Quantum Sieving for Code-Based Cryptanalysis and Its Limitations for ISD

Lynn Engelberts[1,2], Simona Etinski[1], and Johanna Loyer[1]
firstname.lastname@cwi.nl

[1] CWI, the Netherlands
[2] QuSoft, the Netherlands

Abstract. Sieving using near-neighbor search techniques is a well-known method in lattice-based cryptanalysis, yielding the current best runtime for the shortest vector problem in both the classical [BDGL16] and quantum [BCSS23] setting. Recently, sieving has also become an important tool in code-based cryptanalysis. Specifically, using a sieving subroutine, [GJN23, DEEK24] presented a variant of the information-set decoding (ISD) framework, which is commonly used for attacking cryptographically relevant instances of the decoding problem. The resulting sieving-based ISD framework yields complexities close to the best-performing classical algorithms for the decoding problem such as [BJMM12, BM18]. It is therefore natural to ask how well quantum versions perform. In this work, we introduce the first quantum algorithms for code sieving by designing quantum variants of the aforementioned sieving subroutine. In particular, using quantum-walk techniques, we provide a speed-up over the best known classical algorithm from [DEEK24] and over a variant using Grover's algorithm [Gro96]. Our quantum-walk algorithm exploits the structure of the underlying search problem by adding a layer of locality-sensitive filtering, inspired by the quantum-walk algorithm for lattice sieving from [CL21]. We complement our asymptotic analysis of the quantum algorithms with numerical results, and observe that our quantum speed-ups for code sieving behave similarly as those observed in lattice sieving. In addition, we show that a natural quantum analog of the sieving-based ISD framework does not provide any speed-up over the first presented quantum ISD algorithm [Ber10]. Our analysis highlights that the framework should be adapted in order to outperform the state-of-the-art of quantum ISD algorithms [KT17, Kir18].

Keywords. Quantum cryptanalysis, Quantum walks, Near-neighbor search, Code sieving, Decoding problem, Information-set decoding

1 Introduction

A fundamental problem in code-based cryptography is the decoding problem: given a linear code $\mathcal{C}$, find a codeword $\mathbf{x}_c \in \mathcal{C}$ of (small) fixed weight w.[3] The decoding problem is NP-hard in the worst case [BMvT78]. More important for cryptographic purposes, it is known that w can be chosen to guarantee the existence of exactly one solution on average, which gives us the so-called *unique decoding regime*. In this regime, the decoding problem is believed to be hard for a random instance of the problem. In particular, all known algorithms for attacking a random instance of this problem run in time and memory exponential in the input size. The best generic attacks[4] belong to the so-called *information-set decoding* (ISD) framework.

The ISD framework was originally introduced in the work of Prange [Pra62], and further improved using various techniques, including the meet-in-the-middle approach and its generalizations [Ste88, Dum91,SS81], representation techniques [MMT11,BJMM12], near-neighbor search techniques [MO15, Car20], and their combinations. Some of these techniques were adapted to the quantum setting [Ber10,KT17,Kir18]. To find the solution to the decoding problem, current ISD algorithms search for many partial solutions (by solving an instance of the decoding problem in a smaller dimension), and then check if any of those yields a solution to the original problem.

[3] This problem is called the codeword-finding problem in [DEEK24], and can be seen as a homogeneous version of the well-known syndrome decoding problem.

[4] For certain parameter regimes, the so-called statistical decoding attacks (also known as dual attacks in the lattice literature) are more efficient. As we are not particularly interested in a specific parameter regime, we will only consider attacks in the ISD framework.

Recently, a new ISD algorithm was proposed in [GJN23] and further generalized and improved in [DEEK24]. This approach uses a so-called *sieving technique* to find the partial solutions, which is well-known and widely applied in lattice-based cryptanalysis. The idea of a sieving algorithm is to start with a list of arbitrary elements and iteratively combine elements from the current list to obtain a new list of elements that satisfy a certain property which makes them (in some precise mathematical sense) 'closer' to being a solution. At the end, the algorithm outputs a list of solutions. In the code setting, the sieve starts with a list of arbitrary vectors, and in each iteration it combines pairs of vectors that satisfy an additional code constraint, eventually ending up with vectors in the code.

In each iteration of the sieve, the task of combining list elements to obtain a new list of suitable pairs can be formulated as an instance of a near-neighbor search problem (NNS). [DEEK24] presented several methods for solving this NNS problem, the best of which are based on locality-sensitive filtering (LSF) techniques. Using these methods, [DEEK24] obtained a sieving-based ISD algorithm for the decoding problem whose asymptotic runtime is close to that of the best known classical algorithms (such as [BJMM12, BM18]), with an improved time-memory trade-off over the previously known techniques.

While sieving algorithms are new in code-based cryptanalysis, they are abundant in lattice-based cryptanalysis, and belong to the state-of-the-art of classical *and* quantum algorithms for the shortest-vector problem. Classical algorithms for lattice sieving were first introduced by [AKS01] and later improved in [NV08, MV10, BDGL16], the latter of which introduced the LSF techniques in the Euclidean metric that were adapted in [DEEK24] for the Hamming metric. The addition of LSF gave a significant improvement in the runtime of lattice sieving. The first quantum speed-up for lattice sieving was obtained using a Grover-based algorithm [Laa15], which was further improved using quantum-walk techniques in [CL21]. This quantum-walk algorithm was further improved in [BCSS23] using reusable-walk techniques.

The various quantum speed-ups in lattice sieving raise a natural question of how well the corresponding quantum techniques perform when adapted to the setting of codes, which is the focus of this work. More precisely, in this work we aim to answer the following questions.

1. What is (an upper bound on) the quantum complexity of code sieving?
2. What is the runtime and memory of the quantum ISD algorithm that results from using our quantum algorithms for code sieving as a subroutine?

1.1 Main Contributions

Quantum algorithms for code sieving. We introduce the first quantum algorithms for code sieving by combining the classical sieving algorithm from [DEEK24] with the state-of-the-art techniques from quantum algorithms for lattice sieving [Laa15, CL21, BCSS23]. The sieving algorithm in [DEEK24] appears as a subroutine of their ISD algorithm, and solves a decoding problem with the goal of finding many codewords in a given code. This subroutine repeatedly solves a near-neighbor search problem of the following form: given a list $\mathcal{L}$ of N vectors in $\mathbb{F}_2^n$ of (Hamming) weight w, find all pairs in $\mathcal{L}$ whose sum is of weight w. The complexity of the best known algorithms for this NNS problem, and thereby the overall complexity of the sieving-based algorithm, depends on the cost of a subroutine that we call `FindSolutions` which, in short, searches for 'solutions' in a structured subset of the list $\mathcal{L}$. Similar to the lattice setting, our quantum algorithms aim to speed up this subroutine. We thus obtain quantum algorithms (and speed-ups) for this NNS problem, which might be of independent interest.

We present several quantum algorithms for `FindSolutions` and analyze their asymptotic time and memory complexities. Our first quantum algorithm is a straightforward application of Grover, and serves as a baseline for comparison. Our other quantum algorithms are a quantum-walk algorithm and a variant thereof, where we apply the sparsification technique from [CL21]. Both quantum-walk algorithms make use of locality-sensitive filtering (LSF) to obtain a speed-up over Grover in the search for 'solutions'. The use of LSF inside the quantum walk was first suggested by [CL21] in the context of lattice sieving, and one of our contributions is to show how to apply this layer of LSF in the context of codes (see Section 4.4). Specifically, we introduce a Hamming-metric variant of the 'residual vectors' used in [CL21], which behave somewhat differently than their Euclidean-metric analogs, but still allow us to apply LSF.

Besides providing an asymptotic analysis of the time and memory complexities of our quantum algorithms, we evaluate their performance through numerical optimization. Our numerical results

illustrate the obtained quantum speed-up over the classical algorithm from [DEEK24], as well as a comparison between our different quantum algorithms. In addition, we show that the quantum speed-ups we obtain align with the speed-ups observed in the state-of-the-art of lattice cryptanalysis (see Table 1). The Python code used for the numerical results of this work is available at `https://github.com/lynnengelberts/quantum-sieving-for-codes-public`.

Application to quantum ISD. The sieving framework from [GJN23, DEEK24] was introduced as a subroutine in an ISD algorithm to solve the decoding problem in the unique decoding regime. Since the resulting sieving-based ISD algorithm is shown to asymptotically perform nearly as good as the classical state-of-the-art for this problem, we focus on the question whether quantum analogs of this algorithm could have similar performance as the quantum state-of-the-art. In particular, we consider a natural quantum analog of sieving-based ISD that is obtained by allowing for a *quantum* sieving subroutine and using amplitude amplification. We show, using a combination of analysis and numerical optimization, that this quantum analog of sieving-based ISD *cannot* even improve on the first quantum algorithm for the decoding problem from [Ber10].

More precisely, we observe that the main limiting factor is a lower-bound constraint on the list size that is imposed in each iteration of the sieve, and which results in a lower bound on the time complexity of the resulting quantum ISD algorithm. Our results indicate that the sieving-based ISD framework should be adapted if it wants to compete with the best-performing quantum algorithms for the decoding problem. We suggest two natural attempts to adapt the sieving-based ISD framework, and explain why these do not allow for overcoming the found limitations.

In the end, our results show that code-based cryptosystems are still resilient to these quantum methods for code sieving inside the standard ISD framework. Given the quantum speed-ups in lattice-based cryptanalysis and the recent introduction of sieving techniques for codes, it was essential to evaluate their impact on the security of code-based schemes. Our new quantum algorithms for the NNS problem in the Hamming metric have the potential to be used in future algorithms for the decoding problem (for instance within new sieving algorithms that overcome the limitations addressed in this work, or within algorithms with completely different approaches). Identifying specific applications remains an open question for future work.

1.2 Outline

The paper is organized as follows. In Section 2, we introduce our notation and the relevant preliminaries. Section 3 describes the framework for code sieving, as well as near-neighbor search methods using locality-sensitive filtering. It also explains how the complexity of NNS and code sieving depends on the complexity of `FindSolutions`. In Section 4, we present our quantum algorithms for `FindSolutions` and their asymptotic complexity. Section 5 presents numerical results for the asymptotic runtime and memory of the introduced quantum algorithms. Finally, in Section 6, we adapt the classical sieving-based ISD framework from [DEEK24] to the quantum setting, and discuss its limitations.

1.3 Acknowledgements

The authors are grateful to Ronald de Wolf, Léo Ducas, and Elena Kirshanova for useful comments on the manuscript. The authors also thank Jean-Pierre Tillich for discussing this project in its early stage. LE was supported by the Dutch National Growth Fund (NGF), as part of the Quantum Delta NL program. SE and JL were supported by the ERC Starting Grant 947821 (ARTICULATE).

2 Preliminaries

2.1 Notation

Binary finite fields. We denote by $\mathbb{F}_2$ the binary finite field, and by $\mathbb{F}_2^n$ the corresponding vector space of dimension n. We write $+$ (resp. $\wedge$) for the bitwise 'XOR' (resp. 'AND') between two vectors in $\mathbb{F}_2^n$.

Scalars, vectors, matrices. A scalar is denoted by a non-bold, small letter, a vector is denoted by a bold small letter, and a matrix by a bold capital letter.

Asymptotic notation. We use standard Landau notation. Namely, we write $f(n) = O(g(n))$ if there exist constants $c, n_0 \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all integers $n \geq n_0$. We write $f(n) = \Omega(g(n))$ if there exist constants $c, n_0 \geq 0$ such that $f(n) \geq c \cdot g(n)$ for all integers $n \geq n_0$. We write $f(n) = \Theta(g(n))$ if both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. We define $\tilde{O}(f(n)) := O\left(f(n) \cdot \text{polylog}(f(n))\right)$, where $\text{polylog}(f(n)) := \log(f(n))^{O(1)}$, and define $\tilde{\Omega}$, $\tilde{\Theta}$ in a similar way. We write $f(n) = o(g(n))$ if for all constants $c > 0$, there exists $n_0 > 0$ such that $0 \leq f(n) < c \cdot g(n)$ for all integers $n \geq n_0$.

Other. The non-negative integers are denoted by $\mathbb{N}$. For any $n, k \in \mathbb{N}$ with $k \leq n$, we define the binomial coefficient by $\binom{n}{k} := \frac{n!}{k!(n-k)!}$.

2.2 Hamming Space

We are interested in computational problems over $\mathbb{F}_2^n$ endowed with the *Hamming metric* for $n \in \mathbb{N}$. In particular, we define the *Hamming weight* $|\cdot|$ of a vector as

$$\forall \mathbf{x} \in \mathbb{F}_2^n, \quad |\mathbf{x}| := \{i \in \{0, 1, \ldots, n-1\} \colon x_i \neq 0\}.$$

(The notation $|\cdot|$ will also be used to define the cardinality of a set.)

Definition 2.1 (Hamming sphere). *For any integer $0 \leq w \leq n$, we define the weight-w (Hamming) sphere as*

$$\mathcal{S}_w^n := \{\mathbf{x} \in \mathbb{F}_2^n \colon |\mathbf{x}| = w\}.$$

The surface area of a sphere (i.e., the size of the set $\mathcal{S}_w^n$) is calculated as $|\mathcal{S}_w^n| = \binom{n}{w}$.

Definition 2.2 (Region). *For any $\mathbf{c} \in \mathbb{F}_2^n$ and integer $0 \leq \alpha \leq |\mathbf{c}|$, we define*

$$\text{Region}_\alpha^n(\mathbf{c}) := \{\mathbf{x} \in \mathbb{F}_2^n \colon |\mathbf{x} \wedge \mathbf{c}| = \alpha\}.$$

The surface area of a region is $|\text{Region}_\alpha^n(\mathbf{c})| = \sum_{v=\alpha}^{n} \binom{|\mathbf{c}|}{\alpha}\binom{n-|\mathbf{c}|}{v-\alpha}$.

The intersection of a sphere with a region is called a *(spherical) cap*.

Definition 2.3 (Spherical cap). *For any $\mathbf{c} \in \mathbb{F}_2^n$ and integers α, v with $0 \leq \alpha \leq v \leq n$ and $\alpha \leq |\mathbf{c}|$, we define*

$$\mathcal{C}_{v,\alpha}^n(\mathbf{c}) = \mathcal{S}_v^n \cap \text{Region}_\alpha^n(\mathbf{c}) := \{\mathbf{x} \in \mathcal{S}_v^n \colon |\mathbf{x} \wedge \mathbf{c}| = \alpha\}.$$

The surface area of a spherical cap is $|\mathcal{C}_{v,\alpha}^n(\mathbf{c})| = |\mathcal{S}_v^n \cap \text{Region}_\alpha^n(\mathbf{c})| = \binom{|\mathbf{c}|}{\alpha}\binom{n-|\mathbf{c}|}{v-\alpha}$.

Furthermore, the intersection of two caps (on the same sphere) is called a *(spherical) wedge*.

Definition 2.4 (Spherical wedge). *For any $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ and integers α, v with $0 \leq \alpha \leq v \leq n$ and $\alpha \leq \min(|\mathbf{x}|, |\mathbf{y}|)$, we define*

$$\begin{aligned}\mathcal{W}_{v,\alpha}^n(\mathbf{x}, \mathbf{y}) &:= \mathcal{C}_{v,\alpha}^n(\mathbf{x}) \cap \mathcal{C}_{v,\alpha}^n(\mathbf{y}) = \mathcal{S}_v^n \cap \text{Region}_\alpha^n(\mathbf{x}) \cap \text{Region}_\alpha^n(\mathbf{y}) \\ &= \{\mathbf{c} \in \mathcal{S}_v^n \colon |\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = \alpha\}.\end{aligned}$$

We are particularly interested in the case where $\mathbf{x}$ and $\mathbf{y}$ are of equal Hamming weight. Then the surface area of a spherical wedge is given by the following lemma.

Lemma 2.5 (Surface area of a wedge). *Let $\mathbf{x}, \mathbf{y} \in \mathcal{S}_w^n$ and define $w^* := |\mathbf{x} \wedge \mathbf{y}|$. For all integers α, v with $0 \leq \alpha \leq v \leq n$ and $\alpha \leq w$, we have that*

$$|\mathcal{W}_{v,\alpha}^n(\mathbf{x}, \mathbf{y})| = \sum_{e=\max(0,2\alpha-v)}^{\min(\alpha,w^*)} \binom{w^*}{e}\binom{w-w^*}{\alpha-e}^2\binom{n-2w+w^*}{v-2\alpha+e}.$$

Note that e ranges over all possible values of $|\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}|$.

Proof. (The case $w^* = \frac{w}{2}$ was already proven in [DEEK24].) We want to count the number of $\mathbf{c} \in S_v^n$ such that $|\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = \alpha$. For each such vector $\mathbf{c}$, it holds that $\max(0, 2\alpha - v) \leq |\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}| \leq \min(\alpha, w^*)$, where the inequality $2\alpha - v \leq |\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}|$ follows from the fact that $v = |\mathbf{c}|$ must be at least $|\mathbf{x} \wedge \mathbf{c} \wedge \mathbf{y}| + |\mathbf{x} \wedge \mathbf{c} \wedge \overline{\mathbf{y}}| + |\mathbf{y} \wedge \mathbf{c} \wedge \overline{\mathbf{x}}| = |\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}| + 2(\alpha - |\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}|) = 2\alpha - |\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}|$ (where we write $\overline{\mathbf{x}}$ for the bitwise negation of $\mathbf{x}$). In particular, we have the partition

$$\mathcal{W}_{v,\alpha}^n(\mathbf{x}, \mathbf{y}) = \bigsqcup_{e=\max(0,2\alpha-v)}^{\min(\alpha,w^*)} S_e$$

where $S_e := \{\mathbf{c} \in \mathcal{S}_v^n : |\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = \alpha, |\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}| = e\}$ (note that the sets S_e are clearly disjoint). The claim then follows from the observation that $|S_e| = \binom{w^*}{e}\binom{w-w^*}{\alpha-e}^2\binom{n-(2w-w^*)}{v-2\alpha+e}$. □

Remark 2.6 (Surface area only depends on the center weight). Note that the surface area of a cap (resp. wedge) only depends on the *weight* of the center $\mathbf{c}$ (resp. on the weight of $\mathbf{x}, \mathbf{y}$ and on their overlap).

2.3 Linear Codes and Random Codes

The problems and algorithms in this work consider binary linear codes.

Definition 2.7 (Binary linear code). *An $[n, k]$ binary linear code $\mathcal{C}$ is defined as a linear subspace of size 2^k of $\mathbb{F}_2^n$. Elements of $\mathcal{C}$ are called codewords.*

A code can be represented by a *generator matrix*: a full-rank matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ whose rows form a basis of the subspace $\mathcal{C}$. Conversely, any such matrix defines a code.

We are primarily interested in *random binary linear codes* (abbr. *random codes*), namely codes obtained from sampling a generator matrix uniformly at random from $\mathbb{F}_2^{k \times n}$. These codes can be sampled in time polynomial in n.

2.4 Quantum Computation

We say that a quantum algorithm has time complexity T if the circuit describing it uses at most T elementary quantum gates. When T is polynomial in the number of bits needed to write down the input, we say the algorithm is *efficient*.

QRAM model and complexities. Our results hold in the quantum circuit model model where QRAM operations are assumed efficiently implementable. We consider separately the quantum random access to classical memory (QRACM) and quantum random access to quantum memory (QRAQM). More precisely, a QRACM operation performs the unitary $O_x : |i\rangle|0\rangle \mapsto |i\rangle|x_i\rangle$ for some register $x = \{x_0, \ldots, x_{N-1}\}$ that is classically stored and quantumly accessible. A QRAQM operator applies the unitary: $O'_x : \left(\bigotimes_{j=0}^{N-1} |x_j\rangle\right)|i\rangle|0\rangle \mapsto \left(\bigotimes_{j=0}^{N-1} |x_j\rangle\right)|i\rangle|x_i\rangle$. For the memory complexities, we will specify the sizes of the classical and quantum memories, and more specifically the sizes of the classical and quantum registers that the respective QRAM operator has access to.

Grover's algorithm and amplitude amplification. We recall Grover's algorithm and a generalization of it, commonly known as amplitude amplification.

Theorem 2.8 (Grover's algorithm). *For $N \in \mathbb{N}$, let $f : [N] \to \{0, 1\}$ be a function and define $t := |\{i \in [N] : f(i) = 1\}|$. There exists a quantum algorithm, called* Grover's algorithm*, that returns i such that $f(i) = 1$, if such an i exists, using $O(\sqrt{N/\max\{1, t\}})$ queries to f and $\tilde{O}(\sqrt{N/\max\{1, t\}})$ elementary gates with probability at least $2/3$. It uses $\max(\log_2(x_i))$ quantum memory and requires efficient QRAM access to the register $x \in \{0, 1\}^N$ with $x_i = f(i)$. In particular, if the register is classically stored, it requires QRACM of size N, and if it is quantumly stored, QRAQM of size N.*

Grover's algorithm as originally presented in [Gro96] applies to the case of a unique solution; the extension to multiple solutions was detailed in [BBHT98]. When t is known, then Grover's algorithm can be adapted to have success probability 1. When t is unknown, then it is still possible to find $i \in [N]$ such that $f(i) = 1$ with an expected number of $O(\sqrt{N/t})$ queries to f (and an expected number of $\tilde{O}(\sqrt{N/t})$ elementary gates) if $t > 0$. In this work, when we apply Grover's algorithm we always know an estimate of the number of solutions t.

Theorem 2.9 (Amplitude amplification [BHMT02]). *For $N \in \mathbb{N}$, let $f\colon [N] \to \{0,1\}$ be a function. Suppose there is an efficient quantum circuit U_f that maps $|i\rangle \to (-1)^{f(i)}|i\rangle$. Suppose $\mathcal{A}$ is an algorithm that can be implemented as a reversible quantum circuit and that returns $i \in [N]$ such that $f(i) = 1$ in time T with success probability $p > 0$. Then there exists a quantum algorithm, called* amplitude amplification*, that returns $i \in [N]$ such that $f(i) = 1$ with probability at least $\max(1-p, p)$ in time $O(1/\sqrt{p}) \cdot T$. If algorithm $\mathcal{A}$ uses quantum access to a classical register (resp. quantum), then running amplitude amplification requires QRACM (resp. QRAQM) operators.*

Moreover, for known p, there is a variant of amplitude amplification with success probability 1, which uses $O(1/\sqrt{p})$ applications of $\mathcal{A}$ and U_f [BHMT02, Theorem 4]. When p is unknown, then there is a variant that finds an $i \in [N]$ such that $f(i) = 1$ using an expected number of $\Theta(1/\sqrt{p})$ applications of $\mathcal{A}$ and U_f as long as $p > 0$ [BHMT02, Theorem 3].

Finally, note that we can reduce the error probability of Grover's algorithm and amplitude amplification using standard methods: using $O(\log_2(1/\epsilon))$ repetitions it can be bounded from above by an arbitrarily small $\epsilon > 0$.

Quantum walk. We will consider the quantum walks as presented in [MNRS11]. A quantum walk starts with an undirected graph $G = (V, E)$, with V the set of vertices and $E \subseteq V \times V$ the set of edges. The set $M \subseteq V$ contains elements said marked, and the goal of a quantum walk is to return a marked vertex $v \in M$. For any vertex $v \in V$, we define $N(x) := \{v' : (v, v') \in E\}$ the set of neighbors of v, and $|p_v\rangle = \sum_{y \in N(v)} \frac{1}{\sqrt{|N(v)|}} |v'\rangle$. We now define the following quantities:

- Set-up cost S is the cost of the unitary map $|0\rangle \mapsto \frac{1}{\sqrt{|V|}} \sum_{v \in V} |v\rangle |p_v\rangle$.
- Update cost U is the cost of the unitary map $|v\rangle |0\rangle \mapsto |v\rangle |p_v\rangle$.
- Checking cost C is the cost of computing the function $f : V \to \{0,1\}$ where $f(v) = 1 \Leftrightarrow v \in M$.
- $\epsilon = \frac{|M|}{|V|}$ is the fraction of marked vertices.
- δ is the spectral gap of G, which is defined as $\delta := 1 - |\lambda|$, where λ is the second largest eigenvalue (in magnitude) of the normalized adjacency matrix of G.

Proposition 2.10. *[MNRS11] There exists a quantum-walk algorithm that finds a marked element $v \in M$ in time*

$$O\left(\mathsf{S} + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)\right).$$

Johnson graph. For positive integers $r < n$, the Johnson graph $J(n, r)$ is a standard graph to run quantum walks, and is for instance used in quantum-walk algorithms for solving collision problems. Each vertex v consists of r distinct (unordered) points $x_1, \ldots, x_r \in [n]$ and some additional data $D(v)$. Two vertices $v = (x_1, \ldots, x_r, D(v))$ and $v' = (x'_1, \ldots, x'_r, D(v'))$ form an edge in $J(n, r)$ iff $|\{x_1, \ldots, x_r\} \cap \{x'_1, \ldots, x'_r\}| = r - 1$. It is well-known that the graph $J(n, r)$ has spectral gap $\delta = \Omega(\frac{1}{r})$.

[Amb07] and [BJLM13] presented quantum data structures that use efficient QRAM to perform efficient insertion and deletion of elements in quantum superposition. We refer the reader to these papers for more details.

3 Code Sieving using NNS Techniques

Sieving using a near-neighbor search (NNS) subroutine is a well-known method in lattice-based cryptanalysis (e.g., see [BDGL16]) and recently has become an important tool in code-based cryptanalysis as well. Our quantum algorithms for code sieving are based on the sieving framework that was introduced in [GJN23] and further improved and generalized in [DEEK24]. In this section, we recall this sieving framework and its complexity (Theorem 3.8). We begin by stating the computational problem that is tackled by the sieving framework, namely, the *decoding problem*.

Problem 3.1 (Decoding problem, DP(n, k, w, N)). Given an $[n, k]$ binary linear code $\mathcal{C}$ and an integer value w, find N codewords $\mathbf{x}_c \in \mathcal{C}$ of weight $|\mathbf{x}_c| := w$.

We focus on the expected runtime of algorithms that solve a *random* instance of Problem 3.1. That is, we assume that $\mathcal{C}$ is a random $[n, k]$ linear code. Given such a code $\mathcal{C}$, the expected number of codewords in $\mathcal{C}$ of weight w is $\binom{n}{w}/2^{n-k}$. For the decoding problem $\mathrm{DP}(n, k, w, N)$ to be well-defined, we thus require $N = O\left(\frac{\binom{n}{w}}{2^{n-k}}\right)$.

Remark 3.2. In the introduction (and later in Section 6), we considered the decoding problem as Problem 3.1 with $N = 1$, as the presumed hardness of this variant underlies the security of code-based cryptographic primitives. The case for arbitrary N naturally appears as a subinstance in the ISD framework, and in particular it is the problem that is tackled by the sieving subroutine in the ISD algorithms from [GJN23, DEEK24]. Therefore, in the remainder of this section, we consider this problem for arbitrary N.

3.1 Framework for Code Sieving

Recently, sieving-based algorithms were presented for this decoding problem [GJN23,DEEK24]. These algorithms make use of an oracle that solves NNS in $\mathbb{F}_2^n$ endowed with Hamming metric, which is defined as follows.

Problem 3.3 (Near-neighbor search, NNS(n, w, N)). Let $n, w \in \mathbb{N}$ with $w \leq n$. Given a list $\mathcal{L}$ of N vectors sampled independently and uniformly at random from S_w^n, find a $(1 - o(1))$-fraction of all pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}^2$ satisfying $|\mathbf{x} + \mathbf{y}| = w$ (called *solution pairs*).

The basic idea of code sieving [GJN23, DEEK24] for solving $\mathrm{DP}(n, k, w, N)$ is to start from a list of arbitrary vectors of Hamming weight w and then iteratively add code constraints to obtain a required number of codewords of Hamming weight w. These iteratively added code constraints define a so-called *tower of codes*: a collection $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$ of codes with $\mathcal{C}_{i-1} \supseteq \mathcal{C}_i$ and dimension decrements of 1, starting with the 'initial' code $\mathbb{F}_2^n$ and ending with the input code $\mathcal{C}$.

When going from $\mathcal{C}_{i-1}$ to $\mathcal{C}_i$, the addition of one constraint results in halving the linear subspace $\mathcal{C}_{i-1}$. Thus, in expectation only half of the vectors from the code $\mathcal{C}_{i-1}$ are in the code $\mathcal{C}_i$. To avoid an exponential drop, Algorithm 1 therefore instead creates new elements from the elements of the previous iteration. Specifically, this is done by adding pairwise sums of the elements from the previous list $\mathcal{L}_{i-1}$ to the new list $\mathcal{L}_i$, which is a technique also used in lattice sieving. To improve the algorithm's performance, instead of searching through all pairs of elements from $\mathcal{L}_{i-1}$, the algorithm searches through the list of *near neighbors*, $\{(\mathbf{x}, \mathbf{y}) \colon \mathbf{x}, \mathbf{y} \in \mathcal{L}_{i-1}, |\mathbf{x} + \mathbf{y}| = w\}$, namely the elements that are at small Hamming distance in $\mathbb{F}_2^n$. Among the near neighbors $(\mathbf{x}, \mathbf{y})$, the algorithm then adds $\mathbf{x} + \mathbf{y}$ to $\mathcal{L}_i$ for those $\mathbf{x} + \mathbf{y}$ that belong to $\mathcal{C}_i$.

Algorithm 1 presents the overall sieving framework for solving the decoding problem (Problem 3.1) in more detail. Note that the time complexity of this sieving algorithm (Algorithm 1) is determined by the cost of one iteration, i.e., by the cost of solving $\mathrm{NNS}(n, w, N)$ (Problem 3.3). We therefore state the time and memory complexity of Algorithm 1 after presenting the state-of-the-art algorithm from [DEEK24] for $\mathrm{NNS}(n, w, N)$ in Section 3.2.

Remark 3.4 (Difference with lattice sieving). Note the difference with lattice sieving, where one starts with a list of long lattice vectors, and iteratively combines them to obtain *shorter* lattice vectors. Another difference is that code sieving was presented in [GJN23, DEEK24] as a subroutine in an ISD algorithm, whereas lattice sieving is the main algorithm. (We elaborate on the application to ISD in Section 6.)

Besides the upper bound on the output size N that is imposed by the decoding problem itself, the construction of the sieving algorithm also puts a lower bound on the size N used in the algorithm. Specifically, Lemma 3.5 shows that if N is too small, then the expected number of solutions to $\mathrm{NNS}(n, w, N)$ is significantly smaller than N. As a result, the current list size would shrink in each iteration, and the output list cannot be of size $\Omega(N)$.

Algorithm 1: Code sieving using NNS

Input : $[n, k]$ binary linear code $\mathcal{C} \subseteq \mathbb{F}_2^n$, weight w, output size N, oracle $\mathcal{A}_{\text{NNS}}$ for $\text{NNS}(n, w, N)$
Output: $\mathcal{L} \subseteq \mathcal{C} \cap \mathcal{S}_w^n$ of size N

Initialization:

Sample a tower of codes $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$, with dimension decrements of 1.
Sample N vectors independently and uniformly at random from $\mathcal{S}_w^n$ and add them to a list $\mathcal{L}_0$.

Sieving part:

for $i = 1$ *to* $n - k$ **do**
    Invoke $\mathcal{A}_{\text{NNS}}$ on $\mathcal{L}_{i-1}$ to obtain $\mathcal{L}_i' := \{(\mathbf{x}, \mathbf{y}) \colon \mathbf{x}, \mathbf{y} \in \mathcal{L}_{i-1}, |\mathbf{x} + \mathbf{y}| = w\}$.
    for $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}_i'$ **do**
        if $\mathbf{x} + \mathbf{y} \in \mathcal{C}_i$ **then**
            Add $\mathbf{x} + \mathbf{y}$ to $\mathcal{L}_i$.
    Discard some elements if $|\mathcal{L}_i| > N$.
return $\mathcal{L}_{n-k}$

Lemma 3.5 ([DEEK24]). *Let $\mathcal{L}$ be a set containing N vectors sampled uniformly and independently at random from S_w^n. Then the expected number of pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}^2$ satisfying $|\mathbf{x} + \mathbf{y}| = w$ is*

$$N^2 \frac{\binom{w}{w/2}\binom{n-w}{w/2}}{\binom{n}{w}}.$$

In particular, the expected number of such pairs is at least N if and only if $N \geq \frac{\binom{n}{w}}{\binom{w}{w/2}\binom{n-w}{w/2}}$.

Proof. (This was also observed in [DEEK24].) For $\mathbf{x}, \mathbf{y} \in \mathcal{S}_w^n$ sampled independently and uniformly at random, the probability that they satisfy $|\mathbf{x} + \mathbf{y}| = w$ is equal to $\binom{w}{w/2}\binom{n-w}{w/2}/\binom{n}{w}$. To see this, one could use the fact that for any $\mathbf{x}, \mathbf{y} \in \mathcal{S}_w^n$, the condition $|\mathbf{x} + \mathbf{y}| = w$ is equivalent to the condition that $|\mathbf{x} \wedge \mathbf{y}| = w/2$. It follows that the expected number of pairs satisfying this condition is $N^2\binom{w}{w/2}\binom{n-w}{w/2}/\binom{n}{w}$. Therefore, the expected number of such pairs is at least N if and only if $N \geq \binom{n}{w}/\left(\binom{w}{w/2}\binom{n-w}{w/2}\right)$. □

By Lemma 3.5, if $N \geq \binom{n}{w}/\left(\binom{w}{w/2}\binom{n-w}{w/2}\right)$, then $\text{NNS}(n, w, N)$ has at least N solutions in expectation. However, note that, in each iteration i of Algorithm 1, on average only a quarter of the NNS solutions $(\mathbf{x}, \mathbf{y})$ found by $\mathcal{A}_{\text{NNS}}$ will be added to the new list $\mathcal{L}_i$. Indeed, half of them is discarded as each pair is counted twice: $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{y}, \mathbf{x})$. Secondly, the algorithm discards those $\mathbf{x} + \mathbf{y}$ that do not belong to $\mathcal{C}_i$. Since two subsequent codes $\mathcal{C}_{i-1}, \mathcal{C}_i$ differ by one dimension, it follows that in expectation another half of the NNS solutions are discarded. Therefore, [DEEK24] propose to take $N \geq 4 \cdot \binom{n}{w}/\left(\binom{w}{w/2}\binom{n-w}{w/2}\right)$ to maintain a list size of N through all iterations in the sieving part.

It is important to emphasize that the application of Lemma 3.5 in the above discussion is justified if the distribution of the list elements does not change throughout the iterations. In fact, to guarantee that the output list contains N vectors (for N within the stated bounds), the sieving algorithm from [DEEK24] relies on the following heuristic.

> *In each iteration of Algorithm 1, the elements in the current list 'behave like' vectors distributed independently and uniformly at random over $\mathcal{S}_w^n$ in the sense that, even if some correlations appear in an iteration of the algorithm, it does not significantly affect the runtime of the algorithm.*

See [DEEK24] for a more precise formulation of this heuristic and experimental verification.[5]

[5] A similar heuristic is used in the most efficient lattice-based sieving approaches (in particular, [NV08,MV10] and their classical and quantum derivatives [BDGL16, Laa15, CL21, BCSS23]).

3.2 NNS using Locality-Sensitive Filtering (LSF)

The algorithms for the NNS problem in the Hamming metric proposed in [Car20, GJN23, DEEK24] can all be formulated using the locality-sensitive filtering (LSF) framework in the Hamming metric. Notably, the best-performing algorithms for NNS in the Euclidean metric [BDGL16] also employ LSF techniques. We start by recalling this approach.

The underlying idea of locality-sensitive filtering (in $\mathbb{F}_2^n$) is the following: vectors $\mathbf{x}, \mathbf{y}$ satisfying $|\mathbf{x} + \mathbf{y}| = w$, also known as *near neighbors*, can be found more efficiently if we restrict our search to *local regions* of $\mathbb{F}_2^n$. The algorithm proceeds as follows. It starts with covering the space $\mathbb{F}_2^n$ with (potentially overlapping) regions, each region corresponding to a certain center $\mathbf{c} \in \mathcal{C}_f$, for some subset $\mathcal{C}_f \subseteq \mathbb{F}_2^n$. Each vector $\mathbf{x}$ from the input list $\mathcal{L}$ is then *filtered* according to the centers, namely, it is inserted into a *bucket* corresponding to a center $\mathbf{c}$ if and only if $|\mathbf{x} \wedge \mathbf{c}| = \alpha$.[6] This is what we refer to as the *bucketing phase*. Each bucket thus contains all the elements from $\mathcal{L}$ that belong to the same region. The algorithm then searches for near neighbors within each bucket, which we refer to as the *checking phase*. As the checking phase significantly affects the overall cost of the algorithm, our primary goal will be to improve its runtime by quantizing this part of the algorithm.

We formalize the notion of bucket as follows, and define the set of *valid centers* for a given vector.

Definition 3.6 (Bucket). *For $\mathbf{c} \in \mathbb{F}_2^n$, integers $0 \leq \alpha \leq w \leq n$ with $\alpha \leq |\mathbf{c}|$ and $\mathcal{L} \subseteq \mathcal{S}_w^n$, define*

$$\mathcal{B}_\alpha(\mathbf{c}) := \{\mathbf{x} \in \mathcal{L} \colon |\mathbf{x} \wedge \mathbf{c}| = \alpha\}.$$

Definition 3.7 (Valid centers). *For $\mathbf{x} \in \mathbb{F}_2^n$, integers $0 \leq \alpha \leq n$ with $\alpha \leq |\mathbf{x}|$ and $\mathcal{C}_f \subseteq \mathbb{F}_2^n$, define*

$$\mathcal{VC}_\alpha(\mathbf{x}) := \{\mathbf{c} \in \mathcal{C}_f \colon |\mathbf{x} \wedge \mathbf{c}| = \alpha\}.$$

The resulting NNS algorithm is presented in Algorithm 2. It uses two subroutines: `FindValidCenters` and `FindSolutions`. For a given $\mathbf{x} \in \mathcal{L}$ and bucketing parameter α, the `FindValidCenters` subroutine returns a set $\mathcal{VC}_\alpha(\mathbf{x}) = \{\mathbf{c} \in \mathcal{C}_f \mid |\mathbf{x} \wedge \mathbf{c}| = \alpha\}$. For a suitable choice of $\mathcal{C}_f$ (using so-called random product codes), the time complexity of this subroutine has been shown by [DEEK24] to be $|\mathcal{VC}_\alpha(\mathbf{x})| + 2^{o(n)}$, as explained in the next section.

The subroutine `FindSolutions` returns all pairs $(\mathbf{x}, \mathbf{y})$ of vectors in the bucket $\mathcal{B}_\alpha(\mathbf{c})$ satisfying $|\mathbf{x}+\mathbf{y}| = w$. Classically, we obtain an algorithm for `FindSolutions` with expected runtime $\mathbb{E}[|\mathcal{B}_\alpha(\mathbf{c})|^2]$ by searching through all pairs in $\mathcal{B}_\alpha(\mathbf{c})$. (Although [DEEK24] consider a slightly different approach for the checking phase, it results in the same asymptotic runtime for NNS.) In Section 4, we focus on speeding up the search in the buckets by presenting several quantum algorithms for `FindSolutions`.

Algorithm 2: NNS using locality-sensitive filtering

Input : weight w, input list $\mathcal{L} \subseteq \mathcal{S}_w^n$ of size N, set of centers $\mathcal{C}_f$, bucketing parameter α
Output: list $\mathcal{L}'$ containing pairs $\mathbf{x}, \mathbf{y} \in \mathcal{L}$ with $|\mathbf{x} + \mathbf{y}| = w$

Initialization:
for $\mathbf{c} \in \mathcal{C}_f$ **do**
  $\mathcal{B}_\alpha(\mathbf{c}) = \emptyset$.

Bucketing Phase:
for $\mathbf{x} \in \mathcal{L}$ **do**
  for $\mathbf{c} \in$ `FindValidCenters`$(\mathcal{C}_f, \mathbf{x}, \alpha)$ **do**
    Add $\mathbf{x}$ to $\mathcal{B}_\alpha(\mathbf{c})$.

Checking Phase:
$\mathcal{L}' = \emptyset$.
for $\mathbf{c} \in \mathcal{C}_f$ **do**
  Add `FindSolutions`$(\mathcal{B}_\alpha(\mathbf{c}))$ to $\mathcal{L}'$.
return $\mathcal{L}'$

[6] The analog in the lattice setting is that a unit vector is added to a bucket corresponding to some unit vector $\mathbf{c} \in \mathbb{R}^n$ if it has (absolute) inner product at least α with $\mathbf{c}$.

Depending on the size of $|\mathcal{C}_f|$, Algorithm 2 has to be repeated a certain number of times to find all solutions. The size of $|\mathcal{C}_f|$ also affects other factors of the time and memory complexity of Algorithm 2, and should thus be chosen carefully. The best time and memory complexity [DEEK24, Cor. 4.2] is obtained by choosing $|\mathcal{C}_f|$ such that $\mathbb{E}[|\mathcal{VC}_\alpha(\mathbf{x})|] = 2^{o(n)}$, ensuring that the expected runtime of `FindValidCenters` is $2^{o(n)}$. (An algorithm with similar time and memory complexities as [DEEK24, Cor. 4.2] was independently obtained by Carrier [Car20, Cor. 8.2.6].)

More generally, given a classical or quantum algorithm for `FindSolutions`, we obtain the following time and memory complexities for $\mathrm{NNS}(n, w, N)$, and thus for $\mathrm{DP}(n, k, w, N)$ by instantiating the oracle $\mathcal{A}_{\mathrm{NNS}}$ in Algorithm 1 with the obtained algorithm for $\mathrm{NNS}(n, w, N)$.

Theorem 3.8 (Variant of [Car20, Cor. 8.2.6] and [DEEK24, Cor. 4.2, Theorem 3.2]). *Consider $n, w, N \in \mathbb{N}$ with $w = \Theta(n)$ such that NNS(n, w, N) (Problem 3.3) is well-defined. For positive integers $v, \alpha = \Theta(n)$ and arbitrary $\mathbf{x}, \mathbf{y} \in \mathcal{S}_w^n$ such that $|\mathbf{x} + \mathbf{y}| = w$, let $|\mathcal{C}_f| = 2^{o(n)} \cdot |\mathcal{S}_v^n|/|\mathcal{C}_{v,\alpha}^n(\mathbf{x})|$ and $R = 2^{o(n)} \cdot |\mathcal{C}_{v,\alpha}^n(\mathbf{x})|/|\mathcal{W}_{v,\alpha}^n(\mathbf{x}, \mathbf{y})|$.*

Then, given a classical, resp. quantum, algorithm for `FindSolutions` *with expected runtime T, there exists a classical, resp. quantum, algorithm that solves NNS(n, w, N) in expected time*

$$R \cdot (2^{o(n)} \cdot N + |\mathcal{C}_f| \cdot T)$$

using R calls to Algorithm 2. Moreover, the classical memory is of expected size $2^{o(n)} \cdot N$, and the other memory complexities are the same as for the `FindSolutions` *subroutine.*

In addition, if $4 \cdot \binom{n}{w} / \left(\binom{w}{w/2}\binom{n-w}{w/2}\right) \leq N \leq \binom{n}{w}/2^{n-k}$ and $k = \Theta(n)$ is such that the binary-sieve heuristic in [DEEK24] holds for (n, k, w, N), then there exists a classical, resp. quantum, sieving-based algorithm that solves DP(n, k, w, N) (Problem 3.1) with the same time and memory complexity, up to polylogarithmic factors.

Note that the first part follows from [Car20, Cor. 8.2.6] and [DEEK24, Cor. 4.2] when allowing for a quantum subroutine, and that the second part follows from [DEEK24, Theorem 3.2].

3.3 Random Product Codes and an Efficient Algorithm for `FindValidCenters`

To efficiently perform the `FindValidCenters` subroutine, [DEEK24] suggest to let $\mathcal{C}_f$ be a binary linear code for which there exists an efficient decoding algorithm. More precisely, they use the notion of *random product codes*, originally introduced in [BDGL16] for $\mathbb{R}^n$, and defined as follows for $\mathbb{F}_2^n$.

Remark 3.9. A similar notion of random product codes and a corresponding decoding algorithm was also independently presented (in French) in [Car20, Section 9.1] for a related problem.

Definition 3.10 (Random product code (RPC) in $\mathbb{F}_2^n$). *An (n, v, t)-RPC of size κ is an element $\mathcal{C}$ drawn uniformly at random from the set*

$$R_{n,v,t,\kappa} := \{\mathcal{C} = \mathcal{C}^{(1)} \times \cdots \times \mathcal{C}^{(t)} \colon \mathcal{C}^{(i)} \subseteq S_{v/t}^{n/t} \text{ such that } \forall i, |\mathcal{C}^{(i)}| = \kappa^{1/t}\}.$$

We write $\mathcal{C} \sim R_{n,v,t,\kappa}$.

The set of centers is then sampled uniformly at random from $R_{n,v,t,\kappa}$ to guarantee:

(1) Efficient decodability: in some reasonable parameter regimes, one can compute $\mathcal{VC}_\alpha(\mathbf{x}) := \{\mathbf{c} \in \mathcal{C} \colon |\mathbf{x} \wedge \mathbf{c}| = \alpha\}$ in time asymptotically equal to its size $|\mathcal{VC}_\alpha(\mathbf{x})|$.
(2) Random behavior: for t not too large, a sample $\mathcal{C}$ behaves like a fully random code in the sense that the success probability that $\mathcal{C} \sim R_{n,v,t,\kappa}$ 'captures' a pair $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}^2$ is the same (up to factors subexponential in n) as for a fully random code in S_v^n.[7]

For more details, we refer to [DEEK24]. As we will refer back to the first property later in this work, we state the corresponding result here (which implicitly follows from the application of [DEEK24, Lemma 4.5] in the proof of [DEEK24, Theorem 4.4]).

[7] This 'randomness' property puts a constraint on t. More precisely, as described in [DEEK24] (and in [BDGL16] for $\mathbb{R}^n$), we would like t to be small enough to guarantee that we can approximate a cap/wedge in $\mathbb{F}_2^n$ by the Cartesian product of t caps/wedges in $\mathbb{F}_2^{n/t}$. By Lemma 4.6 and Lemma 4.7 in [DEEK24], these approximations are satisfied up to a subexponential factor in n if $t = o(n/\log(n))$.

Theorem 3.11 (Efficient decodability of RPC [DEEK24]). *Let $n \in \mathbb{N}$ and let $w, v, \alpha = \Theta(n), \kappa$ be positive integers. For $t = \Theta(\sqrt{n})$, let $\mathcal{C}$ be an (n, v, t)-RPC of size exponential in n. Then there exists a classical algorithm that, for any $\mathbf{x} \in \mathcal{S}_w^n$, computes the set $\mathcal{VC}_\alpha(\mathbf{x}) := \{\mathbf{c} \in \mathcal{C} \colon |\mathbf{x} \wedge \mathbf{c}| = \alpha\}$ in time $|\mathcal{VC}_\alpha(\mathbf{x})| + 2^{o(n)}$.*

4 Quantum Algorithms for NNS and Code Sieving

In this section, we present the first quantum algorithms for code sieving. More precisely, we describe different quantum algorithms for the `FindSolutions` procedure in the NNS subroutine (Algorithm 2), and analyze the time and memory complexities of our algorithms. By plugging our results (i.e., Theorem 4.5 and Theorem 4.7) into Theorem 3.8, we obtain the time and memory complexity of the resulting quantum algorithms for NNS and, hence, code sieving. Our numerical results in Section 5 illustrate the quantum speed-ups obtained for these resulting algorithms.

We recall the context in which `FindSolutions` is used. That is, consider the successful completion of the bucketing phase in Algorithm 2 on an input list $\mathcal{L} \subseteq \mathcal{S}_w^n$: for some bucketing parameter α, we have sampled a set $\mathcal{C}_f \subseteq \mathcal{S}_v^n$ of bucket centers and an associated data structure containing for each $\mathbf{c} \in \mathcal{C}_f$ the bucket

$$\mathcal{B}_\alpha(\mathbf{c}) := \{\mathbf{x} \in \mathcal{L} \colon |\mathbf{x} \wedge \mathbf{c}| = \alpha\} \subseteq \mathrm{Region}_\alpha(\mathbf{c}) := \{\mathbf{x} \in \mathcal{S}_w^n : |\mathbf{x} \wedge \mathbf{c}| = \alpha\}. \tag{1}$$

Then the goal of `FindSolutions` is to find all $\mathbf{x}, \mathbf{y} \in \mathcal{B}_\alpha(\mathbf{c})$ satisfying $|\mathbf{x} + \mathbf{y}| = w$.

Note that if $\mathcal{L} \subseteq \mathcal{S}_w^n$ is sampled independently and uniformly at random, then the vectors in $\mathcal{B}_\alpha(\mathbf{c})$ are distributed independently and uniformly over $\mathrm{Region}_\alpha(\mathbf{c})$. Hence, formally, our quantum algorithms for `FindSolutions` solve the following problem.

Problem 4.1 (Bucket search). Let $B, n, w, v, \alpha \in \mathbb{N}$ with $w, v \leq n$ and $\alpha \leq \max\{v, w\}$. Let $\mathbf{c} \in \mathcal{S}_v^n$. Given a list $\mathcal{B}_\alpha(\mathbf{c})$ of B vectors that are sampled independently and uniformly at random from $\{\mathbf{x} \in \mathcal{S}_w^n : |\mathbf{x} \wedge \mathbf{c}| = \alpha\}$, find a $(1 - o(1))$-fraction of all pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{B}_\alpha(\mathbf{c})^2$ satisfying $|\mathbf{x} + \mathbf{y}| = w$ (called *solution pairs*). We say that this problem has *parameters* (B, n, w, v, α).

Note that this problem is a variant of the near-neighbor search problem (Problem 3.3) defined in Section 3. An important difference here is that the input list $\mathcal{B}_\alpha(\mathbf{c})$ is not uniformly random on $\mathcal{S}_w^n$ but on the set $\{\mathbf{x} \in \mathcal{S}_w^n : |\mathbf{x} \wedge \mathbf{c}| = \alpha\}$. Consequently, the probability (denoted p below) that a uniformly random pair from the input list forms a solution pair does not only depend on n, w, but also on the parameters v, α, as further shown in the next section.

Remark 4.2 (Expected number of solutions). Note that the expected number of solutions to Problem 4.1 with parameters (B, n, w, v, α) is $\binom{B}{2}p$, where p denotes the probability that a uniformly random pair $\mathbf{x}, \mathbf{y}$ in $\mathcal{B}_\alpha(\mathbf{c})$ forms a solution pair.

In the remainder of Section 4, we fix parameters (B, n, w, v, α). Before presenting our quantum algorithms for solving Problem 4.1, we calculate the probability p of forming a solution pair, as it is used in the analysis of our algorithms.

4.1 Probability p of Forming a Solution Pair

As before, let p denote the probability that a uniformly random pair $\mathbf{x}, \mathbf{y} \in \mathcal{B}_\alpha(\mathbf{c})$ satisfies $|\mathbf{x}+\mathbf{y}| = w$. That is,

$$p := \Pr_{\mathbf{x},\mathbf{y} \in \mathcal{B}_\alpha(\mathbf{c})}[|\mathbf{x} + \mathbf{y}| = w] = \Pr_{\mathbf{x},\mathbf{y} \in \mathcal{S}_w^n}[|\mathbf{x} + \mathbf{y}| = w \mid |\mathbf{x} \wedge \mathbf{c}| = \alpha \text{ and } |\mathbf{y} \wedge \mathbf{c}| = \alpha]. \tag{2}$$

We can express p in terms of the parameters (n, w, v, α) as follows.

Lemma 4.3 (Probability p). *The probability p as defined in Equation (2) is equal to*

$$p(n, w, v, \alpha) = \frac{\binom{w}{w/2}\binom{n-w}{w/2}}{\binom{v}{\alpha}\binom{n-v}{w-\alpha}\binom{w}{\alpha}\binom{n-w}{v-\alpha}} \cdot |\mathcal{W}_{v,\alpha}^n(\mathbf{x}, \mathbf{y})|$$

where $|\mathcal{W}^n_{v,\alpha}(\mathbf{x},\mathbf{y})| = \sum_{e=\max(0,2\alpha-v)}^{\min(\alpha,w/2)} \binom{w/2}{e}\binom{w/2}{\alpha-e}^2\binom{n-3w/2}{v-2\alpha+e}$ *is the surface area of a wedge for arbitrary* $\mathbf{x},\mathbf{y} \in \mathcal{S}^n_w$ *with* $|\mathbf{x}+\mathbf{y}| = w$.[8]

Before proceeding with the proof of Lemma 4.3, we state the following lemma.[9] Also, it might help to keep Figure 1 in mind.

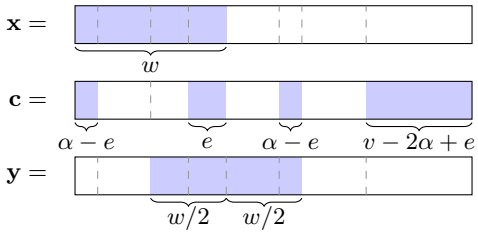

Fig. 1. An example of the overlaps between $\mathbf{x},\mathbf{y} \in \mathcal{S}^n_w$ and $\mathbf{c} \in \mathcal{S}^n_v$, where $|\mathbf{x}+\mathbf{y}| = w$, $|\mathbf{x}\wedge\mathbf{c}| = |\mathbf{y}\wedge\mathbf{c}| = \alpha$, and $|\mathbf{x}\wedge\mathbf{y}\wedge\mathbf{c}| = e$. All cases of such vectors under these conditions are permutations of this example.

Lemma 4.4. *For all* $\mathbf{c} \in \mathcal{S}^n_v$ *and all* $\mathbf{x}',\mathbf{y}' \in S^n_w$ *satisfying* $|\mathbf{x}'+\mathbf{y}'| = w$*, we have*

$$\Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{c}| = \alpha \text{ and } |\mathbf{y}\wedge\mathbf{c}| = \alpha \mid |\mathbf{x}\wedge\mathbf{y}| = w/2] = \frac{|\mathcal{W}^n_{v,\alpha}(\mathbf{x}',\mathbf{y}')|}{|\mathcal{S}^n_v|}.$$

Proof. For fixed $\mathbf{c} \in \mathcal{S}^n_v$ and $\mathbf{x}',\mathbf{y}' \in S^n_w$ satisfying $|\mathbf{x}'+\mathbf{y}'| = w$, the stated probability is equal to

$$\frac{|\{(\mathbf{x},\mathbf{y}) \in (\mathcal{S}^n_w)^2 \colon |\mathbf{x}\wedge\mathbf{c}| = \alpha, |\mathbf{y}\wedge\mathbf{c}| = \alpha, |\mathbf{x}\wedge\mathbf{y}| = w/2\}|}{|\{(\mathbf{x},\mathbf{y}) \in (\mathcal{S}^n_w)^2 \colon |\mathbf{x}\wedge\mathbf{y}| = w/2\}|} = \frac{\sum_e \binom{v}{\alpha}\binom{n-v}{w-\alpha}\binom{\alpha}{e}\binom{v-\alpha}{\alpha-e}\binom{w-\alpha}{w/2-e}\binom{n-(w+v-\alpha)}{w/2-\alpha+e}}{\binom{n}{w}\binom{w}{w/2}\binom{n-w}{w/2}}$$

where e ranges over all possible values of $|\mathbf{x}\wedge\mathbf{y}\wedge\mathbf{c}|$. Now, using that $\frac{\binom{v}{\alpha}\binom{n-v}{w-\alpha}}{\binom{n}{w}} = \frac{\binom{w}{\alpha}\binom{n-w}{v-\alpha}}{\binom{n}{v}}$, $\binom{w}{\alpha}\binom{\alpha}{e}\binom{w-\alpha}{w/2-e} = \binom{w}{w/2}\binom{w/2}{e}\binom{w-w/2}{\alpha-e}$, and $\binom{n-w}{v-\alpha}\binom{v-\alpha}{\alpha-e}\binom{(n-w)-(v-\alpha)}{w/2-(\alpha-e)} = \binom{n-w}{w/2}\binom{w/2}{\alpha-e}\binom{(n-w)-w/2}{(v-\alpha)-(\alpha-e)}$, it can be seen that this indeed equals $\frac{|\mathcal{W}^n_{v,\alpha}(\mathbf{x}',\mathbf{y}')|}{\binom{n}{v}}$. □

Proof (Proof of Lemma 4.3). By definition,

$$\begin{aligned} p &= \Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{y}| = w/2 \mid |\mathbf{x}\wedge\mathbf{c}| = \alpha \text{ and } |\mathbf{y}\wedge\mathbf{c}| = \alpha] \\ &= \Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{c}| = \alpha \text{ and } |\mathbf{y}\wedge\mathbf{c}| = \alpha \mid |\mathbf{x}\wedge\mathbf{y}| = w/2] \cdot \frac{\Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{y}| = w/2]}{\Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{c}| = |\mathbf{y}\wedge\mathbf{c}| = \alpha]} \\ &= \frac{|\mathcal{W}^n_{v,\alpha}(\mathbf{x}',\mathbf{y}')|}{\binom{n}{v}} \cdot \frac{\Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{y}| = w/2]}{\Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{c}| = |\mathbf{y}\wedge\mathbf{c}| = \alpha]} \end{aligned}$$

for arbitrary $\mathbf{x}',\mathbf{y}' \in S^n_w$ satisfying $|\mathbf{x}'+\mathbf{y}'| = w$. Note that the last equality follows from Lemma 4.4. Since $\Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{y}| = w/2] = \frac{\binom{n}{w}\binom{w}{w/2}\binom{n-w}{w/2}}{\binom{n}{w}^2}$ and $\Pr_{\mathbf{x},\mathbf{y}\in_R\mathcal{S}^n_w}[|\mathbf{x}\wedge\mathbf{c}| = \alpha \text{ and } |\mathbf{y}\wedge\mathbf{c}| = \alpha] = \frac{\binom{v}{\alpha}^2\binom{n-v}{w-\alpha}^2}{\binom{n}{w}^2}$, it follows that

$$p = \frac{|\mathcal{W}^n_{v,\alpha}(\mathbf{x}',\mathbf{y}')|}{\binom{n}{v}} \frac{\binom{n}{w}\binom{w}{w/2}\binom{n-w}{w/2}}{\binom{v}{\alpha}^2\binom{n-v}{w-\alpha}^2}.$$

Notice that $\binom{n}{v}\binom{v}{\alpha}\binom{n-v}{w-\alpha} = \binom{n}{w}\binom{w}{\alpha}\binom{n-w}{v-\alpha}$, hence the desired result follows. □

[8] Recall (see Remark 2.6) that $\mathcal{W}^n_{v,\alpha}(\mathbf{x},\mathbf{y})$ does not depend on the specific choice of $\mathbf{x}$ and $\mathbf{y}$, but only on $|\mathbf{x}|, |\mathbf{y}|, |\mathbf{x}+\mathbf{y}|$. Consequently, the probability p is independent of the particular selection of $\mathbf{x}$ and $\mathbf{y}$.

[9] In the following proofs, we will use the fact that for all $a,b,c,d \in \mathbb{N}$ (for which the binomial coefficients are well-defined), $\binom{a}{b}\binom{b}{d}\binom{a-b}{c-d} = \binom{a}{c}\binom{c}{d}\binom{a-c}{b-d}$.

4.2 Quantum Algorithm for `FindSolutions` Using Grover's Algorithm

As a warm-up, we give a straightforward quantum algorithm for solving Problem 4.1 (and thus `FindSolutions`) using Grover's algorithm [Gro96]. This will be used as a baseline to compare our more advanced algorithms with.

Theorem 4.5 (`FindSolutions` using Grover). *Let (B, n, w, v, α) be well-defined parameters for Problem 4.1 and let $p = p(n, w, v, \alpha)$ be according to Lemma 4.3. Then there exists a quantum algorithm that solves Problem 4.1 with parameters (B, n, w, v, α) in expected time $\tilde{O}(B^2\sqrt{p})$ using classical memory and QRACM of expected size $M_C = M_{QRACM} = B$ and quantum memory of expected size $M_Q = n^{O(1)}$.*

Proof. The statement follows from repeatedly applying Grover's algorithm (Theorem 2.8) with the set of all $\binom{B}{2}$ pairs as the search space. The expected number of solutions is $t := \binom{B}{2}p$. Using a coupon-collector argument, it follows that all solutions can be found in expected time $\tilde{O}(\sqrt{t\binom{B}{2}}) = \tilde{O}(B^2\sqrt{p})$. The algorithm classically stores the B list vectors. Grover's algorithm requires quantum access to them (QRACM) and a number of qubits that is polynomial in n. □

This Grover-based quantum algorithm can be used as the subroutine `FindSolutions` in Theorem 3.8 to obtain a quantum algorithm for NNS, and thus for code sieving.

4.3 Quantum Algorithm for `FindSolutions` Using Quantum Walks

We will now replace the role of Grover's algorithm with quantum-walk techniques. In particular, we describe a `FindSolutions` subroutine that searches for solution pairs $(\mathbf{x}, \mathbf{y}$ such that $|\mathbf{x} + \mathbf{y}| = w)$ inside a given list $\mathcal{B}_\alpha(\mathbf{c})$ using a quantum walk. To speed up the search for solution pairs, we add a layer of locality-sensitive filtering (LSF) as part of the data of each vertex. This idea was inspired by [CL21], and we show that we can indeed transfer most of their ideas from the Euclidean metric to the Hamming metric. We analyze the complexity of the resulting quantum walk, and consider a version of the walk where we apply the 'sparsification' technique from [CL21], resulting in a different balance of the complexity parameters.

MNRS-style quantum walk. We use the quantum-walk framework from [MNRS11] (see Section 2.4). Our quantum walk is defined on the Johnson graph $J(B, s)$, for some parameter $s \leq B$ that must be carefully chosen later. The vertices of the graph are identified with the size-s subsets of the input list $\mathcal{B}_\alpha(\mathbf{c})$, and two vertices are adjacent if and only if the corresponding subsets differ in exactly one element. The goal of the quantum walk is to return a *marked vertex*, which is a vertex containing a solution pair. As aforementioned, we add some additional *data* to each vertex to speed up the search for marked vertices.

We briefly sketch the inner steps of the walk (for more details see Section 4.6). The set-up step (of cost S) constructs a uniform superposition over all vertices and their corresponding data. In the update step (of cost U), the current vertex (i.e., a size-s subset of vectors) is mapped to a uniform superposition of its neighbors. During the update step, the algorithm also keeps track of whether it has encountered a solution pair, which will be facilitated by the data associated to the vertices. As a result, the checking step is immediate (and thus has cost $\mathsf{C} = \tilde{O}(1)$). Writing ϵ for the fraction of marked vertices and δ for the spectral gap of the graph, the quantum walk returns a marked vertex in time $\mathsf{S} + \frac{1}{\sqrt{\epsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})$. Each run of our quantum walk finds one solution to Problem 4.1, so we repeat it until a $(1 - o(1))$-fraction of all solution pairs is found.[10]

Adding a layer of LSF. We will now describe the data added to each vertex of the Johnson graph $J(B, s)$ to facilitate the detection of marked vertices during the update step. The first part of the update step can essentially be viewed as a map between two adjacent vertices (which in $J(B, s)$ differ by exactly one element): given a subset $S \subseteq \mathcal{B}_\alpha(\mathbf{c})$, a neighbor S' of S is obtained by removing a vector $\mathbf{x}_{old}$ from S and adding a vector $\mathbf{x}_{new}$ from $\mathcal{B}_\alpha(\mathbf{c}) \setminus S$. In the second part of the update step, the algorithm checks whether the newly added vector $\mathbf{x}_{new}$ forms a solution pair with one of

[10] The reader might notice we do not apply the reusable quantum-walk techniques from [BCSS23]. The reason is that our numerical experiments do not show a significant speed-up, see Remark 5.2.

the non-removed vertices (i.e., those in $S \setminus \{\mathbf{x}_{old}\}$). This check can be performed by simply applying Grover, but we achieve better complexities if we add a layer of locality-sensitive filtering.

More precisely, we identify the input list $\mathcal{B}_\alpha(\mathbf{c}) \subseteq S_w^n$ with the list $\mathcal{L}' = \{\pi_{\mathbf{c}}(\mathbf{x})\colon \mathbf{x} \in \mathcal{B}_\alpha(\mathbf{c})\} \subseteq \mathcal{S}_\alpha^v$, where $\pi_{\mathbf{c}}(\mathbf{x}) := \mathbf{x} \wedge \mathbf{c}$ is the v-dimensional vector obtained by projecting $\mathbf{x}$ onto the support of $\mathbf{c}$.[11] As further detailed in Section 4.4, this allows us to apply the LSF techniques from Section 3.2 and Section 3.3. In particular, at the start of our algorithm, we sample an RPC $\mathcal{C}_f' \subseteq S_{v'}^v$ for some parameter v'. Each vector $\mathbf{c}' \in \mathcal{C}_f'$ forms the center of a bucket $\mathcal{B}_\beta(\mathbf{c}')$ for some bucketing parameter β. However, these buckets will only be defined per vertex: for a vertex corresponding to subset S, we fill $\mathcal{B}_\beta(\mathbf{c}')$ with the vectors in S that are in a certain sense 'close' to $\mathbf{c}'$ with respect to the parameter β. Specifically, we let

$$\mathcal{B}_\beta(\mathbf{c}') := \{\mathbf{x} \in S\colon |\pi_{\mathbf{c}}(\mathbf{x}) \wedge \mathbf{c}'| = \beta\}$$

The filled buckets $(\mathcal{B}_\beta(\mathbf{c}'))_{\mathbf{c}' \in \mathcal{C}_f'}$ are then added as part of the data of the vertex corresponding to S.

Consequently, instead of checking for solution pairs within a vertex S, the algorithm will only check for solution pairs within the $|\mathcal{C}_f'|$ β-buckets. This may induce some false negatives, but this can be controlled by choosing β carefully. Just like the parameter α was used to quantify the probability that two vectors in an α-bucket $\mathcal{B}_\alpha(\mathbf{c})$ form a solution to Problem 3.3 (NNS), the parameter β quantifies the probability that two vectors in a β-bucket $\mathcal{B}_\beta(\mathbf{c}')$ form a solution to Problem 4.1 (bucket search). By choosing the parameters (including β) carefully, it allows us to obtain speed-ups, which is further demonstrated by our numerical results in Section 5.

To state our main result on our quantum-walk algorithm, we need the following notions. For fixed $t' \in \Theta(\sqrt{v})$ and C, we define $q = q_C = \Pr[\exists \mathbf{c}' \in \mathcal{C}_f' \text{ s.t. } |\pi_{\mathbf{c}}(\mathbf{x}) \wedge \mathbf{c}'| = \beta \text{ and } |\pi_{\mathbf{c}}(\mathbf{y}) \wedge \mathbf{c}'| = \beta]$, where the probability is taken over $\mathcal{C}_f' \sim R_{v,v',t',C}$ and uniformly random $\mathbf{x}, \mathbf{y} \in \text{Region}_\alpha(\mathbf{c})$ satisfying $|\mathbf{x}+\mathbf{y}| = w$. Informally, for an arbitrary solution pair $(\mathbf{x}, \mathbf{y})$, q denotes the probability that they share a valid β-center. Furthermore, for any $\mathcal{C}_f' \sim R_{v,v',t',C}$, we define

$$\mathcal{VC}_\beta(\mathbf{x}) := \{\mathbf{c}' \in \mathcal{C}_f'\colon |\pi_{\mathbf{c}}(\mathbf{x}) \wedge \mathbf{c}'| = \beta\}$$

to be the set of valid β-bucket centers $\mathbf{c}'$ for any given $\mathbf{x} \in \text{Region}_\alpha(\mathbf{c})$. We write $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|]$ for the expected size of $\mathcal{VC}_\beta(\mathbf{x})$, where the expectation is taken over $\mathcal{C}_f' \sim R_{v,v',t',C}$. Furthermore, we write $\mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]$ for the expected size of a bucket $\mathcal{B}_\beta(\mathbf{c}')$, where the expectation is taken over $\mathbf{x}_1, \ldots, \mathbf{x}_s \in \text{Region}_\alpha(\mathbf{c})$ sampled independently and uniformly at random. Note that $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|]$ and $\mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]$ are the same for all $\mathbf{x} \in \text{Region}_\alpha(\mathbf{c})$ and $\mathbf{c}' \in \mathcal{S}_{v'}^v$, respectively.

The proof of Theorem 4.7 will be given in Section 4.6.

Remark 4.6. With a slight abuse of notation, we write $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|])$ as shorthand for $\max(1, \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|])$ to simplify the expressions of the time and memory complexity. We do the same for $\mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]$.

Theorem 4.7 (`FindSolutions` using quantum walks). *Let (B, n, w, v, α) be well-defined parameters for Problem 4.1 and let $p = p(n, w, v, \alpha)$ be according to Lemma 4.3. For non-negative integers $s = \tilde{O}(\frac{1}{\sqrt{p}})$, $\beta \leq \alpha$, $v' \leq v$, $t' = \Theta(\sqrt{v})$, and C, define $q = q_C$, $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|]$, and $\mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]$ as above.*

Then there exists a quantum-walk algorithm that solves Problem 4.1 with parameters (B, n, w, v, α) in expected time $\tilde{O}\left(B^2 p \cdot (\mathsf{S} + \frac{1}{\sqrt{\epsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}))\right)$, where, omitting factors subexponential in n,

$$\begin{aligned}
\mathsf{S} &= s \cdot \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|] \\
\mathsf{U} &= \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|] + \sqrt{\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|] \cdot \mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]} \\
\mathsf{C} &= 1 \\
\delta &= \frac{1}{s} \\
\epsilon &= s^2 pq.
\end{aligned}$$

This quantum-walk algorithm uses classical memory and QRACM of expected size $\tilde{O}(B)$, and quantum memory and QRAQM of expected size $\tilde{O}(s \cdot \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|])$.

[11] Note that $\mathbf{x} \wedge \mathbf{c}$ is actually an n-dimensional vector. With abuse of notation, we sometimes view it as a v-dimensional vector. The dimension should be clear from the context.

Note that the expected size of $\mathcal{VC}_\beta(\mathbf{x})$ plays an important role in the complexity of our quantum-walk algorithm. In Section 4.5, we describe how this expected size (and thus the complexity) crucially depends on the choice of $|\mathcal{C}'_f|$, and we analyze two different choices. Besides the choice of $|\mathcal{C}'_f|$, the precise complexity of Theorem 4.7 also depends on the optimal choice of the parameters s, v', and β.

Similar to the Grover-based algorithm, we can use the above quantum-walk algorithm as the subroutine `FindSolutions` in Theorem 3.8 to obtain a quantum algorithm for NNS and code sieving.

4.4 Adding a Layer of Filtering

In this section, we present the technical details to justify adding a layer of LSF to the data of our quantum walk. Recall that our input list $\mathcal{B}_\alpha(\mathbf{c})$ is a subset of $\{\mathbf{x} \in \mathcal{S}_w^n : |\mathbf{x} \wedge \mathbf{c}| = \alpha\}$. In order to apply the LSF techniques from [DEEK24] (described in Section 3.3), we will relate the problem of finding solution pairs in $\mathcal{B}_\alpha(\mathbf{c})$ to a problem where we search for solution pairs in a list $\mathcal{L}'$ of vectors sampled independently and uniformly over a suitable (smaller-dimensional) Hamming sphere.

This is motivated by the following observations. Note that any $\mathbf{x} \in \mathcal{B}_\alpha(\mathbf{c})$ can be written as $\mathbf{x} = \mathbf{x} \wedge \mathbf{c} + \mathbf{x} \wedge \overline{\mathbf{c}}$ for the center $\mathbf{c}$. Suppose, for a moment, that $\mathbf{c}$ is of weight α (i.e., $v = \alpha$). Then, for any $\mathbf{x}, \mathbf{y} \in \mathcal{B}_\alpha(\mathbf{c})$ we have that $\mathbf{x} \wedge \mathbf{c} = \mathbf{c}$ and $\mathbf{y} \wedge \mathbf{c} = \mathbf{c}$, and thus $\mathbf{x} + \mathbf{y} = \mathbf{x} \wedge \overline{\mathbf{c}} + \mathbf{y} \wedge \overline{\mathbf{c}}$. In other words, for $\mathbf{x}, \mathbf{y} \in \mathcal{B}_\alpha(\mathbf{c})$, we have that $|\mathbf{x}+\mathbf{y}| = w$ if and only if $|\mathbf{x} \wedge \overline{\mathbf{c}} + \mathbf{y} \wedge \overline{\mathbf{c}}| = w$. Therefore, we could restrict to searching for solutions among the vectors $\mathbf{x} \wedge \overline{\mathbf{c}}$ instead. However, if $v > \alpha$, then this equivalence breaks down: whether $\mathbf{x} + \mathbf{y}$ is of weight w does no longer only depend on $|\mathbf{x} \wedge \overline{\mathbf{c}} + \mathbf{y} \wedge \overline{\mathbf{c}}|$, but also on $|\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}|$ (that is, the latter need no longer be 0). Specifically, for a fixed pair $\mathbf{x}, \mathbf{y} \in \mathcal{B}_\alpha(\mathbf{c})$, we have that $|\mathbf{x} + \mathbf{y}| = w$ if and only if $|\mathbf{x} \wedge \overline{\mathbf{c}} + \mathbf{y} \wedge \overline{\mathbf{c}}| = w - 2\alpha + 2e$, where $e := |\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}|$. Although the value of e may differ for different pairs $(\mathbf{x}, \mathbf{y})$, Theorem 6.4 shows that for most solution pairs $(\mathbf{x}, \mathbf{y})$ it is the same. The proof will be given at the end of this section.

Theorem 4.8. *Let $\mathcal{B}_\alpha(\mathbf{c})$ be an instance of Problem 4.1 with parameters (B, n, w, v, α). There exists an integer e^* such that the expected number of pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{B}_\alpha(\mathbf{c})^2$ satisfying $|\mathbf{x} + \mathbf{y}| = w$ is asymptotically equal to the expected number of pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{B}_\alpha(\mathbf{c})^2$ satisfying $|\mathbf{x} + \mathbf{y}| = w$ and $|\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}| = e^*$.*

In other words, Problem 4.1 is asymptotically equivalent to the following problem:

Find (almost) all $\mathbf{x}, \mathbf{y}$ in $\mathcal{B}_\alpha(\mathbf{c})$ satisfying $|\mathbf{x} + \mathbf{y}| = w$ and $|\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{c}| = e^*$.

We will therefore restrict to solving this latter problem instead.

Let $\mathcal{L}' := \{\pi_{\mathbf{c}}(\mathbf{x}) \colon \mathbf{x} \in \mathcal{B}_\alpha(\mathbf{c})\}$, where we view each $\pi_{\mathbf{c}}(\mathbf{x}) := \mathbf{x} \wedge \mathbf{c}$ as a vector in S_α^v (i.e., we ignore all coefficients $(\mathbf{x} \wedge \mathbf{c})_i$ where $\mathbf{c}_i = 0$). Note that the vectors in $\mathcal{L}'$ are independently and uniformly distributed over S_α^v. In particular, the problem of finding all $\mathbf{x}', \mathbf{y}' \in \mathcal{L}'$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$ is a variant of an NNS problem (where the target weight e^* does not necessarily equal the weight α of the vectors in the input list $\mathcal{L}'$), so we can apply the LSF techniques from [DEEK24]. Furthermore, the set of those pairs $(\mathbf{x}', \mathbf{y}')$ (or actually the corresponding $(\mathbf{x}, \mathbf{y})$) forms a superset of the set $\{(\mathbf{x}, \mathbf{y}) \in \mathcal{B}_\alpha(\mathbf{c})^2 \colon |\mathbf{x} + \mathbf{y}| = w\}$. Therefore, to asymptotically solve Problem 4.1, it suffices to find all $\mathbf{x}', \mathbf{y}' \in \mathcal{L}'$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$, and keep those for which it also holds that $|\mathbf{x} + \mathbf{y}| = w$.

In order to asymptotically find all solution pairs in the bucket $\mathcal{B}_\alpha(\mathbf{c})$, we will thus focus on finding all pairs $(\mathbf{x}', \mathbf{y}') \in \mathcal{L}' \times \mathcal{L}'$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$ and $|\mathbf{x} + \mathbf{y}| = w$. We will call such a pair $(\mathbf{x}', \mathbf{y}')$ an *$\mathcal{L}'$-solution*.

Remark 4.9 ('Residual vectors for codes'). For the reader familiar with the techniques from [CL21]: we consider the vectors in $\mathcal{L}'$ as a code analog of the 'residual vectors' in [CL21]. The lattice equivalent of `FindSolutions` asks to find all pairs $(\mathbf{v}, \mathbf{w})$ in a given bucket that satisfy $\langle \mathbf{x}, \mathbf{y} \rangle = \theta$ for some given θ, where the bucket is defined for some center $\mathbf{c} \in \mathbb{R}^n$ (of unit norm) and bucketing parameter $\alpha \in (0, 1)$, and consists of unit (lattice) vectors $\mathbf{v} \in \mathbb{R}^n$ satisfying $\langle \mathbf{v}, \mathbf{c} \rangle \geq \alpha$. (Here, $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product.) Note that if $\mathbf{v}$ is in the bucket of $\mathbf{c}$, then it can be written as $\mathbf{v} = \alpha\mathbf{c} + \sqrt{1 - \alpha^2}\mathbf{v}'$ for some unit vector $\mathbf{v}'$ that is orthogonal to $\mathbf{c}$. In particular, the problem of finding all $(\mathbf{v}, \mathbf{w})$ satisfying $\langle \mathbf{v}, \mathbf{w} \rangle = \theta$ is equivalent to the problem of finding all $(\mathbf{v}', \mathbf{w}')$ satisfying $\langle \mathbf{v}', \mathbf{w}' \rangle = \theta'$ for some θ' depending on θ. These (left-over) vectors $\mathbf{v}'$ are called *residual vectors* in [CL21], and are the analogs of the vectors $\mathbf{x}'$ that we defined for codes. However, note that in the setting of codes, we don't have an exact equivalence between the condition $|\mathbf{x} + \mathbf{y}| = w$ and a condition $|\mathbf{x}' + \mathbf{y}'| = w'$ for some w' depending on w' (unless $v = \alpha$). Nevertheless, Theorem 4.8 shows that we can still turn it into an 'asymptotic' equivalence.

It remains to prove Theorem 4.8.

Proof (Proof of Theorem 4.8). Recall that p denotes the probability that two independent and uniformly random $\mathbf{x}, \mathbf{y} \in \mathcal{B}_\alpha(\mathbf{c})$ satisfy $|\mathbf{x}+\mathbf{y}| = w$. From the definition of p, we observe that

$$p = \Pr_{\mathbf{x},\mathbf{y}\in_R \mathcal{B}_\alpha(\mathbf{c})}[|\mathbf{x}+\mathbf{y}| = w] = \sum_{e=\max(0,2\alpha-v)}^{\min(\alpha,w/2)} p(e)$$

where $p(e) := \Pr_{\mathbf{x},\mathbf{y}\in_R \mathcal{B}_\alpha(\mathbf{c})}[|\mathbf{x}+\mathbf{y}| = w \text{ and } |\mathbf{x}\wedge\mathbf{y}\wedge\mathbf{c}| = e]$. In particular, $p = \tilde{\Theta}(p(e^*))$, where $e^* := \operatorname{argmax}_e p(e)$.[12] (Note that an explicit formula for $p(e)$ is given in Lemma 4.3.) This implies that

$$\mathbb{E}[|\{(\mathbf{x},\mathbf{y}) \in \mathcal{B}_\alpha(\mathbf{c})^2 \colon |\mathbf{x}+\mathbf{y}| = w\}|] \quad \text{and} \quad \mathbb{E}[|\{(\mathbf{x},\mathbf{y}) \in \mathcal{B}_\alpha(\mathbf{c})^2 \colon |\mathbf{x}+\mathbf{y}| = w \text{ and } |\mathbf{x}\wedge\mathbf{y}\wedge\mathbf{c}| = e^*\}|]$$

are asymptotically equal, as we wanted to show. □

4.5 On the Choice of $|\mathcal{C}'_f|$

We will now explain how the choice of the size of the RPC $\mathcal{C}'_f \subseteq S^v_{v'}$ (together with the choice of β) affects the time complexity of our quantum-walk algorithm (Theorem 4.7) and present two choices of $|\mathcal{C}'_f|$ that we focus on. We start by showing how the two components $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|]$ and q in the time complexity depend on $|\mathcal{C}'_f|$. Since we identify the input list $\mathcal{B}_\alpha(\mathbf{c}) \subseteq S^n_w$ with the list $\mathcal{L}' \subseteq S^v_\alpha$, we will from now on write $\mathcal{VC}_\beta(\mathbf{x}')$ for $\mathbf{x}' \in \mathcal{L}'$, instead of $\mathcal{VC}_\beta(\mathbf{x})$ for $\mathbf{x} \in \mathcal{B}_\alpha(\mathbf{c})$.

For all $\mathbf{x}' \in S^v_\alpha$, the probability that a uniformly random $\mathbf{c}' \in S^v_{v'}$ satisfies $|\mathbf{x}'\wedge\mathbf{c}'| = \beta$ is

$$p_\beta := \Pr_{\mathbf{c}'\in S^v_{v'}}[|\mathbf{x}'\wedge\mathbf{c}'| = \beta] = \frac{|\mathcal{C}^v_{v',\beta}(\mathbf{x}')|}{|S^v_{v'}|}.$$

Note that this also equals the probability that (for fixed $\mathbf{c}' \in S^v_{v'}$) a uniformly random $\mathbf{x}' \in S^v_\alpha$ satisfies $|\mathbf{x}'\wedge\mathbf{c}'| = \beta$. Therefore, $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|] = |\mathcal{C}'_f| \cdot p_\beta$ (up to factors subexponential in n, see [DEEK24, Lemma 4.6]) and $\mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|] = s \cdot p_\beta$, where we recall that s denotes the vertex size of the Johnson graph.[13]

Next, we will show how the probability q depends on the size of $\mathcal{C}'_f$, where we recall that q denotes the probability that a given solution pair can be found in a same β-bucket. First, note that we can express q as follows.

Lemma 4.10. *We have* $q = \Pr[\exists \mathbf{c}' \in \mathcal{C}'_f \text{ s.t. } \mathbf{x}', \mathbf{y}' \in \mathcal{B}_\beta(\mathbf{c}')]$ *for all* $\mathbf{x}', \mathbf{y}' \in \mathcal{L}'$ *satisfying* $|\mathbf{x}'\wedge\mathbf{y}'| = e^*$.

Proof. Notice that the following two probabilities are the same:

- For fixed $\mathbf{x}', \mathbf{y}' \in S$ satisfying $|\mathbf{x}'\wedge\mathbf{y}'| = e^*$, the probability that a uniformly random $\mathbf{c}' \in S^v_{v'}$ satisfies $|\mathbf{x}'\wedge\mathbf{c}'| = \beta$ and $|\mathbf{y}'\wedge\mathbf{c}'| = \beta$.
- For fixed $\mathbf{x}', \mathbf{y}' \in S$ satisfying $|\mathbf{x}'\wedge\mathbf{y}'| = e^*$ and $|\mathbf{x}+\mathbf{y}| = w$, the probability that a uniformly random $\mathbf{c}' \in S^v_{v'}$ satisfies $|\mathbf{x}'\wedge\mathbf{c}'| = \beta$ and $|\mathbf{y}'\wedge\mathbf{c}'| = \beta$.

Indeed, by a counting argument, one can show that both probabilities are equal to $\frac{|\mathcal{W}^v_{v',\beta}(\mathbf{x}',\mathbf{y}')|}{|S^v_{v'}|}$. □

Fix an arbitrary pair $(\mathbf{x}', \mathbf{y}') \in \mathcal{L}'^2$ satisfying $|\mathbf{x}'\wedge\mathbf{y}'| = e^*$, and define

$$W := \Pr_{\mathbf{c}'\in_R \mathcal{C}'_f}[|\mathbf{x}'\wedge\mathbf{c}'| = \beta \text{ and } |\mathbf{y}'\wedge\mathbf{c}'| = \beta] = \frac{|\mathcal{W}^v_{v',\beta}(\mathbf{x}',\mathbf{y}')|}{|S^v_{v'}|}.$$

Then we can write q as $q = 1-(1-W)^{|\mathcal{C}'_f|}$, so $q = \Theta(|\mathcal{C}'_f|W)$ if $|\mathcal{C}'_f|W \le 1$, and $q = \Theta(1)$ otherwise. (Here, we are implicitly viewing $\mathcal{C}'_f$ as a random code; a formal justification for RPCs follows from [DEEK24, Lemma 4.8].)

We will now consider two variants of the previous quantum-walk algorithm, where we vary the choice of $|\mathcal{C}'_f|$, i.e., the number of β-buckets, and see how that affects the two factors of the time complexity of our quantum-walk algorithm:

$$\mathsf{S} = s\cdot\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|] \quad \text{and} \quad \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}U + C\right) = \frac{1}{\sqrt{spq}}\left(\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|] + \sqrt{\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|]\cdot\mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]}\right).$$

(Recall that the expected values should be replaced by 1 if they are smaller than 1, see Remark 4.6.)

[12] The value of e^* can be computed numerically. For instance, see [Car20, DEEK24].

[13] In the remainder of Section 4, we often omit writing $\tilde{O}(\cdot)$ or factors of the form $2^{o(n)}$ for ease of reading.

Variant 1: Choosing $|\mathcal{C}'_f|$ such that q is maximized. A first approach is to choose $|\mathcal{C}'_f|$ such that each $\mathcal{L}'$-solution $\mathbf{x}', \mathbf{y}'$ have at least one bucket in common, i.e., there are no false-negatives. In particular, for similar reasons as in the proof of [DEEK24, Theorem 4.4] (but then considered for our *second* layer of LSF) taking $|\mathcal{C}'_f| = \Omega(\frac{1}{W})$ ensures $q = \Theta(1)$ and $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|] = \frac{|\mathcal{C}^v_{v',\beta}(\mathbf{x}')|}{|\mathcal{W}^v_{v',\beta}(\mathbf{x}',\mathbf{y}')|} = \frac{p_\beta}{W}$ (which is ≥ 1).

Then the two contributors to the quantum-walk cost are as follows:

$$\mathsf{S} = s \cdot \frac{p_\beta}{W} \quad \text{and} \quad \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) = \sqrt{\frac{p_\beta}{spW}} \cdot \left(\sqrt{\frac{p_\beta}{W}} + \max(1, \sqrt{sp_\beta})\right). \tag{3}$$

Variant 2: Sparsification. A way to obtain a better time complexity is to apply a technique that we refer to as *sparsification*, which was already used in [Laa15, CL21]. Reducing the size $|\mathcal{C}'_f|$ (taking a *sparser* code $\mathcal{C}'_f$), reduces both q and $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|]$, resulting in an increase in the cost $1/\sqrt{\epsilon}$, but a decrease in both the set-up S and update U costs. Altogether, this results in a different balance between the terms S and $\frac{1}{\sqrt{\epsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})$ in the runtime of the quantum walk subroutine.

Following the same reasoning as in [CL21], we take $|\mathcal{C}'_f| = \frac{|S^v_{v'}|}{|\mathcal{C}^v_{v',\beta}(\mathbf{x}')|} = \frac{1}{p_\beta}$. Note that now $q = \Omega(\frac{W}{p_\beta})$ and $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|] = 1$ (recall that we omit writing $2^{o(n)}$), indeed resulting in a different balance between the contributors of the quantum-walk cost:

$$\mathsf{S} = s \quad \text{and} \quad \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) = \sqrt{\frac{p_\beta}{spW}}(1 + \max(1, \sqrt{sp_\beta})). \tag{4}$$

More precisely, comparing Equation (4) to Equation (3), shows that sparsification has reduced the set-up cost S by a factor of $\frac{p_\beta}{W}$ and part of the cost $\frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)$ by a factor of $\sqrt{\frac{p_\beta}{W}}$.

We will present further in Section 5 our numerical results of these formulas (optimized to minimize the time complexity), which illustrate that the second choice – i.e., sparsification – indeed provides a speed-up over the first.

4.6 Proof of the Complexity of `FindSolutions` Using Quantum Walks

We will now prove Theorem 4.7. Recall (see Remark 4.6) that we write $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|])$ and $\mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]$ as shorthand for $\max(1, \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x})|])$ and $\max(1, \mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|])$, respectively.

Time complexity analysis

Proof (Proof of Theorem 4.7: Part 1/2, time complexity). Let $\mathcal{B}_\alpha(\mathbf{c})$ be an instance of Problem 4.1 with parameters (B, n, w, v, α), and let $p = p(n, w, v, \alpha)$ be according to Lemma 4.3.

We start by describing the details of the quantum walk according to the MNRS framework [MNRS11]. We identify $\mathcal{B}_\alpha(\mathbf{c})$ with the list $\mathcal{L}' \subseteq S^v_\alpha$ defined in Section 4.4, and recall that each vector $\mathbf{x} \in \mathcal{B}_\alpha(\mathbf{c})$ has an associated $\mathbf{x}' \in \mathcal{L}'$ (more precisely, $\mathcal{L}' = \{\mathbf{x} \wedge \mathbf{c} \colon \mathbf{x} \in \mathcal{B}_\alpha(\mathbf{c})\}$ where $\mathbf{x} \wedge \mathbf{c}$ is viewed as a vector in S^v_α). By the arguments from Section 4.4 (and for the e^* defined there), it suffices to instead search for $\mathcal{L}'$-solutions, i.e., pairs $(\mathbf{x}', \mathbf{y}') \in \mathcal{L}' \times \mathcal{L}'$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$ and $|\mathbf{x} + \mathbf{y}| = w$. We will find those $\mathcal{L}'$-solutions by applying a quantum walk using an extra layer of LSF. More precisely, our walk will be over subsets S of $\mathcal{L}'$, and we aim to reach a vertex S that contains an $\mathcal{L}'$-solution. This search for a vertex containing an $\mathcal{L}'$-solution will be facilitated by the use of bucketing methods; however, this time we do both the bucketing and checking phase within the quantum walk.

Graph. Consider a quantum walk on the Johnson graph $J(|\mathcal{L}'|, s) = (V, E)$ for some suitable parameter $s < |\mathcal{L}'| = B$ satisfying $s = \tilde{O}(\frac{1}{\sqrt{p}})$.[14] In other words, the set of vertices is $V = \{S \subseteq \mathcal{L}' \colon |S| = s\}$ and two vertices S_1, S_2 are adjacent (i.e., $(S_1, S_2) \in E$) if and only if $|S_1 \cap S_2| = s - 1$.

[14] This constraint on s ensures that the number of $\mathcal{L}'$-solutions per vertex is $\tilde{O}(1)$ on average. It is used in our analysis of the set-up and update costs, as well as in our derivation of ϵ.

Data. To each vertex $S \in V$ we add some additional data structure $\textsc{data}(S)$ that allows us to efficiently check in the update step whether a newly added element forms an $\mathcal{L}'$-solution with one of the existing elements in a vertex (instead of having to search for a 'colliding element' among all $s-1$ other vertex elements). More precisely, the idea (originating from [CL21]) is as follows. We will invoke the RPC framework from [DEEK24] (Section 3.3) again and sample a random product code $\mathcal{C}'_f \sim R_{v,v',t',C}$ (i.e., $\mathcal{C}'_f \subseteq S^v_{v'}$) for $t' = \Theta(\sqrt{v})$ with v' to be determined. Let β be another parameter to be determined later. Instead of searching for any $\mathcal{L}'$-solution in S, the idea is to only search for solutions that both have overlap β with a bucket center from $\mathcal{C}'_f$.[15] Therefore, we will add the following data $\textsc{data}(S)$ to each vertex $S \in V$:

- The buckets $\mathcal{B}_\beta(\mathbf{c}') := \{\mathbf{x}' \in S \colon |\mathbf{x}' \wedge \mathbf{c}'| = \beta\}$ for each $\mathbf{c}' \in \mathcal{C}'_f$. These will be 'stored' using a data structure that allows for efficient inserting and removing of elements. (See [Amb07, BJLM13] for details.)
- To each bucket $\mathcal{B}_\beta(\mathbf{c}')$, we will add a list containing the $\mathcal{L}'$-solutions in the bucket, if there are any.
- We keep track of a bit indicating whether S is marked according to the following definition.

Marked vertices. Define the set $M \subseteq V$ of marked vertices as

$$M = \{S \in V \colon \exists\, \mathbf{c}' \in \mathcal{C}'_f, \exists\, \mathbf{x}', \mathbf{y}' \in \mathcal{B}_\beta(\mathbf{c}') \text{ s.t. } (\mathbf{x}', \mathbf{y}') \text{ is an } \mathcal{L}'\text{-solution}\}.$$

Note that this is a subset of the set of vertices containing an $\mathcal{L}'$-solution. In particular, an $\mathcal{L}'$-solution will only be detected if it (the pair) can be found in a $\mathcal{C}'_f$-bucket.

We will now describe the parameters determining the expected time complexity of the quantum walk; the memory complexity will be discussed after. We will repeatedly make use of the fact that, for any $\mathbf{x}' \in \mathcal{L}'$, the set $\mathcal{VC}_\beta(\mathbf{x}') := \{\mathbf{c}' \in \mathcal{C}'_f \colon |\mathbf{x}' \wedge \mathbf{c}'| = \beta\}$ of its valid buckets can be computed in time $|\mathcal{VC}_\beta(\mathbf{x}')| + 2^{o(n)}$ by Theorem 3.11. Furthermore, for ease of presentation, we will often omit writing down $\tilde{O}(\cdot)$ and factors of the form $2^{o(n)}$.

S*, set-up cost.* Constructing a uniform superposition over all $S \in V$ costs $\tilde{O}(s)$. In addition, we need to compute the data for each vertex S. We will construct the quantum data structure storing the buckets $\mathcal{B}_\beta(\mathbf{c}')$ by computing, for each $\mathbf{x}' \in S$, the set $\mathcal{VC}_\beta(\mathbf{x}')$ of valid buckets, and then insert $\mathbf{x}'$ to the right locations in the quantum data structure. Since insertion can be done efficiently, for instance using one of the quantum data structures in [Amb07, BJLM13], this takes $\tilde{O}(s \cdot \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|])$ time in expectation. It remains to construct the second and third part of the data. Note that the condition on s implies that the expected number of $\mathcal{L}'$-solutions per vertex is $\binom{s}{2}p = \tilde{O}(1)$. Therefore, applying Grover over all pairs in S allows for finding these (in expectation) $\binom{s}{2}p = \tilde{O}(1)$ pairs in time $\tilde{O}(s)$. For any found pair $(\mathbf{x}', \mathbf{y}')$, it then suffices to go over the valid β-buckets of $\mathbf{x}'$ to add $(\mathbf{x}', \mathbf{y}')$ to any bucket that also contains $\mathbf{y}'$.[16] If at least one pair is added, we add 1 as third component of the data, and 0 otherwise. Thus, altogether the construction of this second and third part of the data cost at most $\tilde{O}(s + \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|])$ time. We conclude that $\mathsf{S} = \max(s, s \cdot \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|], s + \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|]) = s \cdot \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|]$.

C*, checking cost.* Since we only need to check the last component of the data, we have $\mathsf{C} = 1$.

U*, update cost.* The dominating cost will be that of updating the data when going from a vertex S to a neighbor S'. Updating the first part of the data, i.e., the buckets, can be done in time $\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|]$, since it suffices to compute the sets of valid buckets for the old and new element, to remove the old element from its valid buckets, and to add the new element to its valid buckets. Updating the second part of the data (and the third part, if needed) can be done in time $\sqrt{\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|] \cdot \mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]}$: apply Grover to the union of all valid buckets of the old, respectively new, element to see if they are part of an $\mathcal{L}'$-solution; if so, remove, respectively add, this $\mathcal{L}'$-solution. (Here, we again make use of the fact that the condition on s implies that the number of solutions per vertex is $s \cdot p \leq s \cdot \sqrt{p} = \tilde{O}(1)$.) It follows that $\mathsf{U} = \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|] + \sqrt{\mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|] \cdot \mathbb{E}[|\mathcal{B}_\beta(\mathbf{c}')|]}$.

[15] This could mean that there are some false negatives: it might happen that S contains an $\mathcal{L}'$-solution, but that it isn't captured in one of the buckets. However, by choosing the parameters $|\mathcal{C}'_f|$ and β carefully we can control how likely this is to happen. (For more details, see the explanation in our derivation of ϵ.)

[16] This step could possibly be sped up, but there is no reason to as it does not dominate the set-up cost.

δ, spectral gap. As mentioned in Section 2, the spectral gap of the Johnson graph satisfies $\delta = \Omega(\frac{1}{s})$.

For the derivation of ϵ, we use the following lemma.

Lemma 4.11. *Let $\mathcal{C}'_f \sim R_{v,v',t',C}$ and let S be subset of $\mathcal{L}'$ of size s sampled independently and uniformly at random. Write E_1 for the event that there exists $\mathbf{c}' \in \mathcal{C}'_f$ with $\mathbf{x}', \mathbf{y}' \in \mathcal{B}_\beta(\mathbf{c}') := \{\mathbf{x}' \in S \mid |\mathbf{x}' \wedge \mathbf{c}'| = \beta\}$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$ and $|\mathbf{x} + \mathbf{y}| = w$ (i.e., the β-bucket of $\mathbf{c}'$ contains an $\mathcal{L}'$-solution). Then*

$$\Pr[E_1] \geq \min(q, \Omega(qs^2p))$$

where $q := \Pr[\exists \mathbf{c}' \in \mathcal{C}'_f \text{ s.t. } \mathbf{x}', \mathbf{y}' \in \mathcal{B}_\beta(\mathbf{c}')]$ for an arbitrary $\mathbf{x}', \mathbf{y}' \in \mathcal{L}'$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$.

Proof (Proof of Lemma 4.11). Define E_0 to be the event that there exist $\mathbf{x}', \mathbf{y}' \in S$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$ and $|\mathbf{x}+\mathbf{y}| = w$. Note that $\Pr[E_0] = \min(1, \Theta(s^2p(e^*))) = \min(1, \Theta(s^2p))$. Therefore, we have that

$$\Pr[E_1] = \frac{\Pr[E_1 \mid E_0]\Pr[E_0]}{\Pr[E_0 \mid E_1]} \geq \Pr[E_1 \mid E_0]\Pr[E_0] = \Pr[E_1 \mid E_0]\min(1, \Theta(s^2p)).$$

So it remains to show that $\Pr[E_1 \mid E_0] \geq \Omega(q)$. So suppose that E_0 holds. Then there exists a pair $\mathbf{x}', \mathbf{y}' \in S$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$ and $|\mathbf{x}+\mathbf{y}| = w$. It then follows that the probability that E_1 holds (conditional on E_0) is at least the probability that there exists $\mathbf{c}' \in \mathcal{C}'_f$ such that this particular pair $\mathbf{x}', \mathbf{y}'$ is in the bucket of $\mathbf{c}'$. In other words,

$$\Pr[E_1 \mid E_0] \geq \Pr[\exists \mathbf{c}' \in \mathcal{C}'_f \text{ s.t. } \mathbf{x}', \mathbf{y}' \in \mathcal{B}_\beta(\mathbf{c}')]$$

for any $\mathbf{x}', \mathbf{y}' \in S$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$ and $|\mathbf{x}+\mathbf{y}| = w$. By Lemma 4.10, we have that $\Pr[\exists \mathbf{c}' \in \mathcal{C}'_f \text{ s.t. } \mathbf{x}', \mathbf{y}' \in \mathcal{B}_\beta(\mathbf{c}')] = q$ from which it follows that $\Pr[E_1 \mid E_0] = \Omega(q)$, finishing the proof. □

ϵ, probability of a vertex being marked. Note that $\epsilon = \Pr[E_1]$ for E_1 as defined in the statement of Lemma 4.11. It follows from the lemma that $\epsilon = \min(q, \Omega(q\binom{s}{2}p))$, where $q := \Pr[\exists \mathbf{c}' \in \mathcal{C}'_f \text{ s.t. } \mathbf{x}', \mathbf{y}' \in \mathcal{B}_\beta(\mathbf{c}')]$ for any $\mathbf{x}', \mathbf{y}'$ satisfying $|\mathbf{x}' \wedge \mathbf{y}'| = e^*$. Our condition on s implies that $\binom{s}{2}p = \tilde{O}(1)$, and thus $\epsilon = \tilde{\Omega}(qs^2p)$.

Conclusion on expected runtime. Since the number of solution pairs in a bucket $\mathcal{B}_\alpha(\mathbf{c})$ is in expectation $t = O(|\mathcal{B}_\alpha(\mathbf{c})|^2p)$, the expected runtime of the quantum walk, repeated t times, is as given in the theorem statement. □

Remark 4.12. The proof can be adapted to relax the condition $s = \tilde{O}(1/\sqrt{p})$ to $s = \tilde{O}(1/\sqrt{pq})$ if we replace $\epsilon = s^2pq$ by $\epsilon = \min(q, s^2pq)$. However, when we make these changes in our implementation of the algorithm, a numerical optimization of the parameters does not result in better complexities. Therefore, we only present the proof for $s = \tilde{O}(1/\sqrt{p})$ here.

It remains to prove the claims on the memory complexity.

Memory complexity analysis

Proof (Proof of Theorem 4.7: Part 2/2, memory complexity).

Classical memory. We need to store the bucket $\mathcal{B}_\alpha(\mathbf{c})$ (or, equivalently, the list $\mathcal{L}'$), so the algorithm uses classical space $\tilde{O}(|\mathcal{B}_\alpha(\mathbf{c})|)$.

QRACM. The quantum walk needs quantum access to the bucket $\mathcal{B}_\alpha(\mathbf{c})$, stored classically and of size $M_{QRACM} = \tilde{O}(|\mathcal{B}_\alpha(\mathbf{c})|)$.

Quantum memory and QRAQM. The quantum walk stores a superposition of vertices. Each vertex contains s elements and some additional data, including the buckets of the second layer. (The other components of the data will not dominate the QRAQM cost, so we can safely ignore those.) Within each vertex, a vector $\mathbf{x}'$ is inserted in $|\mathcal{VC}_\beta(\mathbf{x}')|$ buckets. All together, we need to store this on a quantum register of size $M_Q = s + s \cdot \mathbb{E}[|\mathcal{VC}_\beta(\mathbf{x}')|]$. This is the required number of qubits to run the quantum walk. Since we use the data structures from [Amb07] (or [BJLM13]) to efficiently delete and insert vectors in the quantum superposition, we require this whole quantum register to allow efficient QRAQM, hence we need $M_{QRAQM} = M_Q$. □

5 Numerical Results

In this section, we summarize our numerical results on the asymptotic runtime and memory of four different algorithms for the near-neighbor search (NNS) problem defined in Section 3 (Problem 3.3), and consequently for code sieving. As our work is inspired by its lattice-based analog, we complete the analysis by also including the complexities of the lattice-based equivalents of the four algorithms. The algorithms we compare are the following:

- Classical: The sieving algorithm originally introduced in [DEEK24]. It is based on the RPC approach described in Section 3.3, which is the best known approach in the classical case. Its lattice-based analog is presented in [BDGL16].
- Grover: The first quantum algorithm introduced in our paper (Section 4.2). It is a natural modification from the classical RPC algorithm to the quantum model, obtained using Grover's algorithm. A lattice-based analog was introduced in [Laa15].
- QW + LSF: A more general quantum approach based on quantum walks in combination with the additional layer of locality-sensitive filtering, as explained in Section 4.3. Its lattice-based analog was introduced in [CL21, Sec. 4].
- QW + LSF + Sparsification: A variant of our quantum-walk algorithm obtained using sparsification, as explained in Section 4.5. The lattice-based analog is given in [CL21, Sec. 5].

Figure 2 shows the asymptotic runtime of the four algorithms in the code-based setting. The figures are obtained by calculating the asymptotic runtime in 100 equidistant values of $\omega := w/n$ ranging in $[0, 0.5)$, where n, w are parameters of Problem 3.3 and w is chosen such that there is a unique solution to the problem on average (i.e., we analyze the problem in the unique-decoding regime).

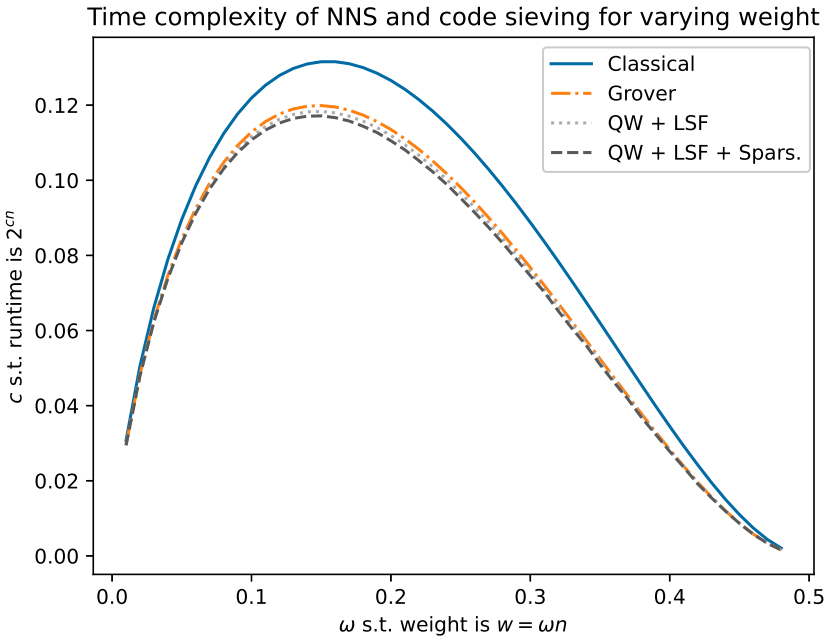

Fig. 2. Comparison of the asymptotic runtime of the four algorithms solving $\mathrm{NNS}(n, w, N)$ for $w = \omega \cdot n$, where $\omega \in [0, 0.5)$.

Table 1 shows the calculated asymptotic runtime and memory for the hardest instances of NNS in the code-based and lattice-based settings, where by the *hardest instances* we mean those values of ω for which the runtime curve in Figure 2 reaches its peak. The asymptotic memory is then calculated for the same value of ω. As we distinguish four different types of memory, we denote them by $M_C, M_Q, M_{QRACM}, M_{QRAQM}$ and refer to them as classical memory, quantum memory, QRACM, and QRAQM, respectively.

	Algorithms	t	M_C	M_Q	M_{QRACM}	M_{QRAQM}
Codes	Classical [DEEK24]	0.132	0.093	-	-	-
	Grover	0.120	0.094	0	0.026	-
	QW + LSF	0.118	0.094	0.024	0.031	0.024
	QW + LSF + Spars.	0.117	0.094	0.023	0.036	0.023
Lattices	Classical [BDGL16]	0.292	0.208	-	-	-
	Grover [Laa15]	0.265	0.208	0	0.058	-
	QW + LSF [CL21]	0.261	0.208	0.053	0.069	0.053
	QW + LSF + Spars. [CL21]	0.257	0.208	0.050	0.077	0.050

Table 1. The exponents of the asymptotic runtime (i.e., t s.t. runtime is $2^{tn+o(n)}$) and corresponding memory exponents for the hardest instances of NNS for codes (i.e., for those ω that yield the highest runtime for each considered algorithm), and the asymptotic runtime and memory of their lattice analogs.

Remark 5.1. We emphasize that Table 1 does not provide the asymptotic complexity of information-set decoding algorithms, in contrast to what was presented in [DEEK24]. In particular, the number 0.132 in the table refers to the runtime of the classical *sieving* algorithm from [DEEK24] (here referred to as Classical), and is not explicitly stated in their paper.

All numerical results were obtained using Python code. Our repository is available at `https://github.com/lynnengelberts/quantum-sieving-for-codes-public`.

Remark 5.2. We also implemented a variant of QW + LSF + Sparsification where we add the reusable quantum-walk techniques from [BCSS23] (in a similar way as they apply their techniques to [CL21]). Similar to their result on lattice sieving, we only obtain a minor improvement, namely an exponent of $t = 0.1169$ instead of 0.1171. (In lattice sieving, [BCSS23] obtain an exponent of $t = 0.2563$ instead of 0.2570.) Our implementation is available in our repository. We leave the elaboration of the analytical details of this approach as potential future work.

Some observations. Our numerical results show that our quantum algorithms provide a speed-up in comparison with the classical runtime [DEEK24]. While this is non-surprising, it is not necessarily guaranteed. Despite the structural differences between codes and lattices (particularly, their metric), we observe that similar techniques yield comparable improvements in the asymptotic runtime for sieving in both contexts. Grover's algorithm applied to the classical version delivers the most substantial quantum speed-up. Nevertheless, our quantum-walk algorithms outperform the version using Grover, with a slight improvement obtained using the sparsification technique. It appears that fundamentally different quantum techniques are needed to obtain more significant improvements.

While our work only focused on improving the asymptotic time complexities of sieving algorithms, it is also important to quantify the memory costs required by these algorithms, and minimize them. In particular, note that the improved runtime obtained when moving from classical to Grover, and subsequently to the quantum-walk algorithms, comes at the cost of an increased memory complexity. In particular, the use of Grover requires quantum memory and QRACM access. The use of quantum walks additionally requires QRAQM, which is an even stronger assumption on the memory model. Note that this trade-off between improved runtime and stronger memory requirements is consistent with the observations from other quantum algorithms in code-based (and lattice-based) cryptanalysis.

6 On the Application to Quantum Information-Set Decoding

Information-set decoding (ISD) algorithms are known as the most efficient generic attacks against the decoding problem for a wide range of parameters.[17]. Each of these can be seen as improvements of the algorithm originally introduced by Prange [Pra62]. We view ISD algorithms as a framework, as formulated in [FS09]. Specifically, following [GJN23] and [DEEK24] we see the sieving algorithm as a subroutine of an ISD algorithm within the framework. We refer to the resulting algorithm as *SievingISD*. In this section, we analyze a quantum analog of SievingISD.

Remark 6.1. In this section, the parameters n, w have a different meaning than in previous sections.

6.1 ISD and SievingISD

The central computational problem underlying the security of code-based primitives is the aforementioned decoding problem (Problem 3.1) with $N = 1$. For completeness, we restate the $N = 1$ variant here. We are primarily interested in the complexity of the cryptographically relevant instances of this problem. Namely, we analyze the problem for $\mathcal{C}$ being a random $[n, k]$ code, and w chosen such that there is only one solution to the problem on average. We refer to these instances as a *unique decoding instance* with parameters (n, k, w).

Problem 6.2 (Decoding problem, DP(n, k, w)). Given an $[n, k]$ binary linear code $\mathcal{C}$ and an integer value w, find a codeword $\mathbf{x}_\mathcal{C} \in \mathcal{C}$ of weight $|\mathbf{x}_\mathcal{C}| := w$.

Algorithm 3 presents the ISD framework, using the same formulation as in [DEEK24]. As part of the input, the algorithm is given an oracle $\mathcal{A}$ that, given a $[n', k]$ binary linear code $\mathcal{C}'$ and an integer w', returns N independent and uniformly random weight-w' codewords in $\mathcal{C}'$. We use the notation $\pi_\mathbf{x}(\cdot)$ for code puncturing, which is defined as follows.

Definition 6.3 (Code puncturing, $\pi_\mathbf{x}(\cdot)$). *For a linear $[n, k]$ code $\mathcal{C}$ and a binary vector $\mathbf{x} \in \mathbb{F}_2^n$ with $|\mathbf{x}| = n'$ we define by $\pi_\mathbf{x} : \mathbf{c} \mapsto \mathbf{c} \wedge \mathbf{x}$ the puncturing function relative to the support of $\mathbf{x}$, and we define $\pi_\mathbf{x}(\mathcal{C})$ to be the corresponding punctured code, which is a $[n', k]$ binary linear code.*

Algorithm 3: Information-set decoding (ISD)

Input : $[n, k]$ linear code $\mathcal{C}$, weight w, and an oracle $\mathcal{A}$ as described above (for fixed $n' > k$, w', N)
Output: $\mathbf{c} \in \mathcal{C}$ such that $|\mathbf{c}| = w$

```
while False do
    Choose x ∈ S^n_{n'} uniformly at random and repeat if dim(π_x(C)) ≠ k.
    L ← A(π_x(C), w').
    if ∃ y ∈ L: |π_x^{-1}(y)| = w then
        return π_x^{-1}(y)
```

We remark that for a uniformly random $\mathbf{x} \in \mathbb{F}_2^n$ with $|\mathbf{x}| = n'$, it happens with constant probability that $\dim(\pi_\mathbf{x}(\mathcal{C})) = k$ [Coo00].

We refer to *SievingISD* as any ISD algorithm in the form of Algorithm 3 that uses a sieving algorithm (Algorithm 1) as input oracle $\mathcal{A}$. Recall that by Lemma 3.5 and the subsequent discussion that this requires $N = \Omega\left(\frac{\binom{n'}{w'}}{\binom{w'}{w'/2}\binom{n'-w'}{w'/2}}\right)$.

[17] For a more specific range of parameters, there exist more efficient attacks such as statistical decoding [CDMT22].

6.2 Quantum ISD and Quantum SievingISD

We will now discuss quantum analogs of the ISD framework and SievingISD. Quantum ISD algorithms (e.g., [KT17,Kir18]) are based on the idea that one can apply amplitude amplification (AA) [BHMT02] to speed up the search for the solution of the decoding problem over multiple iterations of the ISD. Furthermore, the subroutine $\mathcal{A}$ is allowed to be quantum. It results in the following quantum analog of [DEEK24, Theorem 3.1]. Here, and in the remainder of Section 6, we only focus on the runtime of the (quantum) ISD algorithms, not its space usage.

Theorem 6.4 (Quantum ISD). *Let $\mathcal{C}$ be a a unique decoding instance with parameters (n, k, w). Let $n' > k$ and let $w' \leq n'$. Suppose $\mathcal{A}$ is a algorithm that returns N independent and uniformly random weight-w' codewords in a given $[n', k]$ binary linear code, where $N \leq \binom{n'}{w'}/2^{n'-k}$.*
If the expected runtime of $\mathcal{A}$ is $T_{\mathcal{A}}$, then there is a quantum algorithm that returns a weight-w codeword in $\mathcal{C}$, if one exists, in expected time

$$\tilde{O}\left(\frac{T_{\mathcal{A}}}{\sqrt{p_1 p_2}}\right)$$

where $p_1 := \frac{\binom{n'}{w'}\binom{n-n'}{w-w'}}{\binom{n}{w}}$ and $p_2 := \frac{N \cdot 2^{n'-k}}{\binom{n'}{w'}}$.

Proof. (This result is not new, but we sketch the proof for completeness.) The correctness follows from the proof of Theorem 3.1 in [DEEK24]. Moreover, since the success probability of the ISD algorithm (Algorithm 3) is given by $p_1 p_2$, amplitude amplification (Theorem 2.9) allows the algorithm to succeed after $\frac{1}{\sqrt{p_1 p_2}}$ iterations. One iteration is dominated by the time it takes for algorithm $\mathcal{A}$, which is $T_{\mathcal{A}}$. □

Remark 6.5. We recall from Section 3 that the upper bound on N is to ensure that there exist N codewords on average (for a random code $\mathcal{C}$) as output of Algorithm 1. Note that it ensures $p_2 \leq 1$.

If $N = \Omega\left(\binom{n'}{w'} / \left(\binom{w'}{w'/2}\binom{n'-w'}{w'/2}\right)\right)$, we can use a (classical or quantum) sieving algorithm as the subroutine $\mathcal{A}$, resulting in a natural quantum analog of SievingISD. We refer to it as *quantum SievingISD*. It turns out that quantum SievingISD does not allow for runtime close to the best known quantum ISD algorithms, as we will now show.

6.3 Limitations of Quantum SievingISD

We numerically illustrate that, contrary to the classical setting, *quantum* SievingISD cannot do better than what we will refer to as *quantum Prange*, the classical Prange algorithm quantized with AA due to [Ber10]. Quantum Prange was the first quantum ISD algorithm, and therefore a natural starting point for comparison.[18] We recall its time complexity.

Lemma 6.6 (Quantum Prange, [Ber10]). *Consider a unique decoding instance of DP with parameters n, k, w. Then there is a quantum algorithm that solves it in time*

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{w}}{\binom{n-k}{w}}}\right).$$

Let us first formalize our claim. Recall that any (classical or quantum) SievingISD algorithm requires N to satisfy

$$N = \Omega\left(\frac{\binom{n'}{w'}}{\binom{w'}{\frac{w'}{2}}\binom{n'-w'}{\frac{w'}{2}}}\right) \quad \text{and} \quad N = O\left(\frac{\binom{n'}{w'}}{2^{n'-k}}\right). \tag{5}$$

While the ISD framework itself imposes the upper bound (see Remark 6.5), the lower bound is imposed due to using a *sieving* subroutine (recall Lemma 3.5). The lower bound turns out to be a bottleneck in being able to improve over quantum Prange, as the following claim (and its justification) illustrates. Since our justification of the claim is partly numerical, we do not refer to it as a 'theorem'.

[18] We remark that the more recent quantum ISD algorithms in [KT17, Kir18] have even better time complexities than quantum Prange.

Claim. Consider a unique decoding instance of DP with parameters n, k, w. For all $n', w', N \in \mathbb{N}$ satisfying Equation (5), there is no (classical or quantum) algorithm for the oracle $\mathcal{A}$ in Algorithm 3 such that the resulting quantum ISD algorithm has a better runtime than quantum Prange [Ber10].

Justification of the claim. A trivial lower bound on the runtime of *any* classical or quantum algorithm for the oracle $\mathcal{A}$ in Algorithm 3 (with corresponding parameters n', w', N) is given by N, since the framework requires it to output N solutions. Thus, given any such algorithm $\mathcal{A}$ and parameters n', w', N satisfying Equation (5), the runtime T of the resulting quantum ISD algorithm must satisfy

$$T \geq \frac{N}{\sqrt{p_1 p_2}} = \sqrt{\frac{N\binom{n}{w}}{2^{n'-k}\binom{n-n'}{w-w'}}}$$

where $p_1 := \frac{\binom{n'}{w'}\binom{n-n'}{w-w'}}{\binom{n}{w}}$ and $p_2 := \frac{N 2^{n'-k}}{\binom{n'}{w'}}$ as in Theorem 6.4. For fixed n', w', this lower bound on T is minimal when N is as small as possible, i.e., when $N = \Theta(N_{n',w'})$ for $N_{n',w'} := \binom{n'}{w'} / \left(\binom{w'}{\frac{w'}{2}}\binom{n'-w'}{\frac{w'}{2}}\right)$. (In the remainder of the argument, we will leave out constant factors and assume for simplicity that $N = N_{n',w'}$.) It follows that T is lower bounded by

$$\min_{(n',w',N) \text{ satisfying (5)}} \sqrt{\frac{N\binom{n}{w}}{2^{n'-k}\binom{n-n'}{w-w'}}} = \min_{(n',w') \text{ s.t. } \binom{w'}{\frac{w'}{2}}\binom{n'-w'}{\frac{w'}{2}} \geq 2^{n'-k}} \sqrt{\frac{\binom{n'}{w'}\binom{n}{w}}{\binom{w'}{\frac{w'}{2}}\binom{n'-w'}{\frac{w'}{2}}2^{n'-k}\binom{n-n'}{w-w'}}}. \quad (6)$$

We want to show that this is never better than the runtime of quantum Prange, i.e., $\sqrt{\frac{\binom{n}{w}}{\binom{n-k}{w}}}$.

Numerically, we minimized Equation (6). We obtain that, for all n', w' satisfying $\binom{w'}{\frac{w'}{2}}\binom{n'-w'}{\frac{w'}{2}} \geq 2^{n'-k}$, it asymptotically holds that

$$\min_{(n',w') \text{ s.t. } \binom{w'}{\frac{w'}{2}}\binom{n'-w'}{\frac{w'}{2}} \geq 2^{n'-k}} \sqrt{\frac{\binom{n'}{w'}\binom{n}{w}}{\binom{w'}{\frac{w'}{2}}\binom{n'-w'}{\frac{w'}{2}}2^{n'-k}\binom{n-n'}{w-w'}}} \geq \sqrt{\frac{\binom{n}{w}}{\binom{n-k}{w}}}.$$

Our code is publicly available in the aforementioned GitHub repository. (Note here that the constraint $\binom{w'}{\frac{w'}{2}}\binom{n'-w'}{\frac{w'}{2}} \geq 2^{n'-k}$ is induced by Equation (5).) □

In fact, the optimal values obtained through numerical optimization essentially correspond to the values characterizing quantum Prange: $n' = k$ and $w' = 0$.

Remark 6.7 (Comparison with the classical setting). The same argument does not apply to *classical* SievingISD (non-surprisingly, since [DEEK24] have shown that it indeed outperforms classical Prange). There, a trivial lower bound on the runtime of the oracle $\mathcal{A}$ (for any parameters n', k, w', N) would again be N, giving a (possibly non-tight) lower bound on the time complexity of classical SievingISD of $N/(p_1 p_2)$. Unlike in the quantum setting, the dependency (and, in particular, the lower bound) on N disappeared as it is canceled out by p_2, namely $N/(p_1 p_2) = \binom{n}{w} / \left(\binom{n-n'}{w-w'} 2^{n'-k}\right)$ since p_2 is linear in N.

6.4 On Overcoming the Limitations of Quantum SievingISD

The previous section illustrates limitations of the presented quantum SievingISD framework and implies that the framework should be adapted to outperform quantum Prange [Ber10]. Specifically, the main bottleneck appears to be the *lower bound* on the output size N of the sieving subroutine imposed by Lemma 3.5 (where we emphasize that in this section we use n', w' to specify the dimension and weight). Recall that the factors $T_{\mathcal{A}}$ and p_2 in the expected runtime $\tilde{O}(T_{\mathcal{A}}/\sqrt{p_1 p_2})$ of a quantum ISD algorithm both depend on N. Here, we present two natural approaches for potentially overcoming these limitations and explain why neither of them works.

Approach 1: From Pairs to Tuples. In this paper, we were considering finding pairs $(\mathbf{x}_1, \mathbf{x}_2)$ of vectors in our input list $\mathcal{L}$ such that $|\mathbf{x}_1 + \mathbf{x}_2| = w'$ (called *solution pairs*). To guarantee the existence of $N = |\mathcal{L}|$ solution pairs, Lemma 3.5 implied that we need the lower bound on N. Inspired by lattice-based cryptanalysis, an idea to obtain a smaller lower bound on N is to focus instead on finding t-tuples (for some $t > 2$).[19] That is, the sieving algorithm is instructed to (repeatedly) find N tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_t) \in \mathcal{L}^t$ satisfying $|\mathbf{x}_1 + \ldots + \mathbf{x}_t| = w'$ for a given list $\mathcal{L}$ of size N. We refer to such tuples as *t-solutions*. Note that, for $t = 2$, we recover the original setting in which the algorithm searches for solution pairs.

Specifically, for $t \geq 2$, we say that a *t-sieving algorithm* is an algorithm constructed similarly to the 2-sieving algorithm in Algorithm 1, but instead searching for t-solutions: in each iteration i of the sieving part, it aims to find N t-solutions given a list $\mathcal{L}$ of N independent and uniformly random vectors from $\mathcal{S}_{w'}^{n'}$. Writing $p^{(t)}$ for the probability that a t-tuple of independent and uniformly random vectors in $\mathcal{S}_{w'}^{n'}$ forms a t-solution, the expected number of t-solutions in $\mathcal{L}$ is then given by $N^t p^{(t)}$. To ensure that, on average, there exist at least N t-solutions, the list size N thus needs to satisfy $N^t p^{(t)} \geq N$, possibly resulting in a reduced lower bound on the output size N.

We refer to *quantum t-sieving ISD* as the resulting quantum ISD algorithm where we instantiate the oracle $\mathcal{A}$ with a t-sieving algorithm, where we recall that $\mathcal{A}$ aims to find N solutions to $\mathrm{DP}(n', k, w')$. We keep the meaning of p_1 and p_2 from Theorem 6.4, and write $p_2 = Nq_2$ to highlight that p_2 depends on the output size N. (Note that p_1 and q_2 do not depend on N or t.) The expected runtime of quantum t-sieving ISD is then $\tilde{O}\left(T_{\mathcal{A}}/\sqrt{p_1 q_2 N}\right)$, where $T_{\mathcal{A}}$ is the expected runtime of $\mathcal{A}$.

However, even if the use of t-tuples for $t > 2$ might potentially reduce the lower bound on the output size N, we obtain the following lower bound on the runtime of quantum t-sieving ISD. We remark that the assumption holds, for instance, if the runtime of any t-sieving algorithm is lower bounded by the optimal runtime of 2-sieving (since $1/p^{(2)}$ is a lower bound on the output size of a 2-sieving algorithm).

Proposition 6.8 (A lower bound on quantum t-sieving ISD). *Assume that the runtime of any t-sieving algorithm is at least $1/p^{(2)}$. Consider a quantum ISD algorithm with the aforementioned t-sieving algorithm as oracle $\mathcal{A}$. The expected runtime of this algorithm is $\Omega(1/\sqrt{p_1 q_2 p^{(2)}})$.*

By Section 6.3, the latter is (asymptotically) never better than the runtime of quantum Prange. Hence, we can conclude that, under the stated assumption, quantum t-sieving ISD does no better than quantum Prange for all $t \geq 2$.

Proof. We analyze the cases $N \leq 1/p^{(2)}$ and $N \geq 1/p^{(2)}$ separately, where N (as usual) denotes the output size of $\mathcal{A}$. First, suppose that $N \leq 1/p^{(2)}$. By the assumption, the runtime $T_{\mathcal{A}}$ of any t-sieving algorithm is at least $1/p^{(2)}$, so the runtime of the resulting quantum ISD algorithm is lower bounded by $T_{\mathcal{A}}/\sqrt{p_1 q_2 N} \geq 1/\left(p^{(2)}\sqrt{p_1 q_2 N}\right) \geq 1/\sqrt{p_1 q_2 p^{(2)}}$. On the other hand, if $N \geq 1/p^{(2)}$, then $T_{\mathcal{A}}/\sqrt{p_1 q_2 N} \geq \sqrt{N}/\sqrt{p_1 q_2} \geq 1/\sqrt{p_1 q_2 p^{(2)}}$ since $T_{\mathcal{A}} \geq N$. (Note that we omit writing $\tilde{O}(\cdot)$ throughout the proof.) □

Approach 2: Varying List Sizes. The lower bound on the output size N of the sieving subroutine comes from maintaining the same list size N throughout each iteration in the sieving algorithm. A natural question to ask is whether it would be possible to vary the list size in order to overcome this intrinsic limitation of quantum SievingISD.

We propose the following approach for varying list sizes. Suppose that for given parameters n', w' there exists a (classical/quantum) algorithm for the subroutine $\mathcal{A}$ in the quantum ISD algorithm that iteratively applies a sieving step of the following form, where the values N_i are arbitrary. It starts by sampling a list $\mathcal{L}_0$ of N_0 independent and uniformly random vectors from $\mathcal{S}_{w'}^{n'}$. For iteration $i = 1$ up to $i = n' - k$, the algorithm finds N_i pairs $(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{L}_{i-1}^2$ satisfying $|\mathbf{x}_1 + \mathbf{x}_2| = w'$, yielding a new list $\mathcal{L}_i$ of size N_i. Note that the expected number of solution pairs is $N_{i-1}^2 p$, where $p := \binom{w'}{w'/2}\binom{n'-w'}{w'/2}/\binom{n'}{w'}$ is the probability that a uniformly random pair is a solution pair (previously denoted by $p^{(2)}$). Therefore, we will only consider N_i satisfying $N_i \leq N_{i-1}^2 p$.

[19] The intuition comes from tuple-sieving algorithms (e.g., [KMPM19]) which consider $t > 2$ to reduce the lower bound on N.

Remark 6.9. If $N_i = N_0$ for all i, then we get the same lower bound on the list size as we considered so far, namely $1/p$, resulting in the original setting. We now instead consider the situation where the N_i's might differ.

We now sketch the proof that there is no choice of $N_0, \ldots, N_{n'-k}$ satisfying $N_i \leq N_{i-1}^2 p$ for which this algorithm yields runtime better than that of quantum Prange (Lemma 6.6).

We start by observing that the overall runtime of this quantum ISD algorithm is essentially $T := T_{\max}/\sqrt{p_1 q_2 N_{n'-k}}$, where $T_{\max}$ is the maximum among the runtimes of the iterations, and p_1 and $p_2 = q_2 N_{n'-k}$ are the probabilities from Theorem 6.4. (Note that the output size $N_{n'-k}$ was previously denoted by N.) Since $T_{\max} \geq N_{\max} := \max_{i \geq 0} N_i$, we have $T \geq N_{\max}/\sqrt{p_1 q_2 N_{n'-k}}$. Recall from Section 6.3 that, for all allowed (n', w'), $1/\sqrt{p_1 q_2 p}$ is asymptotically lower bounded by the runtime of quantum Prange. Therefore, it suffices to show that there is no choice of $N_0, \ldots, N_{n'-k}$ such that $N_{\max}/\sqrt{N_{n'-k}} < 1/\sqrt{p}$.

Suppose for contradiction there is a choice of $N_0, \ldots, N_{n'-k}$ satisfying $N_i \leq N_{i-1}^2 p$ for all $i \geq 1$ and $N_{\max}/\sqrt{N_{n'-k}} < 1/\sqrt{p}$. We analyze the two cases $N_{n'-k} \geq \frac{1}{p}$ and $N_{n'-k} < \frac{1}{p}$ separately. We start with the case $N_{n'-k} \geq 1/p$. Then $N_{\max}/\sqrt{N_{n'-k}} \geq \sqrt{N_{n'-k}} \geq 1/\sqrt{p}$.

It remains to consider the case $N_{n'-k} < 1/p$. If $N_0 \geq 1/p$, then $N_0 > N_{n'-k}$, so $N_{\max}/\sqrt{N_{n'-k}} \geq N_0/\sqrt{N_{n'-k}} > \sqrt{N_0} \geq 1/\sqrt{p}$. Finally, consider the case that $N_0 < 1/p$. Note that $N_i \leq N_{i-1}^2 p$ for all $i \geq 1$ implies that $N_i \leq N_0^{2^i} p^{2^i - 1}$ for all $i \geq 0$. In particular, the size of the output list satisfies $N_{n'-k} \leq N_0^{2^{n'-k}} p^{2^{n'-k}-1}$. Therefore, we obtain that

$$\frac{N_{\max}}{\sqrt{N_{n'-k}}} \geq \frac{N_0}{\sqrt{N_{n'-k}}} \geq \sqrt{\frac{N_0^2}{N_0^{2^{n'-k}} p^{2^{n'-k}-1}}} = \frac{1}{\sqrt{p}} \frac{1}{\sqrt{N_0^{2^{n'-k}-2} p^{2^{n'-k}-2}}} > \frac{1}{\sqrt{p}}$$

since $N_0 p < 1$. That is, in all possible cases we showed that $N_{\max}/\sqrt{N_{n'-k}} \geq 1/\sqrt{p}$, so we reached a contradiction. We conclude that there is no suitable choice for the N_i that enables to outperform quantum Prange.

References

AKS01. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 601–610. ACM, 2001.

Amb07. A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007.

BBHT98. M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.

BCSS23. X. Bonnetain, A. Chailloux, A. Schrottenloher, and Y. Shen. Finding many collisions via reusable quantum walks - application to lattice sieving. In *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, *Lecture Notes in Computer Science* 14008, pages 221–251. Springer, 2023.

BDGL16. A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 10–24. SIAM, 2016.

Ber10. D. J. Bernstein. Grover vs. McEliece. In *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*, *Lecture Notes in Computer Science* 6061, pages 73–80. Springer, 2010.

BHMT02. G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information*, page 53–74, 2002.

BJLM13. D. J. Bernstein, S. Jeffery, T. Lange, and A. Meurer. Quantum algorithms for the subset-sum problem. In *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, *Lecture Notes in Computer Science* 7932, pages 16–33. Springer, 2013.

BJMM12. 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 2012*, *LNCS* 7237, pages 520–536. Springer, Heidelberg, April 2012.

BM18. L. Both and A. May. Decoding linear codes with high error rate and its impact for LPN security. In *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, *Lecture Notes in Computer Science* 10786, pages 25–46. Springer, 2018.

BMvT78. E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Inf. Theory*, 24(3):384–386, 1978.

Car20. K. Carrier. *Recherche de Presque-Collisions pour le Décodage et la Reconnaissance de Codes Correcteurs. (Near-collisions finding problem for decoding and recognition of error correcting codes)*. PhD thesis, Sorbonne University, France, 2020.

CDMT22. K. Carrier, T. Debris-Alazard, C. Meyer-Hilfiger, and J.-P. Tillich. Statistical decoding 2.0: Reducing decoding to LPN. In *ASIACRYPT 2022, Part IV*, *LNCS* 13794, pages 477–507. Springer, Heidelberg, December 2022.

CL21. A. Chailloux and J. Loyer. Lattice sieving via quantum random walks. In *ASIACRYPT 2021, Part IV*, *LNCS* 13093, pages 63–91. Springer, Heidelberg, December 2021.

Coo00. C. Cooper. On the distribution of rank of a random matrix over a finite field. *Random Struct. Algorithms*, 17(3-4):197–212, 2000.

DEEK24. L. Ducas, A. Esser, S. Etinski, and E. Kirshanova. Asymptotics and improvements of sieving for codes. In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, *Lecture Notes in Computer Science* 14656, pages 151–180. Springer, 2024.

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

FS09. M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In *ASIACRYPT 2009*, *LNCS* 5912, pages 88–105. Springer, Heidelberg, December 2009.

GJN23. Q. Guo, T. Johansson, and V. Nguyen. A new sieving-style information-set decoding algorithm. Cryptology ePrint Archive, Report 2023/247, 2023.

Gro96. L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219. ACM, 1996.

Kir18. E. Kirshanova. Improved quantum information set decoding. In *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 507–527. Springer, Heidelberg, 2018.

KMPM19. E. Kirshanova, E. Mårtensson, E. W. Postlethwaite, and S. R. Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In *ASIACRYPT 2019, Part I*, *LNCS* 11921, pages 521–551. Springer, Heidelberg, December 2019.

KT17. G. Kachigar and J.-P. Tillich. Quantum information set decoding algorithms. In *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 69–89. Springer, Heidelberg, 2017.

Laa15. T. Laarhoven. *Search problems in cryptography, From fingerprinting to lattice sieving*. PhD thesis, Eindhoven University of Technology, 2015.

MMT11. A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *ASIACRYPT 2011*, *LNCS* 7073, pages 107–124. Springer, Heidelberg, December 2011.

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

MO15. A. May and I. Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT 2015, Part I*, *LNCS* 9056, pages 203–228. Springer, Heidelberg, April 2015.

MV10. D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1468–1480. SIAM, 2010.

NV08. P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Math. Cryptol.*, 2(2):181–207, 2008.

Pra62. E. Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962.

SS81. R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems. *SIAM J. Comput.*, 10(3):456–464, 1981.

Ste88. J. Stern. A method for finding codewords of small weight. In *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, *Lecture Notes in Computer Science* 388, pages 106–113. Springer, 1988.