Advances in Mathematics of Communications
Volume 10, No. 1, 2016, 195–207
doi:10.3934/amc.2016.10.195

ALGORITHMS FOR THE MINIMUM WEIGHT OF LINEAR CODES

Petr Lisoněk and Layla Trummer

Department of Mathematics
Simon Fraser University
Burnaby, BC, V5A 1S6, Canada

(Communicated by Marcus Greferath)

Abstract. We outline the algorithm for computing the minimum weight of a linear code over a finite field that was invented by A. Brouwer and later extended by K.-H. Zimmermann. We show that matroid partitioning algorithms can be used to efficiently find a favourable (and sometimes best possible) sequence of information sets on which the Brouwer-Zimmermann algorithm operates. We present a new algorithm for computing the minimum weight of a linear code. We use a large set of codes to compare our new algorithm with the Brouwer-Zimmermann algorithm. We find that for about one third of codes in this sample set, our algorithm requires to generate fewer codewords than the Brouwer-Zimmermann algorithm.

1. Introduction

For a prime power q let $\mathbb{F}_q$ denote the field with q elements. We assume that q is small, hence all arithmetic operations in $\mathbb{F}_q$ are performed at unit cost. For $x \in \mathbb{F}_q^n$ let $\mathrm{wt}(x)$ denote the Hamming weight of x. By an $[n,k]_q$ linear code we mean a k-dimensional subspace of $\mathbb{F}_q^n$. By an $[n,k,d]_q$ linear code C we mean an $[n,k]_q$ linear code C such that the minimum Hamming distance of distinct codewords in C is d. Then d is also the minimum Hamming weight of non-zero codewords in C, denoted $\mathrm{wt}(C)$. Under the standard definitions of coding theory [1, 10] the code C can detect up to $d-1$ errors and it can correct up to $\lfloor (d-1)/2 \rfloor$ errors. Thus determining the value of d is critical for understanding of the error detection/correction capability of C.

Vardy [12] showed that for general binary linear codes, computing the minimum weight is an NP-hard problem, and the corresponding decision problem is NP-complete. Hence any general algorithm for computing the minimum weight will run in superpolynomial time, unless P=NP.

The first part of this paper is concerned with the algorithm for computing the minimum weight of a linear code that was invented by A. Brouwer and later extended by K.-H. Zimmermann. This algorithm is outlined in Section 3. Before that, in Section 2 we review background from matroid theory. In Section 4 we propose an extension to the Brouwer-Zimmermann algorithm which consists in an efficient construction of a good (sometimes best possible) sequence of information

2010 *Mathematics Subject Classification:* Primary: 94B05; Secondary: 05B35.

Key words and phrases: Linear code, minimum weight algorithm, Brouwer-Zimmermann algorithm, information set, matroid partition.

Research of both authors was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

sets for the given code. We compare our approach to the previous literature on the Brouwer-Zimmermann algorithm. Throughout the paper we also make references to the implementation of the Brouwer-Zimmermann algorithm that is available in Magma [3].

In the second part of the paper (Section 5) we propose a new algorithm for computing the minimum weight of a linear code. We use a large set of codes to compare our new algorithm with the Brouwer-Zimmermann algorithm. We find that for about one third of codes in this sample set, our algorithm requires to generate fewer codewords than the Brouwer-Zimmermann algorithm.

2. Matroid partitioning

A *matroid* is a pair $M = (E, I)$ where E is the set of elements of M and I is a collection of subsets of E, called the *independent sets*, that satisfies certain axioms. Matroids are an axiomatic abstraction of the theory of linear dependence in vector spaces. We recommend [11] for a current and comprehensive survey of matroid theory.

A *matroid partitioning algorithm* takes as input matroids $M_i = (E, I_i)$ where $1 \leq i \leq r$. Note that all M_i have the same ground set E but in general their sets of independent sets may be different. The algorithm decides whether there exist sets $S_1, \ldots, S_r$ such that $S_i \in I_i$ for $1 \leq i \leq r$ (that is, S_i is an independent set with respect to the i-th matroid) and $\bigcup_{i=1}^{r} S_i = E$ and $S_i \cap S_j = \emptyset$ whenever $i \neq j$. If such a partition does exist, then the algorithm finds one such partition.

The first matroid partitioning algorithm was invented by Edmonds in 1965. A description of Edmonds' algorithm can be found for example in Section 8.7 of [9]. Our implementation of Edmonds' algorithm is based on consulting the references [6] and [9]; many other references exist as well. Assuming that matroid partition(s) do exist, the version of Edmonds' algorithm given in [9] finds a partition $S_1, \ldots, S_r$ such that the sequence $(|S_1|, |S_2|, \ldots, |S_r|)$ is *lexicographically maximal* among all sequences $(|U_1|, |U_2|, \ldots, |U_r|)$ where $U_1, \ldots, U_r$ is a matroid partition. Throughout the paper $|X|$ denotes the cardinality of set X.

The complexity analysis of Edmonds' algorithm in [9] shows that the algorithm runs in time $O(m^3 t)$ where $m = |E|$ and t is the maximum time required for testing whether $F \in I_i$, where F is some subset of E and i is some number between 1 and r.

For the type of matroids that we use in this paper, specialized matroid partitioning algorithms exist, see [5] and later references, and their time complexity is lower. However we choose to not go into more detail here, as all matroid partitioning algorithms run in polynomial time whereas the algorithms for computing the minimum weight of a linear code run overall in superpolynomial time (unless P=NP).

3. Brouwer-Zimmermann minimum weight algorithm

An algorithm for computing the minimum weight of a linear code over a finite field was designed by A. Brouwer and subsequently extended by K.-H. Zimmermann. Henceforth we will refer to it as *Brouwer-Zimmermann algorithm,* abbreviated *BZ algorithm.* Descriptions of the BZ algorithm can be found in [1, Section 1.8], [2, Section 1.3] and [7]. A thorough implementation of the algorithm is available in Magma [3], and a description of the Magma implementation is given in [7]. In [8] the algorithm was adapted to finding minimum weight of $\mathbb{Z}_4$-linear quadratic residue codes.

Let C be a linear code whose minimum weight d we wish to determine. Let C^* be the set of non-zero codewords of C. Upon completion of each step, the BZ algorithm considers C^* as a disjoint union

$$C^* = C' \sqcup C''.$$

At this point, all elements of C' have been listed explicitly (but none of them needs to be stored permanently), thus yielding an upper bound $d \leq \overline{d}$ where $\overline{d}$ is the minimum weight of elements of C'. At the same time, the algorithm establishes a lower bound $\mathrm{wt}(c) \geq \underline{d}$, where c denotes an arbitrary element of C'', without listing any elements of C'' explicitly. If $\underline{d} \geq \overline{d}$, then the algorithm terminates with the message that the minimum weight of C equals $\overline{d}$. Otherwise, in the next step the algorithm augments the set C' so that it remains easy to lower bound the weights of elements of the new set C'', and the same process is repeated. It is desired that upon termination the set C' is as small as possible, since almost all effort of the algorithm is spent on listing the elements of C'.

Example 3.1. Consider the following two matrices G_1 and G_2 over $\mathbb{F}_2$:

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

These are two generator matrices for the same $[10, 5]_2$ linear code C, and again let d denote the minimum weight of C. Consider $C^* = C' \sqcup C''$ where C' consists of rows of G_1 and rows of G_2. By examining elements of C' one-by-one, we obtain $d \leq 4$. Now let $c \in C''$ and write $c = (c^1|c^2)$ where $c^1, c^2 \in \mathbb{F}_2^5$. Since $c \notin C'$, it follows that c is a linear combination of at least two rows of G_1, hence $\mathrm{wt}(c^1) \geq 2$. Similarly, c is a linear combination of at least two rows of G_2, hence $\mathrm{wt}(c^2) \geq 2$ and $\mathrm{wt}(c) = \mathrm{wt}(c^1) + \mathrm{wt}(c^2) \geq 4$. The algorithm terminates with the output "$d = 4$."

Let G be a generator matrix for an $[n, k]_q$ code C. A subset $T \subseteq \{1, 2, ..., n\}$ of size $|T| = k$ is called an *information set* for C if the corresponding columns in G are linearly independent. Then there also exists a generator matrix G_T for C such that the columns of G_T specified by T form an identity matrix. Each codeword of C is of the form uG_T for some $u \in \mathbb{F}_q^k$. We have

$$\mathrm{wt}(uG_T) \geq \mathrm{wt}(u) \qquad \text{for all } u \in \mathbb{F}_q^k. \tag{1}$$

3.1. Outline of the BZ algorithm. Let C be an $[n, k]_q$ linear code given by a generator matrix G. The BZ algorithm will determine the minimum weight of C as follows.

The algorithm will find subsets $T_1, \ldots, T_\ell$ of $\{1, 2, \ldots, n\}$ such that each T_i is an information set for C and $\bigcup_{i=1}^{\ell} T_i = \{1, 2, \ldots, n\}$. After finding set T_i, Gauss-Jordan elimination is applied to construct matrix G_i which is a generator matrix for C and it has the identity matrix in the columns specified by T_i.

The sequence of non-negative integers $r_1, \ldots, r_\ell$ is determined by

$$r_i := \left| T_i \setminus \bigcup_{j=1}^{i-1} T_j \right| \qquad \text{for } 1 \leq i \leq \ell. \tag{2}$$

The integers r_i are called *relative ranks* in [7], and we will follow this terminology. For two reasons it is customary to write the sequence of relative ranks in the non-increasing order, that is,

$$r_1 \geq r_2 \geq \ldots \geq r_\ell. \tag{3}$$

Firstly, the methods used in [1, 2, 7] for constructing sets T_i and matrices G_i produce them in such a way that the sequence of relative ranks is non-increasing. Secondly, it is more beneficial for the BZ algorithm to operate on matrices with larger relative ranks prior to operating on matrices with smaller relative ranks. We will assume that (3) always holds.

While the algorithm is operating, it may decide to discard some matrices G_i because their relative ranks are so small that they will not get an opportunity to contribute to the lower bound on weights of codewords in C'' before the termination of the algorithm. Such decisions may be made on the fly, whenever the upper bound $\overline{d}$ decreases as a consequence of discovering a codeword of weight less than the previous value of $\overline{d}$. A numerical example of this phenomenon is given in [7]. From the complexity analysis point of view, predicting when such events will occur is not possible, yet the impact of these events on the running time of the algorithm is significant. In our analysis, we will assume that the algorithm operates throughout the entire computation on the sequence of matrices $G_1, \ldots, G_D$ where D is some integer such that $1 \leq D \leq \ell$. (The only exception may be the last iteration of the computation when the algorithm operates on $G_1, \ldots, G_z$ for some $1 \leq z \leq D$.) We say that the algorithm operates up to *depth* D. We will show below that this assumption is consistent with one of the modes in which the BZ algorithm is used in practice. The value of D is determined once the sets $T_1, \ldots, T_\ell$ have been constructed. In the descriptions of the algorithm given in [1] and [2] the parameter D is not discussed; thus $D = \ell$ is implicitly used in these descriptions. In [7] the choices for D are discussed: In the mode where the algorithm computes the minimum weight, the value of D is adjusted dynamically according to the changes of the value of $\overline{d}$. In the mode where the algorithm verifies a lower bound on the minimum weight, the optimal value of D can be determined in advance and it stays constant throughout the execution of the algorithm.

Each step of the algorithm is characterized by a pair of integers (w, j) where $1 \leq w \leq k$ and $1 \leq j \leq D$. The initial values of the main variables are $(w, j) := (1, 1)$ and $\overline{d} := n - k + 1$ (Singleton bound).

In step (w, j) the algorithm enumerates all codewords uG_j such that $\mathrm{wt}(u) = w$. During this process, whenever a codeword x is generated such that $\mathrm{wt}(x) < \overline{d}$, then we set $\overline{d} := \mathrm{wt}(x)$. In other words, the algorithm updates $\overline{d}$ according to

$$\overline{d} := \min\left(\overline{d}\,,\ \min\{\mathrm{wt}(uG_j)\ :\ u \in \mathbb{F}_q^k,\ \mathrm{wt}(u) = w\}\right) \tag{4}$$

without storing the set of codewords in memory. The new value of $\underline{d}$ is determined as

$$\underline{d} := \sum_{i=1}^{j} \max(0, w + 1 - k + r_i) + \sum_{i=j+1}^{D} \max(0, w - k + r_i). \tag{5}$$

Using (1) it is easy to see [7] that any codeword $x \in C^*$ that has not been generated by the algorithm up to this point satisfies $\mathrm{wt}(x) \geq \underline{d}$. The algorithm now tests whether $\underline{d} \geq \overline{d}$. If this is the case, then the algorithm terminates with the message that the minimum weight of C equals $\overline{d}$. Otherwise, if $j < D$, then the algorithm proceeds to step $(w, j+1)$, otherwise it proceeds to step $(w+1, 1)$. Should the algorithm ever reach the step $(k, 1)$, then it will terminate after completing this step, since all codewords have been enumerated.

It is noted in [7] that the overall work of the algorithm can be reduced by the factor of $q-1$ by using only left-normalized vectors u in (4), since codewords that are non-zero scalar multiples of each other have the same Hamming weight. The overall running time of the BZ algorithm is essentially determined by the total number of codewords of C that the algorithm generates during its execution. This value is called the *work factor* in [7], and we will use this term in our paper too. Assuming that the algorithm terminates upon completing step (w, j), the number of codewords that have been generated is

$$W(D, w, j) = j \sum_{z=1}^{w} \binom{k}{z} (q-1)^{z-1} + (D-j) \sum_{z=1}^{w-1} \binom{k}{z} (q-1)^{z-1}. \tag{6}$$

3.2. Proving a lower bound on the minimum weight.

The BZ algorithm can operate in different modes, and this is reflected for example by the fact that the Magma implementation of it [7] offers several different commands through which the algorithm can be invoked. The version of the algorithm that we outlined in Section 3.1 computes the minimum weight of C. It is also possible to use the algorithm to solve the following decision problem:

Given a linear code C and a positive integer L, is it true that the minimum weight of C is greater than or equal to L?

In order to solve this problem, the only modification required to the algorithm outlined in Section 3.1 is that the algorithm will terminate with output "yes" as soon as the inequalities $\overline{d} \geq L$ and $\underline{d} \geq L$ are both satisfied. If the algorithm ever comes across a codeword of weight less than L, then it will terminate with output "no." This mode of operation of BZ algorithm is available in Magma via the command `VerifyMinimumDistanceLowerBound`.

If the algorithm outputs "no," then it is in general impossible to predict the time at which the algorithm comes across a codeword of weight less than L. We will analyze the work factor in the case when the algorithm outputs "yes." Hence we are analyzing the complexity of using the BZ algorithm to *prove a lower bound on the minimum weight of a linear code.* In this case, the inequality $\overline{d} \geq L$ holds true throughout the execution of the algorithm. Hence we only need to analyze the work needed to obtain the inequality $\underline{d} \geq L$.

Given C and L as above, the algorithm starts by determining the information sets $T_1, \dots, T_\ell$ and the corresponding relative ranks $r_1, \dots, r_\ell$. Afterwards, the algorithm will consider in turn all possible values $D = 1, 2, \dots, \ell$. For each such D the algorithm will determine the earliest (in the order of execution) pair (w, j) such that the right-hand side of (5) is greater than or equal to L, and it will compute the corresponding work factor $W(D, w, j)$ using (6). The value $D = D_0$ that minimizes the work factor required will be found, and the BZ algorithm will be invoked at this optimal depth D_0 to deliver the proof that L is a lower bound on the minimum weight of C.

4. Construction of information sets for the BZ algorithm

The issue of finding information sets $T_1, \dots, T_\ell$ that yield a favourable sequence of relative ranks $r_1, \dots, r_\ell$ is a problem of its own. We address it in this section. It is intuitively clear from (5) that larger values of r_i should make the lower bound $\underline{d}$

grow faster, which means a faster completion of the algorithm. (Note that $r_i \leq k$ for all i.) This motivates the following definition.

Definition 4.1. An *α-partition* of an $[n,k]_q$ linear code C is a partition of its set of coordinates into $\lfloor n/k \rfloor$ linearly independent sets of size k and, in case that k does not divide n, one linearly independent set of size $n \bmod k$.

Note that C has an α-partition if and only if there exists a sequence of information sets for C such that the corresponding sequence of relative ranks is

$$(k,\, \ldots,\, k,\, n \bmod k) \tag{7}$$

where the last term is omitted if k divides n. It is easy to convert one object into the other one.

Proposition 1. *Let C be an $[n,k]_q$ linear code. There exists an algorithm with time complexity $O(n^3k^3)$ that decides whether C has an α-partition, and it outputs an α-partition of C if it exists.*

Proof. Let G be a generator matrix for C. Let $E = \{1, \ldots, n\}$ and let $r := \lceil n/k \rceil$. For $1 \leq i \leq r$ consider matroids $M_i = (E, I_i)$ defined as follows. Each set I_i consists of precisely those subsets $F \subseteq E$ such that the columns of G indexed by F are a linearly independent set in $\mathbb{F}_q^k$. Apply Edmonds' algorithm (Section 2) to $M_1, \ldots, M_r$. The existence of an α-partition of C is equivalent to the existence of a matroid partition $S_1, \ldots, S_r$ such that $(|S_1|, \ldots, |S_r|) = (k, \ldots, k)$ if k divides n, or $(|S_1|, \ldots, |S_r|) = (k, \ldots, k, n \bmod k)$ if k does not divide n. In either case this is the lexicographically maximal matroid partition possible, thus it will be found by Edmonds' algorithm in case that it exists.

For the conclusion about the running time, recall from Section 2 that Edmonds' algorithm runs in time $O(m^3t)$ where $m = |E|$ and t is the maximum time required for testing whether $F \in I_i$, where F is some subset of E and i is some number between 1 and r. In our case $m = |E| = n$. Let Z be an arbitrary subset of $E = \{1, \ldots, n\}$ and suppose that we want to test whether $Z \in I_i$. (Recall that all sets I_i are equal, hence the value of i is of no consequence.) If $|Z| > k$, then $Z \notin I_i$. If $|Z| \leq k$, then let Q denote the submatrix of G consisting of those columns of G indexed by Z. Then $Z \in I_i$ if and only if $\mathrm{rank}(Q) = |Z|$. This can be decided by Gauss-Jordan elimination in time $O(k^3)$. Overall testing whether $Z \in I_i$ can be done in time $O(k^3)$. Hence Edmonds' algorithm will run in time $O(n^3k^3)$. □

In [1, Section 1.8] it is noted that the BZ algorithm works efficiently if the code under consideration has many information sets which are pairwise disjoint. We now show that this objective can be achieved deterministically in polynomial time.

Proposition 2. *Let C be an $[n,k]_q$ linear code. There exists an algorithm with time complexity $O(n^3k^3)$ that determines N, the maximum number of pairwise disjoint information sets for C, and it finds a set of N pairwise disjoint information sets for C.*

Proof. As in the proof of Proposition 1 we form the matroids M_i from the code C, except that we now take $r = n$, and we apply Edmonds' algorithm to them. Since Edmonds' algorithm delivers a matroid partition $S_1, \ldots, S_r$ such that the sequence $(|S_1|, \ldots, |S_r|)$ is lexicographically maximal among all matroid partitions, in particular the sequence $S_1, \ldots, S_r$ will contain the maximum possible number of pairwise disjoint information sets for C. These will be the sets $S_1, \ldots, S_N$ where N is the largest number i such that $|S_i| = k$. □

We next show that if an α-partition of C exists, then it is the optimal choice for the mode of the BZ algorithm that we study.

Proposition 3. *If R is a sequence of relative ranks corresponding to an α-partition of C, then the work factor of the BZ algorithm for proving a lower bound on the minimum weight of C using R is less than or equal to the work factor when using any other sequence of relative ranks for C.*

Proof. Let L be a positive integer and suppose that the BZ algorithm is proving that the minimum weight of C is at least L. Recall that, once the sequence of relative ranks has been fixed, the algorithm will choose the best value of the depth parameter before it starts executing. If C is an $[n,k]_q$ code, then $\sum_{i=1}^{\ell} r_i = n$ for each sequence of relative ranks (r_i).

Let $r = (r_1, \ldots, r_\ell)$ be an arbitrary sequence of relative ranks for C, and recall that r is written in non-increasing order. As in the statement of the proposition, let R be the sequence of relative ranks (7). Let D be the optimal depth for running the BZ algorithm using the sequence r. Consider an execution of the BZ algorithm using the sequence R with the same depth D. (If the length of R is less than D, insert "empty" steps where no work is performed.) Let $\mathrm{lb}(w,j)$ be the value (5) computed by the algorithm that uses the sequence r after completing the step (w,j), and similarly let $\mathrm{lb}'(w,j)$ be the value (5) computed by the algorithm that uses the sequence R after completing the step (w,j). We have $\mathrm{lb}'(w,j) \geq \mathrm{lb}(w,j)$ for all (w,j). Recall that the termination conditions for the two algorithms are $\mathrm{lb}(w,j) \geq L$ and $\mathrm{lb}'(w,j) \geq L$ respectively. Thus the optimal work factor for the instance of the algorithm that uses R is less than or equal to the optimal work factor for the instance of the algorithm that uses r. □

4.1. Implementation. We implemented Edmonds' algorithm in Magma [3]. For a sample set of codes, in Table 1 we give the running time of our algorithm in the second column, as well as the (exact or predicted) running time of the BZ algorithm offered by Magma in the third column. Our implementation is very crude and it only serves as a proof of concept; it is very likely that the timings in the second column of Table 1 can be reduced significantly. Our timings were obtained using Magma 2.19-6 running on Intel Core i7 CPU at 3.2 GHz. All timings are given in seconds. Random matrices were used as generator matrices for the linear codes listed in the table. In general we observed that for many codes the overhead for finding a lexicographically maximal partition is smaller by many orders of magnitude than the running time of the BZ algorithm, hence it becomes negligible.

parameters of code	lex.max. partition (sec)	BZ algorithm (sec)
$[100,40]_2$	0.1	0.5
$[150,40]_2$	0.3	50
$[200,100]_2$	1.7	$\sim 10^7$
$[210,120]_2$	2.2	$\sim 10^{15}$
$[140,20]_3$	0.2	0.9
$[140,30]_3$	0.3	4526
$[160,110]_3$	1.8	$\sim 10^{14}$

Table 1. CPU time for finding a lexicographically maximal partition for a linear code and running time of BZ algorithm.

Example 4.2. Consider the binary linear code C of length 1344 and dimension 128 which Canteaut, Lallemand and Naya-Plasencia recently used in their attack on the block cipher PICARO. The code C is defined in Section 4.1 of [4]; we obtained a generator matrix G for C from the authors. The attack requires to find codewords of C of the minimum weight. In [4] theoretical arguments are used to prove that the minimum weight of C is 18 and to construct codewords of weight 18. When we attempted to compute the minimum weight of C with the built-in `MinimumWeight` function in Magma, after one day of running the system estimated that the time required for completing the task will be about 10^6 years. We then applied our matroid partitioning algorithm to G and in 91 seconds we found an α-partition for C, that is, a partition of G into $\lfloor 1344/128 \rfloor = 10$ square invertible matrices (say $M_1, \ldots, M_{10}$) plus a linearly independent set of 64 columns. Then it was sufficient to perform Gauss-Jordan elimination on G to obtain generator matrices $G_1, \ldots, G_{10}$ that have the identity matrix in the columns specified by $M_1, \ldots, M_{10}$ respectively. Any non-zero codeword of C of weight less than 20 occurs as a row of at least one of the matrices $G_1, \ldots, G_{10}$ since any other non-zero codeword has weight at least $10 \cdot 2 = 20$. Thus, in mere 94 seconds we determined that the minimum weight of C is 18 and we found the eight codewords of weight 18 listed in [4].

4.2. Comparison with previous versions of the BZ algorithm. The BZ algorithm starts its execution by finding information sets $T_1, \ldots, T_\ell$ that yield the sequence of relative ranks $r_1, \ldots, r_\ell$. It is clear from (5) that the sequence (r_i) has impact on the operation of the algorithm, hence spending some effort on making a choice among available sequences (r_i) appears to be well justified.

In [1] and [2] the issue of choosing among different sequences (r_i) is not considered. The information sets are produced by one sweep of the generator matrix from left to right, by a sequence of Gaussian eliminations performed on rectangular matrices of decreasing size. This method guarantees $r_1 = k$ but not much can be inferred about the sequence (r_i) as a whole.

In [7] the choice among different sequences (r_i) is considered. The generator matrix is swept from left to right as in [2] and [1]. If the first pass fails to produce the sequence of relative ranks (7), then a random permutation is applied to the columns of the generator matrix, and the process is repeated over and over. The Magma implementation of the BZ algorithm uses this heuristic [7, p. 293].

In this paper we present a deterministic algorithm running in time polynomial in n and k that finds information sets yielding the relative rank sequence (7), which has been deemed to be the most favourable situation for the BZ algorithm [7, p. 293], or it proves that this relative rank sequence can not be achieved for the code under investigation. We make some assumptions about the mode of operation of the BZ algorithm that allow us to formally prove that the sequence (7) is optimal in cases when it is achievable. We note that the same algorithm also always finds the maximum number of pairwise disjoint information sets for the given code, which is another objective that can be pursued [1].

5. A new algorithm for computing minimum weight

In the BZ algorithm, each coordinate of C contributes to the lower bound $\underline{d}$, given in equation (5), via at most one information set. It can happen that many coordinates do not contribute to $\underline{d}$ at all, as can be seen for example in Proposition 4 below. After stating this proposition, in Section 5.1 we propose a new algorithm in which a coordinate of C can contribute to $\underline{d}$ via more than one information set. Then in Section 5.2 we perform a comparison of our algorithm and BZ algorithm using a large set of linear codes.

Proposition 4. *Let C be an $[n,k]_q$ code such that $n \leq \frac{3}{2}k$ and let $L \leq n-k+1$. In order to prove that the minimum weight of C is at least L, the number of codewords generated by the Brouwer-Zimmermann algorithm is*

$$\sum_{z=1}^{L-1} \binom{k}{z} (q-1)^{z-1}. \tag{8}$$

Proof. It follows from the Singleton bound that the assumption $L \leq n-k+1$ is not restrictive, as it only excludes the cases where the answer is trivial. We have $L \leq n-k+1 \leq \frac{k}{2}+1$. For the relative ranks as defined in (2) we have $r_1 = k$ and $r_i \leq \frac{k}{2}$ for $i \geq 2$. Considering the terms corresponding to $i \geq 2$ and $w \leq L-2$ in (5) we get

$$w+1-k+r_i \leq L-1-k+\frac{k}{2} \leq 0,$$

hence the terms corresponding to $i \geq 2$ and $w \leq L-2$ will never make a positive contribution to $\underline{d}$ in (5). It follows that, for proving that the minimum distance of C is at least L, the Brouwer-Zimmermann algorithm will operate to depth $D=1$ only and according to (6) the work factor for this will be

$$W(1,L-1,1) = \sum_{z=1}^{L-1} \binom{k}{z} (q-1)^{z-1}. \qquad \square$$

Looking at the proof of Proposition 4 we see that up to one third of the coordinates of C are useless in the computation of the minimum weight of C, since their contributions to codeword weights are disregarded by the BZ algorithm. We will now describe our new algorithm which allows more flexibility for the manner in which each coordinate of the code may contribute towards a faster completion of the algorithm.

5.1. The new algorithm.

Our algorithm is supported by the following proposition.

Proposition 5. *Let C be an $[n,k]_q$ code. Suppose that a,b are positive integers and that there exist sets $T_1,\ldots,T_a$ such that $T_j \subset \{1,\ldots,n\}$ for $j=1,\ldots,a$ and each element of $\{1,\ldots,n\}$ belongs to at most b sets T_j. Assume that $c \in C$ is a codeword and r is an integer such that the weight of c restricted to T_j is at least $r+1$, for all $j=1,\ldots,a$. Then the weight of c is at least $\frac{a}{b}(r+1)$.*

Proof. Let $1 \leq i \leq n$. Let $w_i = 0$ if $c_i = 0$ and $w_i = 1$ if $c_i \neq 0$. For $1 \leq j \leq a$ let $t_{ij} = 1$ if the i-th coordinate of C belongs to T_j, and let $t_{ij} = 0$ otherwise. Then

$$a(r+1) \leq \sum_{j=1}^{a}\sum_{i=1}^{n} t_{ij}w_i = \sum_{i=1}^{n} w_i \sum_{j=1}^{a} t_{ij} \leq \left(\sum_{i=1}^{n} w_i\right) b = b \cdot \mathrm{wt}(c)$$

and the result follows. $\square$

Our new algorithm for proving a lower bound on the minimum weight of a linear code is given in Table 2. In step 1, the information sets $T_1,\ldots,T_a$ are found by applying the matroid partitioning algorithm to the matroid M whose elements are triples of the form (i,j,u_j) where $1 \leq i \leq b$, $1 \leq j \leq n$ and u_j is the j-th column of

```
Given: generator matrix $G$ for an $[n,k]_q$ code $C$
       positive integer $L$
       positive integer $b$
Task: Determine whether the minimum weight of $C$ is at least $L$.

1. Determine the largest integer $a$ such that there exist information sets
   $T_1,\ldots,T_a$ for $C$, such that each coordinate of $C$ belongs to at most $b$ sets $T_j$.
   Construct a sequence of information sets $T_1,\ldots,T_a$ satisfying this condition.
2. Let $r := \lceil \frac{bL}{a} - 1 \rceil$. For each $j = 1,\ldots,a$ find all non-zero codewords $c \in C$
   such that the weight of $c$ restricted to $T_j$ is at most $r$. If any of these
   codewords has weight less than $L$, then return "false."
3. Return "true."
```

Table 2. Algorithm for proving a lower bound on the minimum weight of a linear code.

G. So the number of elements of M is bn. We define that a set X of elements of M is an independent set in M exactly when the multiset of the third components of elements of X is an independent subset of $\mathbb{F}_q^k$. The matroid partitioning algorithm applied to M will return a sequence $(I_1,\ldots,I_s)$ which is a partition of the set of all elements of M into independent sets I_i such that the sequence $(|I_1|,\ldots,|I_s|)$ is lexicographically maximal among all such partitions. Then the value of a in step 1 is determined as the maximum value of t such that $|I_t| = k$. For $1 \le j \le a$, the set T_j is obtained in step 1 by setting T_j equal to the set of the second components of all elements of I_j.

Step 2 is performed in the same manner as in the BZ algorithm. By applying row operations to G, one finds generator matrices $G_1,\ldots,G_a$ such that G_j has the identity matrix in the columns specified by T_j, and then all linear combinations of at most r rows of G_j are found for $1 \le j \le a$. As was remarked earlier in this paper, when forming these linear combinations, the first non-zero coefficient can be normalized to 1 since codewords that are non-zero scalar multiples of each other have the same Hamming weight.

The number of codewords that have to be generated by our algorithm is

$$\sum_{z=1}^{\lceil \frac{bL}{a} - 1 \rceil} a \binom{k}{z} (q-1)^{z-1}. \tag{9}$$

It is important to note that the value of b in our algorithm is a "free" parameter, in the sense that the user is free to choose the value of b before running the algorithm. Choosing $b = 1$ corresponds to our extension of the BZ algorithm as given earlier in Section 4, in the sense that it constitutes the proof of Proposition 2. Choosing value $b > 1$ may result in a smaller number of codewords generated in comparison to using BZ algorithm. We performed computational experiments whose results we report in Section 5.2.

As with the BZ algorithm, our algorithm can be also used in the mode where it computes the minimum weight of C (instead of just proving a lower bound on the minimum weight). To operate in this mode, in step 2 the value of L (and hence the value of r) is adjusted dynamically whenever a codeword of weight less than L is found.

5.2. Computational results. By the considerations given earlier in this paper, the complexity of step 1 of our algorithm (Table 2) is $O(b^3n^3k^3)$, since the number of elements of the matroid is bn. To assess the applicability of our algorithm with values $b > 1$ we performed the following experiment for a large set of linear codes C:

Given an $[n, k, d]_q$ code C, we computed the work factor (number of codewords that have to be generated) required to prove $\mathrm{wt}(C) \geq d$ by the BZ algorithm, as well as by our algorithm (with $b = 2, 3, 4$). To determine the work factor for the BZ algorithm we applied the matroid partitioning algorithm to find the lexicographically maximal sequence of relative ranks (r_i) and then we applied formula (6) with the optimal value of depth D (see the last paragraph of Section 3.2, and set $L = d$ there). To determine the work factor for our algorithm (with a fixed value of b), we executed its step 1 (thereby obtaining the value of a) and then we applied formula (9) with $L = d$. At this point, the user will choose to use the algorithm with the smallest work factor required to prove that $\mathrm{wt}(C) \geq d$. We measured the time that it took to perform step 1 of our algorithm. Our timings were obtained using Magma 2.19-6 running on Intel Core i7 CPU at 3.2 GHz.

We performed this experiment on the set of best known $[n, k]_q$ linear codes stored in the data base of Magma such that $n \leq 170$, $1 < k < n$ and $q \in \{2, 3, 4\}$. We only considered those codes that do not contain a coordinate equal to 0 in all codewords. The sizes of these sets are as follows: 5965 codes for $q = 2$, 7072 codes for $q = 3$, and 7239 codes for $q = 4$.

In Tables 3, 4 and 5, we report the results for binary, ternary and quaternary codes, respectively. In each row of each table, the first entry indicates the values of b for which the results are reported in the row. The second entry indicates the percentage of codes for which our algorithm has a smaller work factor (for at least one value of b under consideration) than the BZ algorithm. The third entry indicates the total combined time (in seconds) spent in step 1 of our algorithm for all values of b under consideration.

values of b	improvement proportion	maximum overhead time (sec)
2	17.4%	10
$2, 3$	26.7%	31
$2, 3, 4$	30.6%	70

Table 3. Results for linear codes with $n \leq 170$ and $q = 2$.

values of b	improvement proportion	maximum overhead time (sec)
2	16.0%	11
$2, 3$	28.1%	37
$2, 3, 4$	33.1%	84

Table 4. Results for linear codes with $n \leq 170$ and $q = 3$.

We emphasize that we used only a very crude implementation of the simplest matroid partitioning algorithm, and very likely our timings in the third columns of Tables 3, 4 and 5 can be decreased significantly. These timings only serve to prove the fact that the overhead required by our algorithm will be quite acceptable in many cases, since the BZ algorithm often requires much longer time to complete. (Indeed

values of b	improvement proportion	maximum overhead time (sec)
2	13.8%	12
$2, 3$	26.7%	39
$2, 3, 4$	32.8%	90

Table 5. Results for linear codes with $n \leq 170$ and $q = 4$.

computing the minimum weight of some binary linear codes of length $n \geq 108$ takes more than a day using Magma's built-in function `MinimumWeight`, and computing the minimum weight of some binary linear codes of length $n \geq 124$ takes more than a year using the same function.) Also for the sake of keeping the tables small we reported only one timing in each row; this timing is typically achieved for n and k both being close to 170. Then, for smaller values of n and k the expected time for step 1 of our algorithm can be estimated from this data and from the knowledge that the time complexity of step 1 is $O(b^3n^3k^3)$.

5.3. Conclusion. We conclude that the new algorithm presented in this section provides a competitive alternative to the classical BZ algorithm. We expect that our algorithm will be used in the following way: It will be supported by an efficient implementation of a fast matroid partitioning algorithm. The time required to execute step 1 of our algorithm will be determined empirically as a function of b, n and k. In order to determine (or to lower bound) the minimum weight of any given linear code C, the user will first invoke the BZ algorithm on C to the point when a good estimate for the completion time is obtained. If the estimated completion time of the BZ algorithm is much higher than the time required for step 1 of our algorithm, then step 1 of our algorithm will be executed, and afterwards the best performing algorithm will be chosen to complete the task.

Acknowledgments

The authors thank Luis Goddyn for helpful comments and discussions. The authors thank the anonymous referee for valuable comments.

References

[1] A. Betten, M. Braun, H. Fripertinger, A. Kerber, A. Kohnert and A. Wassermann, *Error-Correcting Linear Codes*, Springer-Verlag, 2006.

[2] A. Betten, H. Fripertinger, A. Kerber, A. Wassermann and K.-H. Zimmermann, *Codierungstheorie–Konstruktion und Anwendung linearer Codes*, Springer-Verlag, 1998.

[3] W. Bosma, J. Cannon and C. Playoust, The Magma algebra system. I. The user language, *J. Symbolic Comput.*, **24** (1997), 235–265.

[4] A. Canteaut, V. Lallemand and M. Naya-Plasencia, Related-key attack on full-round PICARO, in *Proc. 22nd Int. Conf. Sel. Areas Crypt. 2015* (eds. O. Dunkelman and L. Keliher), Springer, 2015.

[5] W. H. Cunningham, Improved bounds for matroid partition and intersection algorithms, *SIAM J. Comput.*, **15** (1986), 948–957.

[6] J. Edmonds, Matroid partition, *Math. Decis. Sci.*, **11** (1968), 335–345.

[7] M. Grassl, Searching for linear codes with large minimum distance, in *Discovering Mathematics with Magma - Reducing the Abstract to the Concrete* (eds. W. Bosma and J. Cannon), Springer, 2006, 287–313.

[8] M. Kiermaier and A. Wassermann, Minimum weights and weight enumerators of $\mathbb{Z}_4$-linear quadratic residue codes, *IEEE Trans. Inf. Theory*, **58** (2012), 4870–4883.

[9] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, 1976.

[10] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, 1977.
[11] J. Oxley, *Matroid Theory*, 2nd edition, Oxford University Press, 2011.
[12] A. Vardy, The intractability of computing the minimum distance of a code, *IEEE Trans. Inf. Theory*, **43** (1997), 1757–1766.

Received December 2014; revised November 2015.

E-mail address: plisonek@sfu.ca
E-mail address: ltrummer@sfu.ca