Revisiting Nearest-Neighbor-Based Information Set Decoding

Andre Esser

Technology Innovation Institute, UAE
andre.esser@tii.ae

Abstract. The syndrome decoding problem lies at the heart of code-based cryptographic constructions. Information Set Decoding (ISD) algorithms are commonly used to assess the security of these systems. The most efficient ISD algorithms rely heavily on nearest neighbor search techniques. However, the runtime result of the fastest known ISD algorithm by Both-May (PQCrypto '17) was recently challenged by Carrier et al. (Asiacrypt '22), which introduce themselves a new technique called RLPN decoding which yields improvements over ISD for codes with small rates $\frac{k}{n} \leq 0.3$.
In this work we first revisit the Both-May algorithm, by giving a clean exposition and a corrected analysis. We confirm the claim by Carrier et al. that the initial analysis is flawed. However, we find that the algorithm still (slightly) improves on time complexity and significantly improves on memory complexity over previous algorithms. Our first main contribution is therefore to set the correct baseline for further improvements.
As a second main contribution we then show how to improve on the Both-May algorithm, lowering the worst case running time in the full distance decoding setting to $2^{0.0948n}$. We obtain even higher time complexity gains for small rates, shifting the break even point with RLPN decoding to rate $\frac{k}{n} = 0.25$. Moreover, we significantly improve on memory for all rates $\frac{k}{n} < 0.5$. We obtain our improvement by introducing a novel technique to combine the list construction step and the list filtering step commonly applied by ISD algorithms. Therefore we treat the nearest neighbor routine in a non-blackbox fashion which allows us to embed the filtering into the nearest neighbor search. In this context we introduce the fixed-weight nearest neighbor problem, and propose a first algorithm to solve this problem. Besides resulting in an improved decoding algorithm this opens the direction for further improvements, since any faster algorithm for solving this fixed-weight nearest neighbor variant is likely to lead to further improvements of our ISD algorithm.

Keywords: representation technique · syndrome decoding · nearest neighbor search · code-based cryptography

1 Introduction

Cryptography based on the hardness of the decoding problem, known as code-based cryptography, is a promising candidate for post quantum secure systems.

The ongoing fourth round standardisation effort of NIST includes three candidates, all of them being code-based constructions. Therefore it is certain that after the end of this round at least one code-based scheme will be selected for standardisation. This makes analysis of those schemes, their security and especially strengthening our understanding of the hardness of the underlying problem an important task.

The *binary syndrome decoding problem* can be formulated as given the parity-check matrix $\mathbf{H}$ of a binary linear code of length n and dimension k as well as a syndrome $\mathbf{s} = \mathbf{He}$, recover the low Hamming weight vector $\mathbf{e}$. The fastest known algorithms for solving generic instances of this problem are usually Information Set Decoding (ISD) algorithms, pioneered by the original work of Prange in 1962 [17]. Since then there have been numerous improvements on Prange's algorithm [1–4, 7, 14–16, 18], mostly by extending the initial algorithm by an enumeration step. These works usually improve the asymptotic runtime exponent as long as the error-weight, i.e., the Hamming weight of $\mathbf{e}$, is as high as $\Omega(n)$. In this case the asymptotic running time is of the form 2^{cn}, where the constant c depends on the precise code parameters and the ISD algorithm. However, most code-based constructions do not fall into this regime by using an error-weight as small as $o(n)$. Moreover, it has been shown that the asymptotic advantage of all these improvements vanishes for a sublinear choice of the error weight [19]. And yet, the best known algorithms for attacking code-based schemes are still exactly these ISD extension of Prange's algorithm, still improving second order terms or polynomial factors in this regime.

Usually, the theoretical study of algorithmic improvements in the constant or high weight regime serves as an indicator which variations lead to practical improvements in the code-based setting. Just recently the ISD algorithms by May-Meurer-Thomae (MMT) [15] and the one by Becker-Joux-May-Meurer (BJMM) [1], both initially studied and proposed in the constant weight regime, were used to obtain new computational records in the cryptographic setting [10]. In their work, Esser, May and Zweydinger [10] identify the memory consumption of these algorithms as one of the major bottlenecks for practical applications. Further, the memory consumption, or more precisely the slowdown emerging from the *memory access cost* that goes along with accessing large amounts of random access memory (RAM), is essential for currently proposed parameter sets to reach the necessary security goals [6, 8, 10]. Therefore, for the security of code-based constructions as well as for the practical adaptation of advanced ISD techniques it is important to understand how and if this memory usage can be reduced. Recently, first time-memory trade-offs to achieve this goal were introduced, but those techniques always come at the cost of an increased time complexity.

The most recent ISD algorithms speed up the enumeration step by the use of nearest neighbor search techniques [3, 4, 16]. The fastest of these algorithms by Both-May [4] claims significant improvements on the time and memory complexity of previous proposals. However, in a recent work, Carrier, Debris-Alazard, Meyer-Hilfiger and Tillich [5] challenge the result of Both-May, by pointing out a flaw in

the analysis of its time complexity. Therefore, for now it is unclear what the best asymptotic runtime exponent of ISD algorithms is and, hence, a baseline for new improvements is missing. This baseline is of major importance to classify the gain of new ISD and other decoding algorithms, as for instance the newly proposed RLPN technique of Carrier et al. [5], which achieves runtime improvements over ISD in some regimes. In this work we clarify those doubts by giving a corrected analysis of the Both-May algorithm showing that it (slightly) improves the running time, while yielding significant memory imporovments over previous algorithms. Overall, this is in line with the results from Esser and Bellini [8] who performed a more practical study of the algorithm also observing mostly memory rather than time improvements.

After we set the bar for new ISD improvements, we extend the algorithm by Both-May improving its running time and memory complexity. We obtain our improvement by a novel technique of combining two steps which are usually performed sequentially in the enumeration part of the algorithm – the nearest neighbor search and a subsequent filtering of the found solutions according to some criterion. Therefore we treat the nearest neighbor search in non black-box fashion which allows us to directly embed the filtering into the procedure.

Our Contribution. The contribution of this work is twofold. First we provide a clean description of the most recent ISD algorithm by Both-May and a corrected analysis. Our first main contribution is therefore to provide the correct baseline for further improvements. In this context, we confirm the claim of Carrier et al. that the initial analysis of the algorithm is flawed. However, we show that the algorithm still (slightly) improves on running time and significantly lowers the memory consumption of previous ISD algorithms. More precisely, we find that the worst-case runtime in the full distance decoding setting is reduced from $2^{0.0953n}$ down-to $2^{0.0951n}$, while the memory consumption is lowered from $2^{0.092n}$ to $2^{0.076n}$, yielding the largest memory improvement made by any ISD algorithm so far.

Our second main contribution is an improvement of the algorithm by Both-May, which yields a reduced time complexity and again a significant improvement in the memory complexity. From an algorithmic perspective we achieve our improvement by a novel combination of the nearest neighbor search and a subsequently applied filtering step. We therefore treat the nearest neighbor search in a non-black box fashion to embed the filtering, such that a single application of the adapted algorithm yields the already filtered lists. In this context, we introduce a variation of the nearest neighbor problem, the *fixed-weight* nearest neighbor problem and propose a first algorithm solving the problem. Besides improving the state of the art, we therefore lead the way to further improvements, as any faster algorithm for solving this variation is likely to yield a faster ISD algorithm. Such research directions are especially desirable where recent results by Kirshanova and Laarhoven [13] rule out significant speedups of ISD algorithms via generic improvements of nearest neighbor search techniques.

Our algorithm obtains the largest improvements for small code rates, i.e., small values of k/n. Hence, the running time in the full distance setting with large worst-case rate of 0.423 is only slightly reduced to $2^{0.0948n}$ with still a significant memory improvement down-to $2^{0.071n}$. We illustrate the improvement of our algorithm (Both-May$^+$) over the algorithm by Both-May for all rates in Figure 1. The graphic shows the difference $c - c^+$, where the time (resp. memory) complexity of the Both-May and Both-May$^+$ algorithm are 2^{cn} and 2^{c^+n} respectively. Additionally we illustrate the gain of the two previous ISD improvements by Both and May [3,4] over their chronological predecessor.[1] It can be observed that our adaptation obtains by far the largest absolute improvement in time exponent over all rates. Further, it can be observed that our algorithm significantly improves the constant in the memory complexity exponent. Note that, this is the second largest memory improvement made by an ISD algorithm that also improves on time since 1962, only surpassed by the gain of the Both-May algorithm.

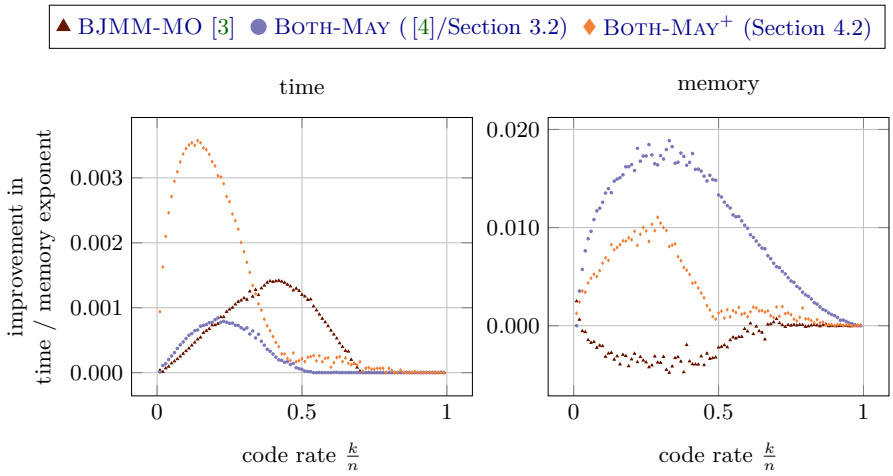

Fig. 1: Improvement in time (left) and memory exponent (right) of ISD algorithms over their chronological predecessor in the full distance setting.

Furhter, due to our improvement for small rates we shift the break even point from where the recent RLPN technique from [5] becomes preferable to rates smaller than 0.25 from initially 0.3. Note that even if RLPN is preferable for small rates, the technique requires to find low weight codewords, which is accomplished by executing an ISD algorithm as a subroutine. Hence, ISD improvements remain substantial also for improving the RLPN decoder.

[1] The chronological order of the latest four improvements is (old to new): May-Ozerov [16], BJMM-MO [3], Both-May [4] (corrected result in Section 3.2), Both-May$^+$ (Section 4.2).

All used optimization code is made available under https://github.com/Memphisd/Revisiting-NN-ISD.

Outline. In Section 2 we cover necessary basics on nearest neighbor search, the syndrome decoding problem and the general technique of ISD. In Section 3 we recall the Both-May algorithm and give a corrected analysis. Subsequently, we provide in Section 4 our improved algorithm. We conclude with a numerical optimization of our algorithm's complexity and comparison to other decoding techniques in Section 5.

2 Preliminaries

We denote vectors by bold lower case and matrices by bold upper case letters. All logarithms are base two. We use standard landau notation for complexity statements. We denote by $H(x) := -x\log(x) - (1-x)\log(1-x)$ the binary entropy function. To approximate binomial coefficients, we make use of the well known approximation

$$\binom{n}{k} = \tilde{\Theta}\left(2^{nH(k/n)}\right). \tag{1}$$

For a vector $\mathbf{v}$ we denote by v_i the projection to the i-th coordinate of $\mathbf{v}$. We extend this notation to sets of coordinates, i.e., for a set $I \subseteq \{1,\dots,n\}$, where n is the length of $\mathbf{v}$ we denote by $\mathbf{v}_I$ the projection of $\mathbf{v}$ to the coordinates indexed by I. For a binary vector $\mathbf{x} \in \mathbb{F}_2^n$, we let $\mathrm{wt}(\mathbf{x}) := |\{i \mid x_i = 1\}|$ be its Hamming weight. We refer to the set of vectors of length n and Hamming weight w as $\mathcal{B}(n,w) := \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathrm{wt}(\mathbf{x}) = w\}$.

Nearest neighbor search. Most recent ISD techniques rely on subroutines to solve a specific kind of nearest neighbor search problem. Informally, given two lists of binary vectors and a distance ε the problem asks to find all pairs with distance ε between the two lists. In our analysis we use the algorithm by May and Ozerov [16] to solve this problem, which achieves the best known time complexity. More precisely, we use a recent adaptation of the algorithm by Esser, Kübler and Zweydinger [9], which generalizes May-Ozerov's result to arbitrary list sizes and distances. The following lemma (compare to [9, Theorem 1]) states the time complexity of the algorithm

Lemma 2.1 (May-Ozerov Nearest Neighbor [9, 16]). *Let $\varepsilon \in [\![0, \frac{1}{2}]\!]$ and $\lambda \in [\![0,1]\!]$, $n \in \mathbb{N}$. Given two lists L_1, L_2 of size $|L_i| = 2^{\lambda n}$ containing uniformly at random drawn elements from $\mathbb{F}_2^n$. Then there is an algorithm that returns all pairs $(\mathbf{x}_1, \mathbf{x}_2), \mathbf{x}_i \in L_i$ with $\mathrm{wt}(\mathbf{x}_1 + \mathbf{x}_2) = \varepsilon n$ in expected time $2^{\vartheta n(1+o(1))}$, where*

$$\vartheta = \begin{cases} (1-\varepsilon)\left(1 - H\left(\frac{\delta^\star - \frac{\varepsilon}{2}}{1-\varepsilon}\right)\right) & \text{for } \varepsilon \le \varepsilon^\star \\ 2\lambda + H(\varepsilon) - 1 & \text{for } \varepsilon > \varepsilon^\star \end{cases},$$

with $\delta^\star := H^{-1}(1-\lambda)$ and $\varepsilon^\star := 2\delta^\star(1-\delta^\star)$ using memory $|L_i|^{(1+o(1))}$.

We encounter a slightly different setting where the vectors contained in the lists are of length $\ell \cdot n$ for some constant $\ell \in [\![0, 1]\!]$ instead of length n. It is easy to see that by normalizing ε and λ to ℓ we can still make use of Lemma 2.1 in this case.

Corollary 2.1. *Let $\varepsilon' \in [\![0, \frac{1}{2}]\!]$ and $\lambda', \ell \in [\![0, 1]\!]$, $n \in \mathbb{N}$. Given two lists L_1, L_2 of size $|L_i| = 2^{\lambda n}$ containing uniformly at random drawn elements from $\mathbb{F}_2^{\ell n}$. Then there is an algorithm that returns all pairs $(\mathbf{x}_1, \mathbf{x}_2), \mathbf{x}_i \in L_i$ with $\mathrm{wt}(\mathbf{x}_1 + \mathbf{x}_2) = \varepsilon' n$ in expected time $2^{\vartheta \cdot \ell n(1+o(1))}$, where ϑ is as in Lemma 2.1 for $\varepsilon := \frac{\varepsilon'}{\ell}$ and $\lambda := \frac{\lambda'}{\ell}$.*

Decoding. A binary linear code $\mathcal{C}$ of length n and dimension k is a k-dimensional subspace of $\mathbb{F}_2^n$. Such a code can be represented via the kernel of a *parity-check matrix* $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, i.e. $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n \mid \mathbf{Hc} = \mathbf{0}\}$. The task of recovering a codeword $\mathbf{c} \in \mathcal{C}$ from a given faulty version $\mathbf{c}' = \mathbf{c} + \mathbf{e}$ is known as the *decoding problem*. This problem is polynomial-time equivalent to the *syndrome decoding problem*, which asks to recover the error term $\mathbf{e}$ from the given *syndrome* $\mathbf{Hc}' = \mathbf{H}(\mathbf{c} + \mathbf{e}) = \mathbf{He}$.

Definition 2.1 (Syndrome Decoding Problem). *Let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be a random linear code of dimension k with constant* rate $\frac{k}{n}$ *and parity-check matrix $\mathbf{H}$. Given a syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$ and an integer $\omega < n$ the* syndrome decoding problem *asks to find a vector $\mathbf{e} \in \mathbb{F}_2^n$ of Hamming weight $\mathrm{wt}(\mathbf{e}) = \omega$ that satisfies $\mathbf{He} = \mathbf{s}$. We call $\mathbf{e}$ the solution and $(\mathbf{H}, \mathbf{s})$ an instance of the problem.*

Note that ω is usually rather small and that without this restriction on the Hamming weight the problem could easily be solved by Gaussian elimination. The most commonly considered setting is the *full* distance decoding setting, which bounds ω by the minimum distance of the code. The minimum distance d of a code $\mathcal{C}$ is the minimal weight of the sum of two codewords of $\mathcal{C}$, i.e., $d := \min_{\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}} \mathrm{wt}(\mathbf{c}_1 + \mathbf{c}_2) = \min_{\mathbf{c} \in \mathcal{C}} \mathrm{wt}(\mathbf{c})$. Random linear codes are known to asymptotically achieve a minimum distance of $\omega = H^{-1}(1 - k/n)n$ [11, 20]. Now, the *full distance decoding* setting bounds $\omega \leq d$, which implies that for each uniformly random choice of $(\mathbf{H}, \mathbf{s})$ there exists one solution in expectation.

Information Set Decoding (ISD). The best known strategy to solve generic instances of the syndrome decoding problem is ISD. Given an instance $(\mathbf{H}, \mathbf{s}')$ of the syndrome decoding problem, ISD algorithms first apply a random permutation $\mathbf{P}$ to the columns of $\mathbf{H}$ to obtain a permuted instance $(\mathbf{HP}, \mathbf{s}')$ with solution $\mathbf{P}^{-1}\mathbf{e}$. Then $\mathbf{HP}$ is transformed into systematic-form by multiplication with an invertible matrix $\mathbf{Q}$, which yields the identity

$$(\mathbf{QHP})(\mathbf{P}^{-1}\mathbf{e}) = (\mathbf{I}_{n-k} \mid \mathbf{H}_1)(\mathbf{e}_1, \mathbf{e}_2) = \mathbf{e}_1 + \mathbf{H}_1\mathbf{e}_2 = \mathbf{Qs}' =: \mathbf{s},$$

where $\mathbf{P}^{-1}\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$. The permutation step aims at distributing the weight on $\mathbf{P}^{-1}\mathbf{e}$ such that $\mathrm{wt}(\mathbf{e}_1) = \omega - p$ and $\mathrm{wt}(\mathbf{e}_2) = p$, where p hast to be optimized.

In a last step the algorithm then recovers $\mathbf{e}_2$ and $\mathbf{e}_1$ from the identity

$$\mathbf{H}_1\mathbf{e}_2 + \mathbf{s} = \mathbf{e}_1.$$

The subroutines to accomplish this last step differ between ISD algorithms, but commonly they rely on enumeration of the weight-p vector $\mathbf{e}_2$ and try to identify those for which $\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}$ is of small weight $\omega - p$. If this does not lead to a solution the weight was not distributed as desired and the algorithm starts over with a new random permutation.

ISD and nearest neighbor. The identity $\mathbf{H}_1\mathbf{e}_2 + \mathbf{s} = \mathbf{e}_1$ defines a nearest neighbor problem. Therefore let $\mathbf{e}_2 = (\mathbf{e}_{21}, \mathbf{e}_{22})$ and rewrite the identity as

$$\mathbf{H}_1(\mathbf{e}_{21}, \mathbf{0}) = \mathbf{H}_1(\mathbf{0}, \mathbf{e}_{22}) + \mathbf{s} + \mathbf{e}_1.$$

Since $\mathbf{e}_1$ is not known, but of small Hamming weight $\omega - p$ we have

$$\mathbf{H}_1(\mathbf{e}_{21}, \mathbf{0}) \approx \mathbf{H}_1(\mathbf{0}, \mathbf{e}_{22}) + \mathbf{s}.$$

We can solve this identity directly by applying Lemma 2.1. Therefore, enumerate all $\mathbf{e}_{2i}$ and store the left (resp. right) side of the above identity in list L_i, and let the target distance be $\varepsilon = \omega - p$.

However, prior to the result of Both-May, ISD algorithms solve the identity mostly by guessing (or enumerating) the bits of $\mathbf{e}_1$ on some projection π of its coordinates. This leads to an exact identity $\pi(\mathbf{H}_1\mathbf{e}_2) = \pi(\mathbf{s} + \mathbf{e}_1)$ where the value of $\pi(\mathbf{s} + \mathbf{e}_1)$ is known. Now the algorithms solve the problem on the projection π after which they check if they fulfill the identity on all coordinates.

Modern ISD algorithms split e_2 in multiple addends and then solve the exact identity in a binary tree fashion, where at the leaves candidates for the summands are enumerated (similar to the two list example above).

3 The Algorithm by Both-May

The algorithm by Both-May differs from previous works in how it solves the nearest-neighbor identity

$$\mathbf{H}_1\mathbf{e}_2 + \mathbf{s} = \mathbf{e}_1 \tag{2}$$

In contrast to previous works the algorithm does not enumerate coordinates of $\mathbf{e}_1$ to obtain an exact identity. Instead it solves the nearest neighbor identity directly by using the May-Ozerov nearest neighbor search algorithm.

The algorithm still relies on a search-tree to construct $\mathbf{e}_2$. Therefore it splits $\mathbf{e}_2 = \mathbf{z}_1 + \mathbf{z}_2$ in the sum of two addends. From Equation (2) it follows that $\mathbf{H}_1\mathbf{z}_1$ and $\mathbf{H}_1\mathbf{z}_2 + \mathbf{s}$ are $\mathrm{wt}(\mathbf{e}_1)$ close, since $\mathbf{e}_1$ is of small weight this implies

$$\mathbf{H}\mathbf{z}_1 \approx \mathbf{H}\mathbf{z}_2 + \mathbf{s}. \tag{3}$$

Now the algorithm makes the bet that both sides of the equation are itself small on some projection π of the coordinates, i.e., that $\mathrm{wt}\big(\pi(\mathbf{H}\mathbf{z}_1)\big) = \omega_a^{(1)}$ and $\mathrm{wt}\big(\pi(\mathbf{H}\mathbf{z}_2 + \mathbf{s})\big) = \omega_a^{(1)}$ for some small $\omega_a^{(1)}$. Then it splits $\mathbf{z}_1 = \mathbf{y}_1 + \mathbf{y}_2$ and $\mathbf{z}_2 = \mathbf{y}_3 + \mathbf{y}_4$ again in the sum of two addends. Assuming both sides of Equation (3) are indeed small on the projection π, we obtain the two nearest neighbor identities

$$\pi(\mathbf{H}\mathbf{y}_1) \approx \pi(\mathbf{H}\mathbf{y}_2) \text{ and } \pi(\mathbf{H}\mathbf{y}_3) \approx \pi(\mathbf{H}\mathbf{y}_4 + \mathbf{s}). \tag{4}$$

3.1 Depth-2 Variant

For didactic reasons let us start with the algorithm using a search tree in depth two to construct the solution $\mathbf{e}_2$. Therefore, in the base lists L_i, $i = 1, \ldots, 4$ all possible values for the $\mathbf{y}_i$ are enumerated. Then L_1, L_2 and L_3, L_4 are combined by solving the respective nearest neighbor identities from Equation (4). This yields two new lists $L_1^{(1)}$ and $L_1^{(2)}$ containing candidates for $\mathbf{z}_1$ and $\mathbf{z}_2$ respectively. In a final step the lists $L_1^{(1)}$ and $L_1^{(2)}$ are combined by solving the nearest neighbor identity from Equation (3) to find $\mathbf{e}_2$. This process is illustrated in Figure 2. A pseudocode description of the algorithm is given by Algorithm 1. In the graphic as well as in the algorithmic description the projection π is chosen to map to the first ℓ_a bits of the given vector.

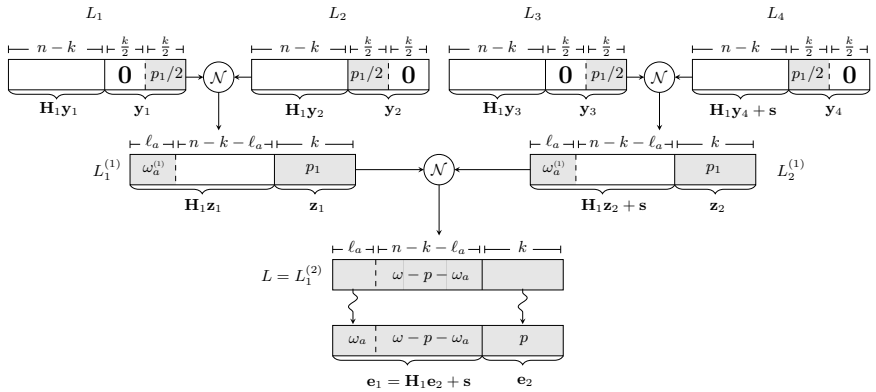

Fig. 2: Both-May algorithm in depth-2. Weight in gray regions differs from weight of uniformly random vectors. Numbers inside gray areas indicate regions of fixed weight. Curly arrows illustrate final check for contained solution, $\mathcal{N}$ indicates nearest neighbor search.

Finding a representation of the solution. Let the permutation induce a weight distribution, such that $\mathrm{wt}(\mathbf{e}_2) = p$ and $\mathrm{wt}(\pi(\mathbf{e}_1)) = \omega_a$, where π is, as

defined in Algorithm 1, the projection to the first ℓ_a coordinates of $\mathbf{e}_1$, while p, ω_a and ℓ_a have to be optimized. Also let $\mathbf{z}_i \in \mathcal{B}(k, p_1)$, $i = 1, 2$, for some p_1 that has to be optimized. Observe that this implies multiple *representations* of $\mathbf{e}_2$, i.e. multiple different pairs $(\mathbf{z}_1, \mathbf{z}_2)$ that sum to $\mathbf{e}_2$. Precisely there are

$$\mathcal{R}_1 = \binom{p}{p/2}\binom{k-p}{p_1-p/2}$$

such representations, where $\mathbf{z}_1, \mathbf{z}_2$ have both weight p_1. Here the first term counts the possibilities to distribute $p/2$ out of the p one entries of $\mathbf{e}_2$ on $\mathbf{z}_1$, while the remaining $p/2$ ones must be set in $\mathbf{z}_2$. The second factor then counts how the remaining $p_1 - p/2$ one entries in $\mathbf{z}_1$ and $\mathbf{z}_2$ can cancel out. The goal of the algorithm is to enumerate only an $1/\mathcal{R}_1$ fraction of these representations, as any representation leads to $\mathbf{e}_2$. To achieve this, a constraint on the space of representations is enforced via the weight-guess $\omega_a^{(1)}$ made on the projection π of both sides of Equation (3). The parameter $\omega_a^{(1)}$ has to be optimized as well.

On the base level all possible $\mathbf{y}_i$ are enumerated in list L_i, where we let $\mathbf{y}_1, \mathbf{y}_3 \in \mathcal{B}(k/2, p_1/2) \times 0^{k/2}$ and $\mathbf{y}_2, \mathbf{y}_4 \in 0^{k/2} \times \mathcal{B}(k/2, p_1/2)$, i.e., we perform a meet-in-the-middle split of $\mathbf{z}_1$ and $\mathbf{z}_2$. The lists L_1 and L_2 are then combined by searching those pairs $\mathbf{y}_1, \mathbf{y}_2$ with $\mathrm{wt}(\pi(\mathbf{H}_1(\mathbf{y}_1 + \mathbf{y}_2))) = \omega_a^{(1)}$. The lists L_3 and L_4 are combined analogously by previously adding $\mathbf{s}$.

Let us analyze the probability that any representation of the solution fulfills the weight-guess $\omega_a^{(1)}$ on the projection. More precisely, let the probability that for any representation $(\mathbf{z}_1, \mathbf{z}_2)$ of $\mathbf{e}_2$ we have $\mathrm{wt}(\pi(\mathbf{H}_1\mathbf{z}_1)) = \mathrm{wt}(\pi(\mathbf{H}_1\mathbf{z}_2 + \mathbf{s})) = \omega_a^{(1)}$ be q. Then we have

$$\begin{aligned} q :&= \Pr\left[\mathrm{wt}(\pi(\mathbf{H}_1\mathbf{z}_1)) = \mathrm{wt}(\pi(\mathbf{H}_1\mathbf{z}_2 + \mathbf{s})) = \omega_a^{(1)} \mid \mathbf{e}_2 = \mathbf{z}_1 + \mathbf{z}_2, \mathrm{wt}(\pi(\mathbf{e}_1)) = \omega_a\right] \\ &= \Pr\left[\mathrm{wt}(\mathbf{a}_1) = \mathrm{wt}(\mathbf{a}_2) = \omega_a^{(1)} \mid \mathbf{e}_1' = \mathbf{a}_1 + \mathbf{a}_2, \mathrm{wt}(\mathbf{e}_1') = \omega_a, \mathbf{e}_1' \in \mathbb{F}_2^{\ell_a}\right] \qquad (5) \\ &= \frac{\binom{\omega_a}{\omega_a/2}\binom{\ell_a-\omega_a}{\omega_a^{(1)}-\omega_a/2}}{2^{\ell_a}}, \end{aligned}$$

since there exist 2^{ℓ_a} pairs $\mathbf{a}_1, \mathbf{a}_2$ that fulfill $\mathbf{e}_1' = \mathbf{a}_1 + \mathbf{a}_2$, but only $\binom{\omega_a}{\omega_a/2}\binom{\ell_a-\omega_a}{\omega_a^{(1)}-\omega_a/2}$ of them have correct weight $\omega_a^{(1)}$.[2] Note that the first equality follows from the randomness of $\mathbf{H}$ and the fact that $\mathbf{e}_1 = \mathbf{H}_1\mathbf{e}_2 + \mathbf{s}$. Concluding, as long as $q \cdot \mathcal{R}_1 \geq 1$, we expect the two lists $L_1^{(1)}$ and $L_1^{(2)}$ to contain at least one representation of $\mathbf{e}_2$.

Note that our construction of $L_i^{(1)}$ (via a meet-in-the-middle split) only allows to obtain balanced $\mathbf{z}_i$, i.e., elements with weight $p_1/2$ on both halves of their coordinates. However, balanced elements form a polynomial fraction of all elements, since using Equation (1) we obtain

$$\frac{\binom{k/2}{p_1/2}^2}{\binom{k}{p_1}} = \tilde{\Theta}(1).$$

[2] This term corresponds to the number of representations of one weight-ω_a vector of length ℓ_a as sum of two weight-$\omega_a^{(1)}$ vectors.

Therefore we still can construct $\mathcal{R}_1$ representations up to a polynomial factor.

Algorithm 1: BOTH-MAY DEPTH-2

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s}' \in \mathbb{F}_2^{n-k}, \omega \in \mathbb{N}$
Output : $\mathbf{e} \in \mathbb{F}_2^n, \mathbf{He} = \mathbf{s}'$ with $\mathrm{wt}(\mathbf{e}) = \omega$

Choose optimal $p, p_1, \ell_a, \omega_a, \omega_a^{(1)}$ and define

$$\pi\colon \mathbb{F}_2^{n-k} \to \mathbb{F}_2^{\ell_a} \quad , \pi(x_1, \ldots, x_{n-k}) = \{x_1, \ldots, x_{\ell_a}\}$$
$$\bar{\pi}\colon \mathbb{F}_2^{n-k} \to \mathbb{F}_2^{n-k-\ell_a}, \bar{\pi}(x_1, \ldots, x_{n-k}) = \{x_{\ell_a+1}, \ldots, x_{n-k}\}$$

Enumerate

$$L_j = \{\mathbf{y}_j \mid \mathbf{y}_j \in \mathcal{B}(k/2, p_1/2) \times 0^{k/2}\}, \quad j = 1, 3$$
$$L_j = \{\mathbf{y}_j \mid \mathbf{y}_j \in \mathcal{B}(k/2, p_1/2) \times 0^{k/2}\}, \quad j = 2, 4$$

repeat
  choose random permutation matrix $\mathbf{P}$
  $\mathbf{H}' \leftarrow \mathbf{QHP} = \left(\mathbf{I}_{n-k}\ \mathbf{H}_1\right)$, $\mathbf{s} \leftarrow \mathbf{Qs}'$
  Compute via Nearest-Neighbor

$$L_1^{(1)} = \{\mathbf{z}_1 \mid \mathbf{z}_1 = \mathbf{y}_1 + \mathbf{y}_2,\ \mathbf{y}_i \in L_i,\ \mathrm{wt}(\pi(H_1\mathbf{z}_1)) = \omega_a^{(1)}\}$$
$$L_2^{(1)} = \{\mathbf{z}_2 \mid \mathbf{z}_2 = \mathbf{y}_3 + \mathbf{y}_4,\ \mathbf{y}_i \in L_i,\ \mathrm{wt}(\pi(H_1\mathbf{z}_2 + \mathbf{s})) = \omega_a^{(1)}\}$$
$$L \quad = \{\mathbf{e}_2 \mid \mathbf{e}_2 = \mathbf{z}_1 + \mathbf{z}_2,\ \mathbf{z}_i \in L_i^{(1)},\ \mathrm{wt}\left(\bar{\pi}(H_1\mathbf{e}_2 + \mathbf{s})\right) = \omega - \omega_a - p\}$$

  if $\exists \mathbf{e}_2 \in L\colon \mathrm{wt}(\mathbf{e}_2) = p \wedge \mathrm{wt}(\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}) = \omega - p$ **then**
    return $\mathbf{P}(\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}, \mathbf{e}_2)$

Complexity of the algorithm. The probability for the permutation distributing the weight as desired is

$$P = \frac{\binom{n}{\omega}}{\binom{\ell_a}{\omega_a}\binom{k}{p}\binom{n'}{\omega'}},$$

where $n' := n - k - \ell_a$ and $\omega' := \omega - p - \omega_a$. Hence, after P^{-1} iterations we expect to have chosen one permutation that distributes the weight as desired.

Next we investigate the time per iteration of the loop of Algorithm 1, which is dominated by the nearest neighbor search. Therefore, let us first calculate the (expected) list sizes. The base lists L_i are of size

$$\mathcal{L}_0 = \binom{k/2}{p_1/2},$$

while we expect the level-1 lists to be of size

$$\mathcal{L}_1 := \mathbb{E}[L_i^{(1)}] = (\mathcal{L}_1)^2 \cdot \frac{\binom{\ell_a}{\omega_a^{(1)}}}{2^{\ell_a}} = \tilde{\mathcal{O}}\left(\frac{\binom{k}{p_1}\binom{\ell_a}{\omega_a^{(1)}}}{2^{\ell_a}}\right),$$

since by the randomness of $\mathbf{H}$ the probability that $\mathbf{H}_1\mathbf{x}$ for any $\mathbf{x} \neq \mathbf{0}$ has weight $\omega_a^{(1)}$ on a projection to ℓ_a coordinates is $\frac{\binom{\ell_a}{\omega_a^{(1)}}}{2^{\ell_a}}$.

For the construction of the lists $L_i^{(1)}$ and L we use the May-Ozerov nearest neighbor search algorithm. The complexity of this algorithm to find all ε close pairs on lists of size $\mathcal{L}$ containing length-ℓ vectors is given by Corollary 2.1 and we denote it as $\mathcal{N}_{\mathcal{L},\ell,\varepsilon}$. Therefore the overall time complexity of the algorithm is

$$T = P^{-1} \cdot \max(\mathcal{N}_{\mathcal{L}_1,\ell_a,\omega_a^{(1)}}, \mathcal{N}_{\mathcal{L}_2,n',\omega'}),$$

while the memory complexity is $\max(\mathcal{L}_1, \mathcal{L}_2)$. Note that the final list does not affect the memory complexity, as its elements can be checked on-the-fly for being a solution. Furthermore the construction of this list is at least as expensive as its size, which is why it does not appear in the time complexity.

Complexity exponent. In our optimizations we approximate the binomial coefficients in the analysis using Equation (1). Then for each optimization parameter o_i we let $o_i = \hat{o}_i \cdot n$, where $\hat{o}_i \in [\![0, 1]\!]$. Furthermore, we similarly let $k = \hat{k}n$, where $\hat{k} = \frac{k}{n}$ is the rate of the code. We then minimize the running time over the choices of the $\hat{o}_i$ under the correctness constraint $q\mathcal{R}_1 \geq 1$. Finally we maximize over all possible choices for the rate $\hat{k}$ with corresponding weight $\omega = \hat{\omega}n = H^{-1}(1-\hat{k})n$ (full distance setting). This results in a complexity of the form 2^{cn} for constant c.

The numerical optimization leads to a running time of $T = 2^{0.0982n}$ with memory complexity $M = 2^{0.716n}$ at worst-case rate $\hat{k} = 0.422$ and, hence, $\omega = H^{-1}(1-0.422)n \approx 0.1373$.

We stress that these results essentially match those given in the original work of Both-May [4]. The reason is that in contrast to higher search tree depth variants the depth-2 variant does not make use of a filtering step, which introduced the flaw in the analysis of [4] as we describe in the following section.

3.2 Depth-4 Variant

Both and May obtain their best result for a tree in depth four. Here the splitting of $\mathbf{e}_2$ is continued recursively, i.e. $\mathbf{y}_i = \mathbf{x}_{2i-1} + \mathbf{x}_{2i}$, $i = 1, 2, 3, 4$ and $\mathbf{x}_j = \mathbf{w}_{2j-1} + \mathbf{w}_{2j}$, $j = 1, \ldots, 8$. The algorithm then recursively makes a bet on the smallness of $\mathbf{H}\mathbf{y}_i$ (respectively $\mathbf{H}\mathbf{y}_4 + \mathbf{s}$) and $\mathbf{H}\mathbf{x}_j$ (respectively $\mathbf{H}\mathbf{x}_8 + \mathbf{s}$) on some projections to obtain nearest neighbor identities for each level. Also it enforces a specific weight on the vectors $\mathbf{y}_i$ and $\mathbf{z}_i$ itself. Eventually, all possible $\mathbf{w}_j$ are enumerated in the base lists L_j, $j = 1, \ldots, 16$.

Similar to before the $\mathbf{w}_i$ form a meet-in-the-middle split of the $\mathbf{x}_j$, i.e., $\mathbf{w}_{2i-1} \in \mathcal{B}(k/2, p_1/2) \times 0^{k/2}$ and $\mathbf{w}_{2i} \in 0^{k/2} \times \mathcal{B}(k/2, p_1/2)$, where p_1 is subject to optimization.

Additionally, a filtering step is introduced after the construction of the level-2 and level-3 lists. This filtering step discards all vectors which do not sum to predefined weights or which do not sum to predefined weights on projections that already have fixed weights, i.e., those already used for nearest neighbor search on previous levels (compare to Figure 3).

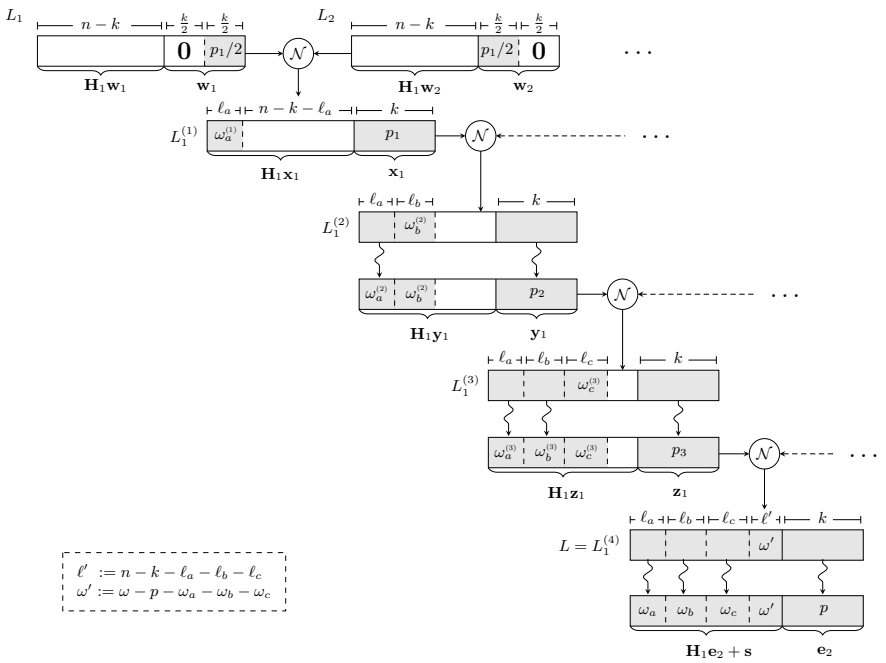

Fig. 3: Leftmost path of depth-4 algorithm from leaves (base lists) to root (final list). Gray areas indicate regions where weight differs from weight of uniformly random vectors. Numbers inside gray areas indicate regions of fixed weight. Curly arrows illustrate filtering process, $\mathcal{N}$ indicates nearest neighbor search.

The pseudocode of the algorithm is given by Algorithm 2 and an illustration in Figure 3. For simplification we choose the projections on each level to be the next ℓ_a, ℓ_b and ℓ_c coordinates respectively. More precisely, we define

$$\begin{aligned}
\pi_a &\colon \mathbb{F}_2^{n-k} \to \mathbb{F}_2^{\ell_a},\ \pi_a(x_1, \ldots, x_{n-k}) = \{x_1, \ldots, x_{\ell_a}\} \\
\pi_b &\colon \mathbb{F}_2^{n-k} \to \mathbb{F}_2^{\ell_b},\ \pi_b(x_1, \ldots, x_{n-k}) = (x_{\ell_a+1}, \ldots, x_{\ell_a+\ell_b}) \\
\pi_c &\colon \mathbb{F}_2^{n-k} \to \mathbb{F}_2^{\ell_c},\ \pi_c(x_1, \ldots, x_{n-k}) = \{x_{\ell_a+\ell_b+1}, \ldots, x_{\ell_a+\ell_b+\ell_c}\}
\end{aligned} \tag{6}$$

Analogously to the depth-2 case, we let

$$\bar{\pi}\colon \mathbb{F}_2^{n-k} \to \mathbb{F}_2^{n-k-\ell'}, \ell' := \ell_a + \ell_b + \ell_c \text{ with } \bar{\pi}(\mathbf{x}) = (x_{\ell'+1}, \dots, x_{n-k}) \tag{7}$$

be the projection to the remaining coordinates.

Remark 3.1 (Block notation). We use letters to refer to different projections (or blocks) of coordinates while we use numbers to indicate different levels of the tree. For instance, $\omega_b^{(3)}$ is the predefined weight of block b on level 3.

Algorithm 2: Both-May Depth-4

Input : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s}' \in \mathbb{F}_2^{n-k}, \omega \in \mathbb{N}$
Output: $\mathbf{e} \in \mathbb{F}_2^n, \mathbf{He} = \mathbf{s}'$ with $\mathrm{wt}(e) = \omega$

Choose optimal $p, p_1, p_2, p_3, \ell_a, \ell_b, \ell_c, \omega_a, \omega_b, \omega_c, \omega_a^{(1)}, \omega_a^{(2)}, \omega_a^{(3)}, \omega_b^{(2)}, \omega_b^{(3)}, \omega_c^{(3)}$
Let $\pi_a, \pi_b, \pi_c, \bar{\pi}$ be defined as in Equations (6) and (7)
Enumerate

$$L_j = \{\mathbf{w}_j \mid \mathbf{w}_j \in \mathcal{B}(k/2, p_1/2) \times 0^{k/2}\}, \quad j = 1, 3, \dots, 15$$
$$L_j = \{\mathbf{w}_j \mid \mathbf{w}_j \in \mathcal{B}(k/2, p_1/2) \times 0^{k/2}\}, \quad j = 2, 4, \dots, 16$$

repeat
  choose random permutation matrix $\mathbf{P}$
  $\mathbf{H}' \leftarrow \mathbf{QHP} = \left(\mathbf{I}_{n-k}\ \mathbf{H}_1\right)$, $\mathbf{s} \leftarrow \mathbf{Qs}'$ and define $\mathbf{s}_{j,i} := \begin{cases} \mathbf{s} & , i = j \\ 0^{n-k} & , \text{else} \end{cases}$
  Compute level-1 lists via nearest neighbor for $i = 1, \dots, 8$

$$L_i^{(1)} = \{\quad \mathbf{x}_i \quad \mid \mathbf{x}_i = \mathbf{w}_{2i-1} + \mathbf{w}_{2i},\ \mathbf{w}_j \in L_j,\ \mathrm{wt}(\pi_a(H_1\mathbf{x}_i + \mathbf{s}_{8,i})) = \omega_a^{(1)}\}$$

  Compute via nearest neighbor then filter level-2 lists for $i = 1, \dots, 4$

$$L_i^{(2)} = \{\quad \mathbf{y}_i \quad \mid \mathbf{y}_i = \mathbf{x}_{2i-1} + \mathbf{x}_{2i},\ \mathbf{x}_j \in L_j^{(1)},\ \mathrm{wt}(\pi_b(H_1\mathbf{y}_i + \mathbf{s}_{4,i})) = \omega_b^{(2)}\}$$
$$L_i^{(2)} \leftarrow \{\mathbf{y} \in L_i^{(2)} \mid \mathrm{wt}\left(\pi_a(\mathbf{H}_1\mathbf{y} + \mathbf{s}_{4,i})\right) = \omega_a^{(2)} \wedge \mathrm{wt}(\mathbf{y}) = p_2\}$$

  Compute via nearest neighbor then filter level-3 lists for $i = 1, 2$

$$L_i^{(3)} = \{\quad \mathbf{z}_i \quad \mid \mathbf{z}_i = \mathbf{y}_{2i-1} + \mathbf{y}_{2i},\ \mathbf{y}_j \in L_j^{(2)},\ \mathrm{wt}(\pi_c(H_1\mathbf{z}_i + \mathbf{s}_{2,i})) = \omega_c^{(3)}\}$$
$$L_i^{(3)} \leftarrow \{\mathbf{z} \in L_i^{(3)} \mid \mathrm{wt}\left(\pi_a(\mathbf{v}_i)\right) = \omega_a^{(3)} \wedge \mathrm{wt}\left(\pi_b(\mathbf{v}_i)\right) = \omega_b^{(3)} \wedge \mathrm{wt}(\mathbf{z}) = p_3\}$$
$$\text{, with } \mathbf{v}_i := \mathbf{H}_1\mathbf{z} + \mathbf{s}_{2,i}$$

  Compute final (level-4) list via nearest neighbor, $\omega' := \omega - \omega_a - \omega_b - \omega_c - p$

$$L = \{\quad \mathbf{e}_2 \quad \mid \mathbf{e}_2 = \mathbf{z}_{2i-1} + \mathbf{z}_{2i},\ \mathbf{z}_j \in L_j^{(3)},\ \mathrm{wt}(\bar{\pi}(\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}')) = \omega'\}$$

  if $\exists \mathbf{e}_2 \in L\colon \mathrm{wt}(\mathbf{e}_2) = p \wedge \mathrm{wt}(\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}') = \omega - p$ **then**
    return $\mathbf{P}(\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}', \mathbf{e}_2)$

Finding a representation of the solution. Let us assume the permutation $\mathbf{P}$ distributes the weight on $\mathbf{P}^{-1}\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$ such that

$$\mathrm{wt}(\mathbf{e}_2) = p \text{ and } \mathrm{wt}\big(\pi_\delta(\mathbf{e}_1)\big) = \omega_\delta \text{ for } \delta \in \{a, b, c\},$$

which implies $\mathrm{wt}\big(\bar{\pi}(\mathbf{e}_1)\big) = \omega - \omega_a - \omega_b - \omega_c - p$.

The algorithm constructs on each level $i = 1, 2, 3$ vectors of weight p_i that should sum to weight-p_{i+1} vectors, where $p_4 := p$. Note that each such weight-p_{i+1} vector has $\mathcal{R}_i$ representations as sum of weight-p_i vectors, where

$$\mathcal{R}_i = \binom{p_{i+1}}{p_{i+1}/2}\binom{k - p_{i+1}}{p_i - p_{i+1}/2}.$$

Therefore, we intend again to enumerate an $1/\mathcal{R}_i$-fraction of all possible representations to ensure that their is one representation on expectation of each weight-p_{i+1} vector contained on level i. Let us analyze the constraint imposed on each level introduced by restricting to a specific weight on the projections π_a, π_b and π_c. We have already seen in Section 3.1 that the probability that any representation of a level-2 element survives the level-1 constraint is

$$q_1 = \frac{\binom{\omega_a^{(2)}}{\omega_a^{(2)}/2}\binom{\ell_a - \omega_a^{(2)}}{\omega_a^{(1)} - \omega_a^{(2)}/2}}{2^{\ell_a}},$$

compare to Equation (5). By the same reasoning if we now on level-2 impose weight restrictions on both projections π_a and π_b, we obtain

$$\begin{aligned} q_2 &:= \prod_{\delta \in \{a,b\}} \Pr\left[\mathrm{wt}(\pi_\delta(\mathbf{H}\mathbf{y}_1)) = \mathrm{wt}(\pi_\delta(\mathbf{H}\mathbf{y}_2)) = \omega_\delta^{(2)} \mid \mathrm{wt}(\pi_\delta(\mathbf{H}(\mathbf{y}_1 + \mathbf{y}_2))) = \omega_\delta^{(3)}\right] \\ &= \prod_{\delta \in \{a,b\}} \Pr_{\mathbf{a}_i \in \mathbb{F}_2^{\ell_\delta}}\left[\mathrm{wt}(\pi_\delta(\mathbf{a}_1)) = \mathrm{wt}(\pi_\delta(\mathbf{a}_2)) = \omega_\delta^{(2)} \mid \mathrm{wt}(\pi_\delta(\mathbf{a}_1 + \mathbf{a}_2)) = \omega_\delta^{(3)}\right] \\ &= \frac{\binom{\omega_a^{(3)}}{\omega_a^{(3)}/2}\binom{\ell_a - \omega_a^{(3)}}{\omega_a^{(2)} - \omega_a^{(3)}/2}}{2^{\ell_a}} \cdot \frac{\binom{\omega_b^{(3)}}{\omega_b^{(3)}/2}\binom{\ell_b - \omega_b^{(3)}}{\omega_b^{(2)} - \omega_b^{(3)}/2}}{2^{\ell_b}}. \end{aligned}$$

Eventually for the last level we obtain analogously

$$\begin{aligned} q_3 &:= \prod_{\delta \in \{a,b,c\}} \Pr\left[\mathrm{wt}(\pi_\delta(\mathbf{H}\mathbf{z}_1)) = \mathrm{wt}(\pi_\delta(\mathbf{H}\mathbf{z}_2)) = \omega_\delta^{(3)} \mid \mathrm{wt}(\pi_\delta(\mathbf{H}(\mathbf{z}_1 + \mathbf{z}_2))) = \omega_\delta\right] \\ &= \prod_{\delta \in \{a,b,c\}} \frac{\binom{\omega_\delta}{\omega_\delta/2}\binom{\ell_\delta - \omega_\delta}{\omega_\delta^{(3)} - \omega_\delta/2}}{2^{\ell_\delta}}. \end{aligned}$$

Now as long as we have $q_i \cdot \mathcal{R}_i \geq 1$ we ensure that in expectation on each level i at least one representation of each possible level-$(i+1)$ element, i.e., of each $\mathbf{x} \in \mathbb{F}_2^k$ with $\mathrm{wt}(\mathbf{x}) = p_{i+1}$ is present. This implies in turn that on level 3 there is a representation of the searched weight-p vector $\mathbf{e}_2$. Since we conditioned on $\mathrm{wt}(\bar{\pi}(\mathbf{e}_1)) = \omega - p - \omega_a - \omega_b - \omega_c$ this representation is found by the level-4 list construction.

Note that to avoid duplicates in the lists we will also optimize parameters according to the constraint $q_i \cdot \mathcal{R}_i \leq 1$, which implies $q_i \cdot \mathcal{R}_i = 1$.

Complexity of the algorithm. The probability for the permutation distributing the weight as desired is

$$P = \frac{\binom{n}{\omega}}{\binom{\ell_a}{\omega_a}\binom{\ell_b}{\omega_b}\binom{\ell_c}{\omega_c}\binom{\ell'}{\omega'}\binom{k}{p}},$$

where $\ell' := n - k - \ell_a - \ell_b - \ell_c$ and $\omega' := \omega - p - \omega_a - \omega_b - \omega_c$. Therefore after P^{-1} iterations we expect one to distribute the weight as desired.

Now let us analyze the cost to construct the tree. First, we argue about the expected list size on each level after filtering, which is exactly where the analysis of [4] goes wrong. The base lists are analogously to the depth-2 variant of size

$$\mathcal{L}_0 = \binom{k/2}{p_1/2}.$$

Now, we have already shown that for suitable parameters, satisfying $q_i\mathcal{R}_i = 1$, on level $i = 1, 2, 3$ there exists exactly one representation of each possible level-$(i+1)$ element, i.e., of each $\mathbf{x} \in \mathbb{F}_2^k$ with $\mathrm{wt}(\mathbf{x}) = p_{i+1}$. Therefore the expected list size on level-i after filtering is

$$\mathcal{L}_i = \binom{k}{p_i} \cdot \rho_i,$$

where ρ_i is the probability that a vector $\mathbf{x} \in \mathbb{F}_2^k$ fulfills the level-i restriction. Since level i imposes a weight restriction on a total of i blocks, we have

$$\rho_1 = \frac{\binom{\ell_a}{\omega_a^{(1)}}}{2^{\ell_a}}, \quad \rho_2 = \frac{\binom{\ell_a}{\omega_a^{(2)}}\binom{\ell_b}{\omega_b^{(2)}}}{2^{\ell_a+\ell_b}} \quad \text{and} \quad \rho_3 = \frac{\binom{\ell_a}{\omega_a^{(3)}}\binom{\ell_b}{\omega_b^{(3)}}\binom{\ell_c}{\omega_c^{(3)}}}{2^{\ell_a+\ell_b+\ell_c}}.$$

In Appendix A we outline the difference to the original analysis of [4].

The time complexity per iteration of the loop is again given by the time it takes to construct all lists. The level-i lists, $i = 1, 2, 3$ are constructed via a nearest neighbor search on lists of size $\mathcal{L}_{i-1}$ including vectors of length ℓ_δ with target weight $\omega_\delta^{(i)}$, $\delta \in \{a, b, c\}$. The final list is then constructed via nearest neighbor search on the remaining ℓ' coordinates for target weight ω'. Therefore the cost for each level i is T_i, where

$$T_1 = \mathcal{N}_{\mathcal{L}_0,\ell_a,\omega_a^{(1)}}, \quad T_2 = \mathcal{N}_{\mathcal{L}_1,\ell_b,\omega_b^{(2)}}, \quad T_3 = \mathcal{N}_{\mathcal{L}_2,\ell_c,\omega_c^{(3)}} \quad \text{and} \quad T_4 = \mathcal{N}_{\mathcal{L}_3,\ell',\omega'},$$

Eventually the total time complexity of Algorithm 2 is given as the number of iterations times the cost for one iteration, giving

$$T = P^{-1} \max_i(T_i).$$

Numerical Optimization of Algorithm 2. We follow the same optimization methodology as for the depth-2 case, under correctness constraints $q_i\mathcal{R}_i = 1$, to obtain the asymptotic running time and memory exponents. We find a worst

case rate for the algorithm of $\hat{k} = 0.42$ with $\omega = H^{-1}(1-\hat{k}) \approx 0.1384$, leading to a time and memory complexity of

$$T = 2^{0.0951n} \quad \text{and} \quad M = 2^{0.076n},$$

for optimal parameters[3]

$$\begin{aligned}
\hat{p} &= 0.05180, & \hat{p}_3 &= 0.04719, & \hat{p}_2 &= 0.03371, & \hat{p}_1 &= 0.01783,\\
\hat{\ell}_a &= 0.05280, & \hat{\ell}_b &= 0.10178, & \hat{\ell}_c &= 0.12367,\\
\hat{\omega}_a &= 0.00651, & \hat{\omega}_a^{(3)} &= 0.00593, & \hat{\omega}_a^{(2)} &= 0.00428, & \hat{\omega}_a^{(1)} &= 0.05,\\
\hat{\omega}_b &= 0.01220, & \hat{\omega}_b^{(3)} &= 0.01091, & \hat{\omega}_b^{(2)} &= 0.09414,\\
\hat{\omega}_c &= 0.01504, & \hat{\omega}_c^{(3)} &= 0.01354.
\end{aligned}$$

While this running time is far greater than the initially claimed $2^{0.0885n}$ [4], it still slightly improves on the previously best running time of $2^{0.0953n}$ reported in [3]. Further, the memory complexity is drastically improved by a factor of $2^{0.0197n}$ from previously $2^{0.0915n}$ to $2^{0.0754n}$. We give a more detailed comparison to other ISD algorithms also for different rates in Section 5.

4 An Improved Algorithm

Note that the Both-May algorithm works on each level in two steps. First it combines two lists of the previous level to obtain vectors which fulfill a weight restriction on a subset of the coordinates. Then in a second step it filters the vectors for the weight restriction on the remaining coordinates. We improve this by embedding the filter process into the nearest neighbor search algorithm, to directly obtain vectors that satisfy the weight restriction on all coordinates.

In the following we first show how to adapt the May-Ozerov algorithm to also perform the filtering step. After that we describe how to integrate this adaptation into the Both-May algorithm and how its complexity changes.

Our adaptation of the May-Ozerov algorithm requires to solve a specific variant of a nearest neighbor problem as a subroutine. Therefore in Section 4.3 we develop an algorithm to solve this variant and upper bound its complexity to finally obtain a numerical complexity estimate for our whole decoding procedure.

4.1 Combining Nearest Neighbor Search and Filtering

Let us first briefly recall how the May-Ozerov nearest neighbor search algorithm finds all ε-close pairs between two same-sized input lists L_1, L_2 containing uniformly random vectors from $\mathbb{F}_2^m$ and how its complexity is composed. For an in-depth explanation and analysis the reader is referred to [9, 16]. First, the

[3] Due to rounding to a precision of 10^{-5} there might be a certain deviation in satisfying the correctness constraints. For the exact numbers we refer to our optimization scripts, which are provided as supplementary material.

algorithm computes an exponential number of list pairs L'_1, L'_2 from the initial lists. For optimal parameter choices it is guaranteed that L'_1, L'_2 each have only polynomial size, while simultaneously any distance-ε pair between L_1 and L_2 is still contained in at least one of the constructed pairs L'_1, L'_2. In a final step the algorithm then finds the ε-close pairs by computing $L'_1 \times L'_2$ for every list pair L'_1, L'_2 naively.

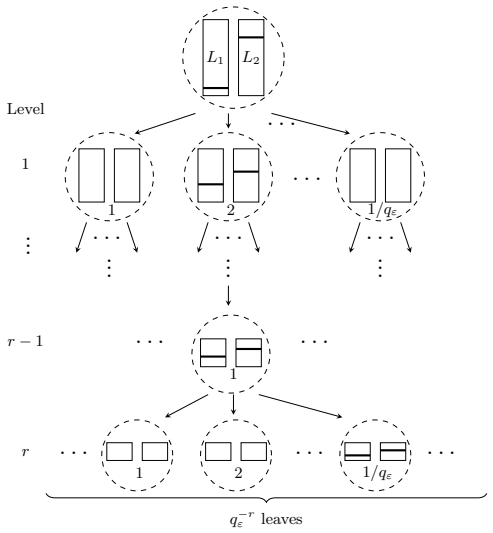

Fig. 4: Illustration of the May-Ozerov nearest neighbor search algorithm. Arrows indicate the application of a locality sensitive filter. Each node branches q_ε^{-1} times. Bold stripes in lists indicate pair of distance ε progressing through all r applied filters.

The list pairs are computed in a tree-like fashion, where the input pair L_1, L_2 forms the root of the tree (compare to Figure 4). This tree is constructed iteratively, level by level. In every step of the algorithm each leaf of the tree is branched $1/q_\varepsilon$ times. A child-node is computed by traversing both lists of the parent node and applying a locality-sensitive filter to each element. This filter discards elements that do not match the filter criterion and, hence, reduces the lists' sizes. Furthermore, it has the property, that an arbitrary element passes the filter with probability q_f, while for an ε-close pair $(\mathbf{x}, \mathbf{y})$ between the lists, $\mathbf{x}$ and $\mathbf{y}$ pass the filter at the same time with probability $q_\varepsilon > q_\mathrm{f}^2$. Therefore close pairs are more likely to pass the filter than non-close pairs. The branching factor of q_ε^{-1} ensures that if there is an element of distance ε contained in the current node, it progresses to the next level through at least one of the filters.

This procedure is repeated r times to construct a tree of depth r containing q_ε^{-r} leaves.[4]

The time complexity is then given (compare to [9, Theorem 2][5]) as

$$T_{\mathrm{MO}} = q_\varepsilon^{-r} \cdot \max\left(|L_1| \cdot q_{\mathrm{f}}^{r-1}, (|L_1| \cdot q_{\mathrm{f}}^{r})^2\right)^{1+o(1)}. \tag{8}$$

Here, the first term in the maximum describes the size of lists after applying the filter $r-1$ times and corresponds to the cost of constructing a single leaf of the tree. In [9] it is shown that the construction of the leaves from its parents dominates the construction of the whole tree. The second term describes the cost for naively finding all ε-close pairs between list pairs contained in the leaves, i.e., after applying the filter r times. The initial factor in front of the maximum accounts for the number of leaves.

Furthermore, for the locality sensitive filter criterion chosen in [9] the probabilities q_ε and q_{f} are parameterized via an optimization parameter $\delta < m$ and are given as

$$q_\varepsilon = \binom{\varepsilon/r}{\varepsilon/2r}\binom{(m-\varepsilon)/r}{(\delta-\varepsilon/2)/r}\left(\frac{1}{2}\right)^{m/r} \quad \text{and} \quad q_{\mathrm{f}} = \binom{m/r}{\delta/r}\left(\frac{1}{2}\right)^{m/r}. \tag{9}$$

Note that for the optimal choice of δ the maximum from Equation (8) is dominated by the first term, as the second one becomes polynomial. Further, optimal parameter choices lead to the result stated in Lemma 2.1.

Algorithm 3: May-Ozerov

Input : lists $L_1, L_2 \in (\mathbb{F}_2^m)^*$, integer ε
Output : all pairs $\mathbf{x}, \mathbf{y} \in L_1 \times L_2$ with $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \varepsilon$

Choose optimal δ, r and let q_ε be as in Equation (9), $L \leftarrow \emptyset$
Construct q_ε^{-r} list pairs L_1', L_2', each by applying a locality-sensitive filter r times to L_1, L_2
foreach *pair* L_1', L_2' **do**
    $L \leftarrow L \cup \{(\mathbf{x}, \mathbf{y}) \in L_1', L_2' \mid \mathrm{wt}(\mathbf{x}+\mathbf{y}) = \varepsilon\}$
return L

Adapting the May-Ozerov Algorithm. In Algorithm 2 the May-Ozerov algorithm operates only on a projection π to m out of n coordinates, e.g., on projection $\pi := \bar{\pi}$ to construct the final level-4 list. However, for the remaining

[4] In [9] r was chosen as $\frac{m}{\log^2 m}$ to ease the analysis. However, in [16] it was shown that a large enough constant is already sufficient.

[5] In comparison to [9] we assume $2^{\lambda m} \cdot q_{\mathrm{f}}^{r-1} \geq 1$ to remove one term from the maximum.

$n - m$ coordinates there are also weight restrictions, which are imposed via the filtering step. We now exchange the naive search for ε-close pairs on projection π at the leaf-level (line 4 of Algorithm 3) by an algorithm that finds those vectors that match the weight restriction on the remaining $\ell := n - m$ coordinates. We then only keep those pairs attaining distance ε on the projection π. A formal description of the adapted algorithm is given by Algorithm 4.

If we denote the time complexity to find all ω_2 close pairs between two lists of size $\mathcal{L}$ containing vectors from $\mathcal{B}(\ell, \omega_1)$ as $\mathcal{F}_{\mathcal{L},\ell,\omega_2,\omega_1}$, the time complexity of Algorithm 4 is given as

$$T_{\text{MO}^+} = \mathcal{N}^{\ell,\omega_1,\omega_2}_{|L_1|,m,\varepsilon} := q_\varepsilon^{-r} \cdot \max\left(|L_1| \cdot q_{\text{f}}^{r-1}, \mathcal{F}_{\mathcal{L},\ell,\omega_1,\omega_2}\right)^{1+o(1)}, \tag{10}$$

where $\mathcal{L} = |L_1| \cdot q_{\text{f}}^r$. As long as $\mathcal{F}_*$ is smaller than $\mathcal{L}^2$ we expect a re-balancing of both terms in the maximum to yield an improved time complexity, i.e. $T_{\text{MO}^+} < T_{\text{MO}}$.

Algorithm 4: MAY-OZEROV$^+$

Input : lists $L_1, L_2 \in (\mathbb{F}_2^m \times \mathcal{B}(\ell, \omega_1))^*$, integer $\varepsilon, \omega_1, \omega_2$
Output: all pairs $\mathbf{x}, \mathbf{y} \in L_1 \times L_2$ with $\mathbf{x} + \mathbf{y} \in \mathcal{B}(m, \varepsilon) \times \mathcal{B}(\ell, \omega_2)$

1 Choose optimal δ, r and let q_ε be as in Equation (9), $L \leftarrow \emptyset$
2 Construct q_ε^{-r} list pairs L_1', L_2', each by applying a locality-sensitive filter r times to L_1, L_2
3 **foreach** *pair* L_1', L_2' **do**
4 Compute then filter:
$$L \leftarrow L \cup \{(\mathbf{x}, \mathbf{y}) \in L_1', L_2' \mid \mathbf{x} + \mathbf{y} \in \mathbb{F}_2^m \times \mathcal{B}(\ell, \omega_2)\}$$
$$L \leftarrow \{(\mathbf{x}, \mathbf{y}) \in L \mid \mathbf{x} + \mathbf{y} \in \mathcal{B}(m, \varepsilon) \times \mathcal{B}(\ell, \omega_2)\}$$
5 **return** L

Note that $\mathcal{F}_*$ describes the complexity to solve a variant of the nearest neighbor problem, where the input vectors have fixed weight. Let us define this problem more formally.

Definition 4.1 (Fixed Weight Nearest Neighbor Problem). *Let ℓ, ω_1, ω_2 be integers with $\omega_1, \omega_2 \leq \ell$. Given two lists L_1, L_2 of same size containing uniformly at random drawn elements from $\mathcal{B}(\ell, \omega_1)$ the* fixed weight nearest neighbor *problem asks to find all pairs $(\mathbf{x}_1, \mathbf{x}_2), \mathbf{x}_i \in L_i$ with $\text{wt}(\mathbf{x}_1 + \mathbf{x}_2) = \omega_2$*

But before we discuss how to solve this problem let us first describe how to incorporate MAY-OZEROV$^+$ into the decoding procedure.

4.2 Both-May$^+$ – Embedding MayOzerov$^+$ into the Decoding Algorithm

In the following we integrate our adapted May-Ozerov algorithm May-Ozerov$^+$ into the decoding algorithm (Algorithm 2) and describe how it affects the time complexity of the decoding procedure.

To be able to exchange the conventional May-Ozerov nearest neighbor search used for list construction in Algorithm 2 by our adaptation, we require that input vectors have a certain amount of coordinates with fixed weight. Therefore note that on every level i the vectors in the lists are ensured to have fixed weight p_i, where $p_4 := p$ (compare to Figure 3). Further, from level $i \geq 1$ on-wards the product $l_{j,i} := \mathbf{H}\mathbf{v}_j + \mathbf{s}_{j,2^{4-i}}$ for any $\mathbf{v}_j \in L_j^{(i)}$ has fixed weight on projection π_a. For $i \geq 2$, $l_{i,j}$ has additionally fixed weight on projection π_b and, finally, for $i \geq 3$ we find that $l_{i,j}$ has fixed weight on all projections π_δ for $\delta \in \{a, b, c\}$.

Therefore, we can exchange the computation of level-2, level-3 and level-4 lists in lines 8, 9 and 10 of Algorithm 2 by directly computing the following (partly) filtered lists (from the lists of the previous level)

$$
\begin{aligned}
L_i^{(2)+} &= \{\mathbf{y}_i \in L_i^{(2)} \mid \mathrm{wt}(\mathbf{y}_i) + \quad\quad\quad\quad \mathrm{wt}(\pi_a(H_1\mathbf{y}_i + \mathbf{s}_{4,i})) = p_2 + \omega_a^{(2)} \quad\quad\} \\
L_i^{(3)+} &= \{\mathbf{z}_i \in L_i^{(3)} \mid \mathrm{wt}(\mathbf{z}_i) + \sum_{\delta\in\{a,b\}} \mathrm{wt}(\pi_\delta(H_1\mathbf{z}_i + \mathbf{s}_{2,i})) = p_3 + \omega_a^{(3)} + \omega_b^{(3)}\} \\
L^{+} &= \{\mathbf{e}_2 \in L \quad\; \mid \mathrm{wt}(\mathbf{e}_2) + \sum_{\delta\in\{a,b,c\}} \mathrm{wt}(\pi_\delta(H_1\mathbf{e}_2 + \mathbf{s})) \quad = p + \sum_{\delta\in\{a,b,c\}} \omega_\delta \;\}.
\end{aligned}
$$

We call the resulting algorithm Both-May$^+$ in the following. Note that the constraints ensure that the weights of all blocks sum to the correct value, while it is not enforced that every block itself has the desired weight. Therefore, to reduce the list size further, we still perform the usual filtering step. Further, since on level one there is by construction no filtering, we stay with the conventional May-Ozerov algorithm for level-1 list creation.

Complexity. The analysis of the time complexity follows along the lines of the analysis in Section 3.2 with the only difference that the time complexities for creating the level-2,-3 and -4 lists, given by T_2, T_3 and T_4, have to be adapted. The time complexity for this list construction is given as T_{MO^+} in Equation (10). This complexity is determined by the parameters of the nearest neighbor problem (the subscripts of $\mathcal{N}$) and the parameters of the fixed weight nearest neighbor problem (the superscripts of $\mathcal{N}$). The parameters of the standard nearest neighbor problem remain the same as in the previous analysis in Section 3.2. Let us determine the parameters of the fixed weight nearest neighbor problem in the following. First we define

$$\ell_2 := k + \ell_a, \quad \ell_3 := k + \ell_a + \ell_b \quad \text{and} \quad \ell_4 := k + \ell_a + \ell_b + \ell_c,$$

and

$$\varepsilon_a^{(i)} = p_i + \omega_a^{(i)}, \quad \varepsilon_b^{(i)} = p_i + \sum_{\delta\in\{a,b\}} \omega_\delta^{(i)} \quad \text{and} \quad \varepsilon_c^{(i)} = p_i + \sum_{\delta\in\{a,b,c\}} \omega_\delta^{(i)}.$$

Now for the level-2 list construction the considered level-1 elements have ℓ_2 coordinates with fixed weight $\varepsilon_a^{(1)}$ on which they should add up to a weight-$\varepsilon_a^{(2)}$ vector. Analogously the parameters for level-3 list construction are $\ell_3, \varepsilon_b^{(2)}, \varepsilon_b^{(3)}$ and for final list construction $\ell_4, \varepsilon_c^{(3)}, \varepsilon_c^{(4)}$, where $\omega_\delta^{(4)} := \omega_\delta$.

Overall this leads to complexities of the list construction steps of

$$T_2 = \mathcal{N}^{\ell_2, \varepsilon_a^{(1)}, \varepsilon_a^{(2)}}_{\mathcal{L}_1, \ell_b, \omega_b^{(2)}}, \quad T_3 = \mathcal{N}^{\ell_3, \varepsilon_b^{(2)}, \varepsilon_b^{(3)}}_{\mathcal{L}_2, \ell_c, \omega_c^{(3)}} \quad \text{and} \quad T_4 = \mathcal{N}^{\ell_4, \varepsilon_c^{(3)}, \varepsilon_c^{(4)}}_{\mathcal{L}_3, \ell', \omega'}.$$

To finally derive a numerical estimate for the time complexity we need a complexity formula for $\mathcal{F}_*$ in T_{MO^+} (Equation (10)), which is the time for solving the fixed weight nearest neighbor problem.

4.3 Solving the Fixed Weight Nearest Neighbor Problem

Note that the May-Ozerov nearest neighbor search is still applicable to the fixed weight nearest neighbor problem, even though, its time complexity changes. However, in [9] the corresponding analysis is performed concluding that the algorithm is not well suited for fixed weight input lists. Therefore we develop in the following an Indyk-Motwani [12] inspired algorithm for solving this fixed weight variant, which achieves a better performance. This algorithm then allows us to derive a formula for $\mathcal{F}_*$ in Equation (10).

Similar to the Indyk-Motwani locality-senstive hashing, our algorithm relies on the fact that for $\mathbf{x}, \mathbf{y} \in L_1, L_2$ with $\text{wt}(\mathbf{x}+\mathbf{y}) = \omega_2$ a projection to arbitrary coordinates of $\mathbf{x}+\mathbf{y}$ is more likely to be zero for small ω_2.

The algorithm samples in each iteration a random subset $I \subset \{1, \dots, \ell\}$ of size $|I| = \alpha$ and hopes that the projection of $\mathbf{z} = \mathbf{x}+\mathbf{y}$ to the coordinates indexed by I is zero, i.e., that $\mathbf{z}_I = \mathbf{0}$ or equivalently $\mathbf{x}_I = \mathbf{y}_I$. Then all elements $\mathbf{x}', \mathbf{y}' \in L_1, L_2$ with $\mathbf{x}'_I = \mathbf{y}'_I$ are constructed and for each it is checked if $\text{wt}(\mathbf{x}'+\mathbf{y}') = \omega_2$. The pseudocode of the algorithm is given by Algorithm 5.

Algorithm 5: INDYK-MOTWANI

Input : integer $\omega_1, \omega_2 \leq \ell$, lists $L_1, L_2 \in \mathcal{B}(\ell, \omega_1)^*$
Output : all $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ with $\text{wt}(\mathbf{x}+\mathbf{y}) = \omega_2$

set α as in Equation (13), $N := \frac{\binom{\ell}{\alpha}}{\binom{\ell-\omega_2}{\alpha}}$
for $i = 1$ ***to*** N **do**
  choose random $I \subseteq \{1, \dots, \ell\}$ with $|I| = \alpha$
  for $(\mathbf{x}, \mathbf{y}) \in \{(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2 \mid \mathbf{x}_I = \mathbf{y}_I\}$ **do**
    if $\text{wt}(\mathbf{x}+\mathbf{y}) = \omega_2$ **then**
      $L \leftarrow L \cup (\mathbf{x}, \mathbf{y})$
return L

Analysis of Algorithm 5. The probability that for a searched pair $\mathbf{x}, \mathbf{y}$, with $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \omega_2$ a random projection to α coordinates of $\mathbf{x}+\mathbf{y}$ is zero is

$$q_{\omega_2} = \Pr_I\left[\mathbf{x}_I = \mathbf{y}_I \mid I \subseteq \{1,\ldots,\ell\}, |I| = \alpha, \mathrm{wt}(\mathbf{x}+\mathbf{y}) = \omega_2\right] = \frac{\binom{\ell-\omega_2}{\alpha}}{\binom{\ell}{\alpha}}. \quad (11)$$

Hence, after $N := q_{\omega_2}^{-1}$ iterations we expect that for one of the chosen subsets this is the case and we recover $\mathbf{x}, \mathbf{y}$.

The time complexity is the number of iterations times the cost for finding the matching elements with $\mathbf{x}_I = \mathbf{y}_I$. The construction of list L can be done via a sort-and-match procedure in time linear in $|L_1|, |L_2|$ and $|L|$. The expected size of L is

$$\mathbb{E}\big[|L|\big] = |L_1 \times L_2| \cdot \underbrace{\Pr_I\left[\mathbf{x}_I = \mathbf{y}_I \mid I \subseteq \{1,\ldots,\ell\}, |I| = \alpha, \mathrm{wt}(\mathbf{x}) = \mathrm{wt}(\mathbf{y}) = \omega_1\right]}_{=:q_{\omega_1}}.$$

In following we derive an upper bound for q_{ω_1}, which gives an upper bound on the expected size of L. The probability that two weight-ω_1 vectors sum to a weight-x vector is

$$\frac{\binom{\ell}{x}\binom{x}{x/2}\binom{\ell-x}{\omega_1-x/2}}{\binom{\ell}{\omega_1}^2}.$$

Estimating the binomial coefficients via Equation (1) and setting x to its expectation $x' := (1-\frac{\omega_1}{\ell})\omega_1$, yields a probability of $\tilde{\Theta}(1)$. This implies that two weight-ω_1 vectors add to a weight-x' vector with inverse polynomial probability. Therefore, let us assume that all vectors in $|L_1 \times L_2|$ sum to weight x', which leads at most to a polynomial deviation. Then we can determine q_{ω_1} as

$$q_{\omega_1} = \frac{\binom{\ell-x'}{\alpha}}{\binom{\ell}{\alpha}} = \frac{(\ell-x')(\ell-x'-1)\cdots(\ell-x'-\alpha+1)}{\ell(\ell-1)\cdots(\ell-\alpha+1)} \leq \left(1-\frac{x'}{\ell}\right)^{\alpha}$$

Eventually, this leads to a time complexity of

$$\begin{aligned} T &= \tilde{\mathcal{O}}\left(N \cdot \max(|L_i|, |L|)\right) \\ &= \tilde{\mathcal{O}}\left(\frac{\binom{\ell}{\alpha} \cdot \max\left(|L_i|, |L_i|^2 \cdot (1-x'/\ell)^{\alpha}\right)}{\binom{\ell-\omega}{\alpha}}\right), \end{aligned} \quad (12)$$

where $x' := (1-\frac{\omega_1}{\ell})\omega_1$. Further analysis shows that a choice of

$$\alpha = \ell \cdot \min\left(1 - \frac{\hat{\omega}_2}{2\hat{\omega}_1(1-\hat{\omega}_1)}, -\frac{\log|L_1|}{\log(2\hat{\omega}_1^2 - 2\hat{\omega}_1 + 1)}\right), \quad (13)$$

minimizes the running time, where $\hat{\omega}_1 := \omega_1/\ell$ and $\hat{\omega}_2 := \omega_2/\ell$.

5 Complexity Results for Decoding

In this section we give the results of the numerical optimization of the worst-case time complexity of our improved decoding algorithm from the previous section. Further, we compare its performance against the three latest ISD improvements, which are the May-Ozerov algorithm [16], the BJMM with nearest neighbor algorithm (BJMM-MO) [3] and the Both-May algorithm [4] for different rates. We also give a comparison to the recently proposed RLPN decoding procedure [5]. We then conclude with possible future work and some remarks on further improvements.

Numerical Optimization of Both-May$^+$. We again follow the same optimization methodology outlined at the end of Section 3.1, where we again impose the correctness constraints $q_i \mathcal{R}_i = 1$. We find the worst case rate for Both-May$^+$ at $\hat{k} = 0.43$ yielding $\omega = H^{-1}(1 - \hat{k}) \approx 0.1346$ with a time and memory complexity of

$$T = 2^{0.0948n} \quad \text{and} \quad M = 2^{0.071n},$$

for optimal parameters[6]

$$\begin{aligned}
\hat{p} &= 0.04514, & \hat{p}_3 &= 0.03919, & \hat{p}_2 &= 0.02469, & \hat{p}_1 &= 0.01235,\\
\hat{\ell}_a &= 0.02625, & \hat{\ell}_b &= 0.06893, & \hat{\ell}_c &= 0.15672, & &\\
\hat{\omega}_a &= 0.00277, & \hat{\omega}_a^{(3)} &= 0.00241, & \hat{\omega}_a^{(2)} &= 0.00153, & \hat{\omega}_a^{(1)} &= 0.00076,\\
\hat{\omega}_b &= 0.00716, & \hat{\omega}_b^{(3)} &= 0.00619, & \hat{\omega}_b^{(2)} &= 0.06511, & &\\
\hat{\omega}_c &= 0.02058, & \hat{\omega}_c^{(3)} &= 0.13524. & & & &
\end{aligned}$$

This improves upon the Both-May algorithm with time complexity $2^{0.0951n}$. Note while this improvement seems rather small, we find that our algorithm improves over Both-May especially for smaller rates. In this regime it even yields larger improvements than previous works [3,4], as we show in the following. Furthermore we again improve the memory complexity for the worst case rate from $2^{0.076n}$ down-to $2^{0.071n}$ with even higher gains for smaller rates.

Comparison to Previous ISD Improvements. In Figure 5 we compare the time exponent c of the running time 2^{cn} of different ISD algorithms for various rates. We observe that for both, time and memory, we obtain our largest improvements for small rates. Especially, the memory complexity, which has found to be a bottleneck in practical implementations [10], is reduced significantly by our new algorithm. We also obtain a noteable decrease in time, comparable if not higher than those of previous improvements.

[6] Due to rounding to a precision of 10^{-5} there might be a certain deviation in satisfying the correctness constraints. For the exact numbers we refer to our optimization scripts, which are provided as supplementary material.

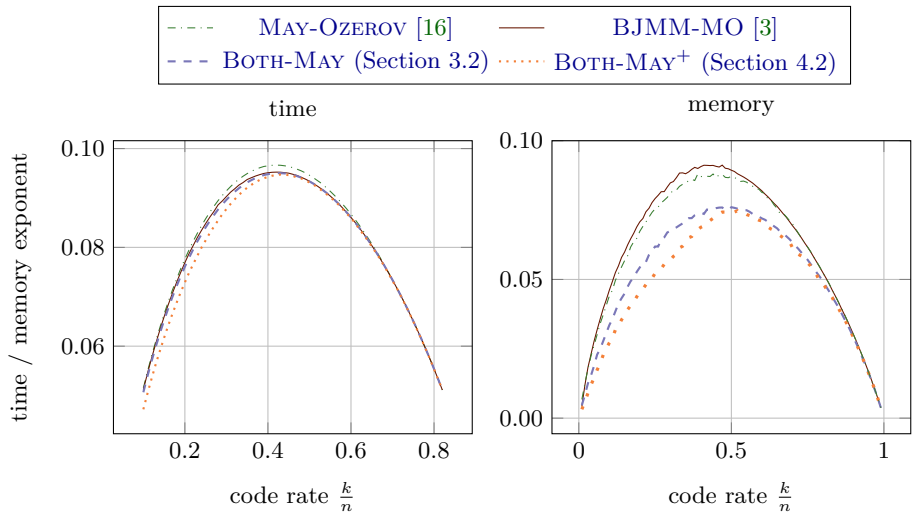

Fig. 5: Comparison between the time (left) and memory (right) exponent of our algorithm against the last three improvements on ISD algorithms in the full distance setting.

Let us compare the exponent reduction achieved by latest ISD improvements over their predecessor. Figure 6 shows the exponent difference $c_p - c$, where the running time (resp. the memory) of the ISD algorithm is 2^{cn}, while $2^{c_p n}$ for its predecessor.[7]

We observe that our algorithm achieves by far the largest running time reduction for small rates. The BJMM-MO algorithm [3] achieves improvements for rates more centered around the worst-case rate. We observe that BJMM-MO achieves its time improvement by investing more memory than its predecessor, which is the May-Ozerov algorithm [16], indicated by the negative values of the memory exponent difference. In terms of memory the Both-May algorithm achieves the largest improvement. However, our improvement is still significant and the second largest memory improvement made by any ISD algorithm (that improves on time) over its predecessor. Note that this even holds if considering all ISD improvements made since Prange's original algorithm in 1962.

Comparison to recently proposed RLPN technique. Recently Carrier et al. [5] proposed a new technique, which reduces the decoding problem to the Learning Parity with Noise (LPN) problem. It then gathers several LPN samples until it is able to solve the LPN instance via majority vote (or more precisely via the fast Fourier transform).

This new technique improved over existing ISD algorithms for small rates $\hat{k} \leq 0.3$. If considering our new algorithm this break-even point, where the RLPN

[7] The chronological order of the latest four improvements is (old to new): May-Ozerov [16], BJMM-MO [3], Both-May [4] (Section 3.2), Both-May+ (Section 4.2).

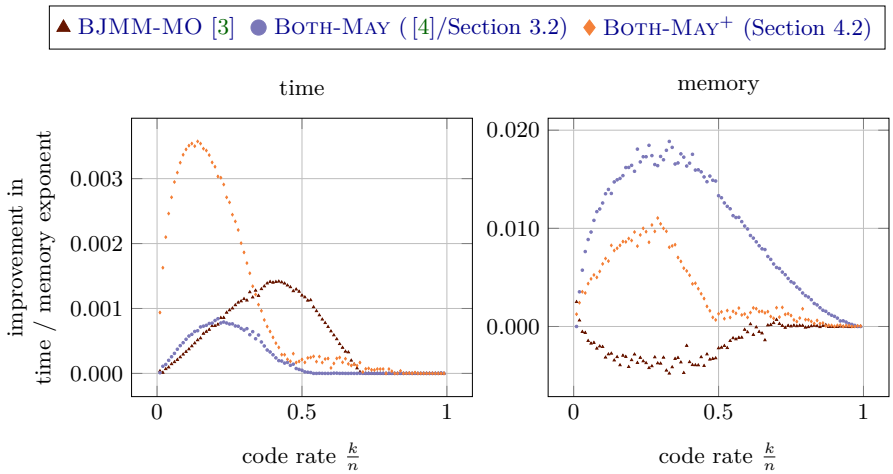

Fig. 6: Improvement in time (left) and memory exponent (right) of ISD algorithms over their predecessor in the full distance setting.

technique becomes preferable is shifted to $\hat{k} \leq 0.25$. In Figure 7 (on the left) we visualize the time and memory exponent difference of the RLPN and our Both-May$^+$ algorithm. We observe that the time improvement for small rates comes along with a significant increase in memory complexity. For higher rates the memory complexity of the RLPN technique drops until it is even lower than the one of our algorithm. However, on the right of Figure 7 we illustrate the exponent difference if we restrict the numerical optimization of the Both-May$^+$ algorithm to parameters which yield a smaller (or equal) memory complexity than the RLPN technique. We observe that still for all rates $\hat{k} > 0.25$ our algorithm obtains time improvements by using less or equal memory than the RLPN technique.

Eventually note that even if RLPN is preferable for small rates $\hat{k} \leq 0.25$, the algorithm generates the needed LPN samples by executing an ISD algorithm as a subroutine. Hence, ISD improvements remain substantial also for improving the RLPN decoder.

Some remarks on future work and further improvements Let us first outline two unsuccessful strategies we explored after which we outline further research directions.

First, our optimization of the algorithm in depth-5 did neither yield time nor memory improvements. Additionally, we explored the strategy of splitting the permutation step across the different levels of the tree, inspired by recently introduced time-memory trade-offs. Here we wanted to exploit the fact that the correct weight distribution on all blocks is not required at every level of the tree. Precisely the first level only requires the correct distribution on block a, the second on block a and b and so forth. This allows to repeat the construction of different levels differently many times (based on the already existing lists of the

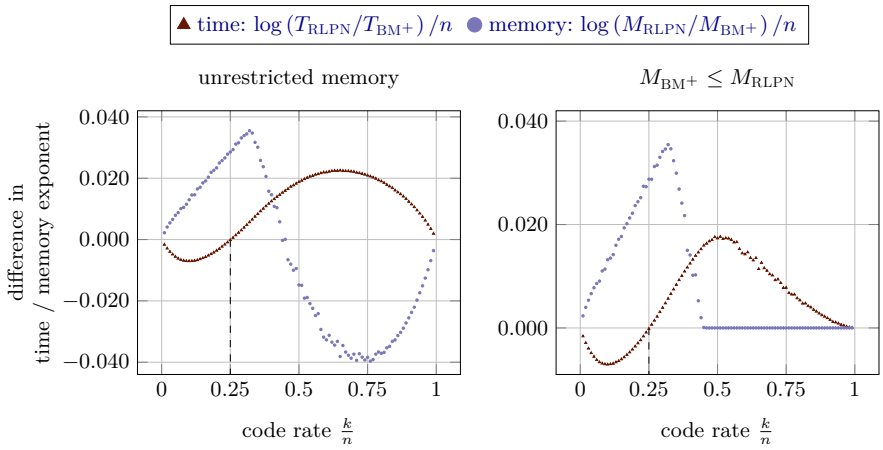

Fig. 7: Difference in time and memory exponent of Both-May$^+$ and the RLPN decoder from [5]. On the right numerical optimization for the Both-May$^+$ was performed by bounding its memory by the memory of the RLPN decoder.

previous level). However, even after re-balancing the cost for the construction of all levels we were not able to obtain time or memory improvements.

We introduced in our work a new technique to combine the so far separately treated list construction (nearest neighbor search) and filtering step of ISD algorithms. Therefore we defined the fixed-weight nearest neighbor problem (see Definition 4.1). Any progress on algorithms for solving this problem is likely to lead to further improvements on our Both-May$^+$ algorithm. Furthermore, we did not get completely rid of the filtering by only ensuring the correct joint weight distribution over all blocks, but not the correct weight in individual blocks. We pose it as an open question if a further refinement of the algorithm allows for time improvements by merging list construction and filtering step completely.

References

1. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_31
2. Bernstein, D.J., Lange, T., Peters, C.: Smaller decoding exponents: Ball-collision decoding. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 743–760. Springer, Heidelberg (Aug 2011). https://doi.org/10.1007/978-3-642-22792-9_42
3. Both, L., May, A.: Optimizing bjmm with nearest neighbors: full decoding in 22/21n and mceliece security. In: WCC workshop on coding and cryptography. p. 214 (2017)
4. Both, L., May, A.: Decoding linear codes with high error rate and its impact for lpn security. In: International Conference on Post-Quantum Cryptography. pp. 25–46. Springer (2018)

5. Carrier, K., Debris-Alazard, T., Meyer-Hilfiger, C., Tillich, J.P.: Statistical decoding 2.0: Reducing decoding to lpn. arXiv preprint arXiv:2208.02201 (2022)
6. Chou, T., Cid, C., UiB, S., Gilcher, J., Lange, T., Maram, V., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., et al.: Classic McEliece: conservative code-based cryptography 10 october 2020 (2020)
7. Dumer, I.: On minimum distance decoding of linear codes. In: Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory. pp. 50–52 (1991)
8. Esser, A., Bellini, E.: Syndrome decoding estimator. In: Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography. Lecture Notes in Computer Science, vol. 13177, pp. 112–141. Springer (2022)
9. Esser, A., Kübler, R., Zweydinger, F.: A faster algorithm for finding closest pairs in hamming metric. In: 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2021)
10. Esser, A., May, A., Zweydinger, F.: McEliece needs a break - solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 433–457. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_16
11. Gilbert, E.N.: A comparison of signalling alphabets. The Bell system technical journal **31**(3), 504–522 (1952)
12. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: 30th ACM STOC. pp. 604–613. ACM Press (May 1998). https://doi.org/10.1145/276698.276876
13. Kirshanova, E., Laarhoven, T.: Lower bounds on lattice sieving and information set decoding. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 791–820. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_27
14. Leon, J.S.: A probabilistic algorithm for computing minimum weights of large error-correcting codes. IEEE Transactions on Information Theory **34**(5), 1354–1359 (1988)
15. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_6
16. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 203–228. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_9
17. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
18. Stern, J.: A method for finding codewords of small weight. In: International Colloquium on Coding Theory and Applications. pp. 106–113. Springer (1988)
19. Torres, R.C., Sendrier, N.: Analysis of information set decoding for a sub-linear error weight. In: Post-Quantum Cryptography. pp. 144–161. Springer (2016)
20. Varshamov, R.R.: Estimate of the number of signals in error correcting codes. Docklady Akad. Nauk, SSSR **117**, 739–741 (1957)

A Details on flaw in original Both-May analysis

In [4] Both and May decide to calculate the expected list size on level i based on the probability that a pair of level-$(i-1)$ elements advances to level i. Let us denote this probability by ϕ_i. Then the expected list size on level i is equal to $L_i = (L_{i-1})^2 \cdot \phi_i$. However, instead Both and May take $L_i = \binom{k}{p_i} \cdot \phi_i$. Note that the square of level-$(i-1)$ lists is usually larger than the number of possible elements with weight p_i, as only an exponential small fraction sums to weight-p_i vectors (making the filtering step effective). In turn, the expected list size is underestimated in the original work.