

An algorithm for computing generalized Hamming weights and the Sage package GHWs

RODRIGO SAN-JOSÉ, Virginia Tech, US and Universidad de Valladolid, Spain

We generalize the Brouwer-Zimmermann algorithm, which is the most efficient general algorithm for computing the minimum distance of a random linear code, to the case of generalized Hamming weights. We also adapt this algorithm to compute the relative generalized Hamming weights of a nested pair of linear codes. In the package GHWs we provide an implementation of this algorithm in Sage, as well as several other utilities for working with generalized Hamming weights. With this implementation, we show that the proposed algorithm is faster than the naive approach of computing the generalized Hamming weights using the definition.

CCS Concepts: • **Mathematics of computing** → **Coding theory**.

Additional Key Words and Phrases: Generalized Hamming weights, Brouwer-Zimmermann algorithm, secret sharing schemes, linear codes.

1 INTRODUCTION

Linear error correcting codes were introduced to study the problem of sending digital information through a noisy channel [33]. Linear codes are able to correct errors by adding redundancy to the message, and they have three main basic parameters, namely the length n , the dimension k , and the minimum distance d . The rate k/n determines how much information the code can transmit per bit (the higher the rate, the lower redundancy we require), and d/n determines the proportion of errors the code can correct. These parameters are related (for example, via the Singleton bound $d \leq n - k + 1$), and finding codes with an optimal trade-off is a difficult problem in general. Over time, linear codes have found many additional applications. To determine the performance of the code for each of them, additional properties have to be considered. Examples of such applications are locally recoverable codes [38], code-based cryptography [36, 37], or quantum codes [6, 44], in which one has to study locality parameters and orthogonality properties.

The generalized Hamming weights (GHWs) of a linear code, first defined in [49], generalize the notion of minimum distance. In the original paper [49], it is shown that these parameters characterize the performance of the code on the wire-tap channel of type II and as a t -resilient function. In [16, 18], the connection with list decoding is established, and it is used to derive better bounds for the list size. Related to the GHWs, the relative generalized Hamming weights (RGHWs) of a nested pair of codes were introduced as an extension of GHWs, mainly to characterize the security of linear ramp secret-sharing schemes [31, 32]. It is widely known that the first RGHW, also known as relative minimum distance, determines the minimum distance

This work was supported in part by the following grants: Grant PID2022-138906NB-C21 funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU, and FPU20/01311 funded by the Spanish Ministry of Universities.

Author's address: Rodrigo San-José, rsanjose@vt.edu, Virginia Tech, Department of Mathematics, Blacksburg, Virginia, US and Universidad de Valladolid, IMUVA-Mathematics Research Institute, 47011, Valladolid, Spain.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s).

ACM 1557-7295/2025/10-ART

<https://doi.org/10.1145/3773284>

of quantum codes [30]. However, the second RGHW also determines the minimum distance of the quantum codes obtained via Steane enlargement [45], as seen in [19, Lem. 5].

These applications have motivated the study of codes with algebraic structure that allows to derive properties about their GHWs and their RGHWs. Some classical results concern cyclic codes [12, 23, 50, 52, 53] and Reed-Muller codes [13, 20], as well as algebraic geometry codes [2, 14, 51], and there have been many recent advances with some well-known classes of codes, such as projective Reed-Muller codes [4, 40], Cartesian codes [3, 10], hyperbolic codes [7], matrix-product codes [41] or norm-trace codes [8].

However, if we consider a random linear code, that is, without any known structure, the computation of its minimum distance is an NP hard problem [48]. The fastest general algorithm for computing the minimum distance of a random code is the Brouwer-Zimmermann algorithm [54] (also see [17] for a review of this algorithm and improvements). Since the GHWs generalize the minimum distance, and the RGHWs extend the GHWs, their computation is at least as intractable as that of the minimum distance. The approach from [24] allows one to compute the GHWs of a linear code from a minimal graded free resolution of the matroid associated with a parity-check matrix of the code. While this provides a fruitful connection to commutative algebra, computing a minimal free resolution typically involves Gröbner basis computations, which is a computationally challenging task [34, 35, 43]. To the best of the author's knowledge, there is currently no efficient algorithm or implementation available in the literature for computing the GHWs of a random linear code over an arbitrary finite field.

The goal of this paper is to generalize the Brouwer-Zimmermann algorithm for computing the GHWs of a random linear code and the RGHWs of a nested pair of linear codes, and give a general implementation of this algorithm. We present the implementation in the form of a Sage package [42, 47], which also includes additional functions related to GHWs, such as a function to compute the higher weight spectra of a linear code. The aim of this package is to provide a general implementation that works for any linear code. In particular, this means that the implementation should work for any finite field, and therefore we have to use specialized mathematical software. The most common ones for working with linear codes are Magma [5], GAP [9, 46] and Sage [47]. We decided to use Sage because it is open-source and it is based on Python, making it easier to analyze and understand, which is another important objective of this package, since we would like this work to pave the way for research on how to compute the GHWs of a linear code more efficiently.

The structure of the paper is as follows. In Section 2 we give the necessary mathematical background related to linear codes and their GHWs and RGHWs. Section 3 explains how to generalize the Brouwer-Zimmermann algorithm to this setting. In Section 4, we study how to construct subspaces of a given dimension with increasing size of their support. This problem translates to constructing all possible matrices in reduced row echelon form with a certain "shape", and we show how to implement the construction of these matrices. Section 5 describes the Sage implementation of our algorithms. We explain which functions are included in the package and we give more details on how exactly we have implemented certain aspects of the algorithm. We test our implementation in Section 6, where we first explain how we can check that the results of the implementation are correct, and then we conduct a performance analysis. Since currently there are no other alternatives, we compare our implementation with a naive implementation, using directly the definition of the GHWs. Our tests show that the difference in performance is considerable, e.g., we need to use a logarithmic scale for the time in the graphics since the difference in speed is exponential. We include in the package an alternative implementation of some of the functions which requires less RAM usage, and we study the difference in performance with respect to the standard implementation. We also compare the time required to compute the GHWs independently one by one (using the function `GHW`), and computing them using the information of the previous ones (using the function `hierarchy`). Finally, in Section 7, we summarize our results and propose new avenues of research.

2 PRELIMINARIES

Let $q = p^s$ be a power of a prime p . We denote by \mathbb{F}_q the finite field with q elements. We say that C is a linear code of length n and dimension k if it is a k dimensional linear subspace of \mathbb{F}_q^n . A matrix $G \in \mathbb{F}_q^{k \times n}$ is a generator matrix of C if its rows form a basis of C . The dual code of C is denoted by C^\perp , and is the orthogonal space of C with respect to the usual Euclidean inner product. Thus, we have $\dim C^\perp = n - k$. A generator matrix $H \in \mathbb{F}_q^{(n-k) \times n}$ of C^\perp is called a parity check matrix for C . The main parameters of the code C are the length n , the dimension k , and the minimum distance d , and we say that C is an $[n, k, d]$ code. The parameter d is defined in terms of the Hamming weight. Given $c \in \mathbb{F}_q^n$, we define its Hamming weight as

$$\text{wt}(c) := |\{i : c_i \neq 0\}|.$$

The minimum distance $d = \text{wt}(C)$ of $C \subset \mathbb{F}_q^n$ is then defined as

$$\text{wt}(C) := \min\{\text{wt}(c) : c \in C, c \neq 0\}.$$

The naive method to compute $\text{wt}(C)$ would be to enumerate all the codewords $c \in C$, and then compute the minimum of their weights. Since $\text{wt}(c) = \text{wt}(\lambda c)$ for any $\lambda \in \mathbb{F}_q \setminus \{0\}$, we would need to enumerate

$$\frac{q^k - 1}{q - 1}$$

codewords (this is the number of nonzero vectors of \mathbb{F}_q^k up to multiples). Now we define the GHWs of a linear code. For this, we have to define first the support of a linear subspace. Given $D \subset \mathbb{F}_q^n$, the support of D is

$$\text{supp}(D) := \{1 \leq i \leq n : \exists c \in D \text{ with } c_i \neq 0\}.$$

Note that if G_D is a generator matrix for D , that is, a matrix whose rows form a basis for D , and $\{g_1, \dots, g_r\}$ are the rows of G_D , then

$$\text{supp}(D) = \bigcup_{i=1}^r \text{supp}(g_i). \quad (1)$$

In particular, this shows that computing the support of a subcode D from a generator matrix G_D is straightforward.

Definition 2.1. The r th generalized Hamming weight of an $[n, k, d]$ code C is

$$d_r(C) := \min \{|\text{supp}(D)| : D \text{ is a subcode of } C \text{ of dimension } r\},$$

where $1 \leq r \leq k$. The weight hierarchy of C is the set

$$\{d_r(C) : 1 \leq r \leq k\}.$$

Remark 2.2. If $r = 1$, the subspaces D of C with dimension 1 are precisely the codewords (up to multiples). The support of a codeword is given by the positions of its nonzero coordinates, and its cardinality is precisely the number of nonzero coordinates, that is, its Hamming weight. Thus, we have $d_1(C) = \text{wt}(C)$.

From [49] we have the following general properties of the GHWs of a code.

THEOREM 2.3 (MONOTONICITY). For an $[n, k]$ linear code C with $k > 0$ we have

$$1 \leq d_1(C) < d_2(C) < \dots < d_k(C) \leq n.$$

COROLLARY 2.4 (GENERALIZED SINGLETON BOUND). For an $[n, k]$ linear code C we have

$$d_r(C) \leq n - k + r, \quad 1 \leq r \leq k.$$

THEOREM 2.5 (DUALITY). *Let C be an $[n, k]$ code. Then*

$$\{d_r(C) : 1 \leq r \leq k\} = \{1, 2, \dots, n\} \setminus \{n+1-d_r(C^\perp) : 1 \leq r \leq n-k\}.$$

Remark 2.6. As a consequence of Theorem 2.3 and Corollary 2.4, if C is an MDS code with parameters $[n, k, d]$ (that is, we have $d = n - k + 1$), then we have

$$d_r(C) = n - k + r$$

for any $1 \leq r \leq k$.

The RGHWs, which extend the GHWs, were introduced in [31, 32]. This was in the context of linear ramp secret-sharing schemes, which can be constructed from a pair of nested linear codes. In these schemes, a dealer distributes shares of a secret to a certain number of participants. Using the shares of a sufficiently large number of participants, it is possible to recover the secret. A ramp secret sharing scheme is said to have (t_1, \dots, t_ℓ) -privacy and (h_1, \dots, h_ℓ) -reconstruction if t_1, \dots, t_ℓ are the largest possible, and h_1, \dots, h_ℓ the smallest possible, such that:

- no set of t_m participants can obtain m q -bits of information about the secret,
- any set of h_m participants can obtain m q -bits of information about the secret.

These parameters can be completely characterized in terms of the RGHWs of the pair of linear codes used for the secret sharing scheme, as well as the RGHWs of their duals [14]. Although this is the application in which the RGHWs are used with complete generality, for $r = 1$ (relative minimum distance), they determine the error-correction capability of quantum codes [30]. Moreover, for $r = 2$, they also determine the minimum distance of the codes derived via Steane enlargement [19, 45]. We introduce now their definition.

Definition 2.7. Let $C_2 \subset C_1 \subset \mathbb{F}_q^n$ be two linear codes, and $k_1 = \dim C_1$, $k_2 = \dim C_2$. Let r with $1 \leq r \leq k_1 - k_2$. The r th relative generalized Hamming weight of C_1 and C_2 , denoted by $M_r(C_1, C_2)$, is

$$M_r(C_1, C_2) = \min\{|\text{supp}(D)| : D \text{ is a subcode of } C_1 \text{ with } \dim D = r, D \cap C_2 = \{0\}\}.$$

The relative weight hierarchy of C_1 with respect to C_2 is the set

$$\{M_r(C_1, C_2), 1 \leq r \leq k_1 - k_2\}.$$

It is clear from the definition that the RGHWs are a generalization of the GHWs obtained by considering $C_2 = \{0\}$. Note that, for $1 \leq r \leq k_1 - k_2$, we have

$$M_r(C_1, C_2) \geq d_r(C_1).$$

The RGHWs satisfy similar properties to GHWs, as shown in [32].

THEOREM 2.8. *Let $C_2 \subset C_1 \subset \mathbb{F}_q^n$ be linear codes with $\dim C_1 = k_1$, $\dim C_2 = k_2$. Then*

$$1 \leq M_1(C_1, C_2) < M_2(C_1, C_2) < \dots < M_{k_1-k_2}(C_1, C_2) \leq n.$$

Moreover, for any $1 \leq r \leq k_1 - k_2$, we have

$$M_r(C_1, C_2) \leq n - k_1 + r.$$

Remark 2.9. The monotonicity from Theorem 2.3 (resp. Theorem 2.8) implies $d_r(C) \geq r$ (resp. $M_r(C_1, C_2) \geq r$).

One may wonder if there is a duality result similar to Theorem 2.5 for RGHWs. This is one example in which having a package such as GHWS can be useful for research, since we can look for random codes and see if it is possible to have such a result. In the next example we show two pairs of codes with the same relative weight hierarchy, but different relative weight hierarchy for the duals, which implies that, in general, the RGHWs of a pair of codes do not determine the RGHWs of their duals.

Example 2.10. Let $q = 2$ and consider the codes C_1, C_2, C'_1, C'_2 generated by the matrices

$$G_1 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_2 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix},$$

$$G'_1 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad G'_2 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix},$$

respectively. It is straightforward to check that $C_2 \subset C'_1$ and $C'_2 \subset C_1$. We have $M_1(C_1, C_2) = M_1(C'_1, C'_2) = 2$ and $M_2(C_1, C_2) = M_2(C'_1, C'_2) = 4$. In other words, the pairs (C_1, C_2) and (C'_1, C'_2) have the same relative weight hierarchy. If we consider now the pairs (C_2^\perp, C_1^\perp) and $((C'_2)^\perp, (C'_1)^\perp)$, we have $M_1(C_2^\perp, C_1^\perp) = M_1((C'_2)^\perp, (C'_1)^\perp) = 2$, but $M_2(C_2^\perp, C_1^\perp) = 3 \neq 4 = M_2((C'_2)^\perp, (C'_1)^\perp)$. That is, the dual pairs do not have the same relative weight hierarchy.

Another object that has been studied in this context is the higher weight spectrum [11, 15, 25–28], a generalization of the weight spectrum. For $1 \leq r \leq k$ and for each $w = d_r(C), \dots, n$, we denote by $A_w^{(r)}(C)$ the number of subcodes $D \subset C$ with $\dim D = r$ and $|\text{supp}(D)| = w$. The ordered multiset $\{A_w^{(r)}(C), d_r(C) \leq w \leq n\}$ is the r th weight spectrum of C . The multiset of r th weight spectra for $r = 0, \dots, k$ is the higher weight spectra of C . Note that for this, we consider $d_0(C) = 0$ and $A_0^{(0)}(C) = 1$. We can similarly define the r th relative weight spectrum and higher weight spectra of C_1 with respect to C_2 by only considering subspaces $D \subset C$ such that $D \cap C_2 = \{0\}$.

3 BROUWER-ZIMMERMANN-LIKE ALGORITHM

We consider a code $C \subset \mathbb{F}_q^n$ with $k = \dim C$, and a generator matrix G of C . Given a subspace $E \subset \mathbb{F}_q^k$, we denote by $\text{enc}_G(E) \subset C$ the subspace obtained as the image of E by the linear transformation defined by G . Let $\text{Gr}(r, \mathbb{F}_q^k)$ be the set of all subspaces of \mathbb{F}_q^k with dimension r (the notation comes from the Grassmannian). A naive algorithm to compute the GHWs of C would consist in enumerating all the elements of $\text{Gr}(r, \mathbb{F}_q^k)$, and then computing the minimum of the supports of $\text{enc}_G(E)$, for every $E \in \text{Gr}(r, \mathbb{F}_q^k)$. The number of such subspaces is given by the following Gaussian binomial:

$$\begin{bmatrix} k \\ r \end{bmatrix}_q = \frac{(q^k - 1)(q^k - q) \cdots (q^k - q^{r-1})}{(q^r - 1)(q^r - q) \cdots (q^r - q^{r-1})}.$$

Since this number grows considerably when increasing q and k , any method that allows us to not have to check all of these subspaces can provide a significant computational advantage. The Brouwer-Zimmermann algorithm for the minimum distance considers the codewords with increasing weight, i.e., it first considers the vectors of \mathbb{F}_q^k with weight 1, then those of weight 2, etc. The equivalent procedure in this case would

correspond to considering subspaces $E \subset \mathbb{F}_q^k$ with increasing cardinality of the support, i.e., first we consider the subspaces E with $|\text{supp}(E)| = r$ (recall Remark 2.9), then subspaces E with $|\text{supp}(E)| = r + 1$, etc. Let $w \geq r$, and consider

$$E_w^r := \{E \subset \mathbb{F}_q^k, \dim E = r, |\text{supp}(E)| = w\}.$$

We have the disjoint union

$$\text{Gr}(r, \mathbb{F}_q^k) = \bigsqcup_{w=r}^k E_w^r.$$

We explain now the Brouwer-Zimmermann-like algorithm, similarly to the way the Brouwer-Zimmermann algorithm is introduced in [17]. The proof of the following result is straightforward from the definitions (also recall (1)).

LEMMA 3.1. *Let $G = (I_k, A)$ a generator matrix for a code $C \subset \mathbb{F}_q^n$ and $E \subset \mathbb{F}_q^k$ with $\dim E = r$. Let G_E be a generator matrix for E . Then*

$$G_D := G_E G = (G_E, G_E A)$$

is a generator matrix for a subcode $D \subset C$ with $\dim D = r$ and $|\text{supp}(D)| \geq |\text{supp}(E)|$.

Remark 3.2. The previous result can also be generalized to other contexts, for example for rank-metric codes (or sum-rank metric codes).

Let $G = (I_k, A)$ be as in Lemma 3.1. Assume we have computed

$$d' = \min_{r \leq i \leq w} \{|\text{supp}(\text{enc}_G(E))| : E \in E_i^r\}$$

for some $w < k$. Then $d_r(C) \leq d'$ and, by Lemma 3.1, for every $E \in E_\ell^r$, with $w < \ell$, we must have

$$|\text{supp}(\text{enc}_G(E))| \geq \ell \geq w + 1.$$

If $w + 1 \geq d'$, we obtain $d_r(C) = d'$. Therefore, by enumerating subspaces with increasing support, we have a lower bound for the cardinality of the support of the remaining subspaces. This lower bound can be improved if, instead of only considering one generator matrix, we consider more. First, we introduce the notation of information sets. Let G be a generator matrix for C . We say that $I \subset \{1, \dots, n\}$ is an information set if the set of columns of G given by the indices of I is linearly independent. In that case, we may always find a generator matrix G' of C such that if we consider only the columns corresponding to I , we obtain the identity matrix. In other words, after a permutation, we have $G' = (I_k, A)$. The Brouwer-Zimmermann takes advantage of using several matrices G_j of that form (G_j is a generator matrix for C such that the columns corresponding to I_j give I_k), for disjoint information sets I_1, \dots, I_m . The argument is as follows. If we have computed

$$d' = \min_{r \leq i \leq w} \left\{ \min_{1 \leq j \leq m} \{|\text{supp}(\text{enc}_{G_j}(E))| : E \in E_i^r\} \right\},$$

for some $w < k$, then for every $E \in E_\ell^r$, with $w < \ell$, such that $\text{enc}_{G_j}(E)$ has not been enumerated before, we have

$$|\text{supp}(\text{enc}_{G_j}(E))| \geq m(w + 1),$$

for any $j = 1, \dots, m$. Indeed, fix j and consider $\text{supp}(\text{enc}_{G_j}(E)) \cap I_i$, for some $1 \leq i \leq m$, and assume the subspace $\text{enc}_{G_j}(E)$ has not been enumerated yet. The cardinality of this intersection is greater than or equal to $w + 1$ (otherwise it would have been already enumerated as $\text{enc}_{G_i}(E')$, for some $E' \in E_{\ell'}^r$ with $\ell' \leq w$). Since this is true for every $i = 1, \dots, m$, and the sets I_1, \dots, I_m are disjoint, we conclude the inequality given above.

A way to compute these information sets is to start with a generator matrix of the form $G_1 = (I_k, A)$ (this can always be assumed by permuting), which gives $\mathcal{I}_1 = \{1, \dots, k\}$, and then using Gaussian elimination to obtain $A = (I_k, A')$ (up to permutation), resulting in a generator matrix $G_2 = (B, I_k, A')$ (up to permutation), etc. At some point, we may encounter, for example, that $\text{rank}(A') < k$. In this case, we have to reuse some columns to obtain an information set. In this way we obtain m information sets $\mathcal{I}_1, \dots, \mathcal{I}_m$, such that $\bigcup_{j=1}^m \mathcal{I}_j = \{1, \dots, n\}$, but they are not necessarily disjoint (one can check [17] for the details). For each $1 \leq i \leq m$, we define the redundancy of \mathcal{I}_i as

$$R_i = \left| \mathcal{I}_i \cap \bigcup_{t=1}^{i-1} \mathcal{I}_t \right|.$$

Arguing as we did with the case of disjoint information sets, when using non-disjoint information sets, each generator matrix G_i contributes $(w + 1) - R_i$ to the lower bound. This leads to Algorithm 1, which shows the basic idea of all the algorithms that we will consider. Note that the cost of computing the information sets, and their corresponding generator matrices, is negligible compared to the rest of the algorithm, and we may omit it from the pseudo-code.

Algorithm 1 Brouwer-Zimmermann-like algorithm for the r th GHW

Input: $C \subset \mathbb{F}_q^n$, $1 \leq r \leq k$.

Output: $d_r(C)$.

```

1:  $\text{GHW}_l = r$ 
2:  $\text{GHW}_u = n - k + r$ 
3: while  $w \leq k$  and  $\text{GHW}_l < \text{GHW}_u$  do
4:   for  $E \in E_w^r$  do
5:     for  $j = 1, \dots, m$  do
6:        $\text{GHW}_u = \min\{\text{GHW}_u, |\text{supp}(\text{enc}_{G_j}(E))|\}$ 
7:     end for
8:   end for
9:    $\text{GHW}_l = \sum_{j=1}^m \max\{0, w + 1 - R_j\}$ 
10:   $w = w + 1$ 
11: end while
12: return  $\text{GHW}_u$ 

```

There is one aspect of Algorithm 1 (and Algorithm 2) that has not been covered yet, which is efficiently constructing E_w^r . We study this problem in Section 4, where we show how to obtain all the subspaces of E_w^r without repeating them.

With respect to RGHWs, we can also use Algorithm 1, except that whenever we find $E \subset \mathbb{F}_q^k$ such that $|\text{supp}(\text{enc}_{G_j}(E))| < \text{GHW}_u$, we first check if $\text{enc}_{G_j}(E) \cap C_2 = \{0\}$, and if that is the case, we set $\text{GHW}_u = |\text{supp}(\text{enc}_{G_j}(E))|$; if that is not the case, we continue with the algorithm, without updating the value of GHW_u .

Regarding the weight hierarchy, one could just use Algorithm 1 for each value of $1 \leq r \leq \dim C$. However, we can improve the performance of the algorithm by considering the monotonicity of the GHWs from Theorem 2.5 (resp. Theorem 2.8 for RGHWs), since it gives that $d_{r-1}(C) + 1$ (resp. $M_{r-1}(C_1, C_2) + 1$) is a

lower bound for $d_r(C)$ (resp. $M_r(C_1, C_2)$). Algorithm 2 gives the corresponding algorithm to compute the weight hierarchy. It can be modified to compute the relative weight hierarchy in a similar way as one can modify Algorithm 1 to compute RGHWS.

Algorithm 2 Brouwer-Zimmermann-like algorithm for the weight hierarchy

Input: $C \subset \mathbb{F}_q^n$.

Output: $\{d_r(C), 1 \leq r \leq k = \dim C\}$.

```

1: hierarchy = [ ]
2: for  $r = 1, \dots, k$  do
3:   if  $r = 1$  then
4:      $\text{GHW}_l = 1$ 
5:   else
6:      $\text{GHW}_l = \text{hierarchy}[r - 1] + 1$ 
7:   end if
8:    $\text{GHW}_u = n - k + r$ 
9:   while  $w \leq k$  and  $\text{GHW}_l < \text{GHW}_u$  do
10:    for  $E \in E_w^r$  do
11:      for  $j = 1, \dots, m$  do
12:         $\text{GHW}_u = \min\{\text{GHW}_u, |\text{supp}(\text{enc}_{G_j}(E))|\}$ 
13:      end for
14:    end for
15:     $\text{GHW}_l = \sum_{j=1}^m \max\{0, w + 1 - R_j\}$ 
16:     $w = w + 1$ 
17:  end while
18:  Append  $\text{GHW}_u$  to hierarchy
19: end for
20: return hierarchy

```

4 ENUMERATING AND CONSTRUCTING SUBSPACES WITH A GIVEN SUPPORT

For both Algorithms 1 and 2 we need to construct the sets E_w^r , for $r \leq w \leq k$. If we denote $e_w^r = |E_w^r|$, then we have

$$e_w^r = \binom{k}{w} \left(\binom{w}{r}_q - \binom{w}{1} \binom{w-1}{r}_q + \binom{w}{2} \binom{w-2}{r}_q + \dots + (-1)^{w-r} \binom{w}{w-r} \binom{r}{r}_q \right).$$

This is obtained by first considering a particular support of cardinality w (there are $\binom{k}{w}$), then considering all subspaces in \mathbb{F}_q^w with dimension r (there are $\binom{w}{r}_q$), and finally discarding those subspaces without full support. This can be done by means of the inclusion-exclusion principle: first we discard those subspaces contained in the hyperplane $\{x_i = 0\}$, for some $i = 1, \dots, w$ (there are $\binom{w}{1}$ such hyperplanes, and for each hyperplane we have $\binom{w-1}{r}_q$ subspaces contained in it), which means we have removed the ones

contained in two of those hyperplanes twice, etc. By the definitions, we have the relation

$$\begin{bmatrix} k \\ r \end{bmatrix}_q = \sum_{w=r}^k e_w^r.$$

Remark 4.1. For the case of the minimum distance ($r = 1$), note that

$$\begin{bmatrix} \ell \\ 1 \end{bmatrix}_q = \frac{q^\ell - 1}{q - 1} = q^{\ell-1} + q^{\ell-2} + \dots + 1,$$

and

$$\begin{aligned} e_w^1 &= \binom{k}{w} \left(\frac{q^w - 1}{q - 1} - \binom{w}{1} \frac{q^{w-1} - 1}{q - 1} + \dots + (-1)^{w-1} \binom{w}{w-1} \frac{q - 1}{q - 1} \right) \\ &= \frac{\binom{k}{w}}{q - 1} \left(q^w - \binom{w}{1} q^{w-1} + \binom{w}{2} q^{w-2} + \dots + (-1)^{w-1} \binom{w}{w-1} q \right. \\ &\quad \left. - 1 + \binom{w}{1} - \binom{w}{2} + \dots + (-1)^w \binom{w}{w-1} \right) \\ &= \frac{\binom{k}{w}}{q - 1} \left((q - 1)^w - (-1)^w - (-1)^{w+1} \right) = \binom{k}{w} (q - 1)^{w-1}, \end{aligned}$$

where we have used that $(1 - 1)^w = 0 = \sum_{i=0}^w \binom{w}{i} (-1)^i$. This is precisely the number of codewords with hamming weight w up to multiples, and we have

$$\sum_{w=1}^k e_w^1 = \sum_{w=1}^k \binom{k}{w} (q - 1)^{w-1} = \frac{1}{q - 1} \left(\sum_{w=1}^k \binom{k}{w} (q - 1)^w \right) = \frac{((q - 1) + 1)^k - 1}{q - 1} = \frac{q^k - 1}{q - 1},$$

which is the number of nonzero vectors in \mathbb{F}_q^k , up to multiples.

If we go back to Algorithm 1, assuming m disjoint information sets, we may approximate the number of subspaces to be enumerated by

$$m \sum_{w=r}^{\lfloor d/m-1 \rfloor} e_w^r, \quad (2)$$

where this is obtained as the number of subspaces that we need to enumerate to get the lower bound $m(w + 1)$ to be greater than or equal to d . If we now approximate e_w^r by $\binom{k}{w} \begin{bmatrix} w \\ r \end{bmatrix}_q$, it is clear that the number of subspaces to be enumerated with this algorithm is, in general, much lower than that of the naive algorithm in which we enumerate $\begin{bmatrix} k \\ r \end{bmatrix}_q = \sum_{w=r}^k e_w^r$ subspaces.

In any case, this computation of e_w^r does not give a way to construct E_w^r in an efficient manner, since the algorithm derived from this enumeration would just compute all subspaces of dimension r with length w , and then remove those without full support. We show now one way to construct these subspaces efficiently.

We will first construct all the subspaces $E \subset \mathbb{F}_q^k$ with $\dim E = r$ and $\text{supp}(E) = \{1, \dots, w\}$. We can recover all the subspaces $E \subset \mathbb{F}_q^k$ with $\text{supp}(E) = \{j_1, \dots, j_w\}$ by mapping the coordinates $\{1, \dots, w\}$ to $\{j_1, \dots, j_w\}$. This can also be seen as adding columns of zeroes in appropriate positions. Thus, once we construct all the subspaces $E \subset \mathbb{F}_q^k$ with $\dim E = r$ and $\text{supp}(E) = \{1, \dots, w\}$, we can derive E_w^r .

Note that we can associate a unique matrix to each subspace $E \subset \mathbb{F}_q^k$, namely the reduced row echelon form of any matrix whose rows are a basis for E . We denote this matrix by $R(E)$. Thus, the problem we want to solve is equivalent to enumerating all the reduced row echelon forms with r rows and k columns, with rank r , and such that all the first w columns are nonzero and the last $k - w$ are zero columns.

Since the last $k - w$ columns are zero, without loss of generality we may think about matrices with w columns instead (or, in other words, we may assume $k = w$). Now we consider $i_1 = 1 < i_2 < \dots < i_r \leq w$. For each choice of (i_1, \dots, i_r) , we will construct the matrices $R(E)$ in reduced row echelon form whose pivots are in those positions. This determines r of the columns of $R(E)$, and also the “shape” of the matrix:

$$R(E) = \begin{pmatrix} 1 & * & \dots & * & 0 & * & \dots & * & 0 & * & \dots & 0 & * & \dots & * \\ 0 & 0 & \dots & 0 & 1 & * & \dots & * & 0 & * & \dots & 0 & * & \dots & * \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 1 & * & \dots & 0 & * & \dots & * \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 1 & * & \dots & * \end{pmatrix}$$

$i_1 \qquad \qquad \qquad i_2 \qquad \qquad \qquad i_3 \qquad \qquad \qquad i_r \qquad \qquad \qquad w$

Thus, we still have freedom to choose $w - r$ columns. For each $\ell \in \{1, \dots, w\} \setminus \{i_1, \dots, i_r\}$, if $i_z < \ell < i_{z+1}$ (we define $i_{r+1} := w + 1$), we have to choose a column $v_\ell \in \mathbb{F}_q^r$ with $1 \leq \text{wt}(v_\ell) \leq z$. Indeed, $1 \leq \text{wt}(v_\ell)$ since all the columns have to be nonzero to have $\text{supp}(E) = \{1, \dots, w\}$, and $\text{wt}(v_\ell) \leq z$ because of the shape of $R(E)$. Each choice of the $w - r$ columns v_ℓ gives a different matrix $R(E)$, which corresponds to a different subspace E . Similarly, the matrices constructed with different sets of pivots correspond to different subspaces. Thus, in this way we construct every subspace E with $\dim E = r$ and $\text{supp}(E) = \{1, \dots, w\}$, without repeating. The corresponding pseudo-code is given in Algorithm 3. As mentioned before, from this we can recover E_r^w by adding columns of zeroes in suitable positions.

To consider all possible columns v_ℓ , we need to construct all the vectors $v_\ell \in \mathbb{F}_q^r$ with $1 \leq \text{wt}(v_\ell) \leq z$. It is straightforward to obtain these vectors by constructing all $v_\ell \in \mathbb{F}_q^r$ with $\text{wt}(v_\ell) = y$, for any $1 \leq y \leq z$.

This can be done by first selecting the position of the y nonzero coordinates (there are $\binom{r}{y}$ possibilities), and then selecting an element of $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$ for each position ($(q - 1)^y$ possibilities). We now have all the tools we need to implement Algorithms 1 and 2 (and their relative version).

5 IMPLEMENTATION

For the implementation, we have chosen Sage [47]. Sage is a free and open-source mathematics software with a Python-based language. Since it is free, anyone will be able to use this implementation, and the fact that Sage is mostly based on Python makes the code easier to understand. This is particularly important since this is the first implementation of these algorithms and it can serve as the starting point for possible future advances in computing GHWs and RGHWS. The fact that Sage includes functions and classes for all the objects we use, e.g., linear codes, also facilitates the understanding of the implementation. Additionally, this implementation aims to be as general as possible, working over any finite field and for any linear code. In particular, this means we need to use mathematical software with a good implementation of finite fields, which heavily restricts the options. The other two options to be considered in this regard would be Magma [5], which is not open-source, and GAP [46], which could be an alternative for Sage in this case, but it uses its own programming language.

Algorithm 3 Construction of $R(E)$, for all $E \subset \mathbb{F}_q^k$ with $\dim E = r$ and $\text{supp}(E) = \{1, \dots, w\}$

Input: q, k, w, r .

Output: List L of $R(E)$ for all $E \subset \mathbb{F}_q^k$ with $\dim E = r$ and $\text{supp}(E) = \{1, \dots, w\}$.

```

1:  $L = []$ 
2: for  $1 = i_1 < i_2 < \dots < i_r \leq w$  do
3:   for  $\ell \in \{1, \dots, w\} \setminus \{i_1, \dots, i_r\}$  do
4:     if  $i_z < \ell < i_{z+1}$  then
5:        $\text{cols}[\ell] = \{v \in \mathbb{F}_q^r, 1 \leq \text{wt}(v) \leq z\}$ 
6:     end if
7:   end for
8:   Define cart as the Cartesian product of the sets in cols. Assume the entries of cart are still indexed
   by  $\ell \in \{1, \dots, w\} \setminus \{i_1, \dots, i_r\}$ .
9:   for  $u \in \text{cart do}$ 
10:    for  $1 \leq \ell \leq w$  do
11:      if  $\ell = i_z$  for some  $1 \leq z \leq r$  then
12:         $M[\ell] = (0, \dots, 0, 1, 0, \dots, 0)$ , where the 1 is in position  $z$ .
13:      else
14:         $M[\ell] = u[\ell]$ 
15:      end if
16:    end for
17:    Append the matrix whose columns are given by  $M$  to  $L$ .
18:  end for
19: end for
20: return  $L$ 

```

The package GHWS includes several auxiliary functions and also some main functions. The auxiliary functions include the function `information`, which computes the required information sets for Algorithms 1 and 2, as well as the corresponding generator matrices and redundancies. This uses the procedure explained in Section 3, and since this is also used in the usual Brouwer-Zimmermann algorithm, we shall not explain it further (see [17]). In any case, the functions relying on `information` have an optional argument to provide alternative information sets and generator matrices if the user wants to compute them in a different way. We also include some auxiliary functions to check if a code is cyclic and to compute its BCH bound, and some small functions required for the main algorithms. All the functions included have a description text explaining what they do and providing examples.

As main functions we have `GHW`, `hierarchy`, `RGHW` and `rhierarchy`, which compute the GHWS, hierarchy, RGHWs, and relative hierarchy, of a code (or a pair codes in the relative case). For the case of `GHW` (the others are analogous), we first check if the code is cyclic to make use of the improvements related to cyclic codes (see Subsection 5.1 below), and we compute the information sets and generator matrices using `information` otherwise. Then we proceed as in Algorithm 1, but at the beginning of each iteration we make a prediction of the expected w_0 required to finish, meaning that $E_{w_0}^r$ would be the last set we expect to enumerate. We use this to remove unnecessary generator matrices if the lower bound at the end of this step will be much higher than the current upper bound (see Subsection 5.1 below). Then we compute $R(E)$ for all subspaces E of \mathbb{F}_q^w with $|\text{supp}(E)| = w$ and $\dim E = r$ using `subspaces(r, w, w, K)` (in this case, $K = \mathbb{F}_q$). This follows the idea of Section 4, and the function `subspaces` computes all the required

$R(E)$ by constructing the reduced row echelon forms as in Algorithm 3. We store all these matrices, since we use them to construct all the $R(E)$, for $E \in E_w^r$ by appropriately adding columns of zero's. In some cases, this may require a substantial RAM usage (for example, if q is large). We provide an alternative implementation of these functions without storing these matrices (GHW_low_mem, hierarchy_low_mem, RGHW_low_mem and rhierarchy_low_mem). However, this is usually slower (see Section 6) since we repeat some of the operations required for the construction of E_w^r several times. After this, for each $E \in E_w^r$, we compute $\text{enc}_{G_i}(E)$, for every generator matrix G_i that contributes to the lower bound, and update the upper bound if the cardinality of the support is lower than the current upper bound. Finally, we compute the new lower bound at the end of the iteration.

For the case of RGHW and rhierarchy (and their low memory version), we add an extra step: if $|\text{supp}(\text{enc}_{G_i}(E))|$ is lower than the current upper bound, then we check if $\text{enc}_{G_i}(E) \cap C_2 = \{0\}$, and we only update the value of the upper bound if that is the case, which is clearly more efficient than checking the condition for every subspace that we consider. To check that $\text{enc}_{G_i}(E) \cap C_2 = \{0\}$, Sage has classes for vector spaces, and we can create the corresponding vector spaces and compute the dimension of the intersection. However, to avoid making these conversions, we note that checking $\dim(\text{enc}_{G_i}(E) \cap C_2) = 0$ is equivalent to checking $\text{enc}_{G_i}(E) \cap C_2 = \{0\}$, and we use instead

$$\dim(\text{enc}_{G_i}(E) \cap C_2) = r - \text{rank}(H_2(R(E)G_i)^t),$$

which can be derived from [1, Prop. 2.2] (H_2 is a parity check matrix of C_2 , that is, a generator matrix of C_2^\perp).

Finally, we have also included the function `higher_spectrum` to compute the higher weight spectrum, and the function `rhhigher_spectrum` to compute the relative higher weight spectrum. For these functions, one necessarily has to enumerate all the subspaces of the code, so there is no improvement in the enumeration as in the Brouwer-Zimmermann-like algorithm, but we have included it since it can be directly computed using the function `subspaces`. For the low memory version, we proceed similarly to what we did with `GHW_low_mem`.

The functions `GHW`, `hierarchy`, `RGHW`, `rhierarchy`, `higher_spectrum`, `rhhigher_spectrum`, and their low memory version, have a verbose argument, which, if set to `True`, it shows real time information about the computation. This is recommended for heavy computations, since it can give an idea of whether a computation will finish within a reasonable time or not.

5.1 Additional considerations

Although our main focus is to compute the GHWs and RGHWs of a random linear code, we have included some improvements for cyclic codes, since they are straightforward to implement. One can check how those improvements work, for instance, in [17] (it is analogous to the case of the minimum distance). In the same reference, one can also find that in some cases we might not use some of the additional generator matrices we are considering. For example, if $w + 1 - R_j \leq 0$ for some j in Algorithm 1, that matrix does not improve the lower bound and we may skip it. Moreover, we can predict what the value of the lower bound is going to be at the end of the iteration, and if the value surpasses the current upper bound by a sufficiently large margin, we may also skip some matrices while still ensuring we finish the algorithm in this iteration (since the lower bound can still be greater than or equal to the current upper bound). These improvements for the last iteration of the algorithm have a great impact, since it is usually the most computationally expensive step. For example, see [21, Lem. 2], in which it is shown that in some cases this iteration is more expensive than all the previous ones, for $r = 1$ and $q = 2$.

Due to Wei's duality from Theorem 2.5, computing the weight hierarchy of a code is equivalent to computing the weight hierarchy of its dual, and we have the function `wei_duality` to compute this. In

many cases it will be more efficient to compute the weight hierarchy of the dual of a code and then use Wei's duality than computing it directly. Technically, we do not know in advance exactly how many computations will be needed. However, since the size of e_w^r is essentially given by $\binom{n}{w} \binom{w}{r}_q$, for $1 \leq r \leq k$ and $r \leq w \leq k$, a good strategy would be to compute the weight hierarchy directly if $k \leq n/2$, and compute the weight hierarchy of the dual if $k > n/2$.

6 TESTS

For the results presented in this section, we have used Arch Linux on a computer with Windows 10 using Windows Subsystem for Linux (WSL). The virtual machine has access to 7.72GB of RAM and an Intel Core i7-6700 processor (3.4GHz, with Turbo Boost disabled). With respect to Sage, we have used the 10.6 version. We also had access to a server with an AMD EPYC 7F52 processor (3.5GHz, up to 3.9GHz with boost), with Ubuntu 20.04.6 LTS and SageMath 10.6, obtaining similar results. We provide several test files to replicate our results. For example, for the test file `test_GHWs.sage`, the corresponding test is ran by writing `sage test_GHWs.sage`.

6.1 Correctness

Since this is a new algorithm, the first thing to test is the correctness of the results. For this, we may consider families of codes with known GHWs and RGHWs, and check that our algorithms return the correct values. By Remark 2.6, the weight hierarchy of MDS codes is known, and their higher weight spectra has been studied in [11]. The most well-known class of MDS codes is given by Reed-Solomon (RS) codes. For a given q and a given dimension k , the corresponding RS code is denoted by $RS_q(k)$ and has parameters $[q, k, q - k + 1]$. These codes require working over large finite fields due to the limitation on the length, requiring a substantial number of operations for the algorithms to finish.

Another family with known weight hierarchy are Reed-Muller (RM) codes [20, 29], and their RGHWs have also been studied in [13]. For a given integer $m > 0$ and a degree $0 \leq v < q$, the corresponding RM code is denoted by $RM_q(v, m)$, and has parameters $\left[q^m, \binom{d+m}{m}, (q-d)q^{m-1} \right]$ (for $v \geq q$, their parameters are also known but are more complicated). Both RS and binary RM codes are directly implemented in Sage, as well as q -ary RM codes for $v < q$. The advantage of RM codes in this case is that we can have longer codes over a smaller finite field. Similarly, cyclic codes are implemented in Sage, some of them have known GHWs [23], and they can provide long codes over small finite field sizes. In the test file `test_GHWs.sage`, we have included several tests to check that our implementation correctly gives the GHWs of RS, RM and cyclic codes. If no error is returned, the test has been successfully passed, and this can be used to also check that all the functions are working properly (a similar test can be ran with the test file `test_GHWs_low_mem.sage` for the low memory functions).

In particular, these tests check the values returned by our main functions compared with the known GHWs of some binary Reed-Muller codes, the RGHWs of some q -ary Reed-Muller codes (with $q = 5$), the GHWs of some binary cyclic codes, and the GHWs, Wei's duality from Theorem 2.5, higher spectra and relative higher spectra of some Reed-Solomon codes. The test `test_GHWs.sage` takes 226.27s on average on our machine, and the test `test_GHWs_low_mem.sage` takes 406.06s. This already shows that the low memory implementation can be slower, and we will explore this later.

6.2 Performance analysis

In terms of performance, since there are no alternatives currently, we cannot compare with other implementations. The only test that we can make in this direction is to compare with the naive approach of enumerating all the subspaces. The most naive approach would just consider all $r \times k$ matrices over \mathbb{F}_q^k with rank r and then compute the minimum support of the resulting matrices after multiplying by G , a generator matrix of C . Instead of this, we can use our function `subspaces`, which will not repeat subspaces (as the previous method would do, since there are several matrices whose row span is the same). However, compared to our Brouwer-Zimmermann-like approach, this naive algorithm may require to compute many more subspaces (recall (2)). This becomes problematic in terms of RAM for medium to large finite field sizes. Thus, in the test file `test_change_r` we have included the function `naive_GHW` (using subspaces and requiring more RAM), and a version in the spirit of `GHW_low_mem`, called `naive_GHW_low_mem`, which requires less RAM. For our tests, we shall only use `naive_GHW_low_mem`, and, to make a fair comparison, we compare with `GHW_low_mem` (in the test files `test_change_k.sage` and `test_change_q.sage` we only include `naive_GHW_low_mem`). Also, to avoid having to handle lists of generator matrices of random codes, we will use families of codes that are implemented in Sage, as before. Experiments with random codes show similar performance for our functions.

First, with `test_change_r.sage` we have compared the times required for computing the GHWs of RM codes with $q = 5, 7$, and $r = 2, 3, 4, 5$, using `GHW_low_mem` and `naive_GHW_low_mem`. The results are given in Tables 1 and 2. We see that the difference in performance is substantial (in some cases there are two orders of magnitude of difference). This will also be the case in many of the subsequent tests, which is why we have chosen tables instead of plots to present most of our results. Since both implementations are in Sage, the difference in time represents the advantage gained by using our Brouwer-Zimmermann-like algorithm with respect to just enumerating all subspaces. All these codes have the same dimension $k = 6$, and for a fixed $q = 5$ or $q = 7$, we only change r . With `test_change_k.sage`, we study the dependence on k by fixing $r = 2$ and $q = 5$, and computing the second GHW of certain subcodes of the code $\text{RM}_5(3, 2)$ with different dimensions k . The results are presented in Table 3. We see that, although the time required for both functions increases rapidly, the increase on time required for `naive_GHW_low_mem` to finish is much larger than that of `GHW_low_mem`. We also note that, for very low values of k , the naive approach can be faster since in that case using several generator matrices may not be beneficial. But these cases are not our main focus, as both functions will complete in a short amount of time. Finally, with `test_change_q.sage`, we consider the dual Hamming code with redundancy 5. This code is a binary code, which means it has a generator matrix given by zeroes and ones. We can consider this matrix as the generator matrix of a code over any finite field, and all the resulting codes will have the same length and dimension. We have computed the second GHW for these codes, for $q = 2, 3, 4, 5, 7, 8$, and the results are shown in Table 4. Clearly, the size of the field seems to impact the performance of the naive approach much more than the performance of `GHW_low_mem`. We have also represented the results of Tables 3 and 4 in Figure 1. Note that we are using a logarithmic scale for the y axis due to the exponential difference in time (and exponential growth of the time required to finish).

Table 1. Time in seconds to compute $d_r(\text{RM}_5(2, 2))$, $2 \leq r \leq 5$.

Function	$d_2(\text{RM}_5(2, 2))$	$d_3(\text{RM}_5(2, 2))$	$d_4(\text{RM}_5(2, 2))$	$d_5(\text{RM}_5(2, 2))$
<code>GHW_low_mem</code>	3.840	1.165	2.564	0.01116
<code>naive_GHW_low_mem</code>	89.840	577.613	139.910	1.55701

Table 2. Time in seconds to compute $d_r(\text{RM}_7(2, 2))$, $2 \leq r \leq 5$.

Function	$d_2(\text{RM}_7(2, 2))$	$d_3(\text{RM}_7(2, 2))$	$d_4(\text{RM}_7(2, 2))$	$d_5(\text{RM}_7(2, 2))$
GHW_low_mem	517.4	714.6	20.08	0.03101
naive_GHW_low_mem	1,342.5	12,519.0	2,287.91	11.90103

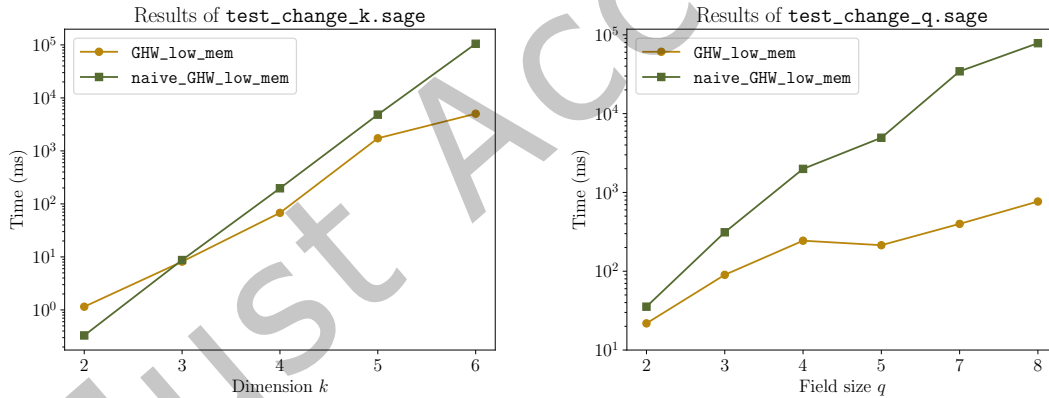
Table 3. Time in milliseconds to compute $d_2(C)$, where $C \subset \text{RM}_5(3, 2)$ with $\dim C = k$ (the definition of C can be found in `test_change_k.sage`).

Function	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
GHW_low_mem	1.15	8.15	67.81	1,731.14	5,031.54
naive_GHW_low_mem	0.33	8.73	197.21	4,831.71	105,041.21

Table 4. Time in milliseconds to compute $d_2(C)$, where C is the Hamming code with redundancy 5 over \mathbb{F}_2 , seen as a code over \mathbb{F}_q , for $q = 2, 3, 4, 5, 7, 8$.

Function	$q = 2$	$q = 3$	$q = 4$	$q = 5$	$q = 7$	$q = 8$
GHW_low_mem	21.8	90.1	244.5	214.3	399.6	766.7
naive_GHW_low_mem	35.5	312.7	1,991.3	4,943.4	34,423.0	78,320.1

Fig. 1. Semi-logarithmic plots of Tables 3 and 4.

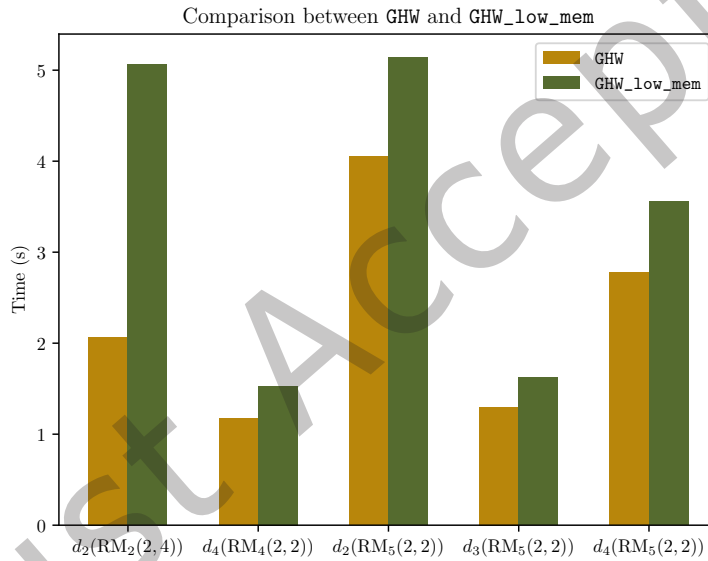


Overall, we see that our algorithm performs much better than the naive approach. Note that for the naive approach we are also using part of our code to construct all subspaces without repeating (Section 4). If we do not use these ideas, and we just consider all $r \times k$ matrices over \mathbb{F}_q^k with rank r (what we called before “the most naive approach”), since every subspace of dimension r has $(q^r - 1)(q^r - q) \cdots (q^r - q^{r-1})$ different bases, we would be multiplying the number of operations by this number (we would consider different matrices with the same row span), making the computation of the GHWs unfeasible in most nontrivial cases.

We study now the performance of GHW with respect to `GHW_low_mem` (similar results are obtained when comparing `hierarchy` with `hierarchy_low_mem`, etc). We have already explained theoretically that, in

general, GHW should be faster, at the cost of requiring more RAM, and we have seen with `test_GHWs.sage` and `test_GHWs_low_mem.sage` that this is indeed the case. With `test_normal_low_mem.sage`, we check the speed of both functions for some RM codes of degree d , for different field sizes and values of r . The results are shown in Figure 2. The difference in performance varies, but in general GHW is faster than `GHW_low_mem`. With respect to RAM, `GHW_low_mem` uses a negligible amount of memory besides the memory required to load Sage, which on our machine was about 2.9% of the total RAM (223.88MB). In Figure 3 we show the peak RAM usage of GHW while computing the GHWs of several RM codes (for low values of q , the memory usage was negligible). For $r = 2, 3$, which are the values for which the list of subspaces required is larger, we see that the RAM requirement can be noticeable when we increase the finite field size q . In most of our experiments, time became an issue before RAM did, in the sense that the cases with a large RAM usage were not expected to finish in a reasonable amount of time, but this will depend on the relation between available RAM and processing power of each machine.

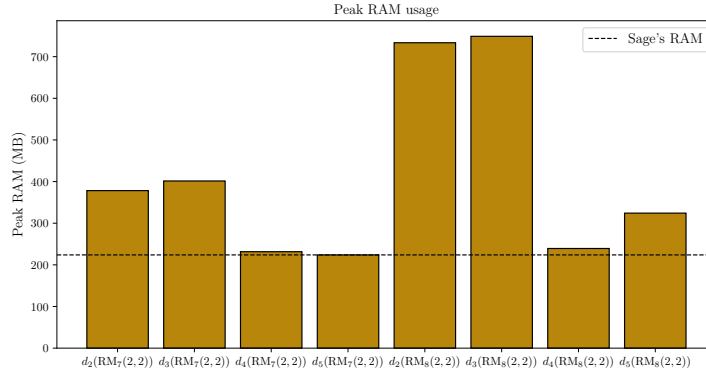
Fig. 2. Time in seconds to compute the GHWs of some RM codes using GHW and `GHW_low_mem`.



We can also make a comparison between GHW and hierarchy for computing the weight hierarchy of a linear code (similarly for the relative version). The only difference between using $\text{GHW}(C, r)$, for $1 \leq r \leq k$, and $\text{hierarchy}(C)$, is that hierarchy can use the value $d_r(C) + 1$ as a lower bound for $d_{r+1}(C)$. Depending on the code, this can be a negligible improvement or a very important one. The most extreme case is given by MDS codes, since hierarchy will end after computing $d_1(C)$ because at that point the generalized Singleton bound from Corollary 2.4 will meet the lower bound for every $2 \leq r \leq k$. We show this in Table 5, where we compute the weight hierarchy of $\text{RS}_{13}(k)$ for different values of k . For other families of codes, the improvement will not be as large, as we see in Table 6, where we have used RM codes instead. These tests can be found in `test_hierarchy_ghw.sage`.

Since our functions are completely general, we do not expect them to be competitive for the case of the minimum distance ($r = 1$) where additional improvements can be considered [21, 22, 39]. Moreover, if one restricts the field size to $q = 2, 3$, then it is possible to use lower level programming languages,

Fig. 3. Peak RAM usage of GHW for several RM codes.

Table 5. Time in milliseconds to compute the weight hierarchy of $\text{RS}_{13}(k)$, for $1 \leq k \leq 12$.

Function	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
hierarchy	0.557	0.846	6.60	11.21	173.2	323.6
GHW	0.559	1.654	9.77	138.8	452.3	6751.9
	$k = 7$	$k = 8$	$k = 9$	$k = 10$	$k = 11$	$k = 12$
hierarchy	547.7	891.2	48.5	53.8	3.54	3.61
GHW	14,803.4	29,319.4	47,821.7	3,061.1	463.5	1,001.0

Table 6. Time in milliseconds to compute the weight hierarchy of some RM codes.

Function	$\text{RM}_2(2,3)$	$\text{RM}_3(2,2)$	$\text{RM}_4(2,2)$	$\text{RM}_5(2,2)$
hierarchy	1.99	37.77	2,770.1	5,634.0
GHW	22.31	153.41	3,443.3	6,933.2

which will result in a better performance. Notwithstanding the foregoing, in our experiments $\text{GHW}(C, 1)$ has performed better than the Sage function `C.minimum_distance()` for higher values of k , although it is slower for lower values of k . We will not analyze the case of $r = 1$ further, since our main objective is to implement the algorithm in the most general form for any r , but the fact that our implementation outperforms specialized implementations for the case $r = 1$ in some cases is an indication of its efficiency.

7 CONCLUSION

In this work we have introduced a generalization of the Brouwer-Zimmermann algorithm for computing the GHWs of a linear code, which required studying how to construct all subspaces with a given support. The resulting algorithm has been implemented in Sage, and we have shown that this implementation is much more efficient than computing the GHWs using the definition.

In the future, we will study improvements for particular cases. The case of $q = 2$ is particularly interesting since we may use a lower level programming language, like C, and the construction of all the subspaces may be optimized due to working with such a small finite field size. We may also consider particular values of r , such as $r = 2$, which, as we have mentioned earlier, appears in some additional applications, e.g., determining the error-correction capability of quantum codes derived via Steane enlargement [19].

REFERENCES

- [1] Sarah E. Anderson, Eduardo Camps-Moreno, Hiram H. López, Gretchen L. Matthews, Diego Ruano, and Ivan Soprunov. 2024. Relative Hulls and Quantum Codes. *IEEE Trans. Inform. Theory* 70, 5 (2024), 3190–3201. <https://doi.org/10.1109/TIT.2024.3373550>
- [2] Angela I. Barbero and Carlos Munuera. 2000. The weight hierarchy of Hermitian codes. *SIAM J. Discrete Math.* 13, 1 (2000), 79–104. <https://doi.org/10.1137/S089548019834342X>
- [3] Peter Beelen and Mrinmoy Datta. 2018. Generalized Hamming weights of affine Cartesian codes. *Finite Fields Appl.* 51 (2018), 130–145. <https://doi.org/10.1016/j.ffa.2018.01.006>
- [4] Peter Beelen, Mrinmoy Datta, and Sudhir R. Ghorpade. 2022. A combinatorial approach to the number of solutions of systems of homogeneous polynomial equations over finite fields. *Mosc. Math. J.* 22, 4 (2022), 565–593. <https://doi.org/10.17323/1609-4514-2022-22-4-565-593>
- [5] Wieb Bosma, John Cannon, and Catherine Playoust. 1997. The Magma algebra system. I. The user language. *J. Symbolic Comput.* 24, 3–4 (1997), 235–265. <https://doi.org/10.1006/jsco.1996.0125> Computational algebra and number theory (London, 1993).
- [6] A. R. Calderbank and Peter W. Shor. 1996. Good quantum error-correcting codes exist. *Phys. Rev. A* 54 (Aug 1996), 1098–1105. Issue 2. <https://doi.org/10.1103/PhysRevA.54.1098>
- [7] Eduardo Camps-Moreno, Ignacio García-Marco, Hiram H. López, Irene Márquez-Corbella, Edgar Martínez-Moro, and Eliseo Sarmiento. 2024. On the generalized Hamming weights of hyperbolic codes. *J. Algebra Appl.* 23, 7 (2024), Paper No. 2550062, 18. <https://doi.org/10.1142/S0219498825500628>
- [8] Eduardo Camps-Moreno, Hiram H. López, Gretchen L. Matthews, and Rodrigo San-José. 2025. The weight hierarchy of decreasing norm-trace codes. *Des. Codes Cryptogr.* 93, 7 (2025), 2873–2894. <https://doi.org/10.1007/s10623-025-01619-7>
- [9] Jasper Cramwinkel, Erik Roijackers, Reinald Baart, Eric Minkes, Lea Ruscio, and David Joyner. [n. d.]. GUAVA, a GAP package for computing with error-correcting codes. Available at <http://www.gap-system.org/Packages/guava.html>.
- [10] Mrinmoy Datta. 2020. Relative generalized Hamming weights of affine Cartesian codes. *Des. Codes Cryptogr.* 88, 6 (2020), 1273–1284. <https://doi.org/10.1007/s10623-020-00745-8>
- [11] Steven T. Dougherty and Sunghyu Han. 2010. Higher weights and generalized MDS codes. *J. Korean Math. Soc.* 47, 6 (2010), 1167–1182. <https://doi.org/10.4134/JKMS.2010.47.6.1167>
- [12] G. L. Feng, K. K. Tzeng, and V. K. Wei. 1992. On the generalized Hamming weights of several classes of cyclic codes. *IEEE Trans. Inform. Theory* 38, 3 (1992), 1125–1130. <https://doi.org/10.1109/18.135653>
- [13] Olav Geil and Stefano Martin. 2017. Relative generalized Hamming weights of q -ary Reed-Muller codes. *Adv. Math. Commun.* 11, 3 (2017), 503–531. <https://doi.org/10.3934/amc.2017041>
- [14] Olav Geil, Stefano Martin, Ryutaroh Matsumoto, Diego Ruano, and Yuan Luo. 2014. Relative generalized Hamming weights of one-point algebraic geometric codes. *IEEE Trans. Inform. Theory* 60, 10 (2014), 5938–5949. <https://doi.org/10.1109/TIT.2014.2345375>
- [15] Sudhir R. Ghorpade, Trygve Johnsen, Rati Ludhani, and Rakhi Pratihar. 2024. Higher weight spectra and Betti numbers of Reed-Muller codes $RM_q(2, 2)$. *ArXiv 2408.02548* (2024). arXiv:2408.02548 [cs.IT]
- [16] Parikshit Gopalan, Venkatesan Guruswami, and Prasad Raghavendra. 2011. List decoding tensor products and interleaved codes. *SIAM J. Comput.* 40, 5 (2011), 1432–1462. <https://doi.org/10.1137/090778274>
- [17] Markus Grassl. 2006. Searching for linear codes with large minimum distance. In *Discovering mathematics with Magma*. Algorithms Comput. Math., Vol. 19. Springer, Berlin, 287–313. https://doi.org/10.1007/978-3-540-37634-7_13
- [18] Venkatesan Guruswami. 2003. List decoding from erasures: bounds and code constructions. *IEEE Trans. Inform. Theory* 49, 11 (2003), 2826–2833. <https://doi.org/10.1109/TIT.2003.815776>
- [19] Mitsuru Hamada. 2008. Concatenated quantum codes constructible in polynomial time: efficient decoding and error correction. *IEEE Trans. Inform. Theory* 54, 12 (2008), 5689–5704. <https://doi.org/10.1109/TIT.2008.2006416>
- [20] Petra Heijnen and Ruud Pellikaan. 1998. Generalized Hamming weights of q -ary Reed-Muller codes. *IEEE Trans. Inform. Theory* 44, 1 (1998), 181–196. <https://doi.org/10.1109/18.651015>
- [21] Fernando Hernando, Francisco D. Igual, and Gregorio Quintana-Ortí. 2019. Algorithm 994: fast implementations of the Brouwer-Zimmermann algorithm for the computation of the minimum distance of a random linear code. *ACM Trans. Math. Software* 45, 2 (2019), Art. 23, 28. <https://doi.org/10.1145/3302389>
- [22] Fernando Hernando, Gregorio Quintana-Ortí, and Markus Grassl. 2024. Fast Algorithms and Implementations for Computing the Minimum Distance of Quantum Codes. *ArXiv 2408.10743* (2024). arXiv:2408.10743 [cs.IT]
- [23] H. Janwa and A. K. Lal. 1997. On the generalized Hamming weights of cyclic codes. *IEEE Trans. Inform. Theory* 43, 1 (1997), 299–308. <https://doi.org/10.1109/18.567719>
- [24] Trygve Johnsen and Hugues Verdure. 2013. Hamming weights and Betti numbers of Stanley-Reisner rings associated to matroids. *Appl. Algebra Engrg. Comm. Comput.* 24, 1 (2013), 73–93. <https://doi.org/10.1007/s00200-012-0183-7>
- [25] Trygve Johnsen and Hugues Verdure. 2020. Higher weight spectra of Veronese codes. *IEEE Trans. Inform. Theory* 66, 6 (2020), 3538–3546. <https://doi.org/10.1109/TIT.2019.2948180>

- [26] Trygve Johnsen and Hugues Verdone. 2021. Higher weight spectra of codes from Veronese threefolds. *J. Pure Appl. Algebra* 225, 7 (2021), Paper No. 106609, 11. <https://doi.org/10.1016/j.jpaa.2020.106609>
- [27] Krishna Kaipa and Puspendu Pradhan. 2024. Higher weight spectra of ternary codes associated to the quadratic Veronese 3-fold. *ArXiv 2405.12011* (2024). arXiv:2405.12011 [cs.IT]
- [28] Nathan Kaplan and Vlad Matei. 2021. Counting plane cubic curves over finite fields with a prescribed number of rational intersection points. *Eur. J. Math.* 7, 3 (2021), 1137–1181. <https://doi.org/10.1007/s40879-021-00472-x>
- [29] Tadao Kasami, Shu Lin, and W. Wesley Peterson. 1968. New generalizations of the Reed-Muller codes. I. Primitive codes. *IEEE Trans. Inform. Theory* IT-14 (1968), 189–199. <https://doi.org/10.1109/tit.1968.1054127>
- [30] Avanti Ketkar, Andreas Klappenecker, Santosh Kumar, and Pradeep Kiran Sarvepalli. 2006. Nonbinary stabilizer codes over finite fields. *IEEE Trans. Inform. Theory* 52, 11 (2006), 4892–4914. <https://doi.org/10.1109/TIT.2006.883612>
- [31] Jun Kurihara, Tomohiko Uyematsu, and Ryutaroh Matsumoto. 2012. Secret Sharing Schemes Based on Linear Codes Can Be Precisely Characterized by the Relative Generalized Hamming Weight. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* E95.A, 11 (2012), 2067–2075. <https://doi.org/10.1587/transfun.E95.A.2067>
- [32] Yuan Luo, Chaichana Mitrpant, A. J. Han Vinck, and Kefei Chen. 2005. Some new characters on the wire-tap channel of type II. *IEEE Trans. Inform. Theory* 51, 3 (2005), 1222–1229. <https://doi.org/10.1109/TIT.2004.842763>
- [33] F. J. MacWilliams and N. J. A. Sloane. 1977. *The theory of error-correcting codes*. North-Holland Publishing Co., Amsterdam-New York-Oxford.
- [34] Ernst W. Mayr. 1997. Some complexity results for polynomial ideals. *J. Complexity* 13, 3 (1997), 303–325. <https://doi.org/10.1006/jcom.1997.0447>
- [35] Ernst W. Mayr and Albert R. Meyer. 1982. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. in Math.* 46, 3 (1982), 305–329. [https://doi.org/10.1016/0001-8708\(82\)90048-2](https://doi.org/10.1016/0001-8708(82)90048-2)
- [36] Robert J McEliece. 1978. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF. A public-key cryptosystem based on algebraic. *JPL DSN Progress Report 42-44* (1978). https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.
- [37] H. Niederreiter. 1986. Knapsack-type cryptosystems and algebraic coding theory. *Problems Control Inform. Theory/Problemy Upravlen. Teor. Inform.* 15, 2 (1986), 159–166.
- [38] Dimitris S. Papailiopoulos and Alexandros G. Dimakis. 2014. Locally repairable codes. *IEEE Trans. Inform. Theory* 60, 10 (2014), 5843–5855. <https://doi.org/10.1109/TIT.2014.2325570>
- [39] Gregorio Quintana-Ortí, Fernando Hernando, and Francisco D. Igual. 2023. Algorithm 1033: parallel implementations for computing the minimum distance of a random linear code on distributed-memory architectures. *ACM Trans. Math. Software* 49, 1 (2023), Art. 8, 24. <https://doi.org/10.1145/3573383>
- [40] Rodrigo San-José. 2024. A recursive construction for projective Reed-Muller codes. *IEEE Trans. Inform. Theory* 70, 12 (2024), 8511–8523. <https://doi.org/10.1109/TIT.2024.3483998>
- [41] Rodrigo San-José. 2025. About the generalized Hamming weights of matrix-product codes. *Comput. Appl. Math.* 44, 4 (2025), Paper No. 186. <https://doi.org/10.1007/s40314-025-03141-x>
- [42] Rodrigo San-José. 2025. GHWs: A Sage package for computing the generalized Hamming weights of a linear code. GitHub repository. Available online: <https://github.com/RodrigoSanJose/GHWs>.
- [43] Frank-Olaf Schreyer. 1986. Syzygies of canonical curves and special linear series. *Math. Ann.* 275, 1 (1986), 105–137. <https://doi.org/10.1007/BF01458587>
- [44] Andrew Steane. 1996. Multiple-Particle Interference and Quantum Error Correction. *Proceedings of the Royal Society of London Series A* 452, 1954 (Nov. 1996), 2551–2577. <https://doi.org/10.1098/rspa.1996.0136> arXiv:quant-ph/9601029 [quant-ph]
- [45] Andrew M. Steane. 1999. Enlargement of Calderbank-Shor-Steane quantum codes. *IEEE Trans. Inform. Theory* 45, 7 (1999), 2492–2495. <https://doi.org/10.1109/18.796388>
- [46] The GAP Group 2024. *GAP – Groups, Algorithms, and Programming, Version 4.14.0*. The GAP Group. <https://www.gap-system.org>
- [47] The Sage Developers. 2025. *SageMath, the Sage Mathematics Software System (Version 10.6)*. <https://www.sagemath.org>.
- [48] Alexander Vardy. 1997. The intractability of computing the minimum distance of a code. *IEEE Trans. Inform. Theory* 43, 6 (1997), 1757–1766. <https://doi.org/10.1109/18.641542>
- [49] Victor K. Wei. 1991. Generalized Hamming weights for linear codes. *IEEE Trans. Inform. Theory* 37, 5 (1991), 1412–1418. <https://doi.org/10.1109/18.133259>
- [50] Maosheng Xiong, Shuxing Li, and Gennian Ge. 2016. The weight hierarchy of some reducible cyclic codes. *IEEE Trans. Inform. Theory* 62, 7 (2016), 4071–4080. <https://doi.org/10.1109/TIT.2016.2570236>
- [51] Kyeongcheol Yang, P. Vijay Kumar, and Henning Stichtenoth. 1994. On the weight hierarchy of geometric Goppa codes. *IEEE Trans. Inform. Theory* 40, 3 (1994), 913–920. <https://doi.org/10.1109/18.335903>
- [52] Minghui Yang, Jin Li, Keqin Feng, and Dongdai Lin. 2015. Generalized Hamming weights of irreducible cyclic codes. *IEEE Trans. Inform. Theory* 61, 9 (2015), 4905–4913. <https://doi.org/10.1109/TIT.2015.2444013>

- [53] Jun Zhang and Keqin Feng. 2018. Relative generalized Hamming weights of cyclic codes. *Finite Fields Appl.* 50 (2018), 338–355. <https://doi.org/10.1016/j.ffa.2017.12.008>
- [54] Karl-Heinz Zimmermann. 1996. Integral Hecke modules, integral generalized Reed-Muller codes, and linear codes. *Tech. Rep.* 3-96, Technische Universität Hamburg– Harburg (1996).