subspace $\mathcal{C} \subset \mathbb{F}_4^n$ such that $\mathcal{C} \subset \mathcal{C}^{\perp_H}$, where duality is with respect to the Hermitian inner product.*

Here the Hermitian inner product of $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$ is $\langle x, y\rangle = \sum_{i=1}^{n} x_i \overline{y}_i$, where $\overline{y} = y^2$. The reason for this definition is that a linear quaternary quantum stabilizer code $\mathcal{C}$ of length n, dimension r and dual distance $\geq d$ (equivalently: of strength $> d$) allows the construction of a pure quantum stabilizer code $[[n, n-2r, d]]_4$; see [4, Theorem 1].

A pure quantum code $[[n, n-2r, 4]]$ which is linear over $\mathbb{F}_4$ is obtained from a cap satisfying certain conditions; see [5, Theorem 2.8]:

Definition 2. *A cap $\mathscr{C}$ in $PG(r-1, 4)$ is a quantum cap if it is not contained in a hyperplane and if it satisfies the following equivalent conditions:*

- *each hyperplane meets the cap in the same parity as the cardinality of the cap;*
- *the corresponding quaternary $[n, r]_4$-code has all its weights even;*
- *the corresponding quaternary $[n, r]_4$-code is self-orthogonal with respect to the Hermitian inner product.*

Theorem 3. *The following are equivalent:*

- *A pure stabilizer quantum code $[[n, n-2r, 4]]$ which is linear over $\mathbb{F}_4$.*
- *A quantum n-cap in $PG(r-1, 4)$.*

The value of $m_2(r, 4)$ is known for $k \leq 4$: $m_2(2, 4) = 6$, $m_2(3, 4) = 17$, and $m_2(4, 4) = 41$.; see [3].

In [7] it is proved that $\overline{m}_2(8, 4) = 2136$, $\overline{m}_2(9, 4) = 5124$, $\overline{m}_2(10, 4) = 15840$, $\overline{m}_2(11, 4) = 36084$ and they also give a 95256-cap in $PG(12, 4)$.

Their results have been obtained by computer-supported recursive constructions. They also present an algorithm for checking completeness of a cap. This algorithm allowed checking the completeness of the caps for $k \leq 11$, but it is too computationally expensive for the case $k = 12$. As they wrote: "But as for checking completeness of larger caps in $PG(r, 4)$, $r \geq 12$, new algorithms are needed."; see [7, Section 5]. We propose a new fast algorithm that allowed to face also this case: we verified that the 95256-cap in $PG(12, 4)$ is complete, so $\overline{m}_2(12, 4) = 95256$. Our algorithm is based on a compact representation of the points of $\mathrm{PG}(r, q)$, q even, and on minimizing the computational costs of the operations more often performed during the check of the completeness of the cap.

Section 2 describes the algorithm and applies it in $PG(12, 4)$. Section 3 contains the generalization of the algorithm to other even values of q and other dimensions.

Table 1: The time and space cost of the algorithm of [7]

Size of cap	2136	5124	15840	36084
Time	7805	46244	428029	2261301

2 A new algorithm for checking completeness of a cap

In [7] an algorithm for checking completeness of a cap $\mathscr{C}$ in $PG(r,4)$ is presented. It is based on a bijective map ϕ between points in $PG(r,4)$ and a subset $T(r)$ of the positive integer set $\mathbb{N}$:

$$\phi : PG(r,4) \to T(r),$$
$$\phi : P \mapsto \phi(P),$$

where

$$P = (x_0, x_1, ..., x_r)^T,$$
$$\phi(P) = 4^r x_0 + 4^{r-1} x_1 + \cdots + 4x^{r-1} + x^r.$$

It can be easily seen that a cap is complete if and only if each point P of $\mathrm{PG}(r,q)$ not belonging to the cap lies on a secant line of the cap. In this case we say that P is covered.

To keep track of the covering of the points, a vector U of size $|T(r)|$ is used. Initially all elements of U are set to be 1.

Then all pairs of points of the cap are considered. For each pair of points (P_i, P_j) the three other points belonging to the line through P_i, P_j are computed. To do this all linear combination $Q = \alpha P_1 + P_2, \alpha \in \mathbb{F}_4 \backslash \{0\}$ are computed. The point Q is normalized, choosing a representation with the leftmost non-zero coordinate equal to 1. Finally the position $\phi(Q)$ of U is set to 0. The process continues until all elements of U became 0 or all pairs of points of the cap have been considered.

At the end the cap is complete if and only if all elements of U are 0. Table 1 reports the time cost of the algorithm using an Intel(R) Xeon(R) CPU E5504 @ 2.00GHz; see [7, Table 1]. However, the paper does not mention the unit of time used in Table 1.

We devised a new algorithm for checking the completeness of a cap in $PG(r,4)$ choosing a representation that optimizes the computational cost of the main operations of the previous algorithm: the computation of $Q = \alpha P_1 + P_2$ and the normalization of a point Q.

Let be $\mathbb{F}_4 = \{0, 1, \omega, \overline{\omega}\}$, where $\omega^2 = \overline{\omega}$, $\overline{\omega} = \omega + 1$, and $\omega^3 = 1$. If we define a representation function $\rho : \mathbb{F}_4 \to \mathbb{N}$ as in the following:

$$\rho(0) \to 0,\ \rho(1) \to 1,\ \rho(\omega) = 2,\ \rho(\overline{\omega}) = 3,$$

then we have

$$a + b = \rho(a)\hat{}\rho(b), \quad a, b \in \mathbb{F}_4,$$

where $\hat{}$ is the bitwise exclusive or operator.

Moreover, if $P = (x_0, x_1, ..., x_r)^T$ then the binary representation of $\phi(P)$ is $\rho(x_0)\rho(x_1)\dots\rho(x_r)$.

It means that if $Q = P_1 + P_2$ then $\phi(Q) = \phi(P_1)\hat{}\phi(P_2)$.

This allows the computation of the sum of two points of $PG(r, 4)$ by one integer operation.

The multiplication of one point P by a scalar is applied only to the points of the cap. It can be pre-computed before the beginning of the check for completeness, so at the cost of having a data structure of size $3|\mathscr{C}|$ all multiplications by a scalar are avoided.

The other expensive operation of the algorithm in [7] is the normalization of a point P. It should be to computed each time $Q = \alpha P_1 + P_2$, $P_1, P_2 \in \mathscr{C}$ is computed, i.e. $3/2|\mathscr{C}|^2$ times. We propose a trade-off between computational time and memory space: we use a vector U of size $3|PG(r, 4)|$ to keep trace of the fact that a point Q is covered by $\mathscr{C}$ or not; initially all elements of U are set equal to 0. When $Q = \alpha P_1 + P_2$ is computed, then the element $\phi(Q)$ is set equal to 1 without before normalizing Q. In this way all the $3/2|\mathscr{C}|^2$ normalization operations are avoided. At the end, when the covering of all points of $PG(r, 4)$ is tested, first the normalized form of a point Q is tested checking the element of U of position $\phi(Q)$; if it is not covered also $\phi(\omega Q)$ and $\phi(\overline{\omega} Q)$ are checked: if any of $\phi(Q), \phi(\omega Q), \phi(\overline{\omega} Q)$ is equal to 1, then Q is covered.

Let be $n = |\mathscr{C}|$, $m = |PG(r, q)|$, $i =$ the size of an integer, $c =$ the size of a character. The total cost of our algorithm is:

$$\text{space: } 3n \cdot i + 3m \cdot c + c_1;$$
$$\text{time: } 3c_2 n + 3/2 c_2 n^2 + 3c_3 m \cdot c + c_4;$$

where $c_1, \dots, c_4$ are constants.

The algorithm has been implemented in C language.

Table 2 reports the time and space cost of the algorithm using an Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz; space is measured in megabyte, while time is measured in milliseconds.

We tested the completeness of the caps presented in [7] in $PG(r, q)$, $r = 8, \dots, 12$. Note that for constructing the 5124-cap in $PG(9, 4)$ and the 36084-cap in $PG(11, 4)$ we were not able to obtain a cap following the selection of columns suggested in [7, Section 3.2]. For the 5124-cap we had to exclude the vectors for $j \in \{2, 14, 24, 25\}$ instead of $j \in \{2, 14, 15, 24\}$ as suggested in the paper, whereas for the 36084-cap we had to exclude the vectors for $i \in \{1, \dots, 16, 271\}$ instead of $i \in \{1, 256, \dots, 271\}$ as suggested in the paper.

3 Generalization of the algorithm

In the previous section we applied our algorithm in $PG(12, 4)$. The key idea, choosing a representation for the elements of $\mathbb{F}_q$ and the points of $\mathrm{PG}(r, q)$ that

Table 2: The time and space cost of the new algorithm

Size of cap	2136	5124	15840	36084	95256
Space (Mb)	5	18	76	324	1382
Time (Milliseconds)	32	78	707	3574	98321

minimize the computational cost of the operations most often performed during the test of completeness of a cap, can be applied for every even q.

When testing the completeness of a cap $\mathscr{C}$ the value $Q = \alpha P_1 + P_2$, $P_1, P_2 \in \mathscr{C}$, $\alpha \in \mathbb{F}_q \backslash \{0\}$ has to be computed.

To avoid to compute the same value αP several times, it is convenient to compute it at the beginning of the algorithm and store the results. Therefore the main operation to compute remains the sum between two vectors representing points of $\mathrm{PG}(r, q)$.

When considering a representation of $\mathbb{F}_{2^k}$, we can either choose a form that facilitate the computation of multiplication (we see the non-zero elements of $\mathbb{F}_{2^k}$ as powers of the primitive element) or can choose a form that facilitate the computation of addition (we see the the elements of $\mathbb{F}_q$ as polynomials of $\mathbb{F}_2[X]$ of degree less than k; addition is defined in the natural way, whereas multiplication is defined modulo a fixed irreducible polynomial of degree k).

We choose the latter representation and define $\rho : \mathbb{F}_{2^k} \to \{0, \ldots, 2^k - 1\}$ as $\rho : p(x) \mapsto p(2)$. We have that $\rho(p(x) + q(x)) = \rho(p(x)) \hat{} \rho(q(x))$, where $\hat{}$ is the bitwise exclusive or. It means that in this representation addition on $\mathbb{F}_{2^k}$ reduces to one bitwise arithmetic operation on integers.

Moreover we can define a representation of the points of $PG(r, 2^k)$ in the following way:

$$\phi : PG(r, 2^k) \to \mathbb{N},$$
$$\phi : P \mapsto \phi(P),$$

where

$$P = (x_0, x_1, ..., x_r)^T,$$
$$\phi(P) = (2^k)^r \rho(x_0) + (2^k)^{r-1} \rho(x_1) + \cdots + (2^k)\rho(x^{r-1}) + \rho(x^r).$$

In this way a point P of $PG(r, 2^k)$ is represented by an integer n. If we consider the binary representation of n, the coordinate x_i is represented by the bits of n in position $(r-i)k+1 \ldots (r-i+1)k$ that are the binary representation $\rho(x_i)$. To compute $\phi(Q)$, $Q = P_1 + P_2$ it is sufficient computing $\phi(P_1) \hat{} \phi(P_2)$, just one bitwise arithmetic operation. In a real implementation, usually an (unsigned) integer has a 32 bit representation, so $\phi(P)$ can be represented by a single integer if $kr \leq 32$, otherwise more integers are needed.

Our algorithm trades computational time for memory space. Let be $n = |\mathscr{C}|$, $m = |PG(r, 2^k)|$, we need $n(2^k - 1)$ integers to represent αP, $P \in \mathscr{C}$, $\alpha \in \mathbb{F}_{2^k} \backslash \{0\}$ and $m(2^k - 1)$ booleans to represent the fact that αP, $P \in PG(r, 2^k)$, $\alpha \in \mathbb{F}_{2^k} \backslash \{0\}$ is saturated or not. As $n < m$, the latter value is more relevant. If the memory requested by the algorithm is too big, then memory space can be traded for computational time. For example the set of points of $PG(r, 2^k)$ can be divided into s subsets small enough to be represented. Than the test for completeness can be repeated s times; each time the completeness of the points of one subset is tested. Note that the computations for the different subsets are independent, so they can be performed in parallel. This is a form of parallelism based on the splitting of data: it is simple and effective.

References

[1] D. Bartoli, J. Bierbrauer, S. Marcugini, F. Pambianco, *Geometric Constructions of quantum codes*. A.A. Bruen and D. Wehlau, Error-Correcting Codes, Cryptography and Finite Geometries **523** (2010), 149–154.

[2] J. Bierbrauer, D. Bartoli, G. Faina, S. Marcugini, F. Pambianco, Y. Edel, *The structure of quaternary quantum caps*, Des. Codes Cryptogr. **72** (2014), 733–747.

[3] J. Bierbrauer, Y. Edel, 41 *is the largest size of a cap in* $PG(4, 4)$, Des. Codes Cryptogr. **59** (1999), 151–160.

[4] J. Bierbrauer, Y. Edel, *Quantum twisted codes*, J. Combin. Des. **8** (2000), 174–188.

[5] J. Bierbrauer, G. Faina, M. Giulietti, S. Marcugini, F. Pambianco, *The geometry of quantum codes*, Innov. Incidence Geom. **6** (2009), 53–71.

[6] R. Calderbank, E. M. Rains, P. W. Shor, N. J. A. Sloane, *Quantum error correction via codes over* $GF(4)$, IEEE Trans. Inform. Theory **44** (1998), 1369–1387.

[7] Q. Fu, R. Li, L. Guo, G. Xu, *Large caps in projective space* $PG(r, 4)$, Finite Fields Appl. **35** (2015), 231–246.

[8] J.W.P. Hirschfeld, Finite Projective Spaces of Three Dimensions, Oxford University Press, Oxford, 1985.

[9] J.W.P. Hirschfeld, L. Storme, *The packing problem in statistics, coding theory, and finite projective spaces: update 2001*, in: A. Blokhuis, J.W.P. Hirschfeld, D. Jungnickel, J.A. Thas (Eds.), Finite Geometries, Proceedings of the Fourth Isle of Thorns Conference, in: Dev. Math., vol.3, Kluwer Academic Publishers, Boston, 2000, pp. 201-246.