Optimizing BJMM with Nearest Neighbors: Full Decoding in $2^{2n/21}$ and McEliece Security

Leif Both and Alexander May

Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
Faculty of Mathematics
leif.both@rub.de, alex.may@rub.de

Abstract. We revisit the Becker-Joux-May-Meurer (BJMM) decoding algorithm in combination with Nearest Neighbor search. May and Ozerov showed that Nearest Neighbor search speeds up the original BJMM algorithm for full decoding of random binary linear codes of length n from $2^{0.1019n}$ to $2^{0.0967n}$. We show that some optimization of the original BJMM algorithm in combination with Nearest Neighbor search further slightly improves the worst case running downto $2^{0.0953n}$. We also provide optimized running times for BJMM for cryptographic instantiations of the McEliece cryptosystem that cast some doubts on the targeted security levels.

Keywords: Decoding binary linear codes, BJMM, Nearest Neighbors

1 Introduction

Decoding random linear k-dimensional codes $\mathcal{C} \subseteq \mathbb{F}_2^n$ is an NP-hard problem that is interesting for many areas of computer science, and especially for the construction of cryptographic systems that are supposed to resist quantum computer attacks [McE78,Ale03,Reg05]. Therefore, it is of paramount importance to know the best algorithms and the true complexity of the decoding problem, which in turn is the key to precisely estimating the security of code-based cryptosystems.

A major step towards understanding the complexity was already achieved in the sixties by Prange [Pra62], who introduced the notion of *Information Set Decoding* (ISD) algorithms. With Prange's algorithm, decoding any constant rate code can be done in time $2^{0.1208n}$. Here the constant 0.1208 in the exponent is maximized over all rates $\frac{k}{n}$, and achieves its maximum slightly below rate $\frac{1}{2}$. Since then, there have been a number of refinements of Prange's original ISD algorithm [Ste88,Dum91,Bar97,BLP11,MMT11,BJMM12,MO15] improving this complexity exponent.

Statistical Decoding is the only known promising generic decoding technique that does not fit into the realm of ISD algorithms. It was first introduced by Al Jabri [Al 01], later optimized by Overbeck [Ove06] and analyzed by Fossorier, Kobara and Imai [FKI07]. However, a recent asymptotic analysis by Debris-Alazard and Tillich [DT17] shows lower bounds for this approach implying that for random linear codes Statistical Decoding is always inferior even to Prange's original ISD algorithm.

Our work builds on the well known ISD algorithm of Becker, Joux, May, and Meurer [BJMM12] that makes use of the combinatorial *representation technique*, which was first invented in the context of subset-sum algorithms [HGJ10,BCJ11]. This BJMM algorithm achieves a worst-case complexity of $2^{0.1019n}$ by computing a binary depth-3 search tree $\mathcal{T}$. The choice of depth-3 is not really justified in [BJMM12]. However, some computations show that shortening $\mathcal{T}$ to depth-2 yields inferior results, while extending $\mathcal{T}$ to depth-4 yields the same results as in depth-3, albeit with a slightly more complex algorithm. Hence, depth-3 seems to be optimal.

In 2015, May and Ozerov showed that by replacing the last computation step in BJMM's search tree $\mathcal{T}$ by some Hamming distance Nearest Neighbor search improves the complexity to $2^{0.0967n}$. Their analysis uses a depth-3 $\mathcal{T}$ like in the original BJMM. However, we show that the combination of BJMM with Nearest Neighbor search achieves its optimal complexity $2^{0.0953n}$ with a depth-4 tree. We also verify that depth-4 is sufficient by showing that depth-5 leads to the same result.

So besides our slight improvement for ISD decoding, a message of our paper is that one should rather consider the BJMM algorithm as a family of algorithms BJMM(m), parametrized by depth-m of its search tree, that has to be optimized anew for any modifications.

We also provide optimizations of BJMM(m) with and without Nearest Neighbors for other settings of interest, e.g. for common instantiations of the McEliece cryptosystem as proposed in Bernstein, Lange and Peters [BLP08]. Our results give some indication that the proposed instantiations with Goppa codes of dimensions $n = 1632, 2960, 6624$ might actually lead to smaller bit security levels than the targeted $80, 128, 256$.

As part of our results, we publish our C-code for optimizing BJMM(m) with or without Nearest Neighbors at `https://github.com/LeifBoth/bjmm2.0-code`.

Our paper is organized as follows. In Section 2, we describe the general realm of Information Set Decoding. Section 3 introduces a formulation of BJMM(m) with parametrized depth. In Section 4, we present our results for optimizing BJMM(m) within different decoding settings.

2 Preliminaries

Let us first give some preliminaries on decoding random binary linear codes, and especially on ISD algorithms.

Let $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$. Then we denote by $\Delta(\mathbf{x}, \mathbf{y})$ the *Hamming distance* of $\mathbf{x}$ and $\mathbf{y}$. The Hamming weight $\Delta(\mathbf{x})$ of $\mathbf{x}$ is defined as the Hamming distance of $\mathbf{x}$ to the all-zero point $\mathbf{0}$.

Let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be a k-dimensional subspace, i.e. a binary linear code. We denote by d the distance of $\mathcal{C}$, which is defined as the minimal Hamming distance between two different codewords in $\mathcal{C}$. Let $\mathcal{C}$ be specified by a random parity check matrix $P \in \mathbb{F}_2^{(n-k)\times n}$, i.e. we choose each entry of P from $\mathbb{F}_2$ uniformly at random. By the definition of a parity check matrix we have

$$\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n \mid P\mathbf{c} = \mathbf{0}\}.$$

Moreover, let $\mathbf{y} = \mathbf{c} + \mathbf{e} \in \mathbb{F}_2^n, \mathbf{c} \in \mathcal{C}$ be an arbitrary point in space. By linearity, we have $\mathbf{s} := P\mathbf{y} = P\mathbf{e}$, which is called the syndrome of $\mathbf{y}$. In the *syndrome decoding problem*, one is given $P, \mathbf{y}, \omega$ and has to output a small Hamming weight $\mathbf{e}$ with $\Delta(\mathbf{e}) = \omega$ such that $P\mathbf{e} = \mathbf{s}$.

Let us assume w.l.o.g. that the last $n-k$ columns of P are linearly independent, which could be arranged via column permutations. Using Gaussian elimination – that can be expressed as left multiplication by some invertible $G \in \mathbb{F}_2^{(n-k)\times(n-k)}$ – we can transform P into systematic form $(H \mid I_{n-k})$, where I_{n-k} is the $(n-k)$-dimensional identity matrix. Our parity check identity therefore becomes

$$GP\mathbf{e} = (H \mid I_{n-k})\mathbf{e} = H\mathbf{e}' + \mathbf{e}'' = G\mathbf{s}, \text{ where } \mathbf{e} = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}.$$

Set $\bar{\mathbf{s}} := G\mathbf{s}$. Then the identity

$$H\mathbf{e}' = \bar{\mathbf{s}} \text{ holds for all } n-k \text{ but } \Delta(\mathbf{e}'') \text{ coordinates.} \tag{1}$$

All ISD algorithms enforce a special weight distribution on $\mathbf{e}$ via some column permutation π of H. For instance, in Prange's original ISD algorithm, one chooses $\Delta(\mathbf{e}') = 0$. Thus, one simply has to check – after applying π – whether

$$\Delta(\bar{\mathbf{s}}) = \Delta(\mathbf{e}'') = \omega.$$

Starting with Dumer's algorithm [Dum91], all Information Set Decoding algorithms allowed weight $\Delta(\mathbf{e}') = p > 0$ for some parameter p and introduced some parameter $\ell \le n-k-\Delta(\mathbf{e}'')$ such that Eq. (1) holds on ℓ coordinates. Mathematically, one transforms P via Gaussian elimination G' into

$$G'P = \begin{pmatrix} H_1 & 0 \\ H_2 & I_{n-k-\ell} \end{pmatrix}, \text{ where } H_1 \in \mathbb{F}_2^{\ell\times(k+\ell)} \text{ and } H_2 \in \mathbb{F}_2^{(n-k-\ell)\times(k+\ell)}.$$

Set $\bar{\mathbf{s}} := G'\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{F}_2^{\ell} \times \mathbb{F}_2^{n-k-\ell}$. Choosing some suitable $\mathbf{e}' = \mathbf{e}_1 + \mathbf{e}_2$ with $\mathbf{e}', \mathbf{e}_1, \mathbf{e}_2 \in \mathbb{F}_2^{k+\ell}$ and $\Delta(\mathbf{e}') = p$, we can write Eq. (1) as

$$H_1\mathbf{e}_1 = H_1\mathbf{e}_2 + \mathbf{s_1} \quad \text{ and } \tag{2}$$

$$\Delta(H_2\mathbf{e}_1, H_2\mathbf{e}_2 + \mathbf{s_2}) = \omega - p. \tag{3}$$

The BJMM algorithm [BJMM12] constructs two lists $L_1^{(1)}$, $L_2^{(1)}$ containing candidates $(\mathbf{e}_1, H_1\mathbf{e}_1)$ and $(\mathbf{e}_2, H_1\mathbf{e}_2 + \mathbf{s_1})$ for solutions of Eq. (2). Originally in the BJMM algorithm, Eq. (3) is checked naively by testing candidates in $L_1^{(1)} \times L_2^{(1)}$.

May and Ozerov [MO15] proposed a more involved Nearest Neighbor search (NNS) that finds elements in $L_1^{(1)} \times L_2^{(1)}$ satisfying Eq. (3). Its run time is subquadratic in the list size, thereby improving the original BJMM algorithm.

Hence, in a nutshell, BJMM provides an efficient algorithm for solving Eq. (2), whereas May-Ozerov provides an efficient algorithm for solving Eq. (3). At first sight, one might independently try to find optimal solutions for Eq. (2) and Eq. (3). This was done in the work of May, Ozerov [MO15], which mainly describes that the original BJMM can be generically sped up via Nearest Neigbor Search (NSS).

However, Eq. (2) and Eq. (3) are linked by the optimization parameters p, ℓ. Therefore, it is unclear whether optimized parameters for BJMM *without* NNS also yield an optimal combination of BJMM *with* NNS. In fact, we will show in Section 4 that in the important *Full Distance* (FD) decoding setting, where $\omega = d$, an optimized version of BJMM *with* NNS requires a depth-4 search tree. This is in contrast to the original BJMM *without* NNS, which is optimal for a depth-3 search tree.

In the following section, we describe the BJMM algorithm with a flexible search tree depth m, as opposed to the original description in [BJMM12] that fixes depth $m = 3$.

3 A General Description of BJMM with Arbitrary Depth

As described in Section 2, our goal is to construct two lists $L_1^{(1)}, L_2^{(1)}$ that contain candidates of the form $(\mathbf{e}_1, H_1\mathbf{e}_1)$ and $(\mathbf{e}_2, H_1\mathbf{e}_2 + \mathbf{s_1})$, respectively. The reader is advised to follow the construction steps also via Fig. 1.

Tree construction in depth 1. In BJMM, the vectors $\mathbf{e}_1, \mathbf{e}_2$ have weight

$$p_1 \geq \frac{p}{2} \text{ for some } p_1 \text{ that has to be optimized.}$$

Recall from Eq. (2) that we construct $\mathbf{e}' = \mathbf{e}_1 + \mathbf{e}_2$ with $\Delta(\mathbf{e}') = p$. Hence, we call $(\mathbf{e}_1, \mathbf{e}_2)$ a *representation* of $\mathbf{e}'$ if the sum $\mathbf{e}_1 + \mathbf{e}_2$ has the correct weight p.

Vice versa, every fixed $\mathbf{e}' \in \mathbb{F}_2^{k+\ell}$ with weight p has

$$R_1 := \binom{p}{p/2}\binom{k+\ell-p}{p_1 - p/2} \text{ representations.}$$

Namely, the p 1-coordinates in $\mathbf{e}'$ can be represented as $0+1$ or $1+0$ additions from the corresponding coordinates in $\mathbf{e}_1, \mathbf{e}_2$. Hence fixing $p/2$ ones (and $p/2$ zeros) in $\mathbf{e}_1$ in the 1-coordinates of $\mathbf{e}'$ already determines the entries in $\mathbf{e}_2$ in those coordinates.

Similarly, the $k+\ell-p$ 0-coordinates in $\mathbf{e}'$ can be represented as $0+0$ or $1+1$ additions from $\mathbf{e}_1, \mathbf{e}_2$. Hence fixing the remaining $p_1 - p/2$ ones in $\mathbf{e}_1$ in the

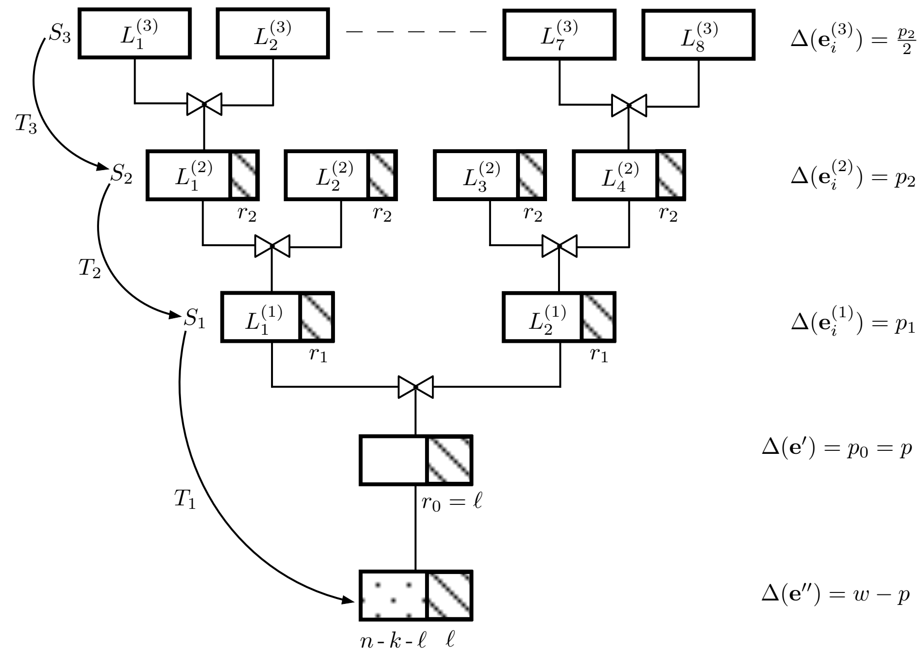

Fig. 1: The original BJMM(3) algorithm without Nearest Neighbors.

0-coordinates of $\mathbf{e}'$ determines the entries in $\mathbf{e}_2$ also to ones in the corresponding coordinates.

The goal in BJMM is now to construct only an $\frac{1}{R_1}$-fraction of $L_1^{(1)}, L_2^{(1)}$, since then in expectation *one representation* of the desired solution $\mathbf{e}'$ of the syndrome decoding problem survives. This is done by constructing only those candidates $(\mathbf{e}_1, H_1\mathbf{e}_1)$ and $(\mathbf{e}_2, H_1\mathbf{e}_2 + \mathbf{s_1})$, for which $H_1\mathbf{e}_1$ and $H_1\mathbf{e}_2 + \bar{\mathbf{s}}_\mathbf{1}$ take a certain (random) value $t_1^{(1)}$ on their last $r_1 = \lfloor \log_2 R_1 \rfloor$ coordinates. In expectation this leads to at least an $\frac{1}{R_1}$-fraction of all R_1 representations of $\mathbf{e}'$.

More precisely, we define by $\mathbf{v}_{[r_1]} \in \mathbb{F}_2^{r_1}$ the projection of any vector $\mathbf{v} \in \mathbb{F}_2^{\ell}$ onto its last r_1 coordinates. Then we compute for some random $t_1^{(1)} \in_R \mathbb{F}_2^{r_1}$ and $t_2^{(1)} := \mathbf{s}_{1[r_1]} + t_1^{(1)}$ the lists

$$L_i^{(1)} = \{(\mathbf{e}_i^{(1)}, H_1\mathbf{e}_i^{(1)}) \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{\ell} \mid \mathit{\Delta}(\mathbf{e}_i^{(1)}) = p_1 \wedge (H_1\mathbf{e}_i^{(1)})_{[r_1]} = t_i^{(1)}\} \text{ for } i = 1, 2.$$

As discussed before, the expected size of the lists is

$$S_1 := \mathbb{E}[|L_1^{(1)}|] = \frac{\binom{k+\ell}{p_1}}{2^{r_1}}.$$

Recall that by Eq. (2) we are looking for solutions to $H_1\mathbf{e}_1^{(1)} = H_1\mathbf{e}_2^{(1)} + \mathbf{s_1} \in \mathbb{F}_2^{\ell}$. Since the elements in $L_1^{(1)}, L_2^{(2)}$ already fulfill this identity on r_1 coordinates, we are looking for matching vectors in $L_1^{(1)} \times L_2^{(2)}$ on the remaining $\ell - r_1$ bits. As was shown in [BJMM12], this can be done by a simple matching algorithm in expected time

$$T_1 := \max\{S_1, \frac{S_1^2}{2^{\ell - r_1}}\}.$$

This ends the description of the tree construction in depth 1.

Tree construction in depth $\mathbf{2, \ldots, m-1}$. In depth 2, the process from depth 1 is repeated recursively. Let us describe the construction of lists $L_1^{(j)}, L_2^{(j)}$ in depth j with $2 \leq j < m$. The remaining lists $L_3^{(j)}, \ldots, L_{2^j}^{(j)}$ are constructed analogously.

We define vectors $\mathbf{e}_1^{(j)}, \mathbf{e}_2^{(j)}$ of weight

$$p_j \geq \frac{p_{j-1}}{2} \text{ for some } p_j \text{ that has to be optimized.}$$

This definition also holds for $j = 1$ if we set $p_0 := p$. For every fixed $\mathbf{e}_1^{(j-1)} = \mathbf{e}_1^{(j)} + \mathbf{e}_2^{(j)} \in \mathbb{F}_2^{k+\ell}$ with weight p_{j-1} this results in

$$R_j := \binom{p_{j-1}}{p_{j-1}/2}\binom{k + \ell - p_{j-1}}{p_j - p_{j-1}/2} \text{ representations. Set } r_j = \lfloor \log R_j \rfloor.$$

One chooses $t_1^{(j)} \in_R \mathbb{F}_2^{r_j}$ and sets $t_2^{(j)} := t_1^{(j)} + t_1^{(j-1)}$. Then one defines the lists

$$L_i^{(j)} = \{(\mathbf{e}_i^{(j)}, H_1\mathbf{e}_i^{(j)}) \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{\ell} \mid \Delta(\mathbf{e}_i^{(j)}) = p_j \wedge (H_1\mathbf{e}_i^{(j)})_{[r_j]} = t_i^{(j)}\} \text{ for } i = 1, 2$$

with expected list sizes of

$$S_j := \mathbb{E}[|L_i^{(j)}|] = \frac{\binom{k+\ell}{p_j}}{2^{r_j}}.$$

The matching of $L_1^{(j)}, L_2^{(j)}$ to $L_1^{(j-1)}$ can be done in expected time

$$T_j := \max\{S_j, \frac{S_j^2}{2^{r_{j-1} - r_j}}\}.$$

This formula also holds for $j = 1$ by setting $r_0 := \ell$.

Tree construction in depth m. Let us describe how to construct $L_1^{(m-1)}$ out of two lists $L_1^{(m)}, L_2^{(m)}$. The construction of $L_3^{(m)}, \ldots, L_{2^m}^{(m)}$ is analogous.

We represent $\mathbf{e}_1^{(m-1)} = \mathbf{e}_1^{(m)} + \mathbf{e}_2^{(m)} \in \mathbb{F}_2^{k+\ell}$, where

$$p_m = \Delta(\mathbf{e}_1^{(m)}) = \Delta(\mathbf{e}_2^{(m)}) := \frac{p_{m-1}}{2} \text{ and } \mathbf{e}_1^{(m)} \in 0^{\frac{k+\ell}{2}} \times \mathbb{F}_2^{\frac{k+\ell}{2}}, \mathbf{e}_2^{(m)} \in \mathbb{F}_2^{\frac{k+\ell}{2}} \times 0^{\frac{k+\ell}{2}}.$$

Let us define the lists

$$L_i^{(m)} = \{(\mathbf{e}_i^{(m)}, H_1\mathbf{e}_i^{(m)}) \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{\ell} \mid \Delta(\mathbf{e}_i^{(m)}) = p_m\} \text{ for } i = 1, 2$$

with size

$$S_m := \binom{\frac{k+\ell}{2}}{p_m}.$$

The matching of $L_1^{(m)}, L_2^{(m)}$ to $L_1^{(m-1)}$ can be done in expected time

$$T_m := \max\{S_m, \frac{S_m^2}{2^{r_{m-1}}}\}.$$

The formula for T_m coincides with the general formula for T_j by setting $r_m := 0$.

Total complexity of the generalized BJMM. On every level j of our search tree we consume expected time T_j and space S_j. Thus in total, we obtain

$$\text{time } T = \max_{1 \le j \le m} \{T_j\} \text{ and space } S = \max_{1 \le j \le m} \{S_j\}.$$

If we are using BJMM with Nearest Neighbor Search (NNS) from [MO15], we have to replace T_1 with

$$T_1 := 2^{y(\frac{\log |S_1|}{n-k-\ell}, \frac{\omega-p}{n-k-\ell})(n-k-\ell)}, \text{ where } y(\lambda,\gamma) := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda) - \frac{\gamma}{2}}{1-\gamma}\right)\right).$$

This results in a slight modification on level 1 of the search tree, as shown in Fig. 2 for BJMM(4). The lists $L_1^{(1)}$ and $L_2^{(1)}$ are now merged via NNS on $n-k-\ell$ bits.

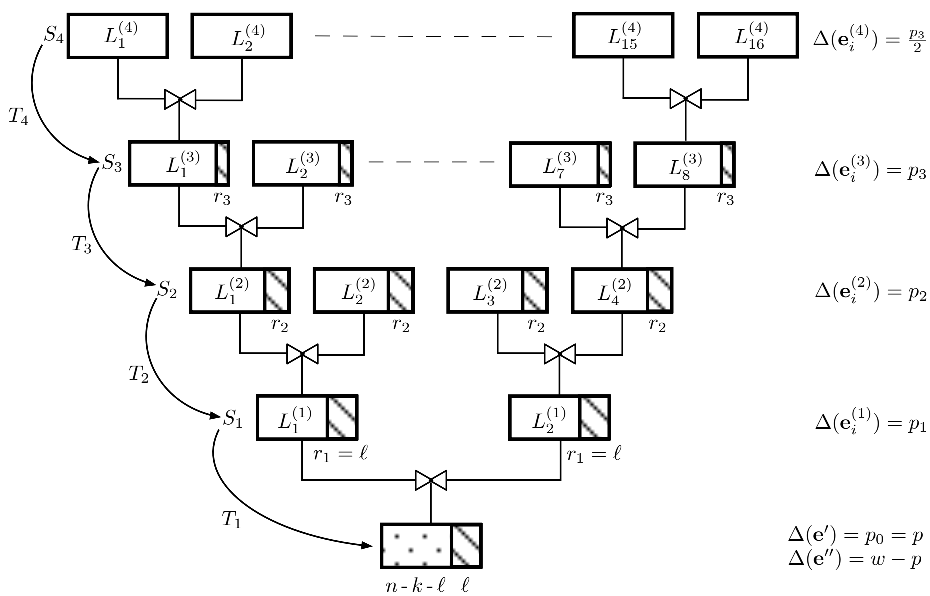

Fig. 2: The BJMM(4) algorithm with Nearest Neighbors.

Total complexity of decoding. As described in Section 2, any algorithm succeeds in constructing a solution of Eq. (2) if and only if the column permutation π induces the correct weight distribution

$$\Delta(\mathbf{e}') = p,\ \Delta(\mathbf{e}'') = \omega - p \text{ on } \mathbf{e}' = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{n-k-\ell}.$$

This happens with probability

$$P_{succ} = \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{\omega-p}}{\binom{n}{\omega}}.$$

Thus, the total expected running time of our decoding algorithm is $T \cdot P_{succ}^{-1}$.

4 Results

In this section, we state upper bounds for the complexity of decoding k-dimensional binary linear codes of length n in various settings. Random linear codes asymptotically achieve relative distance $\frac{d}{n} = H^{-1}\left(1 - \frac{k}{n}\right)$, equal to the Gilbert Varshamov bound. Thus, for Full Distance decoding we set $\omega = d$, whereas for Half Distance decoding we set $\omega = d/2$.

We optimize BJMM(m) for running time T over a large range of rates $\frac{k}{n}$, where we eventually only state the result for the worst-case rate. Running times for other rates may be significantly lower, but can be analyzed in the same manner, e.g. using our code from `https://github.com/LeifBoth/bjmm2.0-code`. Moreover, we state T in the form 2^{cn} for some complexity exponent c that we round up to the 4^{th} digit after the decimal point. This rounding takes account of all polynomial factors that are neglected in our analysis, and provides an upper bound for the running time.

In the following analysis, we use the notion and formulas derived in Section 3.

Theorem 1. *Full Distance decoding for random binary linear codes can be achieved in expected time $2^{0.0953n}$ using $2^{0.0915n}$ space.*

Proof. We use BJMM(4) in combination with Nearest Neighbors. The maximal running time is achieved at rate

$$\frac{k}{n} = 0.423 \text{ with relative distance } \frac{\omega}{n} = \frac{d}{n} = H^{-1}\left(1 - \frac{k}{n}\right) = 0.1373.$$

For this rate we obtain minimal running time using the (relative) parameters

$$\frac{\ell}{n} = 0.2635, \quad \frac{p_0}{n} = 0.0825, \quad \frac{p_1}{n} = 0.0734, \quad \frac{p_2}{n} = 0.0521, \quad \frac{p_3}{n} = 0.0298.$$

This results in the following number of representations

$$R_1 = 2^{0.2635n}, \quad R_2 = 2^{0.1771n}, \quad R_3 = 2^{0.0856n}$$

and list sizes

$$S_1 = 2^{0.0731n}, \quad S_2^{0.0888n}, \quad S_3 = 2^{0.0915n}, \quad S_4 = 2^{0.0915n}.$$

The running times for each level of the search tree are balanced out as

$$T_1 = 2^{0.0915n}, \quad T_2 = 2^{0.0913n}, \quad T_3 = 2^{0.0915n}, \quad T_4 = 2^{0.0915n}.$$

Since the probability for the correct weight distribution is

$$P_{succ} = 2^{-0.0038n},$$

we obtain an overall running time and space consumption of

$$T = 2^{0.0953n} \text{ and } S = 2^{0.0915n}. \quad \square$$

(FD)

m	k	w	$\log(T)$	$\log(S)$	ℓ	p_0	p_1	p_2	p_3	p_4
3	0.422	0.1376	**0.0967**	**0.0873**	0.1901	0.0649	0.0528	0.0304	(0.0152)	-
4	0.423	0.1373	**0.0953**	**0.0915**	0.2635	0.0825	0.0734	0.0521	0.0298	(0.0149)
5	0.420	0.1384	**0.0953**	**0.0910**	0.2632	0.0820	0.0732	0.0522	0.0301	0.0153

(HD)

m	k	w	$\log(T)$	$\log(S)$	ℓ	p_0	p_1	p_2	p_3
2	0.458	0.0622	**0.0492**	**0.0282**	0.0310	0.0107	0.0075	(0.0038)	-
3	0.474	0.0594	**0.0473**	**0.0363**	0.0663	0.0177	0.0150	0.0090	(0.0045)
4	0.475	0.0592	**0.0473**	**0.0351**	0.0635	0.0168	0.0143	0.0085	0.0043

(McEliece w/o NNS)

m	k	w	$\log(T)$	$\log(S)$	ℓ	p_0	p_1	p_2	p_3
2	0.775	0.02	**0.0370**	**0.0255**	0.0509	0.0087	0.0060	(0.0030)	-
3	0.775	0.02	**0.0362**	**0.0262**	0.0610	0.0096	0.0074	0.0038	(0.0019)
4	0.775	0.02	**0.0362**	**0.0271**	0.0631	0.0100	0.0077	0.0039	0.0020

(McEliece w/ NNS)

m	k	w	$\log(T)$	$\log(S)$	ℓ	p_0	p_1	p_2	p_3
2	0.775	0.02	**0.0362**	**0.0264**	0.0280	0.0087	0.0063	(0.0031)	-
3	0.775	0.02	**0.0350**	**0.0280**	0.0429	0.0103	0.0086	0.0048	(0.0024)
4	0.775	0.02	**0.0350**	**0.0280**	0.0429	0.0103	0.0086	0.0048	0.0048

Fig. 3: Upper bounds for time and space, and their optimized parameters. All values are stated relative to n. The optimized parameters all have precision 10^{-4}.

Fig. 3 provides optimized parameters as well as the resulting run time and space consumption for Full Distance (FD) decoding, half distance (HD) decoding and McEliece. For McEliece, we took parameters $\frac{k}{n} = 0.775$, $\frac{w}{n} = 0.02$ as suggested in the Goppa code instantiations in Section 7 of [BLP08]. Fig. 4 provides more fine-grained information for the time and space consumption on each level of the search tree in FD.

m	$\log(T_1)$	$\log(T_2)$	$\log(T_3)$	$\log(T_4)$	$\log(T_5)$	$\log(S_1)$	$\log(S_2)$	$\log(S_3)$	$\log(S_4)$	$\log(S_5)$
3	0.0873	0.0873	0.0873	-	-	0.0692	0.0873	0.0873	-	-
4	0.0915	0.0913	0.0915	0.0915	-	0.0731	0.0888	0.0915	0.0915	-
5	0.0910	0.0910	0.0910	0.0909	0.0910	0.0725	0.0881	0.0909	0.0727	0.0910

(FD)

Fig. 4: Run time and space for all search tree levels (all values relative to n).

FD decoding. We reproduce the complexity exponent 0.0967 for BJMM(3) from [MO15]. However as already shown in Theorem 1, we achieve an improved exponent 0.0953 for BJMM(4), whereas BJMM(5) does not improve further on time – but slightly on space. Interestingly, for BJMM(4) the running times $T_1, \ldots, T_4$ in Fig. 4 equal the large space consumption S and also ℓ, p are quite large. This means that – up to the outer loop for finding a suitable permutation π – BJMM(4) consumes in the inner loop as much space as time, and almost all work is shifted to the inner loop.

Remark on the quantum complexity of ISD algorithms. Recently, there has been progress on transferring ISD algorithms to the quantum setting. For some time, the only known quantum ISD version was Prange's algorithm enhanced with a Grover search for π on the outer loop [Ber10]. Recently, Kachigar and Tillich [KT17] showed that the inner loop – with algorithms like [MMT11] and [BJMM12] – can also be sped up quantumly using quantum random walks. However, Kachigar and Tillich also point out that the complexity improvement is not as significant as in the classical ISD setting.

Our computations provide some explanation for this behavior. Whereas in Prange's ISD algorithm, *all* computation is done in the outer loop for π, BJMM(4) shifts *almost all* computation to the inner loop (according to Fig. 3 0.0915 out of 0.0953 is spent in the inner loop). But Grover search for the outer loop yields a square root improvement, where as quantum random walks yield only a $\frac{2}{3}$-root improvement.

Thus, recent classical ISD algorithms are not optimal in the sense of allowing a complexity preserving transfer to the quantum setting, both in terms of time and space.

HD decoding. As opposed to FD, in the HD case we obtain optimality of the running time already for BJMM(3), thereby reproducing the result of [MO15]. However, while BJMM(4) does not improve the run time, it nevertheless provides a small improvement in space consumption.

McEliece. [BJMM12] already analyzed McEliece parameters for BJMM(3), but the authors inadvertently make the choice $\frac{\omega}{n} = \frac{d}{n} \approx 0.04$, instead of $\frac{\omega}{n} = \frac{t}{n} \approx \frac{d}{2n} \approx 0.02$ for t introduced errors in McEliece encryption, leading to largely overestimated run times.

We achieve $\log(T) = 0.0362n$ without NNS and $\log(T) = 0.0350n$ with NNS. Bernstein, Lange and Peters [BLP08] suggest to use

$$n = 1632, 2960, 6624 \text{ for respective bit security levels of } 80, 128, 256.$$

Naively plugging these values of n into

- $\log(T) = 0.0362n$ yields (rounded) values $59, 107, 240$,
- $\log(T) = 0.0350n$ yields (rounded) values $57, 104, 232$.

These values are certainly *not the bit security levels* of the suggested McEliece instantiations in practice, since our asymptotic analysis neglects all polynomial factors. Nevertheless, they give some cryptanalytic hope that the proposed McEliece instances can be attacked with significant less effort than predicted a decade ago.

References

Al 01. A. Kh. Al Jabri. A statistical decoding algorithm for general linear block codes. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 1–8, Cirencester, UK, December 17–19, 2001. Springer, Heidelberg, Germany.

Ale03. Michael Alekhnovich. More on average case vs approximation complexity. In *44th Annual Symposium on Foundations of Computer Science*, pages 298–307, Cambridge, Massachusetts, USA, October 11–14, 2003. IEEE Computer Society Press.

Bar97. Alexander Barg. Complexity issues in coding theory. 1997.

BCJ11. Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

Ber10. Daniel J Bernstein. Grover vs. mceliece. In *International Workshop on Post-Quantum Cryptography*, pages 73–80. Springer, 2010.

BJMM12. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

BLP08. Daniel J Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the mceliece cryptosystem. In *International Workshop on Post-Quantum Cryptography*, pages 31–46. Springer, 2008.

BLP11. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

DT17. Thomas Debris-Alazard and Jean-Pierre Tillich. Statistical decoding. *CoRR*, abs/1701.07416, 2017.

Dum91. Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, 1991.

FKI07. Marc P. C. Fossorier, Kazukuni Kobara, and Hideki Imai. Modeling bit flipping decoding based on nonorthogonal check sums with application to iterative decoding attack of mceliece cryptosystem. *IEEE Trans. Information Theory*, 53(1):402–411, 2007.

HGJ10. Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

KT17. Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. *arXiv preprint arXiv:1703.00263*, 2017.

McE78. RJ McEliece. A public-key system based on algebraic coding theory, 114-116. deep sace network progress report, 44. *Jet Propulsion Laboratory, California Institute of Technology*, 1978.

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\wr}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

MO15. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

Ove06. Raphael Overbeck. Statistical decoding revisited. In Lynn Margaret Batten and Reihaneh Safavi-Naini, editors, *ACISP 06: 11th Australasian Conference on Information Security and Privacy*, volume 4058 of *Lecture Notes in Computer Science*, pages 283–294, Melbourne, Australia, July 3–5, 2006. Springer, Heidelberg, Germany.

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, Maryland, USA, May 22–24, 2005. ACM Press.

Ste88. Jacques Stern. A method for finding codewords of small weight. In *International Colloquium on Coding Theory and Applications*, pages 106–113. Springer, 1988.