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. In this context we confirm the result by Carrier et al. that the initial analysis is flawed and conclude with the same runtime exponent. Our work aims at fully substantiating the corrected runtime exponent by a detailed analysis. Furthermore, we show that the Both-May algorithm significantly improves on memory complexity over previous algorithms. Our first main contribution is therefore to give the correct perspective on the significance of the Both-May algorithm and to clarify any remaining doubts on the corrected baseline.
As a second main contribution we detail a possible strategy for future improvements of the Both-May algorithm. This strategy is based on 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.
Even though, our current analysis does not yet yield a gain in complexity over the Both-May algorithm, we point out different ways to further improve the proposed technique.

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 [Pra62]. Since then there have been numerous improvements on Prange's algorithm [Ste88, Leo88, Dum91, BLP11, MMT11, BJMM12, MO15, BM17, BM18], 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 ISD improvements vanishes for a sublinear choice of the error weight [TS16]. And yet, the best known algorithms for attacking those code-based schemes are 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 cryptographic setting. Just recently the ISD algorithms by May-Meurer-Thomae (MMT) [MMT11] and the one by Becker-Joux-May-Meurer (BJMM) [BJMM12], both initially studied and proposed in the constant weight regime, were used to obtain new computational records in the cryptographic setting [EMZ22]. In their work, Esser, May and Zweydinger [EMZ22] 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 [EMZ22, EB22, CCU+20]. 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 [EZ22], 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 [MO15, BM17, BM18]. The fastest of these algorithms by Both-May [BM18] claims significant improvements on the time and memory complexity of previous proposals. However, in a recent work, Carrier, Debris-Alazard, Meyer-Hilfiger and Tillich [CDMT22] challenge the result of

Both-May, by pointing out a flaw in the analysis of its time complexity. Note that (a later revision of) [CDMT22] includes the corrected time complexity exponent together with a correction of necessary parts of the original analysis. However, considering the significance of the Both-May algorithm a self-contained corrected analysis providing full details is of major importance for the field. Note that the significance of the Both-May algorithm stems from the fact that the corrected time exponent still slightly improves previous ISD, and the algorithm hence remains the baseline for new improvements.

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. [CDMT22], which achieves runtime improvements over ISD in some regimes. In this work we clarify any left doubts by giving a corrected and simplified analysis of the Both-May algorithm confirming the exponent stated in the revision of [CDMT22]. Furthermore, our analysis also reveals *significant* gains in the memory complexity of the Both-May algorithm over previous works, contradicting the perception that the algorithm after all constitutes only as a slight improvement over previous ISD. Overall, this result is in line with the results from Esser and Bellini [EB22] who performed a more practical study of the algorithm also observing mostly memory rather than time improvements.

Furthermore, we extend the algorithm by Both-May detailing a possible strategy for future improvements. Our idea combines 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-blackbox 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 observation of Carrier et al. [CDMT22] that the initial analysis of the algorithm is flawed and confirm the corrected runtime exponent (delivered previously in a revision of [CDMT22]). Concentrating solely on the minor improvement in the runtime exponent could give the impression that the Both-May algorithm and with it the broader research on extensive nearest neighbor search in the ISD context are of low significance. However, in our analysis we find that the Both-May algorithm significantly lowers the memory consumption of previous ISD algorithms. Considering the importance of the memory-usage observed by multiple recent works, this strongly supports the significance of the algorithm and, more broadly, its research field.

More precisely, we confirm that the Both-May algorithm reduces the worst-case runtime in the full distance decoding setting from $2^{0.0953n}$ down-to $2^{0.0951n}$. On the other hand, we observe that 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 lies in detailing a possible strategy for future improvements of the algorithm by Both-May. Our strategy is a novel combination of the nearest neighbor search and a subsequently applied filtering step. We therefore treat the nearest neighbor search in a non-blackbox 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.

We note that our current analysis does not yield an improvement in the time or memory complexity of the Both-May algorithm. However, we also outline a straightforward way to further improve our construction. The limitation of this technique lies in the switch to a version of the fixed-weight nearest neighbor problem with non-uniform input distributions which would require significantly more work in the analysis.

Furthermore, since our introduction of the fixed-weight nearest neighbor problem it already found other applications in the ISD context as for example in the recent SievingISD algorithm by Guo, Johansson and Nguyen [GJN23]. This initiated study might lead to future improvements on algorithms for solving fixed-weight nearest neighbor problem and in turn lead to an improved decoding procedure via our construction.

Additionally, new research directions for ISD improvements are especially desirable where recent results by Kirshanova and Laarhoven [KL21] rule out significant speedups of ISD algorithms via generic improvements of nearest neighbor search techniques.

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

Remark 1.1 (Updated Version). A previous version of this article claimed an improvement over the Both-May algorithm by leveraging the introduced technique. However, the analysis disregarded a non-uniform distribution of the considered input lists. The current article is a revised version correcting the previous claims.

Outline. In Section 2 we cover necessary basics on nearest neighbor search, the syndrome decoding problem and the general technique of ISD. Subsequently, in Section 3 we recall the Both-May algorithm and give a corrected analysis. Eventually, we provide in Section 4 our strategy for combining the nearest neighbor search and the filtering step, show how to embed it into the decoding procedure and outline future directions.

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 [MO15] 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 [EKZ21], which generalizes May-Ozerov's result to arbitrary list sizes and distances. The following lemma (compare to [EKZ21, Theorem 1]) states the time complexity of the algorithm

Lemma 2.1 (May-Ozerov Nearest Neighbor [MO15, EKZ21]). *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*

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

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 $d = H^{-1}(1 - k/n)n$ [Gil52, Var57]. 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 $\mathbf{e}_2$ 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 1. 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.

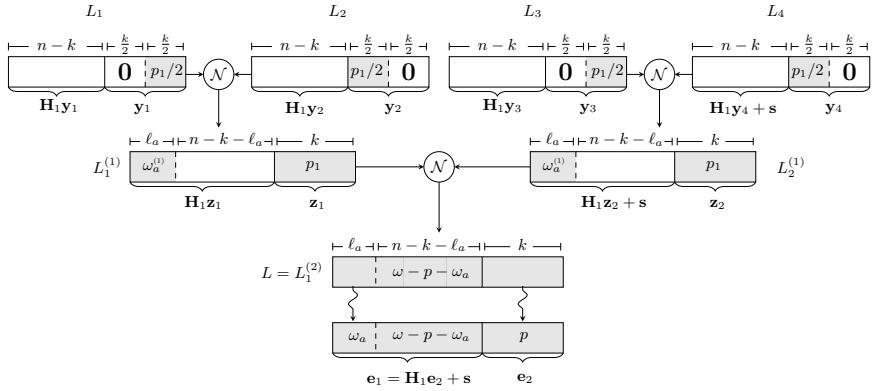

Fig. 1: Both-May algorithm in depth-2. Weight in gray regions differs from weight of uniformly random vectors. Numbers inside gray areas indicate regions of fixed weight. Curly arrows illustrate final check for contained solution, $\mathcal{N}$ indicates nearest neighbor search.

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 $\omega_a^{(1)}$.[1] 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

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

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

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

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

[1] 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.

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 $\mathbf{P}(\mathbf{H}_1\mathbf{e}_2 + \mathbf{s}, \mathbf{e}_2)$

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} with constant c.

To actually find the values of the $\hat{o}_i, \hat{k}$ and eventually c we use a numerical optimization tool provided by the *python* library *scipy*. The way we access this library is inspired by a code of Bonnetain et al. [BBSS20].[2] In general it is possible that such optimizers do not output a global minimum but instead run into some local minimum. However, to increase the confidence in the found optimum we ran the optimization thousands of times with random starting points, until no further improvement could be made.

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

We stress that these results essentially match those given in the original work of Both-May [BM18]. 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 [BM18] 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, $j = 1, \ldots, 16$.

[2] This code is accessible at https://github.com/xbonnetain/optimization-subset-sum.

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 2).

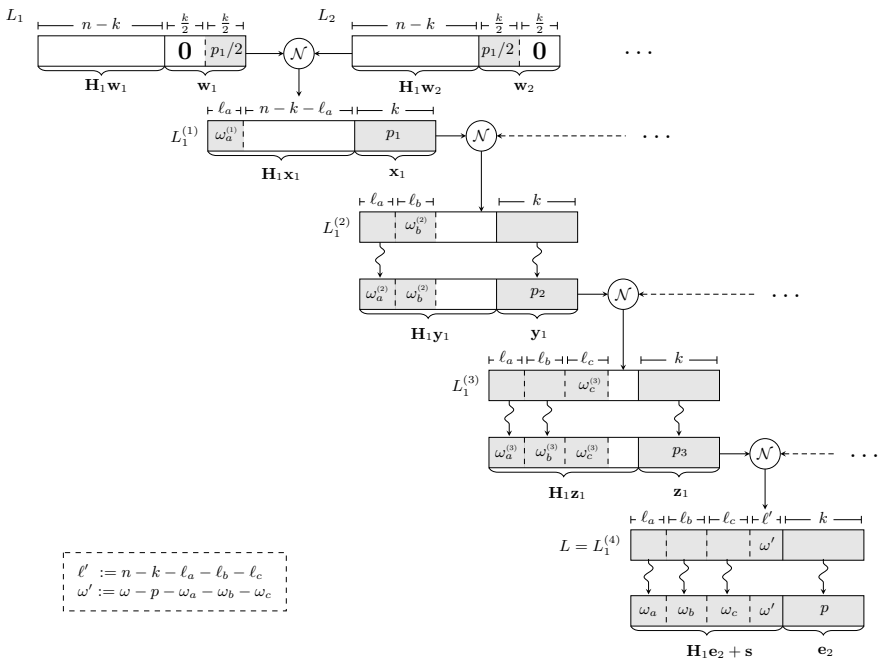

Fig. 2: Leftmost path of depth-4 algorithm from leaves (base lists) to root (final list). Gray areas indicate regions where weight differs from weight of uniformly random vectors. Numbers inside gray areas indicate regions of fixed weight. Curly arrows illustrate filtering process, $\mathcal{N}$ indicates nearest neighbor search.

The pseudocode of the algorithm is given by Algorithm 2 and an illustration in Figure 2. 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 there 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 [BM18] 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 [BM18].

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 $\hat{\omega} = H^{-1}(1-\hat{k}) \approx 0.1384$, 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 $2^{0.0885n}$ [BM18], it still slightly improves on the previously best running time of $2^{0.0953n}$ reported in [BM17]. Further, the memory complexity is drastically improved by a factor of $2^{0.0161n}$ from previously $2^{0.0915n}$ to $2^{0.0754n}$.

In Figure 3 we compare the time and memory exponents of the latest three ISD improvements, which are in chronological order(old to new): May-Ozerov [MO15], BJMM-MO [BM17], Both-May [BM18] (Section 3.2).

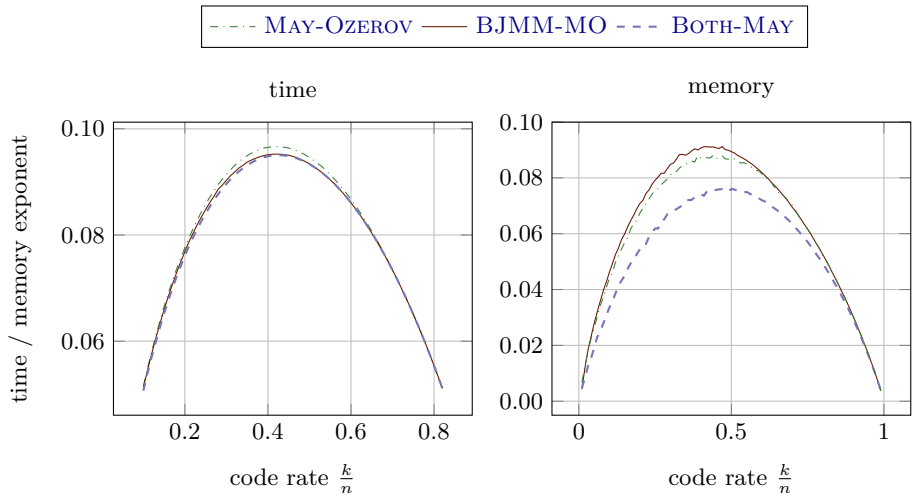

Fig. 3: Comparison between the time (left) and memory (right) exponent of the Both-May, May-Ozerov and BJMM-MO algorithm in the full distance setting.

[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, available at https://github.com/Memphisd/Revisiting-NN-ISD.

4 A Strategy for Future Improvements

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 complexity estimate for the 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 [MO15, Car20, EKZ21]. First, the 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.

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 — individually, no pairs between lists are considered — 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] It is important to note that the algorithm up to the leaf-level only operates on each list individually, i.e., no

[4] In [EKZ21] r was chosen as $\frac{m}{\log^2 m}$ to ease the analysis. However, in [MO15] it was shown that a large enough constant is already sufficient.

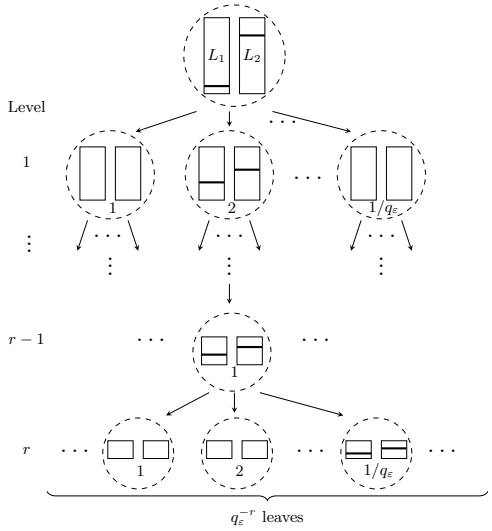

Fig. 4: Illustration of the May-Ozerov nearest neighbor search algorithm. Arrows indicate the application of a locality sensitive filter. Each node branches q_ε^{-1} times. Bold stripes in lists indicate pair of distance ε progressing through all r applied filters.

pairs between the lists are examined. Therefore, lists L_1', L_2' contained in any leaf are subsets of the initial lists L_1, L_2.

The time complexity is then given (compare to [EKZ21, 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 [EKZ21] 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 by computing the product of lists contained in the leaves. The initial factor in front of the maximum accounts for the number of leaves.

Furthermore, for the locality sensitive filter criterion chosen in [EKZ21] the probabilities q_ε and q_{f} are parameterized via an optimization parameter $\delta < m$ and are given as

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

[5] In comparison to [EKZ21] we assume $2^{\lambda m} \cdot q_{\mathrm{f}}^{r-1} \geq 1$ to remove one term from the maximum.

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 $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \varepsilon$

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 $\quad L \leftarrow L \cup \{(\mathbf{x}, \mathbf{y}) \in L_1', L_2' \mid \mathrm{wt}(\mathbf{x}+\mathbf{y}) = \varepsilon\}$
5 **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 $n-m$ 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

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

where $\mathcal{L} = |L_1| \cdot q_\mathrm{f}^r$. 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. $T_{\mathrm{MO}^+} < T_{\mathrm{MO}}$.

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 $\mathrm{wt}(\mathbf{x}_1 + \mathbf{x}_2) = \omega_2$*

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

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

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 2).

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)

$$\begin{aligned} L_i^{(2)+} &= \{\mathbf{y}_i \in L_i^{(2)} \mid \mathrm{wt}(\mathbf{y}_i) = p_2\} \\ L_i^{(3)+} &= \{\mathbf{z}_i \in L_i^{(3)} \mid \mathrm{wt}(\mathbf{z}_i) = p_3\} \\ L^+ &= \{\mathbf{e}_2 \in L \mid \mathrm{wt}(\mathbf{e}_2) = p\ \}. \end{aligned} \tag{11}$$

Note that these lists only correspond to *partly* filtered lists, as 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 $i \geq 2$, $l_{i,j}$ we additionally need 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 $\delta \in \{a, b, c\}$. However, the constructed lists only ensure the desired weight on the $\mathbf{v}_j$ not the $l_{j,i}$. 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.

We call the resulting algorithm that computes the lists from Equation (11) directly BOTH-MAY$^+$ in the following.

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, while the parameters of the fixed-weight variant are given by $(\ell, \omega_1, \omega_2)$. For the construction of the level-2, -3 and final lists the respective choices of $(\ell, \omega_1, \omega_2)$ are $(k, p_1, p_2), (k, p_2, p_3)$ and (k, p_3, p) respectively.

Overall this leads to complexities of the list construction steps of

$$T_2 = \mathcal{N}^{k,p_1,p_2}_{\mathcal{L}_1,\ell_b,\omega_b^{(2)}}, \quad T_3 = \mathcal{N}^{k,p_2,p_3}_{\mathcal{L}_2,\ell_c,\omega_c^{(3)}} \quad \text{and} \quad T_4 = \mathcal{N}^{k,p_3,p}_{\mathcal{L}_3,\ell',\omega'}.$$

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 [EKZ21] 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 [IM98] 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.

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

$$q_{\omega_2} = \Pr_I\left[\mathbf{x}_I = \mathbf{y}_I \mid I \subseteq \{1,\dots,\ell\}, |I| = \alpha, \text{wt}(\mathbf{x}+\mathbf{y}) = \omega_2\right] = \frac{\binom{\ell-\omega_2}{\alpha}}{\binom{\ell}{\alpha}}. \tag{12}$$

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

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 $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \omega_2$

set α as in Equation (14), $N := \frac{\binom{\ell}{\alpha}}{\binom{\ell-\omega_2}{\alpha}}$
for $i = 1$ ***to*** N **do**
  choose random $I \subseteq \{1, \ldots, \ell\}$ 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 $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \omega_2$ **then**
      $L \leftarrow L \cup (\mathbf{x}, \mathbf{y})$
return L

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 $x' := 2(1-\frac{\omega_1}{\ell})\omega_1$, 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

$$\begin{aligned} T &= \tilde{\mathcal{O}}\left(N \cdot \max(|L_i|, |L|)\right) \\ &= \tilde{\mathcal{O}}\left(\frac{\binom{\ell}{\alpha} \cdot \max\left(|L_i|, |L_i|^2 \cdot (1-x'/\ell)^{\alpha}\right)}{\binom{\ell-\omega}{\alpha}}\right), \end{aligned} \tag{13}$$

where $x' := 2(1-\frac{\omega_1}{\ell})\omega_1$. Further analysis shows that a choice of

$$\alpha = \ell \cdot \min\left(\max\left(0, 1-\frac{\hat{\omega}_2}{2\hat{\omega}_1(1-\hat{\omega}_1)}\right), -\frac{\log|L_1|}{\log(2\hat{\omega}_1^2 - 2\hat{\omega}_1 + 1)}\right), \tag{14}$$

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

Note that for $\alpha = 0$ the algorithm simply computes the Cartesian product of both lists to find all ω_2-close pairs. This happens if the target weight ω_2 is larger or equal to its expected value x'.

4.4 Further Improving the Approach

The current version of the algorithm does not yet yield a gain in the time or memory complexity of the decoding procedure.

However, there are several ways to improve the current algorithm that have the potential to lead to an improved decoding routine.

First of all, the current algorithm does also not yet eliminate the need for the the filtering step completely, as detailed in Section 4.2. That is because the constructed lists from Equation (11) only guarantee the correct weight on the vectors $\mathbf{v} \in \mathbb{F}_2^k$ themselves, and not on the corresponding projections $\pi_\delta(\mathbf{Hv}), \delta \in \{a, b, c\}$. Therefore, the algorithm could clearly be improved by constructing the fully filtered lists directly via the adapted nearest neighbor routine. However, this introduces a variant of the fixed-weight nearest neighbor problem with multiple stripes of different given input and output weights. The analysis of Algorithm 5 clearly complicates for such a problem variant, and it is not clear that a straightforward application would be an optimal strategy.

Another direction for improvements are improved algorithms for the fixed-weight nearest neighbor problem. The algorithm we detail here for solving the problem is an adaptation of an early LSH algorithm by Indyk and Motwani. Improvements from the general nearest neighbor case might translate to the fixed-weight setting. Also there might exist entirely different strategies for the fixed-weight setting, such as the approach recently detailed in [GJN23].

Acknowledgements This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project 517817836.
We thank the anonymous reviewers of Crypto '23 for pointing out the flaw in the analysis of a previous version of this work.
Furthermore, we are grateful to the whole team of authors around the publication [CDMT22] for insightful comments and remarks that led to this improved revision. Especially, we would like to express our sincere gratitude to Charles Meyer-Hilfiger for his proactive approach and thoughtful insights which greatly influenced the refinement of our work.

References

BBSS20. Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 633–666. Springer, Heidelberg, December 2020.

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.

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.

BM17. Leif Both and Alexander May. Optimizing bjmm with nearest neighbors: full decoding in 22/21n and mceliece security. In *WCC workshop on coding and cryptography*, page 214, 2017.

BM18. Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for lpn security. In *International Conference on Post-Quantum Cryptography*, pages 25–46. Springer, 2018.

Car20. Kevin Carrier. *Recherche de presque-collisions pour le décodage et la reconnaissance de codes correcteurs.* PhD thesis, Sorbonne université, 2020.

CCU+20. Tung Chou, Carlos Cid, Simula UiB, Jan Gilcher, Tanja Lange, Varun Maram, Rafael Misoczki, Ruben Niederhagen, Kenneth G Paterson, Edoardo Persichetti, et al. Classic McEliece: conservative code-based cryptography 10 october 2020. 2020.

CDMT22. Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. Statistical decoding 2.0: Reducing decoding to lpn. *arXiv preprint arXiv:2208.02201*, 2022.

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

EB22. Andre Esser and Emanuele Bellini. Syndrome decoding estimator. In *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography*, volume 13177 of *Lecture Notes in Computer Science*, pages 112–141. Springer, 2022.

EKZ21. Andre Esser, Robert Kübler, and Floyd Zweydinger. 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.

EMZ22. Andre Esser, Alexander May, and Floyd Zweydinger. McEliece needs a break - solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 433–457. Springer, Heidelberg, May / June 2022.

EZ22. Andre Esser and Floyd Zweydinger. New time-memory trade-offs for subset sum–improving ISD in theory and practice. *Cryptology ePrint Archive*, 2022.

Gil52. Edgar N Gilbert. A comparison of signalling alphabets. *The Bell system technical journal*, 31(3):504–522, 1952.

GJN23. Qian Guo, Thomas Johansson, and Vu Nguyen. A new sieving-style information-set decoding algorithm. *Cryptology ePrint Archive*, 2023.

IM98. Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th ACM STOC*, pages 604–613. ACM Press, May 1998.

KL21. Elena Kirshanova and Thijs Laarhoven. Lower bounds on lattice sieving and information set decoding. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 791–820, Virtual Event, August 2021. Springer, Heidelberg.

Leo88. Jeffrey S Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.

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.

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

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

TS16. Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *Post-Quantum Cryptography*, pages 144–161. Springer, 2016.

Var57. Rom Rubenovich Varshamov. 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 [BM18] 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.