Improved Quantum Information Set Decoding

Elena Kirshanova

Laboratoire LIP, ENS de Lyon
elena.kirshanova@ens-lyon.fr

A flaw was found in the results of the current Section 3 thanks to A.Helm (alexander.helm@ruhr-uni-bochum.de). So the current best quantum complexities for MMT and BJMM information set decoding algorithms remain as in Kachigar-Tillich.

Abstract. In this paper we present quantum information set decoding (ISD) algorithms for binary linear codes. First, we refine the analysis of the quantum walk based algorithms proposed by Kachigar and Tillich (PQCrypto'17). This refinement allows us to improve the running time of quantum decoding in the leading order term: for an n-dimensional binary linear code the complexity of May-Meurer-Thomae ISD algorithm (Asiacrypt'11) drops down from $2^{0.05904n+o(n)}$ to $2^{0.05806n+o(n)}$. Similar improvement is achieved for our quantum version of Becker-Jeux-May-Meurer (Eurocrypt'12) decoding algorithm. Second, we translate May-Ozerov Near Neighbour technique (Eurocrypt'15) to an 'update-and-query' language more common in a similarity search literature. This re-interpretation allows us to combine Near Neighbour search with the quantum walk framework and use both techniques to improve a quantum version of Dumer's ISD algorithm: the running time goes down from $2^{0.059962n+o(n)}$ to $2^{0.059450+o(n)}$.

Keywords information set decoding, quantum walk, near neighbour.

1 Introduction

The Information Set Decoding problem with integer parameters n, k, d asks to find the error-vector $\mathbf{e} \in \mathbb{F}_2^n$ given a matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ and a vector $\mathbf{s} = \mathbf{H}\mathbf{e}^{\mathbf{t}}$ such that the Hamming weight of $\mathbf{e}$, denoted $w := wt(\mathbf{e})$, is bounded by some integer. The matrix $\mathbf{H}$ is called the parity-check matrix of a binary linear $[n, k, d]$-code $\mathcal{C}$, where d is the minimum distance of the code. In this work, we stick to the so-called *full distance decoding* setting, i.e., when we search for $\mathbf{e}$ with $wt(\mathbf{e}) \leq d$. The analysis is easy to adapt to *half-distance decoding*, i.e., when $wt(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$.

The ISD problem is relevant not only in coding theory but also in cryptography: several cryptographic constructions, e.g. [McE78], rely on the hardness of ISD. The problem seems to be intractable even for quantum computers, which makes these constructions attractive for post-quantum cryptography.

First classical ISD algorithm due to Prange dates back to 1962 [Pra62] followed by a series of improvements [Ste89,Dum91,FS09,MMT11,BJMM12], culminating in algorithms [MO15,BM17] that rely on Nearest Neighbour techniques

in Hamming metric. On the quantum side, the ISD problem received much less attention: Bernstein in [Ber10] analysed a quantum version of Prange's algorithm, and recently Kachigar and Tillich [KT17] gave a series of ISD algorithms based on quantum walks. The results presented here extend and improve upon the work of [KT17].

Our contributions:

1. We describe quantum versions of all known ISD algorithms, which have better asymptotical running times than algorithms given in [KT17]. We also explain why for certain algorithms the analysis of [KT17] is sub-optimal;
2. We re-phrase May-Ozerov Near Neighbour algorithm [MO15] in the 'update-and-query' language and give a method to analyse its complexity;
3. We present a quantum version of the May-Ozerov ISD algorithm.

Our second contribution is of independent interest as it provides an alternative but more flexible view on May-Ozerov Near Neighbour algorithm for the Hamming metric. We give simple formulas for analysing its complexity which allow us to stay in the Hamming space, i.e., without reductions from other metrics as it is usually done in the similarity search literature [Chr17]. The third contribution answers the problem left open in [KT17], namely, how to use Near Neighbour technique within quantum walks. Our results are summarized below.

Algorithm	Quantum		Classical	
	Time	**Space**	**Time**	**Space**
Prange [Ber10,Pra62]	0.060350	$--$	0.120600	$--$
Stern/Dumer [Ste89,Dum91]				
+ Shamir-Schroeppel (SS) [KT17]	0.059697	0.00618	0.116035	0.03644
+ Near Neighbour (NN) Sect.4	0.059922	0.00897	0.113762	0.04248
+ SS + NN Sect.4	0.059450	0.00808		
MMT [MMT11]				
– Kachigar-Tillich [KT17]	0.059037	0.01502	0.111468	0.05408
– Sect. 3	0.058059	0.01849		
BJMM [BJMM12]				
– Kachigar-Tillich [KT17]	0.058696	0.01877	0.101998	0.07590
– Sect. 3	0.058040	0.01866		

Table 1: Running time and space complexities of ISD algorithms (full distance decoding). The columns give the exponent-constants c, i.e., runtime and memory complexities are of the form $\mathcal{O}(2^{cn})$. For Prange's algorithm, the space is $\mathrm{poly}(n)$.

For each classical algorithm, Table 1 gives running times and space complexities of their quantum counterparts. This work improves over Kachigar-Tillich quantum versions of MMT in time, and of BJMM in both time and space. The improvement comes from different analysis: as explained in Sect. 2, the (asymptotical) complexity of these two algorithms is the maximum between the runtimes of several subroutines. The authors in [KT17] assume that the maximum

is attained at a certain subroutine and, during the derivation of the algorithm's parameters, enforce that the runtimes of all other subroutines remain below. On the contrary, we assume that another subroutine is dominant. By the 'quantum space' in Table 1, we mean the number of qubits in a quantum state an algorithm operates on.

In Sect. 4 we show how to combine the Near Neighbour search of May and Ozerov [MO15] with quantum version of the ISD algorithm due to Dumer [Dum91]. By itself, this does not improve over the algorithm given in [KT17]. But, combined with the so-called Shamir-Schroeppel trick [SS81] already used in [KT17], we can slightly improve the runtime complexity of the algorithm.

We note that, as in the classical setting, the Near Neighbour technique requires more memory, but we are still far from the Time=Memory regime. It turns out that, as opposed to the classical case, quantum Near Neighbour search does not improve MMT or BJMM. We argue why this is the case at the and of Sect. 4. We leave as an open problem an application of quantum Near Neighbour to MMT/BJMM algorithms as well as quantum speed-ups for algorithm described in [BM17].

2 Preliminaries

We start with overview on classical algorithms for ISD, namely, Prange [Pra62], Stern and its variants [Ste89,Dum91], MMT [MMT11], and BJMM [BJMM12] algorithms. We continue with known quantum speed-ups for these algorithms.

2.1 Classical ISD algorithms

All known ISD algorithms try to find the error-vector $\mathbf{e}$ by a clever enumeration of the search space for $\mathbf{e}$, which is of size $\binom{n}{w} \approx 2^{n \cdot H\left(\frac{w}{n}\right)}$, where $H(x) = -x \log x - (1-x) \log(1-x)$ is the binary entropy function. In the analysis of ISD algorithms, it is common to relate the parameters w (the error-weight), and k (the rank of a code) to dimension n, and simplify the running times to the form 2^{cn} for some constant c.[1] To do this, we make use of Gilbert-Varshamov bound which states that $\frac{k}{n} = 1 - H\left(\frac{w}{n}\right)$ as $n \to \infty$. This gives us a way to express w as a function of n and k. Finally, the running time of an ISD algorithm is obtained by a brute-force search over all $\frac{k}{n} \in [0, \frac{1}{2}]$ (up to some precision) that leads to the worst-case complexity. In the classical setting, this worst-case is reached by codes of rate $\frac{k}{n} \approx 0.447$, while in the quantum regime it is $\frac{k}{n} \approx 0.45$.

Decoding algorithms start by permuting the columns of $\mathbf{H}$ which is equivalent to permuting the positions of 1's in $\mathbf{e}$. The goal is to find a permutation $\pi \in S_n$ such that $\pi(\mathbf{e})$ has exactly $p \leq w$ 1's on the first k coordinates and the remaining weight of $w - p$ is distributed over the last $n - k$ coordinates. All known ISD

---

[1] We omit sub-exponential in n factors throughout, because we are only interested in the constant c. Furthermore, our analysis is for an average case and we sometimes omit the word 'expected'.

algorithms make use of the fact that such a permutation is found. We expect to find a good π after $\mathcal{P}(p)$ trials, where

$$\mathcal{P}(p) = \frac{\binom{k}{p}\binom{n-k}{w-p}}{\binom{n}{w}}. \tag{1}$$

The choice of p and how we proceed with $\pi(\mathbf{H})$ depends on the ISD algorithm.

For example, *Prange's* algorithm [Pra62] searches for a permutation π that leads to $p = 0$. To check whether a candidate π is good, it transforms $\pi(\mathbf{H})$ into systematic form $[\mathbf{Q} \mid \mathbf{I}_{n-k}]$ (provided the last $n - k$ columns of $\pi(\mathbf{H})$ form an invertible matrix which happens with constant success probability). The same transformation is applied to the syndrome $\mathbf{s}$ giving a new syndrome $\bar{\mathbf{s}}$. From the choice of p, it is easy to see that for a good π, we just 'read-off' the error-vector from the new syndrome, i.e., $\pi(\mathbf{e}) = \bar{\mathbf{s}}$, and to verify a candidate π, we check if $wt(\bar{\mathbf{s}}) = w$. We expect to find a good π after $\mathcal{P}(0)$ trials.

From now on, we assume that we work with systematic form of $\mathbf{H}$, i.e.

$$[\mathbf{Q} \mid \mathbf{I}_{n-k}] \cdot \mathbf{e} = \bar{\mathbf{s}} \quad \text{for} \quad \mathbf{Q} \in \mathbb{F}_2^{n-k \times k}. \tag{2}$$

Other than restricting the weight of $\mathbf{e}$ to be 0 on the last $n - k$ coordinates, we may as well allow $p > 0$ at the price of a more expensive check for π. This is the choice of *Stern's* algorithm [Ste89], which was later improved in [Dum91] (see also [FS09]). We describe the improved version. We start by adjusting the systematic form of $\mathbf{H}$ introducing the ℓ-length 0-window, so that Eq. 2 becomes

$$\left[\mathbf{Q} \,\middle|\, \frac{\mathbf{0}}{\mathbf{I}_{n-k-\ell}}\right] \cdot \mathbf{e} = \bar{\mathbf{s}} \quad \text{for} \quad \mathbf{Q} \in \mathbb{F}_2^{n-k \times k+\ell}. \tag{3}$$

Now we search for a permutation π that splits the error as

$$\mathbf{e} = [\mathbf{e}_1 \| \mathbf{0}^{\frac{k+\ell}{2}} \| \mathbf{0}^{n-k-\ell}] + [\mathbf{0}^{\frac{k+\ell}{2}} \| \mathbf{e}_2 \| \mathbf{0}^{n-k}] + [\mathbf{0}^{k+\ell} \| \mathbf{e}_3],$$

such that $wt(\mathbf{e}_1) = wt(\mathbf{e}_2) = p/2$ and $wt(\mathbf{e}_3) = w - p$, where $\mathbf{e}_i$'s are of appropriate dimensions. With such an $\mathbf{e}$, we can re-write Eq. (3) as

$$\mathbf{Q} \cdot [\mathbf{e}_1 \| \mathbf{0}^{\frac{k+\ell}{2}}] + \mathbf{Q} \cdot [\mathbf{0}^{\frac{k+\ell}{2}} \| \mathbf{e}_2] = \bar{\mathbf{s}} + \mathbf{e}_3. \tag{4}$$

We enumerate all possible $\binom{(k+\ell)/2}{p/2}$ vectors of the form $\mathbf{v}_1 := \mathbf{Q}[\mathbf{e}_1 \| \mathbf{0}^{\frac{k+\ell}{2}}]$ into a list $\mathcal{L}_1$ and all vectors of the form $\mathbf{v}_2 := \mathbf{Q}[\mathbf{0}^{\frac{k+\ell}{2}} \| \mathbf{e}_2] + \bar{\mathbf{s}}$ into a list $\mathcal{L}_2$. The above equation tells us that for the correct pair $(\mathbf{e}_1, \mathbf{e}_2)$, the sum of the corresponding list-vectors equals to $\mathbf{0}$ on the first ℓ-coordinates. We search for two vectors $\mathbf{v}_1 \in \mathcal{L}_1, \mathbf{v}_2 \in \mathcal{L}_2$ that are equal on this ℓ-window. We call such pair $(\mathbf{v}_1, \mathbf{v}_2)$ a *match*. We check among these matches if the Hamming distance between $\mathbf{v}_1, \mathbf{v}_2$, denoted $dt(\mathbf{v}_1, \mathbf{v}_2)$, is $w - p$. To retrieve the error-vector, we store $\mathbf{e}_i$'s together with the corresponding $\mathbf{v}_i$'s in the lists. The probability of finding a permutation that meets all the requirements is

$$\mathcal{P}(p, \ell) = \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{w-p}}{\binom{n}{w}}. \tag{5}$$

It would be more precise to have $\left(\binom{(k+\ell)/2}{p/2}\right)^2$ instead of $\binom{k+\ell}{p}$ in the above formula, but these two quantities differ by only a factor of $\mathrm{poly}(n)$ which we ignore. The expected running time of the algorithm is then

$$T = \mathcal{P}(p,\ell)^{-1} \cdot \max\left\{|\mathcal{L}_2|, \frac{|\mathcal{L}_1| \cdot |\mathcal{L}_2|}{2^\ell}\right\}, \tag{6}$$

where the first argument of max is the time to sort $\mathcal{L}_2$, the second is the expected number of pairs from $\mathcal{L}_1 \times \mathcal{L}_2$ that are equal on ℓ, which we check for a solution. See Fig. 1 for an illustration of the algorithm.
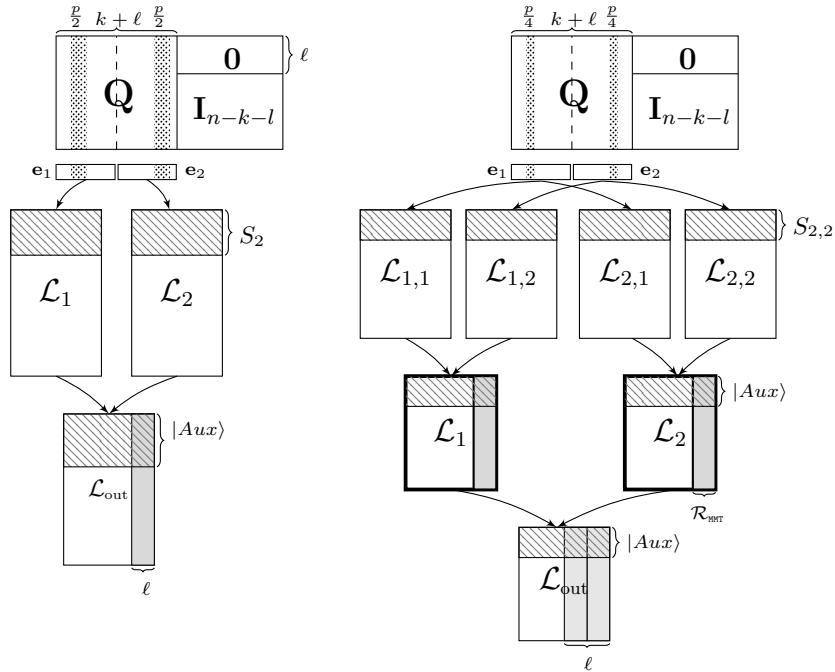

Fig. 1: *On the left*: A variant of Stern's ISD algorithm due to Dumer [Dum91]. The list $\mathcal{L}_1$ is constructed from all possible $p/2$-weight vectors $\mathbf{e}_1 \in \mathbb{F}_2^{(k+\ell)/2}$: $\mathcal{L}_1 = \{(\mathbf{e}_1, \mathbf{Q} \cdot [\mathbf{e}_1 || \mathbf{0}^{(k+\ell)/2}])\}$. $\mathcal{L}_2$ is constructed similarly with $\mathbf{0}^{(k+\ell)/2}$ and $\mathbf{e}_1$ swapped. Gray-shaded vertical strip indicates the coordinates on which the elements $\mathbf{v}_1 \in \mathcal{L}_1$ and $\mathbf{v}_2 \in \mathcal{L}_2$ match. Line-shaded horizontal strips indicate a subset of lists stored on quantum registers during the execution of quantum walk search algorithm.
On the right: May-Meurer-Thomae decoding [MMT11]. The lists $\mathcal{L}_1, \mathcal{L}_2$ are shorter than in Dumer's algorithm as their elements already match on $\mathcal{R}_{\text{MMT}}$-coordinates. It is the creation of the lists $\mathcal{L}_1, \mathcal{L}_2$ (indicated in bold) that dominates over the creation of $\mathcal{L}_{i,j}, i,j \in \{1,2\}$ and of $\mathcal{L}_{\text{out}}$ and hence, determines the running time of the algorithm.

The Representation technique of [BJMM12,MMT11] further improves the search for matching vectors by constructing the lists $\mathcal{L}_1, \mathcal{L}_2$ faster. Now the list $\mathcal{L}_1$ consists of all pairs $(\mathbf{e}_1, \mathbf{Q}\mathbf{e}_1)$ where $\mathbf{e}_1 \in \mathbb{F}_2^{k+\ell}$ (as opposed to $\mathbf{e}_1 \in \mathbb{F}_2^{(k+\ell)/2}$) with $wt(\mathbf{e}_1) = p/2$. Similarly, $\mathcal{L}_2 = \{(\mathbf{e}_2, \mathbf{Q}\mathbf{e}_2) \mid \forall \mathbf{e}_2 \in \mathbb{F}_2^{k+\ell}, wt(\mathbf{e}_2) = p/2\}$. The key observation is that now there are $\mathcal{R}_{\text{MMT}} := \binom{p}{p/2}$ ways to represent the target $\mathbf{e}$ as $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$. Hence, it is enough to construct only an $\mathcal{R}_{\text{MMT}}$−fraction of $\mathcal{L}_1, \mathcal{L}_2$. Such a fraction of $\mathcal{L}_1$ (analogously, for $\mathcal{L}_2$) is built by merging in the meet-in-the-middle way yet another two lists $\mathcal{L}_{1,1}$ and $\mathcal{L}_{1,2}$ filled with vectors of the form $\mathbf{Q}[\mathbf{e}_{1,1}||\mathbf{0}^{(k+\ell)/2}]$ (for $\mathcal{L}_{1,1}$) and $\mathbf{Q}[\mathbf{0}^{(k+\ell)/2}||\mathbf{e}_{1,2}]$ (for $\mathcal{L}_{1,2}$) for all $p/4$-weight $\mathbf{e}_{1,1}$ and $\mathbf{e}_{1,2}$, respectively. These starting lists are of size

$$|L_{i,j}| = \binom{(k+\ell)/2}{p/4}, \; i,j \in \{1,2\}. \tag{7}$$

During the merge, we force vectors from $\mathcal{L}_{1,1}$ be equal to vectors from $\mathcal{L}_{1,2}$ on $\log \mathcal{R}_{\text{MMT}}$ coordinates leaving only one (in expectation) pair $(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ whose sum gives $\mathbf{e}$ (see Fig. 1, right). Here and later, we shall abuse notations slightly: technically, the list elements are pairs $(\mathbf{e}, \mathbf{Q}\mathbf{e})$, but the merge is always done on the second element, and the error retrieval is done on the first.

The number of necessary permutations we need to try is given by Eq. (5). Provided a good π is found, the time to find the correct $\mathbf{e}$ is now given by $\max\{|L_{1,1}|, |L_{1,1}|^2/2^{\mathcal{R}_{\text{MMT}}}, |L_{1,1}|^4/2^{\ell+\mathcal{R}_{\text{MMT}}}\}$. This is the maximum between (I) the size of starting lists, (II) the size of the output after the first merge on $\log \mathcal{R}_{\text{MMT}}$ coordinates, and (III) the size of the final output after merging on the remaining $\ell - \log \mathcal{R}_{\text{MMT}}$ coordinates. Optimization for p, ℓ reveals that (II) is the maximum. Overall, the expected complexity of MMT is

$$T_{\text{MMT}} = \mathcal{P}(p, l) \cdot \frac{|L_{i,j}|^2}{\mathcal{R}_{\text{MMT}}}. \tag{8}$$

Becker-Jeux-May-Meurer in [BJMM12] further improves the merging step (i.e., the dominant one) noticing that zero-coordinates of $\mathbf{e}$ can be split in $\mathbf{e}_1, \mathbf{e}_2$ not only as $0+0$, but also as $1+1$. It turns out that constructing longer starting lists $\mathcal{L}_{i,j}$ using $\mathbf{e}_i$ of weights $wt(\mathbf{e}_i) = p/2 + \varepsilon$ is profitable as it significantly increases the number of representations from $\binom{p}{p/2}$ to $\mathcal{R}_{\text{BJMM}} := \binom{p}{p/2}\binom{k+\ell-p}{\varepsilon}$, thus allowing a better balance between the two merges: the first merge on $\log \mathcal{R}_{\text{BJMM}}$ coordinates and the second on $\ell - \log \mathcal{R}_{\text{BJMM}}$ coordinates. The expected running time of the BJMM algorithm is given by

$$T_{\text{BJMM}} = \mathcal{P}(p, l) \cdot \frac{|L_{i,j}|^2}{\mathcal{R}_{\text{BJMM}}}, \quad \text{where } |L_{i,j}| = \binom{(k+\ell)/2}{p/4 + \varepsilon}. \tag{9}$$

In fact, the actual BJMM algorithm is slightly more complicated than we have described, but the main contribution comes from adding '1+1' to representations, so hereafter we refer to this simplified version as BJMM.

6

2.2 Quantum ISD algorithms

Quantum ISD using Grover's algorithm. To speed-up Prange's algorithm, Bernstein in [Ber10] uses Grover's search over the space of permutations, which is of size $\mathcal{P}(0) = \binom{w}{k}/\binom{n}{w}$. This drops the expected runtime from $2^{0.1206n}$ (classical) down to $2^{0.06035n}$ (quantum), cf. Table 1. The approach has an advantage over all the quantum algorithms we discuss later as it requires quantum registers to store data of only $\text{poly}(n)$ size.

To obtain a quantum speed-up for other ISD algorithms like Stern's, MMT, BJMM, we need to describe quantum walks.

Quantum walks. At the heart of the above ISD algorithms (except Prange's) is the search for vectors from given lists that satisfy a certain relation. This task can be generalized to the k-list matching problem.

Definition 1 (k-list matching problem). *Let k be fixed. Given k equal sized lists $\mathcal{L}_1, \ldots, \mathcal{L}_k$ of binary vectors and a function g that decides whether a k-tuple $(\mathbf{v}_1, \ldots, \mathbf{v}_k) \in \mathcal{L}_1 \times \ldots \times \mathcal{L}_k$ forms a 'match' or not (outputs 1 in case of a 'match'), find all k-tuples $(\mathbf{v}_1, \ldots, \mathbf{v}_k) \in \mathcal{L}_1 \times \ldots \times \mathcal{L}_k$ s.t. $g(\mathbf{v}_1, \ldots, \mathbf{v}_k) = 1$.*

For example, the Stern's algorithm uses $k = 2$ and its g decides for a 'match' whenever a pair $(\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ is equal on certain fixed ℓ coordinates. For MMT or BJMM, we deal with four lists $\mathcal{L}_1, \ldots, \mathcal{L}_4$, and function g decides for the match if $\mathbf{v}_1 + \mathbf{v}_2, \mathbf{v}_3 + \mathbf{v}_4$ are equal on a certain part of coordinates (merge of $\mathcal{L}_1$ with $\mathcal{L}_2$, and $\mathcal{L}_3$ with $\mathcal{L}_4$) and, in addition, $\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3 + \mathbf{v}_4$ is 0 on ℓ.

Quantumly we solve the above problem with the algorithm of Ambainis [Amb04]. Originally it was described only for the case $k = 2$ (search version of the so-called *Element distinctness* problem), but later extended to a more general setting, [CE05]. We note that the complexity analysis in [CE05] is done in terms of *query* calls to the g function, while here we take into account the actual time to compute g.

Ambainis algorithm is best described as a quantum walk on the so-called Johnson Graph.

Definition 2 (Johnson graph and its eigenvalue gap). *The Johnson graph $J(N, r)$ for an N-size list is an undirected graph with vertices labelled by all r-size subsets of the list, and with an edge between two vertices S, S' iff $|S \cap S'| = r - 1$. It follows that $J(N, r)$ has $\binom{N}{r}$ vertices. Its eigenvalue gap is $\delta = \frac{N}{r(N-r)}$, [BCA89].*

Let us briefly explain how we solve the k-list matching problem using quantum walks. Our description follows the so-called MNRS framework [MNRS11] due to Magniez-Nayak-Roland-Santha, which measures the complexity of a quantum walk search algorithm in the costs of their Setup, Update, and Check phases.

To setup the walk, we first prepare a uniform superposition over all r-size subsets $S_i \subset \mathcal{L}_i$ together with an auxiliary register (normalization omitted):

$$\sum_{S_i \subset \mathcal{L}_i, \, |S_i| = r} |S_1\rangle \otimes \ldots \otimes |S_k\rangle \otimes |Aux\rangle.$$

The auxiliary register $|Aux\rangle$ contains all the information needed to decide whether $S_1, \ldots, S_k$ contains a match. In the ISD setting, $|Aux\rangle$ stores intermediate and output lists of the matching process. For example, in Stern's algorithm ($k = 2$) $|Aux\rangle$ contains all pairs $(\mathbf{v}_1, \mathbf{v}_2) \in S_1 \times S_2$ that match on ℓ coordinates. In case the merge is done is several steps like in MTT ($k = 4$), the intermediate sublists are also stored in $|Aux\rangle$ (see Figure 1).

The running time and the space complexity of the Setup phase are essentially the running time and the space complexity of the corresponding ISD algorithm with the input lists of size $|S_i| = r$ instead of $|\mathcal{L}_i|$. By the end of the Setup phase, we have a superposition over all r-sublists $S_1, \ldots, S_k$ of $\mathcal{L}_1, \ldots, \mathcal{L}_k$, where each $(S_1, \ldots, S_k)$ is entangled with the register $|Aux\rangle$ that contains the result of merging $(S_1, \ldots, S_k)$ into $S_{\text{out}} \subset \mathcal{L}_{\text{out}}$. Also, during the creation of S_{out} we can already tell if it contains the error-vector $\mathbf{e}$ that solves the ISD problem. When we talk about quantum space of an ISD algorithm (e.g., Table 1), we mean the size of the $|Aux\rangle$ register.

Next, in the Update phase we choose a sublist S_i and replace one element $\mathbf{v}_i \in S_i$ by $\mathbf{v}'_i \notin S_i$. This is one step of a walk on the Johnson graph. We update the data stored in $|Aux\rangle$: remove all the pairs in the merged lists that involve $\mathbf{v}_i$ and create possibly new matches with $\mathbf{v}'_i$. We assume the sub-lists S_i's are kept sorted and stored in a data-structure that allows fast insertions/removals (e.g., radix trees as proposed in [BJLM13]). We also assume that elements in $S_1, \ldots, S_k$ that result in a match, store pointers to their match. For example, if in Stern's algorithm $\mathbf{v}_1 \in S_1, \mathbf{v}_2 \in S_2$ give a match, we keep a pointer to $\mathbf{v}_1 + \mathbf{v}_2 \in S_{\text{out}}$ and also a pointer from $\mathbf{v}_1 + \mathbf{v}_2$ to $\mathbf{v}_1 \in S_1, \mathbf{v}_2 \in S_2$.

After we have performed $\Theta(1/\sqrt{\delta})$ updates (recall, δ is the eigenvaule gap of $J(N, r)$), we check if the updated register $|S_1\rangle \otimes \ldots \otimes |S_k\rangle \otimes |Aux\rangle$ gives a match. This is the Checking phase.

Thanks to the MNRS framework, once we know the costs of (a) the Setup phase $T_{\mathtt{S}}$, (b) the Update phase $T_{\mathtt{U}}$, and (c) the Checking phase $T_{\mathtt{C}}$, we know that after $T_{\mathtt{QW}}$ many steps, we measure a register $|S_1\rangle \otimes \ldots \otimes |S_k\rangle \otimes |Aux\rangle$ that contains the correct error-vector with overwhelming probability, where

$$T_{\mathtt{QW}} = T_{\mathtt{S}} + \frac{1}{\sqrt{\varepsilon}} \left(\frac{1}{\sqrt{\delta}} \cdot T_{\mathtt{U}} + T_{\mathtt{C}} \right). \tag{10}$$

In the above formula, ε is a fraction of vertices in $J(N, r)$ that contain the correct error-vector. For a fixed k, we have $\varepsilon \approx r^k/N^k$ where $N = |\mathcal{L}_1| = \ldots = |\mathcal{L}_k|$. Strictly speaking, the walk we have just described is a walk on a k-Cartesian product of Johnson graphs – one for each sublist S_i, so the value δ in Eq. (10) must be the eigenvalue gap for such a large graph. As proved in [KT17, Theorem 2], for fixed constant k, it is lower-bounded by $\frac{N}{k \cdot r(N-r)}$. The analysis of [KT17] as well ours are asymptotical, so we ignore the constant factor of $1/k$. An optimal choice for r that minimizes Eq. (10) is discussed in the next section.

Kachigar-Tillich quantum ISD algorithms. The quantum walk search algorithm described above solves the ISD problem provided we have found a permutation π that gives the desired distribution of 1's in the error-vector. Kachigar and Tillich in [KT17] suggest to run Grover's algorithm for π with the 'check-

ing' function for Grover's search being a routine for the k-list matching problem. Their ISD algorithm performs transformations on the quantum state of the form (normalization omitted):

$$\sum_{i=1}^{\mathcal{P}(p,\ell)} |\pi_i\rangle \, |\pi_i(\mathbf{H})\rangle \otimes \underbrace{\left[\sum_{S_i \subset \mathcal{L}_i, \, |S_i|=r} |S_1\rangle \otimes \ldots \otimes |S_k\rangle \otimes |Aux\rangle \right]}_{\text{Quantum Walk = Check for the outer Grover}} \otimes |\text{Is } \pi \text{ good?}\rangle \qquad (11)$$

The outer-search is Grover's algorithm over $\mathcal{P}(p,\ell)$ permutations, where $\mathcal{P}(p,\ell)$ is chosen such that we expect to have one π that leads to a good permutation of 1's in the error-vector (see Eq. (5)). The check if a permutation π is good is realized via quantum walk search for k vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k \in S_1 \times \ldots \times S_k$ that match on certain coordinates and lead to the correct error vector. Note an important difference between classical and quantum settings: during the quantum walk we search over sublists $S_i \subset \mathcal{L}_i$ which are exponentially shorter than $\mathcal{L}_i$.

After T_{QW} steps, the register $|Aux\rangle$ contains a k-tuple $(\mathbf{v}_1, \ldots, \mathbf{v}_k)$ that leads to the correct error vector provided a permutation π is good. Hence, after $\widetilde{\mathcal{O}}(\sqrt{\mathcal{P}(p,\ell)} \cdot T_{\text{QW}})$ steps, the measurement of the first register gives a good π with constant success probability. The resulting state will be entangled with registers that store $S_1, \ldots, S_k$ together with the pointers to the matching elements. Once we measure $S_1, \ldots, S_k$, we retrieve these pointers and, finally, reconstruct the error vector as in the classical case.

Quantum Shamir-Schroeppel technique was introduced in [SS81] to reduce the memory complexity of a generic meet-in-the-middle attack, i.e., the k-list matching problem for $k = 2$. Assume we want to find a pair $\mathbf{v}_1, \mathbf{v}_2 \in \mathcal{L}_1 \times \mathcal{L}_2$ s.t. $\mathbf{v}_1 = \mathbf{v}_2$ on certain ℓ coordinates. Assume further that we can decompose $\mathcal{L}_1 = \mathcal{L}_{1,1} + \mathcal{L}_{1,2}$ s.t. $|\mathcal{L}_{1,1}| = |\mathcal{L}_{1,2}| = \sqrt{|\mathcal{L}_1|}$ (analogously, for $\mathcal{L}_2$). The idea of Shamir and Schroeppel is to guess that the correct vectors $\mathbf{v}_1, \mathbf{v}_2$ are equal to some $\mathbf{t} \in \mathbb{F}_2^{\ell'}$ on $\ell' \leq \ell$ coordinates and enumerate all such pairs. Namely, we enumerate $\mathbf{v}_1$ by constructing $\mathcal{L}_1$ in the meet-in-the-middle way from $\mathcal{L}_{1,1}, \mathcal{L}_{1,2}$ in time $\max\{\sqrt{|\mathcal{L}_1|}, |\mathcal{L}_1|/2^{\ell'}\}$, s.t. $\mathcal{L}_1$ only contains vectors that are equal to $\mathbf{t}$ on ℓ' (same for $\mathcal{L}_2$). Classically, we make $2^{\ell'}$ guesses for $\mathbf{t}$, so the overall time of the algorithm will be $|\mathcal{L}_1|$ (same as naive 2-list matching), but we save in memory.

In [KT17], in order to improve not only in memory, but also in time, the authors run Grover's search over $2^{\ell'}$ guesses for $\mathbf{t}$. Indeed, this gives a speed-up for ISD algorithms that solve the 2-list matching problem (cf. the complexities of Dumer's algorithm in Table 1). However, as we argue in the next section, for ISD algorithms that operate on 4 (or more) lists and whose complexity is *not* dominated by the creation of the starting lists (like MMT or BJMM), quantum Shamir-Schroeppel technique does not bring an advantage.

3 Quantum MMT and BJMM algorithms

In this section we analyse the complexity of quantum versions of MMT and BJMM ISD algorithms and argue that the analysis in [KT17] is slightly sub-

optimal when optimized for time. We note that the way we apply and analyse quantum walks to ISD closely resembles Bernstein's et al. algorithm for Subset Sum [BJLM13].

Let us first look at the generalized version of a quantum ISD algorithm, where we can plug-in any of the ISD algorithms described in Sect. 2. Recall that on input, we receive $(\mathbf{H}, \mathbf{s}) \in \mathbb{F}_2^{(n-k) \times n} \times \mathbb{F}_2^{n-k}$, and are asked to output $\mathbf{e} \in \mathbb{F}_2^n$ of weight $wt(\mathbf{e}) = w$ that satisfies $\mathbf{He^t} = \mathbf{s}$. Alg. 1 below can be viewed as a 'meta' quantum algorithm for ISD.

Algorithm 1 A quantum ISD algorithm

1: Prepare a superposition over $\mathcal{P}$-many permutations π
2: For each π
   a: Setup a superposition $|S_1\rangle \otimes \ldots \otimes |S_k\rangle \otimes |Aux\rangle$ for $S_i \subset \mathcal{L}_i$, $|S_i| = r$
   b: Run a quantum walk search on $|S_1\rangle \otimes \ldots \otimes |S_k\rangle \otimes |Aux\rangle$ to find a matching tuple $(\mathbf{e}_1, \ldots, \mathbf{e}_k) \in S_1 \times \ldots \times S_k$, if exists; indicate otherwise that no tuple is found.
3: Apply amplitude amplification (Grover's search) on Step 1 for those π that led to a match on Step 2.b. Measure the register π and then the register $|Aux\rangle$.

The algorithm is parametrized by (I.) the size of the permutation space $\mathcal{P}$ we iterate over in order to find the desired distribution of 1's in the solution (e.g., Eq. (5) for MMT); (II.) k – the number of staring lists $\mathcal{L}_i$'s an ISD-algorithm considers (e.g., $k = 0$ for Prange, $k = 2$ for Stern/Dumer, $k = 4$ for MMT); (III.) r – the size of S_i's, $1 \le i \le k$. The asymptotic complexity of Alg. 1 will depend on these quantities as we now explain in detail.

Step 1 consists in preparing a superposition $\sum_{i=1}^{\mathcal{P}} |\pi_i\rangle |\pi_i(\mathbf{H})\rangle$, which is efficient. Step 2 is a quantum walk algorithm for the k-list matching problem, i.e. search for all $(\mathbf{e}_1, \ldots, \mathbf{e}_k) \in \mathcal{L}_1 \times \ldots \times \mathcal{L}_k$ from which the solution vector can be constructed. The cost of Step 2 can be split into the cost of the Setup phase (Step 2(a)) and the cost of the Update and Check phases (Step 2(b)).

The cost of the Step 2(a) – preparing a superposition over k-tensor product of $S_i \subset \mathcal{L}_i$ and computing the data for $|Aux\rangle$ – is essentially the cost of a classical ISD algorithm, where on input instead of the lists $\mathcal{L}_i$'s, we consider sublists S_i of size $r \ll |\mathcal{L}_i|$. Recall that 'computing the data for $|Aux\rangle$' means constructing the subset $S_{\text{out}} \subset \mathcal{L}_{\text{out}}$ using only elements from S_i's (see Fig. 1). Step 2(b) performs a quantum walk over the k-Cartesian product of Johnson Graphs, $J(|\mathcal{L}_i|, r)^{\otimes k}$, with eigenvalue gap $\delta = \Theta(|\mathcal{L}_i|/(r \cdot (|\mathcal{L}_i| - r))) \approx 1/r$ for $r \ll |L_i|$. To estimate ε – the fraction of $(S_1, \ldots, S_k)$ that give the solution, note that with probability $\Theta(r/|L_i|)$, an r-size subset S_i contains an element $\mathbf{e}_i$ that contributes to the solution. Hence, k such subsets – one vertex of $J(|\mathcal{L}_i|, r)^{\otimes k}$ – contain the solution $(\mathbf{e}_1, \ldots, \mathbf{e}_k)$ with probability $\varepsilon = (r/|\mathcal{L}_i|)^k$. Once we know the time to create $|Aux\rangle$, δ and ε, we obtain the total complexity of Step 2 from Eq. (10).

Finally, Grover's search over $\mathcal{P}$-many permutations requires $\sqrt{\mathcal{P}}$ calls to a 'checking' function s.t. a measurement will output the good π. The measurement will collapse the state given in Eq. (11) into a superposition of $|S_1\rangle \otimes \ldots \otimes |S_k\rangle \otimes$

$|Aux\rangle$, where the amplitude of those $|Aux\rangle$ that contain the actual solution $\mathbf{e}$ will be amplified. Measurement of $|Aux\rangle$ leads to the solution. Regarding Step 2 as 'checking' routine for amplitude-amplification of Step 1 and assuming that an ISD algorithm on input-lists of size r has classical running time $T_{\mathtt{ISD}}(r)$, we obtain the following complexity of Alg. 1:

Theorem 1. *Assume we run Alg. 1 with a classical ISD algorithm that (I.) expects after $\mathcal{P}$ permutations of the columns of $\mathbf{H}$ to find the desired weight-distribution for the error-vector, (II.) performs the search for the error-vector over k lists each of size $|\mathcal{L}|$ in (classical) time $T_{\mathtt{ISD}}(|\mathcal{L}|)$. Then for $r \ll |\mathcal{L}|$ satisfying $T_{\mathtt{ISD}}(r) = \widetilde{\mathcal{O}}\Big(\sqrt{\frac{|\mathcal{L}|^k}{r^{k-1}}}\Big)$, the running time of Alg. 1 is*

$$T_{\mathtt{ISD_Q}} = \sqrt{\mathcal{P}} \cdot T_{\mathtt{ISD}}(r).$$

In particular, the MMT algorithm [MMT11] has $k = 4$, $\mathcal{P} = \mathcal{P}(p, \ell)$ given in Eq. (5), $|\mathcal{L}|$ given in Eq. (7), and $\mathcal{R}_{\text{\tiny MMT}} := \binom{p}{p/2}$, leading to

$$T_{\mathtt{MMT_Q}} = \widetilde{\mathcal{O}}\Big(\sqrt{\mathcal{P}(p,\ell)} \cdot \frac{|\mathcal{L}|^{\frac{8}{7}}}{\mathcal{R}_{\text{\tiny MMT}}^{\frac{3}{7}}}\Big).$$

Similarly, for the BJMM algorithm [BJMM12] with starting lists-sizes $|\mathcal{L}|$ given in Eq. (9) and $\mathcal{R}_{\text{\tiny BJMM}} := \binom{p}{p/2}\binom{k+\ell-p}{\varepsilon}$, we have

$$T_{\mathtt{BJMM_Q}} = \widetilde{\mathcal{O}}\Big(\sqrt{\mathcal{P}(p,\ell)} \cdot \frac{|\mathcal{L}|^{\frac{8}{7}}}{\mathcal{R}_{\text{\tiny BJMM}}^{\frac{3}{7}}}\Big).$$

Proof. The first statement follows from the above discussion: Grover's search for a good π makes $\sqrt{\mathcal{P}}$ 'calls', where each 'call' is a quantum walk search of complexity $T_{\mathtt{ISD}}(r)$. The condition on r is set such that Steps 2(a) and 2(b) in Alg. 1 are asymptotically balanced, namely, we want $T_{\mathtt{Setup}} = \frac{1}{\sqrt{\varepsilon}} \cdot \frac{1}{\sqrt{\delta}} \cdot \widetilde{\mathcal{O}}(\log r)$, cf. Eq. (10). We remind that S_i's are stored in a data-structure that makes the Update and Check of complexity $\widetilde{\mathcal{O}}(\log r)$. Since $T_{\mathtt{Setup}} = T_{\mathtt{ISD}}(r)$, $\delta \approx 1/r$, and $\varepsilon = (r/|\mathcal{L}_i|)^k$, the optimal choice for r should satisfy $T_{\mathtt{ISD}}(r) = \widetilde{\mathcal{O}}\big(\sqrt{|\mathcal{L}|^k/r^{k-1}}\big)$.

For the MMT algorithm, the dominating step is the construction of the lists $\mathcal{L}_1, \mathcal{L}_2$ whose elements are already equal on a certain number of coordinates denoted $\log \mathcal{R}_{\text{\tiny MMT}} \approx p$ in Sect. 2 (see Fig. 1, right). This step is of complexity $T_{\mathtt{MMT}}(|\mathcal{L}|) = \widetilde{\mathcal{O}}(|\mathcal{L}|^2/\mathcal{R}_{\text{\tiny MMT}})$, where $|\mathcal{L}|$ is the size of the starting lists (the four upper-most lists on Fig. 1). Solving $T_{\mathtt{MMT}}(r) \overset{!}{=} \sqrt{\frac{|\mathcal{L}|^k}{r^{k-1}}}$ for r, we receive $r = |\mathcal{L}|^{\frac{4}{7}} \cdot 2^{\frac{2}{7}p}$ as the optimal size for S_i's. Hence, the running time of the quantum walk for MMT is $T_{\mathtt{MMT}}(r) = \widetilde{\mathcal{O}}(|\mathcal{L}|^{\frac{8}{7}}/\mathcal{R}_{\text{\tiny MMT}}^{\frac{3}{7}}) = \widetilde{\mathcal{O}}(|\mathcal{L}|^{\frac{8}{7}}/2^{\frac{3}{7}p})$.

The BJMM algorithm differs from MMT in the number of representations $\mathcal{R}_{\text{\tiny BJMM}}$ and the size of the starting lists $|\mathcal{L}|$. Similar to MMT, we choose $r = |\mathcal{L}|^{\frac{4}{7}} \cdot \mathcal{R}_{\text{\tiny BJMM}}^{\frac{2}{7}}$, the complexity of the quantum walk for BJMM becomes $\widetilde{\mathcal{O}}(|\mathcal{L}|^{\frac{8}{7}}/\mathcal{R}_{\text{\tiny BJMM}}^{\frac{3}{7}})$. $\square$

The analysis of [KT17] differs from ours in computing $T_{\mathtt{ISD}}(r)$: in [KT17] it is assumed that the dominant part MMT or BJMM algorithm is the first merge

(i.e., merging the starting lists on $\log \mathcal{R}_{\text{\tiny MMT}}$ or $\log \mathcal{R}_{\text{\tiny BJMM}}$ coordinates, which results in choosing $r = |\mathcal{L}|^{4/5}$.). This is why they used the so-called Shamir-Schroeppel trick [SS81]: reduce the size of the starting lists and repeat all the merges many times with the new starting lists of smaller size. Classically, this trick allows to reduce only space, while quantumly one could also hope to improve time using Grover over repetitions of merges. Our optimization, however, shows that Shamir-Schroeppel trick does not give any time improvement as it is the case classically, because there is no reason to optimize a non-dominant step. Again, it helped the analysis of [KT17], but this analysis assumes a non-optimal choice of parameters that makes the first merging step dominant. One can, however, using this idea, save in memory for the MMT algorithm (cf. Table 1)

Classically, the improvements over Prange achieved by recent algorithms are quite substantial: BJMM gains a factor of $2^{0.019 \cdot n}$ in the leading-order term. Quantumly, however, the improvement is less pronounced. The reason lies in the fact that the speed-up coming from Grover's search is much larger than the speed-up offered by the quantum walk. Also, the k-list matching problem become harder (quantumly) once we increase k (the fraction of 'good' subsets ε becomes smaller).

The figures from Table 1 are obtained using the optimization package implemented in Maple. The program is available at `http://perso.ens-lyon.fr/elena.kirshanova/`.

4 Decoding with Near Neighbour

For a reader familiar with Indyk-Motwani locality-sensitive hashing [IM98] for Near Neighbour search (defined below), Stern's algorithm and its improvements [Dum91] essentially implement such hashing by projecting on ℓ-coordinates and applying it to the lists $\mathcal{L}_1, \mathcal{L}_2$. In this section, we consider another Near Neighbour technique.

4.1 Re-interpretation of May-Ozerov Near Neighbour algorithm

The best known classical ISD algorithm is due to May-Ozerov [MO15]. It is based on the observation that ISD is a similarity search problem under the Hamming metric. In particular, Eq. (2) defines the approximate relation:

$$\mathbf{Q}\mathbf{e}_1 \approx \mathbf{Q}\mathbf{e}_2 + \bar{\mathbf{s}}. \tag{12}$$

The approximation sign $\approx$ means that the Hamming distance between the left-hand side and the right-hand side of Eq. (12) is at most $wt(\mathbf{e}_3) = w - p$ (cf. Eq. (4)). Enumerating over all $\mathbf{e}_1$ and $\mathbf{e}_2$, we receive an instance of the $(w - p)$-Near Neighbour (NN) problem:

Definition 3 (γ-Near Neighbour). *Let $\mathcal{L} \subset \mathbb{F}_2^n$ be a list of uniform random binary vectors. The γ-Near Neighbour problem consists in preprocessing $\mathcal{L}$ s.t.*

upon receiving a query vector $\mathbf{q} \in \mathbb{F}_2^n$, we can efficiently find all $\mathbf{v} \in \mathcal{L}$ that are γ-close to $\mathbf{q}$, i.e., all $\mathbf{v}$ with $dt(\mathbf{v}, \mathbf{q}) \leq \gamma \cdot n$ for some $\gamma \leq 1/2$.[2]

Thus the ISD instance given in Eq. (12) becomes a special case of the $(w-p)$-NN problem with $\mathcal{L} = \{\mathbf{Q}\mathbf{e}_1\}$ for all $\mathbf{e}_1 \in \mathbb{F}_2^{(k+\ell)/2} \times \mathbf{0}^{(k+\ell)/2}\times$, and the queries taken from $\{\mathbf{Q}\mathbf{e}_2 + \mathbf{s}\}$ for all $\mathbf{e}_2 \in \mathbf{0}^{(k+\ell)/2} \times \mathbb{F}_2^{(k+\ell)/2}$. In [MO15], the algorithm is described for this special case, namely, when the number of queries is equal to $|\mathcal{L}|$ and all the queries are explicitly given in advance. So it is not immediately clear how to use their result in quantum setting, where we only operate on the sublists of $\mathcal{L}$ and update them with new vectors during the quantum walk.

In this section, we re-phrase the May-Ozerov algorithm in terms more common to the near neighbour literature, namely in the 'Update' and 'Query' quantities. It allows us to use the algorithm in more general settings, e.g., when the number of queries differs from $|\mathcal{L}|$ and when the query-points $\mathbf{q}$ do not come all at once. This view enables us to adapt their algorithm to quantum-walk framework.

The main ingredient of the May-Ozerov algorithm is what became known as Locality-Sensitive Filtering (LSF), see [BDGL16] for an example of this technique in the context of lattice sieving. In LSF we create a set $\mathcal{C} \subset \mathbb{F}_2^n$ of *filtering* vectors $\mathbf{c}$ which divide the Hamming space into (possibly overlapping) regions. These regions are defined as Hamming balls of radius α centred at $\mathbf{c}$, where α is an LSF-parameter we can choose. So each filtering vector $\mathbf{c} \in \mathcal{C}$ defines a region $\text{Region}_{\mathbf{c}}$ as the set of all vectors that are α-close to $\mathbf{c}$, namely, $\text{Region}_{\mathbf{c}} = \{\mathbf{v} \in \mathbb{F}_2^n : dt(\mathbf{v}, \mathbf{c}) \leq \alpha\}$. Drawing an analogy with Locality-Sensitive Hashing, these filtering vectors play role of hash-functions. In LSF, instead of defining a function, we define its pre-image.

The preprocessing for the input list $\mathcal{L}$ consists in creating a large enough set $\mathcal{C}$ of filtering vectors and assigning all $\mathbf{v} \in \mathcal{L}$ to their regions (see the INSERT($\mathbf{v}$) function in Alg. 2 below). This assignment defines the LSF buckets as $\text{Bucket}_{\mathbf{c}} = \text{Region}_{\mathbf{c}} \cap \mathcal{L}$. The LSF data structure $\mathcal{D}$ consists of the union of all the buckets. In the course of quantum walk search, we will also need to remove vectors from $\mathcal{D}$. For that we have the REMOVE($\mathbf{v}$) function which deletes $\mathbf{v}$ from all the buckets $\text{Bucket}_{\mathbf{c}}$ containing $\mathbf{v}$. Note that for each $\text{Bucket}_{\mathbf{c}}$ both INSERT() and REMOVE() can be implemented in time $\widetilde{\mathcal{O}}(\log |\text{Bucket}_{\mathbf{c}}|)$ if we store the buckets as, for example, binary trees. Finally, in order to answer a query $\mathbf{q}$, we look at all buckets $\text{Bucket}_{\mathbf{c}}$ that are β-close to $\mathbf{q}$ (i.e., all $\mathbf{c} \in \mathcal{C}$ with $dt(\mathbf{q}, \mathbf{c}) \leq \beta$), and we check if any of the vectors stored in these β-close buckets gives a solution to γ-Near Neighbour. As it is typically the case for NN-algorithms [Laa15], we have two trade-off parameters (α, β): the closer α to $1/2$, the more buckets we should create, but the query is fast because we may allow small β. Making α smaller reduces the prepocessing cost but requires more work during queries.

---

[2] The (dimensionless) distances we consider here, denoted further γ, α, β, are all $\leq 1/2$, since we can flip the bits of the query point and search for 'close' rather than 'far apart' vectors.

Algorithm 2 Algorithms for Locality-Sensitive Filtering

Parameters:
    α - the insertion parameter
    β - the query parameter
    γ - the target distance
    $\mathcal{C}$ - the set of filtering vectors
    $\mathcal{D}$ - the LSF data structure: $\mathcal{D} = \cup_{\mathbf{c} \in \mathcal{C}} \text{Bucket}_{\mathbf{c}}$

1: **function** INSERT($\mathbf{x}$)   ▷ Add $\mathbf{x}$ to all the relevant buckets of $\mathcal{D}$
2:     **for all** $\mathbf{c} \in \mathcal{C}$ s.t. $dt(\mathbf{c}, \mathbf{x}) \leq \alpha$ **do**
3:         $\text{Bucket}_{\mathbf{c}} \leftarrow \text{Bucket}_{\mathbf{c}} \cup \{\mathbf{x}\}$
4:     **end for**
5: **end function**

1: **function** REMOVE($\mathbf{x}$)   ▷ Remove $\mathbf{x}$ from all buckets
2:     **for all** $\mathbf{c} \in \mathcal{C}$ s.t. $dt(\mathbf{c}, \mathbf{x}) \leq \alpha$ **do**
3:         $\text{Bucket}_{\mathbf{c}} \leftarrow \text{Bucket}_{\mathbf{c}} \setminus \{\mathbf{x}\}$
4:     **end for**
5: **end function**

1: **function** QUERY($\mathbf{q}$)   ▷ Find all $\mathbf{x} \in \mathcal{D}$ with $dt(\mathbf{x}, \mathbf{q}) \leq \beta$
2:     CloseVectors $\leftarrow \emptyset$
3:     **for all** $\mathbf{c} \in \mathcal{C}$ s.t. $dt(\mathbf{c}, \mathbf{q}) \leq \beta$ **do**
4:         **for all** $\mathbf{x} \in \text{Bucket}_{\mathbf{c}}$ **do**
5:             **if** $dt(\mathbf{x}, \mathbf{q}) \leq \gamma$ **then**
6:                 CloseVectors $\leftarrow$ CloseVectors $\cup \{\mathbf{x}\}$
7:             **end if**
8:         **end for**
9:     **end for**
10:    **return** CloseVectors
11: **end function**

Structured filter-vectors or the 'strips technique'. In the main procedures of LSF, UPDATE, REMOVE, and QUERY, we are required to find all close buckets for a given point. Naive search finds these buckets time $|\mathcal{C}|$ which is inefficient. We can do better by making filter-vectors $\mathbf{c}$ structured. The technique has several names, 'strips' in [MO15], 'random product code' in [BDGL16], and 'tensoring' in [Chr17], but either way it amounts to the following. Each vector $\mathbf{c}$ is a concatenation of several codewords from some low-dimensional codes (so, $\mathcal{C}$ is a Cartesian product of all these codes). All $\mathbf{c}$'s close to $\mathbf{x}$ are obtained by iteratively decoding the relevant projections of $\mathbf{x}$ under the codes defined on these projections (say, for $\mathbf{x} = [\mathbf{x}_1 || \ldots || \mathbf{x}_\ell]$, we start by decoding $\mathbf{x}_1$). On each iteration, we filter out those $\mathbf{c}$'s that are guaranteed to be far from $\mathbf{x}$ (i.e, only $\mathbf{c} = [\mathbf{c}_1 || \ldots || \mathbf{c}_\ell]$'s with $\mathbf{c}_1$ close to $\mathbf{x}_1$ are kept). Choosing the lengths of low-dimensional codes carefully enough, we can ensure that $\mathbf{c}$'s are sufficiently close to independent random vectors. This trick allows us find all close buckets in time (up to lower-order terms) equal to the output size. We refer the reader to [BDGL16,MO15] for details.

Before we give complexities for the routines described in Alg. 2 as functions of α, β, γ, we recall the definition of the entropy function for a discrete probability distribution defined by a vector $\mathbf{p}$.

Definition 4. *Let $\mathbf{p} \in \mathbb{R}^t$ be a real vector that represents a certain probability distribution, i.e., $\mathbf{p}$ satisfies $0 \leq \mathbf{p}_i \leq 1$, and $\sum_{i=1}^t \mathbf{p}_i = 1$. Then $H(\mathbf{p})$ is the entropy of the distribution $\mathbf{p}$:*

$$H(\mathbf{p}) = -\sum_i \mathbf{p}_i \log \mathbf{p}_i.$$

We will be using the above definition in the following context: Let $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ be an m-tuple of vectors from $\mathbb{F}_2^n$ and let $\mathbf{p} \in \mathbb{R}^{2^m}$ be a real vector indexed by all m-length binary vectors that represents the distribution of the m-tuple. That is, $\mathbf{p}_{i_1 \ldots i_m}$ counts the number of occurrences (relative to n) of the coordinates' configuration: $\mathbf{p}_{i_1 \ldots i_m} = |\{c : \mathbf{x}_1[c] = i_1, \ldots, \mathbf{x}_m[c] = i_m\}|$. Such $\mathbf{p}$ defines a discrete probability distribution on $\{1, \ldots, 2^m\}$.

For example, consider a random 2-tuple $(\mathbf{x}_1, \mathbf{x}_2)$ with $dt(\mathbf{x}_1, \mathbf{x}_2) = w$. Its distribution vector is $\mathbf{p} = (p_{00}, p_{01}, p_{10}, p_{11})$ satisfying $p_{01} + p_{10} = w$ and $p_{00} = |\{c : \mathbf{x}_1[c] = 0, \mathbf{x}_2[c] = 0\}|$, $p_{01} = |\{c : \mathbf{x}_1[c] = 0, \mathbf{x}_2[c] = 1\}|$, $p_{10} = |\{c : \mathbf{x}_1[c] = 1, \mathbf{x}_2[c] = 0\}|$, $p_{11} = |\{c : \mathbf{x}_1[c] = 1, \mathbf{x}_2[c] = 1\}|$.

In case $\mathbf{x}_1$ is fixed, we can shift the tuple: $(\mathbf{0}, \mathbf{x}_2 - \mathbf{x}_1)$, and obtain $\mathbf{p} = (1 - w, w)$ with $H(\mathbf{p}) = H(w) = -w \log w - (1 - w) \log(1 - w)$, which just counts the number of all binary vectors of weight w.

In the following, we give complexities of the Near Neighbour problem routines. We assume throughout that the target distance $0 \leq \gamma \leq 1/2$ and the parameters $0 \leq \alpha, \beta \leq 1/2$ are fixed. We remark that the choice of $\mathbf{p}$ given in the next lemma corresponds to the so-called *balanced configuration* used for lattice-sieving in [HK17,HKL18]. A *configuration* describes certain pairwise properties between k-tuples of vectors: for lattice-sieving the interesting property is the inner-product between each pair of vectors in a k-tuple, while in our case, it is the Hamming distance we are concerned about. The lemma below describes a configuration $\mathbf{p}$ attained by almost all triples $(\mathbf{v}, \mathbf{x}, \mathbf{q})$) with prescribed pairwise Hamming distances.

Lemma 1 (Size of $\mathcal{C}$). *To answer a Near Neighbour query $\mathbf{q}$ with the QUERY($\mathbf{q}$) procedure from Alg. 2, i.e., output all $\mathbf{v} \in \mathcal{L}$ s.t. $dt(\mathbf{q}, \mathbf{v}) \leq \gamma$ with super-exponentially small error [3], the total number of buckets $\mathcal{C}$ should be (up to subexponential factors)*

$$|\mathcal{C}| = 2^{(1 - (H(\mathbf{p}(\alpha, \beta, \gamma)) - H(\mathbf{p}(\gamma))) \cdot n}, \tag{13}$$

where $\mathbf{p}(\alpha, \beta, \gamma) \in \mathbb{R}^8$ and $\mathbf{p}(\gamma) \in \mathbb{R}^4$ are probability distributions that satisfy

---

[3] By 'error' we mean missing a vector which is γ-close to $\mathbf{q}$.

$$\mathbf{p}(\alpha,\beta,\gamma): \begin{cases} p_{000} = p_{111} = \frac{1}{2} - \frac{1}{4}(\gamma + \beta + \alpha) \\ p_{001} = p_{110} = \frac{1}{4}(\gamma + \beta - \alpha) \\ p_{010} = p_{101} = \frac{1}{4}(\gamma + \alpha - \beta) \\ p_{100} = p_{011} = \frac{1}{4}(\beta + \alpha - \gamma) \end{cases} \qquad \mathbf{p}(\gamma): \begin{cases} p_{00} = p_{11} = \frac{1-\gamma}{2} \\ p_{10} = p_{01} = \frac{\gamma}{2}. \end{cases}$$

Proof. Consider a pair $(\mathbf{v}, \mathbf{q})$ s.t. $dt(\mathbf{v}, \mathbf{q}) = \gamma$. The number of filtering vectors $|\mathcal{C}|$ is determined by the inverse of the probability that a random vector $\mathbf{c}$ will 'find' this pair, namely

$$|\mathcal{C}| = 1/ \Pr_{\mathbf{c} \in \{0,1\}^n}[dt(\mathbf{c}, \mathbf{v}) = \alpha, dt(\mathbf{c}, \mathbf{q}) = \beta \mid dt(\mathbf{v}, \mathbf{q}) = \gamma]. \qquad (14)$$

Note that we switched from the '$\leq$' sign to the '$=$' sign for distances. For α, β cases, this is a legitimate change, since

$$\sum_{\alpha'=0}^{\alpha} \sum_{\beta'=0}^{\beta} \Pr_{\mathbf{c} \in \{0,1\}^n}[dt(\mathbf{c}, \mathbf{v}) = \alpha', dt(\mathbf{c}, \mathbf{q}) = \beta' \mid dt(\mathbf{v}, \mathbf{q}) = \gamma]$$

$$\approx \Pr_{\mathbf{c} \in \{0,1\}^n}[dt(\mathbf{c}, \mathbf{v}) = \alpha, dt(\mathbf{c}, \mathbf{q}) = \beta \mid dt(\mathbf{v}, \mathbf{q}) = \gamma].$$

The approximate equality holds as each summand is exponential in n and $\alpha, \beta \leq 1/2$. So the sum attains its maximum at largest α, β (otherwise, we could have decreased α and/or β, which will not affect the success probability but will make the search faster). So up to $\mathrm{poly}(n)$ factors, the above sum is determined by the maximal summand. At the end of the proof we argue on the validity of the sign change for γ.

If we want to find all but super-exponentially small fraction of $\mathbf{v}$'s for a given $\mathbf{q}$, we increase $|\mathcal{C}|$ by a $\mathrm{poly}(n)$ factor for some large enough polynomial and obtain the result by Chernoff bounds.

The denominator of Eq. (14) is (assuming $\mathbf{c}, \mathbf{q}$, and $\mathbf{v}$ are uniform)

$$\frac{\Pr_{\mathbf{c}, \mathbf{v}, \mathbf{q}}[dt(\mathbf{c}, \mathbf{v}) = \alpha, dt(\mathbf{c}, \mathbf{q}) = \beta, dt(\mathbf{v}, \mathbf{q}) = \gamma]}{\Pr_{\mathbf{v}, \mathbf{q}}[dt(\mathbf{v}, \mathbf{q}) = \gamma]} = \frac{2^{H(\mathbf{p}(\alpha,\beta,\gamma)) \cdot n}/2^{3n}}{2^{H(\mathbf{p}(\gamma)) \cdot n}/2^{2n}} = \frac{1}{|\mathcal{C}|}, \quad (15)$$

for some distribution vectors $\mathbf{p}(\alpha, \beta, \gamma), \mathbf{p}(\gamma)$.

The statement about the entries of the vector $\mathbf{p}(\alpha, \beta, \gamma)$ comes from the following three facts (the entries for $\mathbf{p}(\gamma)$ are straightforward to obtain):
– the distance constraints: three for $\mathbf{p}(\alpha, \beta, \gamma)$ and one for $\mathbf{p}(\gamma)$,
– the uniformity of $\mathbf{v}, \mathbf{c}$ and $\mathbf{q}$ (this allows to assume that the contribution of $p_{01} = p_{010} + p_{011}$ and $p_{10} = p_{100} + p_{101}$ to the distance between two uniform vectors is the same),
– the fact that $\sum_i \mathbf{p}_i = 1$.
This gives us 7 equations for 8 variables leaving 1 degree of freedom. We further assume that $p_{000} = p_{111}$ (essentially, the same choice was done in [MO15, Lemma2]). Solving these linear equations gives $\mathbf{p}_i$'s as stated in the theorem.

It remains to argue that our choice for $|\mathcal{C}|$ also works for distances $dt(\mathbf{v}, \mathbf{q}) \leq \gamma$. Informally, as $\gamma' < \gamma < 1/2$, the Near Neighbour problem becomes easier as γ' decreases.

Note that our choice for $\mathbf{p}$ constraints the choices for α, β, γ : these parameters should be chosen such that $\mathbf{p}_i \geq 0$, in particular, $\alpha - \beta < \gamma < \alpha + \beta$ (wlog. we assume $\alpha > \beta$). Let us fix α, β. We show that for $0 \leq \gamma' \leq \gamma$, the probability (over the choice of $\mathbf{c}$) to find a solution is monotonously increasing as γ decreases, and hence, $|\mathcal{C}|$-many vectors will suffice.

Consider Eq. (15) with $\gamma := \gamma'$. Taking the derivative with respect to γ', reveals that the probability (given by $1/|\mathcal{C}|$) increases as long as

$$(1 - 2\beta)(2\alpha - 1)\gamma'^2 - 2(\alpha - \beta)^2\gamma' + (\alpha - \beta)^2 \leq 0.$$

This is a quadratic equation in γ' with roots at $\gamma' = (\alpha - \beta)/(1 - 2\beta)$ and $\gamma' = (\alpha - \beta)/(2\alpha - 1)$ (the latter is negative as $\alpha > \beta$ and $\alpha \leq 1/2$). If γ' satisfies $(\alpha - \beta)/(1 - 2\beta) < \gamma' < a + b$, the derivative is negative and we can choose α, β the same as we chose for γ, hence, with $\mathcal{C}$ defined by Eq. (15), we find all solutions. For $\gamma' < (\alpha - \beta)/(1 - 2\beta)$, we set $\alpha := (1 - 2\beta)\gamma' + \beta$ with β as in Eq. (15). These parameters lead to $|\mathcal{C}| = 2^{n(1 - H(\beta))}$, which depends only on (fixed) β and, hence, cannot decrease with γ. $\qquad \square$

The next theorem gives asymptotical complexities of the routines in Alg. 2.

Theorem 2 (LSF complexity for Hamming metric). *For the Near Neighbour problem with some fixed target $0 \leq \gamma \leq 1/2$, the routines given in Alg. 2 for some fixed $0 \leq \alpha, \beta \leq 1/2$ and the data structure $\mathcal{D} = \cup_{\mathbf{c}} Bucket_{\mathbf{c}}$, have the following expected costs (up to terms sub-exponential in n):*

- *Each* UPDATE *costs* $\quad T_{\mathtt{Upd}}^{\mathtt{LSF}} = |\mathcal{C}| \cdot 2^{(H(\alpha) - 1)n}$.
- *Preprocessing costs* $\quad T_{\mathtt{Prep}}^{\mathtt{LSF}} = |\mathcal{L}| \cdot |\mathcal{C}| \cdot 2^{(H(\alpha) - 1)n}$.
- *Each* QUERY *costs* $T_{\mathtt{Query}}^{\mathtt{LSF}} = |\mathcal{C}| \cdot 2^{(H(\beta) - 1)n} \cdot \mathbb{E} |Bucket_{\mathbf{c}}|$, *where* $\mathbb{E} |Bucket_{\mathbf{c}}|$ *– the expected size of each bucket – is equal to* $|\mathcal{L}| \cdot 2^{(H(\alpha) - 1)n}$.

Proof. We assume that our buckets $\mathcal{C}$ are constructed using 'structured' filter vectors $\mathbf{c}$, which enables us to find all the buckets within a certain distance to a fixed point in the output time (see the discussion above) and, at the same time, allows us to treat $\mathbf{c}$ as (sufficiently) uniform random vectors.

The expected number of buckets for an update parameter α and a fixed $\mathbf{v}$ is $|\mathcal{C}| \cdot \Pr_{\mathbf{c} \in \{0,1\}^n} [dt(\mathbf{v}, \mathbf{c}) = \alpha] = |\mathcal{C}| \cdot \Pr [wt(\mathbf{c} - \mathbf{v}) = \alpha] = |\mathcal{C}| \cdot 2^{H(\mathbf{p}(\alpha))n} = |\mathcal{C}| \cdot 2^{(H(\alpha) - 1)n}$. Preprocessing calls UPDATE($\mathbf{v}$) for all $\mathbf{v} \in \mathcal{L}$, hence its complexity is $|\mathcal{L}| \cdot T_{\mathtt{Upd}}$.

The probability that $\mathbf{v}$ will be added to a certain bucket during the update is again $2^{(H(\alpha) - 1)n}$, so after $|\mathcal{L}|$ calls to UPDATE, the average bucket-load will be (up to sub-exponential terms) $|\mathcal{L}| \cdot 2^{(H(\alpha) - 1)n}$. Treating $|Bucket_{\mathbf{c}}|$ as a random variable and using standard Chernoff bound arguments, one can easily show that $|Bucket_{\mathbf{c}}|$ does not significantly deviate from its expected value.

During the QUERY($\mathbf{q}$) calls, we find all β-close buckets in time $|\mathcal{C}| \cdot 2^{(H(\beta) - 1)n}$ and for each bucket we look through $|Bucket_{\mathbf{c}}|$ vectors and among them choose all γ-close to $\mathbf{q}$. $\qquad \square$

In the application to ISD, where the number of queries is equal to $|\mathcal{L}|$, it makes sense to setup the NN-parameters α and β s.t. the time spent on preprocessing and the time spent on $|\mathcal{L}|$ queries are equal. Indeed, in May-Ozerov algorithm, we have $\alpha = \beta$ and, furthermore, $\alpha = H^{-1}(1 - \log |\mathcal{L}|)$ to make the expected size of buckets equal to 1. After almost trivial algebraic manipulations with Eq. (13) for these parameters, we obtain $\log |C| = (1 - \gamma) \left(1 - H \left(\frac{H^{-1}(1 - \log |\mathcal{L}|) - \gamma/2}{1 - \gamma} \right) \right)$, which matches the result of [MO15, Theorem1].

4.2 Quantum ISD with Near Neighbour

Here we explain how to embed the Near Neighbour routines into quantum walk search. In classical setting, we would create two lists $\mathcal{L}_1, \mathcal{L}_2$ of equal size (see Fig. 1), setup the data structure $\mathcal{D}$ (i.e., choose enough filter-vectors) and call UPDATE($\mathbf{v}_1$) for all $\mathbf{v}_1 \in \mathcal{L}_1$ with some update parameter α. This is the preprocessing stage. Then, for each $\mathbf{v}_2 \in \mathcal{L}_2$, we call QUERY($\mathbf{v}_2$) for a query parameter β and search through the output of size $|\mathcal{C}| \cdot 2^{(H(\beta)-1)n} \cdot \mathbb{E} |\text{Bucket}_\mathbf{c}|$ for $\mathbf{v}_1 \in \mathcal{D}$ s.t. $dt(\mathbf{v}_1, \mathbf{v}_2) = w - p$. This is the query stage. From the solution pair $(\mathbf{v}_1, \mathbf{v}_2)$, we retrieve the error-vector and solve the 2-list matching problem. If we set $\alpha = \beta$ to balance out the costs for updates and queries, and $\alpha = H^{-1}(1 - \log |\mathcal{L}_1|)$ to balance preprocessing and query stages, we solve the 2-list matching problem for ISD in time $|\mathcal{C}|$ which is exactly what May-Ozerov algorithm achieves.

It is not hard to see that the 'Update-and-Query' description of the Near Neighbour search suits particularly well the quantum walk search framework. Assume we run the walk over a superposition of $(S_1, S_2) \subset \mathcal{L}_1 \times \mathcal{L}_2$, where $|S_1 \cup S_2| = \Theta(r)$ for some r which will be determined later. During the Setup phase we create the LSF data structure for S_1, and call UPDATE($\mathbf{v}_1$) for all $\mathbf{v}_1 \in S_1$. Now, contrary to the classical setting, we apply Grover's search over all $\mathbf{v}_2 \in S_2$ with the Grover checking function being QUERY($\mathbf{v}_2$), which tells us whether (S_1, S_2) is 'marked', i.e., whether it contains $(\mathbf{v}_1, \mathbf{v}_2)$ s.t. $dt(\mathbf{v}_1, \mathbf{v}_2) = w - p$. This allows us to spend more time on QUERY calls choosing $\beta \neq \alpha$.

We do the same during the Update+Check phases: we update $S_1 \cup S_2$ with $\Theta(\sqrt{r})$ new vectors and for each of them we call the LSF UPDATE routine. We also delete $\Theta(\sqrt{r})$ vectors by calling the LSF REMOVE function. The Checking phase of the walk calls QUERY() in the superposition over all $\Theta(\sqrt{r})$ new vectors and decides in time $\Theta(r^{1/4} \cdot T^{\text{LSF}}_{\text{Query}})$ whether the updated $S_1 \cup S_2$ is marked.

So the advantage of quantum walk is two-fold: first, we work only with exponentially shorter sublists S_1, S_2 and, second, during the Checking phase we use Grover over many QUERY calls. Alg.3 below summarizes the above description and should be used at Step (2.b) of Alg. 1.

Finally, one can combine quantum Near Neighbour search with the Shamir-Schroeppel trick (see Sect. 2): instead of working with long lists $\mathcal{L}_1, \mathcal{L}_2$, consider their sublists $\mathcal{L}'_1, \mathcal{L}'_2 \subset \mathcal{L}_1, \mathcal{L}_2$ s.t. $\mathbf{v}_1 \in \mathcal{L}'_1$ and $\mathbf{v}_2 \in \mathcal{L}'_2$ are equal to a certain vector $\mathbf{t} \in \mathbb{F}_2^{\ell'}$ on ℓ'-coordinates. The probability that $\mathcal{L}'_1, \mathcal{L}'_2$ contain the solution is $2^{-\ell'}$. Quantumly, the cost to construct $\mathcal{L}'_1, \mathcal{L}'_2$ that contain the solution is $2^{-\ell'/2}$ (Grover's search). Now run NN-search on shorter lists $\mathcal{L}'_1, \mathcal{L}'_2$ and on the

dimension reduced by ℓ'. This adds one more parameter ℓ' into the optimization problem. Such algorithm offers a slight improvement both in time and memory over plain Stern's algorithm as the next theorem shows.

Algorithm 3 A quantum walk with Near Neighbour

Quantum walk SETUP:
1: Create the LSF data structure $\mathcal{D}$ on the auxiliary register $|Aux\rangle$
2: **for all** $\mathbf{v}_1 \in S_1$ **do** $\triangleright |S_1| = |S_2| = \Theta(r)$
3:    Call UPDATE($\mathbf{v}_1$) $\triangleright$ Update $\mathcal{D}$
4: **end for**
5: Using Grover search over all $\mathbf{v}_2 \in S_2$: QUERY($\mathbf{v}_2$) to check if (S_1, S_2) is marked

Quantum walk UPDATE:
6: $S_{\text{new}} \leftarrow \emptyset$
7: Repeat $\Theta(\sqrt{|S_1|})$ times:
8:    Call UPDATE($\mathbf{v}^\star$) $\triangleright$ Add a new $\mathbf{v}^\star \notin S_1 \cup S_2$ to $\mathcal{D}$
9:    $S_{\text{new}} \leftarrow S_{\text{new}} \cup \{\mathbf{v}^\star\}$
10:    Call REMOVE($\mathbf{v}$) $\triangleright$ Remove $\mathbf{v} \in S_1 \cup S_2$ from $\mathcal{D}$
11:    $(S_1, S_2) \leftarrow (S_1, S_2) \setminus \{\mathbf{v}\}$
12: Update the register (S_1, S_2) with S_{new}

Quantum walk CHECK:
13: Run Grover search over all $\mathbf{v}_2 \in S_{\text{new}}$ using QUERY($\mathbf{v}_2$) to check if (S_1, S_2) is marked

---

Theorem 3 (Quantum Dumer+Near Neighbour). *Assume we run Alg. 1 for Dumer's decoding, where during quantum walk we use the $(w-p)$-Near Neighbour routines from Alg. 3. Then the expected running time of Dumer's algorithm is $\widetilde{\mathcal{O}}(2^{0.059922 \cdot n + o(n)})$ with quantum memory complexity $\widetilde{\mathcal{O}}(2^{0.00897 \cdot n + o(n)})$. Using additionally the Shamir-Schroeppel trick, time and memory can be improved to $\widetilde{\mathcal{O}}(2^{0.059450 \cdot n + o(n)})$ and $\widetilde{\mathcal{O}}(2^{0.00808 \cdot n + o(n)})$.*

Proof. The number of trials $\mathcal{P} = \mathcal{P}(p, \ell)$ until we find a good permutation π for the Near Neighbour version of Dumer's decoding is given in Eq. (5). Grover's search will find a good π in time $\mathcal{O}(\sqrt{\mathcal{P}})$. The checking routine for this search is a quantum walk over the subsets $(S_1, S_2) \subset \mathcal{L}_1 \times \mathcal{L}_2$ with $|\mathcal{L}_1| = |\mathcal{L}_2| = \binom{(k+\ell)/2}{p/2}$, where during the walk we look for an approximate match in $S_1 \cup S_2$ using Alg. 3. Assume $|S_1 \cup S_2| = r$. We want to determine r and the LSF parameters α, β for UPDATE and QUERY that minimize the Near Neighbour search. In the following we omit the $\widetilde{\mathcal{O}}$-notation for all runtimes.

The complexity of the quantum walk Setup is $\max\{|\mathcal{C}|, r \cdot T_{\text{Upd}}^{\text{LSF}}, \sqrt{r} \cdot T_{\text{Query}}^{\text{LSF}}\}$, where $|\mathcal{C}|$ is given in Eq. (13) and T_*^{LSF} is given in Thm. 2. That is, we take the maximum between the time to setup $\mathcal{D}$, call the UPDATE r times and run Grover over the r new elements to decide on marked subsets for the starting superposition. The decision is realized via calling QUERY.

19

In the Update phase, we call $\sqrt{r}$ times UPDATE and REMOVE LSF routines to update $\mathcal{D}$. The complexity of the Update phase is $\sqrt{r} \cdot T_{\mathtt{Upd}}^{\mathtt{LSF}}$, and of the checking phase is $r^{1/4} \cdot T_{\mathtt{Query}}^{\mathtt{LSF}}$. As in the classical case, we set

$$\alpha = H^{-1}(1 - \log r)$$

to guarantee that the expected size of each bucket is 1.[4] Note that this choice also balances $|\mathcal{C}| = r \cdot T_{\mathtt{Upd}}^{\mathtt{LSF}}$ for the quantum walk Setup. Finally, the quantum walk Checking routine runs Grover search over $\sqrt{r}$ new elements in $S_1 \cup S_2$ to update the 'marking' flag for $S_1 \cup S_2$. To balance the Update and the Check phases (i.e, when $\sqrt{r} \cdot T_{\mathtt{Upd}} = r^{1/4} \cdot T_{\mathtt{Query}}$), we set

$$\beta = H^{-1}(1 - \tfrac{3}{4}\log r).$$

Such choice also guarantees that during the Setup, $r \cdot T_{\mathtt{Upd}}^{\mathtt{LSF}} \geq \sqrt{r} \cdot T_{\mathtt{Query}}^{\mathtt{LSF}}$. Moreover, it enables us to setup β slightly larger than α since QUERY becomes cheaper.

Finally, we want to balance $T_{\mathtt{S}}$ for Setup, which is $r \cdot T_{\mathtt{Upd}}^{\mathtt{LSF}}$, with the Update and Check phases, $\frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}} \cdot T_{\mathtt{U}} + T_{\mathtt{C}}\right)$, cf. Eq. (10). Due to our choices of α, β, this expression is equal to $\frac{1}{\sqrt{\varepsilon}} \cdot \sqrt{r} T_{\mathtt{Upd}}^{\mathtt{LSF}}$ since $\delta \approx 1/r$.

For $k = 2$, $\varepsilon = r^2/|\mathcal{L}_1 \cup \mathcal{L}_2|^2$, from where we obtain

$$r = |\mathcal{L}_1 \cup \mathcal{L}_2|^{2/3} \approx \binom{(k+\ell)/2}{p/2}^{2/3}.$$

The last parameter we need to determine in order to give the complexity of decoding is the weight parameter p for which we execute the $(w - p)$-Near Neighbour search. The brute-force search over p reveals that for $p = 0.0027 \cdot n$, $\alpha = 0.4169 \cdot n$, $\beta = 0.4280 \cdot n$, we have $|\mathcal{C}| = 2^{0.00897 \cdot n}$. We obtain the figures stated in the theorem by computing the necessary number of permutation for such p and noting that $|\mathcal{C}|$ determines the memory cost.

If we construct the lists $\mathcal{L}_1, \mathcal{L}_2$ using the Shamir-Schroeppel idea, we start with $k = 4$ list $\mathcal{L}_{i,j}$ each of size $\binom{(k+\ell)/4}{p/4}$. We merge them into 2 lists $\mathcal{L}_1, \mathcal{L}_2$ by enforcing the vectors $(\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ having the same value on ℓ' coordinates. Quantumly, we find the correct value for the ISD solution in time $2^{\ell'/2}$. We solve the 4-list matching problem via quantum walk with the optimal choice for $r = |\mathcal{L}_{i,j}|^{4/5}$. Optimization reveals that the choosing $p = 0.043, \ell' = 0.007, \alpha = 0.4330, \beta = 0.4419$, gives the best running time. □

Why choosing larger k does not help. The more starting lists k an ISD algorithm has, the larger the fraction $1/\varepsilon = |L|^k/r^k$ is for any $r < |L|$. Hence, the running time of approximate search and, consequently, the running time of quantum walk become more expensive. The search for optimal parameters tries to shift the work-load to the Grover search for a good permutation by making p smaller (the smaller p is, the harder it is find a good π but the easier the NN-search).

---

[4] One could also run Grover inside each bucket during the Query phase, when the buckets are larger than 1. This, however, does not seem to bring an improvement.

From the above theorem, we have for $k = 2$, $p = 0.0027$ which is already quite small. An optimization for $k = 4$ (e.g., MMT) chooses $p = 0$ which is Prange's algorithm. This is also the reason why we do not get a quantum speed-up for algorithms proposed in [BM17].

5   Open Problems

1. Extend the reduction 3XOR $\rightarrow$Triangle finding to approximate 3XOR
2. Claw-finding instead of ED
   TODO: check claw-finding algorithms as they seem more suitable for ISD. Checked: does not matter for ISD whether we use claw-finding or ED. Would have made a difference if we knew how to choose $\mathbf{e}_i$'s to make the output lists sorted (hence, could skip many $\mathbf{e}_i$'s); we cannot do it unless $\mathbf{H}$ is somehow structured. Move it to open problems

References

Amb04.   Andris Ambainis. Quantum walk algorithm for element distinctness. *FOCS*, pages 210–239, 2004.

BCA89.   Andries E. Brouwer, Arjeh M. Cohen, and Neumaier Arnold. *Distance-Regular Graphs*. Springer-Verlag, 1989.

BDGL16. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. SODA'16, pages 10–24, 2016.

Ber10.   Daniel J. Bernstein. Grover vs. McEliece. In *Post-Quantum Cryptography: Third International Workshop, PQCrypto*, pages 73–80. Springer Berlin Heidelberg, 2010.

BJLM13. Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In *PQCrypto 2013*, pages 16–33, 2013.

BJMM12. Anja Becker, Antoine Joux, Alexandre May, and Alexandre Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1+1 = 0$ improves information set decoding. In *EUROCRYPT*, pages 520–536, 2012.

BM17.   Leif Both and Alexandre May. Optimizing BJMM with nearest neighbors: Full decoding in $2^{2n/21}$ and McEliece security. In *The Tenth International Workshop on Coding and Cryptography*, 2017.

CE05.   Andrew M. Childs and Jason M. Eisenberg. Quantum algorithms for subset finding. *Quantum Info. Comput.*, 5(7):593–604, 2005.

Chr17.   Tobias Christiani. A framework for similarity search with space-time trade-offs using locality-sensitive filtering. In *SODA*, pages 31–46, 2017.

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

FS09.   Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In *ASIACRYPT'09*, pages 88–105, 2009.

HK17.   Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in Euclidean norm. In *PKC*, 2017.

HKL18.   Gottfried Herold, Elena Kirshanova, and Thjis Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving. *PKC'18, to appear*, 2018.

IM98.   Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

KT17.   Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In *Post-Quantum Cryptography - PQCrypto*, 2017.

Laa15.   Thjis Laarhoven. Tradeoffs for nearest neighbors on the sphere. *CoRR*, abs/1511.07527, 2015.

McE78.   R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, pages 114–116, 1978.

MMT11.   Alexandre May, Alexandre Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.54n})$. In *ASIACRYPT*, volume 7073 of LNCS, pages 107–124, 2011.

MNRS11.   F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.

MO15.   Alexandre May and Ilya Ozerov. On computing nearest neighbors with applications to decodig of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015.

Pra62.   E. Prange. The use of information sets in decoding cyclic codes. In *IRE Transactions on Information Theory*, volume 6, pages 5–9, 1962.

SS81.   Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain np-complete problems. *SIAM J. Comput.*, 10:456–464, 1981.

Ste89.   Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications: 3rd International Colloquium*, pages 106–113, 1989.