Decoding Linear Codes with High Error Rate and its Impact for LPN 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 propose a new algorithm for the decoding of random binary linear codes of dimension n that is superior to previous algorithms for high error rates. In the case of Full Distance decoding, the best known bound of $2^{0.0953n}$ is currently achieved via the BJMM-algorithm of Becker, Joux, May and Meurer. Our algorithm significantly improves this bound down to $2^{0.0885n}$.
Technically, our improvement comes from the heavy use of Nearest Neighbor techniques in all steps of the construction, whereas the BJMM-algorithm can only take advantage of Nearest Neighbor search in the last step.
Since cryptographic instances of LPN usually work in the high error regime, our algorithm has implications for LPN security.

Keywords: Decoding binary linear codes, BJMM, Nearest Neighbors, LPN, Full Distance Decoding, Representations

1 Introduction

The NP-hard decoding problem for random linear codes plays a major role in coding and complexity theory. It is especially suitable for the construction of quantum-secure cryptographic systems like [McE78,Ale03,Reg05]. In view of the upcoming NIST selection of post-quantum public-key cryptosystems [NIS] it is of crucial importance for secure parameter selection to know the best decoding algorithms.

A linear code $\mathcal{C}$ is a k-dimensional subspace of $\mathbb{F}_2^n$. In the decoding problem the attacker gets an erroneous version $\mathbf{x} = \mathbf{c} + \mathbf{e}$ of a codeword $\mathbf{c}$ for some error vector $\mathbf{e}$ with Hamming weight $\Delta(\mathbf{e}) = \omega$. His target is to find $\mathbf{e}$ in order to recover the original codeword $\mathbf{c}$. Sometimes, the weight ω is bounded by the distance d of the code $\mathcal{C}$ (Full Distance Decoding) or by $\frac{d}{2}$ (Half Distance Decoding).

Therefore the running time $T(n, k, d)$ of any decoding algorithm is a function of the parameters n, k and d. It is well known that the Gilbert-Varshamov bound gives us $\frac{k}{n} \approx 1 - H(\frac{d}{n})$ for random linear codes, where $H(\cdot)$ is the binary entropy function $H(p) := -p\log(p) - (1-p)\log(1-p)$. This results in a running time

$T(n,k)$ which is a function of n and k only. Furthermore one often compares worst case running times where we maximize the running time over all rates $\frac{k}{n}$ resulting in a running time $T(n)$.

The best algorithmic paradigm that we know today for random binary linear codes is a class of algorithms called *Information Set Decoding* (ISD). Here, for simplicity we only compare ISD running times in the Full Distance Decoding setting, but see also Fig. 1. For all ISD algorithms the maximal run time is achieved at a rate $\frac{k}{n}$ slightly below $\frac{1}{2}$.

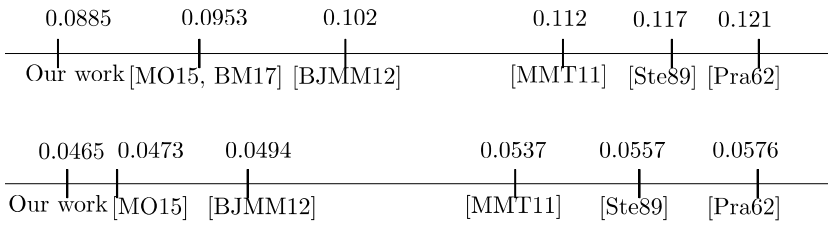

Fig. 1: Comparison for Full/Half Distance of our work and other algorithms.

The first ISD algorithm is due to Prange [Pra62] and achieves worst case running time $2^{0.121n}$. This was improved by Stern and Dumer [Ste88,Dum91] to $2^{0.117n}$. Using the representation technique, May, Meurer, Thomae [MMT11] and later Becker, Joux, May, Meurer [BJMM12] further decreased the run time to $2^{0.112n}$ and $2^{0.102n}$, respectively. The last is called BJMM algorithm and is currently asymptotically the best algorithm for decoding of random linear codes.

In 2015, May and Ozerov [MO15] proposed some Nearest Neighbor (NN) search that further sped up BJMM to $2^{0.0967n}$, which was later optimized in [BM17b] to $2^{0.0953n}$.

Our results. As can be seen from Fig. 1, our new algorithm achieves in the Full Distance Decoding setting $2^{0.0885n}$, which is a quite remarkable improvement over the current state of the art. However, the improvement for the Half Distance Decoding is comparably small. As a rule of thumb, the larger the error rate, the more significant our algorithm's improvement.

As most promising in cryptographic settings, we currently see the application of our algorithm for Learning Parity with Noise (LPN) instances. Every LPN instance of dimension k with error τ is naturally a decoding problem for a random linear code. As shown by Esser, Kübler, May [EKM17] in practice one currently best solves large LPN instances by a hybrid approach. Namely, one first applies a dimension reduction algorithm (such as BKW [GJL14]) at the cost of introducing a large error close to $\frac{1}{2}$, followed by a decoding algorithm. Since our algorithm works especially well in the high-error regime, it seems to be a perfect candidate for solving these transformed LPN instances.

Our algorithm. ISD algorithms with representation technique such as MMT and BJMM currently use a 2-step matching process, where in the first step one does an exact matching of vectors (for eliminating representations) and in a second step one does an approximate matching via NN search. We eliminate this two-step process and perform only an approximate matching in all stages of the algorithm.

This allows us to eliminate representations less restrictive, and to use the full power of NN search in every step of our algorithm. Thus, our approximate matching is in spirit similar to the Ball Decoding approach of Bernstein, Lange and Peters [BLP11]. The heavy use of NN search might also explain the large improvement (only) in the high error-regime, where NN search can show its full strength.

This paper is organized as follows. In Section 2, we review some ISD algorithms. Section 3 introduces a basic version of our new algorithm, whereas the generalized version is given in Section 4. Our results are provided in Section 5.

2 Preliminaries

Syndrome Decoding. Let us start with some preliminaries on linear codes and decoding algorithms. We denote the *Hamming distance* of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ by $\Delta(\mathbf{x}, \mathbf{y})$. The Hamming weight $\Delta(\mathbf{x})$ of $\mathbf{x}$ is defined as the Hamming distance of $\mathbf{x}$ to the zero vector $\mathbf{0}$.

A linear code $\mathcal{C}$ is a k-dimensional subspace of $\mathbb{F}_2^n$. Its distance is defined by $d := \min_{\mathbf{c} \neq \mathbf{c}' \in \mathcal{C}} \{\Delta(\mathbf{c}, \mathbf{c}')\}$. We can specify $\mathcal{C}$ by a generator matrix $G \in \mathbb{F}_2^{k \times n}$ or a parity check matrix $P \in \mathbb{F}_2^{(n-k) \times n}$ via

$$\mathcal{C} := \{\mathbf{x}G \in \mathbb{F}_2^n \mid \mathbf{x} \in \mathbb{F}_2^k\} \text{ or } \mathcal{C} := \{\mathbf{c} \in \mathbb{F}_2^n \mid P\mathbf{c} = \mathbf{0}\}.$$

Random linear codes have a random G or random P, where in both cases each matrix entry is chosen uniformly at random from $\mathbb{F}_2$. For an arbitrary vector $\mathbf{y} = \mathbf{c} + \mathbf{e} \in \mathbb{F}_2^n, \mathbf{c} \in \mathcal{C}$ we define the syndrome of $\mathbf{y}$ as

$$\mathbf{s} := P\mathbf{y} = P\mathbf{c} + P\mathbf{e} = P\mathbf{e}. \tag{1}$$

Definition 1 (Syndrome Decoding Problem). *Let $\mathcal{C}$ be a linear code specified by some parity check matrix $P \in \mathbb{F}_2^{(n-k) \times n}$. Given P, an (erroneous) codeword $\mathbf{y} \in \mathbb{F}_2^n$ and a weight $\omega \in \mathbb{N}$, one has to find an error vector $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathbf{y} + \mathbf{e} \in \mathcal{C}$ and $\Delta(\mathbf{e}) = \omega$.*

We call $(P, \mathbf{s}, \omega)$ with $\mathbf{s} = P\mathbf{y}$ an instance of the Syndrome Decoding Problem. We say that $\mathbf{e} \in \mathbb{F}_2^n$ solves $(P, \mathbf{s}, \omega)$ iff $\mathbf{s} = P\mathbf{e}$ and $\Delta(\mathbf{e}) = \omega$.

A fundamental algorithm for solving the Syndrome Decoding Problem was introduced by Prange [Pra62]. This algorithm is the basis of all of today's so-called Information Set Decoding (ISD) algorithms [Ste88,Dum91,BLP11,MMT11,BJMM12].

In Prange's algorithm, one reduces the dimension of the search space from n down to k via Gaussian elimination.

In more detail, one chooses some invertible $G \in \mathbb{F}_2^{(n-k)\times(n-k)}$ such that $GP = (\bar{P} \mid I_{n-k})$, where I_{n-k} is the $(n-k)$-dimensional identity matrix. Therefore Eq. (1) becomes

$$GP\mathbf{e} = (\bar{P} \mid I_{n-k})\mathbf{e} = \bar{P}\mathbf{e}' + \mathbf{e}'' = G\mathbf{s} =: \bar{\mathbf{s}}, \text{ with } \mathbf{e} = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}. \quad (2)$$

Thus, every instance $(P, \mathbf{s}, \omega)$ with $P \in \mathbb{F}_2^{(n-k)\times n}$ of the Decoding Problem has some (non-unique) standard form $(\bar{P}, \bar{\mathbf{s}}, \omega)$ with $\bar{P} \in \mathbb{F}_2^{(n-k)\times k}$ such that $\mathbf{e} \in \mathbb{F}_2^n$ solves $(P, \mathbf{s}, \omega)$ iff $(\bar{P} \mid I_{n-k})\mathbf{e} = \bar{\mathbf{s}}$.

Definition 2 (Standard form). *For any instance* $(P, \mathbf{s}, \omega) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{n-k} \times \mathbb{N}$ *of the decoding problem, we say that* $(\bar{P}, \bar{\mathbf{s}}, \omega) \in \mathbb{F}_2^{(n-k)\times k} \times \mathbb{F}_2^{n-k} \times \mathbb{N}$ *is a* standard form *of* $(P, \mathbf{s}, \omega)$ *if there exists some invertible* $G \in \mathbb{F}_2^{(n-k)\times(n-k)}$ *such that*

$$GP = (\bar{P} \mid I_{n-k}) \text{ and } G\mathbf{s} = \bar{\mathbf{s}}.$$

The underlying idea of all Information Set Decoding algorithm is to solve a dimension-reduced standard form $(\bar{P}, \bar{\mathbf{s}}, \omega)$ of a Decoding Problem instance instead of its original form $(P, \mathbf{s}, \omega)$.

However, before transforming $(P, \mathbf{s}, \omega)$ to its normal form one applies some column permutation π to P to enforce a special weight distribution on $\mathbf{e} = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$. While Prange chooses $\Delta(\mathbf{e}') = 0$, other ISD algorithms enforce $\Delta(\mathbf{e}') = p$ for some parameter $p \geq 0$ (see Algorithm 1 and 2). Thus it is sufficient to find some $\mathbf{e}' \in \mathbb{F}_2^k$, $\Delta(\mathbf{e}') = p$ such that after applying π and converting to standard form the term $\bar{P}\mathbf{e}'$ is close to $\bar{\mathbf{s}}$, i.e.,

$$\Delta(\bar{P}\mathbf{e}', \bar{\mathbf{s}}) = \Delta(\mathbf{e}'') = \omega - p.$$

Algorithm 1: ISD – Weight Distribution and Standard Form

Input : $P \in \mathbb{F}_2^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_2^{n-k}$, $\omega \in \mathbb{N}$
Output: $\mathbf{e} \in \mathbb{F}_2^n$ with $P\mathbf{e} = \mathbf{s}$ and $\Delta(\mathbf{e}) = \omega$
repeat
  repeat
    $\pi \leftarrow$ random permutation on $\mathbb{F}_2^n$
    $(\cdot \mid Q) \leftarrow \pi(P)$ (permute columns) ▷ $Q \in \mathbb{F}_2^{(n-k)\times(n-k)}$
  until *Q is invertible*
  $(\bar{P} \mid I_{n-k}) \leftarrow G\pi(P)$ and $\bar{\mathbf{s}} \leftarrow G\mathbf{s}$ ▷ $G \in \mathbb{F}^{(n-k)\times(n-k)}$
  $(\mathbf{e}', \mathbf{e}'') = \text{ISDSolve}(\bar{P}, \bar{\mathbf{s}}, \omega)$ ▷ See Algorithm 2.
until $(\mathbf{e}', \mathbf{e}'') \neq \perp$
return $\pi^{-1}(\mathbf{e}' || \mathbf{e}'')$

Algorithm 2: ISDSolve

Input : $\bar{P} \in \mathbb{F}_2^{(n-k)\times k}, \bar{\mathbf{s}} \in \mathbb{F}_2^{n-k}, \omega \in \mathbb{N}$
Output : $(\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$
Parameters: choose optimal $0 \leq p \leq \omega$
for $\mathbf{e}' \in \mathbb{F}_2^k$ *with* $\Delta(\mathbf{e}') = p$ **do**
  $\mathbf{e}'' \leftarrow H\mathbf{e}' + \bar{\mathbf{s}}$
  if $\Delta(\mathbf{e}'') = \omega - p$ **then return**$(\mathbf{e}', \mathbf{e}'')$
end
return $\perp$

Dumer's ISD-algorithm [Dum91] introduces another parameter ℓ and transforms P into a different standard form

$$G'P = \begin{pmatrix} \bar{P}_1 & 0 \\ \bar{P}_2 & I_{n-k-\ell} \end{pmatrix}, \text{ where } \bar{P}_1 \in \mathbb{F}_2^{\ell\times(k+\ell)} \text{ and } \bar{P}_2 \in \mathbb{F}_2^{(n-k-\ell)\times(k+\ell)}.$$

Set $\bar{\mathbf{s}} := G'\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{F}_2^{\ell} \times \mathbb{F}_2^{n-k-\ell}$. We can now write Eq. (2) as

$$\bar{P}_1\mathbf{e}_1^{(1)} = \bar{P}_1\mathbf{e}_2^{(1)} + \mathbf{s}_1 \quad \text{and} \tag{3}$$

$$\Delta(\bar{P}_2\mathbf{e}_1^{(1)}, \bar{P}_2\mathbf{e}_2^{(1)} + \mathbf{s}_2) = \omega - p. \tag{4}$$

splitting $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ with $\mathbf{e}', \mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)} \in \mathbb{F}_2^{k+\ell}$. Hence, by Eq. (3) we have an *exact* matching of $\bar{P}\mathbf{e}'$ and $\bar{\mathbf{s}}$ on ℓ coordinates, and by Eq. (4) an *approximate* matching of the same vectors on the remaining $n-k-\ell$ coordinates.

The BJMM algorithm [BJMM12] solves the exact matching of Eq. (3). In a nutshell, BJMM constructs solutions $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ for Eq. (3) using some depth-3 binary search tree. For optimizing the depth of this search tree, see [BM17b]. All candidate solutions $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ are then checked via Eq. (4).

For the approximate matching, May and Ozerov [MO15] proposed a Nearest Neighbor (NN) search algorithm that, given two lists L_1, L_2, finds in time subquadratic of the list lengths all elements $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$ within some given Hamming distance $\Delta(\mathbf{x}_1, \mathbf{x}_2)$. Thus, May-Ozerov NN search can be used to speed up the check of candidate solutions via Eq. (4) inside the BJMM algorithm.

Theorem 1 ([MO15]). *Given two lists L_1, L_2 with elements taken uniformly at random from $\mathbb{F}_2^n$ and length $|L_1|, |L_2| \leq 2^{\lambda n}$. Then for any $\epsilon > 0$ one can find all but a negligible fraction of the pairs $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$ satisfying $\Delta(\mathbf{x}_1, \mathbf{x}_2) \leq \gamma n$ for some given $0 \leq \gamma \leq \frac{n}{2}$ provided that $\lambda < 1 - H(\frac{\gamma}{2})$ in time*

$$2^{(y(\lambda,\gamma)+\epsilon)n}, \text{ where } y(\lambda,\gamma) := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda) - \frac{\gamma}{2}}{1-\gamma}\right)\right).$$

Please notice that Theorem 1 can only be applied for parameters satisfying the condition $\lambda < 1 - H(\frac{\gamma}{2})$, which will not always be the case for our new

decoding algorithm. Whenever this condition is violated, we will choose one of the following two simple NN search algorithms Alg. 3 or 4.

Algorithm 3: NN-Enumerate-Pairs

Input : $L_1, L_2 \subset \mathbb{F}_2^n$, γ
Output: L
for $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$ **do**
  if $\Delta(\mathbf{x}_1, \mathbf{x}_2) \leq \gamma n$ **then** $L \leftarrow (\mathbf{x}_1, \mathbf{x}_2)$
end
return L

Since Algorithm 3 simply tests the distance of all pairs in $L_1 \times L_2$, it runs in time quadratic in the list lengths

$$2^{2\lambda n}. \tag{5}$$

Notice that here, as in the rest of the paper, we neglect for ease of presentation polynomial factors in the run time.

Algorithm 4: NN-Meet-in-the-Middle

Input : $L_1, L_2 \subset \mathbb{F}_2^n$, γ
Output: L
$L_2' \leftarrow \emptyset$
for $\mathbf{x}_2 \in L_2$ **do**
  for $\mathbf{e} \in \mathbb{F}_2^n$ *with* $\Delta(\mathbf{e}) \leq \frac{\gamma}{2} n$ **do**
    $L_2' \leftarrow L_2' \cup (\mathbf{x}_2 + \mathbf{e}, \mathbf{x}_2)$
  end
end
for $\mathbf{x}_1 \in L_1$ **do**
  for $\mathbf{e} \in \mathbb{F}_2^n$ *with* $\Delta(\mathbf{e}) \leq \frac{\gamma}{2} n$ **do**
    if $(\mathbf{x}_1 + \mathbf{e}, \mathbf{x}_2) \in L_2'$ **then** $L \leftarrow (\mathbf{x}_1, \mathbf{x}_2)$
  end
end
return L

Recall from Theorem 1 that L_1, L_2 contain random vectors from $\mathbb{F}_2^n$. Thus, for any pair $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$ we have $\Pr[\Delta(\mathbf{x}_1, \mathbf{x}_2) \leq \gamma n] = \binom{n}{\gamma n} \cdot 2^{-n}$. As a consequence, using a union bound over all pairs we can upper bound the size of the output list L for any NN algorithm by $|L| \leq \binom{n}{\gamma n} \cdot 2^{(2\lambda - 1)n}$.

This in turn shows that the running time of Alg. 4 is upper bounded by

$$\max\left\{ \binom{n}{\frac{\gamma}{2} n} \cdot 2^{\lambda n}, \binom{n}{\gamma n} \cdot 2^{(2\lambda - 1)n} \right\}. \tag{6}$$

Since our new decoding algorithm improves the decoding with high error rate, it is best suited for attacking instances of the Learning Parity with Noise Problem (LPN).

Definition 3 (LPN). *Let $\tau \in [0, \frac{1}{2})$ be some error parameter, and let $\mathbf{s} \in \mathbb{F}_2^k$ be a secret vector. In the $LPN_{k,\tau}$ problem one has oracle access to samples of*

the form

$$(\mathbf{a}_i, b_i) := (\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i), \quad \textit{for } i = 1, 2, \ldots$$

where $\mathbf{a}_i \in_R \mathbb{F}_2^k$ *and* $e_i \in \{0, 1\}$ *with* $\Pr[e_i = 1] = \tau$. *The goal is to recover* $\mathbf{s}$.

Let us denote by n the number of samples, which can be freely chosen. We write an LPN instance as a matrix-vector tuple

$$(A, \mathbf{b}) \in \mathbb{F}_2^{n \times k} \times \mathbb{F}_2^n \text{ satisfying } A\mathbf{s} = \mathbf{b} + \mathbf{e},$$

where $\mathbf{e} = (e_1, \ldots, e_n)$ and the i^{th} row of A and $\mathbf{b}$ represent the i^{th} LPN sample.

Notice that A is by definition of LPN the generator matrix of a random binary linear $[n, k]$-code, in which we are free to choose n ourselves. Thus, we can make the rate $\frac{k}{n}$ arbitrarily small.

Moreover, $\mathbf{b}$ is a noisy codeword that is decoded to $\mathbf{b} + \mathbf{e}$ with an error $\mathbf{e} \in \mathbb{F}_2^n$ of (large) expected weight $\mathbb{E}[\Delta(\mathbf{e})] = \tau n$. Typical parameters for τ in the cryptographic setting are $\tau = \frac{1}{4}$, or $\tau = \frac{1}{8}$.

3 The Depth-2 algorithm

Our Goal. As described in Section 2, many ISD algorithms like Dumer or BJMM do an exact matching using Eq. (3) $\bar{P}_1 \mathbf{e}_1^{(1)} = \bar{P}_1 \mathbf{e}_2^{(1)} + \mathbf{s}_1$ on ℓ coordinates, and among the candidates $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}) \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{k+\ell}$ that fulfill Eq. (3), they search for those, whose remaining $n-k-\ell$ coordinates approximately match by Eq. (4).

As opposed to the BJMM algorithm, we really go back to the initial Eq. (2) $\bar{P}\mathbf{e}' + \mathbf{e}'' = \bar{\mathbf{s}}$. Splitting $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ for $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)} \in \mathbb{F}_2^k$ yields

$$\bar{P}\mathbf{e}_1^{(1)} = \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}} \text{ on all } n-k \text{ but } \Delta(\mathbf{e}'') = \omega - p \text{ coordinates.} \tag{7}$$

Our goal is to directly construct $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}$ such that $\Delta(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}) = p$ and the corresponding vectors $\bar{P}\mathbf{e}_1^{(1)}, \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}}$ *approximately* match on all $n-k$ but $\omega - p$ coordinates. This immediately yields a solution $(\mathbf{e}', \mathbf{e}'')$ with $\mathbf{e}'' = \bar{P}\mathbf{e}' + \bar{\mathbf{s}}$ and $\Delta(\mathbf{e}'') = \omega - p$ for the Decoding problem in standard form.

In comparison to other ISD algorithms, our vectors $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}$ have length k (like in Prange) instead of $k+\ell$ (like in Dumer, BJMM). This decreases the search space significantly. Moreover, it introduces a less restrictive weight distribution on a solution $(\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$, since usually $p \ll \omega$ and we only need small weight p on the first k coordinates instead of the first $k + \ell$ coordinates. This in turn means that we need less iterations in Alg. 1 to find a permutation π that fulfills our weight distribution.

On the downside, our approximate matching routine is more costly than the exact matching in other ISD algorithms. But as our analysis shows, the benefits outweigh this disadvantage, especially when the weight of our solution is large enough.

Recall that by Eq. (7) our goal is to construct two lists $L_1^{(1)}, L_2^{(1)}$ in depth 1 of a search tree containing entries

$$(\mathbf{e}_1^{(1)}, \bar{P}\mathbf{e}_1^{(1)}) \text{ and } (\mathbf{e}_2^{(1)}, \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}}) \text{ such that}$$
$$\Delta(\mathbf{e}') = \Delta(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}) = p \text{ and } \Delta(\mathbf{e}'') = \omega - p.$$

The two lists $L_1^{(1)}, L_2^{(1)}$ are constructed in a recursive manner out of other lists in a search tree of some depth m that has to be optimized. In this section, we describe our algorithm for depth $m = 2$ only, since this already gives the main ideas.

Let us introduce some useful notion, see also Fig. 2. For any vector $\mathbf{v} = v_1 \dots v_n \in \mathbb{F}_2^n$ and any positive lengths $\ell_1, \dots, \ell_{m+1} \in \mathbb{N}$ with $\sum_{j=1}^{m+1} \ell_i = n$, we define by $\mathbf{v}_{[j]} \in \mathbb{F}_2^{\ell_j}$ the *projection* of $\mathbf{v}$ onto its coordinates $(\sum_{i=1}^{j-1} \ell_i + 1, \dots, \sum_{i=1}^{j} \ell_i)$. We also extend our notion to lists of vectors $L \subset \mathbb{F}_2^n$. In $L_{[j]}$ we project all elements $\mathbf{v} \in L$ to $\mathbf{v}_{[j]}$.

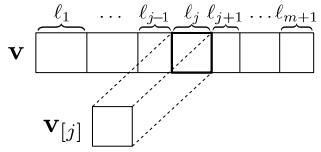

Fig. 2: The projection $\mathbf{v}_{[j]}$ of $\mathbf{v}$.

Outline of depth-2 algorithm. Here we give a high-level overview of our construction with a search tree of depth 2. The reader is advised to follow the description via Fig. 3.

Among the $n - k$ coordinates of $\mathbf{e}''$, we introduce another split into ℓ_1 and $\ell_2 := n - k - \ell_1$ coordinates. In the final list $L^{(2)}$, we enforce some weight ω_1 on the first ℓ_1 coordinates of $\mathbf{e}'' = (\mathbf{e}''_{[1]}, \mathbf{e}''_{[2]})$, and the remaining weight $\omega_2 := \omega - p - \omega_1$ on the remaining ℓ_2 coordinates. The parameters ℓ_1, ω_1 are subject to optimization.

For the construction of $\mathbf{e}''_{[2]}$ we use an NN search for the lists $L_1^{(1)}, L_2^{(1)}$ on level 1 that gives us weight ω_2 on these coordinates.

In those lists $L_1^{(1)}, L_2^{(1)}$ we furthermore enforce weight $p_1 \geq \frac{p}{2}$ on the coordinates of $\mathbf{e}_1^{(1)}$ and $\mathbf{e}_2^{(1)}$. The parameter p_1 is again subject to optimization.

Analogously we restrict to only those $\bar{P}\mathbf{e}_1^{(1)}$, $\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}}$ whose first ℓ_1 coordinates have a weight $\omega_1^{(1)}$. The weight $\omega_1^{(1)}$ has to be optimized. Again, we filter out all vector sums on level 2 whose weight is not exactly ω_1 on these ℓ_1 coordinates.

The lists $L_1^{(1)}, L_2^{(1)}$ are constructed out of four lists $L_i^{(0)}, i = 1, \dots, 4$ on level 0. Here, we describe only the construction of $L_1^{(1)}$, the construction of $L_2^{(1)}$

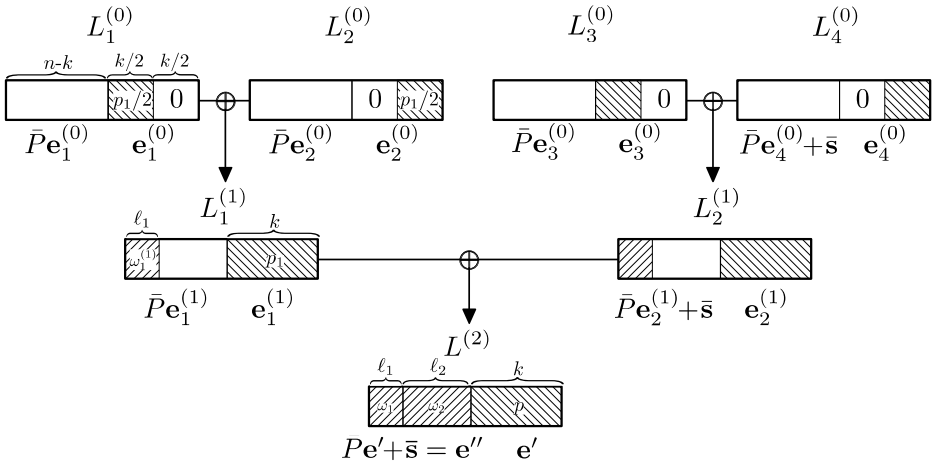

Fig. 3: Our depth-2 algorithm.

is analogous. In $L_1^{(0)}$ we enumerate all vectors $\mathbf{e}_1^{(0)} \in \mathbb{F}_2^{k/2} \times 0^{k/2}$ with weight $p_1/2$. For each of these vectors we compute $\bar{P}\mathbf{e}_1^{(0)}$. Similary, in $L_2^{(0)}$ we enumerate all vectors $\mathbf{e}_2^{(0)} \in 0^{k/2} \times \mathbb{F}_2^{k/2}$ with weight $p_1/2$ and compute $\bar{P}\mathbf{e}_2^{(0)}$. We then run a NN search on the first ℓ_1 coordinates to find all vector sums with weight $\omega_1^{(1)}$ on these coordinates. Note that the vectors $\mathbf{e}_1^{(0)}$ and $\mathbf{e}_2^{(0)}$ automatically add up to a vector of weight p_1 as required for list $L_1^{(1)}$.

This concludes the high-level description of our algorithm. More details can be found in Alg. 5, which has to be used as an ISDSolve-subroutine in Alg. 1 to obtain a full fletched ISD algorithm, including column permutation π and transformation to standard form.

List of objects. For completeness, we provide in the following a precise description of the lists. For the lists of level 0, we have

$$\begin{aligned}
L_1^{(0)} &= \{(\bar{P}\mathbf{e}_1^{(0)}, \mathbf{e}_1^{(0)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^{k/2} \times 0^{k/2} \mid \Delta(\mathbf{e}_1^{(0)}) = p_1/2\}, \qquad (8)\\
L_2^{(0)} &= \{(\bar{P}\mathbf{e}_2^{(0)}, (\mathbf{e}_2^{(0)}) \in \mathbb{F}_2^{n-k} \times 0^{k/2} \times \mathbb{F}_2^{k/2} \mid \Delta(\mathbf{e}_2^{(0)}) = p_1/2\},\\
L_3^{(0)} &= \{(\bar{P}\mathbf{e}_3^{(0)}, \mathbf{e}_3^{(0)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^{k/2} \times 0^{k/2} \mid \Delta(\mathbf{e}_3^{(0)}) = p_1/2\},\\
L_4^{(0)} &= \{(\bar{P}\mathbf{e}_4^{(0)} + \bar{\mathbf{s}}, \mathbf{e}_4^{(0)}) \in \mathbb{F}_2^{n-k} \times 0^{k/2} \times \mathbb{F}_2^{k/2} \mid \Delta(\mathbf{e}_4^{(0)}) = p_1/2\}.
\end{aligned}$$

Thus, all lists on level 0 have size $S_0 = \binom{k/2}{p_1/2}$. Note that $L_1^{(0)} = L_3^{(0)}$. The lists on level 1 are constructed via NN search on the first ℓ_1 coordinates such that we obtain weight $\omega_1^{(1)}$ on these coordinates. This yields

$$\begin{aligned}
L_1^{(1)} &= \{(\bar{P}\mathbf{e}_1^{(1)}, \mathbf{e}_1^{(1)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^{k} \mid \Delta(\mathbf{e}_1^{(1)}) = p_1, \Delta((\bar{P}\mathbf{e}_1^{(1)})_{[1]}) = \omega_1^{(1)}\},\\
L_2^{(1)} &= \{(\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}}, \mathbf{e}_2^{(1)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^{k} \mid \Delta(\mathbf{e}_2^{(1)}) = p_1, \Delta((\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]}) = \omega_1^{(1)}\}.
\end{aligned}$$

By the randomness of $\bar{P}$, both lists have expected size

$$S_1 := \mathbb{E}[|L_i^{(1)}|] = S_0^2 \cdot \Pr[\text{weight } \omega_1^{(1)} \text{ on the first } \ell_1 \text{ coordinates}]$$
$$= \binom{k/2}{p_1/2}^2 \cdot \frac{\binom{\ell_1}{\omega_1^{(1)}}}{2^{\ell_1}} \quad \text{for } i = 1, 2.$$

Eventually, by an NN search on ℓ_2 bits for weight ω_2 on the level-1 lists and subsequent filtering for weight p on the last k coordinates and weight ω_1 on the first ℓ_1 bits, we obtain

$$L^{(2)} = \{(\mathbf{e}'', \mathbf{e}') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k} \mid \Delta(\mathbf{e}') = p, \mathbf{e}'' = \bar{P}\mathbf{e}' + \bar{\mathbf{s}}, \Delta(\mathbf{e}'') = \omega - p\}.$$

Thus, any element $(\mathbf{e}'', \mathbf{e}')$ of $L^{(2)}$ yields a solution $(\mathbf{e}', \mathbf{e}'')$ of a Syndrome Decoding Problem in standard form.

Algorithm 5: DEPTH-2-ISDSOLVE

Input : $\bar{P} \in \mathbb{F}_2^{(n-k)\times k}, \bar{\mathbf{s}} \in \mathbb{F}_2^{n-k}, \omega$
Output : $(\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$
Parameters: Optimize $p, \omega_1, \ell_1, p_1, \omega_1^{(1)}$.
Set $\omega_2 = \omega - p - \omega_1$ and $\ell_2 = n - k - \ell_1$.

1 Create lists $L_i^{(0)}$, $i = 1, 2, 3, 4$ as defined in (8)
2 $L_i^{(1)} \leftarrow$ NN-Search$(L_{2i-1}^{(0)}, L_{2i}^{(0)}, 1, \omega_1^{(1)}), i = 1, 2$
▷ NN-Search(L_1, L_2, i, w) performs a NN search on $(L_1)_{[i]}, (L_2)_{[i]}$
▷ with target weight w while keeping all other coordinates.
3 $L^{(2)} \leftarrow$ NN-Search$(L_1^{(1)}, L_2^{(1)}, 2, \omega_2)$
4 $L^{(2)} \leftarrow$ Filter$(L^{(2)}, 1, \omega_1))$ ▷ Filter(L, i, w) filters L for elements
$L^{(2)} \leftarrow$ Filter$(L^{(2)}, 3, p))$ ▷ with weight w on its projection in $L_{[i]}$.
if $|L^{(2)}| > 0$ **then return** $(\mathbf{e}', \mathbf{e}'')$ for some $(\mathbf{e}'', \mathbf{e}') \in L^{(2)}$
else return $\perp$

Notice that Alg. 5 can only succeed to output a solution $(\mathbf{e}', \mathbf{e}'') \neq \perp$ if there exists some $\mathbf{e}'$ with weight p such that $\bar{P}\mathbf{e}' + \bar{\mathbf{s}} = \mathbf{e}'' = (\mathbf{e}''_{[1]}, \mathbf{e}''_{[2]})$ with $\mathbf{e}''_{[1]}, \mathbf{e}''_{[2]}$ having weights ω_1 and ω_2, respectively. This specific weight distribution has to be induced by the column permutation π of Alg. 1.

Definition 4. *Let $\mathbf{e} \in \mathbb{F}_2^n$ with $\Delta(\mathbf{e}) = \omega$ and $k, p \in \mathbb{N}$. Let $\ell_1, \ell_2 \in \mathbb{N}$ with $\ell_1 + \ell_2 = n - k$, and let $\omega_1, \omega_2 \in \mathbb{N}$ with $\omega_1 + \omega_2 = \omega - p$. We call a permutation π* good *for $\mathbf{e}$ with respect to $(p, \omega_1, \ell_1, \omega_2, \ell_2)$, if $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}''_{[1]}, \mathbf{e}''_{[2]}) \in \mathbb{F}_2^k \times \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell_2}$ with*

$$\Delta(\mathbf{e}') = p, \quad \Delta(\mathbf{e}''_{[1]}) = \omega_1 \text{ and } \Delta(\mathbf{e}''_{[2]}) = \omega_2.$$

A random permutation π is good with probability

$$P_\pi = \frac{\binom{k}{p}\binom{\ell_1}{\omega_1}\binom{\ell_2}{\omega_2}}{\binom{n}{\omega}}.$$

It remains to show that on input a standard form Syndrome Decoding instance $(\bar{P}, \bar{\mathbf{s}}, \omega)$ that stems from a good π, Alg. 5 constructs a non-empty list of solutions $L^{(2)}$.

Lemma 1 (Correctness). *Let* $\mathbf{e}$ *be a solution to the Syndrome Decoding Problem. Let* π *be good for* $\mathbf{e}$ *with respect to any fixed parameters* $(p, \omega_1, \ell_1, \omega_2, \ell_2)$ *as given by Definition 4. Whenever we run Alg. 5 with parameters* $p_1, \omega_1^{(1)} \in \mathbb{N}$ *satisfying*

$$\binom{p}{p/2}\binom{k-p}{p_1-p/2} \geq \frac{2^{\ell_1}}{\binom{\omega_1}{\omega_1/2}\binom{\ell_1-\omega_1}{\omega_1^{(1)}-\omega_1/2}}, \tag{9}$$

then on expectation we have $(\mathbf{e}'', \mathbf{e}') \in L^{(2)}$ *for* $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}'')$.

Thus, Lemma 1 shows that any (possibly unique) solution $\mathbf{e}$ to the Syndrom Decoding Problem is constructed in our sub-routine of Alg. 5 at that point in time when the full-fletched ISD Alg. 1 provides a good permutation π, under the condition that (9) holds.

Before we prove Lemma 1, we would like to show that its statement is not vacuous. Namely, there always exist $p_1, w_1^{(1)}$ such that condition (9) holds.

Using the Binomial Theorem, we have

$$\binom{n}{n/2} < \sum_{i=0}^{n}\binom{n}{i} = 2^n < (n+1)\binom{n}{n/2}.$$

This implies

$$\frac{2^n}{n+1} < \binom{n}{n/2} < 2^n.$$

Thus, up to a linear factor we can approximate $\binom{n}{n/2}$ by 2^n. Hence if we ignore linear factors, condition (9) collapses for the setting $p_1 = k/2$ and $w_1^{(1)} = \ell_1/2$ to

$$2^{p+k-p} \geq 2^{\ell_1-\omega_1-(\ell_1-\omega_1)} \quad \Leftrightarrow \quad k \geq 0,$$

which is trivially fulfilled. Thus, there always exist feasible parameters $p, \omega_1, \ell_1, p_1, \omega_1^{(1)}$ that lead to a solution when running Alg. 5. Among these feasible parameters, we will later minimize running time.

Proof (of Lemma 1). Let $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}'')$ be the solution of our Syndrome Decoding problem in standard form. Since we have standard form, we conclude that $\mathbf{e}'' = \bar{P}\mathbf{e}' + \bar{\mathbf{s}}$ is fully determined by $\mathbf{e}'$. Moreover, since we assume π to be good, $\mathbf{e}''$ is of the correct form. Thus, it suffices to show that Alg. 5 constructs the desired $\mathbf{e}' \in \mathbb{F}_2^k$.

Notice that in our construction $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$, and in turn $\mathbf{e}_1^{(1)} = \mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)}$ (and analogous for $\mathbf{e}_2^{(1)}$).

Let us first argue that in our construction we obtain up to a polynomial factor all $\binom{k}{p_1}$ vectors $\mathbf{e}_1^{(1)} \in \mathbb{F}_2^k$ on level 1. All vector sums $\mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)}$ are by

the definition of $\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}$ different. Up to polynomial factors (denoted by $\approx$), we have by standard approximation via the binary entropy function $\binom{k/2}{p_1/2}^2 \approx 2^{2H(\frac{p_1}{k})k/2} \approx \binom{k}{p_1}$ vectors $\mathbf{e}_1^{(1)}$ that we construct.

Now let us turn to the construction of $\mathbf{e}'$ with weight p on level 2 via $\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ with $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}$ having weight $p_1 \geq p/2$. We call $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ a representation of $\mathbf{e}'$ if $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$. Notice that our desired solution $\mathbf{e}'$ has

$$R_2 := \binom{p}{p/2}\binom{k-p}{p_1-p/2} \text{ representations,} \tag{10}$$

since the set of 1-coordinates in $\mathbf{e}'$ can be represented in $\binom{p}{p/2}$ ways as $1+0$ or $0+1$, and the set of 0-coordinates in $\mathbf{e}'$ can be represented in $\binom{k-p}{p_1-p/2}$ ways as $0+0$ or $1+1$.

From an algorithmic point of view, we do not care which of the R_2 representations is eventually used for constructing $\mathbf{e}'$. It is therefore sufficient that only 1 of these R_2 representations is present in $L_1^{(1)} \times L_2^{(1)}$. Hence, for achieving minimal run time we construct only a random $1/R_2$-fraction of all representations such that on expectation one representation is present in $L_1^{(1)} \times L_2^{(1)}$, and therefore $\mathbf{e}'$ appears in $L^{(2)}$.

For constructing only an $1/R_2$-fraction, we construct on level 1 only those elements $(\bar{P}\mathbf{e}_1^{(1)}, \mathbf{e}_1^{(1)}) \in L_1^{(1)}$ whose first ℓ_1 coordinates have weight $\omega_1^{(1)}$ (analogous for $L_2^{(1)}$). This means that we enforce $\Delta((\bar{P}\mathbf{e}_1^{(1)})_{[1]})) = \omega_1^{(1)}$. Let E be the event that there exists a representation of

$$\mathbf{e}''_{[1]} = (\bar{P}\mathbf{e}_1^{(1)} + \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]} \text{ with } \Delta((\bar{P}\mathbf{e}_1^{(1)})_{[1]}) = \Delta((\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]}) = \omega_1^{(1)}.$$

By randomness of $\bar{P}$, we have

$$p_{2,2} := \Pr[E] = \frac{\binom{\omega_1}{\omega_1/2}\binom{\ell_1-\omega_1}{\omega_1^{(1)}-\omega_1/2}}{2^{\ell_1}},$$

since there are a total of 2^{ℓ_1} possible representations of the form $\mathbf{e}_{[1]} = (\bar{P}\mathbf{e}_1^{(1)} + \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]}$ out of which $\binom{\omega_1}{\omega_1/2}\binom{\ell_1-\omega_1}{\omega_1^{(1)}-\omega_1/2}$ have the correct weight $\omega_1^{(1)}$ for $(\bar{P}\mathbf{e}_1^{(1)})_{[1]}$, $(\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]}$ by the same argument as in Eq. (10).

Thus, the expected number of representations of $\mathbf{e}'$ is $R_2 \cdot p_{2,2}$. Hence on expectation, we construct $\mathbf{e}'$ in $L^{(2)}$ if $R_2 \cdot p_{2,2} \geq 1$, which is equivalent to condition (9). □

Complexity of the Depth-2 Algorithm. Our Alg. 5 starts with the construction of lists $L_i^{(0)}, i = 1, 2, 3, 4$ (step 1) which takes time $S_0 = \binom{k/2}{p_1/2}$. By Theorem 1 and Eq. (5), (6), the Nearest Neighbor search on $(L_i^{(0)})_{[1]}$ (step 2)

takes time

$$T_0 := \begin{cases} 2^{y(\frac{\log(S_0)}{\ell_1}, \frac{\omega_1^{(1)}}{\ell_1})\ell_1} & \text{if } \frac{\log(S_0)}{\ell_1} < 1 - H(\frac{\omega_1^{(1)}}{2\ell_1}) \\ \min\{S_0^2, \max\{\binom{\ell_1}{\frac{\omega_1^{(1)}}{2}} \cdot S_0, S_0^2 \cdot \frac{\binom{\ell_1}{\omega_1^{(1)}}}{2^{\ell_1}}\}\} & \text{else} \end{cases}.$$

resulting in the lists $L_1^{(1)}$, $L_2^{(1)}$ that have expected size $S_1 = \binom{k/2}{p_1/2}^2 \cdot \frac{\binom{\ell_1}{\omega_1^{(1)}}}{2^{\ell_1}}$. These lists are again combined via Nearest Neighbor search (step 3) in time

$$T_1 := \begin{cases} 2^{y(\frac{\log(S_1)}{\ell_2}, \frac{\omega_2}{\ell_2})\ell_2} & \text{if } \frac{\log(S_1)}{\ell_2} < 1 - H(\frac{\omega_2}{2\ell_2}) \\ \min\{S_1^2, \max\{\binom{\ell_2}{\frac{\omega_2}{2}} \cdot S_1, S_1^2 \cdot \frac{\binom{\ell_2}{\omega_2}}{2^{\ell_2}}\}\} & \text{else} \end{cases},$$

resulting in the final list $L^{(2)}$. The filtering (step 4) takes time $S_2 := |L^{(2)}|$. We only need to store the lists $L_i^{(0)}$ of size S_0 as well as the lists $L_1^{(1)}$, $L_2^{(1)}$ of size S_1. The total running time is

$$\text{time } T = \max\{T_0, T_1, S_0, S_2\} = \max\{T_0, T_1\},$$

since $T_0 \geq S_0$ and $T_1 \geq S_2$.

Total complexity of Decoding. Alg. 5 constructs a solution iff π is good which happens according to Definition 4 with probability P_π, resulting in a total expected running time of $T \cdot P_\pi^{-1}$ for our full-fletched ISD algorithm.

In the following Theorem 2 we provide an upper bound for T for the important setting of Full Distance (FD) Decoding, where we correct up to the distance many errors, i.e. $\omega = d$. This serves us a first benchmark for our algorithm.

In depth 2, we achieve $T \leq 2^{0.0982n}$ which already improves over the bound $2^{0.102n}$ of the original BJMM-algorithm [].

Theorem 2. *Alg. 1 in combination with Alg. 5 solves Full Distance decoding for random binary linear codes in expected time* $2^{0.0982n}$ *using* $2^{0.0717n}$ *space.*

Proof. We achieve the worst-case running time at a rate of

$$\frac{k}{n} = 0.43 \text{ with relative distance } \frac{\omega}{n} = \frac{d}{n} = H^{-1}\left(1 - \frac{k}{n}\right) = 0.1346.$$

For this rate, we minimize the running time by choosing the relative weights

$$\frac{p}{n} = 0.03730, \quad \frac{p_1}{n} = 0.02645,$$

resulting in

$$R_2 = 2^{0.09254n}$$

representations. Furthermore, we optimize

$$\frac{\ell_1}{n} = 0.1553 \text{ and } \frac{\omega_1}{n} = 0.01970.$$

Using condition Eq.(9) from Lemma 1, we have

$$\frac{\omega_1^{(1)}}{n} = 0.01765.$$

The resulting list sizes are

$$S_0 = 2^{0.07168n}, \quad S_1 = 2^{0.06741n}.$$

The running times are

$$T_0 = 2^{0.08483n}, \quad T_1 = 2^{0.08482n},$$

using May-Ozerov NN search. The probability for the correct weight distribution satisfying Def. 4 is

$$P_\pi = 2^{-0.01334n}.$$

Thus the overall running time and space consumption are

$$T = 2^{0.0982n} \text{ and } S = 2^{0.0717n}. \quad \square$$

4 The Depth-m algorithm

Our algorithm with depth 2, as described in the previous Section 3, already illustrates the overall idea of approximate matching, but does not yet yield improved running times compared to the BJMM algorithm. Therefore, we generalize to arbitrary depth in this section, which is mostly straight-forward but still includes some subtleties how to proceed with approximate matchings - and their respective weights - over many levels of a search tree.

Outline of depth-m algorithm. Let us start again with a high-level overview for our algorithm with arbitrary depth m. The reader is advised to follow the description via Fig. 4 which shows the algorithm for $m = 3$.

In the final list $L_1^{(m)}$ we now split the first $n-k$ coordinates into m blocks instead of only 2 blocks, i.e we have $\mathbf{e}'' = (\mathbf{e}''_{[1]}, \ldots, \mathbf{e}''_{[m]})$. Block $\mathbf{e}''_{[i]}$ has length ℓ_i and weight $\omega_i^{(m)}$. The parameters ℓ_i, $\omega_i^{(m)}$ are subject to optimization.

On level 0, there are a total of 2^m lists $L_1^{(0)}, \ldots, L_{2^m}^{(0)}$. The construction of the 2^{m-1} lists $L_1^{(1)}, \ldots, L_{2^{m-1}}^{(1)}$ on level 1 out of the level-0 lists is identical to the construction in Section 3.

For the level-m lists, we have

$$\begin{aligned}
L_{j_1}^{(0)} &= \{(\bar{P}\mathbf{e}_{j_1}^{(0)}, \mathbf{e}_{j_1}^{(0)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^{k/2} \times 0^{k/2} \mid \Delta(\mathbf{e}_{j_1}^{(0)}) = p_1/2\}, \qquad (11)\\
L_{j_2}^{(0)} &= \{(\bar{P}\mathbf{e}_{j_2}^{(0)}, \mathbf{e}_{j_2}^{(0)}) \in \mathbb{F}_2^{n-k} \times 0^{k/2} \times \mathbb{F}_2^{k/2} \mid \Delta(\mathbf{e}_{j_2}^{(0)}) = p_1/2\},\\
L_{2^m}^{(0)} &= \{(\bar{P}\mathbf{e}_{2^m}^{(0)} + \bar{\mathbf{s}}, \mathbf{e}_{2^m}^{(0)}) \in \mathbb{F}_2^{n-k} \times 0^{k/2} \times \mathbb{F}_2^{k/2} \mid \Delta(\mathbf{e}_{2^m}^{(0)}) = p_1/2\}.
\end{aligned}$$

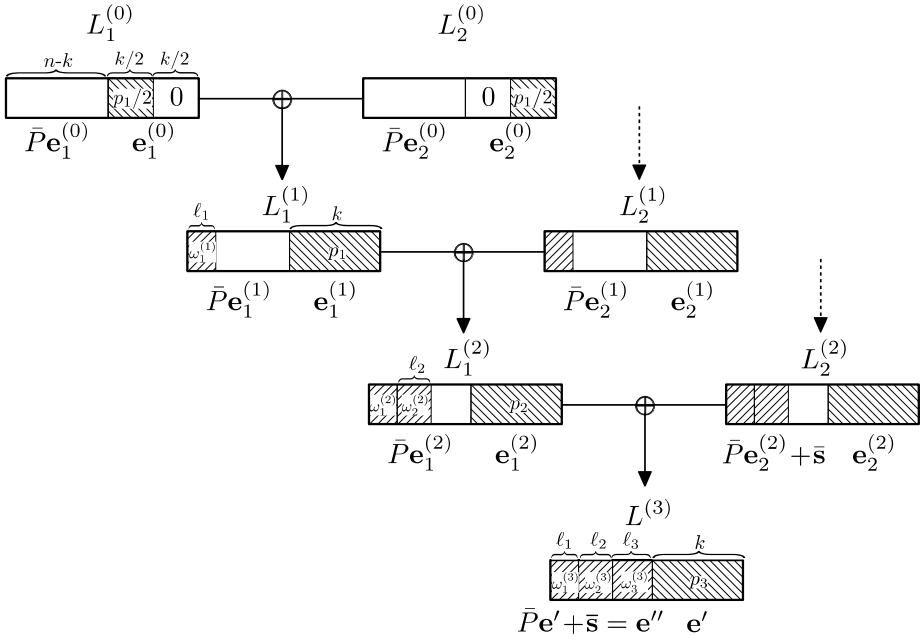

Fig. 4: Our algorithm for depth 3.

for $j_1 = 1, 3, \ldots, 2^m - 1$ and $j_2 = 2, 4, \ldots, 2^m - 2$. All lists on level 0 therefore have size $S_0 = \binom{k/2}{p_1/2}$.

Starting from the level-0 lists, our algorithm combines two lists at a time using NN search in a binary tree wise fashion until we reach the final list $L^{(m)}$. On every level $i = 1, \ldots, m-1$ we construct the first list $L_1^{(i)}$ via NN search on the projected lists $(L_1^{(i-1)})_{[i]}, (L_2^{(i-1)})_{[i]}$ such that we obtain weight $\omega_i^{(i)}$. We furthermore filter for weight p_i on the last k coordinates and a specific weight distribution on the remaining coordinates such that we get

$$L_1^{(i)} = \{(\bar{P}\mathbf{e}_1^{(i)}, \mathbf{e}_1^{(i)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^k \mid \Delta(\mathbf{e}_1^{(i)}) = p_i \,,\, \Delta((\bar{P}\mathbf{e}_1^{(i)})_{[h]}) = \omega_h^{(i)}, h = 1, \ldots, i\}.$$

The other lists $L_j^{(i)}$, $j = 2, \ldots, 2^i$ are created analogously. By randomness of $\bar{P}$ the projection $(\bar{P}\mathbf{e}_1^{(i)})_{[i]}$ with weight $\omega_i^{(i)}$ is in our construction the sum of two random vectors. For every $h = 1, \ldots, i-1$ the projection $(\bar{P}\mathbf{e}_1^{(i)})_{[h]}$ with weight $\omega_h^{(i)}$ is the sum of two random vectors of specific weight $\omega_h^{(i-1)}$. Fixing the first vector, there are $\binom{\ell_h}{\omega_h^{(i-1)}}$ possible second vectors out of which $\binom{\omega_h^{(i-1)}}{\omega_h^{(i-1)} - \omega_h^{(i)}/2}\binom{\ell_h - \omega_h^{(i-1)}}{\omega_h^{(i)}/2}$ yield the correct weight $\omega_h^{(i)}$. Therefore, the expected

list sizes on layer i are upper bounded by

$$\begin{aligned} S_i &\leq |\{\mathbf{x} \in \mathbb{F}_2^k | \Delta(\mathbf{x}) = p_i\}| \cdot \Pr_{\mathbf{x} \in \mathbb{F}_2^{\ell_i}}[\Delta(\mathbf{x}) = \omega_i^{(i)}] \\ &\cdot \prod_{h=i+1}^{m-1} \Pr_{\mathbf{x},\mathbf{y} \in \mathbb{F}_2^{\ell_h}}[\Delta(\mathbf{x}+\mathbf{y}) = \omega_h^{(i)} | \Delta(\mathbf{x}) = \Delta(\mathbf{y}) = \omega_h^{(i+1)}] \\ &= \binom{k}{p_i} \cdot \frac{\binom{\ell_i}{\omega_i^{(i)}}}{2^{\ell_i}} \cdot \prod_{h=1}^{i-1} \frac{\binom{\omega_h^{(i-1)}}{\omega_h^{(i-1)} - \omega_h^{(i)}/2} \binom{\ell_h - \omega_h^{(i-1)}}{\omega_h^{(i)}/2}}{\binom{\ell_h}{\omega_h^{(i-1)}}}. \end{aligned}$$

Eventually, an NN search on $\ell_m = n - k - \sum_{i=1}^{m-1} \ell_i$ bits for weight $\omega_m^{(m)} = \omega - p_m - \sum_{i=1}^{m-1} \omega_i^{(m)}$, $p_m = p$, on the level-$(m-1)$ lists and subsequent filtering for weight p_m on the last k coordinates and weight $\omega_i^{(m)}$ for every projection $\mathbf{e}''_{[i]}$, $i = 1, \ldots, m-1$, we obtain

$$L^{(m)} = \{(\mathbf{e}'', \mathbf{e}') \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^k \mid \Delta(\mathbf{e}') = p_m, \mathbf{e}'' = \bar{P}\mathbf{e}' + \bar{\mathbf{s}}, \Delta(\mathbf{e}'') = \omega - p_m\}.$$

Thus, any element $(\mathbf{e}'', \mathbf{e}')$ of $L^{(m)}$ yields a solution $(\mathbf{e}', \mathbf{e}'')$ of a Syndrome Decoding Problem in standard form.

More details can be found in Alg. 6, which has to be used again as an ISD-Solve-subroutine in Alg. 1 to obtain a full fletched ISD algorithm.

Algorithm 6: Depth-m-ISDSolve

Input : $\bar{P} \in \mathbb{F}_2^{(n-k)\times k}, \bar{\mathbf{s}} \in \mathbb{F}_2^{n-k}, \omega$
Output : $(\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$
Parameters: Optimize $p_1, \ldots, p_m, \omega_1^{(m)}, \ldots, \omega_{m-1}^{(m)}, \ell_1, \ldots, \ell_{m-1}$.
Compute $\omega_m^{(m)} = \omega - p_m - \sum_{i=1}^{m-1} \omega_i^{(m)}$, $\ell_m = n - k - \sum_{i=1}^{m-1} \ell_i$.

Define $\omega_i^{(i)} := \frac{\omega_i^{(i+1)}}{2}$, $i = 1, \ldots, m-2$.
Choose optimal $\omega_j^{(i)}$ such that condition (12) holds.
Create lists $L_j^{(0)}$, $j = 1, \ldots, 2^m$ as defined in (11).
$L_j^{(1)} \leftarrow$ NN-Search$((L_{2j-1}^{(0)})_{[1]}, (L_{2j}^{(0)})_{[1]}, \omega_1^{(1)})$, $j = 1, \ldots, 2^{m-1}$
for $i = 2, \ldots, m$, $j = 1, \ldots, 2^i$ **do**
  $L_j^{(i)} \leftarrow$ NN-Search$((L_{2j-1}^{(i-1)})_{[i]}, (L_{2j}^{(i-1)})_{[i]}, \omega_i^{(i)})$
  $L_j^{(i)} \leftarrow$ Filter$(L_j^{(i)}, h, \omega_h^{(i)}))$, $h = 1, \ldots, i-1$
  $L_j^{(i)} \leftarrow$ Filter$(L_j^{(i)}, m+1, p_i))$
end
if $|L^{(m)}| > 0$ **then return** $(\mathbf{e}', \mathbf{e}'')$ for some $(\mathbf{e}'', \mathbf{e}') \in L^{(m)}$
else return $\perp$

Notice that Alg. 6 can only succeed to output a solution $(\mathbf{e}', \mathbf{e}'') \neq \perp$ if there exists some $\mathbf{e}'$ with weight p_m such that $\bar{P}\mathbf{e}' + \bar{\mathbf{s}} = (\mathbf{e}''_{[1]}, \ldots, \mathbf{e}''_{[m]})$ with $\mathbf{e}''_{[i]}$

having weight $\omega_i^{(m)}$ for all $i = 1, \ldots, m$. This specific weight distribution has to be induced by the column permutation π of Alg. 1.

Definition 5. *Let* $\mathbf{e} \in \mathbb{F}_2^n$ *with* $\Delta(\mathbf{e}) = \omega$ *and* $k, p_m \in \mathbb{N}$. *Let* $\ell_1, \ldots, \ell_m \in \mathbb{N}$ *with* $\sum_{i=1}^m \ell_i = n - k$, *and* $\omega_1^{(m)}, \ldots, \omega_m^{(m)} \in \mathbb{N}$ *with* $\sum_{i=1}^m \omega_i^{(m)} = \omega - p_m$.

We call a permutation π good *for* $\mathbf{e}$ *with respect to* $p_m, (\omega_i^{(m)}, \ell_i)_{i=1,\ldots,m}$, *if* $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}''_{[1]}, \ldots, \mathbf{e}''_{[m]}) \in \mathbb{F}_2^k \times \mathbb{F}_2^{\ell_1} \times \cdots \times \mathbb{F}_2^{\ell_m}$ *with*

$$\Delta(\mathbf{e}') = p_m, \quad \Delta(\mathbf{e}''_{[i]}) = \omega_i^{(m)}, i = 1, \ldots, m.$$

A random permutation π *is good with probability*

$$P_\pi = \frac{\binom{k}{p_m} \prod_{i=1}^m \binom{\ell_i}{\omega_i^{(m)}}}{\binom{n}{\omega}}.$$

We now show that on input a standard form Syndrome Decoding instance $(\bar{P}, \bar{\mathbf{s}}, \omega)$ that stems from a good π, Alg. 6 constructs a non-empty list of solutions $L^{(m)}$.

Lemma 2 (Correctness). *Let* $\mathbf{e}$ *be a solution to the Syndrome Decoding Problem. Let* π *be good for* $\mathbf{e}$ *with respect to any fixed parameters* $p_m, \omega_i^{(m)}, \ell_i$, $i = 1, \ldots, m$ *as given by Definition 5. Whenever we run Alg. 6 with parameters* $p_i, \omega_j^{(i)} \in \mathbb{N}$, *for* $j = 1, \ldots, i$, $i = 1, \ldots, m-1$ *satisfying*

$$\binom{p_i}{p_i/2}\binom{k-p_i}{p_{i-1}-p_i/2} \geq \prod_{h=1}^{i-1} \frac{2^{\ell_h}}{\binom{\omega_h^{(i)}}{\omega_h^{(i)}/2}\binom{\ell_h-\omega_h^{(i)}}{\omega_h^{(i-1)}-\omega_h^{(i)}/2}}, \quad \forall i = 2, \ldots, m \tag{12}$$

then on expectation we have $(\mathbf{e}'', \mathbf{e}') \in L^{(m)}$ *for* $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}'')$.

Analogous to Section 3, we can show that the setting $p_i = k/2$ and $w_h^{(i)} = \ell_h/2$, for $h = 1, \ldots, i-1$, $i = 2, \ldots, m$, yields feasible parameters for Alg. 6 that fulfill condition (12).

Proof (of Lemma 2). Let $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}'')$ be the solution of our Syndrome Decoding problem in standard form. Similar to the reasoning in the proof of Lemma 1, it suffices to show that Alg. 6 constructs the desired $\mathbf{e}' \in \mathbb{F}_2^k$.

Notice that in our construction $\mathbf{e}' = \mathbf{e}_1^{(m-1)} + \mathbf{e}_2^{(m-1)}$, and in turn $\mathbf{e}_j^{(i)} = \mathbf{e}_{2j-1}^{(i-1)} + \mathbf{e}_{2j}^{(i-1)}$, for all $j = 1, \ldots, 2^i$, $i = 1 \ldots, m-1$.

Similar to the reasoning in the proof of Lemma 1, we obtain up to a polynomial factor all $\binom{k}{p_1}$ vectors $\mathbf{e}_j^{(1)} \in \mathbb{F}_2^k$ on level 1.

We now look at the construction of $\mathbf{e}_1^{(i)}$ with weight p_i on level i via $\mathbf{e}_1^{(i-1)} + \mathbf{e}_2^{(i-1)}$ with $\mathbf{e}_1^{(i-1)}, \mathbf{e}_2^{(i-1)}$ having weight $p_{i-1} \geq p_i/2$. This goes analogously for

all vectors on this layer and all layers $i = 2, \ldots, m-1$ as well as the final vector $\mathbf{e}'$ on level m. Notice that our desired vector $\mathbf{e}_1^{(i)}$ has

$$R_i := \binom{p_i}{p_i/2}\binom{k-p_i}{p_{i-1}-p_i/2} \text{ representations.} \tag{13}$$

It is sufficient that one of these representations is present in $L_1^{(i-1)} \times L_2^{(i-1)}$.

For constructing only a random $1/R_i$-fraction of all representations, we compute on level $i-1$ only those elements $(\mathbf{e}_1^{(i-1)}, \bar{P}\mathbf{e}_1^{(i-1)}) \in L_1^{(i-1)})$ satisfying $\Delta((\bar{P}\mathbf{e}_1^{(i-1)})_{[h]})) = \omega_h^{(i-1)}$, $h = 1, \ldots, i-1$ (analogous for $L_2^{(i-1)}$). Let E be the event that there exists a representation of

$$(\bar{P}\mathbf{e}_1^{(i)})_{[h]} = (\bar{P}\mathbf{e}_1^{(i-1)} + \bar{P}\mathbf{e}_2^{(i-1)})_{[h]} \text{ with } \Delta((\bar{P}\mathbf{e}_1^{(i-1)})_{[h]}) = \Delta((\bar{P}\mathbf{e}_2^{(i-1)})_{[h]}) = \omega_h^{(i-1)}.$$

for $h = 1, \ldots, i-1$. By randomness of $\bar{P}$, we have

$$p_{i,m} := \Pr[E] = \prod_{h=1}^{i-1} \frac{\binom{\omega_h^{(i)}}{\omega_h^{(i)}/2}\binom{\ell_h - \omega_h^{(i)}}{\omega_h^{(i-1)} - \omega_h^{(i)}/2}}{2^{\ell_h}},$$

since there are a total of 2^{ℓ_h} possible representations of the form $(\bar{P}\mathbf{e}_1^{(i)})_{[h]} = (\bar{P}\mathbf{e}_1^{(i-1)} + \bar{P}\mathbf{e}_2^{(i-1)})_{[h]}$ out of which $\binom{\omega_h^{(i)}}{\omega_h^{(i)}/2}\binom{\ell_h - \omega_h^{(i)}}{\omega_h^{(i-1)} - \omega_h^{(i)}/2}$ have the correct weight $\omega_h^{(i-1)}$ for $(\bar{P}\mathbf{e}_1^{(i-1)})_{[h]}$, $(\bar{P}\mathbf{e}_2^{(i-1)})_{[h]}$ by the same argument as in Eq. (13).

Thus, the expected number of representations of $\mathbf{e}_1^{(i)}$ is $R_i \cdot p_{i,m}$. Hence on expectation, we construct $\mathbf{e}_1^{(i)}$ in $L_1^{(i)}$ if $R_i \cdot p_{i,m} \geq 1$. Generalizing this condition to all layers yields condition (12). □

Complexity of Alg. 6. The lists $L_j^{(0)}$, $j = 1, \ldots, 2^m$ are created in time S_0 (step 2). The NN search on those lists yields lists $L_j^{(1)}$, $j = 1, \ldots, 2^{m-1}$ (step 3).

Next, another NN search on the new lists returns lists $L_j^{(2)}$, $j = 1, \ldots, 2^{m-2}$ (step 4) which are subsequently filtered (step 5). These two steps of NN search and filtering are repeated until only one list is left. By Theorem 1 and Eq. (5), (6) the NN search layer on $i = 0, \ldots, m-1$ takes time

$$T_i := \begin{cases} 2^{y(\frac{\log(S_i)}{\ell_{i+1}}, \frac{\omega_{i+1}^{(i+1)}}{\ell_{i+1}})\ell_{i+1}} & \text{if } \frac{\log(S_i)}{\ell_{i+1}} < 1 - H(\frac{\omega_{i+1}^{(i+1)}}{2\ell_{i+1}}) \\ \min\{S_i^2, \max\{\binom{\ell_{i+1}}{\frac{\omega_{i+1}^{(i+1)}}{2}} \cdot S_i, S_i^2 \cdot \frac{\binom{\ell_{i+1}}{\omega_{i+1}^{(i+1)}}}{2^{\ell_{i+1}}}\}\} & \text{else} \end{cases}.$$

The filtering takes time S_i on layer $i = 2, \ldots, m-1$ and $S_m := |L^{(m)}|$ on layer m.

On every level i of our search tree we consume time T_i and store lists of size S_i. Thus, we obtain

$$\text{time } T = \max_{i=1,\ldots,m}\{T_i\}$$

using $T_i \geq S_i$ for $i = 0, \ldots, m-1$ and $T_{m-1} \geq S_m$.

Total complexity of Decoding. Alg. 6 constructs a solution iff π is good which happens according to Definition 5 with probability P_π, resulting in a total expected running time of $T \cdot P_\pi^{-1}$ for our full-fletched ISD algorithm.

5 Results

Syndrome Decoding Problem. The best known complexity for Full Distance decoding is currently $2^{0.0953n}$ using BJMM in depth 4 [BM17b], whereas for Half Distance Decoding the best known bound is $2^{0.0473n}$ [MO15].

As stated in Theorem 3, we improve the bound for Full Distance Decoding to $2^{0.885n}$. In the Half Distance Decoding setting, we achieve a small improvement to $2^{0.0465n}$.

Theorem 3. *Alg. 1 in combination with Alg. 6 for $m = 4$ solves Full Distance decoding for random binary linear codes in expected time $2^{0.0885n}$ using $2^{0.0736n}$ space. Half Distance decoding is solved in exptected time $2^{0.0465n}$ using $2^{0.0294n}$ space.*

Proof. For Full Distance Decoding we achieve the maximal running time at code rate

$$\frac{k}{n} = 0.46 \text{ with relative distance } \frac{\omega}{n} = \frac{d}{n} = H^{-1}\left(1 - \frac{k}{n}\right) = 0.1237.$$

For this code rate, we minimize the running time choosing the relative weights

$$\frac{p_1}{n} = 0.00559, \quad \frac{p_2}{n} = 0.01073, \quad \frac{p_3}{n} = 0.02029, \quad \frac{p_4}{n} = 0.03460,$$

resulting in

$$R_2 = 2^{0.01357n}, \quad R_3 = 2^{0.02668n}, \quad R_4 = 2^{0.06028n}$$

representations. Furthermore we set

$$\frac{\ell_1}{n} = 0.0366, \quad \frac{\ell_2}{n} = 0.0547, \quad \frac{\ell_3}{n} = 0.0911,$$

$$\frac{\omega_1}{n} = 0.0066, \quad \frac{\omega_2}{n} = 0.0099, \quad \frac{\omega_3}{n} = 0.0114, \quad \frac{\omega_1^{(3)}}{n} = 0.0232.$$

Optimization showed that

$$\omega_1^{(1)} = \frac{\omega_1^{(2)}}{2}, \quad \omega_2^{(2)} = \frac{\omega_2^{(3)}}{2}$$

is a good choice which yields

$$\frac{\omega_1^{(1)}}{n} = 0.011515, \quad \frac{\omega_1^{(2)}}{n} = 0.023029,$$

$$\frac{\omega_2^{(2)}}{n} = 0.016676, \quad \frac{\omega_2^{(3)}}{n} = 0.033351, \quad \frac{\omega_3^{(3)}}{n} = 0.009993$$

using condition Eq.(12) from Lemma 2. The resulting list sizes are

$$S_0 = 2^{0.02179n}, \quad S_1 = 2^{0.03987n}, \quad S_2^{0.05939n}, \quad S_3 = 2^{0.05975n}.$$

The lists on layer 0 are combined with the NN search of Alg. 3 in time

$$T_0 = 2^{0.04359n},$$

as the condition for May-Ozerov is not satisfied and Alg. 4 is less efficient in this case. On layer 1 we use Alg. 4 in time

$$T_1 = 2^{0.07356n},$$

which is also the space consumption for this step. On the remaining layers, we use May-Ozerov NN search which yields

$$T_2 = 2^{0.07365n}, \quad T_3 = 2^{0.07359n}.$$

The probability for the correct weight distribution satisfying Def. 5 is

$$P_\pi = 2^{-0.01485n}.$$

Thus the overall running time and space consumption is

$$T = \frac{T_2}{P_\pi} = 2^{0.0885n} \text{ and } S = T_1 = 2^{0.0736n}.$$

The complexity for Half Distance decoding can be shown analogously for a code rate

$$\frac{k}{n} = 0.47 \text{ with relative distance } \frac{\omega}{n} = \frac{d}{n} = H^{-1}\left(1 - \frac{k}{n}\right) = 0.06011$$

using the parameters

$$\frac{p_1}{n} = 0.002038, \quad \frac{p_2}{n} = 0.003855, \quad \frac{p_3}{n} = 0.007490, \quad \frac{p_4}{n} = 0.012200$$

$$\frac{\ell_1}{n} = 0.0125, \quad \frac{\ell_2}{n} = 0.0204, \quad \frac{\ell_3}{n} = 0.0350$$

$$\frac{\omega_1}{n} = 0.0012, \quad \frac{\omega_2}{n} = 0.0019, \quad \frac{\omega_1}{n} = 0.0019$$

$$\frac{\omega_1^{(1)}}{n} = 0.003581, \quad \frac{\omega_1^{(2)}}{n} = 0.007161, \quad \frac{\omega_1^{(3)}}{n} = 0.0062$$

$$\frac{\omega_2^{(2)}}{n} = 0.005906, \quad \frac{\omega_2^{(3)}}{n} = 0.011812, \quad \frac{\omega_3^{(3)}}{n} = 0.002200. \square$$

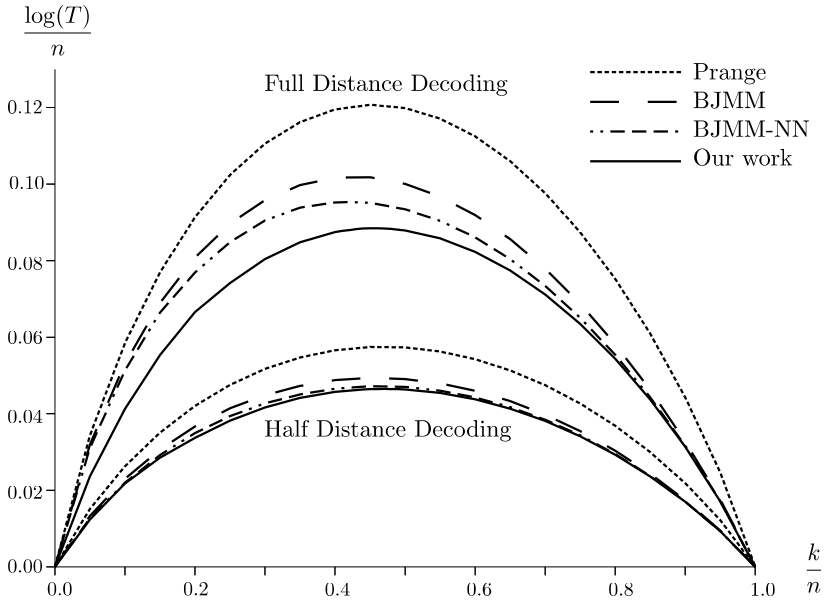

Fig. 5: [Pra62], [BJMM12], [MO15] and our algorithm for varying code rates $\frac{k}{n}$.

While Theorem 3 states the run time for the worst-case rate, Fig. 5 illustrates and compares the run time as a function of all constant rates $\frac{k}{n}$ of our algorithm to other decoding algorithms like Prange, BJMM and BJMM with NN-search, called BJMM-NN.

We also provide the C-code for optimizing all these algorithms at `https://github.com/LeifBoth/Decoding-LPN`.

Fig. 6 compares in more detail for varying depths m the complexity of our algorithm to BJMM-NN, as analyzed in [BM17b]. Here, we consider FD, HD and typical McEliece instances with $k = 0.775$ and $\omega = 0.02$ [BLP08].

In the Full Distance (FD) setting, our algorithm is superior to BJMM-NN in all depths $m = 2, 3, 4$. Already for depth $m = 3$, we beat the current FD record. Moreover, the improvement of the exponent from $0.0953n$ to only $0.0885n$ is quite significant. Another quite surprising benefit of our algorithm when compared to BJMM-NN is its modest space consumption. We were not able to improve our running times for $m = 5$, due to the large parameter space for optimization. Whether further improvements are possible currently remains an open problem.

In the Half Distance (HD) setting, our algorithm also outperforms BJMM-NN, slightly reducing the running time from $2^{0.0473n}$ to $2^{0.0465n}$. Unfortunately this improvement is not as significant as in the FD setting. The same happens for McEliece instances with a typically small error weight way below HD, where our improvement from $2^{0.0350n}$ to $2^{0.0347n}$ is only marginal.

We suspect that the strong dependency of our algorithm on the error-weight is due to the heavy reliance on Nearest Neighbor search on every layer, which needs a sufficiently large weight ω to show its strength. We will also see this effect in the case of LPN.

	[BM17b]		Our algorithm		
m	$\log(T)/n$	$\log(S)/n$	$\log(T)/n$	$\log(S)/n$	
2	0.1003	0.0781	**0.0982**	0.0717	
3	0.0967	0.0879	**0.0926**	0.0647	(FD)
4	0.0953	0.0915	**0.0885**	0.0736	
2	0.0491	0.0309	0.0488	0.0290	
3	0.0473	0.0363	0.0478	0.0290	(HD)
4	0.0473	0.0351	**0.0465**	0.0294	
2	0.0362	0.0264	0.0360	0.0260	
3	0.0350	0.0280	0.0360	0.0252	(McEliece)
4	0.0350	0.0280	**0.0347**	0.0251	

Fig. 6: Running time and memory consumption of our algorithm compared to the optimized BJMM-NN variant of [BM17b].

LPN Problem. Let us apply our algorithm to the $LPN_{k,\tau}$ problem (Def. 3). In $\mathrm{LPN}_{k,\tau}$ we have to solve a (n, k, ω)-decoding problem with expected weight $\omega = \tau n$ and fixed k. However, we are free to choose the number of samples n, and can therefore make the code rate $\frac{k}{n}$ arbitrarily small. Thus, for every fixed instance (k, τ) we minimize the running time $T(n, k, \tau)$ of our decoding algorithm over all n. The optimal number of samples for our algorithm for the cryptographically popular $\mathrm{LPN}_{512,\frac{1}{4}}$-instances is $n \approx 140.000$.

In Fig. 7, we compare different decoding algorithms for directly attacking $\mathrm{LPN}_{512,\frac{1}{4}}$, where we suppress polynomial overheads. Here BJMM-NN would take 2^{180} steps, our algorithm has complexity 2^{169}.

It is however important to stress that stand-alone decoding is not the best way to attack LPN instances. As shown by Esser, Kübler and May [EKM17] a combination of the BKW algorithm [GJL14] and decoding algorithm is due to its flexible memory requirements currently the best way to tackle LPN instances in practice. Here, one first uses BKW to turn $\mathrm{LPN}_{k,\tau}$ instances into $\mathrm{LPN}_{k',\tau'}$ instances with reduced dimension $k' < k$ at the cost of increased error $\tau' > \tau$. Then in a second step, $\mathrm{LPN}_{k',\tau'}$ is solved via decoding.

Since our decoding algorithm shows its strength for large errors τ', its seems like a perfect choice in such a hybrid BKW-decoding algorithm. In a typical attack on $\mathrm{LPN}_{512,\frac{1}{4}}$, like the ones described in [EKM17], BKW would turn $\mathrm{LPN}_{512,\frac{1}{4}}$ into $\mathrm{LPN}_{117,\frac{255}{512}}$ instances, which are subsequently decoded. The calculations in Fig. 7 give us good indication that such instances with large error τ' close to $\frac{1}{2}$ can be much faster decoded by our new algorithm. However, the full extent of our improvement has yet to be determined by real experiments.

	$\mathrm{LPN}_{512,\frac{1}{4}}$		$\mathrm{LPN}_{117,\frac{255}{512}}$	
	$\log(T)$	$\log(S)$	$\log(T)$	$\log(S)$
Prange [Pra62]	213	-	117	-
BJMM [BJMM12]	190	114	117	62
BJMM-NN [MO15]	180	122	117	64
Our algorithm	**169**	138	**75**	47

Fig. 7: Complexities of different decoding algorithms for LPN instances.

Fig. 8 shows the asymptotic behavior of our algorithm on LPN-instances for varying weights τ, which also illustrates the strength of our algorithm in the high error regime. Notice that the graph of our new algorithm's complexity can be very well approximated by a line, which yields the simple formula

$$T_{LPN}(k,\tau) = 2^{1.3k\tau}.$$

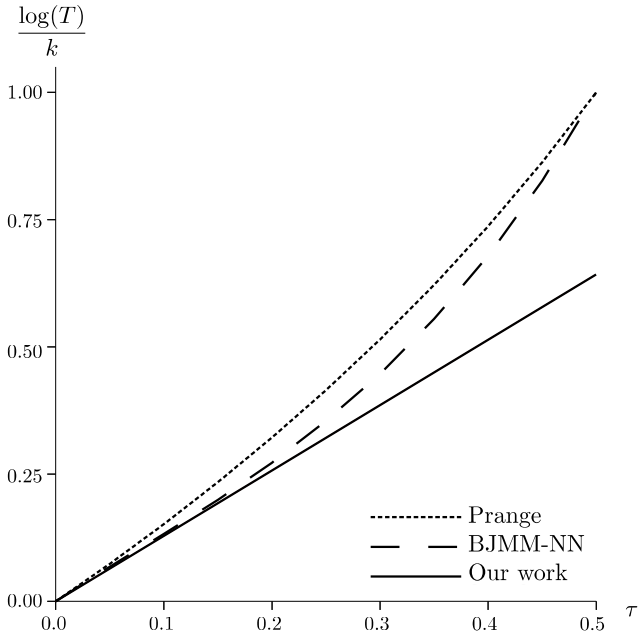

Fig. 8: Dependence on LPN error τ of [Pra62], [MO15] and our algorithm.

References

Ale03. Michael Alekhnovich. More on average case vs approximation complexity. In *44th FOCS*, pages 298–307. IEEE Computer Society Press, October 2003.

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, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, Heidelberg, April 2012.

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, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 743–760. Springer, Heidelberg, August 2011.

BM17a. Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for lpn security (full version). *Cryptology ePrint Archive: Report 2017/1139*, 2017.

BM17b. Leif Both and Alexander May. Optimizing BJMM with nearest neighbors: full decoding in $2^{2n/21}$ and McEliece security. *International Workshop on Coding and Cryptography (WCC 2017)*, 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.

EKM17. Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 486–514. Springer, Heidelberg, August 2017.

GJL14. Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2014.

McE78. RJ McEliece. A public-key system based on algebraic coding theory, 114-116. deep space 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 $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, Heidelberg, December 2011.

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, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 203–228. Springer, Heidelberg, April 2015.

NIS. NIST evaluation criteria. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`. Accessed: 2017-11-24.

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 ACM STOC*, pages 84–93. ACM Press, May 2005.

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