LINCODE – COMPUTER CLASSIFICATION OF LINEAR CODES

SASCHA KURZ

ABSTRACT. We present an algorithm for the classification of linear codes over finite fields, based on lattice point enumeration. We validate a correct implementation of our algorithm with known classification results from the literature, which we partially extend to larger ranges of parameters.

Keywords: linear code, classification, enumeration, code equivalence, lattice point enumeration
ACM: E.4, G.2, G.4

1. INTRODUCTION

Linear codes play a central role in coding theory for several reasons. They permit a compact representation via generator matrices as well as efficient coding and decoding algorithms. Also multisets of points in the projective space $\operatorname{PG}(k-1, \mathbb{F}_q)$ of cardinality n correspond to linear $[n, k]_q$ codes, see e.g. [7]. So, let q be a prime power and $\mathbb{F}_q$ be the field of order q. A q-ary linear code of length n, dimension k, and minimum (Hamming) distance at least d is called an $[n, k, d]_q$ code. If we do not want to specify the minimum distance d, then we also speak of an $[n, k]_q$ code or of an $[n, k, \{w_1, \dots, w_l\}]_q$ if the non-zero codewords have weights in $\{w_1, \dots, w_k\}$. If for the binary case $q = 2$ all weights w_i are divisible by 2, we also speak of an even code. We can also look at those codes as k-dimensional subspaces of the Hamming space $\mathbb{F}_q^n$. An $[n, k]_q$ code can be represented by a generator matrix $G \in \mathbb{F}_q^{k\times n}$ whose row space gives the set of all q^k codewords of the code. In the remaining part of the paper we always assume that the length n of a given linear code equals its effective length, i.e., for every coordinate there exists a codeword with a non-zero entry in that coordinate. While a generator matrix gives a compact representation of a linear code it is far from being unique. Special generator matrices are so-called systematic generator matrices, which contain a $k \times k$ unit matrix in the first k columns. If we apply row operations of the Gaussian elimination algorithm onto a generator matrix we do not change the code itself but just its representation via a generator matrix. Also column permutations or applying field automorphisms do not change the essential properties of a linear code. Applying all these transformations, we can easily see that each $[n, k]_q$ code admits an isomorphic code with a systematic generator matrix. Already in 1960 Slepian has enumerated binary linear codes for small parameters up to isomorphism (or isometry) [21]. The general classification problem for $[n, k]_q$ codes has not lost its significance since then, see e.g. [2]. In [11] all optimal binary linear $[n, k, d]_2$ codes up to length 30 have been completely classified, where in this context optimal means that no $[n-1, k, d]_2$, $[n+1, k+1, d]_2$, or $[n+1, k, d+1]_2$ code exists. Classification algorithms for linear codes have been presented in [18], see also [12, Section 7.3]. A software package `Q-Extension` is publicly available, see [4] for a description. The further development to a new version `QextNewEdition` was recently presented in [5].

The aim of this paper is to present an algorithmic variant for the classification problem for linear codes. It is implemented in an evolving software package `LinCode`. As the implementation of such a software is a delicate issue, we exemplarily verify several classification results from the literature and partially extend them. That the algorithm is well suited for parallelization is demonstrated e.g. by classifying the $1\,656\,768\,624$ even $[21, 8, 6]_2$ codes. As mentioned in [18],

2000 *Mathematics Subject Classification.* Primary 94B05; Secondary 05E20.

one motivation for the exhaustive enumeration of linear codes with some specific parameters is that afterwards the resulting codes can be easily checked for further properties. Exemplarily we do here so for the number of minimal codewords of a linear code, see Subsection 3.1.

The remaining part of the paper is organized as follows. In Section 2 we present the details and the theoretical foundation of our algorithm. Numerical enumeration and classification results for linear codes are listed in Section 3. Finally, we draw a brief conclusion in Section 4.

2. EXTENDING LINEAR CODES

As mentioned in the introduction, we represent an $[n,k]_q$ code by a systematic generator matrix $G \in \mathbb{F}_q^{k\times n}$, i.e., G is of the form $G = (I_k|R)$, where I_k is the $k \times k$ unit matrix and $R \in \mathbb{F}_q^{k\times(n-k)}$. While this representation is quite compact, it nevertheless can cause serious storage requirements if the number of codes get large. Storing all generator matrices of the even $[21,8,6]_2$ codes, mentioned in the introduction, needs more than $2.78 \cdot 10^{11}$ bits ($1.72 \cdot 10^{11}$ bits, if the unit matrices are omitted).

Our general strategy to enumerate linear codes is to start from a (systematic) generator matrix G of a code and to extend G to a generator matrix G' of a "larger" code. Of course, there are several choices how the shapes of the matrices G and G' can be chosen, see e.g. [5, 18] for some variants. Here we assume the form

$$G' = \begin{pmatrix} I_k & 0\dots0 & R \\ 0 & \underbrace{1\dots1}_{r} & \star \end{pmatrix}$$

where $G = (I_k|R)$ and $r \ge 1$. Note that if G is a systematic generator matrix of an $[n,k]_q$ code, then G' is a systematic generator matrix of an $[n+r,k+1]_q$ code. Typically there will be several choices for the $\star$s and some of these can lead to isomorphic codes. So, in any case we will have to face the problem that we are given a set $\mathcal{C}$ of linear codes and we have to sift out all isomorphic copies. In the literature several variants of definitions of isomorphic codes can be found. Here we stick to [2, Definition 1.4.3] of linearly isometric codes, i.e., linearity and the Hamming distance between pairs of codewords are preserved. This assumption boils down to permutations of the coordinates and applying field automorphisms, see e.g. [2, Section 1.4] for the details. A classical approach for this problem is to reformulate the linear code as a graph, see [3], and then to compare canonical forms of graphs using the software package `Nauty` [17], see also [18]. In our software we use the implementation from `Q-Extension` as well as another direct algorithmic approach implemented in the software `CodeCan` [8]. In our software, we can switch between these two tools to sift out isomorphic copies and we plan to implement further variants. The reason to choose two different implementations for the same task is to independently validate results.[1]

It remains to solve the extension problem from a given generator matrix G to all possible extension candidates G'. To this end we utilize the geometric description of the linear code generated by G as a multiset $\mathcal{M}$ of points in $\mathrm{PG}(k-1,\mathbb{F}_q)$, where

$$\mathcal{M} = \left\{\left\{\langle g^i\rangle \,:\, 1 \le i \le n\right\}\right\},^{2}$$

g^i are the n columns of G, and $\langle v\rangle$ denotes the row span of a column vector v. In general, the 1-dimensional subspaces of $\mathbb{F}_q^k$ are the points of $\mathrm{PG}(k-1,\mathbb{F}_q)$. The $(k-1)$-dimensional subspaces

[1]Moreover, there are some technical limitations when applying `Q-Tools` from `Q-Extension` to either many codes or codes with a huge automorphism group. Also the field size is restricted to be at most 4. As far as we know, the new version `QextNewEdition` does not have such limitations.

[2]We use the notation $\{\{\cdot\}\}$ to emphasize that we are dealing with multisets and not ordinary sets. A more precise way to deal with a multiset $\mathcal{M}$ in $\mathrm{PG}(k-1,\mathbb{F}_q)$ is to use a characteristic function χ which maps each point P of $\mathrm{PG}(k-1,\mathbb{F}_q)$ to an integer, which is the number of occurences of P in $\mathcal{M}$. With this, the cardinality $\#\mathcal{M}$ can be writen as the sum over $m(P)$ for all points P of $\mathrm{PG}(k-1,\mathbb{F}_q)$.

of $\mathbb{F}_q^k$ are called the hyperplanes of $\operatorname{PG}(k-1,\mathbb{F}_q)$. By $m(P)$ we denote the multiplicity of a point $P \in \mathcal{M}$. We also say that a column g^i of the generator matrix has multiplicity $m(P)$, where $P = \langle g^i \rangle$ is the corresponding point, noting that the counted columns can differ by a scalar factor. Similarly, let $\mathcal{M}'$ denote the multiset of points in $\operatorname{PG}((k+1)-1,\mathbb{F}_q)$ that corresponds to the code generated by the generator matrix G'. Note that our notion of isomorphic linear codes goes in line with the notion of isomorphic multisets of points in projective spaces, see [7]. Counting column multiplicities indeed partially takes away the inherent symmetry of the generator matrix of a linear code, i.e., the ordering of the columns and multiplications of columns with non-zero field elements is not specified explicitly any more. If the column multiplicity of every column is exactly one, then the code is called projective.

Our aim is to reformulate the extension problem $G \to G'$ as an enumeration problem of integral points in a polyhedron. Let $W \subseteq \{i\Delta : a \le i \le b\} \subseteq \mathbb{N}_{\ge 1}$ be a set of feasible weights for the non-zero codewords, where we assume $1 \le a \le b$ and $\Delta \ge 1$.[3] Linear codes where all weights of the codewords are divisible by Δ are called Δ-divisible and introduced by Ward, see e.g. [22, 23].

The non-zero codewords of the code generated by the generator matrix G correspond to the non-trivial linear combinations of the rows of G (over $\mathbb{F}_q$). In the geometric setting, i.e., where an $[n,k]_q$ code C is represented by a multiset $\mathcal{M}$, each non-zero codeword $c \in C$ corresponds to a hyperplane H of the projective space $\operatorname{PG}(k-1,\mathbb{F}_q)$. (More precisely, $\mathbb{F}_q^* \cdot c$ is in bijection to H, where $\mathbb{F}_q^* = \mathbb{F}_q \backslash \{0\}$.) With this, the Hamming weight of a codeword c is given by

$$n - \sum_{P \in \operatorname{PG}(k-1,\mathbb{F}_q)\,:\, P \in \mathcal{M},\, P \le H} m(P),$$

see [7]. By $\mathcal{P}_k$ we denote the set of points of $\operatorname{PG}(k-1,\mathbb{F}_q)$ and by $\mathcal{H}_k$ the set of hyperplanes.

Lemma 2.1. *Let G be a systematic generator matrix of an $[n,k]_q$ code C whose non-zero weights are contained in $\{i\Delta : a \le i \le b\} \subseteq \mathbb{N}_{\ge 1}$. By $c(P)$ we denote the number of columns of G whose row span equals P for all points P of $\operatorname{PG}(k-1,\mathbb{F}_q)$ and set $c(\mathbf{0}) = r$ for some integer $r \ge 1$. With this let $\mathcal{S}(G)$ be the set of feasible solutions of*

$$\Delta y_H + \sum_{P \in \mathcal{P}_{k+1}\,:\, P \le H} x_P = n - a\Delta \qquad \forall H \in \mathcal{H}_{k+1} \tag{2.1}$$

$$\sum_{q \in \mathbb{F}_q} x_{\langle (u|q) \rangle} = c(\langle u \rangle) \qquad \forall \langle u \rangle \in \mathcal{P}_k \cup \{\mathbf{0}\} \tag{2.2}$$

$$x_{\langle e_i \rangle} \ge 1 \qquad \forall 1 \le i \le k+1 \tag{2.3}$$

$$x_P \in \mathbb{N} \qquad \forall P \in \mathcal{P}_{k+1} \tag{2.4}$$

$$y_H \in \{0,\dots,b-a\} \qquad \forall H \in \mathcal{H}_{k+1}, \tag{2.5}$$

where e_i denotes the ith unit vector in $\mathbb{F}_q^{k+1}$. Then, for every systematic generator matrix G' of an $[n+r,k+1]_q$ code C' whose first k rows coincide with G and whose weights of its non-zero codewords are contained in $\{i\Delta : a \le i \le b\}$, we have a solution $(x,y) \in \mathcal{S}(G)$ such that G' has exactly x_P columns whose row span is equal to P for each $P \in \mathcal{P}_{k+1}$.

Proof. Let such a systematic generator matrix G' be given and x_P denote the number of columns of G' whose row span is equal to P for all points $P \in \mathcal{P}_{k+1}$. Since G' is systematic, Equation (2.3) is satisfied. As G' arises by appending a row to G, also Equation (2.2) is satisfied for all $P \in \mathcal{P}_k$. For $P = \mathbf{0}$ Equation (2.2) is just the specification of r. Obviously, the x_P are non-negative

[3]Choosing $\Delta = 1$ such a representation is always possible. Moreover, in many applications we can choose $\Delta > 1$ quite naturally. I.e., for optimal binary linear $[n,k,d]_2$ codes with even minimum distance d, i.e., those with maximum possible d, we can always assume that there exists an *even* code, i.e., a code where all weights are divisible by 2.

integers. The conditions (2.1) and (2.5) correspond to the restriction that the weights are contained in $\{i\Delta : a \le i \le b\}$. □

We remark that some of the constraints (2.1) are automatically satisfied since the subcode C of C' satisfies all constraints on the weights. If there are further forbidden weights in $\{i\Delta : a \le i \le b\}$ then, one may also use the approach of Lemma 2.1, but has to filter out the integer solutions that correspond to codes with forbidden weights. Another application of this first generate, then filter strategy is to remove some of the constraints (2.1), which speeds up, at least some, lattice point enumeration algorithms. In our implementation we use `Solvediophant` [24], which is based on the LLL algorithm [15], to enumerate the integral points of the polyhedron from Lemma 2.1.

Noting that each $[n', k', W]_q$ code, where $W \subseteq \mathbb{N}$ is a set of weights, can indeed be obtained by extending[4] all possible $[n'-r, k'-1, W]_q$ codes via Lemma 2.1, where $1 \le r \le n'-k'+1$, already gives an algorithm for enumerating and classifying $[n', k', W]_q$ codes. (For $k' = 1$ there exists a unique code for each weight $w \in W$, which admits a generator matrix consisting of w ones.) However, the number of codes C with generator matrix G that yield the same $[n', k', W]_q$ code C' with generator matrix G' can grow exponentially with k'. We can limit this growth a bit by studying the effect of the extension operation and its reverse on some code invariants.

Lemma 2.2. *Let C' be an $[n', k', W]_q$ code with generator matrix G'. If G' contains a column g' of multiplicity $r \ge 1$, then there exists a generator matrix G of an $[n'-r, k'-1, W]_q$ code C such that the extension of G via Lemma 2.1 yields at least one code that is isomorphic to C'. Moreover, if Λ is the maximum column multiplicity of G', without counting the columns whose row span equals $\langle g' \rangle$, then the maximum column multiplicity of G is at least Λ.*

Proof. Consider a transform $\tilde{G}$ of G' such that the column g' of G' is turned into the jth unit vector e_j for some integer $1 \le j \le k'$. Of course also $\tilde{G}$ is a generator matrix of C'. Now let $\hat{G}$ be the $(k'-1) \times (n'-r)$-matrix over $\mathbb{F}_q$ that arises from $\tilde{G}$ after removing the r occurrences of the columns with row span $\langle e_j \rangle$ and additionally removing the jth row. Note that the non-zero weights of the linear code generated by $\hat{G}$ are also contained in W. If G is a systematic generator matrix of the the linear code C generated by $\hat{G}$, then Lemma 2.1 applied to G with the chosen parameter r yields especially a linear code with generator matrix G' as a solution. By construction the effective length of C is indeed $n'-r$. Finally, note that removing a row from a generator matrix does not decrease column multiplicities. □

Corollary 2.3. *Let C' be an $[n', k', W]_q$ code with generator matrix G' and minimum column multiplicity r. Then there exists a generator matrix G of an $[n'-r, k'-1, W]_q$ code C with minimum column multiplicity at least r such that the extension of G via Lemma 2.1 yields at least one code that is isomorphic C'.*

Corollary 2.3 has multiple algorithmic implications. If we want to classify all $[n, k, W]_q$ codes, then we need the complete lists of $[\le n-1, k-1, W]_q$ codes, where $[\le n', k', W_q']$ codes are those with an effective length of at most n'. Given an $[n', k-1, W]_q$ code with $n' \le n-1$ we only need to extend those codes which have a minimum column multiplicity of at least $n-n'$ via Lemma 2.1. If $n-n' > 1$ this usually reduces the list of codes, where an extensions needs to be computed. Once the set $\mathcal{S}(G)$ of feasible solutions is given, we can also sift out some solutions before applying the isomorphism sifting step. Corollary 2.3 allows us to ignore all resulting codes which have a minimum column multiplicity strictly smaller than $n-n'$. Note that when we know $x_P > 0$, which we do know e.g. for $P = \langle e_i \rangle$, where $1 \le i \le k+1$, then we can add the

[4]This operation is also called *lengthening* in the coding theoretic literature, i.e., both the effective length n and the dimension k is increased, while one usually assumes that the redundancy $n-k$ remains fix. The reverse operation is called *shortening*.

valid inequality $x_P \ge n - n'$ to the inequality system from Lemma 2.1. We call the application of the extension step of Lemma 2.1 under these extra assumptions *canonical length extension* or *canonical lengthening*.

As an example we consider the $[7,2]_2$ code that arises from two codewords of Hamming weight 4 whose support intersect in cardinality 1, i.e., their sum has Hamming weight 6. A direct construction gives the generator matrix

$$G_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

which can be transformed into

$$G_2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Now column permutations are necessary to obtain a systematic generator matrix

$$G_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Note that G_2 and G_3 do not generate the same but only isomorphic codes. Using the canonical length extension the systematic generator matrix

$$G_0 = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$$

of a single codeword of Hamming weight 4 cannot be extended to G_3, since we would need to choose $r = 3$ to get from a $[4,1]_2$ code to a $[7,2]_2$ code, while the latter code has a minimum column multiplicity of 1. However, the unique codeword with Hamming weight 6 and systematic generator matrix

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

can be extended to

$$G_4 = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix},$$

which generates the same code as G_3. So, we needed to consider an extension of a $[6,1]_2$ code to a $[7,2]_2$ code. Now let us dive into the details of the integer linear programming formulation of Lemma 2.1. In our example we have $k = 1$ and $q = 2$, so that $\mathcal{P}_1 = \{\langle(1)\rangle\}$, and

$$\mathcal{P}_2 = \left\{ \left\langle \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\rangle, \left\langle \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\rangle, \left\langle \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\rangle \right\}.$$

The multiplicities corresponding to the columns of G and r are given by

$$c(\langle(1)\rangle) = 6 \quad \text{and} \quad c(\langle(0)\rangle) = 1.$$

Due to constraint (2.2) we have

$$x_{\langle e_1\rangle} + x_{\langle e_1+e_2\rangle} = 6 \quad \text{and} \quad x_{\langle e_2\rangle} = 1.$$

Constraint (2.3) reads

$$x_{\langle e_1\rangle} \ge 1 \quad \text{and} \quad x_{\langle e_2\rangle} \ge 1.$$

In order to write down constraint (2.1), we need to specify the set W of allowed weights. Let us choose $W = \{4,6\}$, i.e., $\Delta = 2$, $a = 2$, and $b = 3$. If we label the hyperplanes by $\mathcal{H} = \{1,2,3\}$, for the ease of notation, we obtain

$$\begin{aligned} 2y_1 + x_{\langle e_2\rangle} &= 3, \\ 2y_2 + x_{\langle e_1+e_2\rangle} &= 3, \text{ and} \\ 2y_3 + x_{\langle e_1\rangle} &= 3. \end{aligned}$$

Since the y_i are in $\{0,1\}$ we have $x_{\langle e_1\rangle} \le 3$ and $x_{\langle e_1+e_2\rangle} \le 3$, so that $x_{\langle e_1\rangle} = 3$ and $x_{\langle e_1+e_2\rangle} = 3$. The remaining variables are given by $x_{\langle e_2\rangle} = 1$, $y_1 = 1$, $y_2 = 0$, and y_3. Thus, in our example there is only one unique solution, which then corresponds to generator matrix G_4 (without specifying the exact ordering of the columns of G_4).

Note that for the special situation $k+1=2$, every hyperplane of $\mathcal{P}_2$ consists of a unique point. The set of column or point multiplicities is left invariant by every isometry of a linear code. For hyperplanes in $\operatorname{PG}(k+1,\mathbb{F}_q)$ or non-zero codewords of C' a similar statement applies. To this end we introduce the weight enumerator $w_C(x) = \sum_{i=0}^{n} A_i x^i$ of a linear code C, where A_i counts the number of codewords of Hamming weight exactly i in C. Of course, the weight enumerator $w_C(x)$ of a linear code C does not depend on the chosen generator matrix C. The geometric reformulation uses the number a_i of hyperplanes $H \in \mathcal{H}_k$ with $\#H \cap \mathcal{M} := \sum_{P\in\mathcal{P}_k\,:\,P\in\mathcal{M},\,P\le H} m(P) = i$. The counting vector $(a_0,\dots,a_n)$ is left unchanged by isometries. One application of the weight enumerator in our context arises when we want to sift out isomorphic copies from a list $\mathcal{C}$ of linear codes. Clearly, two codes whose weight enumerators do not coincide, cannot be isomorphic. So, we can first split $\mathcal{C}$ according to the occurring different weight enumerators and then apply one of the mentioned algorithms for the ismorphism filtering on the smaller parts separately. We can even refine this invariant a bit more. For a given $[n,k]_q$ code C with generator matrix G and corresponding multiset $\mathcal{M}$ let $\widetilde{\mathcal{M}}$ be the set of different elements in $\mathcal{M}$, i.e., $\#\mathcal{M} = \sum_{P\in\widetilde{\mathcal{M}}} m(P)$, which means that we ignore the multiplicities in $\widetilde{\mathcal{M}}$. With this we can refine Lemma 2.2:

Lemma 2.4. *Let C be an $[n,k,W]_q$ code with generator matrix G and $\mathcal{M}$, $\widetilde{\mathcal{M}}$ as defined above. For each $P \in \widetilde{\mathcal{M}}$ there exists a generator matrix G_P of an $[n-m(P),k-1]_q$ code such that the extension of G_P via Lemma 2.1 yields at least one code that is isomorphic to C.*

Now we can use the possibly different weight enumerators of the subcodes generated by G_P to distinguish some of the extension paths.

Corollary 2.5. *Let C' be an $[n',k',W]_q$ code with generator matrix G', minimum column multiplicity r, and $\mathcal{M}$, $\widetilde{\mathcal{M}}$ as defined above. Then there exists a generator matrix G of an $[n'-r,k'-1,W]_q$ code C such that the extension of G via Lemma 2.1 yields at least one code that is isomorphic C' and the weight enumerator $w_C(x)$ is lexicographically minimal among the weight enumerators $w_{C_P}(x)$ for all $P \in \widetilde{\mathcal{M}}$ with column multiplicity r in C', where C_P is the linear code generated by the generator matrix G_P from Lemma 2.4.*

We remark that the construction for subcodes, as described in Lemma 2.4, can also be applied for points $P \in \mathcal{P}_k\backslash\mathcal{M}$. And indeed, we obtain an $[n-m(P),k-1]_q = [n,k-1]_q$ code, i.e., the effective length does not decrease, while the dimension decreases by one.

The algorithmic implication of Corollary 2.5 is the following. Assume that we want to extend an $[n,k,W]_q$ code C with generator matrix G to an $[n+r,k+1,W]_q$ code C' with generator matrix G'. If the minimum column multiplicity of C is strictly smaller than r, then we do not need to compute any extension at all. Otherwise, we compute the set $\mathcal{S}(G)$ of solutions according to Lemma 2.1. If a code C' with generator matrix G', corresponding to a solution in $\mathcal{S}(G)$, has a minimum column multiplicity which does not equal r, then we can skip this specific solution. For all other candidates let $\overline{\mathcal{M}} \subseteq \mathcal{P}_{k+1}$ the set of all different points spanned by the columns of G' that have multiplicity exactly r. By our previous assumption $\overline{\mathcal{M}}$ is not the empty set. If $w_C(x)$ is the lexicographically minimal weight enumerator among all weight enumerators $w_{C_P}(x)$, where $P \in \overline{\mathcal{M}}$ and C_P is generated by the generator matrix G_P from Lemma 2.4, then we store C' and skip it otherwise. We call the application of the extension step of Lemma 2.1 under these extra assumptions *lexicographical extension* or *lexicographical lengthening*.

Lexicographical lengthening drastically decrease the ratio between the candidates of linear codes that have to be sifted out and the resulting number of non-isomorphic codes. This approach also allows parallelization of our enumeration algorithm, i.e., given an exhaustive list $\mathcal{C}$ of all $[n,k,W]_q$ codes and an integer $r \geq 1$, we can split $\mathcal{C}$ into subsets $\mathcal{C}_1,\dots,\mathcal{C}_l$ according to their weight enumerators. If the $[n+r,k+1,W]_q$ code C' arises by lexicographical lengthening from a code in $\mathcal{C}_i$ and the $[n+r,k+1,W]_q$ code C'' arises by lexicographical lengthening from a code in $\mathcal{C}_j$, where $i \neq j$, then C' and C'' cannot be isomorphic. As an example, when constructing the even $[21,8,6]_2$ codes from the 17 927 353 $[20,7,6]_2$ codes, we can split the construction into more than 1000 parallel jobs. If we do not need the resulting list of 1 656 768 624 linear codes for any further computations, there is no need to store the complete list of codes during the computation.

3. Numerical results

As the implementation of a practically efficient algorithm for the classification of linear codes is a delicate issue, we exemplarily verify several classification results from the literature. Efficiency is demonstrated by partially extending some of these enumeration results. In Subsection 3.1 we show up some applications how exhaustive lists of linear codes can be used to find the extremal values of certain parameters of linear codes.

In [12, Research Problem 7.2] the authors ask for the classification of $[n,k,3]_2$ codes for $n > 14$. In Table 1 we extend their Table 7.7 to $n \leq 16$.

n/k	1	2	3	4	5	6	7	8	9	10	11
3	1										
4	1										
5	1	1									
6	1	3	1								
7	1	4	4	1							
8	1	6	10	5							
9	1	8	23	23	5						
10	1	10	42	76	41	4					
11	1	12	71	207	227	60	3				
12	1	15	115	509	1012	636	86	2			
13	1	17	174	1127	3813	4932	1705	110	1		
14	1	20	255	2340	12836	31559	24998	4467	127	1	
15	1	23	364	4606	39750	176582	293871	132914	11507	143	1
16	1	26	505	8685	115281	896316	2955644	3048590	733778	28947	144

TABLE 1. The number of inequivalent $[n,k,3]_2$ codes for $n \leq 16$

We remark that the entries [12, Table 7.7] are given for the number of $[\leq n,k,3]_2$ codes in our notation, i.e., the numbers in Table 1 above an entry have to be summed up to be directly compareable. Blank entries correspond to the non-existence of any code with these parameters, i.e., there is no $[4,2,3]_2$ code and also no $[16,12,3]_2$ code. Obviously, there is a unique $[n,1,3]_2$ codes for each $n \geq 3$ and it is not too hard to show that the number of inequivalent $[n,2,3]_2$ codes is given by $\left\lceil \sqrt{\frac{(n-4)(n-3)(2n-7)}{6}} \right\rceil$ for each $n \geq 3$. For each dimension $k \geq 1$ the maximum possible length n of an $[n,k,3]_2$ code is also known. I.e., for each integer $r \geq 2$ there exists a unique $[2^r-1,2^r-r-1,3]_2$ code, which is called the $(2^r-1,2^r-r-1)$ *Hamming code*. Other "optimal" codes can be obtained by shortening. E.g., there exist $[16+l,11+l,3]_2$ codes for $0 \leq l \leq 15$. Their numbers are given by 144, 129, 113, 91, 67, 50, 34, 21, 14, 9, 5, 3, 2, 1, 1, 1. More precisely, not all these codes can be obtained by shortening, but we have completely classified them. In [18] also the number of inequivalent $[\le 15, 7, 3]_2$ codes was stated, which coincides with our enumeration. The entire computation of Table 1 took less than 11 hours of computation time on a single core of a 2.80GHz laptop bought in 2015. As said in [18], it is not impossible to further extend the range of the classification, but we will focus on more interesting enumerations in order to demonstrate that also much larger numbers of codes can be classified. For completeness, we remark that we have also replicated the counts in tables 2,3 from [18].

k	4	5	6	7	8	9	10	11	12	13
#	8561	129586	1813958	16021319	60803805	73340021	22198835	1314705	11341	24

TABLE 2. The number of inequivalent even $[\le 19, k, 4]_2$ codes for $4 \le k \le 13$

In [5, Table 5] the counts of the even $[\le 18, k, 4]_2$ codes are stated. We have verified these results and present the counts for the even $[\le 19, k, 4]_2$ codes in Table 2. The counts of the even $[\le 20, k, 6]_2$ codes are presented in [5, Table 4]. We have verified these results and extended them to length $n \le 21$ in Table 3 (excluding the enumeration of the even $[21, 9, 6]_2$ codes[5]). To turn these multitude of codes into something more manageable, we have used those results to classify all even $[k + 10, k, 6]_2$ codes. For $k \ge 12$ their numbers are given by 127, 8, and 1, i.e., there is a unique even $[24, 14, 6]_2$ code, which is e.g. generated by

$$\begin{pmatrix}
111111100010000000000000\\
000111111101000000000000\\
111011111100100000000000\\
001101100100010000000000\\
011010101000001000000000\\
110001110000000100000000\\
111101011000000010000000\\
101110001000000001000000\\
110110110100000000100000\\
101010110000000000010000\\
101011000100000000001000\\
100010011100000000000100\\
110101000100000000000010\\
101001101000000000000001
\end{pmatrix},$$

has weight enumerator

$$w_C(x) = x^0 + 336x^6 + 1335x^8 + 3888x^{10} + 5264x^{12} + 3888x^{14} + 1335x^{16} + 336x^{18} + x^{24},$$

and has an automorphism group of order 96. The non-existence of a $[25, 15, 6]_2$ code is well-known [20].

k	3	4	5	6	7	8	10	11
#	726	12817	358997	11697757	246537467	1697180017	62180809	738

TABLE 3. The number of even $[\le 21, k, 6]_2$ codes for $3 \le k \le 11$, $k \ne 9$

For length $n = 20$ the most time expensive step, i.e., extending the $[19, 7, 6]_2$ codes to $[20, 8, 6]_2$ codes, took roughly 250 hours of computation time on a single core of a 2.80GHz laptop. We

[5]Already the 17 927 353 even $[20, 7, 6]_2$ codes can be extended to 1 656 768 624 even $[21, 8, 6]_2$ codes, so that we skipped the extension of the 39 994 046 even $[20, 8, 6]_2$ codes.

remark that the $[19, k, 4]_2$ codes, where $k \in \{7, 8, 9, 10\}$, and the $[21, k, 6]_2$ codes, where $k \in \{7, 8, 10\}$, were enumerated in parallel, i.e., we have partially used the computing nodes of the *High Performance Computing Keylab* from the University of Bayreuth. We have used the oldest cluster btrzx5 that went into operation in 2009.[6] This setup is chosen as an endurance test for our algorithm with hundred parallel jobs. During execution a few hard disks and CPUs died. We have tried our very best to detect possible hardware failures and to rerun all suspicious jobs. However, we are not 100% sure that in those mentioned cases, which run on the computing cluster, the stated numbers are correct, which makes it a perfect opportunity for independent verification by other algorithms.

n/k	2	3	4	5	6	7	8
35	0	1	4	4	3	1	0
36	4	10	22	13	4	0	0
37	0	2	7	10	3	1	0
38	0	1	6	12	10	3	1
39	3	15	34	41	23	8	2
40	0	6	25	40	30	10	1
41	0	0	0	0	0	0	0
42	2	17	52	44	15	0	0
43	0	6	32	40	16	3	0
44	0	2	14	22	17	6	1
45	5	31	141	190	72	13	0
46	0	6	56	122	71	18	3
47	0	2	29	92	89	36	8
48	5	44	297	705	468	128	28
49	0	15	177	613	596	219	37
50	0	2	39	217	295	149	40
51	3	54	572	2405	2263	712	165
52	0	18	333	1828	2909	1595	448
53	0	6	116	1008	3512	3018	815
54	8	91	1427	11121	23835	16641	2718
55	0	19	651	4682	5839	1789	212

TABLE 4. The number of 9-divisible $[n, k, 9]_3$ codes for $35 \le n \le 55$ and $2 \le k \le 8$

Moreover, we have verified

- the explicit numbers of the optimal binary codes of dimension 8 in [5, Table 8];
- the enumerations results for the uniqueness of the $[46, 9, 20]_2$ code presented in [14];
- the enumeration of the projective 2, 4-, and 8-divisible binary linear codes from [9];
- the counts of 9-divisible ternary codes in [5, Table 6]; and
- the counts of 4-divisible quaternary codes in [5, Table 7].

Just to also have an extended example for a field size $q > 2$ we have extended the results from [5, Table 6] on 9-divisible ternary codes to dimensions $k \le 8$ and length $n \le 55$, see Table 4. The conspicuous zero row for length $n = 41$ has a theoretical explanation, i.e., there is no 9-divisible $[41, k]_3$ code at all, see [13, Theorem 1].[7]

[6] The precise technical details can be found at `https://www.bzhpc.uni-bayreuth.de/de/keylab/Cluster/btrzx5_page/index.html`.

[7] More precisely, $41 = 2 \cdot 13 + 2 \cdot 12 - 1 \cdot 9$ is a certificate for the fact that such a code does not exist, see [13, Theorem 1, Example 6].

3.1. **Applications.** In this subsection we want to exemplarily show up, that exhaustive enumeration results of linear codes can of course be used to obtain results for special subclasses of codes and their properties by simply checking all codes. For our first example we remark that the support of a codeword is the set of its non-zero coordinates. A non-zero codeword c of a linear code C is called minimal if the support of no other non-zero codeword is contained in the support of c, see e.g. [1]. By $m_2(n,k)$ we denote the minimum number of minimal codewords of a projective[8] $[n,k]_2$ code. In Table 5 we state the exact values of $m_2(n,k)$ for all $2 \le k \le n \le 15$ obtained by enumerating all projective codes with these parameters.

n/k	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	3	3												
4		4	4											
5		6	5	5										
6		7	6	6	6									
7		7	8	7	7	7								
8			8	9	8	8	8							
9			12	9	9	9	9	9						
10			14	10	10	10	10	10	10					
11			14	15	11	11	11	11	11	11				
12			15	15	13	12	12	12	12	12	12			
13			15	16	14	13	13	13	13	13	13	13		
14			15	16	14	15	14	14	14	14	14	14	14	
15			15	16	17	15	16	15	15	15	15	15	15	15

TABLE 5. $m_2(n,k)$ for $3 \le n \le 15, 1 \le k \le 9$

In our second example we want to use the enumeration results from Table 4 on ternary 9-divisible linear codes. In [10] it was mentioned that the smallest length n of a projective ternary 9-divisible linear code whose existence is unknown is $n = 70$. The possible weights are 9, 18, 27, 36, 45, and 54, since a codeword with weight 63 would yield a projective 3-divisible $[7,k]_3$ code, which does not exist, see [13]. Of course it is in principle possible to enumerate all 9-divisible $[70,k]_3$ codes. However, there are already 85037 such $[70,4]_3$ codes and their numbers explode with increasing dimension k. So, let us first derive some conditions on a hypothetical 9-divisible $[70,k]_3$ code C. By A_i we denote the number of codewords of weight i of C and by B_i the number of codewords of weight i of the dual code of C. The first equations of the well-known MacWilliams identities, see e.g. [16], are given by:

$$1+A_9+A_{18}+A_{27}+A_{36}+A_{45}+A_{54}=3^k \tag{3.1}$$

$$70+61A_9+52A_{18}+43A_{27}+34A_{36}+25A_{45}+16A_{54}=70\cdot 3^{k-1} \tag{3.2}$$

$$2415+1830A_9+1326A_{18}+903A_{27}+561A_{36}+300A_{45}+120A_{54}=2415\cdot 3^{k-2} \tag{3.3}$$

$$54740+35990A_9+22100A_{18}+12341A_{27}+5984A_{36}+2300A_{45}+560A_{54}=(54740+B_3)\,3^{k-3} \tag{3.4}$$

20 times Equation (3.1) minus 2 times Equation (3.2) plus $\frac{1}{10}$ times Equation (3.3) gives

$$\frac{3^5}{2}+81A_9+\frac{243A_{18}}{5}+\frac{243A_{27}}{10}+\frac{81A_{36}}{10}=\frac{3^k}{6},$$

[8]Duplicating columns in a binary linear code generated by the $k \times k$ unit matrix results in exactly k minimal codewords, which is the minimum for all k-dimensional codes.

so that $k \ge 6$, since $A_i \ge 0$. For $k = 6$ the polyhedron given by equations (3.1)-3.4) and the nonegativity constraints $A_i, B_3 \ge 0$ contains the unique point

$$A_9 = A_{18} = A_{27} = A_{36} = 0, A_{45} = 588, A_{54} = 140, \text{ and } B_3 = 280.$$

However, a linear code C with these parameters would be a 2-weight code and the corresponding strongly regular graph does not exist, see e.g. [6] for the details. (We have also excluded this case by exhaustively enumerating the (non-existent) $[70, 6, \{45, 54\}]_3$ codes.) Thus, we can assume $k \ge 7$. For $k = 7$ we can again consider the polyhedron given by equations (3.1)-3.4) and the nonegativity constraints $A_i, B_3 \ge 0$. Additionally we can assume that the A_i are even integers. By solving the corresponding integer linear programs we can verify $A_9 \le 2$, $A_{18} \le 4$, $A_{27} \le 10$, and $A_{36} \le 20$. Moreover, the first two constraints can be tightened to $2A_9 + A_{18} \le 4$. We also can derive a condition on the length and the minimum column multiplicity, i.e., if a 9-divisible $[n, k]_3$ code C has minimum column multiplicity Λ and $n + (7 - k) \cdot \Lambda < 70$, then C cannot be extended to a 9-divisible $[70, 7]_3$ code via canonical lengthening, since in each extension step the length can increase by at most Λ. With those conditions we have performed a restricted generation of linear codes. We have indeed constructed a few hundred of $[69, 6, \{9, 18, 27, 36, 45, 54\}]_3$ codes with maximum column multiplicity 3. However, none of these was extendable to a projective 9-divisible $[70, 7]_3$ code and we conjecture that no such code exists. Nevertheless, the above extra conditions drastically reduce the search space, it is still too large for our current implementation. In our computational experiments we have stopped the extension using `Solvediophant` after 10 minutes for each code, while we have seen unfinished lattice point enumerations lasting several hours. Moreover, we were not able to extend all 5-dimensional codes due to their large number.

4. Conclusion

We have presented an algorithm for the classification of linear codes over finite fields based on lattice point enumeration. The lattice point enumeration itself and sifting out isomorphic copies is so far done with available scientific software packages. Using invariants like the weight enumerator of subcodes, see Corollary 2.5, the number of candidates before sifting could kept reasonably small. The resulting algorithm is quite competitive compared to e.g. the recent algorithm described in [5]. There the authors used the appealing technique of canonical augmentation or orderly generation, see e.g. [19]. The advantage that no pairs of codes have to be checked whether they are isomorphic comes at the cost that the computation of the canonical form is relatively costly, see [5]. Allowing not only a single canonical extension, but a relatively small number of extensions that may lead to isomorphic codes, might be a practically efficient alternative. We have also demonstrated that the algorithm can be run in parallel.

However, we think that our implementation can still be further improved. In some cases the used lattice point enumeration algorithm `Solvediophant` takes quite long to verify that a certain code does not allow an extension, while integer linear programming solvers like e.g. `Cplex` quickly verify infeasibility. Especially the computational experiments at the end of Subsection 3.1 suggest, that it is worthwhile to try to speed up the lattice point enumeration. We propose the extension of Table 4 as a specific open problem.

Also it would be beneficial if at least some restriction of a lexicographical extension could be directly formulated as valid constraints in the integer linear programming formulation of Lemma 2.1. So far we have not used known automorphisms of the linear code that should be extended. It is not implausible to expect that there for different parameter ranges different algorithmic choices can perform better. In any case, we have demonstrated that it is indeed possible to exhaustively classify sets of linear codes of magnitude 10^9, which was not foreseeable at the time of [12].

Currently the implementation of the evolving software package `LinCode` is not that progressed to be made publicy available. So, we would like to ask the readers to sent their interesting enumeration problems of linear codes to the author directly.

REFERENCES

[1] A. Ashikhmin and A. Barg. Minimal vectors in linear codes. 44(5):2010–2017, 1998.

[2] A. Betten, M. Braun, H. Fripertinger, A. Kerber, A. Kohnert, and A. Wassermann. *Error-correcting linear codes: Classification by isometry and applications*, volume 18. Springer Science & Business Media, 2006.

[3] I. Bouyukliev. About the code equivalence. In T. Shaska, W. Huffman, D. Joyner, and V. Ustimenko, editors, *Advances in Coding Theory and Cryptology*, pages 126–151. 2007.

[4] I. Bouyukliev. What is q-extension? *Serdica Journal of Computing*, 1(2):115–130, 2007.

[5] I. Bouyukliev and S. Bouyuklieva. Classification of linear codes using canonical augmentation. *arXiv preprint 1907.10363*, 2019.

[6] A. E. Brouwer, A. M. Cohen, and A. Neumaier. *Distance-regular Graphs*. Springer, 1989.

[7] S. Dodunekov and J. Simonis. Codes and projective multisets. *The Electronic Journal of Combinatorics*, 5(1):37, 1998.

[8] T. Feulner. The automorphism groups of linear codes and canonical representatives of their semilinear isometry classes. *Advances in Mathematics of Communication*, 3(4):363–383, 2009.

[9] D. Heinlein, T. Honold, M. Kiermaier, S. Kurz, and A. Wassermann. Projective divisible binary codes. In *The Tenth International Workshop on Coding and Cryptography 2017 : WCC Proceedings*. Saint-Petersburg, September 2017.

[10] T. Honold, M. Kiermaier, S. Kurz, and A. Wassermann. The lengths of projective triply-even binary codes. *IEEE Transactions on Information Theory*, pp. 4, to appear. doi: 10.1109/TIT.2019.2940967.

[11] D. B. Jaffe. Optimal binary linear codes of length ≤ 30. *Discrete Mathematics*, 223(1-3):135–155, 2000.

[12] P. Kaski and P. R. Östergård. *Classification algorithms for codes and designs*, volume 15. Springer, 2006.

[13] M. Kiermaier and S. Kurz. On the lengths of divisible codes. *IEEE Transactions on Information Theory*, pp. 10, to appear.

[14] S. Kurz. The $[46, 9, 20]_2$ code is unique. *arXiv preprint 1906.02621*, 2019.

[15] H. W. Lenstra, A. K. Lenstra, and L. Lová s. Factoring polynomials with rational coeficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[16] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[17] B. McKay. Nauty user's guide (version 1.5). *Technical report*, 1990.

[18] P. R. Östergård. Classifying subspaces of hamming spaces. *Designs, Codes and Cryptography*, 27(3):297–305, 2002.

[19] G. F. Royle. An orderly algorithm and some applications in finite geometry. *Discrete Mathematics*, 185(1-3):105–115, 1998.

[20] J. Simonis. Binary even $[25, 15, 6]$ codes do not exist. *IEEE Transactions on Information Theory*, 33(1):151–153, 1987.

[21] D. Slepian. Some further theory of group codes. *Bell System Technical Journal*, 39(5):1219–1252, 1960.

[22] H. Ward. Divisible codes-a survey. *Serdica Mathematical Journal*, 27(4):263p–278p, 2001.

[23] H. N. Ward. Divisible codes. *Archiv der Mathematik*, 36(1):485–494, 1981.

[24] A. Wassermann. Attacking the market split problem with lattice point enumeration. *Journal of Combinatorial Optimization*, 6(1):5–16, 2002.

SASCHA KURZ, DEPARTMENT OF MATHEMATICS, PHYSICS AND INFORMATICS, UNIVERSITY OF BAYREUTH, BAYREUTH, GERMANY

Email address: `sascha.kurz@uni-bayreuth.de`