On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes

Alexander May* and Ilya Ozerov**

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

Abstract. We propose a new decoding algorithm for random binary linear codes. The so-called information set decoding algorithm of Prange (1962) achieves worst-case complexity $2^{0.121n}$. In the late 80s, Stern proposed a sort-and-match version for Prange's algorithm, on which all variants of the currently best known decoding algorithms are build. The fastest algorithm of Becker, Joux, May and Meurer (2012) achieves running time $2^{0.102n}$ in the full distance decoding setting and $2^{0.0494n}$ with half (bounded) distance decoding.
In this work we point out that the sort-and-match routine in Stern's algorithm is carried out in a non-optimal way, since the matching is done in a two step manner to realize an *approximate matching* up to a small number of error coordinates. Our observation is that such an approximate matching can be done by a variant of the so-called High Dimensional Nearest Neighbor Problem. Namely, out of two lists with entries from $\mathbb{F}_2^m$ we have to find a pair with closest Hamming distance. We develop a new algorithm for this problem with sub-quadratic complexity which might be of independent interest in other contexts.
Using our algorithm for full distance decoding improves Stern's complexity from $2^{0.117n}$ to $2^{0.114n}$. Since the techniques of Becker et al apply for our algorithm as well, we eventually obtain the fastest decoding algorithm for binary linear codes with complexity $2^{0.097n}$. In the half distance decoding scenario, we obtain a complexity of $2^{0.0473n}$.

Keywords: linear codes, nearest neighbor problem, approximate matching, meet-in-the-middle

1 Introduction

The NP-hard decoding problem for random linear codes is one of the most fundamental combinatorial problems in coding and complexity theory. Due to its purely combinatorial structure it is the major source for constructing cryptographic hardness assumptions that retain their hardness even in the presence of

* Supported by DFG as part of GRK 1817 Ubicrypt and SPP 1736 Big Data
** Supported by DFG as part of SPP 1307 Algorithm Engineering

quantum computers. Almost all code-based cryptosystems, such as the McEliece encryption scheme [15], rely on the fact that random linear codes are hard to decode.

The way cryptographers usually embed a trapdoor into code-based constructions is that they start with some structured code C, which allows for efficient decoding, and then use a linear transformation to obtain a scrambled code C'. The cryptographic transformation has to ensure that the scrambled code C' is indistinguishable from a purely random code. Unless somebody is able to unscramble the code, a cryptanalyst faces the problem of decoding a random linear code. Hence, for choosing appropriate security parameters for a cryptographic scheme it is crucial to know the best performance of generic decoding algorithms for linear codes.

Also closely related to random linear codes is the so-called Learning Parity with Noise Problem (LPN) that is frequently used in cryptography [9, 12]. In LPN, one directly starts with a generator matrix that defines a random linear code C and the LPN search problem is a decoding problem on C. Cryptographers usually prefer decision versions of hard problems for proving security of their schemes. However, for LPN there is a reduction from the decision to the search version that directly links the cryptographic hardness of the underlying schemes to the task of decoding random linear codes.

The Learning with Errors Problem (LWE) that was introduced for cryptographic purposes in the work of Regev [19, 20] can be seen as a generalization of LPN to codes defined over larger fields. LWE is closely related to well-studied problems in learning theory, and it proved to be a fruitful source within the last decade for many cryptographic constructions that provide new functionalities [5, 7, 16]. Although we focus in this work on random linear codes over $\mathbb{F}_2$, we do not see any obstacles in transferring our techniques to larger finite fields $\mathbb{F}_q$ as this was done in [17]. Surely, our techniques will also lead to some improvement for arbitrary fields, but we believe that our improvements are best tailored to $\mathbb{F}_2$, easiest to explain for the binary field, and we feel that binary codes define the most fundamental and widely applied class of linear codes.

Let us define some basics of linear codes and review the progress that decoding algorithms underwent. A (random) binary linear code C is a (random) k-dimensional subspace of $\mathbb{F}_2^n$. Therefore, a code defines a mapping $\mathbb{F}_2^k \to \mathbb{F}_2^n$ that maps a message $\mathbf{m} \in \mathbb{F}_2^k$ to a codeword $\mathbf{c} \in \mathbb{F}_2^n$. On a noisy channel, a receiver gets an erroneous version $\mathbf{x} = \mathbf{c} + \mathbf{e}$ for some error vector $\mathbf{e} \in \mathbb{F}_2^n$. The decoding problem now asks for finding $\mathbf{e}$, which in turn enables to reconstruct $\mathbf{c}$ and $\mathbf{m}$. Usually, we assume that during transmission of $\mathbf{c}$ not too many errors occurred, such that $\mathbf{e}$ has a small Hamming weight $\mathbf{wt}(\mathbf{e})$ and $\mathbf{c}$ is the closest codeword to $\mathbf{x}$. This defines the search space of $\mathbf{e}$, which is closely linked to the distance d of C.

So naturally, the running time $T(n, k, d)$ of a decoding algorithm is a function of all code parameters n, k and d. However, we know that asymptotically random linear codes reach the Gilbert-Varshamov bound $\frac{k}{n} \leq 1 - H(\frac{d}{n})$, where $H(\cdot)$ is the binary entropy function (see e.g. [22]). In the *full distance decoding* setting we

are looking for an error vector $\mathbf{e}$ s.t. $\mathbf{wt}(\mathbf{e}) \leq d$, whereas in *half distance decoding* we have $\mathbf{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$. Thus, in both cases we can upper bound the running time by a function $T(n,k)$ of n and k only. When we speak of worst-case running time, we maximize $T(n,k)$ for all k where the maximum is obtained for code rates $\frac{k}{n}$ near $\frac{1}{2}$. Usually, it suffices to compare worst-case complexities since all known decoding algorithms with running time T, T' and worst-case complexity $T(n) < T'(n)$ satisfy $T(n,k) < T'(n,k)$ for all k.

The simplest algorithm is to enumerate naively over $\mathbf{e}$'s search space and check whether $\mathbf{x}+\mathbf{e} \in C$. However, it was already noticed in 1962 by Prange [18] that the search space for $\mathbf{e}$ can considerably be lowered by applying simple linear algebra. Prange's algorithm consists of an enumeration step with exponential complexity and some Gaussian elimination step with only polynomial complexity. The worst-case complexity of Prange's algorithm is $2^{0.121n}$ in the full distance decoding case and $2^{0.0576n}$ with half distance decoding.

In 1989, Stern [21] noticed that Prange's enumeration step can be accelerated by enumerating two lists $\mathbf{L}, \mathbf{R}$ within half of the search space, a typical time-memory trade-off. The lists $\mathbf{L}, \mathbf{R}$ are then sorted and one looks for a matching pair. This is a standard trick for many combinatorial search problems and is usually called a sort-and-match or Meet-in-the-middle approach in the literature. Stern's algorithm however is unable to directly realize an approximate matching of lists, where one wants to find a pair of vectors from $\mathbf{L} \times \mathbf{R}$ with small Hamming distance. In Stern's algorithm this is solved by a non-optimal two-step approach, where one first matches vector pairs *exactly* on some portion of the coordinates, and then in a second step checks whether any of these pairs has the desired distance on all coordinates. Stern's algorithm led to a running time improvement to $2^{0.117n}$ (full distance) and $2^{0.0557n}$ (half distance), respectively.

Our contribution: In this work, we propose a different type of matching algorithm for Stern's algorithm that directly recovers a pair from $\mathbf{L} \times \mathbf{R}$ with small Hamming distance. Fortunately, this problem is well-known in different variants in many fields of computer science as the High Dimensional Nearest Neighbor Problem [6, 8, 23] or the Bichromatic Closest Pair Problem [2]. The best bounds that are known for the Hamming metric are due to Dubiner [6] and Valiant [24].

However, we were not able to apply Dubiner's algorithm to our decoding problem, since we have a bounded vector size, which is referred to as the *limited amount of data* case in [6]. Although it is stated in Dubiner's work [6] that his algorithm might also be applicable to the *limited* case, the analysis is only done for the *unlimited amount of data* case. The algorithm of Valiant [24] is only optimal in special cases (i.e. large Hamming distances) and unfortunately doesn't apply to our problem either. Thus we decided to give an own algorithmic solution, which gives us the required flexibility for choosing parameters that are tailored to the decoding setting.

We provide a different and quite general algorithm for finding a pair of (limited or unlimited size) vectors in two lists that fulfill some consistency criterion, like e.g. in our case being close in some distance metric. The way we solve this

problem is by checking parts of the vectors locally, and thus sorting out vector pairs that violate consistency locally, since these pairs are highly unlikely to fulfill consistency globally. In our special case, where $|\mathbf{L}| = |\mathbf{R}|$ and where we want to find the closest pair of vectors from $\mathbb{F}_2^m$ with Hamming distance $\gamma m, \gamma \leq \frac{1}{2}$, we obtain an algorithm that approaches sub-quadratic complexity $|\mathbf{L}|^{\frac{1}{1-\gamma}}$. In our analysis in Sections 4 and 5 we propose an algorithm for bounded vector size and show that in the *unlimited amount of data* case (which is not important for the decoding problem) our algorithm approaches Dubiner's bound.

Using this matching algorithm in the full distance decoding setting directly leads to an improved decoding algorithm for random binary linear codes with complexity $2^{0.114n}$, as opposed to Stern's complexity $2^{0.117n}$. In 2011, Stern's algorithm was improved by Bernstein, Lange and Peters to $2^{0.116n}$. Then, using recent techniques from improving subset sum algorithms [11, 3], May, Meurer, Thomae [14] and Becker, Joux, May, Meurer [4] further improved the running time to $2^{0.102n}$. Fortunately, these techniques directly apply to our algorithm as well and result in a new worst-case running time as small as $2^{0.097n}$. We obtain similar results in the case of half distance decoding.

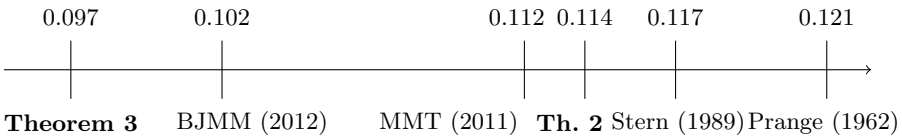

Fig. 1. History of Information Set Decoding: full distance decoding (FDD)

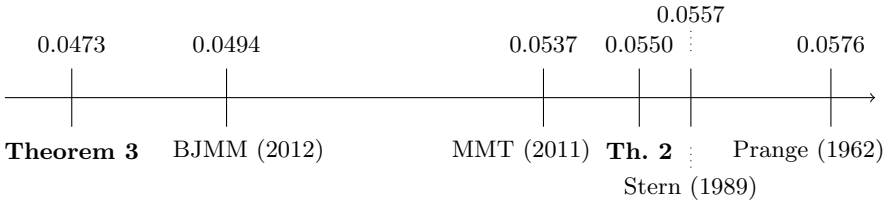

Fig. 2. History of Information Set Decoding: half distance decoding (HDD)

The paper is organized as follows. In Section 2, we elaborate a bit more on previous work and explain the basic idea of our matching approach. This leads to a sort-and-match decoding algorithm that we describe and analyze in Section 3, including the application of the improvement from Becker et al [4]. Our decoding algorithm calls a new matching subroutine that finds a pair of

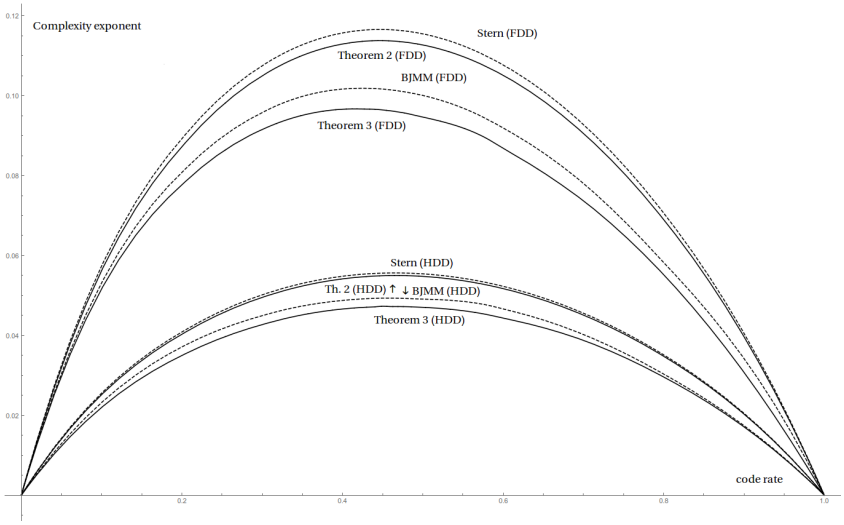

Fig. 3. Stern, Th. 2, BJMM and Th. 3 for BDD/FDD and all code rates k/n

vectors with minimal Hamming distance in two lists. We describe this matching procedure in Section 4.

2 Previous Work – Information Set Decoding

A binary linear code C is a k-dimensional subspace of $\mathbb{F}_2^n$. Thus, C is generated by a matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ that defines a linear mapping $\mathbb{F}_2^k \to \mathbb{F}_2^n$. If $\mathbf{G}$'s entries are chosen independently and uniformly at random from $\mathbb{F}_2^{k \times n}$ with the restriction that $\mathbf{G}$ has rank k, then we call C a random binary linear code. The distance d of C is defined by the minimum Hamming distance of two different codewords in C.

Let $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ be a basis of the kernel of C. If C is random then it is not hard to see that the entries of $\mathbf{H}$ are also independently and uniformly at random distributed in $\mathbb{F}_2$. Therefore, we have $\mathbf{H}\mathbf{c}^t = \mathbf{0}$ for every $\mathbf{c} \in C$. For simplicity, we omit from now on all transposition of vectors and write $\mathbf{c}$ instead of $\mathbf{c}^t$.

For every erroneous codeword $\mathbf{x} = \mathbf{c} + \mathbf{e}$ with error vector $\mathbf{e}$, we obtain $\mathbf{H}\mathbf{x} = \mathbf{H}\mathbf{e}$ by linearity. We call $\mathbf{s} := \mathbf{H}\mathbf{x} \in \mathbb{F}_2^{n-k}$ the *syndrome of* a message $\mathbf{x}$. In order to decode $\mathbf{x}$, it suffices to find a low weight vector $\mathbf{e}$ such that $\mathbf{H}\mathbf{e} = \mathbf{s}$. Once $\mathbf{e}$ is found, we can simply recover $\mathbf{c}$ from $\mathbf{x}$. This process is called syndrome decoding, and the problem is known to be NP-hard.

In the case of *full distance decoding* (FDD), we receive an arbitrary point $\mathbf{x} \in \mathbb{F}_2^n$ and want to decode to the closest codeword in the Hamming metric. We want to argue that there is always a codeword within (roughly) Hamming

distance d. Therefore, we observe that the syndrome equation $\mathbf{He} = \mathbf{s}$ is solvable as long as the search space for $\mathbf{e}$ roughly equals 2^{n-k}. Hence, for weight-d vectors $\mathbf{e} \in \mathbb{F}_2^n$ we obtain $\binom{n}{d} \approx 2^{H(\frac{d}{n})n} \approx 2^{n-k}$, where $H(\cdot)$ is the binary entropy function. This implies $H(\frac{d}{n}) \approx 1 - \frac{k}{n}$, a relation that is known as the Gilbert-Varshamov bound. Moreover, it is well-known that random codes asymptotically reach the Gilbert-Varshamov bound [22]. This implies that for every $\mathbf{x}$ we can always expect to find a closest codeword within distance d.

In the other case of *half distance decoding* (HDD), we obtain the promise that the error vector is within the error correction distance, i.e. $\mathbf{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$, which is for example the case in cryptographic settings, where an error is added artificially by e.g. the encryption of a message.

Since the decoding algorithms that we study use $\omega := \mathbf{wt}(\mathbf{e})$ as an input, we run the algorithms in the range $\omega \in [0, d]$ or $\omega \in [0, \lfloor \frac{d-1}{2} \rfloor]$, respectively. However, all our algorithms attain their maximum running time for their maximal weight $\omega = d$, respectively $\omega = \lfloor \frac{d-1}{2} \rfloor$. In the following we assume that we know ω.

Let us return to our syndrome decoding problem $\mathbf{He} = \mathbf{s}$. Naively, one can solve this equation by simply enumerating all weight-ω vectors $\mathbf{e} \in \mathbb{F}_2^n$ in time $\widetilde{\mathcal{O}}\left(\binom{n}{\omega}\right)$.

Prange's Information Set decoding: At the beginning of the 60s, Prange showed that the use of linear algebra provides a significant speedup. Notice that we can simply reorder the positions of the error vector $\mathbf{e}$ by permuting the columns of $\mathbf{H}$. For some column permutation π let $\mathbf{Q} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ denote the quadratic matrix at the right hand side of $\pi(\mathbf{H}) = (\cdot||\mathbf{Q})$. Assume that $\mathbf{Q}$ has full rank, which happens with constant probability and define $\bar{\mathbf{s}} = \mathbf{Q}^{-1} \cdot \mathbf{s}$ and $\bar{\mathbf{H}} = \mathbf{Q}^{-1} \cdot \pi(\mathbf{H}) = (\cdot|\mathbf{I})$ for an $(n-k) \times (n-k)$ identity matrix $\mathbf{I}$. Let $\pi(\mathbf{e}) = \mathbf{e}_1 + (0^k||\mathbf{e}_q)$ with $\mathbf{e}_1 \in \mathbb{F}_2^k \times 0^{n-k}$ and $\mathbf{e}_q \in \mathbb{F}_2^{n-k}$ be the permuted error vector. Assume that $\mathbf{e}_1 = 0^n$. In this case, we call the first k error-free coordinates an *information set*. Having an information set, we can rewrite our syndrome equation as

$$\bar{\mathbf{H}}\pi(\mathbf{e}) = \bar{\mathbf{H}}\mathbf{e}_1 + \mathbf{e}_q = \bar{\mathbf{s}}, \text{ where } \mathbf{wt}(\mathbf{e}_1) = 0 \text{ and } \mathbf{wt}(\mathbf{e}_q) = \omega.$$

Since $\mathbf{e}_1$ is the zero vector, we can simplify as $\mathbf{e}_q = \bar{\mathbf{s}}$. Thus we only have to check whether $\bar{\mathbf{s}}$ has the correct weight $\mathbf{wt}(\bar{\mathbf{s}}) = \omega$.

Notice that all complexity in Prange's algorithm is moved to the initial permutation π of $\mathbf{H}$'s columns, whereas the remaining step has polynomial complexity. To improve upon the running time, it is reasonable to lower the restriction that the information set has no 1-entries in $\mathbf{e}_1$, which was done in the work of Lee-Brickell [13]. Assume that the information set carries exactly p 1-positions. Then we enumerate over all $\binom{k}{p}$ possible $\mathbf{e}_1 \in \mathbb{F}_2^k \times 0^{n-k}$ with weight p. Therefore, we can test whether $\mathbf{wt}(\mathbf{e}_q) = \mathbf{wt}(\bar{\mathbf{s}} - \bar{\mathbf{H}}\mathbf{e}_1) = \omega - p$. On the downside, this trade-off between lowering the complexity for finding a good π and enumerating weight-p vectors does not pay off. Namely, asymptotically (in n) the trade-off achieves its optimum for $p = 0$.

On the positive side, we can improve on the simple enumeration of weight-p vectors by enumerating two lists of weight-$\frac{p}{2}$ vectors[1]. This classical time-memory trade-off, usually called a Meet-in-the-middle approach, allows to reduce the enumeration time from $\binom{k}{p}$ to $\binom{k/2}{p/2}$ by increasing the memory to the same amount. Such a Meet-in-the-middle approach was introduced by Stern for Prange's Information Set decoding in 1989 [18]. In a nutshell, Stern's variant splits the first k columns of $\pi(\mathbf{e}) = \mathbf{e}_1 + \mathbf{e}_2 + (0^k||\mathbf{e}_q)$ with $\mathbf{e}_q \in \mathbb{F}_2^{n-k}$ in two parts $\mathbf{e}_1 \in \mathbb{F}_2^{k/2} \times 0^{k/2} \times 0^{n-k}$ and $\mathbf{e}_2 \in 0^{k/2} \times \mathbb{F}_2^{k/2} \times 0^{n-k}$. Additionally, we want a *good* permutation π to achieve

$$\bar{\mathbf{H}} \cdot (\mathbf{e}_1 + \mathbf{e}_2) + \mathbf{e}_q = \bar{\mathbf{s}} \text{ with } \mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p \text{ and } \mathbf{wt}(\mathbf{e}_q) = \omega - p. \tag{1}$$

Thus we have

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

Since $\mathbf{wt}(\mathbf{e}_q) = \omega - p$, *for all but* $\omega - p$ of the $n - k$ coordinates of the vectors we have

$$\bar{\mathbf{H}}\mathbf{e}_1 = \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}. \tag{3}$$

Remember that $\mathbf{Q}$ was defined as the right hand part of $\pi(\mathbf{H})$. It can be shown that $\mathbf{Q}$ is invertible with constant probability over the choice of $\mathbf{H}$ and π. Thus inverting $\mathbf{Q}$ can be ignored for the computation of the time complexity that suppresses polynomial factors.

Definition 1. *A permutation π is* good *if $\mathbf{Q}$ is invertible and if for $\pi(\mathbf{e})$ there exists a solution $(\mathbf{e}_1^*, \mathbf{e}_2^*)$ satisfying (1).*

In Stern's algorithm, one computes for every candidate $\mathbf{e}_1$ the left-hand side of Eq. (3) and stores the result in a sorted list $\mathbf{L}$. Then, one computes for every $\mathbf{e}_2$ the right-hand side and looks whether the result is in $\mathbf{L}$. But recall that the above equation only holds for all but $\omega - p$ coordinates. Thus, one cannot simply match a candidate solution to one entry in $\mathbf{L}$.

The solution to this problem in Stern's algorithm is to introduce another parameter ℓ and to test whether there is an *exact* match on ℓ out of all $n - k$ coordinates. For those pairs $(\mathbf{e}_1, \mathbf{e}_2)$ whose result matches on ℓ coordinates, one checks whether they in total match on all but $\omega - p$ coordinates. However, this two-step approach for *approximately* matching similar vectors introduces another probability that enters the running time – namely that both vectors match on the chosen ℓ coordinates. Clearly, one would like to have some algorithm that directly addresses the approximate matching problem given by identity (2). Altogether, Stern's algorithm leads to the first asymptotical improvement since Prange's algorithm.

[1] Throughout the paper we ignore any rounding issues.

3 Our Decoding Algorithm

3.1 Application to Stern's algorithm

Our main observation is that the matching in Stern's algorithm can be done in a smarter way by an algorithm for approximately matching vectors. We propose such an algorithm in Section 4. Our algorithm NearestNeighbor works on two lists $\mathbf{L}, \mathbf{R}$ with uniformly distributed and pairwise independent entries from $\mathbb{F}_2^m$, where one guarantees the existence of a pair from $\mathbf{L} \times \mathbf{R}$ that has small Hamming distance γm, $0 \leq \gamma \leq \frac{1}{2}$. On lists of equal size $|\mathbf{L}| = |\mathbf{R}|$ we achieve a running time that approaches $\widetilde{\mathcal{O}}(|\mathbf{L}|^{\frac{1}{1-\gamma}})$. Notice that our running time is sub-quadratic for any $\gamma < \frac{1}{2}$. The smaller γ the better is our algorithm. In the case of $\gamma = 0$, we achieve linear running time (up to logarithmic factors) which coincides with the simple sort-and-match routine.

Our new matching algorithm immediately gives us an improved complexity for decoding random binary linear codes. First we proceed as in Stern's algorithm by enumerating over all weight-$\frac{p}{2}$ candidates for $\mathbf{e}_1 \in \mathbb{F}_2^{k/2} \times 0^{k/2} \times 0^{n-k}$. We compute the left-hand side $\bar{\mathbf{H}}\mathbf{e}_1$ of Eq. (3) and store the result in a list $\mathbf{L}$. For the right-hand side we proceed similar and store $\bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}$ in a list $\mathbf{R}$.

Notice that by construction each pair $(\mathbf{e}_1, \mathbf{e}_2)$ of enumerated error vectors yields an error vector $\mathbf{e}_1 + \mathbf{e}_2 + (0^k || \mathbf{e}_q)$ with $\mathbf{e}_q = \bar{\mathbf{H}}(\mathbf{e}_1 + \mathbf{e}_2) + \bar{\mathbf{s}}$ such that identity (2) holds. Moreover, by construction we have $\mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p$. Thus all that remains is to find among all tuples $(\bar{\mathbf{H}}\mathbf{e}_1, \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}) \in \mathbf{L} \times \mathbf{R}$ one with small Hamming distance $\omega - p$.

Our complete algorithm Decode is described in Algorithm 1.

Before we analyze correctness and running time of algorithm Decode, we want to explain the idea of the subroutine NearestNeighbor.

Basic Idea of NearestNeighbor: Let m be the length of the vectors in $\mathbf{L}$ and $\mathbf{R}$ and define a constant λ such that $|\mathbf{L}| = |\mathbf{R}| = 2^{\lambda m}$. Assume the permutation π is *good* and hence $(\mathbf{e}_1^*, \mathbf{e}_2^*)$ exist such that $\mathbf{wt}(\mathbf{e}_1^* + \mathbf{e}_2^*) = p$ and $\mathbf{wt}(\bar{\mathbf{H}}\mathbf{e}_1^* + \bar{\mathbf{H}}\mathbf{e}_2^* + \bar{\mathbf{s}}) = \omega - p =: \gamma m$ for some $0 < \gamma < \frac{1}{2}$ holds. Let $\mathbf{u}^* := \bar{\mathbf{H}}\mathbf{e}_1^*$ and $\mathbf{v}^* := \bar{\mathbf{H}}\mathbf{e}_2^* + \bar{\mathbf{s}}$. Our algorithm NearestNeighbor gets the input $(\mathbf{L}, \mathbf{R}, \gamma)$ and outputs a list $\mathbf{C}$, where $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{C}$ with overwhelming probability. Thus our algorithm solves the following problem.

Definition 2 (NN problem). *Let $m \in \mathbb{N}$, $0 < \gamma < \frac{1}{2}$ and $0 < \lambda < 1$. In the (m, γ, λ)-Nearest Neighbor (NN) problem, we are given γ and two lists $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$ of equal size $2^{\lambda m}$ with uniform and pairwise independent vectors. If there exists a pair $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L} \times \mathbf{R}$ with Hamming distance $\Delta(\mathbf{u}^*, \mathbf{v}^*) = \gamma m$, we have to output a list $\mathbf{C}$ that contains $(\mathbf{u}^*, \mathbf{v}^*)$.*

Notice that in Definition 2 the lists $\mathbf{L}$, $\mathbf{R}$ themselves do not have to be independent, e.g. $\mathbf{L} = \mathbf{R}$ is allowed. A naive algorithm solves the NN problem by simply computing the Hamming distance of each $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ in quadratic time $\widetilde{\mathcal{O}}((2^{\lambda m})^2)$. In the following we describe a sub-quadratic algorithm for the NN problem.

Algorithm 1 DECODE

1: **procedure** DECODE
2: **Input:** n, k, $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, $\mathbf{x} \in \mathbb{F}_2^n$
3: **Output:** $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathbf{He} = \mathbf{Hx}$ and $\mathbf{wt}(\mathbf{e}) \leq d$ (FDD), $\mathbf{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$ (HDD)
4: $\mathbf{s} \leftarrow \mathbf{Hx}$ ▷ compute the syndrome
5: $d \leftarrow H^{-1}(1 - \frac{k}{n}) \cdot n$ ▷ H: bin. entropy function, inverse H^{-1} maps to $[0, \frac{1}{2}]$
6: **for** $\omega \leftarrow 0 \dots d$ **do** ▷ (FDD) or $\omega \leftarrow 0 \dots \lfloor \frac{d-1}{2} \rfloor$ in the HDD case
7: Choose $0 < p < \omega$ ▷ We find an optimal choice of p numerically
8: **repeat** $\mathsf{poly}(n) \cdot \frac{1}{\mathbb{P}[\pi \text{ is good}]}$ many times
9: $\pi \leftarrow$ random permutation on $\mathbb{F}_2^n$.
10: $(\cdot||\mathbf{Q}) \leftarrow \pi(\mathbf{H})$ (permute columns) with $\mathbf{Q} \leftarrow \mathbb{F}_2^{(n-k)\times(n-k)}$
11: choose another permutation (goto line 9), if $\mathbf{Q}$ is not invertible
12: $\bar{\mathbf{H}} \leftarrow \mathbf{Q}^{-1}\pi(\mathbf{H})$ and $\bar{\mathbf{s}} \leftarrow \mathbf{Q}^{-1}\mathbf{s}$
13: $\mathbf{L} \leftarrow \bar{\mathbf{H}}\mathbf{e}_1$ for all $\mathbf{e}_1 \in \mathbb{F}_2^{k/2} \times 0^{k/2} \times 0^{n-k}$ with $\mathbf{wt}(\mathbf{e}_1) = \frac{p}{2}$
14: $\mathbf{R} \leftarrow \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}$ for all $\mathbf{e}_2 \in 0^{k/2} \times \mathbb{F}_2^{k/2} \times 0^{n-k}$ with $\mathbf{wt}(\mathbf{e}_2) = \frac{p}{2}$
15: $\mathbf{C} \leftarrow$ NEARESTNEIGHBOR$(\mathbf{L}, \mathbf{R}, \frac{\omega-p}{n-k})$
16: **if** $(\mathbf{u}, \mathbf{v}) \in \mathbf{C} \cap (\mathbf{L} \times \mathbf{R})$ with Hamming distance $\Delta(\mathbf{u}, \mathbf{v}) = \omega - p$ **then**
17: find $(\mathbf{e}_1, \mathbf{e}_2)$ s.t. $\mathbf{u} = \bar{\mathbf{H}}\mathbf{e}_1$ and $\mathbf{v} = \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}$ ▷ binary search in $\mathbf{L}, \mathbf{R}$
18: **return** $\pi^{-1}(\mathbf{e}_1 + \mathbf{e}_2 + (0^k||\mathbf{u} + \mathbf{v}))$
19: **end if**
20: **until**
21: **end for**
22: **end procedure**

Given an initial list pair $\mathbf{L}, \mathbf{R}$, our main idea is to create exponentially many pairs of sublists $\mathbf{L}', \mathbf{R}'$. Each sublist is computed by first choosing a random partition $A \subset [m]$ of the columns of size $\frac{m}{2}$. We keep only those elements in $\mathbf{L}', \mathbf{R}'$ that have a certain Hamming weight $h \cdot \frac{m}{2}$ on the columns defined by A, for some $0 < h < \frac{1}{2}$ that only depends on λ. The parameter h will be chosen s.t. each of the $\mathbf{L}', \mathbf{R}'$ have expected polynomially (in m) many elements. We create as many sublists $\mathbf{L}', \mathbf{R}'$ s.t. with overwhelming probability there exists a pair of sublists $\mathbf{L}^*, \mathbf{R}^*$ with $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L}^* \times \mathbf{R}^*$. For each sublist pair $\mathbf{L}', \mathbf{R}'$ we check naively for a possible "good" vector by computing the Hamming distance $\Delta(\mathbf{u}', \mathbf{v}')$ for all $(\mathbf{u}', \mathbf{v}') \in \mathbf{L}' \times \mathbf{R}'$. Notice that this results only in a polynomial blow-up, because the list sizes are polynomial. We store all vectors $(\mathbf{u}, \mathbf{v})$ with the correct Hamming distance in the output list $\mathbf{C}$.

The idea of the algorithm is summarized in Fig. 4.

We will discuss the algorithm NEARESTNEIGHBOR in more detail in Section 4. The following theorem that we prove in Section 5 states its correctness and time complexity.

Theorem 1. *For any constant $\varepsilon > 0$ and any $\lambda < 1 - H(\frac{\gamma}{2})$,* NEARESTNEIGHBOR *solves the (m, γ, λ)-NN problem with overwhelming probability (over both the coins of the algorithm and the random choice of the input) in time*

$$\widetilde{\mathcal{O}}\left(2^{(y+\varepsilon)m}\right) \quad \textit{with} \quad y := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda) - \frac{\gamma}{2}}{1-\gamma}\right)\right).$$

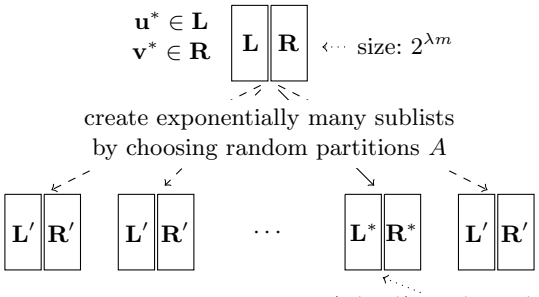

Fig. 4. Main idea of our algorithm NearestNeighbor

In Definition 2, we defined our list sizes $|\mathbf{L}| = |\mathbf{R}| = 2^{\lambda m}$ to be exponential in m, which is the cryptographically relevant scenario. A naive solution of the NN problem yields an exponent of $2\lambda m$, so we are interested in quotients $\frac{y}{\lambda} < 2$. In the following corollary we achieve a complexity of $\widetilde{\mathcal{O}}(|\mathbf{L}|^{\frac{1}{1-\gamma}})$ in the case of polynomial list sizes $|\mathbf{L}| = |\mathbf{R}|$. This is the best case scenario for our algorithm, which is in the literature often referred to as the *unlimited amount of data* case. Notice that the quotient $\frac{y}{\lambda}$ is strictly increasing in λ until we reach the prerequisite bound $\lambda = 1 - H(\frac{\gamma}{2})$, beyond which our algorithm does no longer work. Finding a better dependency for the NN problem on λ would immediately result in further improvements for the decoding bounds from Theorems 2 and 3.

Corollary 1. *In the case of a list size $|\mathbf{L}| = |\mathbf{R}|$ that is polynomial in m, we obtain a complexity exponent $\lim_{\lambda\to 0} y/\lambda = \frac{1}{1-\gamma}$, i.e. our complexity is $\widetilde{\mathcal{O}}(|\mathbf{L}|^{\frac{1}{1-\gamma}})$.*

Proof. Notice that we defined the inverse of the binary entropy function as $H^{-1}(\cdot)$ and that $H^{-1}(1) = \frac{1}{2}$. The derivative of the binary entropy function is $H'(x) = \log_2\left(\frac{1}{x} - 1\right)$ and the derivative of the inverse of the binary entropy function is $(H^{-1}(1-\lambda))' = \frac{-1}{\log_2(\frac{1}{H^{-1}(1-\lambda)} - 1)}$. We obtain the result by the following calculation, using L'Hospital's rule twice.

$$\begin{aligned}
\lim_{\lambda\to 0} \frac{y}{\lambda} &= \lim_{\lambda\to 0} -(1-\gamma)\log_2\left(\frac{1-\gamma}{H^{-1}(1-\lambda) - \frac{\gamma}{2}} - 1\right)\frac{1}{1-\gamma}\frac{-1}{\log_2(\frac{1}{H^{-1}(1-\lambda)} - 1)} \\
&= \lim_{\lambda\to 0} \ln\left(\frac{1-\gamma}{H^{-1}(1-\lambda) - \frac{\gamma}{2}} - 1\right) / \ln\left(\frac{1}{H^{-1}(1-\lambda)} - 1\right) \\
&= \lim_{\lambda\to 0} \frac{\left(\frac{1-\gamma}{H^{-1}(1-\lambda) - \frac{\gamma}{2}} - 1\right)^{-1} \frac{(-1)(1-\gamma)}{\left(H^{-1}(1-\lambda) - \frac{\gamma}{2}\right)^2} (H^{-1}(1-\lambda))'}{\left(\frac{1}{H^{-1}(1-\lambda)} - 1\right)^{-1} \frac{(-1)}{(H^{-1}(1-\lambda))^2} (H^{-1}(1-\lambda))'} \\
&= \lim_{\lambda\to 0} \frac{\frac{(-1)(1-\gamma)}{\left(H^{-1}(1-\lambda) - \frac{\gamma}{2}\right)^2}}{\frac{(-1)}{(H^{-1}(1-\lambda))^2}} = \lim_{\lambda\to 0}(1-\gamma)\left(\frac{H^{-1}(1-\lambda)}{H^{-1}(1-\lambda) - \frac{\gamma}{2}}\right)^2 = \frac{1}{1-\gamma} \qquad \square
\end{aligned}$$

In the following Theorem we show that DECODE is correct and prove its time complexity.

Theorem 2 (complexity and correctness). DECODE *solves the decoding problem with overwhelming probability in time* $\mathcal{O}(2^{0.114n})$ *in the full and time* $\mathcal{O}(2^{0.0550n})$ *in the half distance decoding setting.*

Proof. Let us define

$$\gamma := \frac{\omega - p}{n-k}, \quad r := H\left(\frac{\omega}{n}\right) - \frac{k}{n} \cdot H\left(\frac{p}{k}\right) - \left(1 - \frac{k}{n}\right) \cdot H\left(\gamma\right), \quad \mu := \frac{k}{2n} \cdot H\left(\frac{p}{k}\right)$$

and

$$y := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1 - \frac{\mu n}{n-k}) - \frac{\gamma}{2}}{1-\gamma}\right)\right).$$

We want to show that for any $\varepsilon > 0$ DECODE solves the decoding problem with overwhelming probability in time

$$\widetilde{\mathcal{O}}\left(2^{rn}\left(2^{\mu n} + 2^{(y+\varepsilon)(n-k)}\right)\right). \tag{4}$$

In line 8 of DECODE, we repeat $\mathsf{poly}(n) \cdot \frac{1}{\mathbb{P}[\pi \text{ is good}]}$ times. A permutation π is good, whenever $p/2$ ones are in the first $k/2$ columns, $p/2$ in the subsequent $k/2$ columns and $\omega - p$ ones in the last $n-k$ columns. Additionally, $\mathbf{Q}$ (as defined in line 10) has to be invertible (which happens with constant probability). Therefore, the number of repetitions until we find a *good* π is

$$\mathsf{poly}(n) \cdot \frac{\binom{n}{\omega}}{\binom{k/2}{p/2}^2 \cdot \binom{n-k}{\gamma(n-k)}} = \widetilde{\mathcal{O}}\left(2^{n \cdot H(\omega/n) - k \cdot H(p/k) - (n-k) \cdot H(\gamma)}\right) = \widetilde{\mathcal{O}}(2^{rn}).$$

We fix a repetition that leads to a good permutation π. In this repetition, we therefore have $\mathbf{e}_1^*, \mathbf{e}_2^*$ with $\mathbf{wt}(\mathbf{e}_1^* + \mathbf{e}_2^*) = p$ and $\Delta(\mathbf{u}^*, \mathbf{v}^*) = \omega - p =: \gamma(n-k)$ with $\mathbf{u}^* := \bar{\mathbf{H}}\mathbf{e}_1^*$ and $\mathbf{v}^* := \bar{\mathbf{H}}\mathbf{e}_2^* + \bar{\mathbf{s}}$. In lines 13 and 14, the algorithm creates two lists of uniform and pairwise independent vectors of size $n-k$ s.t. by construction $\mathbf{u}^* \in \mathbf{L}$ and $\mathbf{v}^* \in \mathbf{R}$ and

$$|\mathbf{L}| = |\mathbf{R}| = \binom{k/2}{p/2} = \widetilde{\mathcal{O}}\left(2^{k/2 \cdot H(p/k)}\right) = \widetilde{\mathcal{O}}(2^{\mu n}).$$

Notice that $\mu n = \lambda(n-k)$ controls the list size. By a suitably small choice of p one can always satisfy the prerequisite $\lambda < 1 - H(\frac{\gamma}{2})$ of Theorem 1. Thus we obtain an instance $(\mathbf{L}, \mathbf{R}, \gamma)$ of the $(n-k, \gamma, \mu\frac{n}{n-k})$-NN problem, for which Theorem 1 guarantees to output a list $\mathbf{C}$ that contains the solution $(\mathbf{u}^*, \mathbf{v}^*)$ with overwhelming probability. Notice that any vector $(\mathbf{u}, \mathbf{v}) \in \mathbf{C} \cap (\mathbf{L} \times \mathbf{R})$ with a Hamming distance of $\omega - p$ solves our problem, because the corresponding $\mathbf{e}_1$ with $\bar{\mathbf{H}}\mathbf{e}_1 = \mathbf{u}$ and $\mathbf{e}_2$ with $\bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}} = \mathbf{v}$ have the property $\mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p$. This in turn leads to $\mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2 + (0^k || (\mathbf{u} + \mathbf{v})) = \omega$. In line 17, the $(\mathbf{e}_1, \mathbf{e}_2)$ can be

found by a binary search in slightly modified lists $\mathbf{L}, \mathbf{R}$ (that also store $\mathbf{e}_1$ and $\mathbf{e}_2$). Thus, with overwhelming probability, the algorithm outputs a solution to the decoding problem in line 18.

Also by Theorem 1, an application of algorithm NearestNeighbor has time complexity $\widetilde{\mathcal{O}}(2^{(y+\varepsilon)(n-k)})$ for any $\varepsilon > 0$. Notice that this complexity is independent of whether we have a good permutation π or not. This complexity has to be added to the creation time of the input lists $\mathbf{L}, \mathbf{R}$. Recall that the loop has to be repeated $\widetilde{\mathcal{O}}(2^{rn})$ times, leading to the runtime of Eq. (4).

Numerical optimization in the half distance decoding case yields time complexity $\mathcal{O}(2^{0.0550n})$ in the worst case $k/n \approx 0.466$ with $p/n \approx 0.00383$. In the full distance decoding case we get a runtime of $\mathcal{O}(2^{0.114n})$ in the worst case $k/n \approx 0.447$ with $p/n \approx 0.01286$. □

3.2 Application to the BJMM algorithm

It is possible to apply our idea to the decoding algorithm of Becker, Joux, May and Meurer (BJMM) [4]. We already explained that BJMM is a variant of Stern's algorithm and thus a Meet-in-the-Middle algorithm that constructs two list $\tilde{\mathbf{L}}, \tilde{\mathbf{R}}$. The major difference is that $\tilde{\mathbf{L}}, \tilde{\mathbf{R}}$ are not directly enumerated as the lists $\mathbf{L}, \mathbf{R}$ in Stern's algorithm. Instead, $\tilde{\mathbf{L}}, \tilde{\mathbf{R}}$ are constructed in a more involved tree-based manner. This has the benefit that the list length is significantly smaller than in Stern's construction, which in turn leads to an improved running time. This similarity however enables us to directly apply our technique to the BJMM algorithm. Namely, we have to simply replace in Decode the construction of $\mathbf{L}, \mathbf{R}$ by the BJMM-construction of $\tilde{\mathbf{L}}, \tilde{\mathbf{R}}$, on which we apply our NearestNeighbor-algorithm.

Notice that as opposed to Section 3.1 not all possible vector pairs in $\mathbf{C}$ with the correct Hamming distance solve the decoding problem. The issue is that the corresponding $\mathbf{e}_1, \mathbf{e}_2$ do not necessarily have a Hamming distance of p. Thus, additionally to $\Delta(\mathbf{u}, \mathbf{v}) = \omega - p$, we have to verify that also $\Delta(\mathbf{e}_1, \mathbf{e}_2) = p$ holds.

Algorithm DecodeBJMM describes the application of our algorithm in the BJMM framework. Notice that up to line 22 the algorithm is identical to the one in [4]. The only difference is the final step (from line 23). Instead of using the exact matching of Stern, we use our NearestNeighbor algorithm that searches for two vectors that are close (i.e. have Hamming distance $\omega - p$) on the remaining $n - k - \ell$ coordinates.

Algorithm DecodeBJMM uses a subroutine BaseLists. As described in [4], BaseLists chooses a random partition of the first $k + \ell$ columns into two sets $P_1, P_2 \subseteq [k + \ell]$ of equal size. Then a list $\mathbf{B}_1$ is created that contains all vectors $\mathbf{b}_1 \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$ with $\mathbf{wt}(\mathbf{b}_1) = \frac{p}{8} + \frac{\varepsilon_1}{4} + \frac{\varepsilon_2}{2}$ that are zero on the coordinates from P_2. Analogously, a list $\mathbf{B}_2$ is build, that contains all vectors $\mathbf{b}_2 \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$ of the same Hamming weight as above, but with zeros on the coordinates from P_1. Thus, with inverse polynomial probability, a fixed vector of size $k + \ell$ with Hamming weight $\frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2$ can be represented as a sum of an element in $\mathbf{B}_1$ and an element in $\mathbf{B}_2$. Repeating this choice a polynomial number of times guarantees the representation with overwhelming probability.

Algorithm 2 DecodeBJMM

procedure DecodeBJMM
  Input: $n, k, \mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, $\mathbf{x} \in \mathbb{F}_2^n$
  Output: $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathbf{He} = \mathbf{Hx}$ and $\mathbf{wt}(\mathbf{e}) \leq d$ (FDD), $\mathbf{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$ (HDD)
  $\mathbf{s} \leftarrow \mathbf{Hx}$
  $d \leftarrow H^{-1}(1 - \frac{k}{n}) \cdot n$ ▷ H: bin. entropy function, inverse H^{-1} maps to $[0, \frac{1}{2}]$.
  for $\omega \leftarrow 0 \dots d$ **do** ▷ (FDD) or $\omega \leftarrow 0 \dots \lfloor \frac{d-1}{2} \rfloor$ in the HDD case
    Choose $0 < p, \varepsilon_1, \varepsilon_2 < \omega$, $0 < \ell_2 < \ell < n - k$ ▷ optimize numerically
    repeat $\mathsf{poly}(n) \cdot \frac{1}{\mathbb{P}[\pi \text{ is good}]}$ many times
      $\pi \leftarrow$ random permutation on $\mathbb{F}_2^n$.
      $(\cdot || \mathbf{Q}) \leftarrow \pi(\mathbf{H})$ (permute columns) with $\mathbf{Q} \leftarrow \mathbb{F}_2^{(n-k)\times(n-k)}$
      choose another permutation (goto line 9), if $\mathbf{Q}$ is not invertible
      $\bar{\mathbf{H}} \leftarrow \mathbf{Q}^{-1}\pi(\mathbf{H})$ and $\bar{\mathbf{s}} \leftarrow \mathbf{Q}^{-1}\mathbf{s}$
      $\mathbf{t}_L \in_R \mathbb{F}_2^{\ell}, \mathbf{t}_{L_0}, \mathbf{t}_{L_1}, \mathbf{t}_{R_0} \in_R \mathbb{F}_2^{\ell_2}$ ▷ choose uniformly at random
      $\mathbf{t}_R = [\bar{\mathbf{s}}]_\ell - \mathbf{t}_L$ ▷ $[\cdot]_c$ restricts to first c columns
      $\mathbf{t}_{R_1} = [\bar{\mathbf{s}}]_{\ell_2} - \mathbf{t}_{L_0} - \mathbf{t}_{L_1} - \mathbf{t}_{R_0}$ ▷ $[\cdot]^c$ restricts to last c columns
      $\mathbf{L}_0 \leftarrow$ BaseLists$(\bar{\mathbf{H}}, p, \varepsilon_1, \varepsilon_2, \mathbf{t}_{L_0})$ ▷ list of $\mathbf{b} \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$
      $\mathbf{L}_1 \leftarrow$ BaseLists$(\bar{\mathbf{H}}, p, \varepsilon_1, \varepsilon_2, \mathbf{t}_{L_1})$ ▷ with $\mathbf{wt}(\mathbf{b}) = \frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2$
      $\mathbf{R}_0 \leftarrow$ BaseLists$(\bar{\mathbf{H}}, p, \varepsilon_1, \varepsilon_2, \mathbf{t}_{R_0})$ ▷ s.t. $[\bar{\mathbf{H}}\mathbf{b}]_{\ell_2} = \mathbf{t}_{L_0}$
      $\mathbf{R}_1 \leftarrow$ BaseLists$(\bar{\mathbf{H}}, p, \varepsilon_1, \varepsilon_2, \mathbf{t}_{R_1})$
      $\mathbf{L} \leftarrow [\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y})]^{n-k-\ell}$ for all $\mathbf{x} \in \mathbf{L}_0, \mathbf{y} \in \mathbf{L}_1$ with $[\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y})]_\ell = \mathbf{t}_L$
      $\mathbf{R} \leftarrow [\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y}) + \bar{\mathbf{s}}]^{n-k-\ell}$ for all $\mathbf{x} \in \mathbf{R}_0, \mathbf{y} \in \mathbf{R}_1$ with $[\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y})]_\ell = \mathbf{t}_R$
      (In lines 20, 21: only keep elements with $\mathbf{wt}(\mathbf{x} + \mathbf{y}) = \frac{p}{2} + \varepsilon_1$.)
      $\mathbf{C} \leftarrow$ NearestNeighbor$(\mathbf{L}, \mathbf{R}, \frac{\omega - p}{n-k-\ell})$
      for all $(\mathbf{u}, \mathbf{v}) \in \mathbf{C} \cap (\mathbf{L} \times \mathbf{R})$ with distance $\Delta(\mathbf{u}, \mathbf{v}) = \omega - p$ **do**
        find $(\mathbf{e}_1, \mathbf{e}_2)$ s.t. $\mathbf{u} = [\bar{\mathbf{H}}\mathbf{e}_1]^{n-k-\ell}$ and $\mathbf{v} = [\bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}]^{n-k-\ell}$
        if $\mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p$ **then**
          return $\pi^{-1}(\mathbf{e}_1 + \mathbf{e}_2 + (0^{k+\ell} || \mathbf{u} + \mathbf{v}))$
        end if
      end for
    until
  end for
end procedure

BaseLists continues by computing the corresponding values $\bar{\mathbf{H}}\mathbf{b}_1$ for each element $\mathbf{b}_1 \in \mathbf{B}_1$, stores these elements and sorts the list by these values. Eventually an output list of vectors in $\mathbf{b} \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$ with weight $\frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2$ is computed by a standard meet-in-the-middle technique s.t. $\bar{\mathbf{H}}\mathbf{b}$ equals the input target value $\mathbf{t}$ on the first ℓ_2 coordinates.

The same technique is used in lines 20 and 21. In the computation of $\mathbf{L}$, for each pair of vectors $(\mathbf{x}, \mathbf{y}) \in \mathbf{L}_0 \times \mathbf{L}_1$ a list of sums $\mathbf{x} + \mathbf{y}$ is obtained such that $\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y})$ matches a uniformly chosen target value $\mathbf{t}_L$ on the first ℓ coordinates. After this step we also restrict to only those elements that have a certain Hamming weight of $\frac{p}{2} + \varepsilon_1$, since by [4] the target solution splits in two vectors of this particular weight. The computation tree is illustrated in Figure 5.

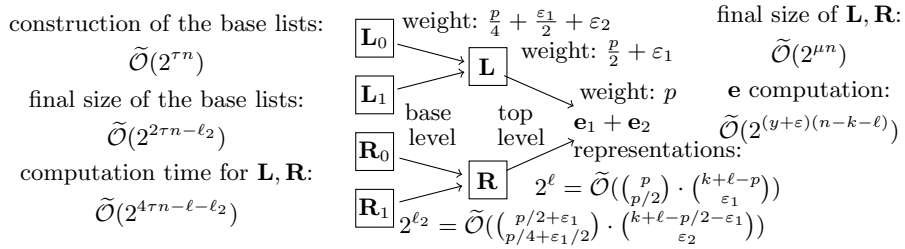

Fig. 5. Computation tree of DecodeBJMM

Theorem 3. DecodeBJMM *solves the decoding problem with overwhelming probability in time* $\mathcal{O}(2^{0.097n})$ *in the full distance decoding setting and time* $\mathcal{O}(2^{0.0473n})$ *in the half distance decoding setting.*

Proof. Let us define

$$\gamma := \frac{\omega - p}{n-k-\ell}, \quad r := H\left(\frac{\omega}{n}\right) - \frac{k+\ell}{n} \cdot H\left(\frac{p}{k+\ell}\right) - \left(1 - \frac{k+\ell}{n}\right) \cdot H(\gamma),$$

$$\tau := \frac{k+\ell}{2n} \cdot H\left(\frac{\frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2}{k+\ell}\right), \quad \ell \stackrel{!}{=} p + (k+\ell-p) \cdot H\left(\frac{\varepsilon_1}{k+\ell-p}\right)$$

$$\mu := \frac{k+\ell}{n} \cdot H\left(\frac{\frac{p}{2} + \varepsilon_1}{k+\ell}\right) - \frac{\ell}{n}, \ell_2 := \frac{p}{2} + \varepsilon_1 + (k+\ell-\frac{p}{2}-\varepsilon_1) \cdot H\left(\frac{\varepsilon_2}{k+\ell-\frac{p}{2}-\varepsilon_1}\right)$$

and

$$y := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1 - \frac{\mu n}{n-k-\ell}) - \frac{\gamma}{2}}{1-\gamma}\right)\right).$$

We want to show that for any $\varepsilon > 0$ the decoding problem can be solved with overwhelming probability in time

$$\widetilde{\mathcal{O}}\left(2^{\tau n}\left(2^{\tau n} + 2^{2\tau n-\ell_2} + 2^{4\tau n-\ell-\ell_2} + 2^{\mu n} + 2^{(y+\varepsilon)(n-k-\ell)}\right)\right).$$

The correctness and time complexity of the first part of the algorithm (i.e. the computation of $\mathbf{L}$ and $\mathbf{R}$ up to line 22) was already shown in [4]. Let us summarize the time complexity for this computation. First of all we have a loop that guarantees a *good* distribution with overwhelming probability. In our case, a good splitting would be p ones on the first $k+\ell$ coordinates and $\omega-p$ ones on the remaining $n-k-\ell$ coordinates. Thus the necessary number of repetitions is $\widetilde{\mathcal{O}}(\binom{n}{\omega}/[\binom{k+\ell}{p}\cdot\binom{n-k-\ell}{\omega-p}]) = \widetilde{\mathcal{O}}(2^{rn})$.

The algorithm of BJMM [4] makes use of the so-called *representation technique* introduced by Joux and Howgrave-Graham [11]. The main idea is to blow up the search space such that each error vector can be represented as a sum of two vectors in many different ways. In the algorithm, all but one of these *representations* are filtered out using some restriction. Inside the loop, we therefore first need to make sure that 2^ℓ is the number of *representations* on the top level, because we restrict to ℓ binary coordinates. On the top level we split the $\mathbb{F}_2^{k+\ell}$ vector with p ones in a sum of two vectors in $\mathbb{F}_2^{k+\ell}$ with $\frac{p}{2}+\varepsilon_1$ ones each. Thus there are $\binom{p}{p/2}$ ways to represent the ones (as $1+0$ or $0+1$) and $\binom{k+\ell-p}{\varepsilon_1}$ ways to represent the zeros (as $0+0$ or $1+1$). Hence we need to choose ℓ such that $2^\ell = \widetilde{\Theta}(\binom{p}{p/2}\cdot\binom{k+\ell-p}{\varepsilon_1})$. On the bottom level, vectors with $\frac{p}{2}+\varepsilon_1$ ones are represented as sums of two vectors with $\frac{p}{4}+\frac{\varepsilon_1}{2}+\varepsilon_2$ ones each. In this case there are $\binom{p/2+\varepsilon_1}{p/4+\varepsilon_1/2}$ ways to represent the ones and $\binom{k+\ell-p/2-\varepsilon_1}{\varepsilon_2}$ ways to represent the zeros. Thus we choose $2^{\ell_2} = \widetilde{\Theta}(\binom{p/2+\varepsilon_1}{p/4+\varepsilon_1/2}\cdot\binom{k+\ell-p/2-\varepsilon_1}{\varepsilon_2})$.

The computation starts by creating the four base lists. In the first step two lists with vectors of size $\frac{k+\ell}{2}$ and $\frac{p}{8}+\frac{\varepsilon_1}{4}+\frac{\varepsilon_2}{2}$ ones are created, which takes time $\widetilde{\mathcal{O}}(\binom{\frac{k+\ell}{2}}{\frac{p}{8}+\frac{\varepsilon_1}{4}+\frac{\varepsilon_2}{2}}) = \widetilde{\mathcal{O}}(2^{\tau n})$. These two lists are merged, considering each pair of one vector of the first list and one vector of the second list such that the first ℓ_2 coordinates of the sum are a fixed value (i.e. restricting to one special *representation*). The number of elements in the base lists is therefore $\widetilde{\mathcal{O}}(2^{2\tau n}/2^{\ell_2})$.

In lines 20 and 21 the top level lists $\mathbf{L}$ and $\mathbf{R}$ are computed from the base lists. In this step, the vectors are restricted to additional $\ell-\ell_2$ coordinates, resulting in a total restriction of 2^ℓ. Therefore, the time complexity is $\widetilde{\mathcal{O}}((2^{2\tau n}/2^{\ell_2})^2/(2^{\ell-\ell_2})) = \widetilde{\mathcal{O}}(2^{4\tau n-\ell-\ell_2})$.

In line 22 the algorithm restricts the lists $\mathbf{L}$ and $\mathbf{R}$ to those vectors with $\frac{p}{2}+\varepsilon_1$ ones. Due to the fact that the vectors are restricted to fixed ℓ coordinates, the number of elements can be upper bounded by $\widetilde{\mathcal{O}}(\binom{k+\ell}{p/2+\varepsilon_1}/2^\ell) = \widetilde{\mathcal{O}}(2^{\mu n})$.

In the final step we have two lists of uniform (because the elements are a linear combination of the columns of $\mathbf{H}$) and pairwise independent (because each element is computed by a linear combination of pairwise different columns) vectors in $\mathbb{F}_2^{n-k-\ell}$.

From the analysis in [4] we know that there are $\mathbf{e}_1^*, \mathbf{e}_2^* \in \mathbb{F}_2^{k+\ell}\times 0^{n-k-\ell}$ with $\mathbf{wt}(\mathbf{e}_1+\mathbf{e}_2)=p$ such that $\mathbf{u}^* = [\bar{\mathbf{H}}\mathbf{e}_1^*]^{n-k-\ell} \in \mathbf{L}$ and $\mathbf{v}^* = [\bar{\mathbf{H}}\mathbf{e}_2^*+\bar{\mathbf{s}}]^{n-k-\ell} \in \mathbf{R}$, where $\mathbf{wt}(\mathbf{u}^*+\mathbf{v}^*) = \omega-p$. Therefore, by Theorem 1, NearestNeighbor outputs a list $\mathbf{C}$ that contains $(\mathbf{u}^*, \mathbf{v}^*)$. The $(\mathbf{e}_1^*, \mathbf{e}_2^*)$ can be found in line 25 by binary searching in slightly modified $\mathbf{L}, \mathbf{R}$ (that also contain $\mathbf{x}+\mathbf{y}$).

Thus $\pi^{-1}(\mathbf{e}_1^* + \mathbf{e}_2^* + (0^{k+\ell}||\mathbf{u}^* + \mathbf{v}^*))$ (with weight ω) is a correct solution to the problem, because

$$\begin{aligned}\bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^* + (0^{k+\ell}||\mathbf{u}^* + \mathbf{v}^*)) &= \bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + (0^\ell||\mathbf{u}^* + \mathbf{v}^*) \\ &= \bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + (0^\ell||[\bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + \bar{\mathbf{s}}]^{n-k-\ell}) \\ &= \bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + (\bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + \bar{\mathbf{s}}) = \bar{\mathbf{s}}.\end{aligned}$$

Notice that $[\bar{\mathbf{H}}(\mathbf{e}_1^* + \mathbf{e}_2^*) + \bar{\mathbf{s}}]_\ell = 0^\ell$ holds by construction. By Theorem 1 the time complexity of NearestNeighbor is $2^{(y+\varepsilon)(n-k-\ell)}$.

In the half distance decoding case, for the worst case $k/n \approx 0.45$ we get a time complexity of $\mathcal{O}(2^{0.0473n})$ with $p/n \approx 0.01667, \varepsilon_1/n \approx 0.00577$ and $\varepsilon_2/n \approx 0.00124$. In the full distance decoding setting, we have a runtime of $\mathcal{O}(2^{0.097n})$ with $k/n \approx 0.42, p/n \approx 0.06284, \varepsilon_1/n \approx 0.02001$ and $\varepsilon_2/n \approx 0.00391$.

□

4 Solving the Nearest Neighbor Problem

In this section we will describe NearestNeighbor, an algorithm that solves the Nearest Neighbor problem from Definition 2. We will prove correctness and time complexity of our algorithm in the subsequent section.

As already outlined in the previous section, given the input lists $\mathbf{L}$ and $\mathbf{R}$ with $|\mathbf{L}| = |\mathbf{R}| = 2^{\lambda m}$, our idea is to create exponentially many sublists $\mathbf{L}', \mathbf{R}'$ that are of expected polynomial size. The sublists are chosen such that with overwhelming probability our unknown solution $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L} \times \mathbf{R}$ with $\Delta(\mathbf{u}^*, \mathbf{v}^*) = \gamma m$ is contained in at least one of these sublists. The sublists $\mathbf{L}'$ (resp. $\mathbf{R}'$) are defined as all elements of $\mathbf{L}$ (resp. $\mathbf{R}$) that have a Hamming weight of $h\frac{m}{2}$ on the columns defined by a random partition $A \subset [m]$ of size $\frac{m}{2}$. In Lemma 3, we will prove that $h := H^{-1}(1-\lambda)$ with $0 \leq h \leq \frac{1}{2}$ is a suitable choice, because it leads to sublists of expected polynomial size.

We will prove in Lemma 2 that the required number of sublists such that $(\mathbf{u}^*, \mathbf{v}^*)$ is contained in one of these sublists is $\widetilde{\mathcal{O}}(2^{ym})$ with

$$y := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda) - \frac{\gamma}{2}}{1-\gamma}\right)\right). \tag{5}$$

We will make use of the fact that $y > \lambda$ for any constant $0 < \lambda < 1$, $0 < \gamma < \frac{1}{2}$, which can be verified numerically.

There is still one problem to solve, because we never discussed how to compute the $\mathbf{L}', \mathbf{R}'$ given $\mathbf{L}, \mathbf{R}$. A naive way to do so would be to traverse the original lists linearly and to check the weight condition. Unfortunately, this would have to be done for each sampled A, which would result in an overall complexity of $\widetilde{\mathcal{O}}(2^{ym} \cdot 2^{\lambda m})$.

Instead, as illustrated in Fig. 6, we do not proceed with the whole m coordinates at once, but first start with a strip $\{1, \ldots, \alpha_1 m\}$ of the left hand side columns and filter only on that strip. The resulting list pairs $\mathbf{L}^1, \mathbf{R}^1$ are still of

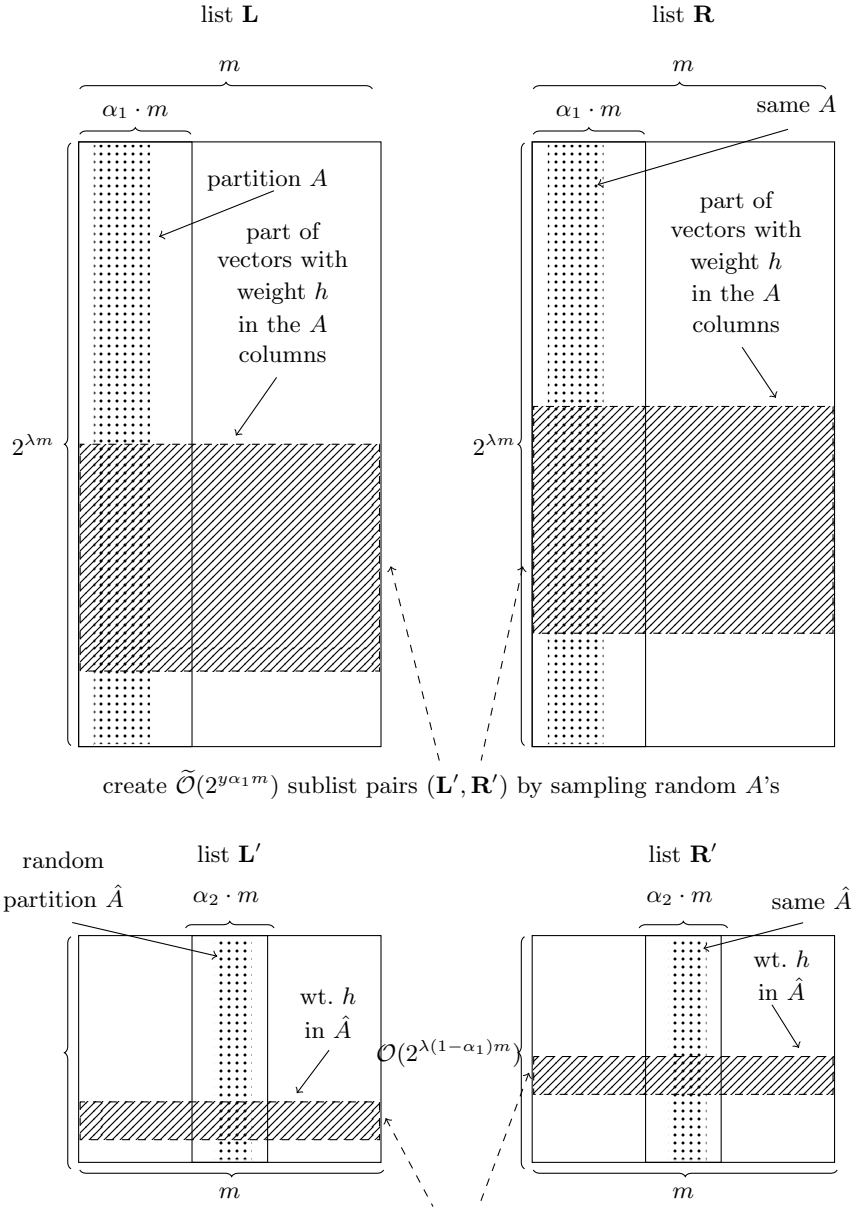

Fig. 6. Step-by-step computation of NearestNeighbor

exponential, but smaller, size. In the second step, we proceed on the subsequent $\alpha_2 m$ columns $\{\alpha_1 m + 1, \ldots, (\alpha_1 + \alpha_2)m\}$ to generate sublists $\mathbf{L}^2, \mathbf{R}^2$ that contain all elements that in addition have a small Hamming weight on the second strip. The advantage of this technique is that we are able to use the smaller lists $\mathbf{L}^1, \mathbf{R}^1$ to construct $\mathbf{L}^2, \mathbf{R}^2$, again by traversing these lists, instead of using $\mathbf{L}, \mathbf{R}$ for all the m columns.

As illustrated in Fig. 7, we grow a search tree of *constant* depth t, where the leaves are pairs of lists $\mathbf{L}^t, \mathbf{R}^t$, which were filtered on $(\alpha_1 + \ldots + \alpha_t)m$ coordinates. We choose $\alpha_1 + \ldots + \alpha_t = 1$ to cover all coordinates. The choice of the α_j will be given in Theorem 1 and is basically done in a way to balance the costs in each level of the search tree, which allows us to solve the problem in time $\widetilde{\mathcal{O}}(2^{(y+\varepsilon)m})$ for any constant $\varepsilon > 0$. Notice that we never actually compute the Hamming distance between elements from the list pairs, except for the very last step, where we obtain list pairs $\mathbf{L}^t, \mathbf{R}^t$ of small size $\widetilde{\mathcal{O}}(2^{\frac{\varepsilon}{2}m})$, and compute the Hamming distance of all pairs by a naive quadratic algorithm, which has time complexity $\widetilde{\mathcal{O}}(2^{\varepsilon m})$. Because we proceed analogously for any of the $\widetilde{\mathcal{O}}(2^{ym})$ sublists, we obtain an overall time complexity of $\widetilde{\mathcal{O}}(2^{(y+\varepsilon)m})$.

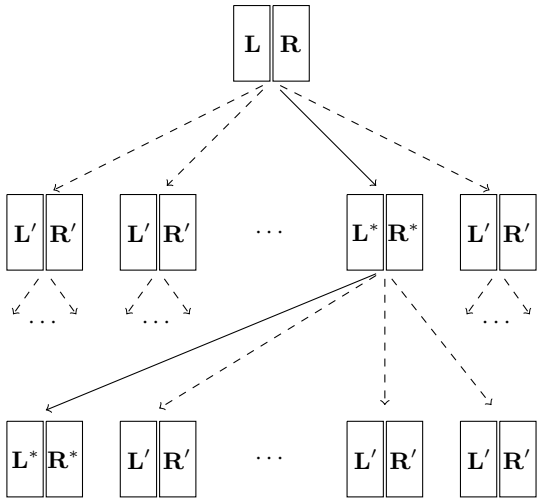

Fig. 7. Example: computation tree of depth $t = 2$ with one good path ($\rightarrow$)

In total, our algorithm heavily relies on the observation that the property of the pair $(\mathbf{u}^*, \mathbf{v}^*)$ with small Hamming distance $\Delta(\mathbf{u}^*, \mathbf{v}^*) = \gamma m$ also holds *locally* on each of the t strips. In case that the differing coordinates of $(\mathbf{u}^*, \mathbf{v}^*)$ would cluster in any of the strips $\alpha_j m$, we would by mistake sort out the pair. However, this issue can be easily resolved by rerandomizing the position of the coordinates in both input lists $\mathbf{L}$ and $\mathbf{R}$. Denote $\mathbf{z}^* := \mathbf{u}^* + \mathbf{v}^* \in \mathbb{F}_2^m$ and the splitting $\mathbf{z}^* = (\mathbf{z}_1^*, \ldots, \mathbf{z}_t^*) \in \mathbb{F}_2^{\alpha_1 m} \times \mathbb{F}_2^{\alpha_2 m} \times \ldots \times \mathbb{F}_2^{\alpha_t m}$ according to the t

strips. We will prove in Lemma 1 that after randomly permuting the m columns a polynomial in m number of times, there will be one permutation such that $\mathbf{wt}(\mathbf{z}_j) = \gamma\alpha_j m$ for all $1 \leq j \leq t$.

Furthermore, we want to enforce that $\mathbf{wt}(\mathbf{u}^*) = \mathbf{wt}(\mathbf{v}^*) = \frac{1}{2}m$, which also has to hold on each of the t strips. Define $\mathbf{u}^* = (\mathbf{u}_1^*, \ldots, \mathbf{u}_t^*)$ and $\mathbf{v}^* = (\mathbf{v}_1^*, \ldots, \mathbf{v}_t^*)$ as above. We therefore want to make sure that $\mathbf{wt}(\mathbf{u}_j^*) = \mathbf{wt}(\mathbf{v}_j^*) = \frac{1}{2}\alpha_j m$ for all $1 \leq j \leq t$. Notice that this can also be achieved by rerandomization. Concretely, we pick a uniformly random vector $\mathbf{r} \in \mathbb{F}_2^m$ and add this vector to all elements of both input lists $\mathbf{L}, \mathbf{R}$. Notice that the Hamming weight of our solution pair $(\mathbf{u}^*, \mathbf{v}^*)$ isn't changed by this operation. We will also show in Lemma 1 that after applying this process a polynomial (in m) number of times, the vectors $\mathbf{u}^*$ and $\mathbf{v}^*$ have the desired Hamming weight in at least one step.

Algorithm 3 NearestNeighbor

procedure NearestNeighbor($\mathbf{L}, \mathbf{R}, \gamma$) ▷ $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$, $0 < \gamma < \frac{1}{2}$
  compute vectors length m and size λ from $\mathbf{L}, \mathbf{R}$
  $y := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma}\right)\right)$ ▷ as defined in Theorem 1
  choose a constant $\varepsilon > 0$ ▷ could also be an input
  $t := \left\lceil \frac{\log(y-\lambda+\frac{\varepsilon}{2})-\log(\frac{\varepsilon}{2})}{\log(y)-\log(\lambda)} \right\rceil$ ▷ as defined in the proof of Theorem 1
  $\alpha_1 := \frac{y-\lambda+\frac{\varepsilon}{2}}{y}$ ▷ as defined in the proof of Theorem 1
  for $2 \leq j \leq t$ **do**
    $\alpha_i := \frac{y}{\lambda} \cdot \alpha_{i-1}$ ▷ as defined in the proof of Theorem 1
  end for
  for poly(m) uniformly random permutations π of $[m]$ **do**
    for poly(m) unif. rand. $\mathbf{r} \in \mathbb{F}_2^{\alpha_1 m} \times \ldots \times \mathbb{F}_2^{\alpha_t m}$ (wt. $\alpha_j \frac{m}{2}$ on each strip) **do**
      $\bar{\mathbf{L}} \leftarrow \pi(\mathbf{L}) + \mathbf{r}$
      $\bar{\mathbf{R}} \leftarrow \pi(\mathbf{R}) + \mathbf{r}$ ▷ permute columns and add $\mathbf{r}$ to all elements
      Remove all vectors from $\bar{\mathbf{L}}, \bar{\mathbf{R}}$ that are not of weight $\alpha_j \frac{m}{2}$ on each strip
      return NearestNeighborRec($\bar{\mathbf{L}}, \bar{\mathbf{R}}, m, t, \gamma, \lambda, \alpha_1, \ldots, \alpha_t, y, \varepsilon, 1$)
    end for
  end for
end procedure

Our algorithm NearestNeighbor starts by computing the length of the vectors m and a λ such that $|\mathbf{L}| = |\mathbf{R}| = 2^{\lambda m}$ from $\mathbf{L}$ and $\mathbf{R}$. This list size is used to compute the repetition parameter y. The algorithm chooses a constant $\varepsilon > 0$ that is part of the asymptotic time complexity. The parameter ε also determines the number of strips t and the relative sizes of the strips $\alpha_1, \ldots, \alpha_t$. Eventually, the two input lists are rerandomized by permuting the columns and adding a random vector. Another recursive algorithm NearestNeighborRec is then called with the rerandomized lists as input.

In the algorithm NearestNeighborRec we sample random partitions A of the columns, until it is guaranteed with overwhelming probability that the solution is in at least one of the created sublists. The sublists are created by naively

traversing the input lists for all vectors with a Hamming weight of $H^{-1}(1-\lambda)\frac{\alpha_j m}{2}$ on the columns defined by the random partition. We only continue, if $\mathbf{L}'$ and $\mathbf{R}'$ don't grow too large, as defined in Lemma 3. In line 10, we apply the algorithm recursively on the subsequent $\alpha_2 m$ columns, and so on. Eventually, we compare the final list pairs naively.

Algorithm 4 NearestNeighborRec

1: **procedure** NearestNeighborRec($\mathbf{L}, \mathbf{R}, m, t, \gamma, \lambda, \alpha_1, \ldots, \alpha_t, y, \varepsilon, j$) ▷ init $j = 1$
2: **if** $j = t + 1$ **then**
3: run the naive algorithm to compute $\mathbf{C}$, a list of correct pairs
4: **end if**
5: **for** $\widetilde{\Theta}(2^{y\alpha_j m})$ times **do**
6: $A \leftarrow \mathsf{partition}(\alpha_j m)$ ▷ random partition of size $\frac{\alpha_j m}{2}$ of the $\alpha_j m$ columns
7: $\mathbf{L}' \leftarrow$ all $\mathbf{v_L} \in \mathbf{L}$ with wt. $H^{-1}(1-\lambda)\frac{\alpha_j m}{2}$ on A-columns ▷ naive search
8: $\mathbf{R}' \leftarrow$ all $\mathbf{v_R} \in \mathbf{R}$ with wt. $H^{-1}(1-\lambda)\frac{\alpha_j m}{2}$ on A-columns ▷ naive search
9: **if** $|\mathbf{L}'|, |\mathbf{R}'|$ don't grow too large **then** ▷ as defined in Lemma 3
10: $\mathbf{C} \leftarrow \mathbf{C} \cup$ NearestNeighborRec($\mathbf{L}', \mathbf{R}', m, t, \gamma, \lambda, \alpha_1, \ldots, \alpha_t, y, \varepsilon, j+1$)
▷ solve the problem recursively
11: **end if**
12: **end for**
13: **return C** ▷ output a list of correct pairs
14: **end procedure**

5 Analysis of Our Algorithm

In this section, we first show that NearestNeighbor achieves a good distribution in at least one of the polynomially many repetitions. We define a *good computation path* and show that NearestNeighbor has at least one of them with overwhelming probability over the coins of the algorithm. We continue by showing that the lists on that computation path achieve their expected size with overwhelming probability over the random choice of the input. We conclude with Theorem 1 that combines these results and shows the correctness and time complexity of NearestNeighbor.

Lemma 1 (good distribution). *Let $(\mathbf{L}, \mathbf{R}, \gamma)$ be an instance of an (m, γ, λ) Nearest Neighbor problem with unknown solution vectors $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L} \times \mathbf{R}$. Let $\mathbf{z}^* := \mathbf{u}^* + \mathbf{v}^*$ and for any constant t let $\mathbf{u}^* := (\mathbf{u}_1^*, \ldots, \mathbf{u}_t^*)$, $\mathbf{v}^* := (\mathbf{v}_1^*, \ldots, \mathbf{v}_t^*)$ $\mathbf{z}^* := (\mathbf{z}_1^*, \ldots, \mathbf{z}_t^*)$ be a splitting of the vectors in t strips with sizes $\alpha_j m$ for all $1 \le j \le t$ with $\alpha_1 + \ldots + \alpha_t = 1$. Then the double rerandomization of algorithm* NearestNeighbor *guarantees with overwhelming probability that*

$$\mathbf{wt}(\mathbf{z}_j^*) = \gamma\alpha_j m \quad \textit{and} \quad \mathbf{wt}(\mathbf{u}_j^*) = \mathbf{wt}(\mathbf{v}_j^*) = \tfrac{1}{2}\alpha_j m \quad \textit{for all } 1 \le j \le t$$

in at least one of the rerandomized input lists.

Proof. In the first loop, random permutations π of the m columns are chosen. Thus the probability for $\mathbf{wt}(\mathbf{z}_j^*) = \gamma\alpha_j m$ for all $1 \leq j \leq t$ is

$$\binom{\alpha_1 m}{\gamma\alpha_1 m} \cdot \ldots \cdot \binom{\alpha_t m}{\gamma\alpha_t m} / \binom{m}{\gamma m}.$$

A cancellation of the common terms, an application of Stirling's formula and the fact that t is constant shows the claim. In the second loop we choose uniformly random $\mathbf{r} = (\mathbf{r}_1, \ldots, \mathbf{r}_t) \in \mathbb{F}_2^{\alpha_1 m} \times \ldots \mathbb{F}_2^{\alpha_t m}$ with weight $\frac{1}{2}\alpha_j m$ on each of the t strips and add them to all elements in $\mathbf{L}$ and $\mathbf{R}$. Fix one of the strips j and consider the vectors $(\mathbf{u}_j^*, \mathbf{v}_j^*)$. Let c_{01} denote the number of columns such that $\mathbf{u}_j^*$ has a 0-coordinate and $\mathbf{v}_j^*$ has a 1-coordinate. Define c_{00}, c_{10} and c_{11} analogously. We define $\mathbf{r}$ to be good on the strip j, if it has exactly $\frac{1}{2}c_{xy}$ ones in all four parts xy. The probability for that is

$$\binom{c_{00}}{\frac{1}{2}c_{00}}\binom{c_{01}}{\frac{1}{2}c_{01}}\binom{c_{10}}{\frac{1}{2}c_{10}}\binom{c_{11}}{\frac{1}{2}c_{11}} / \binom{\alpha_j m}{\frac{1}{2}\alpha_j m}.$$

Notice that this is again inverse polynomial, because $c_{00} + c_{01} + c_{10} + c_{11} = \alpha_j m$ per definition. Thus the probability stays polynomial for all t strips, since t is constant.

We conclude that a good $\mathbf{r}$ solves the problem, because on each strip

$$\mathbf{wt}(\mathbf{u}_j^* + \mathbf{r}_j) = \mathbf{wt}(\mathbf{v}_j^* + \mathbf{r}_j) = \tfrac{1}{2}(c_{00} + c_{01} + c_{10} + c_{11}) = \tfrac{1}{2}\alpha_j m. \qquad \square$$

In the following we use the notion of a *good* computation path inside the computation tree of our algorithm. See Fig. 7 for an example. In this figure the good path is marked as $\rightarrow$, whereas all the other paths are marked as dashed arrows.

Definition 3 (good computation path). *Let $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L} \times \mathbf{R}$ be the target solution. A computation path of* NearestNeighbor *is called good, if $(\mathbf{u}^*, \mathbf{v}^*)$ is contained in all t sublist pairs from the root to a leaf.*

Lemma 2 (correctness). *Let $t \in \mathbb{N}$ be the (constant) depth of* NearestNeighbor *and $\lambda < 1 - H(\frac{\gamma}{2})$. Then the computation tree of* NearestNeighbor *has a good computation path with overwhelming probability over the coins of the algorithm.*

Proof. By construction, the target solution $(\mathbf{u}^*, \mathbf{v}^*)$ is contained in the initial list pair $\mathbf{L} \times \mathbf{R}$ on level 1 of the computation tree. In the following we show that if the solution is in one of the input lists on a level j, then with overwhelming probability it is also in one of the output lists on level j (which are either the input lists on level $j + 1$ or the input lists for the naive algorithm on the last level). Thus, if this holds for any $1 \leq j \leq t$, we have a *good* path by induction.

Let us create $2^{y\alpha_j m}$ sublist pairs for each input list pair on level j with y from (5). On each level j we therefore have a total of $2^{y(\alpha_1 + \ldots + \alpha_j)m}$ output list pairs, resulting in a total of 2^{ym} output list pairs on the last level.

Fix a level j and a target solution $(\mathbf{u}^*, \mathbf{v}^*)$ in one of the input pairs $\mathbf{L}, \mathbf{R}$ on that level. On this level, we work on a strip of size $\alpha_j m$. Let $\mathbf{u}_j^*$ and $\mathbf{v}_j^*$ be the restrictions of $\mathbf{u}^*$, resp. $\mathbf{v}^*$ to that strip. Due to the rerandomization in our algorithm, it is guaranteed by Lemma 1 that $\mathbf{wt}(\mathbf{u}_j^*) = \mathbf{wt}(\mathbf{v}_j^*) = \frac{1}{2}\alpha_j m$ and that their Hamming distance is $\Delta(\mathbf{u}_j^*, \mathbf{v}_j^*) = \gamma\alpha_j m$.

Thus, if we look at the pairwise coordinates of $(\mathbf{u}_j^*, \mathbf{v}_j^*)$ this implies that we have exactly $\frac{1-\gamma}{2}\alpha_j m$ $(0,0)$-pairs and $(1,1)$-pairs and exactly $\frac{\gamma}{2}\alpha_j m$ $(0,1)$-pairs and $(1,0)$-pairs, respectively. We illustrate this *input distribution* of the target pair $(\mathbf{u}_j^*, \mathbf{v}_j^*)$ in Fig. 8.

$$\begin{array}{ll} \mathbf{u}_j^* = & 0\ldots\ldots 0\ 0\ldots 0\ 1\ldots 1\ 1\ldots\ldots 1 \\ \mathbf{v}_j^* = & 0\ldots\ldots 0\ 1\ldots 1\ 0\ldots 0\ 1\ldots\ldots 1 \\ \text{weight} & \frac{1-\gamma}{2}\alpha_j m\ \ \frac{\gamma}{2}\alpha_j m\ \ \frac{\gamma}{2}\alpha_j m\ \ \frac{1-\gamma}{2}\alpha_j m \end{array}$$

Fig. 8. input distribution ($\alpha_j m$-strip)

The algorithm constructs sublists $\mathbf{L}', \mathbf{R}'$ by choosing a random partition A of the α_j-strip with $|A| = \frac{1}{2}\alpha_j m$. The algorithm only keeps those vectors of the input lists that have a relative Hamming weight of $h := H^{-1}(1-\lambda)$ on the columns defined by A, a choice that will be justified in Lemma 3. The choice of h implies that the number of $(1,0)$ overlaps on the columns defined by A plus the number of $(1,1)$ overlaps on the columns defined by A is $h\alpha_j\frac{m}{2}$. This is also the case for the number of $(0,1)$ overlaps plus the number of $(1,1)$ overlaps. Finally, we also know that the sum of all overlaps $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$ is $\alpha_j\frac{m}{2}$. Compared to the input distribution, this leaves one degree of freedom, which we denote by a parameter $0 \le c \le h$. We obtain the *output distribution* shown in Fig. 9.

$$\begin{array}{ll} \mathbf{u}_{j,A}^* = & 0\ldots\ldots\ldots\ldots\ldots 0\ 0\ldots 0\ 1\ldots 1\ 1\ldots\ldots\ldots\ldots 1 \\ \mathbf{v}_{j,A}^* = & 0\ldots\ldots\ldots\ldots\ldots 0\ 1\ldots 1\ 0\ldots 0\ 1\ldots\ldots\ldots\ldots 1 \\ \text{weight} & (1-h-c)\,\alpha_j\frac{m}{2}\ \ c\,\alpha_j\frac{m}{2}\ \ c\,\alpha_j\frac{m}{2}\ \ (h-c)\,\alpha_j\frac{m}{2} \end{array}$$

Fig. 9. output distribution (A-columns of an $\alpha_j m$-strip)

In order to compute the necessary number of repetitions, we have to compute the number of *good* partitions A that lead to an output distribution of Fig. 9 for any $0 \le c \le h$. This can be computed by multiplying the number of choices we have for each overlap of zeros and ones for any possible value of $c\,\alpha_j\frac{m}{2}$, which is

$$\sum_{c\alpha_j\frac{m}{2}=0}^{h\alpha_j\frac{m}{2}} \binom{\frac{1-\gamma}{2}\alpha_j m}{(1-h-c)\alpha_j\frac{m}{2}} \cdot \binom{\frac{\gamma}{2}\alpha_j m}{c\,\alpha_j\frac{m}{2}}^2 \cdot \binom{\frac{1-\gamma}{2}\alpha_j m}{(h-c)\alpha_j\frac{m}{2}}.$$

For a fixed c, it is for example possible to choose $c\,\alpha_j \frac{m}{2}$ (0,1) overlaps in the A-area from an overall number of $\frac{\gamma}{2}\alpha_j m$ (0,1)'s in the whole strip.

Notice that some choices of c might lead to no possible choice for A, e.g. if $c > \gamma$. We determine a c that maximizes the number of *good* partitions. Numerical optimization shows that this maximum is obtained at $c = \frac{\gamma}{2}$. Notice that this is a valid choice for c (i.e. none of the binomial coefficients is zero) due to the restriction $\lambda < 1 - H(\frac{\gamma}{2})$ of Theorem 1 that implies $h > \frac{\gamma}{2}$. Thus the number of good partitions (up to polynomial factors) is

$$\binom{\frac{1-\gamma}{2}\alpha_j m}{(1-h-\frac{\gamma}{2})\alpha_j \frac{m}{2}} \cdot \binom{\frac{\gamma}{2}\alpha_j m}{\frac{\gamma}{2}\alpha_j \frac{m}{2}}^2 \cdot \binom{\frac{1-\gamma}{2}\alpha_j m}{(h-\frac{\gamma}{2})\alpha_j \frac{m}{2}} = \widetilde{\mathcal{O}}\left((2^{\alpha_j m})^{\gamma+(1-\gamma)H(\frac{h-\frac{\gamma}{2}}{1-\gamma})}\right)$$

The total number of partitions is $\binom{\alpha_j m}{\frac{1}{2}\alpha_j m} = \widetilde{\mathcal{O}}(2^{\alpha_j m})$. Thus the expected number r of repetitions until we find the correct pair is the total number of partitions divided by the number of *good* partitions, which is $r = \widetilde{\mathcal{O}}(2^{y\alpha_j m})$, which justifies our choice of y in identity (5).

It suffices to choose mr repetitions in order to find the correct pair with overwhelming probability, since the probability to *not* find the correct pair can be upper bounded by

$$(1-1/r)^{mr} \leq 2^{-m}.$$

Thus the probability that the algorithm goes wrong in *any* of its $2t$ calls on a *good* computation path can be upper bounded by $2t \cdot 2^{-m}$, which is negligible. Notice that $mr = \widetilde{\mathcal{O}}(2^{y\alpha_j m})$, since polynomial factors vanish in $\widetilde{\mathcal{O}}$-notation. □

Lemma 3 (list sizes). *Let $t \in \mathbb{N}$ be the (constant) depth of* NearestNeighbor *and $\varepsilon > 0$. Consider a good computation path of depth t. Then with overwhelming probability the $2t$ lists inside the good computation path have sizes $\widetilde{\mathcal{O}}((2^{\lambda m})^{1-\sum_{i=1}^{j}\alpha_i+\frac{\varepsilon}{2}})$ for all $1 \leq j \leq t$ and thus are not cut off by the algorithm.*

Proof. Fix some $1 \leq j \leq t$ and an initial input list $\mathbf{L}$ with $|\mathbf{L}| = 2^{\lambda m}$ (the argument is analogous for $\mathbf{R}$). For each vector $\mathbf{v}_k \in \mathbf{L}$ we define random variables X_k such that

$$X_k = \begin{cases} 1 & \text{if } \bigwedge_{i=1}^{j} \mathbf{v}_k \in \mathbf{L^i} \\ 0 & \text{otherwise} \end{cases}.$$

Let $X := \sum_{k=1}^{|\mathbf{L}|} X_k$. Thus the random variable X counts the number of elements in the output list $\mathbf{L^j}$. Recall that NearestNeighbor restricts to relative weight $h = H^{-1}(1-\lambda)$ on the columns in A. Since we know that the computation path is *good*, there is one $\mathbf{v}_{k^*} \in \mathbf{L}$ with $\mathbb{P}[X_{k^*} = 1] = 1$. Notice that all the other elements are independent of $\mathbf{v}_{k^*}$ and are uniformly chosen among all vectors with weight $\alpha_j \frac{m}{2}$ on each strip j. Thus for all $\mathbf{v}_k \in \mathbf{L} \setminus \{\mathbf{v}_{k^*}\}$ we have

$$\mathbb{P}[X_k = 1] = \prod_{i=1}^{j} \binom{\frac{\alpha_i m}{2}}{h\frac{\alpha_i m}{2}} \binom{\frac{\alpha_i m}{2}}{(1-h)\frac{\alpha_i m}{2}} / \binom{\alpha_i m}{\frac{\alpha_i m}{2}},$$

which is, for each of the first j strips, the number of vectors that have relative weight h on the A-columns divided by the number of all possible vectors. Thus the expected size of the output list is

$$\mathbb{E}[X] = 1 + (2^{\lambda m} - 1) \cdot \prod_{i=1}^{j} \binom{\frac{\alpha_i m}{2}}{h\frac{\alpha_i m}{2}} \binom{\frac{\alpha_i m}{2}}{(1-h)\frac{\alpha_i m}{2}} / \binom{\alpha_i m}{\frac{\alpha_i m}{2}}.$$

Notice that obviously $\mathbb{E}[X] \geq 1$. Applying Chebyshev's inequality, we get

$$\mathbb{P}\left[|X - \mathbb{E}[X]| \geq 2^{\frac{\varepsilon}{2}m}\mathbb{E}[X]\right] \leq \frac{\mathbb{V}[X]}{2^{\varepsilon m}\mathbb{E}[X]^2} \leq \frac{1}{2^{\varepsilon m}\mathbb{E}[X]} \leq 2^{-\varepsilon m},$$

using $\mathbb{V}[X] = \mathbb{V}[\sum_k X_k] = \sum_k \mathbb{V}[X_k] = \sum_k (\mathbb{E}[X_k^2] - \mathbb{E}[X_k]^2) \leq \sum_k \mathbb{E}[X_k] = \mathbb{E}[X]$. We have $\mathbb{V}[\sum_k X_k] = \sum_k \mathbb{V}[X_k]$, because the X_k are pairwise independent. Thus (for both lists on each level) we obtain $\mathbb{P}\left[X\text{too large in any of the steps}\right] \leq 2t \cdot 2^{-\varepsilon m}$, applying the union bound.

From Stirling's formula it also follows that $\mathbb{E}[X] \leq (2^{\lambda m})^{1-\sum_{i=1}^{j}\alpha_i}$ for each $1 \leq j \leq t$. Hence, with overwhelming probability, the list sizes are as claimed. □

Now we are able to prove Theorem 1.

Theorem 1. *For any constant $\varepsilon > 0$ and any $\lambda < 1 - H(\frac{\gamma}{2})$,* NearestNeighbor *solves the (m, γ, λ) NN problem with overwhelming probability (over both the coins of the algorithm and the random choice of the input) in time*

$$\widetilde{\mathcal{O}}\left(2^{(y+\varepsilon)m}\right) \quad \textit{with} \quad y := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda) - \frac{\gamma}{2}}{1-\gamma}\right)\right).$$

Proof. By Lemma 2, there is a path that includes the solution (with overwhelming probability over the coins of the algorithm). Fix that path. We show that by Lemma 3 (with overwhelming probability over the random choice of the input) NearestNeighbor's time complexity is the maximum of the times

$$\widetilde{\mathcal{O}}\left((2^m)^{\lambda(1-\sum_{i=1}^{j-1}\alpha_i)+y\sum_{i=1}^{j}\alpha_i+\frac{\varepsilon}{2}}\right) \tag{6}$$

to create all sublists on level j for all levels $1 \leq j \leq t$ and the time

$$\widetilde{\mathcal{O}}(2^{(y+\varepsilon)m}) \tag{7}$$

to naively solve the problem on the last level.

From Lemma 3 we know that on each level j the sizes of the *input* lists are $\widetilde{\mathcal{O}}((2^m)^{\lambda(1-\sum_{i=1}^{j-1}\alpha_i)+\frac{\varepsilon}{2}})$. Notice that if the lists grow too large, we simply abort. On level j we construct $\widetilde{\mathcal{O}}(2^{y\alpha_j m})$ new sublists for each input list so that we have a total number of $\widetilde{\mathcal{O}}((2^{ym})^{\sum_{i=1}^{j}\alpha_i})$ sublists on this level. Each of these sublists is computed by naively searching through the input lists. Thus, we have

to multiply the sizes of the input lists with the number of sublists which results in complexity (6).

In NearestNeighbor's last step we use the naive algorithm to join a total number of $\widetilde{\mathcal{O}}(2^{ym})$ pairs of sublists of size $\widetilde{\mathcal{O}}(2^{\frac{\varepsilon}{2}m})$ each, resulting in complexity (7). The overall complexity is the maximum of complexities (6) and (7).

We want to continue by computing the overall time complexity of the steps defined by (6) for any fixed t by setting the complexities of (6) equal. Thus we choose

$$\lambda\left(1-\sum_{i=1}^{j-1}\alpha_i\right)+y\sum_{i=1}^{j}\alpha_i+\frac{\varepsilon}{2}=\lambda\left(1-\sum_{i=1}^{j}\alpha_i\right)+y\sum_{i=1}^{j+1}\alpha_i+\frac{\varepsilon}{2}$$

for all $1 \leq j \leq t-1$, which implies $\alpha_{j+1} = (\lambda/y) \cdot \alpha_j$. Additionally, we need $\sum_{i=1}^{t} \alpha_i = 1$, thus using $y > \lambda$ from Eq. (5) we get $1 = \sum_{i=1}^{t} \alpha_i = \alpha_1 \cdot \sum_{i=0}^{t-1} (\lambda/y)^i = \alpha_1 \cdot \frac{1-(\lambda/y)^t}{1-(\lambda/y)}$. This implies $\alpha_1 = \frac{1-(\lambda/y)}{1-(\lambda/y)^t}$ and finally (uniquely) determines all $(\alpha_1, \ldots, \alpha_t)$. Since by our choice all complexities (6) are the same, we get an overall running time of $\widetilde{\mathcal{O}}((2^m)^{\lambda+y\cdot\alpha_1+\frac{\varepsilon}{2}})$.

Notice that the (constant) choice $t = \left\lceil \frac{\log(y-\lambda+\frac{\varepsilon}{2}) - \log(\frac{\varepsilon}{2})}{\log(y)-\log(\lambda)} \right\rceil \in \mathbb{N}$ leads to $\alpha_1 = \frac{y-\lambda+\frac{\varepsilon}{2}}{y}$ and we finally obtain the same complexity $\widetilde{\mathcal{O}}(2^{(y+\varepsilon)m})$ as in (7), which makes (7) the time complexity of the whole algorithm.

We want to conclude the proof by showing that for any fixed t it is indeed optimal to set all time complexities in (6) equal. Let $(\alpha_1, \ldots, \alpha_t)$ be the (unique) choice defined above. Assume this is not optimal, thus a choice $(\widetilde{\alpha}_1, \ldots, \widetilde{\alpha}_t)$ that is different from the first one improves upon the overall time complexity. Decreasing one of the time complexities implies that there is a $1 \leq k \leq t$ with $\widetilde{\alpha}_k < \alpha_k$, because $y > \lambda$. Let k be minimal with that property.

Case 1: There is an $1 \leq \ell < k$ s.t. $\widetilde{\alpha}_\ell > \alpha_\ell$ and let ℓ be minimal with that property. Then $\sum_{i=1}^{\ell-1} \widetilde{\alpha}_i = \sum_{i=1}^{\ell-1} \alpha_i$. Thus $\lambda + (y-\lambda) \cdot \sum_{i=1}^{\ell-1} \widetilde{\alpha}_i + y \cdot \widetilde{\alpha}_\ell + \frac{\varepsilon}{2} > \lambda + (y-\lambda) \cdot \sum_{i=1}^{\ell-1} \alpha_i + y \cdot \alpha_\ell + \frac{\varepsilon}{2}$.

Case 2: Otherwise, we know that $\sum_{i=1}^{k-1} \widetilde{\alpha}_i = \sum_{i=1}^{k-1} \alpha_i$ and thus $\sum_{i=1}^{k} \widetilde{\alpha}_i < \sum_{i=1}^{k} \alpha_i$. Notice that it also has to hold that $\sum_{i=1}^{t} \widetilde{\alpha}_i = \sum_{i=1}^{t} \alpha_i = 1$. Hence there is an $k < \ell \leq t$ s.t. $\sum_{i=1}^{\ell-1} \widetilde{\alpha}_i < \sum_{i=1}^{\ell-1} \alpha_i$ and $\sum_{i=1}^{\ell} \widetilde{\alpha}_i \geq \sum_{i=1}^{\ell} \alpha_i$. Thus $\lambda - \lambda \cdot \sum_{i=1}^{\ell-1} \widetilde{\alpha}_i + y \cdot \sum_{i=1}^{\ell} \widetilde{\alpha}_i > \lambda - \lambda \cdot \sum_{i=1}^{\ell-1} \alpha_i + y \cdot \sum_{i=1}^{\ell} \alpha_i$.

In both cases the time complexity on some level $\ell \neq k$ (and therefore the overall time complexity) strictly increases, which makes the new choice inferior to the original one.

□

Acknowledgment

We would like to thank the anonymous Eurocrypt reviewers for their helpful and detailed comments that improved and clarified our work.

Open Problem

Our NearestNeighbor algorithm uses a recursion tree of constant depth t. This leads to a large *polynomial* blow-up for our decoding algorithm (in the size of m^t), which asymptotically vanishes but in practice might lead to an undesirably large break-even point with the BJMM algorithm. We pose it as an open problem to get rid of this polynomial overhead.

References

1. Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf. Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs. In *Discrete & Computational Geometry 6*, pages 407-422, 1991.
2. M. Alekhnovich. More on Average Case vs Approximation Complexity. In *44th Symposium on Foundations of Computer Science (FOCS), pages 298–307, 2003*
3. A. Becker, J.-S. Coron, and A. Joux. Improved generic algorithms for hard knapsacks. In *EUROCRYPT*, pages 364–385, 2011.
4. A. Becker, A. Joux, A. May, and A. Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *EUROCRYPT*, pages 520–536, 2012.
5. Zvika Brakerski and Vinod Vaikuntanathan Efficient Fully Homomorphic Encryption from (Standard) LWE In *FOCS*, pages 97–106, 2011.
6. Moshe Dubiner Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. In *IEEE Transactions on Information Theory 56(8)*, pages 4166-4179, 2010.
7. Craig Gentry, Chris Peikert, Vinod Vaikuntanathan Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.
8. Sariel Har-Peled, Piotr Indyk, Rajeev Motwani Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In *Theory of Computing 8(1)*, pages 321-350, 2012.
9. N. J. Hopper and M. Blum. Secure human identification protocols. In *ASIACRYPT*, pages 52–66, 2001.
10. E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974.
11. N. Howgrave-Graham and A. Joux. New generic algorithms for hard knapsacks. In *EUROCRYPT*, pages 235–256, 2010.
12. E. Kiltz, K. Pietrzak, D. Cash, A. Jain and D. Venturi. Efficient Authentication from Hard Learning Problems. In *Advances in Cryptology - EUROCRYPT 2011*, pages 7-26. Springer, 2001.
13. P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Advances in Cryptology - EUROCRYPT 1988*, pages 275–280, 1988.

14. A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *ASIACRYPT*, pages 107–124, 2011.
15. R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. In *Jet Propulsion Laboratory DSN Progress Report* 42–44, pages 114–116, 1978.
16. Chris Peikert, Brent Waters Lossy trapdoor functions and their applications. In *STOC*, pages 187–196, 2008.
17. C. Peters. Information-Set Decoding for Linear Codes over $\mathbb{F}_q$. In *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010*, pages 81-94. Springer, 2010.
18. E. Prange. The Use of Information Sets in Decoding Cyclic Codes. *IRE Transaction on Information Theory*, volume 8, issue 5, pages 5-9, 1962.
19. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
20. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.
21. J. Stern. A method for finding codewords of small weight. In *Proceedings of the 3rd International Colloquium on Coding Theory and Applications*, pages 106–113, London, UK, 1989. Springer-Verlag.
22. M. Sudan. Algorithmic Introduction to Coding Theory. Lecture Notes (available online), 2001.
23. A. Andoni, P. Indyk, H. L. Nguyen and I. Razenshteyn. Beyond Locality-Sensitive Hashing. In *SODA*, pages 1018–1028, 2014.
24. G. Valiant. Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas. In *FOCS*, pages 11–20, 2012.