Quantum Cryptanalysis on Lattices and Codes

Johanna Loyer

Sorbonne Université

École doctorale Informatique, Télécommunications et Électronique (Paris)

Inria de Paris / Equipe-projet COSMIQ

Quantum Cryptanalysis on Lattices and Codes

Thèse de doctorat d'informatique

présentée par

Johanna LOYER

dirigée par André Chailloux et Nicolas Sendrier

soutenue publiquement le 18 décembre 2023

devant un jury composé de :

André Chailloux	Inria, Paris	Directeur
Nicolas Sendrier	Inria, Paris	Co-directeur
Elena Kirshanova	Technology Innovation Institute, Abu Dhabi	Rapporteure
Sean Hallgren	Pennsylvania State University	Rapporteur
André Schrottenloher	Inria, Rennes	Examinateur
Damien Vergnaud	LIP6, Paris	Président

Remerciements

Je veux tout d'abord te remercier, André, pour ces presque quatre ans qui sont passés si vite. Merci pour ta pédagogie, ta patience, ton encadrement qui me laissait libre tout en ne me laissant jamais me perdre. Merci pour toutes ces discussions passionnantes sur la science et ce qui l'entoure, pour ton humour, et surtout de m'avoir permis de réaliser cette thèse sur les pistes cyclables quantiques. J'ai beaucoup appris à tes côtés et j'espère continuer encore. Cela va très vite me manquer d'être "ta doctorante qui fait des remarques désobligeantes". Tous mes voeux de bonheur pour ta petite famille qui s'agrandit !

Merci Nicolas d'avoir accepté d'être directeur pour la partie la plus dure de la thèse (l'administratif). Merci de m'avoir initiée au monde occulte des codes grâce à ta grande pédagogie, et de t'être intéressé à mes travaux malgré nos sujets de recherche au départ un peu éloignés.

I would like to thank the jury members of my Ph.D. thesis, Elena Kirshanova, Sean Hallgren, Damien Vergnaud, and André Schrottenloher, for agreeing to be part of my jury and for their helpful comments that improved the manuscript.

Merci aux membres de l'équipe COSMIQ : Augustin, Aurélien, Axel, Charles, Clara, Clémence, Dounia, Maxime, Nicholas, Nicolas D., Pierre, Jules, Virgile, Anne, Anthony, Gaëtan, Jean-Pierre, Léo, Maria, Pascale ; et ceux qui gravitent souvent autour : Thomas, André S. et Yixin. Merci à Christelle de m'avoir sauvée à de nombreuses reprises des dédales administratifs.

Thank you Simona for all our interesting discussions during our shared lunches, for making me bilingual while I started from far; and for the *amazing* (to be pronounced with emphasis) journey to Amsterdam where I will join you soon (јер ми већ превише недостајеш!).

Merci Aurélie pour tes conseils avisés en choix de notations (*Let* ך *be a code...*) et pour les fous rires de décompression entre deux séances intenses à rédiger nos thèses. Pas merci de m'avoir abandonnée à QIP pour les "cool kids" de ta summer school, mais ne t'inquiète pas, je tairai quand même le terrible secret que tu m'as confié.

Merci Agathe pour tes idées innovantes, ta motivation, ton talent et ton inspiration (est-ce que j'en ai dit assez ?). Plus sérieusement, c'était un plaisir de t'avoir comme co-bureau. Prend soin du bureau quantique du bâtiment B que nous te léguons, et je compte sur toi pour y perpétuer la tyrannie des gâteaux sur les prochains petits stagiaires.

Thanks to the cryptology team at CWI for their warm welcome at the country of tulips, windmills, and foosball tournaments. Thanks Lisa, Eamonn, Pedro, Shane, Yu-Hsuan, Ludo, Patrick, and Lynn. Bedankt en tot snel!

Un merci particulier à Léo pour m'avoir fait me rendre compte que j'ai encore tant à apprendre en cryptologie, rien que dans le domaine des réseaux. Merci pour ta confiance (et notamment en mes compétences de baby-sitting pour baby-Johanna).

Merci à toutes les personnes formidables rencontrées et/ou revues en conférences, notamment Lucie, Clément, Morgane, Chloé, Maxime B. de Limoges, Daniel et Arjan de l'IRIF, et Angélique, Thomas, Pierre, Eric et Valentin de Thalès.

Merci à ma famille, de sang ou de cœur, d'avoir fait si bien semblant de vous intéresser à mes formules de maths pendant toutes ces années. Merci d'avoir été là, et de m'avoir tant apporté sur tout ce qui ne s'apprend pas dans les livres.

Merci Jérôme d'accepter de me partager avec mon amour pour la Science sans jalousie (ou si peu). Merci pour ton soutien infaillible et ton amour. Où que la vie nous mène, nous serons toujours ensemble face à l'adversité.

Contents

Notations

The following notations are used in this thesis. The abbreviation *iff.* means "if and only if", and *s.t.* means "such that".

Bachmann-Landau notations

- Given two functions f, g, we write $f(n) = \mathcal{O}(g(n))$ when $f(n) \leq g(n) \cdot k$ for some absolute constant k for n lare enough. The $\widetilde{\mathcal{O}}$ notation ignores polynomial factors in n: $\widetilde{\mathcal{O}}(f(n)) := \mathcal{O}(\text{polylog}(f(n))) = \mathcal{O}(\log(n)^k)$ for some absolute constant k. We will sometimes omit $\mathcal{O}$ or $\widetilde{\mathcal{O}}$ as we always focus on the asymptotic complexities in this thesis.
- $f = \Theta(g)$ means that for n large enough, there exists absolute constants k_1, k_2 such that $k_1 g(n) \leq f(n) \leq k_2 g(n)$.

Vectors

- Vectors are denoted in bold topped with an arrow, as $\vec{\boldsymbol{v}}$. We amalgamate vectors in $\mathbb{R}^n$ and points in $\mathbb{R}^n$. Even if not explicit, v_i stands for the i-th coordinate of $\vec{\boldsymbol{v}}$, for $i \in [n] := \{1, ..., n\}$.
- For a vector $\vec{\boldsymbol{v}} \in \mathbb{R}^n$, $\|\vec{\boldsymbol{v}}\| = \sqrt{\sum_{i=1}^n v_i^2}$ is the Euclidean norm of $\vec{\boldsymbol{v}}$. $\frac{\vec{\boldsymbol{v}}}{\|\vec{\boldsymbol{v}}\|}$ is the vector $\vec{\boldsymbol{v}}$ normalized.
- For two vectors $\vec{\boldsymbol{u}}, \vec{\boldsymbol{v}} \in \mathbb{R}^n$, $\langle \vec{\boldsymbol{u}} | \vec{\boldsymbol{v}} \rangle = \sum_i u_i \cdot v_i$ denotes their scalar product, and $\theta(\vec{\boldsymbol{u}}, \vec{\boldsymbol{v}}) = \arccos\left(\frac{\langle \vec{\boldsymbol{u}} | \vec{\boldsymbol{v}} \rangle}{\|\vec{\boldsymbol{u}}\| \|\vec{\boldsymbol{v}}\|}\right)$ denotes their non-oriented angle. $\vec{\boldsymbol{u}}$ and $\vec{\boldsymbol{v}}$ are orthogonal ($\vec{\boldsymbol{u}} \perp \vec{\boldsymbol{v}}$) if $\langle \vec{\boldsymbol{u}} | \vec{\boldsymbol{v}} \rangle = 0$.
- We denote the concatenated vector of $\vec{\boldsymbol{u}} = (u_i)_{i \in [n]}$ and $\vec{\boldsymbol{v}} = (v_i)_{i \in [m]}$ by $(\vec{\boldsymbol{u}} \| \vec{\boldsymbol{v}}) = (\vec{\boldsymbol{u}}_1, \ldots, \vec{\boldsymbol{u}}_n, \vec{\boldsymbol{v}}_1, \ldots, \vec{\boldsymbol{v}}_m)$.
- For a vector $\vec{\boldsymbol{v}} = (v_0, \ldots, v_{n-1})$, we denote $\vec{\boldsymbol{v}}_{|_{[i,j]}}$ the vector $(v_i, \ldots, v_{j-1})$ restricted on coordinates i to $j-1$.

Matrices

- For a complex matrix U we denote $U^\top$ its transpose, and $U^\dagger$ its conjugate transpose. U is a unitary matrix iff. $UU^\dagger = U^\dagger U = I$.
- For U, U' two matrices, we denote $U \otimes U'$ their tensor product (or Kronecker product), and for $n \in \mathbb{N}$, $U^{\otimes n} := \underbrace{U \otimes \cdots \otimes U}_{\text{n times}}$.
- Quantum states are denoted with the ket notation $|\cdot\rangle$. Two juxtaposed quantum states can be written indifferently $|\phi\rangle \otimes |\psi\rangle = |\phi\rangle|\psi\rangle = |\phi, \psi\rangle$.

Codes

- We denote codes by the Gothic character $\mathfrak{C}$ (as in the text C already denotes configurations and $\mathcal{C}$ a check cost). In Chapter 6, as there is no ambiguity, we simply use C.
- For $\boldsymbol{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{F}_q^n$ and $\mathbf{M} = (M_{i,j})_{\substack{0 \leq i < r-1 \\ 0 \leq j < n-1}} \in \mathbb{F}_q^{r \times n}$, we define their row-wise star product $\boldsymbol{x} \star \mathbf{M} := (x_j M_{i,j})_{\substack{0 \leq i < r-1 \\ 0 \leq j < n-1}}$.

1. Introduction

1.1. Background and issues

Public-key cryptography. In 1977, Rivest, Shamir and Adleman introduced the famous RSA encryption protocol [RSA77] after a party with plenty of wine [Sin99]. The security of this cryptosystem relies on a surprisingly simple idea: picking two large prime numbers p and q, it is easy to compute their product $N = p \cdot q$, while it is very hard from N to recover p and q. One can use N as a public key to encrypt messages, and the secret key (p, q) allows decrypting. A spy who gets an encrypted message has to solve this hard problem if they want to decrypt it without knowing the secret key. This protocol may have been secretly discovered by the British GCHQ twelve years before and only declassified in 1997 [Coc73], but this is a different story.

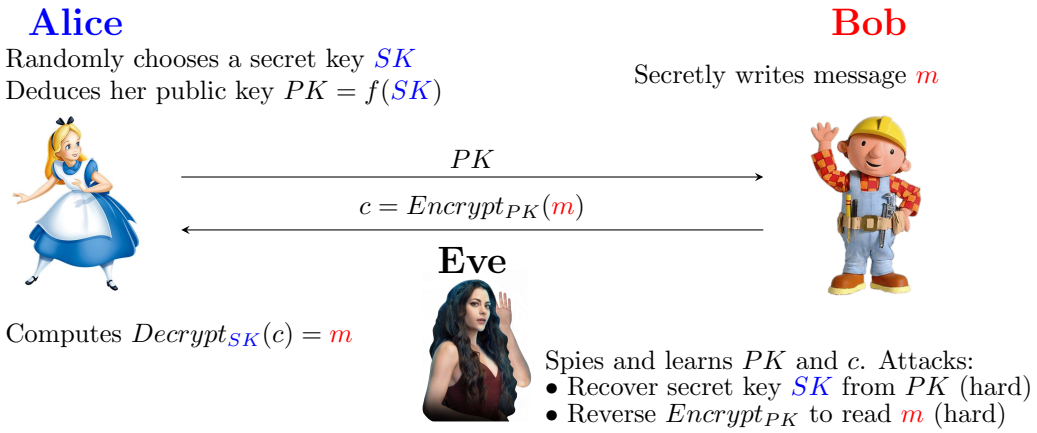


Figure 1.1: Encryption protocol: Bob sends a message non-understandable to anyone but Alice. Both agree on an efficiently computable one-way function $f(SK) = PK$ such that $Decrypt_{SK} = Encrypt_{PK}^{-1}$.

Several decades of research passed and the RSA problem attracted a lot of attention, but no one has been able to solve it for large numbers. Factoring a number in the order of 2^{2048} using a classical computer would take 300 trillion years [Lab19], which is more than the time since the beginning of the universe ("only" 13.8 billion years ago). So there was strong confidence in the hardness of this problem and in the security of cryptosystems based on it.

Quantum information theory. Since the 80s, several physicists suggested the idea of a computer that takes profit from quantum laws, instead of confining it to classical physics, in order to enlarge the possibilities of computations. The motivation for constructing such a computer would be quantum simulation [Fey82] that has applications in molecular chemistry and materials study.

In 1994, Shor published a breakthrough article [Sho94] in which he showed that the factorization problem would be efficiently solved by a quantum computer. Consequently, it theoretically breaks all the cryptosystems based on this problem like RSA and, killing two birds with one stone, also those based on the discrete logarithm problem [DH76; JMV01] that are also widely used. In stride during the late 90s, the first small quantum calculators were constructed [CGK98]. In addition to academic laboratories, several actors have begun to take an interest in the development of a quantum computer, among them giant companies (Google, IBM, Microsoft, Alibaba), specialized start-ups (D-Wave, Rigetti, Pasqal, Quandela, Alice & Bob, ...) and

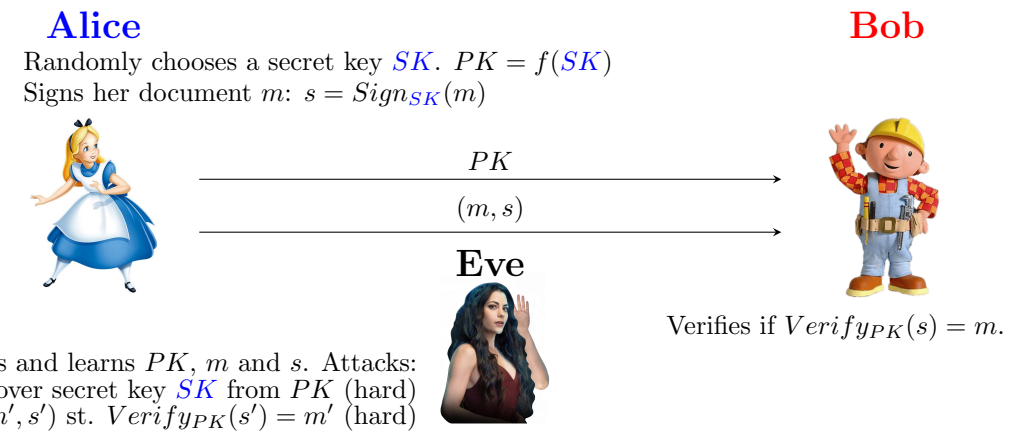


Figure 1.2: Electronic signature: certify that a document authentically comes from Alice. Both agree on an efficiently computable one-way function $f(SK) = PK$ such that $Sign_{SK} = Verify_{PK}^{-1}$.

national departments of defense (NSA according to Edward Snowden's leaks [RG14], and surely many other states). The obstacles remain significant: quantum error correction is required to face the decoherence of the qubits, and a good choice of architecture is needed to construct all the quantum gates that perform correctly.

Will a practical quantum computer ever exist? As this thesis does not deal with divination we sadly cannot answer this question. One can remember the beginnings of the classical computer that is today incredibly ubiquitous but originally started with a first attempt by Babbage whose architecture was not viable for scaling. The NMR architecture of the first quantum calculators was not either, but many other leads are being explored[1]. Until a practical quantum computer is constructed, or a discovery in physics states the impossibility of its construction... there is no knowledge about it, only questions, hopes, and beliefs.

Post-quantum cryptography. There is, however, one certainty: We cannot leave our digital security based on an arbitrary gamble. In the mid-2010s, began to emerge the idea that our security systems must be resistant to potential quantum attacks. The NSA stated that the US government should use quantum-safe cryptography, so in 2016, the American National Institute of Standards and Technology (NIST) announced a competition for quantum-safe encryption and digital signature schemes. Despite its "national" prerogatives, the choice of standardized schemes by the NIST influences the adoption of those schemes worldwide.

New cryptographic problems were proposed from either lattice theory, codes, multivariate polynomials, or isogenies. After five years during which cryptanalysts from all around the world searched for security flaws, four schemes were finally selected. Will be standardized: Kyber [Bos+18] for encryption, Dilithium [Duc+19], Falcon [Fou+18], and Sphincs+ [Ber+19] for digital signatures. The three first listed are lattice-based, and the last one is hash-based. Lattices seem to provide good performances, *i.e.* fast running time and short sizes of the keys and ciphers or signatures. However, having almost all the standardized schemes based on a single class of problems exposes to the danger of a potential attack that would break them all, as Shor's did with pre-quantum cryptography. Thus, in 2022, the NIST launched a second call for additional digital signatures. Once again, lattice and codes are under the spotlight.

The important question is: Are these schemes as secure as we think? Only time can allow us to trust a security system after a crowd of cryptanalysts tried unsuccessfully to solve their underlying problems. They have been less studied than that of RSA. To date, it is difficult to fully assess their current claimed security.

[1] You will find at this link a timeline sourced and regularly updated with the progress in the development of the quantum computer: `https://en.wikipedia.org/wiki/Timeline_of_quantum_computing_and_communication`.

1.2. Contributions

Starting my Ph.D. years in this context, I wanted to contribute to the global effort of estimating the hardness of the problems on which our security will rely. How fast and with what material resources can an attacker solve them? This is the question I tried to answer. My work has focused on two of these problems at the core of lattice-based and code-based cryptography:

- Shortest Vector Problem (SVP): A lattice is the set of all integer linear combinations of the vectors of a given basis. The well-named SVP asks to find the shortest vector in a lattice.
- Decoding problem (DP): A code is the set of all vectors product of its given generator matrix by any vector. The decoding problem asks to find the nearest code vector from a noisy one.

New faster algorithm to solve SVP using quantum walks

We present in Chapter 4 a new quantum algorithm that heuristically solves the Shortest Vector Problem using quantum walks. This is the first improvement in the asymptotic running time of quantum sieving algorithms since the work of Laarhoven [Laa15], bringing down the time from $2^{0.265n+o(n)}$ to $2^{0.257n+o(n)}$.

Our algorithm belongs to sieving algorithms [NV08], a class of heuristic algorithms that start with long lattice vectors and iteratively construct shorter ones by summing lattice vectors pairwise until it finds one short enough. The main idea behind our algorithm is to replace Grover's search in Laarhoven's algorithm with a quantum walk to search for pairwise reducing vectors. A quantum walk algorithm finds a subset in a larger set satisfying a wanted property. Typically in our case, we wanted to find a subset that contains two vectors that reduce together. It was not *a priori* clear how to adapt the algorithm to integrate quantum walks as there are many ways of constructing them and most of them do not give speedups. We have used the MNRS framework [Mag+11] combined with a locality-sensitive filtering (LSF) technique [Bec+16] to find more efficiently pairs of close vectors, as they have higher chances of reducing.

Our result shows that filtering twice does have a benefit, contrary to what was believed previously. Moreover, our running time goes below the conditional lower bound that [KL21] stated for a restricted class of sieving algorithms. Our generalization of their framework has opened new horizons for more efficient sieving algorithms. We also show the best classical fits our framework, as well as the previous best quantum algorithm. We finally present two space-time tradeoffs. The first one is computed for fixed quantum memory, to make our algorithm flexible in the case the qubits are a limited resource, so the algorithm runs with less quantum memory and a higher time to compensate. Similarly, we present a tradeoff for fixed QRAM.

This result comes from joint work with André Chailloux and previously appeared in the proceedings of ASIACRYPT 2021 [CL21].

A new filtering technique and its application to solving SVP with low memory

Chapter 5 explores a variant of the sieve, the k-sieve, and gets improved tradeoffs. The k-sieve sums k points together to find shorter ones, and its memory requirements reduce when k increases. Reducing the memory would make the attack more materially practical, especially when it comes to quantum memory which is very limited for implementations.

Searching reducing k-tuples in a sieving step is a problem that reduces to the configuration problem, *i.e.* searching k vectors satisfying given constraints on their pairwise scalar products. The choice of the target configuration impacts time and memory. Taking a balanced configuration with all scalar products equal minimizes the memory. On the other hand, searching for unbalanced k-tuples slightly increases the memory to in counterpart reduce the running time. It is another tool that helped us to get better time-memory tradeoffs.

We introduce a new filtering technique tailored for the k-sieve. It extends the construction of random product codes (RPC) of [Bec+16]. A k-RPC is an efficiently decodable code such that each codeword belongs to a k-tuple that sums to the null vector. We use it to describe a new framework for the k-sieve

and get improved tradeoffs for classical and quantum 3-sieve and 4-sieve algorithms. It looks for k-tuples of lattice points each such that their sum is shorter. We first perform a filtering step: we use a k-RPC to construct lists $L_1, \ldots, L_k$ such that each L_i contains lattice points close to a codeword $\mathbf{A}_i$ in the k-RPC where $\mathbf{A}_1 + \cdots + \mathbf{A}_k = \vec{\mathbf{0}}$. This forms a tuple-filter $L_1 \times \cdots \times L_k$ in which the k-tuples of vectors are more likely to reduce than by taking k random vectors. We then use known algorithms on configuration search on these k-tuples of lists to find reducible ones, [HK17] for the classical setting and [Kir+19] for the quantum one. All our quantum algorithms require only a polynomial number of qubits. Here too, our results prove that the conditional optimality of [KL21] can be beaten.

This is based on joint work with André Chailloux and these results appeared in PQCrypto 2023 [CL23].

Quantum security analysis of Wave

Chapter 6 focuses on the cryptanalysis of the Wave code-based signature scheme. Its security relies on the hardness of forging a signature and distinguishing the secret key from a uniform. The best known attacks do this by tackling two instances of the Decoding Problem. We improve the message attack on Wave in the quantum setting that slightly reduces its claimed security respectively by 1, 3, and 5 bits for the three security levels in the NIST submission [Ban+23].

Our algorithm uses the Information Set Decoding (ISD) framework [Pra62; FS09], whose idea is to find many solutions to a simpler instance of the decoding problem, and then to check whether one of these solutions allows to find a whole solution of good weight to the DP instance that one wanted to solve. Our work proposes an improved way to compute the candidate sub-solutions. Our algorithm combines different approaches. It is based on Wagner's algorithm [SS81] with the idea of [Sen11] to search for a solution to the decoding problem considering exponentially many syndromes, known as the Decoding One Out of Many problem. We adapt this algorithm in the quantum setting in a way inspired by state-of-the-art [CDE21], and improve it by adding a smoothing technique of [Bri+20]. We also develop a complete time complexity analysis for the four best known attacks on Wave: message and forgery in both classical and quantum settings. For each of them, we provide explicit expressions in the function of the Wave parameters. So the claimed security level can easily be updated with new sets of parameters using our formulas.

This work resulted in a preprint [Loy23] and took part in a joint submission [Ban+23] to the NIST with Gustavo Banegas, Kévin Carrier, André Chailloux, Alain Couvreur, Thomas Debris-Alazard, Philippe Gaborit Pierre Karpman, Ruben Niederhagen, Nicolas Sendrier, Benjamin Smith, and Jean-Pierre Tillich.

1.3. Outline of the thesis.

- Chapter 1 is the introduction. You are here!
- Chapter 2 provides preliminaries on quantum information theory.
- Chapter 3 defines the Shortest Vector Problem (SVP) and introduces the best known method to solve it, called lattice sieving.
- Chapter 4 presents a new algorithm for SVP using quantum walks that improves the best asymptotic time. As a direct application, this reduces the theoretical quantum security of lattice-based schemes.
- Chapter 5 describes a new tailored filtering technique for the k-sieve, and its application provides better time-memory tradeoffs.
- Chapter 6 estimates the security of Wave, a code-based digital signature scheme.
- Chapter 7 concludes with an overview of the results and the possible directions for future research.

2. Quantum computing

In this chapter, we review concepts from quantum information theory relevant to this thesis.

2.1. Quantum circuit model

Before designing complex quantum algorithms, we need to define the lowest-level operations: the quantum circuits. All the circuits presented here are said "agnostic", meaning that they do not depend on the architecture of the computer on which they are implemented.

Classical computing stores and processes information coded in the form of states of electronic components. Similarly, we speak about quantum computing when data are coded by states of quantum particles. Then classical physics no longer stands and quantum physics theory takes over. That allows us to exploit the particular properties of these particles to do operations that were proven impossible with classical computing, such as constructing superposed or entangled states. And this can speed up calculations.

2.1.1. Qubits

In classical computing, the smallest information processing unit is called a bit. It has a value of 0 or 1 and can change from one state to another, but its state is always fixed and unique at a precise instant. Whereas for a quantum particle, its state can be a quantum superposition of 0 and 1 at the same time. Physically, these values can be coded by atoms, photons, or nuclear spins for example. If we measure it, there are probabilities of obtaining 0 or 1, but is it impossible to predict in advance which result we will find. Once it is measured, the particle loses its quantum behavior and stays in the state we measured. We say that the particle collapses. However, probabilities alone are not enough to explain some behaviors of a quantum particle like the interference effect. Thus, amplitudes are used to characterize its state. Physically this notion is related to the wave function of the particle.

Let $|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ be two basis vectors representing the two pure states.

Definition 2.1 (Qubit)**.** A qubit at a quantum state $|\psi\rangle$ can be described by its amplitudes $(\alpha, \beta) \in \mathbb{C}^2$:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix},$$

where $|\alpha|^2$ (resp. $|\beta|^2$) is the probability of finding the state 0 (resp. 1) when we perform a measurement. To make sense we then must have $|\alpha|^2 + |\beta|^2 = 1$.

If we manipulate a register with n qubits, we can consider a basis $(|0...00\rangle, |0...01\rangle, ..., |1...11\rangle)$. Let us rewrite it $(|0\rangle, |1\rangle, ..., |2^n - 1\rangle)$ for simplicity, where the first notation is the binary writing of the second one. A register with n qubits can be in any superposition in this form:

$$|\psi\rangle = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle, \qquad \text{where} \sum_{j=0}^{2^n-1} |\alpha_j|^2 = 1.$$

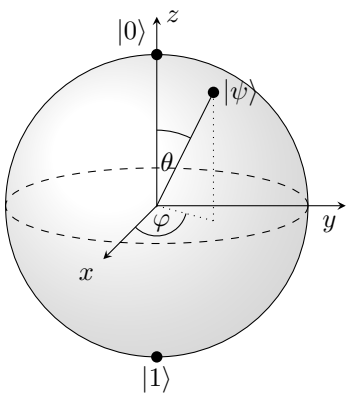


Figure 2.1: Geometrical representation of the qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ on the Bloch sphere, where $\alpha = \cos\left(\frac{\theta}{2}\right)$ and $\beta = e^{i\varphi}\sin\left(\frac{\theta}{2}\right)$. The probability to measure 0 (or 1) only depends on angle θ. Angle φ is the phase of the state. Bloch sphere is a more precise representation than Schrödinger's cat popular image, which does not represent the phase φ (and requires the strong hypothesis that cats respect quantum laws).

2.1.2. Quantum gates

Once we have data stored in qubits, we would like to process it. Quantum circuits generalize the idea of classical circuits, where the AND, OR, and NOT gates are replaced by quantum gates. As qubits are elements of $\mathbb{C}^2$, we can represent a quantum gate by a unitary matrix, *i.e.* a matrix U such that $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ is the conjugate transpose of U. We can transform a state $|\psi\rangle$ into $|\psi'\rangle = U|\psi\rangle$ by applying the gate of unitary U. Notice that for one qubit (a 2-dimensional complex vector), the dimension of its matrix is 2×2. Here are a few usually used quantum gates.

Pauli matrices.

$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ Identity gate.

$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ Bitflip gate, or quantum NOT-gate: adds π to angle θ (see figure 2.1).

$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$

$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ Phaseflip gate: adds π to angle φ (see figure 2.1).

These are the single-qubit Pauli gates. The n-qubits Pauli matrices are obtained from these four matrices by taking a tensor product of n of these matrices. We define this operation just below.

Hadamard gate.

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Applying Hadamard's gate H on the state $|0\rangle$ gives $|+\rangle := H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Measuring this state gives an equal probability of observing $|0\rangle$ or $|1\rangle$, and the same for $|1\rangle$. Notice that $H^2 = I$.

Phase rotation. This single-qubit gate adds an angle ϕ to the phase (see figure 2.1).

$$R_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$$

One can construct two-qubit gates from single-qubit ones, and the dedicated matrix operation is as follows.

Definition 2.2 (Tensor product or Kronecker product)**.** The tensor product of matrices $A = (a_{i,j}) \in \mathbb{C}^{n\times n'}$ and $B = (b_{i,j}) \in \mathbb{C}^{m\times m'}$ is

$$A \otimes B := \begin{pmatrix} a_{1,1}B & a_{1,2}B & \cdots & a_{1,n'}B \\ a_{2,1}B & a_{2,2}B & \cdots & a_{2,n'}B \\ \vdots & \vdots & & \vdots \\ a_{n,1}B & a_{n,2}B & \cdots & a_{n,n'}B \end{pmatrix} \in \mathbb{C}^{(n\cdot m)\times(n'\cdot m')}.$$

As an example, the tensor product of $A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$ and $B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}$ is

$$A \otimes B = \begin{pmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{pmatrix}.$$

For two single-qubit gates A and B, their tensor product $A \otimes B$ represents a two-qubit gate of dimension 4×4. Indeed, if we consider two qubits $|\psi\rangle$ and $|\psi'\rangle$, applying A on $|\psi\rangle$ and B on $|\psi'\rangle$ is equivalent to applying $A \otimes B$ on $|\psi\rangle \otimes |\psi'\rangle$, *i.e.* $(A|\psi\rangle)(B|\psi'\rangle) = (A \otimes B)|\psi\rangle|\psi'\rangle)$.

Controlled-NOT gate. This gate operates on two qubits. The first one is called the controlled qubit. If the first one, called the controlled qubit, has the value 1 then the second qubit is flipped.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The sub-matrix $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ on the bottom right can be replaced by any unitary matrix. It leads to only applying a gate under the condition that the first qubit is $|1\rangle$, which implements the quantum version of the $IF...THEN$ operation.

Theorem 2.3 (Solovay-Kitaev theorem)**.** *[NC00, Appendix 3] With H, $CNOT$ and $R_{\pi/4}$, it is possible to construct an approximation of any possible quantum circuit. For an approximation up to an error ϵ, it only requires a number* $\text{polylog}(1/\epsilon)$ *of these gates.*

This theorem will allow us to measure the complexity of n-qubit gates.

Complexity evaluation. We consider here quantum circuits with 1 or 2-qubit gate, without any locality constraint, meaning that we can apply a 2-qubit gate from a universal set of gates to any pair of qubits in constant time. We only study asymptotic running time here so we are not interested in the choice of this universal gate set, as they are all essentially equivalent from the Solovay-Kitaev theorem.

We use the textbook gate model where the **time complexity** of a quantum circuit is the number of gates used. A circuit is said **efficient** if it can be implemented with a number of gates polynomial in the size of the input. The width of a circuit is the number of qubits it operates on, including the ancilla qubits. This quantity is important as it represents the number of qubits that have to be manipulated simultaneously and coherently. We will call this quantity **quantum memory**.

Another quantity of interest is the depth of the circuit, which is important as a high-depth quantum circuit will be much harder to achieve because of decoherence. Moreover, while the number of gates is a good measure of time when each gate has to be computed separately, depth is a good measure of time when we can perform many gates at the same time, typically when we have multiple quantum processors.

When we will know much more precisely what quantum architectures look like, it will be possible to make these models more precise and replace the gate model with something more adequate. The gate model is still the most widely used in the scientific community and is very practical to compare different algorithms. We will use the gate model as our main model for computing quantum times but we will also include other interesting figures of merit, such as Quantum Random Access Memory, that we will develop in Section 2.2.

One of the properties inherent to quantum particles is **quantum superposition**. We saw that a Hadamard gate maps a state $|0\rangle$ to a quantum superposition $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. The Quantum Fourier Transform generalizes it by constructing a uniform quantum superposition over n qubits.

Definition 2.4 (Quantum Fourier Transform)**.** Let $|k\rangle$ be a register of n qubits, and $N = 2^n$. The quantum Fourier transform (QFT) is the following operation

$$QFT_N|k\rangle := \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \left(e^{2i\pi/N}\right)^{jk} |j\rangle.$$

QFT_N is a linear function. In its matrix form, denoting $\omega := e^{2i\pi/N}$, it can be written

$$QFT_N = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}.$$

$QFT_N|0^n\rangle = H^{\otimes n}|0^n\rangle$ is the uniform quantum superposition over all states on n qubits. Note that for $N = 2$ we recover $QFT_2 = H$.

Theorem 2.5. *[Sho94] The QFT_{2^n} can be exactly implemented with $\mathcal{O}(n^2)$ gates H and controlled-$R_{2\pi/2^m}$ for $1 \leqslant m \leqslant n$, or approximately implemented with $\mathcal{O}(n \cdot \log(n))$ of these gates.*

Another important aspect in quantum computing is **entanglement**. When two particles are entangled, if we measure one then the other is immediately impacted. For example, consider the 2-qubit register $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$, that can be obtained by applying the quantum circuit of Figure 2.2. If we measure any of the two qubits, we get the same probability of measuring 0 or 1. But if we then measure the other one, we will necessarily get the same measurement.

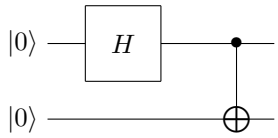


Figure 2.2: Entanglement circuit. The left gate is the Hadamard H, and the right two-qubit is a CNOT-gate, where the above qubit is the controlled one.

Query to a function. For a function f that we are able to compute classically or quantumly, the oracle to f is the following operation:

$$O_f : |a\rangle|0^n\rangle \to |a\rangle|f(a)\rangle.$$

From a classical circuit that computes f with T gates, O_f can be constructed with $\mathcal{O}(T)$ quantum gates. Note that the oracle entangles the two registers. Indeed, if we measured the second one and obtain the

state $|f(a_0)\rangle$ for some a_0, then the first one will transform to a quantum superposition of the a's such that $f(a) = f(a_0)$.

2.1.3. Shor's algorithm

A famous algorithm in the quantum circuit model is Shor's algorithm [Sho94], which factorizes a given integer N. We do not enter into the computational details, but here is a high-level idea of the algorithm. It considers a periodic function $f(i) = x^i \mod N$ for a randomly chosen x. We denote $n := \lceil \log_2(n) \rceil$, QFT_N the quantum Fourier transform and O_f the query oracle to function f. Then it applies the following quantum circuit, whose measurement provides a value that can be used to recover the period of f with good probability.

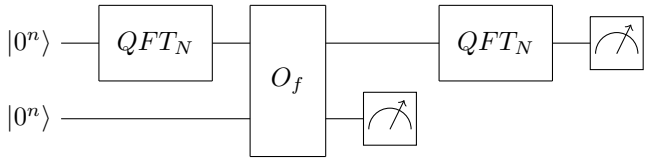


Figure 2.3: Quantum circuit in Shor's algorithm. The right-most boxes represent the measurements.

Knowing r the period of f, we have by definition $f(r) = f(0)$, *i.e.* $x^r = 1 \mod N$. If x is well chosen, this allows to write $(x^{r/2}+1)(x^{r/2}-1) = 0 \mod N$, so $(x^{r/2}+1)(x^{r/2}-1) = k \cdot N$ for some k. Then, we can hope that one of these $(x^{r/2} \pm 1)$ has a non-trivial common factor with N. Otherwise, we choose another random x until we succeed, which happens with a good probability.

Complexity. The time complexity of this circuit is the number of its elementary gates. By Theorem 2.5, we know that the Quantum Fourier Transform can be implemented with $\mathcal{O}(n \cdot \log(n))$ elementary gates. Evaluating the function $f : i \mapsto x^i \mod N$ can be done in efficient time ($\mathcal{O}(n^2 \log(n))$, then the call to the query oracle O_f is so. The measurements take time 1. Then, it solves the factorization problem in polynomial time. For comparison, the best (known) classical factoring algorithm [BJP06] runs in time of about $2^{\sqrt[3]{\log(N)}}$.

Implications on cryptography. The consequences of this algorithm are daunting. If we can find the periodicity of a function, then we can easily solve the problems of factorization and discrete logarithm, which are precisely the problems on which the security of the worldwide used RSA [RSA77], elliptic curves encryption [NIS91] and Diffie-Hellman key exchange [DH76] rely. To our knowledge to this day, in the classical model, no algorithm does it in polynomial time. While here in the quantum circuit model, Shor's algorithm solves it in a polynomial time, completely breaking its security. This is the reason why we must update our security systems by choosing problems resistant to quantum attacks. We will present in Chapters 3 and 6 two of the structures that offer problems believed to be hard, namely lattices and codes.

2.2. Quantum random access memory

Quantum computing relies on physical and technical advances to build a usable quantum computer. Since we cannot predict how far these advances will go, there exist several computing models.

QRACM

The Quantum Random Access Memory is an operation added to the quantum circuit model. We denote by QRACM the quantum-accessible classical memory. Considering N classical registers $x_0, ..., x_{N-1} \in \{0,1\}$ stored in memory, then a QRACM operation consists of applying the following unitary

$$O_x : |i\rangle|b\rangle \to |i\rangle|x_i \oplus b\rangle.$$

In the QRACM model, the above unitary is considered to be constructed efficiently. In particular, we assume that given list L there exists an efficient quantum circuit for $\frac{1}{\sqrt{|L|}}\sum_i |i\rangle|0\rangle \to \frac{1}{\sqrt{|L|}}\sum_i |i\rangle|L[i]\rangle$. With a QRACM access to L, this can be done by applying Hadamard gates to state $|0\rangle$ to create a superposition over all indices, and then by querying $L[i]$ for each i in the superposition.

Definition 2.6 (Quantum superposition of a list)**.** Given a list L, we call the quantum superposition of L the state $|\psi_L\rangle := \frac{1}{\sqrt{|L|}}\sum_{x\in L} |\mathrm{ind}_L(x)\rangle|x\rangle$, where $\mathrm{ind}_L(x)$ denotes the index of the element x in the list L. In the QRAM model, if L is classically stored and quantumly accessible then there exists an efficient quantum algorithm that constructs the state $|\psi_L\rangle$.

QRAQM

The Quantum Random Access to Quantum Memory (QRAQM) authorizes access to data in superposition. Assume that the quantum circuit holds qubits registers $x_0, ..., x_{N-1}$. A QRAQM operator does the following:

$$O_x : \left(\bigotimes_{j=0}^{N-1} |x_j\rangle\right)|i\rangle|0\rangle \to \left(\bigotimes_{j=0}^{N-1} |x_j\rangle\right)|i\rangle|x_i\rangle$$

Computational models

There are several computing models, here ranked from the most realistic to the most futuristic:

- Classical
- Quantum circuit model (and low qubit model: only use $\log(n)$ or $\mathrm{poly}(n)$ quantum memory)
- Quantum circuit model with efficient QRACM
- Quantum circuit model with efficient QRACM and QRAQM

It is very premature to know whether QRAM operations will be efficiently available one day for quantum computers. This would definitely require a major hardware breakthrough, but so would quantum computing in general. Some proposals for efficiently building QRAM operations exists, such as [GLM08], even though its robustness has been challenged in [Aru+15]. To this day it is still subject to controversy [GR04; Ber09; JR23] and research is still going on [All+23].

This uncertainty limits future possible uses we can predict. For concrete applications in business, it makes sense to avoid operations that we may not be able to implement. However, the logic of cryptanalysis is reversed. For the aim of security, we must consider that the adversaries have access to the best possible resources, in order to protect ourselves from the worst-case scenario by choosing the most robust security. It is more reasonable to be pessimistic when it comes to security, and to protect oneself too much than not enough, even if it means adjusting security settings afterwards. Indeed, in the case QRAM exists one day and if the settings of used cryptosystems neglected to take it into account, then encrypted files from before will be decryptable. It is not a risk that one can tolerate.

Even in case QRAM never sees the light of day, this computing model remains interesting to explore. If we find a quantum algorithm that efficiently solves a cryptographic problem, we may wonder if there exists one in classical. De-quantization has already happened in the past. There could also be "de-QRAMisation" of some quantum algorithms [JS20]. Furthermore, building quantum algorithms in this computing model makes it possible to better understand the structure of objects that we manipulate (lattices, codes, etc.). So all the work within the QRAM model will not be lost even in that situation.

2.3. Search algorithms

2.3.1. Grover's algorithm

Introduced in 1996, Grover's algorithm [Gro96] was presented as searching for a solution in a database. For one with N entries, classically, it is impossible to get a better complexity than $\Theta(N)$ queries, by examining each entry one by one until we find a solution. The quantum Grover algorithm solves this search problem in $\mathcal{O}(\sqrt{N/t})$ queries and with $\mathcal{O}(\sqrt{N}\log(N))$ other gates.

Let n be an integer and $N = 2^n$. We are given arbitrary $x_1, ..., x_N \in \{0,1\}^n$. The goal is to find an i such that $x_i = 1$, and to output "no solution" if there is no such i. We assume we know the number of such solutions i, denoted t, or at least an approximation of t. This problem was first presented as a search in an unordered database. In the original article, Grover fixed $t = 1$, which has been generalized. The idea of the algorithm is to separate "good" ($x_i = 1$) and "bad" ($x_i = 0$) indices i, and increase step by step the amplitudes of the "good" state. The action of the algorithm can be understood thanks to geometric arguments. First, we set:

$$\theta := \operatorname{asin}\left(\sqrt{\frac{t}{N}}\right) = \operatorname{acos}\left(\sqrt{\frac{N-t}{N}}\right)$$

$$|G\rangle := \frac{1}{\sqrt{t}} \sum_{\substack{i\in[N] \\ x_i=1}} |i\rangle \quad \text{and} \quad |B\rangle := \frac{1}{\sqrt{N-t}} \sum_{\substack{i\in[N] \\ x_i=0}} |i\rangle$$

$$|U\rangle := \frac{1}{\sqrt{N}} \sum_{i\in[N]} |i\rangle = \sqrt{\frac{t}{N}}\,|G\rangle + \sqrt{\frac{N-t}{N}}\,|B\rangle$$

$|U\rangle$ is the uniform state over all indices. $|G\rangle$ and $|B\rangle$ stand respectively for the "good" and "bad" states, depending of the x_i's values. We consider the 2-dimensional space induced by $|G\rangle$ and $|B\rangle$. It will be the basis on which we will take measurements. We can represent this on a circle of radius 1.

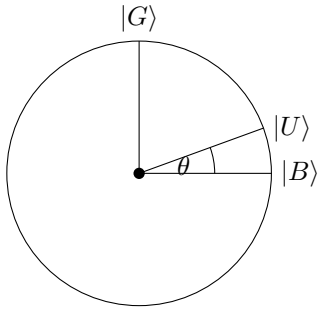


Figure 2.4: Geometrical representation of the initial state of the N-qubit register.

Geometrically, θ represents the angle between the states $|B\rangle$ and $|U\rangle$. We know that $\frac{t}{N}$ is the probability of measuring a good solution, so $\sqrt{t/N}$ is the amplitude of the "good" state $|G\rangle$ at the beginning of the algorithm. Thus we have $\theta = \operatorname{asin}(\sqrt{t/N})$. So, we can write the following relation:

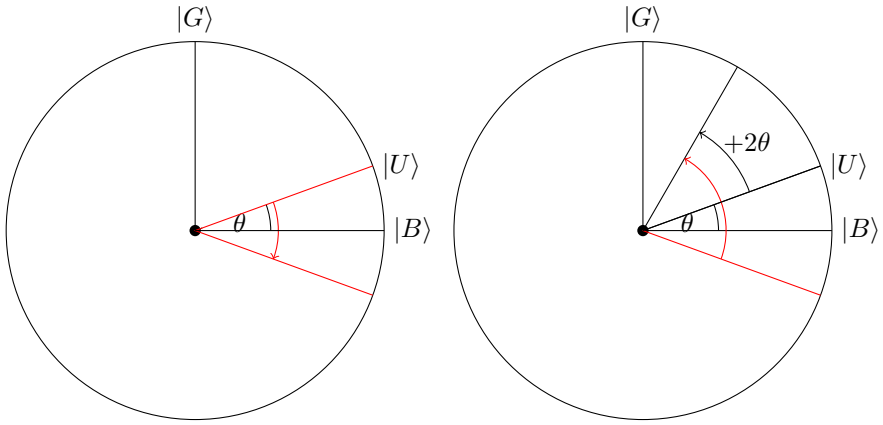
$$\cos(\theta)|B\rangle + \sin(\theta)|G\rangle = |U\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle.$$

Grover's algorithm will require Hadamard gates H, and $R|i\rangle = \begin{cases} -|i\rangle & \text{if } |i\rangle \neq |0^n\rangle \\ |0^n\rangle & \text{else.} \end{cases}$ which is realizable in $\mathcal{O}(n)$ elementary gates. It also requires an efficient query oracle $O_x|i\rangle = (-1)^{x_i}|i\rangle$. Notice that if this oracle uses RAM, then its use in Grover's procedure will require quantumly accessible x_i for the QRAM operations.

Lemma 2.7. *O_x acts as a reflection through $|B\rangle$ in the plane Span$\{|B\rangle, |G\rangle\}$, and $H^{\otimes n}RH^{\otimes n}$ as a reflection through $|U\rangle$.*

Proof. $O_x|i\rangle = (-1)^{x_i}|i\rangle$ does not change the state $|B\rangle$ if applied on, because we have in this case $x_i = 0$. And for all other states $|i\rangle$, it changes its sign. So by definition, it is a reflection through $|B\rangle$. Then, $H^{\otimes n}RH^{\otimes n} = H^{\otimes n}\Big(2|0^n\rangle\langle 0^n| - I\Big)H^{\otimes n} = 2|U\rangle\langle U| - I$ which is a reflection through $|U\rangle$. □

A Grover step is $\mathcal{G} = H^{\otimes n}RH^{\otimes n}O_x$, which corresponds to a reflection through $|B\rangle$ followed by a reflection through $|U\rangle$. The algorithm starts with $|U\rangle$ and at each iteration of $\mathcal{G}$, we move forward by the angle 2θ. Each application of $\mathcal{G}$ performs only one query to the oracle O_x.



(a) O_x, reflection through $|B\rangle$. (b) $H^{\otimes n}RH^{\otimes n}$, reflection through $|U\rangle$.

Now we have to choose the number k, the number of iterations of $\mathcal{G}$. By the above, the probability of measuring a good solution (state $|G\rangle$) after k applications of $\mathcal{G}$ is

$$\Pr(k) = \sin\Big((2k+1)\theta\Big)^2.$$

If we choose k too low or even too high, we see that the probability is not optimal. To maximize this probability, we want $(2k+1)\theta \approx \frac{\pi}{2}$, so we set $k \simeq \frac{\pi}{4\theta} - \frac{1}{2}$.

Algorithm 1 Grover's search [Gro96]

Require: $x_1, \ldots, x_N$; the number t (or its approximation) of solutions $x_i = 1$
Ensure: $i \in [N]$ such that $x_i = 1$, or "no solution"
  Initialize a register $|0^n\rangle$.
  Apply $H^{\otimes n}$ on the whole register to get the state $|U\rangle := H^{\otimes n}|0^n\rangle$
  Set $\theta := \text{asin}(\sqrt{t/N})$
  for $k \approx \frac{\pi}{4\theta} - \frac{1}{2}$ iterations **do** ▷ nearest integer approximation
    Apply O_x ▷ Reflection through $|B\rangle$
    Apply $H^{\otimes n}RH^{\otimes n}$ ▷ Reflection through $|U\rangle$
  Measure and check if the resulting i is a solution.

Theorem 2.8 (Grover's search [Gro96; Boy+98])**.** *Let $N = 2^n$. We are given $x_1, ..., x_N \in \{0, 1\}$ and an efficiently implementable unitary $O_x : |i\rangle|b\rangle \to |0\rangle|b \oplus x_i\rangle$, such that there are t solutions $i \in [N]$ that verify $x_i = 1$. Grover's quantum algorithm returns a solution i with probability greater than $1/2$ using $\mathcal{O}(\sqrt{|L|/t})$ calls to O_f. If t is unknown, it can be estimated and the running time is $\widetilde{\mathcal{O}}(\sqrt{|L|/t})$.*

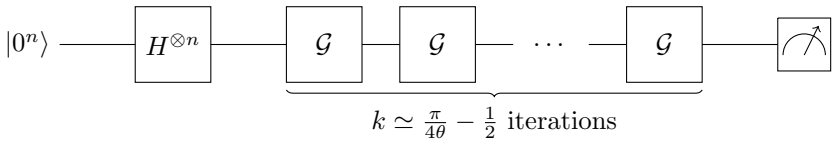


Figure 2.6: Quantum circuit of Grover's algorithm where $\mathcal{G} = (H^{\otimes n} R H^{\otimes n}) O_x$. This representation is equivalent to the pseudocode of Algorithm 1.

Grover's search can find an element $\boldsymbol{e}_i$ satisfying a certain wanted property within a given quantumly accessible list $L = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_N\}$. The description of the check function $f : L \mapsto \{0, 1\}$ encapsulates the definition of what we call a "solution". We then fix that $\forall i \in [N], x_i := f(\boldsymbol{e}_i)$. Therefore applying Grover returns the index i of one of the solutions $\boldsymbol{e}_i \in L$ such that $f(\boldsymbol{e}_i) = 1$; if such a solution exists. Assuming the efficiency of QRAM operations, this takes time $\widetilde{\mathcal{O}}(\sqrt{|L|/t})$ where t is the number of solutions.

2.3.2. Amplitude Amplification

Amplitude Amplification [Bra+02] generalizes Grover's algorithm. We are given a check function $f : \mathbb{Z}^n \to \{0, 1\}$, and we suppose we have an algorithm $\mathcal{A}$ which returns a solution z such that $f(z) = 1$ with a success probability of p. The Amplitude Amplification returns a solution with a success probability $1 - \epsilon > 1/2$.

We assume we have the gates H, R, the oracle function $O_f|z\rangle = (-1)^{f(z)}|z\rangle$, and a unitary that implements $\mathcal{A}$.

Algorithm 2 Amplitude Amplification

Require: The probability p of success of the algorithm $\mathcal{A}$.
Ensure: $z \in \mathbb{Z}^n$ such that $f(z) = 1$, or "no solution"
  Initialize a register $|0^n\rangle$.
  Apply $H^{\otimes n}$ on the whole register to get the state $|U\rangle := H^{\otimes n}|0^n\rangle$
  for $\mathcal{O}(1/\sqrt{p})$ iterations **do**
    Apply O_f ▷ Reflection through $|B\rangle$
    Apply $\mathcal{A}R\mathcal{A}^{-1}$ ▷ Reflection through $|U\rangle$
  Measure and check if the resulting z is a solution.

Remark that Grover's algorithm takes $O_f = O_x$, $\mathcal{A} = H^{\otimes n}$, and $p = \frac{t}{N}$.

Theorem 2.9 (Amplitude amplification [Bra+02])**.** *Let $\mathcal{A}$ be an algorithm without measurements that finds a solution $z \in \mathbb{Z}_{2^n}$ such that $f(z) = 1$ with a success probability p. Amplitude amplification on $\mathcal{A}$ returns a solution with probability $1/2$ using $\mathcal{O}(1/\sqrt{p})$ queries to O_f.*

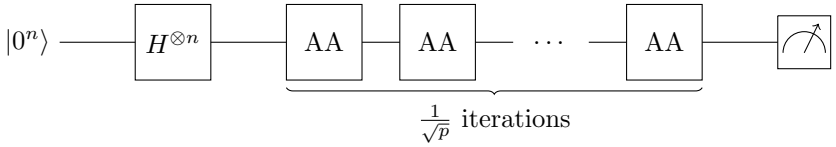


Figure 2.7: Quantum circuit of the amplitude amplification where $\mathrm{AA} = (\mathcal{A}R\mathcal{A}^{-1})O_f$.

Moreover, one can make the success probability of these algorithms go exponentially close to 1 by repeating them a constant number of times:

Proposition 2.10. *[Bra+02] Grover's algorithm can have a success probability of $1 - 2^{-\eta}$ with $\mathcal{O}(\eta\sqrt{|L|/t})$ queries to the function oracle O_f. Respectively, Amplitude Amplification with $\mathcal{O}(\eta/\sqrt{p})$ queries to O_f succeeds with probability $1 - 2^{-\eta}$.*

When we refer to Grover or quantum amplification, it will be implicit that we consider this version that erases the error.

2.3.3. Quantum Walks

A random walk is an algorithm that, given a set $\{x_1, \ldots, x_N\}$, returns a subset v of fixed size that satisfies a desired property if such v exists. In this case, v is said "marked". Quantum walks adapt classical random walks to the quantum setting. The difference is that in classical ones, we start from a vertex in a graph and move to a randomly chosen neighbor vertex until we find a marked one; while quantum walks construct a quantum superposition of all neighbor vertices. So we walk to all the neighbor vertices at the same time somehow, and this speeds up the search for solution.

The MNRS quantum walk framework [Mag+11].

The constraints to "walk" from a subset v to another are modeled by edges of an undirected graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. We do not allow self-loops which means that $\forall v \in V, (v, v) \notin E$ and as the graph is undirected there is $(v, u) \in E \Rightarrow (u, v) \in E$. There is a subset $M \subseteq V$ of marked vertices and the goal of the walk is to find one $v \in M$. Let $\epsilon = \frac{|M|}{|V|}$ be the fraction of marked vertices.

We denote δ the spectral gap of the graph G. For a regular graph, if $\lambda_1 > ... > \lambda_{|V|}$ are the eigenvalues of the normalized adjacency matrix of G, then $\delta = \lambda_1 - \max_{i=2...n} |\lambda_i|$. This value is non-negative but small ($\delta \ll 1$) and is correlated to the number of steps needed to reach a random vertex in the graph. Roughly speaking, no matter from which vertex we start, after $\Theta(\frac{1}{\delta})$ steps to a random neighbor, we will arrive at a random vertex in G. Let also $N(v) = \{u : (v, u) \in E\}$ be the set of neighbors of v. For any vertex v, we define $|p_v\rangle = \frac{1}{\sqrt{|N(v)|}} \sum_{u \in N(v)} |u\rangle$. We now define the following quantities:

- Setup step S (cost $\mathcal{S}$) constructs the quantum state

$$\frac{1}{\sqrt{|V|}} \sum_{v \in V} |v\rangle |p_v\rangle.$$

- Update step U (cost $\mathcal{U}$) consists in applying the unitary

$$U : |v\rangle|0\rangle \to |v\rangle|p_v\rangle.$$

  In order to compute the update cost $\mathcal{U}$, we consider time of classically going from one vertex v to one of its neighbors in $N(v)$. Then, we can use this procedure in quantum superposition to construct the unitary U.

- Check step C (cost $\mathcal{C}$) computes the check function $f : V \to \{0, 1\}$ where $f(v) = 1$ if $v \in M$ and 0 otherwise.

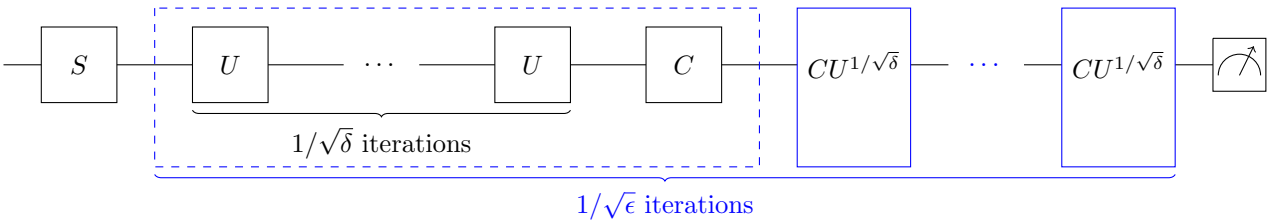


Figure 2.8: Circuit of a quantum walk of the MNRS framework.

We provide a very rough sketch of the quantum walk. The setup costs $\mathcal{S}$. The update operation U is applied $\lfloor 1/\sqrt{\delta} \rfloor$ times, then we check if the obtained vertex is marked using the check operation $\mathcal{C}$. This process $CU^{\lfloor 1/\sqrt{\delta} \rfloor}$ takes time $\lceil 1/\sqrt{\delta} \rceil \cdot \mathcal{U} + \mathcal{C}$ and has a probability of success ϵ. So we perform an Amplitude Amplification that has $\frac{1}{\sqrt{\epsilon}}$ iterations so that a measurement would give a marked vertex with high probability. Conserving the above notations, this leads to the following proposition.

Proposition 2.11. *[Mag+11] There exists a quantum walk algorithm that finds a marked element in time*

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

Reusable quantum walks.

In case we do not only search after one but K marked vertices, one can perform a quantum walk from scratch K times, which would cost

$$K \cdot \left(\mathcal{S} + \frac{1}{\sqrt{\epsilon}} \left(\frac{\mathcal{U}}{\sqrt{\delta}} + \mathcal{C} \right) \right).$$

Instead, [Bon+23] showed how to run the setup S only a single time, and then repeat $K/\sqrt{\epsilon}$ times the rest of the circuit. This restarts the walk from an already computed quantum superposition, and from there recovers a uniform random superposition used to search the next marked vertex. For some regimes, the cost to find K marked elements becomes

$$\mathcal{S} + \frac{K}{\sqrt{\epsilon}} \left(\frac{\mathcal{U}}{\sqrt{\delta}} + \mathcal{C} \right).$$

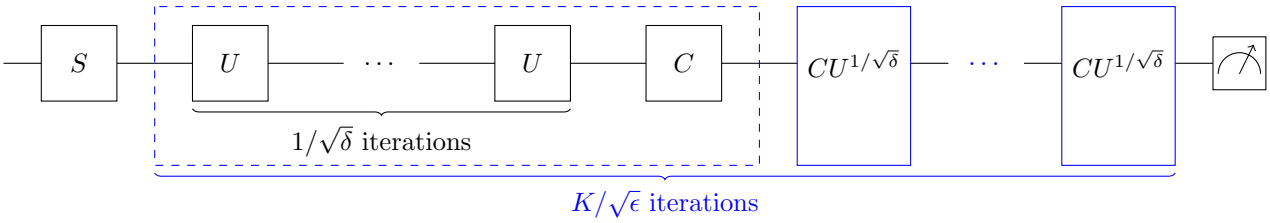


Figure 2.9: Quantum circuit of a reusable quantum walk.

These reusable quantum walks only apply in certain cases, such as when searching for a marked vertex can be expressed as a collision problem.

Collision problem and Johnson graph.

Here is a notorious problem with a large range of applications.

Definition 2.12 (Collision problem)**.** Given $n \in \mathbb{N}$, a set S and a function $f : S \to \{0,1\}^n$, find $x_1, \ldots, x_k \in S$ such that $f(x_1) = \cdots = f(x_k)$.

A standard way to solve a collision problem by a quantum walk would be to replace one element from a vertex $v \subseteq S$ with another one $x_{new} \in S \backslash \{v\}$, then check if v contains some $x_2, \ldots x_k$ that forms together a solution with x_{new}, and repeat until such a collision is found. A graph that encapsulates this process is the Johnson graph.

Definition 2.13 (Johnson graph)**.** We are given a set S of size n. For parameters n, r such that $r \leq n$, $J(n,r)$ denotes the Johnson graph. Each vertex v in this graph is a set of r distinct (unordered) elements in S as well as some additional data $D(v)$ that depends on the random walk we want to perform. Every possible vertex with this definition appears once in the graph. Two vertices $v = (x_1, \ldots, x_r, D(v))$ and $v' = (x'_1, \ldots, x'_r, D(v'))$ form an edge in $J(n,r)$ if and only if we can go from v to v' by removing exactly one value and then adding another one from set S.

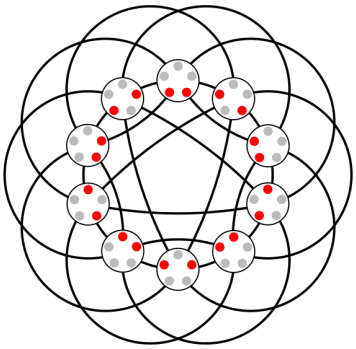


Figure 2.10: Johnson graph $J(5,2)$. There is a set S of $n = 5$ points. Each vertex contains $r = 2$ elements from S. And there is an edge between two vertices *iff.* they differ by exactly one element. (Author of the figure: Tilman Piesk)

Lemma 2.14. *[Wol23] The spectral gap of a Johnson graph $J(n,r)$ is $\delta = \frac{n}{r(n-r)}$. Then $\delta \approx \frac{1}{r}$ when $r \ll n$.*

Classical walks do not help to solve the collision problem faster. However, in the quantum model, Johnson graphs are very standard to perform quantum walks on. To improve the complexity of the algorithm, the basic idea is that when the update step replaces one element in the vertex v with a new x_{new} from S, it also checks if x_{new} belongs to a collision with some other elements in v. This might increase the update time, but makes the check step immediate, as it is already done during the update. The additional data $D(v)$ is computed during the setup and then updated at each step. It aims to reduce the time to find the elements in v that collide with x_{new}, if they exist. What it contains depends on what is relevant to the collision problem we want to solve. To get the optimal time of the walk, it is often a question of balancing the time to compute $D(v)$ and the time to check using $D(v)$ whether x_{new} forms a solution. We will go into more detail in Chapter 4, where we will present an example application.

A time analysis of quantum walks on the Johnson graph was done in [Amb07] when studying the element distinctness problem. There, Ambainis presented a quantum data structure that uses efficient QRAM that allows in particular insertion and deletion in $\mathcal{O}(\log(n))$ time where n is the database size while maintaining this database in quantum superposition. Another paper [Ber+13] on a quantum algorithm for the subset-sum problem using quantum walks also presents a detailed analysis of a quantum data structure based on radix trees to perform efficient insertion and deletion in quantum superposition. All of these data structures require as many QRAM registers as the number of registers to store the whole database and this running time holds only in the QRAM model.

3. Lattice sieving

Definition 3.1 (Lattice)**.** Given a basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^m$ of linearly independent vectors, the lattice $\mathcal{L} \subset \mathbb{R}^m$ generated by basis $\mathbf{B}$ is the set of all integer linear combinations of vectors of $\mathbf{B}$:

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} \lambda_i \boldsymbol{b}_i,\ \lambda_i \in \mathbb{Z} \right\}$$

A lattice is said of full-rank if $n = m$. In the following, we will only consider full rank lattices for simplicity. The determinant of $\mathcal{L}$ is $\det \mathcal{L} = |\det \mathbf{B}|$. Notice that the basis $\mathbf{B}$ is not unique, and one can give another basis of the same lattice by applying an arbitrary unimodular transformation.

3.1. Lattice-based cryptography

3.1.1. Lattice problems

For a lattice $\mathcal{L}$, we denote by convention $\lambda_1(\mathcal{L})$ the length of *a* shortest non-zero vector of $\mathcal{L}$. Notice that by definition, lattices are symmetric to the point $\vec{\mathbf{0}}$. So there are always at least two vectors of length $\lambda_1(\mathcal{L})$, and this is why we say *a* shortest vector and not *the*.

Definition 3.2 (Exact-SVP − Shortest Vector Problem)**.** Given a lattice $\mathcal{L}$, find a shortest non-zero vector $\vec{\boldsymbol{x}} \in \mathcal{L}$ *i.e.* of norm $\|\vec{\boldsymbol{x}}\| = \lambda_1(\mathcal{L})$.

Definition 3.3 (Approximate γ-SVP)**.** Given a lattice $\mathcal{L}$ and $\gamma > 1$, find a non-zero vector $\vec{\boldsymbol{x}} \in \mathcal{L}$ such that $\|\vec{\boldsymbol{x}}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$.

The bigger the approximation factor γ is, the easier γ-SVP becomes. For $\gamma = 1$, γ-SVP and SVP are the same problem. For cryptanalytic uses, solving γ-SVP for small γ and not necessarily a shortest is sufficient [Alb+19]. One can apply Minkowski theorem to get an upper bound $\lambda_1(\mathcal{L}) \leq \sqrt{n} \cdot |\det \mathcal{L}|^{1/n}$, or use the estimation of the length of a shortest vector given by the SVP challenge [Sch+]:

$$\lambda_1(\mathcal{L}) \approx 1.05 \cdot \frac{\Gamma(n/2+1)^{1/n}}{\sqrt{\pi}} \cdot (\det \mathcal{L})^{1/n}$$

where $\Gamma(z) := \int_0^{+\infty} t^{z-1} e^{-t} dt$ is the extension of the factorial function.

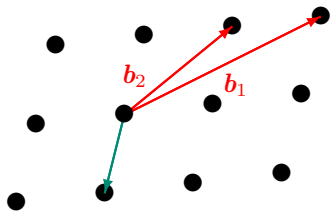


Figure 3.1: Given $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ two basis vectors that generate a lattice, the Shortest Vector Problem asks to recover the vector in green.

The Shortest Vector Problem is a central problem in complexity theory and the foundation of many cryptographic constructions. The average-case to worst-case reductions [Ajt99; Reg09] ensure cryptographic

schemes relying on SVP to be secure as long as there exists a lattice where finding a short vector is hard. However, in practice, lattice-based cryptographic schemes are not directly based on SVP but on the problems that follow.

Definition 3.4 (Closest Vector Problem (CVP))**.** Given a lattice $\mathcal{L}$ and a target vector $\vec{\boldsymbol{t}} \in \mathbb{R}^n$, find the vector $\vec{\boldsymbol{x}} \in \mathcal{L}$ the closest to $\vec{\boldsymbol{t}}$.

Definition 3.5 (NTRU [HPS98])**.** Let $q \geq 2$, $\mathcal{R} := \mathbb{Z}/f(x)\mathbb{Z}[x]$ a polynomial ring, and $f, g \in \mathcal{R}$ be "short" with f invertible mod q. Given $h := f^{-1} \cdot g \mod q$, find f and g.

The NTRU problem is equivalent to solving the Shortest Vector Problem in the lattice $\mathcal{L}_{h,q} = \{(x, y) \in \mathcal{R}^2 \ : \ xh - y = 0 \mod q\}$.

Definition 3.6 (LWE − Learning With Errors [Reg05])**.** Let n, m, q be positive integers, χ be a probability distribution on $\mathbb{Z}$ (often a Gaussian) and $\boldsymbol{s}$ be a uniformly random vector in $\mathbb{Z}_q^n$. We are given m independent samples $(a_i, a_i \cdot s_i + e_i)$ where the a_i's are uniformly random on $\mathbb{Z}_q^n$ and the errors e_i follows distribution χ. Find the secret $\boldsymbol{s}$.

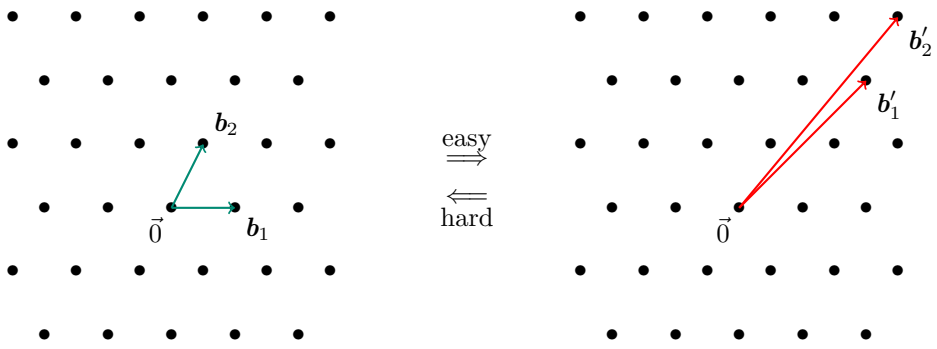
The LPN problem (Learning Parity with Noise), closely related to code-based problems, can be seen as an instance of LWE for modulus $q = 2$.

Definition 3.7 (SIS − Short Integer Solution [Ajt96a])**.** Given a matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$, find a vector $\vec{\boldsymbol{x}} \in [-\beta, \beta]^m \backslash \{0\}$ for some β such that $\mathbf{A}\vec{\boldsymbol{x}} = 0 \mod q$.

The SIS problem is the dual variant of LWE: solving SIS in a given lattice $\mathcal{L} \subset \mathbb{R}^n$ is exactly LWE in the dual lattice $\mathcal{L}^\perp := \{\vec{\boldsymbol{x}} \in \mathbb{R}^n \mid \langle \vec{\boldsymbol{x}} | \vec{\boldsymbol{y}} \rangle = 0, \forall \vec{\boldsymbol{y}} \in \mathcal{L}\}$.

3.1.2. Lattice-based schemes overview

The general idea of encryption and signature protocols was briefly introduced in Figures 1.1 and 1.2 in the introduction. To construct cryptographic schemes, we need a function that is efficiently computable and hard to reverse. Lattice structure provides such one-way functions. Indeed, one can generate a basis of short vectors that stands for the secret key, and then modify it to end up with a "bad" basis of very long vectors that constitutes the public key. Reversing this function, *i.e.* computing a good basis from a bad one, is equivalent to solving the NP-hard Shortest Vector Problem. [DW22] gave the conditions to construct schemes: any lattice makes it possible to design an identification scheme, a decodable lattice gives an encryption scheme and a Gaussian sampleable lattice an electronic signature scheme.



(a) Good basis of $\mathcal{L}$: $(\boldsymbol{b}_1, \boldsymbol{b}_2)$. Secret key.

(b) Bad basis of $\mathcal{L}$: $(\boldsymbol{b}'_1, \boldsymbol{b}'_2)$. Public key.

Figure 3.2: Bases $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ and $(\boldsymbol{b}'_1, \boldsymbol{b}'_2)$ generate the same lattice $\mathcal{L}$.

Ajtai [Ajt96b] showed how to generate hard lattice problems for cryptographic uses and [GGH97] used it to create the historically first lattice-based signature scheme, which relies on the Closest Vector Problem.

The NTRU scheme [HPS98; HPS01] followed, based on the eponymous problem. These original schemes were broken by [NR09]. [GPV08] introduced a "Hash-and-Sign" framework for signatures which provides provable security, and reduces to worst-case lattice problems. However, the megabytes-long signatures made them unusable in practice. [Lyu09] proposed a signature scheme based on the Fiat-Shamir framework [FS86] relying on the Shortest Vector problem in ideal lattices. This time, the performance of the scheme was close to being practical. On the encryption side, [Reg09] introduced a provably secure lattice-based scheme. [Gen09] proposed Fully Homomorphic Encryption, a form of encryption that enables computations on plaintexts without decryption. Most FHE schemes are based on either the LWE problem or its Ring or Modular variants for storing plaintexts. Structured lattices, such as ideal lattices, Ring, or Modular versions, are useful for reducing the size of the keys and the running time of the protocols in comparison to generic lattices. However, we will not develop on structured lattices.

As no efficient algorithm is known to solve lattice problems, lattice-based cryptography is believed to be quantum-safe. Then lattice-based cryptography gained a lot of visibility during the NIST call for post-quantum cryptography. Among the four submissions to be standardized, three are based on lattice problems. CRYSTALS-KYBER [Bos+18] is an encryption scheme relying on the modular version of the LWE/SIS problem. CRYSTALS-Dilithium [Duc+19] is an electronic signature scheme from the Fiat-Shamir [FS86] framework, and it is also based on MLWE-MSIS. And FALCON [Fou+18] is a "Hash-and-Sign" scheme relying on a variant of the NTRU problem.

In 2023, the NIST launched a new call for post-quantum signatures [NIS23]. Among the lattice-based submissions, HAWK relies on the lattice isomorphism problem, HuFu and Squirrels rely on the LWE/SIS problem, and EagleSign, HAETAE and Racoon rely on its modular version M-LWE/M-SIS.

3.1.3. Cryptanalysis

Lattice-based cryptography mostly relies on the hardness of LWE, its dual version SIS, and their structured Ring or Modular variants. Attacks fall into different categories: lattice reduction [LLL82; SE94], combinatorics [BKW03; Bud+20] or algebraic attacks [AG11]. In this work, we consider generic lattices, not depending on their specific structure, so we will focus on lattice reduction. To defeat a lattice-based cryptosystem, one can run the BKZ algorithm [SE94], which itself uses an SVP-solver as a subroutine. It returns a reduced basis of the lattice, that is enough to compute in polynomial time the solution to the LWE instance underlying the scheme. The hardness of the attack can be estimated by the complexity of solving the Shortest Vector Problem in a chosen cost model (See Section 2.2).

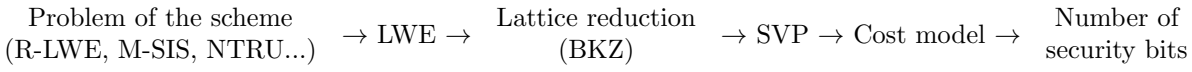


Figure 3.3: Cascade of problem reductions.

All the fastest known algorithms to solve SVP run in exponential time in n the lattice dimension − or even super-exponential in the case of enumeration [Kan83]. The current best enumeration algorithm [Alb+20] runs in time $n^{0.125n}$, and [ANS18] showed that quantum gives a quadratic gain. It only requires a polynomial memory. The other methods run in time and memory $2^{cn+o(n)}$ for some constant $c = \Theta(1)$. Among these algorithms there are sieving [KS01; NV08; MV10], Voronoi cell [MV10], and Gaussian sampling [Agg+15]. To this day, only sieving algorithms are competitive for large dimensions. Sieving is a class of heuristic algorithms, therefore the returned vector has no guarantee to be exactly the shortest, and the algorithm may not succeed on arbitrary lattices. But actually, an approximate solution to SVP suffices for cryptanalysis purposes [Alb+19], and sieving has largely proven its efficiency in practice as all the 20 records to the SVP challenge [Sch+] are due to sieving algorithms.

3.2. Sieving algorithms

A sieving algorithm starts with a list of long lattice vectors and successively applies sieving steps to diminish the norms of the vectors. After a few sieving steps, we hope to find a short vector. Such algorithms solve approximate SVP in time and space $2^{\Omega(n)}$, with n the dimension of the lattice. The NV-sieve, originally introduced in [NV08], constructs shorter vectors by summing vectors from the input list by pairs.

There also exists provable sieving algorithms [Agg+15; Agg+21], whose analysis does not rely on heuristics. But these algorithms run in practice much faster than the theoretical asymptotic time of their proven analyses. Heuristic algorithms in counterpart, despite that they rely on an approximation of reality, their analysis provides a good estimation of the running time of the attacks. This explains why the analysis of sieving algorithms requires a heuristic and unfortunately, why we cannot work without it. The relevance of the heuristic has been studied, and for now it gives good estimations of the running time.

3.2.1. The NV-sieve

Given a list of lattice vectors of norm at most R and a reducing factor $\gamma < 1$, a sieving step will return a list of lattice vectors of norm at most γR. To obtain these reduced vectors, the NV-sieve computes the difference of each pair of vectors in the input list and fills the output list with those that are of norm at most γR. Then, it iteratively builds lists of shorter lattice vectors by applying the sieve step. The first list of lattice vectors can be sampled with Klein's algorithm [Kle00] for example. As the norms of the list vectors reduce by a factor $\gamma < 1$ at each sieving step, the output list will hopefully contain a non-zero shortest lattice vector after a polynomial number of iterations of the NV-sieve step.

Algorithm 3 NV-sieve step [NV08]

Require: List L of N lattice vectors of norm at most R, a reducing factor $\gamma < 1$.
Ensure: List L_{out} of N lattice vectors of norm at most γR.
  for $\vec{x}_1 \in L$ **do**
    for $\vec{x}_2 \in L$ **do**
      if $\|\vec{x}_1 - \vec{x}_2\| \leq \gamma R$ **then** add $\vec{x}_1 - \vec{x}_2$ to L_{out}
  return L_{out}

Algorithm 4 Solve SVP by the sieving method

Require: basis $\mathbf{B}$ of a lattice $\mathcal{L}$, a reducing factor $\gamma < 1$.
Ensure: a short vector of $\mathcal{L}$ (probably)
  $L \leftarrow$ generate N lattice vectors using Klein's algorithm on basis $\mathbf{B}$
  while L does not contain a short vector **do**
    $L \leftarrow$ **Sieve-step**(L, γ)
  return $\min(L)$

Klein's algorithm initializes a list with long vectors and we denote their greatest norm R. After applying a sieving step only $\text{poly}(n)$ times, the output vectors are of norm at most $R \cdot \gamma^{\text{poly}(n)}$ with $\gamma < 1$, and this value is exponentially smaller than the initial norm. The list will then hopefully contain a short vector. The number of while loop iterations is polynomial in n so it does not affect the asymptotic time. We only need to estimate the running time of the sieving step.

We present here two simplifications of notation. First, we will only consider the case $R = 1$. Indeed, all the sieving algorithms we consider will be independent of R, and one can easily normalize the vectors. Also, in practice, we take $\gamma \approx 1$ (typically $\gamma = 1 - \frac{1}{\text{poly}(n)}$). We will fix $\gamma = 1$ to simplify the analysis of the algorithm.

The sieve step described in Algorithm 3 succeeds under the following heuristic.

Heuristic 3.8. *Lattice points of norm at most* 1 *are distributed uniformly at random on the sphere* $\mathcal{S}^{n-1} := \{\vec{\boldsymbol{x}} \in \mathbb{R}^n : \|\vec{\boldsymbol{x}}\| = 1\}$.

Actually, random uniform points in the ball $\{\vec{\boldsymbol{x}} \in \mathbb{R}^n : \|\vec{\boldsymbol{x}}\| \leq 1\}$ are with high probability very close to the border of the ball, so on the outer sphere $\mathcal{S}^{n-1}$. It is easier to intuit this phenomenon when one thinks in high dimensions. There is more volume in the area of the border of the sphere, called a "shell", and when we increase the dimension, the volume concentrates even more on the border. For example, we already see that the ball of dimension 3 has a shell whose ratio volume is wider than that in dimension 2. Thus, uniform vectors of norm at most 1 are with a high probability of norm very close to 1 for the high dimensions n used for cryptography. A simplification we can do is then to consider that uniform points are exactly lying on the sphere $\mathcal{S}^{n-1}$ of radius 1. The heuristic consists in assuming that lattice vectors are distributed as uniform points with this simplification, ignoring the structure of the lattice. The relevance of this heuristic has been studied in [NV08] and confirmed by experiments. It becomes invalid when the vectors become short, but in this case, we can assume we have solved SVP.

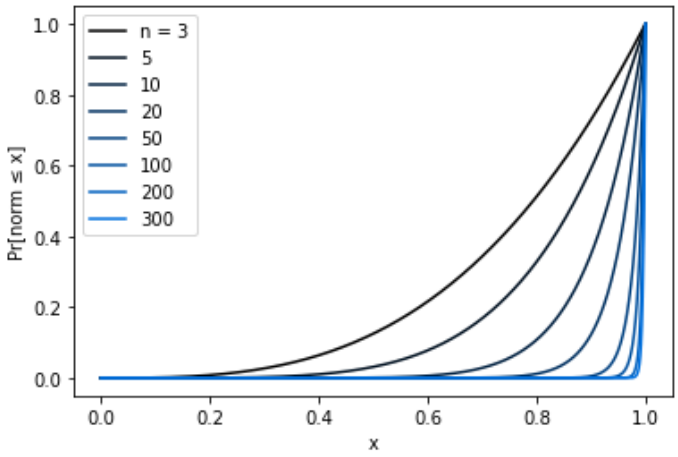


Figure 3.4: Distribution of norms of uniformly random vectors in the ball of radius 1 and dimension n. The function of density for the vector norm x of dimension n is $f_n(x) = \frac{VolumeBall(n,x)}{VolumeBall(n,1)} = x^n$, with $VolumeBall(n,x) = \frac{\pi^{n/2}x^n}{\Gamma(\frac{n}{2}+1)}$. For high dimensions, we can see the high concentration near the norm 1.

Complexity analysis of the NV-sieve

We introduce here a geometry notion that will be useful for the complexity analysis of sieving algorithms. Recall that $\mathcal{S}^{n-1} := \{\vec{\boldsymbol{x}} \in \mathbb{R}^n : \|\vec{\boldsymbol{x}}\| = 1\}$. The spherical cap of center $\vec{\boldsymbol{s}}$ and angle α is defined as follows

$$\mathcal{H}_{\vec{\boldsymbol{s}},\alpha} := \{\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1} \mid \theta(\vec{\boldsymbol{x}}, \vec{\boldsymbol{s}}) \leq \alpha\} = \{\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1} \mid \langle \vec{\boldsymbol{x}} | \vec{\boldsymbol{s}} \rangle \geq \cos(\alpha)\}.$$

Proposition 3.9 ([Bec+16], Lemma 2.1)**.** *For arbitrary angle* $\alpha \in (0, \pi/2)$ *and a vector* $\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}$*, the ratio of the volume of a spherical cap* $\mathcal{H}_{\vec{\boldsymbol{x}},\alpha}$ *to the volume of the sphere* $\mathcal{S}^{n-1}$ *is*

$$\mathcal{V}_n(\alpha) := \mathrm{poly}(n) \cdot \sin^n(\alpha).$$

In other words, for an arbitrary angle $\alpha \in (0, \pi/2)$, if we fix $\vec{\boldsymbol{s}} \in \mathcal{S}^{n-1}$ and consider a uniformly random vector $\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}$, then we have

$$\Pr_{\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}}[\langle \vec{\boldsymbol{x}} | \vec{\boldsymbol{s}} \rangle \geq \cos(\alpha)] = \Pr_{\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}}[\vec{\boldsymbol{x}} \in \mathcal{H}_{\vec{\boldsymbol{s}},\alpha}] = \mathcal{V}_n(\alpha).$$

The sieving step starts with a list L of lattice vectors, randomly distributed on the sphere $\mathcal{S}^{n-1}$ according to the heuristic 3.8. We first need to know how many vectors are required in input to end up with as many

vectors in output. Indeed, having to deal with too many vectors just wastes time, and on the contrary, taking too few vectors leads to quickly running out of vectors and not finding a short one. Therefore the size of the lists must be kept constant and minimal.

A pair of vectors $\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2 \in L$ yields a reduced vector with norm $\|\vec{\boldsymbol{x}}_1 - \vec{\boldsymbol{x}}_2\| \leq 1$ if and only if their angle satisfies $\theta(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \leq \frac{\pi}{3}$. Then, each random pair in L reduces with probability $\mathcal{V}_n(\pi/3)$ by Proposition 3.9. Since there are $\mathcal{O}(|L|^2)$ pairs of points in L, we have on average $|L|^2 \cdot \mathcal{V}_n(\pi/3)$ pairs in L that reduce. This quantity needs to be equal to $|L|$ in order to keep the input and output list sizes equal. So we set $|L| = 1/\mathcal{V}_n(\pi/3) = (4/3)^{n/2+o(n)} = 2^{0.208n+o(n)}$.

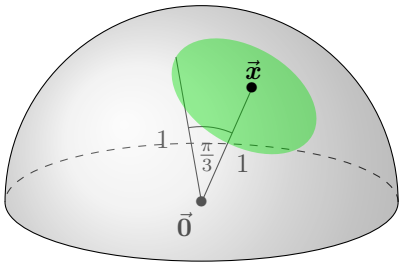


Figure 3.5: Subtracting any vector from the green area with $\vec{\boldsymbol{x}}$ yields a shorter vector.

Proposition 3.10. *[NV08] The NV-sieve heuristically solves SVP in time* $2^{0.415n+o(n)}$ *and space* $2^{0.208n+o(n)}$.

Proof. The NV-sieve step checks each pair in list L, so its time complexity is $|L|^2 = 2^{0.415n+o(n)}$, and we perform $\text{poly}(n)$ sieve steps. Its space complexity is the number of vectors we need to store at the same time, so the space complexity is $|L| = 2^{0.208n+o(n)}$.

□

Complexity analysis of the quantum NV-sieve

Algorithm 5 Quantum NV-sieve step

Require: List L of N lattice vectors of norm at most 1, a reducing factor $\gamma < 1$.
Ensure: List L_{out} of N lattice vectors of norm at most γ.
  for $\vec{\boldsymbol{x}}_1 \in L$ **do**
    $\vec{\boldsymbol{x}}_2 \leftarrow$ Grover on $L \backslash \{\vec{\boldsymbol{x}}_1\}$
    if $\|\vec{\boldsymbol{x}}_1 - \vec{\boldsymbol{x}}_2\| \leq \gamma$ **then** add $\vec{\boldsymbol{x}}_1 - \vec{\boldsymbol{x}}_2$ to L_{out}
  return L_{out}

Proposition 3.11. *The quantum NV-sieve heuristically solves SVP in time* $2^{0.311n+o(n)}$ *using classical memory in* $2^{0.208n+o(n)}$ *under the assumption of efficient QRACM.*

Proof. The quantum NV-sieve step replaces the exhaustive search for a reducing $\vec{\boldsymbol{x}}_2$ in L by a Grover's search in L. The time complexity is $|L|\sqrt{|L|} = 2^{0.311n+o(n)}$ and it requires a classical memory quantumly accessible of size $|L| = 2^{0.208n+o(n)}$. □

Sieving in practice. The Gauss-Sieve [MV10], another family of sieving algorithms, is more used than the NV-sieve in practice. Despite having no known bound on its time complexity, it usually performs faster than the implementations of the NV-sieve. The idea of the Gauss-Sieve is to start with a short list of vectors and then sample new lattice vectors. Vectors are reduced by pairs too and newly computed shorter vectors

replace the longer ones in the list. There have been several subexponential improvements, like the dimension for free [Duc17; DLW20] and progressive sieving [LM18]. All the top 20 current highest records [Sch+] for solving SVP have been reached thanks to sieving algorithms. To this day, the highest dimension where SVP has been solved is $d = 186$, while the order of the dimensions for lattices concretely used in schemes remains far superior. For example, the dimension of the lattice used in Dilithium [Duc+19, Table 2] is 475.

3.3. Locality Sensitive Filtering

A great improvement of the sieving algorithms is to use Neighbor Nearest Search (NNS) techniques. The NNS problem is: given a list L of vectors, preprocess L such that one can efficiently find the nearest vector in L to a target vector given later. Applied in the NV-sieve, the preprocessing step partitions the input list into several buckets of lattice points, with each bucket being associated with a hash function. During the querying step, when we search for each point $\vec{\boldsymbol{x}}$ in the list a reducing one, we search for a reducing one in the buckets where $\vec{\boldsymbol{x}}$ is inserted instead of looking at the whole list, which is much larger.

A method to solve NNS was locality-sensitive hashing (LSH) introduced in [IM98] and then improved in [Cha02; And+14; AR15; TT07]. It uses a hash function that has a high probability for two elements to collide if they are close, and a low one if they are far. More recently, [Bec+16] improved NNS for the Euclidean norm by introducing locality-sensitive filtering (LSF). The latter technique uses the structure of random product codes that allies both the randomness of the distribution and an efficient list-decoding algorithm, which provides a great improvement in lattice sieving.

3.3.1. Random Product Code and Hypercone Filters

Definition 3.12 (Random Product Code (RPC))**.** We assume $n = m \cdot b$, for $m = \widetilde{\mathcal{O}}(n)$ and a block size b. The vectors in $\mathbb{R}^n$ will be identified with tuples of m vectors in $\mathbb{R}^b$. A random product code $\mathfrak{C}$ of parameters (n, m, B) on subsets of $\mathbb{R}^n$ and of size B^m is defined as a code of the form $\mathfrak{C} = Q \cdot (\mathfrak{C}_1 \times \mathfrak{C}_2 \times \cdots \times \mathfrak{C}_m)$, where Q is a uniformly random rotation over $\mathbb{R}^n$ and the subcodes $\mathfrak{C}_1, \ldots, \mathfrak{C}_m$ are sets of B vectors, sampled uniformly and independently random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$, so that codewords are points of the sphere $\mathcal{S}^{n-1} := \{\vec{\boldsymbol{x}} \in \mathbb{R}^n : \|\vec{\boldsymbol{x}}\| = 1\}$. We can have a full description of $\mathfrak{C}$ by storing mB points corresponding to the codewords of $\mathfrak{C}_1, \ldots, \mathfrak{C}_m$ and by storing the rotation Q.

Random product codes have the interesting property of being efficiently decoded in some parameter range:

Proposition 3.13 ([Bec+16])**.** *Let N be a number exponential in n, and $\mathfrak{C}$ be a random product code of parameters (n, m, B) with $m = \widetilde{\mathcal{O}}(n)$ and $B^m = N^{o(1)}$. For any $\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}$ and $\alpha \in [0, \pi/2]$, there is an algorithm that computes the set $\mathfrak{C} \cap \mathcal{H}_{\vec{\boldsymbol{x}},\alpha}$ in time $N^{o(1)} \cdot |\mathfrak{C} \cap \mathcal{H}_{\vec{\boldsymbol{x}},\alpha}|$.*

Claim 3.14. *[Bec+16, Lemma 5.1, proof in Appendix C] Points in a random product code are indistinguishable from uniformly and independently random points on $\mathcal{S}^{n-1}$.*

Definition 3.15 (Hypercone filter)**.** A filter is characterized by a center $\boldsymbol{c} \in \mathcal{S}^{n-1}$ and an angle α, that defines a hypercone. A filter of center $\boldsymbol{c}$ is said α-close with a vector $\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}$ iff. $\boldsymbol{c}$ and $\vec{\boldsymbol{x}}$ are of angle at most α. The filter is associated with a set $f_\alpha(\boldsymbol{c})$, often called "bucket" in the literature, that is initially empty and can be filled with α-close vectors.

Condition of reduction.

In the NV-sieve (Algorithms 3 and 5), we searched pairs of vectors of angle at most $\pi/3$ in the list L. To add a filtering layer, we preprocess the list by filling each filter bucket $f_\alpha(\vec{\boldsymbol{s}})$ with α-close vectors in L. Then, for each vector $\vec{\boldsymbol{x}}$ in L, we search for a reducing one within its filters instead of checking the much larger whole list. Vectors in $\vec{\boldsymbol{x}}$'s filters have a higher probability of reducing with $\vec{\boldsymbol{x}}$ in comparison with random vectors from L. We will quantify this, but first of all, let us do an observation that leads to a quite useful simplification. Notice that for any angle $\alpha \in [\pi/3, \pi/2]$ and $\epsilon > 0$ such that $\epsilon < \alpha$, we have

$\mathcal{V}_n(\alpha-\epsilon) = \text{poly}(n)\cdot\sin^n(\alpha-\epsilon) = \mathcal{V}_n(\alpha)\cdot(\epsilon')^n$ with $\epsilon' = \cos\epsilon - \frac{\sin(\epsilon)\cos(\alpha)}{\sin(\alpha)} < 1$ for $\alpha > \epsilon$. This leads to $\mathcal{V}_n(\alpha) \gg \mathcal{V}_n(\alpha-\epsilon)$. So the probability for a point to be at angle α with the center of the cap is exponentially higher than to be at angle $\alpha-\epsilon$. That justifies that, for a filter of center $\vec{s}$ and angle α, vectors in $\mathcal{H}_{\vec{s},\alpha}$ lie very close to the border of the cap. So we do the following simplification.

Claim 3.16. *We consider that vectors in a filter bucket $f_\alpha(\vec{s})$ of center $\vec{s}$ and angle α are actually lying on the border of the filter $\mathcal{B}_{\vec{s},\alpha} := \{\vec{x}\in\mathcal{S}^{n-1} \mid \theta(\vec{x},\vec{s}) = \alpha\}$. A vector $\vec{x}\in\mathcal{B}_{\vec{s},\alpha}$ then can be decomposed into*

$$\vec{x} = \cos(\alpha)\vec{s} + \sin(\alpha)\vec{y}$$

for some $\vec{y}$ of norm 1 and orthogonal to $\vec{s}$. Such vector $\vec{y}$ is called a residual vector of $\vec{x}$ in the filter $f_\alpha(\vec{s})$.

Lemma 3.17. *Let $\vec{s}\in\mathcal{S}^{n-1}$ and $\alpha\in(0,\pi/2]$, and a random vector $\vec{x}$ random uniform in $f_{\vec{s}}$. Decompose $\vec{x} = \cos(\alpha)\vec{s}+\sin(\alpha)\vec{y}$ with $\vec{y}\perp\vec{s}$ and $\|\vec{y}\| = 1$. Then the distribution of $\vec{y}$ is uniform in an $(n-2)$-dimensional sphere inside the orthogonal complement of $\vec{s}$.*

Proof. $\vec{x}$ is uniformly distributed and its uniformity is invariant under rotations. Then the distribution of $\sin(\alpha)\vec{y}$ is invariant under rotation around $\vec{s}$. Normalizing gives the uniformity of $\vec{y}$ in $\mathcal{S}^{n-2}$. □

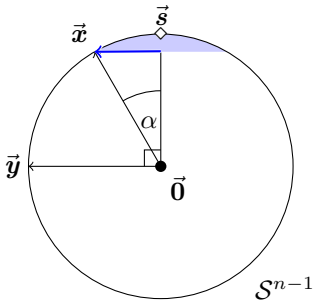


Figure 3.6: The vector $\vec{x}$ is in the filter of center $\vec{s}$ and angle α. The blue arrow represents the non-normalized vector $\sin(\alpha)\vec{y}$.

Proposition 3.18. *Let a filter of center $\vec{s}$ and angle $\alpha\in\left[\frac{\pi}{3},\frac{\pi}{2}\right]$. Given vectors $\vec{x}_1,\vec{x}_2\in\mathcal{B}_{\vec{s},\alpha}$, we can write $\vec{x}_1 = \cos(\alpha)\vec{s}+\sin(\alpha)\vec{y}_1$ and $\vec{x}_2 = \cos(\alpha)\vec{s}+\sin(\alpha)\vec{y}_2$ for some $\vec{y}_1,\vec{y}_2$ of norm 1 and orthogonal to $\vec{s}$. For $\alpha\in\left[\frac{\pi}{3},\frac{\pi}{2}\right]$ we have the equivalence*

$$\theta(\vec{x}_1,\vec{x}_2)\leqslant\frac{\pi}{3} \iff \theta(\vec{y}_1,\vec{y}_2)\leqslant 2\arcsin\left(\frac{1}{2\sin(\alpha)}\right).$$

Proof. We denote for simplicity $\theta_y := \theta(\vec{y}_1,\vec{y}_2)$. Combining both equations gives the condition, for any $\alpha\in\left[\frac{\pi}{3},\frac{\pi}{2}\right]$,

$$\|\vec{x}_1-\vec{x}_2\|^2\leqslant 1 \Leftrightarrow \sin^2(\alpha)\|\vec{y}_1-\vec{y}_2\|^2\leqslant 1 \Leftrightarrow \sin^2(\alpha)(2-2\cos(\theta_y))\leqslant 1 \Leftrightarrow \cos(\theta_y)\geqslant 1-\frac{1}{2\sin^2(\alpha)}$$
$$\Leftrightarrow \theta_y\leqslant\arccos\left(1-\frac{1}{2\sin^2(\alpha)}\right) = 2\arcsin\left(\frac{1}{2\sin(\alpha)}\right)$$

One can easily check using a computing engine that the final equality is verified for all $\alpha\in\left[\frac{\pi}{3},\frac{\pi}{2}\right]$. □

Corollary 3.19. *Consider a filter of center $\vec{s}\in\mathcal{S}^{n-1}$ and angle $\alpha\in(0,\pi/2]$. Let $\vec{x}_1,\vec{x}_2\in\mathcal{S}^{n-1}$ be random vectors such that $\theta(\vec{x}_1,\vec{s}) = \theta(\vec{x}_2,\vec{s}) = \alpha$. Then the pair $(\vec{x}_1,\vec{x}_2)$ reduces with probability $\mathcal{V}_{n-1}(\theta^*_\alpha)$ where*

$$\theta^*_\alpha := 2\arcsin\left(\frac{1}{2\sin(\alpha)}\right).$$

As the angle θ^* is larger than $\pi/3$, we have $\mathcal{V}_n(\theta^*)\geq\mathcal{V}_n(\pi/3)$. The left term can be interpreted as the probability that two random vectors in a shared filter of angle α reduce together, while the right one is the probability that two random vectors in $\mathcal{S}^{n-1}$ reduce together.

3.3.2. Sieving with locality-sensitive filtering

Before presenting the algorithms, let us remember that the algorithms stand under the following assumptions.

- **Heuristic 3.8**. The input lattice points are uniformly randomly distributed on the sphere $\mathcal{S}^{n-1} := \{\vec{\boldsymbol{x}} \in \mathbb{R}^n : \|\vec{\boldsymbol{x}}\| = 1\}$.
- **Claim 3.14**. The points of a random product code are indistinguishable from random independent points in $\mathcal{S}^{n-1}$.

We recall that the spherical cap of center $\vec{\boldsymbol{s}}$ and angle α is denoted $\mathcal{H}_{\vec{\boldsymbol{s}},\alpha} := \{\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1} \mid \theta(\vec{\boldsymbol{x}}, \vec{\boldsymbol{s}}) \leq \alpha\}$.

Proposition 3.20 ([Bec+16] with the correction of [Laa15]). *Let angles $\alpha, \beta, \theta \in \left(0, \frac{\pi}{2}\right)$ be such that $\cos(\theta) \leq \min\left\{\frac{\cos(\alpha)}{\cos(\beta)}, \frac{\cos(\beta)}{\cos(\alpha)}\right\}$. For two vectors $\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2 \in \mathcal{S}^{n-1}$ such that $\langle \vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2 \rangle = \cos(\theta)$, the ratio of the volume of the wedge $\mathcal{H}_{\vec{\boldsymbol{x}}_1,\alpha} \cap \mathcal{H}_{\vec{\boldsymbol{x}}_2,\beta}$ to the volume of the sphere $\mathcal{S}^{n-1}$ is*

$$\mathcal{W}_n(\alpha, \beta, \theta) := \mathrm{poly}(n) \cdot \left(1 - \gamma^2\right)^{n/2} \text{ with } \gamma = \sqrt{\frac{\cos^2(\alpha) + \cos^2(\beta) - 2\cos(\alpha)\cos(\beta)\cos(\theta)}{\sin^2(\theta)}}.$$

And in particular, when $\alpha = \beta$,

$$\mathcal{W}_n(\alpha, \theta) := \mathcal{W}_n(\alpha, \alpha, \theta) = \mathrm{poly}(n) \cdot \left(1 - \frac{2\cos^2(\alpha)}{1 + \cos(\theta)}\right)^{n/2}.$$

Then given two random uniform and independent vectors $\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2 \in \mathcal{S}^{n-1}$, we have $\Pr_{\vec{\boldsymbol{s}} \in \mathcal{S}^{n-1}}[\vec{\boldsymbol{s}} \in \mathcal{H}_{\vec{\boldsymbol{x}}_1,\alpha} \cap \mathcal{H}_{\vec{\boldsymbol{x}}_2,\beta}] = \mathcal{W}_n(\alpha, \beta, \theta)$.

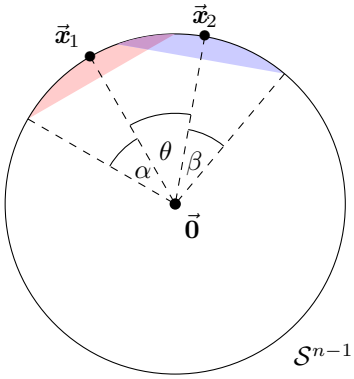


Figure 3.7: The spherical cap $\mathcal{H}_{\vec{\boldsymbol{x}}_1,\alpha}$ is the intersection of the red area with the sphere $\mathcal{S}^{n-1}$, and $\mathcal{H}_{\vec{\boldsymbol{x}}_2,\beta}$ is the intersection of the blue area with $\mathcal{S}^{n-1}$. The wedge $\mathcal{H}_{\vec{\boldsymbol{x}}_1,\alpha} \cap \mathcal{H}_{\vec{\boldsymbol{x}}_2,\beta}$ with $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle = \cos(\theta)$ in purple at the intersection of the two caps.

Classical NV-sieve with LSF.

Theorem 3.21. *[Laa15] There exists a classical algorithm that heuristically solves SVP in dimension n in time and memory $2^{0.292n+o(n)}$.*

Proof. The algorithm searches all the pairs $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \in L^2$ of angle $\pi/3$, so that they yield shorter vectors by subtracting one to the other. We start by sampling a random product code $\mathfrak{C}$ (Definition 3.12) to generate the filters (Step 2). $\mathcal{W}_n(\alpha, \beta, \pi/3)$ is the probability for two reducing vectors to share a common filter, so we set its size such that $|\mathfrak{C}| \cdot \mathcal{W}_n(\alpha, \beta, \pi/3) = 1$, ensuring a high probability to find the reducing pair through

Algorithm 6 Classical NV-sieve with locality-sensitive filtering [Bec+16]

Require: List L with N lattice vectors of norm at most 1, reducing factor $\gamma < 1$; angles $\alpha, \beta \in (0, \pi/2]$.
Ensure: List L_{out} of N lattice vectors of norm at most γ.
1: $L_{out} = \{\}$
2: Sample a random product code $\mathbb{C}$ and initialize the buckets $f_\beta(\vec{s})$ at empty for each $\vec{s} \in \mathbb{C}$
3: **for each** $\vec{x} \in L$ **do**
4: Compute $\mathbb{C} \cap \mathcal{H}_{\vec{x},\beta}$ ▷ Set of all β-close filters of $\vec{x}$.
5: **for each** $\vec{s} \in (\mathbb{C} \cap \mathcal{H}_{\vec{x},\beta})$, add $\vec{x}$ to the bucket $f_\beta(\vec{s})$.
6: **for each** $\vec{x}_1 \in L$ **do**
7: Compute $\mathcal{F} = \mathbb{C} \cap \mathcal{H}_{\vec{x}_1,\alpha}$ ▷ Set of all α-close filters of $\vec{x}_1$.
8: Construct the set $\mathcal{B} = \left(\bigcup_{\vec{s}\in\mathcal{F}} f_\beta(\boldsymbol{c})\right) \setminus \{\vec{x}_1\}$ ▷ Set of all vectors sharing a bucket with $\vec{x}_1$
9: **for** $\vec{x}_2 \in \mathcal{B}$ **do**
10: **if** $\|\vec{x}_1 - \vec{x}_2\| \leq \gamma$ **then** add $\vec{x}_1 - \vec{x}_2$ to L_{out}
11: **return** L_{out}

a collision in a filter. Then, for each code word $\boldsymbol{c} \in \mathbb{C}$, it generates a filter associated with a bucket $f_\beta(\boldsymbol{c})$. The preprocessing step corresponds to the lines 3-5. For each $\vec{x}_1 \in L$, it uses the efficient list decoding algorithm from Proposition 3.13 to compute all the codewords in $\mathbb{C}$ that are of angle at most β with $\vec{x}_1$. This constructs the set $\mathbb{C} \cap \mathcal{H}_{\vec{x}_1,\beta}$ in time $\mathcal{O}\left(|\mathbb{C} \cap \mathcal{H}_{\vec{x}_1,\beta}|\right) = \mathcal{O}\left(\frac{\mathcal{V}_n(\beta)}{\mathcal{W}_n(\alpha,\beta,\pi/3)}\right)$. The querying step (Lines 6-10) computes, for each $\vec{x}_1 \in L$, the set $\mathcal{F} = \mathbb{C} \cap \mathcal{H}_{\vec{x}_1,\alpha}$ of all filter centers of angle at most α with $\vec{x}_1$. This takes time $\mathcal{O}\left(|\mathbb{C}| \cdot \mathcal{V}_n(\alpha)\right) = \mathcal{O}\left(\frac{\mathcal{V}_n(\alpha)}{\mathcal{W}_n(\alpha,\beta,\pi/3)}\right)$. It then construct the joined set $\mathcal{B} = \left(\bigcup_{\boldsymbol{c}\in\mathcal{F}} f_\alpha(\vec{s})\right) \setminus \{\vec{x}_1\}$, filled with all vectors sharing a bucket with $\vec{x}_1$. Then for each $\vec{x}_2$ in $\mathcal{B}$, if $\vec{x}_1 - \vec{x}_2$ is of norm at most γ, it adds it to the output list L_{out}. This takes time $|\mathcal{B}| = |L| \cdot \mathcal{V}_n(\alpha)$. The overall cost of this algorithm is then $\frac{\mathcal{V}_n(\beta)}{\mathcal{W}_n(\alpha,\beta,\pi/3)} + |L| \cdot \mathcal{V}_n(\alpha)$, where $|L| = \frac{1}{\mathcal{V}_n(\pi/3)}$. Taking $\alpha = \beta = \frac{\pi}{3}$ gives an algorithm of both time and memory in $2^{0.292n+o(n)}$. □

There is a modified version of this algorithm [Laa15, Section 13-4-4] that keeps the memory minimum. Instead of filling the buckets with all their close vectors, which has a high memory cost, we add the vectors sequentially to their buckets. The time complexity does not change but the memory requirement is kept at the minimum memory amount N.

Theorem 3.22. *[Laa15] There exists a classical algorithm that solves SVP in dimension n in time $2^{0.292n+o(n)}$ using memory $2^{0.208n+o(n)}$.*

Quantum NV-sieve with LSF.

Theorem 3.23. *There exists a quantum algorithm that heuristically solves SVP in dimension n in time $2^{0.265n+o(n)}$ and memory $2^{0.208n+o(n)}$ under the assumption of efficient QRACM operations.*

Proof. Lattice points are assumed randomly uniformly distributed on the sphere $\mathcal{S}^{n-1}$ by Heuristic 3.8. As in the classical version (Algorithm 6), the size of the code is $|\mathbb{C}| = \frac{1}{\mathcal{W}_n(\alpha,\beta,\pi/3)}$, and the preprocessing step takes time $\frac{\mathcal{V}_n(\beta)}{\mathcal{W}_n(\alpha,\beta,\pi/3)}$. The queries step replaces the exhaustive classical search by a Grover's search, which takes time $\sqrt{|\mathcal{B}|} = \sqrt{|L| \cdot \mathcal{V}_n(\alpha)}$ under the assumption that QRACM operations are efficiently implementable. Then the overall time is $\frac{\mathcal{V}_n(\beta)}{\mathcal{W}_n(\alpha,\beta,\pi/3)} + \sqrt{\frac{\mathcal{V}_n(\alpha)}{\mathcal{V}_n(\pi/3)}}$. Choosing $\alpha = \beta = \arccos\left(\frac{\sqrt{3}}{2}\right)$ leads to the result. □

The same trick as for the classical version reduces the required memory to $N = 2^{0.208n+o(n)}$, by sequentially adding the vectors to their buckets.

Algorithm 7 Quantum NV-sieve with locality-sensitive filtering [Laa15]

Require: List L with N lattice vectors of norm at most 1, reducing factor $\gamma < 1$; angles $\alpha, \beta \in (0, \pi/2]$.
Ensure: List L_{out} of N lattice vectors of norm at most γ.

$L_{out} = \{\}$
Sample a random product code $\mathfrak{C}$ and initialize the buckets $f_\beta(\boldsymbol{c})$ at empty for each $\boldsymbol{c} \in \mathfrak{C}$
for each $\vec{\boldsymbol{x}}_1 \in L$ **do**
    Compute $\mathfrak{C} \cap \mathcal{H}_{\vec{\boldsymbol{x}}_1,\beta}$ ▷ Set of all β-close filters of $\vec{\boldsymbol{x}}_1$.
    For each $\boldsymbol{c} \in (\mathfrak{C} \cap \mathcal{H}_{\vec{\boldsymbol{x}},\beta})$, add $\vec{\boldsymbol{x}}_1$ to the bucket $f_\beta(\boldsymbol{c})$.
for each $\vec{\boldsymbol{x}}_1 \in L$ **do**
    Compute $\mathcal{F} = \mathfrak{C} \cap \mathcal{H}_{\vec{\boldsymbol{x}}_1,\alpha}$ ▷ Set of all β-close filters of $\vec{\boldsymbol{x}}_1$.
    $\vec{\boldsymbol{x}}_2 \leftarrow$ Grover on set $\left(\bigcup_{\boldsymbol{c}\in\mathcal{F}} f_\beta(\boldsymbol{c})\right) \setminus \{\vec{\boldsymbol{x}}_1\}$ with function $f_{\text{check}} : \vec{\boldsymbol{x}}_2 \mapsto \begin{cases} 1 \text{ if } \|\vec{\boldsymbol{x}}_1 - \vec{\boldsymbol{x}}_2\| \leq \gamma \\ 0 \text{ otherwise.} \end{cases}$
    Add $\vec{\boldsymbol{x}}_1 - \vec{\boldsymbol{x}}_2$ to list L_{out}
return L_{out}

Conditional optimality. [KL21] have shown that the hypercone filters [Bec+16] are optimal in the lattice sieving setting for the Euclidean norm. Based on this result, they gave a conditional lower bound within the framework of running a lattice sieve with pairwise NNS techniques and showed that the [Bec+16] algorithm is optimal in the classical model. However, as we will show later, it is possible to go below this conditional lower bound by considering algorithms that do not fit in their framework. Indeed, in Chapter 4, we will study a new algorithm that uses quantum walks to find close vectors, instead of a simple Grover's search. They also claim that "*Similar optimality results extend to the tuple sieving results of Herold–Kirshanova-Laarhoven [HKL18], the pairwise sieve with quantum speedups [Laa15]*". We will see in Chapter 5 that the running time of this algorithm is not absolutely optimal either, as we use a different filtering technique from theirs.

Quantum filtering. The quantum algorithm 7 still performs the calculation of the closest filters classically. After our work, [Hei21] has proposed to use Grover's search to sample a random product code. The filter buckets are only constructed as a quantum superposition of centers, and this gives a quadratic speed up on the cost of computing all the relevant filters of a given query vector. Applied to the sieving, it gives the following theorem.

Theorem 3.24. *There exists a quantum algorithm that heuristically solves SVP in time* $2^{0.2571n+o(n)}$ *and* $2^{0.2075n+o(n)}$ *memory.*

3.4. k-Sieves

Sieving algorithms reach the best time complexities to solve SVP, but they have the drawback of requiring an exponentially large memory. A way to reduce it is by the k-sieve, introduced in [BLS16]. The idea is to sum k lattice points instead of pairs at each sieving step. This decreases the number N of lattice points that we need at each step to find the same number N of shorter lattice points. However, this will drastically increase the time to perform the sieving step. For instance, a naive exhaustive search of each k-tuple takes time $\mathcal{O}(N^k)$, and the fact that N is smaller does not outweigh this increased exponent.

Definition 3.25 (Approximate k-List problem)**.** Given k lists $L_1 \dots, L_k$ of equal exponential (in n) size N and whose elements are *i.i.d.* uniformly chosen vectors from $\mathcal{S}^{n-1}$, the approximate k-list problem is to find N k-tuples $(\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_k) \in L_1 \times \cdots \times L_k$ satisfying $\|\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k\| \leq 1$.

This problem reduces to the sieving problem. Its condition $\|\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k\| \leq 1$ can be rewritten

$$\|\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k\|^2 = \sum_{i=1}^{k} \|\vec{\boldsymbol{x}}_i\|^2 + 2 \sum_{i,j\neq i}^{k} \langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle.$$

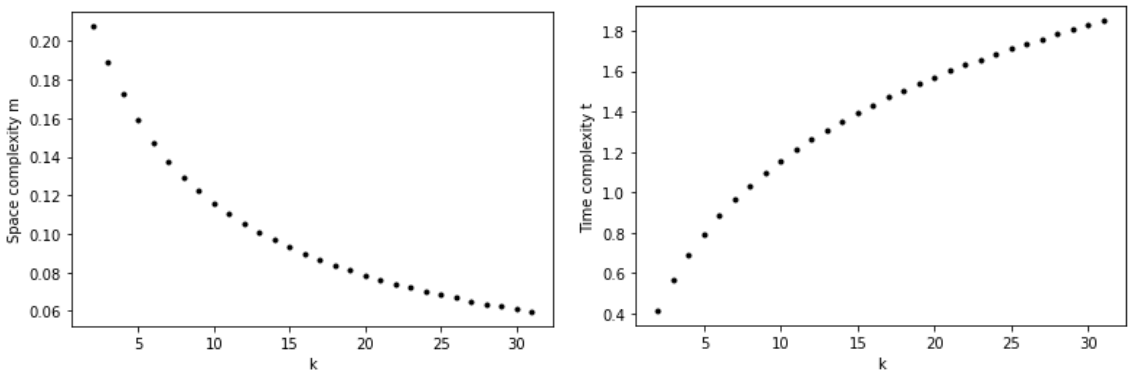


Figure 3.8: Space and time asymptotic complexities for the naive k-sieve. The space $N = 2^{mn+o(n)}$ decreases when k increases, but the running time $N^k = 2^{tn+o(n)}$ explodes. The size N will be expressed in Theorem 3.28.

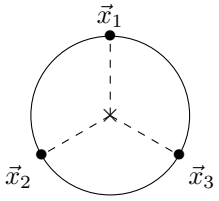
It means that a tuple reduces if some constraints on its scalar products $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle$ are verified. This motivates the introduction of vector configurations.

Algorithm 8 Naive k-Sieve

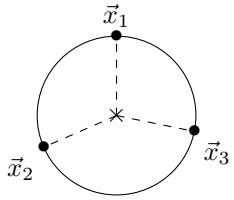
Require: Lists $L_1, \ldots, L_k$ of N lattice vectors of norm at most 1, a reducing factor $\gamma < 1$.
Ensure: List L_{out} of N lattice vectors of norm at most γ.
  for $(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k) \in L_1 \times \cdots \times L_k$ **do**
    if $\left\| \sum_{i=1}^k \vec{\boldsymbol{x}}_i \right\| \leq \gamma$ **then** add $\sum_{i=1}^k \vec{\boldsymbol{x}}_i$ to L_{out}
  return L_{out}

Definition 3.26 (Configuration)**.** The configuration C of k vectors $\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k \in \mathcal{S}^{n-1}$ is the Gram matrix of the vectors $\vec{\boldsymbol{x}}_i$, *i.e.* $C_{i,j} = \langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle$, with necessarily $\forall i, C_{i,i} = 1$. A configuration is said balanced when $C_{i,j} = -1/k$ for $i \neq j$, and $C_{i,i} = 1$. In this case, the tuple points will form the summits of a regular polyhedron inscribed in the sphere. For $I \subset [k]$, we denote by $C[I]$ the $|I| \times |I|$ submatrix of C obtained by restricting C to the rows and columns whose indexes are in I.

Notice that if two random vectors $\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{x}}_j$ satisfy $C_{i,j} \leq \langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle$, then we will have $C_{i,j} \approx \langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle$ with high probability. This will be formalized later in Lemma 3.16.



(a) Balanced configuration $C_{1,2} = C_{1,3} = C_{2,3} = -\frac{1}{2}$.



(b) Configuration with $C_{1,2} \approx -0.4$, $C_{1,3} \approx -0.2$ and $C_{2,3} \approx -0.8$.

Definition 3.27 (Configuration problem)**.** Let $k \in \mathbb{N}$ and $\epsilon > 0$. We are given k lists $L_1, \ldots, L_k$ all of exponential (in n) size N whose elements are *i.i.d.* uniform from $\mathcal{S}^{n-1}$. For a given target configuration $C = (C_{i,j})_{i,j \in [k]} \in \mathbb{R}^{k \times k}$, the configuration problem consists in finding a $1 - o(\epsilon)$ fraction of all k-tuples $(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k) \in L_1 \times \cdots \times L_k$ such that $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle \leq C_{i,j}$ for all $i \neq j$.

The configuration C can be chosen among the set of symmetric positive semi-definite matrices in $\mathbb{R}^{k \times k}$ with $\forall i, C_{i,i} = 1$. A k-tuple $(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k)$ satisfying C implies $\|\sum_i \vec{\boldsymbol{x}}_i\| = \sum_{i,j} C_{i,j} = \mathbf{1}^t C \mathbf{1}$, where $\mathbf{1}$ is a column

vector of 1's. Thus a k-tuple satisfying configuration C is a solution to the approximate k-list problem if and only if we have $\mathbf{1}^t C \mathbf{1} \leq 1$. Actually, considering the configuration problem brings an additional constraint, the k-tuples have to satisfy a constraint on their configuration instead of just having $\|\sum_i \vec{\boldsymbol{x}}_i\| \leq 1$.

Theorem 3.28. *[HK17, Theorem 1] The probability that a k-tuple of i.i.d. uniformly random points on $\mathcal{S}^{n-1}$ satisfies a given configuration $C \in \mathbb{R}^{k\times k}$ is $\det(C)^{n/2}$.*

Remark. Both Lemmas 3.9 for the volume of a spherical cap and 3.20 for the volume of a wedge can be obtained by applying the above theorem. It suffices to choose well the configurations. Let be $\alpha, \beta \in (0,1)$ and consider two vectors $\vec{\boldsymbol{v}}, \vec{\boldsymbol{w}} \in \mathcal{S}^{n-1}$ of angle at most $\theta \in (0, \pi/2)$ with $\beta \geq \alpha\cos(\theta)$ and $\alpha \geq \beta\cos(\theta)$. Then we can recover the formula of the wedge from 3.20:

$$\begin{aligned}
\Pr_{\boldsymbol{s}\in\mathcal{S}^{n-1}}[\langle\vec{\boldsymbol{v}}|\boldsymbol{s}\rangle \geq \alpha, \langle\vec{\boldsymbol{w}}|\boldsymbol{s}\rangle \geq \beta \mid \langle\vec{\boldsymbol{v}}|\vec{\boldsymbol{w}}\rangle \geq \cos(\theta)] &= \frac{\Pr_{\vec{\boldsymbol{v}},\vec{\boldsymbol{w}},\boldsymbol{s}\in\mathcal{S}^{n-1}}[\langle\vec{\boldsymbol{v}}|\boldsymbol{s}\rangle \geq \alpha, \langle\vec{\boldsymbol{w}}|\boldsymbol{s}\rangle \geq \beta, \langle\vec{\boldsymbol{v}}|\vec{\boldsymbol{w}}\rangle \geq \cos(\theta)]}{\Pr_{\vec{\boldsymbol{v}},\vec{\boldsymbol{w}}\in\mathcal{S}^{n-1}}[\langle\vec{\boldsymbol{v}}|\vec{\boldsymbol{w}}\rangle \geq \cos(\theta)]} \\
&= \frac{\begin{vmatrix} 1 & \cos(\theta) & \alpha \\ \cos(\theta) & 1 & \beta \\ \alpha & \beta & 1 \end{vmatrix}^{n/2}}{\begin{vmatrix} 1 & \cos(\theta) \\ \cos(\theta) & 1 \end{vmatrix}^{n/2}} \quad \text{by applying Theorem 3.28.} \\
&= \frac{(1-\cos^2(\theta)+2\alpha\beta\cos(\theta)-\alpha^2-\beta^2)^{n/2}}{(1-\cos^2(\theta))^{n/2}} \\
&= (1-\gamma^2)^{n/2} \quad \text{with } \gamma := \sqrt{\frac{\alpha^2+\beta^2-2\alpha\beta\cos(\theta)}{\sin^2(\theta)}}.
\end{aligned}$$

After a sieving step with all lists of size N, we want to get $N^k \cdot \det(C)^{n/2}$ k-tuples satisfying the chosen configuration C so that they are reducing k-tuples. We need this number of solutions to be equal to N. So we can deduce the required size for N, in the function of the target configuration:

$$N = \widetilde{\mathcal{O}}\left(\left(\frac{1}{\det(C)}\right)^{\frac{n}{2(k-1)}}\right). \tag{3.1}$$

The k-Sieve aims to improve the time-memory tradeoffs, as searching for tuples requires less memory than searching for pairs. The required memory amount decreases when k increases. However, we have to pay the price by a higher time complexity. For a fixed k, the minimum value of N is reached when the configuration C is balanced. In particular, with a balanced configuration for $k=2$ we require $2^{0.208n}$ points ; $2^{0.189n}$ points for $k=3$ and $2^{0.172n}$ for $k=4$. Figure 3.8 displays some other values for higher k. Considering non-balanced configurations leads to not reaching the lower bound for memory but it can improve time. This is useful to get better time-memory tradeoffs.

Proposition 3.29 (Size of the filtered lists $L_i(x_j)$ given $C_{i,j}$ [Kir+19])**.** *We are given a configuration $C \in \mathbb{R}^{k\times k}$ and lists $L_1, \dots L_k \subset \mathcal{S}^{n-1}$ each of size $|L_j|$. For $\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_i \in \mathcal{S}^{n-1}$, we denote*

$$L_j(\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_i) := \{\vec{\boldsymbol{x}}_j \in L_j \ : \ \langle\vec{\boldsymbol{x}}_1|\vec{\boldsymbol{x}}_j\rangle \leq C_{1,j}, \ \dots \ , \langle\vec{\boldsymbol{x}}_i|\vec{\boldsymbol{x}}_j\rangle \leq C_{i,j}\}.$$

Then, for a i-tuple $\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_i$ satisfying the configuration $C[1\dots i]$, the expected size of $L_j(\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_i)$ is

$$\mathbb{E}(|L_j(\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_i)|) = |L_j| \cdot \left(\frac{\det(C[1, \dots, i, j])}{\det(C[1, \dots, i])}\right)^{n/2}.$$

In particular,

$$\mathbb{E}(|L_j(\vec{\boldsymbol{x}}_i)|) = |L_j| \cdot \left(1 - C_{i,j}^2\right)^{n/2}.$$

k-List algorithms The first classical algorithm for the configuration problem for $k \geq 2$ was given by [BLS16]. The idea is to fix a first vector $\vec{\boldsymbol{x}}_1 \in L_1$ and to construct $L_2(\vec{\boldsymbol{x}}_1)$, then fix $\vec{\boldsymbol{x}}_2 \in L_2(\vec{\boldsymbol{x}}_1)$ to construct $L_3(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2)$, and so on until constructing $L_k(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_{k-1})$. A tuple $(\vec{\boldsymbol{x}}_1, \ldots \vec{\boldsymbol{x}}_k)$ taken in these lists yields one solution to the configuration problem. It then iterates on each $\vec{\boldsymbol{x}}_1 \in L_1$ to find all the solutions. This algorithm was later improved by [HK17] by also constructing intermediate lists $L_i(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_{i-1})$ at each level i. The algorithm [HKL18] adds locality-sensitive filtering to the algorithm [HK17] to speed up pairwise searches. It applies the filtering described in Algorithm 6 where the angle constraint becomes $\operatorname{acos}(C_{i,j})$ instead of $\frac{\pi}{3}$. [Kir+19] proposed quantum versions of BLS and HKL algorithms, and they show that a hybrid version performs more efficiently, by starting with HKL and then continuing with BLS. This hybrid version tends to have the same tradeoffs as the quantum BLS algorithm when k is high. [Kir+19, Appendix B] also adds pairwise filtering in their quantum hybrid algorithm. It is not straightforward to know how to optimally combine the k-sieve structure with filtering. We will see in Chapter 5 a new way to add filtering to the k-sieve, that is complementary to the pairwise filtering approach of previous works. We will bring more details on k-List algorithms in Chapter 5. Tables 5.4 and 5.5 recap the time and memory complexities of these algorithms, respectively in the classical and quantum settings.

4. Lattice sieving via quantum walks

The work in this chapter has been published in ASIACRYPT 2021 [CL21] and is a joint work with André Chailloux.

4.1. Overview

Sieving with LSF. A sieving step starts from a list L of $N := (4/3)^{n/2}$ of lattice vectors in dimension n of norm at most 1, and outputs N lattice vectors of norm at most $\gamma < 1$. We actually take γ very close to 1 at each iteration to simplify the analysis. The 2-sieve searches reducing pairs of vectors, *i.e.* of angle at most $\frac{\pi}{3}$. Please look at Section 3.2.1 for more details on sieving algorithms.

[Bec+16] presented an initial framework for sieving with LSF to solve SVP which we presented in Section 3.3. Their algorithm samples a random product code which generates centers of the filters. Then, during the preprocessing step, it goes through the input list and inserts each list vector into all its nearest filters. During the querying step, for each list vector, it searches for a reducing one within its filters. [Laa15] replaced the classical exhaustive search of the querying part with Grover's search, which describes a new best quantum algorithm for SVP. Doubling the filtering layer did not improve the complexity, so it was believed useless.

Quantum walks. We introduced quantum walks in Section 2.3.3. Given a set of elements $S = \{x_1, \dots, x_N\}$, a quantum walk is an algorithm that finds a subset $v \subseteq S$ of size r that satisfies a wanted property, if such "marked vertex" v exists. Typically in our case, we want to find a subset v that contains two vectors that reduce together. The main idea behind our algorithm is to replace Grover's searches in the quantum sieve of [Laa15] with quantum walks. It was not *a priori* clear how to adapt the algorithm to integrate quantum walks as there are many ways of constructing them and most of them do not give speedups.

Contributions. In this chapter, we present a new quantum sieving algorithm for solving SVP using quantum walks. This is the first improvement in the asymptotic running time of quantum sieving algorithms since the work of Laarhoven [Laa15], bringing down the time from $2^{0.2653d+o(d)}$ to $2^{0.2570d+o(d)}$. We also show that the state-of-the-art classical and quantum sieving algorithms actually fit into our framework. Finally, we present two trade-offs: for fixed quantum memory and for fixed QRAM, and we show that the best classical and previous best quantum algorithms fit our framework.

Outline. In Section 4.2, we present the general framework we use for our sieving algorithm. Next, we perform a first study of its complexity in Section 4.3, whose Section 4.4 improves with an additional idea, the sparsification. In Section 4.5, we present the space-time trade-offs. In Section 4.6, we present how a later improvement of the circuit of quantum walks slightly improves our attack. Finally, we discuss the results in Section 4.7 and talk about the parallelization of our algorithm as well as possible improvements. The SageMath code used for the numerical results of this chapter is available here: `https://github.com/johanna-loyer/sieving-via-QRW`.

4.2. Framework for sieving algorithms using filtering

Notations. $\mathcal{S}^{n-1}$ is the sphere in $\mathbb{R}^n$ of radius 1. The spherical cap of center $\vec{s} \in \mathcal{S}^{n-1}$ and angle α is $\mathcal{H}_{\vec{s},\alpha} := \{\vec{x} \in \mathcal{S}^{n-1} \mid \theta(\vec{x}, \vec{s}) \leq \alpha\}$. Proposition 3.9 gave the ratio volume of the spherical cap $\mathcal{V}_n(\alpha) = \Pr_{\vec{x}\in\mathcal{S}^{n-1}}[\vec{x} \in \mathcal{H}_{\vec{s},\alpha}]$, and Proposition 3.20 gave those of the wedge $\mathcal{W}_n(\alpha, \theta) = \Pr_{\vec{s}\in\mathcal{S}^{n-1}}[\vec{s} \in \mathcal{H}_{\vec{x}_1,\alpha} \cap \mathcal{H}_{\vec{x}_2,\alpha}]$.

The core idea behind the framework of our sieving algorithms is the following:

1. Prefilter the list vectors,
2. Search all reduced pairs within each filter,
3. Repeat steps 1. and 2. until most of the reduced pairs are found.

Our algorithm takes as input a parameter $c_\alpha \in (0,1)$ which is linked with the size of the filters during the preprocessing step. We will discuss later how to choose it optimally.

Algorithm 9 Framework for sieving with LSF prefiltering

Input: List L of N lattice vectors of norm at most 1, reducing factor $\gamma < 1$ and parameter $c_\alpha \in (0,1)$.
Output: List L_{out} of N lattice vectors of norm at most γ.
Algorithm:

$L_{out} := \{\}$
Fix the angle $\alpha \in [\pi/3, \pi/2]$ such that $N \cdot \mathcal{V}_n(\alpha) = N^{c_\alpha}$.
while $|L_{out}| \leq N$ **do** ▷ $\mathrm{NB}_{\mathrm{REP}}$ repeats
  Sample a random product code $\mathfrak{C}$ of size $1/\mathcal{V}_n(\alpha)$ and denote its codewords $\vec{s}_i$
  for each $\vec{x}$ in L **do**
    $\vec{s}_i \leftarrow$ **Decode**$(\vec{x}, \mathfrak{C})$ ▷ Algorithm from Proposition 3.13
    Add $\vec{x}$ to the bucket $f_\alpha(\vec{s}_i)$
  for each filter numbered $i \in [N^{1-c_\alpha}]$ **do**
    $Sol \leftarrow \mathsf{FindAllReducing}(f_\alpha(\vec{s}_i))$ ▷ Finds N^ζ solutions
    $L_{out} \leftarrow L_{out} \cup Sol$
return L_{out}

The $\mathsf{FindAllReducing}(f_\alpha(\vec{s}_i))$ subroutine starts from a list of vectors $\vec{x}_1, \ldots, \vec{x}_{N^{c_\alpha}} \in f_\alpha(\vec{s}_i)$ and outputs all vectors of the form $\vec{x}_i - \vec{x}_j$ (with $i \neq j$) of norm lower than γ. We want to find asymptotically all the solutions and not strictly all of them. Sometimes, there are no solutions so the algorithm outputs an empty list.

Complexity analysis of framework algorithm 9

We first present the heuristic arguments and simplifying assumptions we use for our analysis. All of them were introduced and discussed previously.

Heuristic and simplifying assumptions. We remind the reader that the complexity analysis of the algorithms presented in this chapter relies on the following assumptions:

- **Heuristic 3.8**. The input lattice points are uniformly randomly distributed on the sphere $\mathcal{S}^{n-1} := \{\vec{x} \in \mathbb{R}^n : \|\vec{x}\| = 1\}$.
- **Claim 3.14**. The points of a random product code are indistinguishable from random independent points in $\mathcal{S}^{n-1}$.
- **Claim 3.16**. Given a point $\vec{s} \in \mathcal{S}^{n-1}$, we assume that a random vector $\vec{x}$ of angle at most α with $\vec{s}$ is exactly at angle α, and then can be decomposed $\vec{x} = \cos(\alpha)\vec{s} + \sin(\alpha)\vec{y}$ with $\vec{y} \perp \vec{s}$ and $\|\vec{y}\| = 1$. The residual vector $\vec{y}$ is random uniform in $\mathcal{S}^{n-2}$ an orthogonal complement to $\vec{s}$.
- We suppose that there exists a quantum circuit that efficiently implements QRACM and QRAQM operations. (See Section 2.2)

1. Prefiltering (lines 4-7) We start by sampling a random product code $\mathfrak{C}$ (defined in Part 3.12) of size $1/\mathcal{V}_n(\alpha)$ whose codewords are denoted $\vec{s}_i$. For each point $\vec{x} \in L$, we compute $\mathcal{H}_{\vec{x},\alpha} \cap \mathfrak{C}$ using the efficient decoding algorithm. We pick inside it a codeword $\vec{s}_i$ and update the corresponding buckets $f_\alpha(\vec{s}_i)$. We have $|\mathfrak{C}| = N^{1-c_\alpha}$ and we chose α such that $\mathcal{V}_n(\alpha) = N^{-(1-c_\alpha)}$, so the expected value of $|\mathcal{H}_{\vec{v},\alpha} \cap \mathfrak{C}|$ is $|\mathfrak{C}| \cdot \mathcal{V}_n(\alpha) = 1$. For each point $\vec{x}$, we can compute $\mathcal{H}_{\vec{v},\alpha} \cap \mathfrak{C}$ in time $N^{o(1)}|\mathcal{H}_{\vec{v},\alpha}|$ using the algorithm from Proposition 3.13. From there, we can conclude that we can compute the filters for the N points in time $\mathcal{O}(N)$.

After preprocessing the list L of size N, each α-filter is filled with $N \cdot \mathcal{V}_n(\alpha) = N^{c_\alpha}$ vectors on average. Those N^{c_α} points are randomly distributed in this filter. The number of filters is

$$\text{NbFilters} := N^{1-c_\alpha} = \frac{1}{\mathcal{V}_n(\alpha)}.$$

2. Find all solutions within an α-filter (lines 8-10) Within an α-filter, two vectors reduce if their respective residual vectors in the filter are of angle at most θ^*_α whose expression is given in Corollary 3.19. The expected number of reducing pairs is $N^{2c_\alpha} \cdot \mathcal{V}_{n-1}(\theta^*_\alpha) =: N^\zeta$.

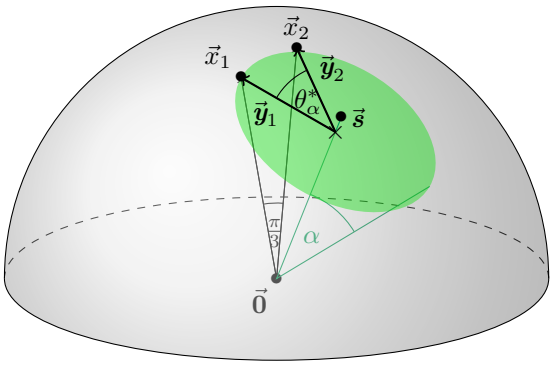


Figure 4.1: Given $\vec{s} \in \mathcal{S}^{n-1}$, and $\vec{x}_1, \vec{x}_2 \in f_\alpha(\vec{s})$, then we have $\theta(\vec{x}_1, \vec{x}_2) = \pi/3 \Leftrightarrow \theta(\vec{y}_1, \vec{y}_2) = \theta^*_\alpha$. The point denoted $\times$ is the center of the sphere $\mathcal{S}^{n-2}$ of points orthogonal to $\vec{s}$.

We denote by $T(\text{FAR})$ the running time of the FindAllReducing algorithm. We will see in the next Section a proposal of such a subroutine with its complexity analysis. The querying step runs a FindAllReducing subroutine for each of the α-filters which are at number N^{1-c_α}. So it runs in time $\text{NbFilters} \cdot T(\text{FAR})$. The average number of solutions found is N^ζ for each call to FindAllReducing, so we find $N^{1-c_\alpha+\zeta}$ solutions in total after one iteration in while loop. Notice that we can have $\zeta < 0$, which means that we can find on average much less than one solution for each call to FindAllReducing.

3. Number of repeats (while loop 3) Each call to FindAllReducing finds N^ζ reduced vectors. After searching all the solutions within every filter, we expect to find $|\mathfrak{C}| \cdot N^\zeta = N^{1-c_\alpha+\zeta}$ solutions.

To complete the sieve, we need N reduced lattice vectors. Thus steps 1. and 2. have to be repeated until we reach this number of solutions. The number of repetitions of the while loop is

$$\text{NbRep} = \max\{1, N^{1-(1-c_\alpha+\zeta)+o(1)}\} = \max\{1, N^{c_\alpha-\zeta+o(1)}\}.$$

Let us summarize this complexity analysis in the following proposition.

Proposition 4.1. *Consider $c_\alpha \in [0,1]$ and angle $\alpha \in [\pi/3, \pi/2]$ satisfying $N \cdot \mathcal{V}_n(\alpha) = N^{c_\alpha}$. We are given a subroutine FindAllReducing that finds N^ζ in time $T(\text{FAR})$.*

The Algorithm 9 with this subroutine and parameter c_α runs in time

$$T = \text{NbRep} \cdot (N + \text{NbFilters} \cdot T(\text{FAR}))$$

Notice that the above running time only depends on c_α (since α and ζ derives from c_α) and on the running time of the FindAllReducing subroutine. This framework encompasses state-of-the-art classical and quantum sieving algorithms.

State-of-the-art classical algorithm. In order to retrieve the algorithm of [Laa15, Algorithm 13.3] (Theorem 3.22 in this thesis), we take $c_\alpha \to 0$, which implies $\alpha \to \pi/3$. We can compute $\theta^*_{\pi/3} \approx 1.23\text{rad} \approx 70.53^o$ and $\zeta = -0.4094$. The FindAllReducing subroutine simply performs an exhaustive search among all the pairs in an α-filter. In this case, we have $T(\text{FAR}) = \mathcal{O}(1)$. From the above proposition, we get a total running time of $T = N^{1.4094+o(1)} = 2^{0.2925n+o(n)}$.

State-of-the-art quantum algorithm. We take $c_\alpha = 0.2782$. This value actually corresponds to the case where $\zeta = 0$, so we have on average $N^\zeta = 1$ solution per α-filter. For the FindAllReducing subroutine, we can apply Grover's algorithm on pairs of vectors in the filter of size N^{c_α}. It finds a solution in time $\sqrt{N^{2c_\alpha}} = N^{c_\alpha}$, so $T(\text{FAR}) = N^{c_\alpha}$. Putting this together, we obtain an overall time $T = N^{1+c_\alpha+o(1)} = N^{1.2782+o(1)} = 2^{0.2653n+o(n)}$. So we get the same complexities as the algorithm [Laa15] (Algorithm 7 in this thesis) in the low memory regime with $N = 2^{0.2075n+o(n)}$ classical memory.

4.3. Finding reducing pairs by quantum walks

We focus now on describing an algorithm for FindAllReducing whose input is a list of vectors in a filter of center $\vec{s} \in \mathcal{S}^{n-1}$ and angle $\alpha \in [\pi/3, \pi/2]$, and we want to return a list of all the reducing pairs within $f_\alpha(\vec{s})$. We describe here such a subroutine that uses quantum walks to achieve this. Once we find a marked vertex, it contains a pair $(\vec{y}_i, \vec{y}_j)$ such that $\theta(\vec{y}_i, \vec{y}_j) \leq \theta^*_\alpha$ from which we directly get a reducible pair $(\vec{x}_i, \vec{x}_j)$.

4.3.1. Constructing the graph

We consider the unordered list $L_x = \{\vec{x}_1, \ldots, \vec{x}_{N^{c_\alpha}}\} \subseteq f_\alpha(\vec{s})$ of distinct points with α satisfying $\mathcal{V}_n(\alpha) = N^{-(1-c_\alpha)}$. For each $i \in [N^{c_\alpha}]$, we can write $\vec{x}_i = \cos(\alpha)\vec{s} + \sin(\alpha)\vec{y}_i$ where each $\vec{y}_i$ is of norm 1 and orthogonal to $\vec{s}$. Let $L_y = \{\vec{y}_1, \ldots, \vec{y}_{N^{c_\alpha}}\}$ be the list of all the corresponding residual vectors of the $\vec{x}_i$'s in L_x. Recall from Proposition 3.18 that a pair $(\vec{x}_i, \vec{x}_j)$ reduces if their residual vectors $\vec{y}_i, \vec{y}_j$ satisfy $\theta(\vec{y}_i, \vec{y}_j) \leq \theta^*_\alpha := 2\arcsin(\frac{1}{2\sin(\alpha)})$.

We will present a quantum walk to find pair $(\vec{y}_i, \vec{y}_j)$ such that $\theta(\vec{y}_i, \vec{y}_j) \leq \theta^*_\alpha$ more efficiently than Grover's search. The graph in which we perform the walk is the usual one for the collision problem, the Johnson graph (see Section 2.3.3). Our quantum walk takes two extra parameters $c_V \in [0, c_\alpha]$ and $c_\beta \in [0, c_V]$. From these two parameters, let $\beta \in \left[\frac{\pi}{3}, \frac{\pi}{2}\right]$ such that $\mathcal{V}_n(\beta) = N^{c_\beta - c_V}$ and ρ_0 such that $N^{\rho_0} = \frac{\mathcal{V}_n(\beta)}{\mathcal{W}_n(\beta, \theta^*_\alpha)}$. We start by sampling a random product code $\mathfrak{C}_\beta$ with parameters $\left[(n-1), \log(n-1), N^{\frac{\rho_0 + c_V - c_\beta}{\log(n-1)}}\right]$ which has therefore $N^{\rho_0 + c_V - c_\beta} = \frac{1}{\mathcal{W}_n(\beta, \theta^*_\alpha)}$ points denoted $\vec{t}_1, \ldots, \vec{t}_{N^{\rho_0 + c_V - c_\beta}}$. We perform our quantum walk on a graph $G = (V, E)$ where each vertex $v \in V$ contains:

- An unordered list $L^v_y = \{\vec{y}_1, \ldots, \vec{y}_{N^{c_V}}\}$ of distinct points taken from L_y.
- For each $\vec{t}_i \in \mathfrak{C}_\beta$, we store the list of elements of $J^v(\vec{t}_i) := f_\beta(\vec{t}_i) \cap L^v_y$, using a QRAQM data structure allowing us to efficiently add and delete in quantum superposition. Notice that we have on average

  $$|J^v(\vec{t}_i)| = N^{c_V} \cdot \mathcal{V}_n(\beta) = N^{c_\beta},$$

  and we need to store in total $|\mathfrak{C}_\beta| \cdot N^{c_\beta} = N^{c_V + \rho_0}$ such elements in total for each vertex.

- A bit that says whether the vertex is marked (we detail the marked condition below).

The vertices of G consist of the above description for all possible lists L_y^v. We set that $(v,w) \in E$ if we can go from L_y^v to L_y^w by changing exactly one element. In other words,

$$(v,w) \in E \Leftrightarrow \exists \vec{\boldsymbol{y}}_{old} \in L_y^v, \exists \vec{\boldsymbol{y}}_{new} \in L_y \backslash L_y^v \text{ st. } L_y^w = \left(L_y^v \backslash \{\vec{\boldsymbol{y}}_{old}\}\right) \cup \{\vec{\boldsymbol{y}}_{new}\}.$$

This means the graph G is a Johnson graph $J(N^{c_\alpha}, N^{c_V})$ where each vertex also has some additional information as we described above.

Condition for a vertex to be marked. We first define the set M_0 of vertices that contain a reducing pair of points.

$$M_0 := \{v \in V : \exists \vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \in L_y^v, \vec{\boldsymbol{y}}_j \neq \vec{\boldsymbol{y}}_i, \theta(\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j) \leq \theta_\alpha^*\}.$$

Ideally, we would want to mark each vertex in M_0, however, this would induce a too-large update cost when updating the bit that specifies whether the vertex is marked or not. Instead, we will consider marked vertices subsets of M_0 for which the update can be done more efficiently, but losing only a small fraction of these vertices. For each $J^v(\vec{\boldsymbol{t}}_i) = \{\vec{\boldsymbol{y}}'_1, \ldots, \vec{\boldsymbol{y}}'_{|J^v(\vec{\boldsymbol{t}}_i)|}\}$, we define $\widetilde{J}^v(\vec{\boldsymbol{t}}_i) = \{\vec{\boldsymbol{y}}'_j \in J^v(\vec{\boldsymbol{t}}_i), j \leq \min\{|J^v(\vec{\boldsymbol{t}}_i)|, 2N^{c_\beta}\}\}$ which consists of the first $2N^{c_\beta}$ elements of $J^v(\vec{\boldsymbol{t}}_i)$, considering a global ordering of elements of L_y, for example with respect to their index and $\widetilde{J}^v(\vec{\boldsymbol{t}}_i)$ consists of the $2N^{c_\beta}$ elements of $J^v(\vec{\boldsymbol{t}}_i)$ which are the smallest with respect to this ordering. In the case where $|J^v(\vec{\boldsymbol{t}}_i)| \leq 2N^{c_\beta}$, we set $\widetilde{J}^v(\vec{\boldsymbol{t}}_i) = J^v(\vec{\boldsymbol{t}}_i)$. We define the set of marked elements M as follows:

$$M := \{v \in V : \exists \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta, \exists \vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \in \widetilde{J}^v(\vec{\boldsymbol{t}}), \vec{\boldsymbol{y}}_j \neq \vec{\boldsymbol{y}}_i, \text{ st. } \theta(\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j) \leq \theta_\alpha^*\}.$$

The reason for using such a condition for marked vertices is that when we perform an update, hence removing a point $\vec{\boldsymbol{y}}_{old}$ from a vertex and adding a point $\vec{\boldsymbol{y}}_{new}$, we will just need to look at the points in $\widetilde{J}^v(\vec{\boldsymbol{t}})$ for $\vec{\boldsymbol{t}} \in \mathcal{H}_{\vec{\boldsymbol{y}}_{new},\beta} \cap \mathfrak{C}_\beta$ which can be done faster than by looking at all the points of the vertex. If we used $J^v(\vec{\boldsymbol{t}})$ instead of $\widetilde{J}^v(\vec{\boldsymbol{t}})$ then the argument would be simpler but we would only be able to argue about the average running time of the update but the quantum walk framework requires to give an upper bound of the update time for any pair of adjacent vertices. Also notice that each vertex still contain the sets $J^v(\vec{\boldsymbol{t}}_i)$ (from which one can easily compute $\widetilde{J}^v(\vec{\boldsymbol{t}}_i)$).

Once we have found such a pair $(\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j)$, we can immediately recover the corresponding lattice vectors $(\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{x}}_j)$ and compute their difference to get a shorter vector.

4.3.2. Complexity analysis

Let us first consider the following lemma that will be useful in the complexity analysis of our algorithm.

Lemma 4.2. *Consider a vector $\vec{\boldsymbol{s}} \in \mathcal{S}^{n-1}$, an angle $\alpha \in (0, \pi/2]$, and a set $S = \{\vec{\boldsymbol{s}}_1, \ldots, \vec{\boldsymbol{s}}_M\}$ of i.i.d. uniformly random vectors in $\mathcal{S}^{n-1}$. We assume we have $M\mathcal{V}_n(\alpha) = N^x$ with a constant $x > 0$. Then we have:*

$$\Pr[|S \cap \mathcal{H}_{\vec{\boldsymbol{s}},\alpha}| \geq 2N^x] \leq e^{-\frac{N^x}{3}}.$$

Proof. For any $i \in [M]$, both $\vec{\boldsymbol{x}}_i$ and $\vec{\boldsymbol{s}}$ are independent and uniform random points on the sphere $\mathcal{S}^{n-1}$, so by definition 3.9 we have $\forall i \in [M], \Pr[\vec{\boldsymbol{x}}_i \in \mathcal{H}_{\vec{\boldsymbol{s}},\alpha}] = \Pr[\vec{\boldsymbol{s}} \in \mathcal{H}_{\vec{\boldsymbol{x}}_i,\alpha}] = \mathcal{V}_n(\alpha)$. So we immediately have that $\mathbb{E}[|S \cap \mathcal{H}_{\vec{\boldsymbol{x}}_i,\alpha}|] = M\mathcal{V}_n(\alpha)$. Let X_i be the random variable which is equal to 1 if $\vec{\boldsymbol{x}}_i \in \mathcal{H}_{\vec{\boldsymbol{s}},\alpha}$ and is equal to 0 otherwise. Let $Y = \sum_{i=1}^M X_i$ so $\mathbb{E}[Y] = N^x$. Y is equal to the quantity $|S \cap \mathcal{H}_{\vec{\boldsymbol{s}},\alpha}|$. We then apply the multiplicative Chernoff bound. As a reminder, it states that if $X_1, ..., X_M$ are independent random variables taking values in $\{0,1\}$, $Y = \sum_{i=1}^M$ and $\delta > 0$, then we have $\Pr[Y \geq (1+\delta)\mathbb{E}[Y]] \leq e^{-\frac{\delta^2 \mathbb{E}[Y]}{3}}$. In our context, we get $\Pr[Y \geq 2N^x] \leq e^{-\frac{N^x}{3}}$, which is the desired result. □

We are now ready to analyze the different costs of our quantum walk. For a fixed chosen $\beta \in \left[\frac{\pi}{3}, \frac{\pi}{2}\right]$, $\mathcal{H}_{\vec{\boldsymbol{y}}_i,\beta} \cap \mathfrak{C}_\beta$ is the set of β-close filters to the vector $\vec{\boldsymbol{y}}_i$, and we have on average

$$|\mathcal{H}_{\vec{\boldsymbol{y}}_i,\beta} \cap \mathfrak{C}_\beta| = N^{\rho_0 + c_V - c_\beta} \cdot \mathcal{V}_n(\beta) = N^{\rho_0}.$$

Using Lemma 4.2, we have for each $i \in [N^{c_V}]$,

$$\Pr[|\mathcal{H}_{\vec{\boldsymbol{y}}_i,\beta} \cap \mathfrak{C}_\beta| > 2N^{\rho_0}] \leq e^{-\frac{N^{\rho_0}}{3}} \tag{4.1}$$

and using a union bound, we have for any absolute constant $\rho_0 > 0$:

$$\Pr[\forall i \in [N^{c_\alpha}],\ |\mathcal{H}_{\vec{\boldsymbol{y}}_i,\beta} \cap \mathfrak{C}_\beta| \leq 2N^{\rho_0}] \geq 1 - N^{c_\alpha} e^{-\frac{N^{\rho_0}}{3}} = 1 - o(1). \tag{4.2}$$

So for a fixed α-filter, we have with high probability that each $|\mathcal{H}_{\vec{\boldsymbol{y}}_i,\beta} \cap \mathfrak{C}_\beta|$ is bounded by $2N^{\rho_0}$ and we assume we are in this case. The sets $\mathcal{H}_{\vec{\boldsymbol{y}}_i,\beta} \cap \mathfrak{C}_\beta$ can hence be constructed in time $N^{\rho_0 + o(1)}$ using the efficient list decoding algorithm (Proposition 3.13) with code $\mathfrak{C}_\beta$.

We consider the quantum walk [Mag+11]-framework, whose complexity was analyzed in Proposition 2.11. We have to express the costs of the setup, update, check, ϵ the fraction of marked vertices, and δ the spectral gap of the graph.

Setup cost. In order to construct a full vertex v from a list $L_y^v = \{\vec{\boldsymbol{y}}_1, \ldots, \vec{\boldsymbol{y}}_{N^{c_V}}\}$, the main cost is to construct the lists $J^v(\vec{\boldsymbol{t}}_i) = f_\beta(\vec{\boldsymbol{t}}_i) \cap L_y^v$. To do this, we start from empty lists $J^v(\vec{\boldsymbol{t}}_i)$. For each $\vec{\boldsymbol{y}}_i \in L_y^v$, we construct the list $f_\beta(\vec{\boldsymbol{y}}_i) \cap \mathfrak{C}_\beta$ and for each codeword $\vec{\boldsymbol{t}}_j$ in this set, we add $\vec{\boldsymbol{y}}_i$ in $J^v(\vec{\boldsymbol{t}}_i)$. This takes time $N^{c_V} \cdot N^{\rho_0 + o(1)}$. We can construct a uniform superposition of the vertices by performing the above procedure in quantum superposition. This can also be done in $N^{c_V} \cdot N^{\rho_0 + o(1)}$ since we use a quantum data structure that performs these insertions in $J^v(\vec{\boldsymbol{t}}_i)$ efficiently. The setup cost is then

$$\mathcal{S} = N^{c_V + \rho_0 + o(1)}.$$

Update cost. We show here how to go from a vertex v with associated list L_y^v to a vertex w with $L_y^w = \left(L_y^v \backslash \{\vec{\boldsymbol{y}}_{old}\}\right) \cup \{\vec{\boldsymbol{y}}_{new}\}$. The vertex v also contains the lists $J^v(\vec{\boldsymbol{t}}_i) = f_\beta(\vec{\boldsymbol{t}}_i) \cap L_y^v$ for each $\vec{\boldsymbol{t}}_i \in \mathfrak{C}_\beta$ that need to be updated. In order to get the lists $J^w(\vec{\boldsymbol{t}}_i)$, we first compute $\mathcal{H}_{\vec{\boldsymbol{y}}_{old},\beta} \cap \mathfrak{C}_\beta$ and for each $\vec{\boldsymbol{t}}_i$ in this set, we remove $\vec{\boldsymbol{y}}_{old}$ from $J^v(\vec{\boldsymbol{t}}_i)$. Then, we compute $\mathcal{H}_{\vec{\boldsymbol{y}}_{new},\beta} \cap \mathfrak{C}_\beta$ and for each $\vec{\boldsymbol{t}}_i$ in this set, we add $\vec{\boldsymbol{y}}_{new}$ to $J^v(\vec{\boldsymbol{t}}_i)$, and thus we obtain all the $J^w(\vec{\boldsymbol{t}}_i)$. Constructing the two lists takes time on average $N^{\rho_0 + o(1)}$ and we then perform at most $2N^{\rho_0}$ deletion and insertion operations which are done efficiently. So the update of the filter lists takes time $N^{\rho_0 + o(1)}$.

If v was marked and $\vec{\boldsymbol{y}}_{old}$ is not part of the reducible pair then we do not change the last registers for L_y^w. If v was not marked, then we have to ensure that adding $\vec{\boldsymbol{y}}_{new}$ does not make it marked. So we need to check whether there exists $\vec{\boldsymbol{y}}_0 \neq \vec{\boldsymbol{y}}_{new}$ such that

$$\exists \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta \ :\ \vec{\boldsymbol{y}}_0, \vec{\boldsymbol{y}}_{new} \in \widetilde{J}^w(\vec{\boldsymbol{t}}) \text{ and } (\vec{\boldsymbol{y}}_{new}, \vec{\boldsymbol{y}}_0) \text{ are reducible.}$$

If such a point $\vec{\boldsymbol{y}}_0$ exists, it necessarily lies in the set $\bigcup_{\vec{\boldsymbol{t}} \in \mathcal{H}_{\vec{\boldsymbol{y}}_{new},\beta} \cap \mathfrak{C}_\beta} \widetilde{J}^v(\vec{\boldsymbol{t}})$ which is of size at most $2N^{\rho_0} \cdot 2N^{c_\beta} = 4N^{\rho_0 + c_\beta}$. We perform a Grover's search on this set to determine whether there exists a $\vec{\boldsymbol{y}}_0 \in \bigcup_{\vec{\boldsymbol{t}} \in \mathfrak{C}_\beta} \widetilde{J}^v(\vec{\boldsymbol{t}})$ that reduces with $\vec{\boldsymbol{y}}_{new}$, and this takes time $N^{\frac{\rho_0 + c_\beta + o(1)}{2}}$. In conclusion, the average update time is

$$\mathcal{U} = N^{\rho_0 + o(1)} + N^{\frac{\rho_0 + c_\beta + o(1)}{2}} \leq N^{\max\left\{\rho_0, \frac{\rho_0 + c_\beta}{2}\right\} + o(1)}.$$

Checking cost. Each vertex has a bit that says whether it is marked or not, which is already checked during the update step, so we have

$$\mathcal{C} = 1.$$

Fraction of marked vertices ϵ.

Proposition 4.3. $\epsilon \geq \Theta\left(\min\left\{N^{2c_V} \cdot \mathcal{V}_n(\theta_\alpha^*), 1\right\}\right)$.

Proof. We consider a random vertex v in the graph. A sufficient condition for v to be marked is to satisfy the following two events :

- E_1 : $\exists \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta, \exists \vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \neq \vec{\boldsymbol{y}}_i \in J^v(\vec{\boldsymbol{t}}),$ st. $\theta(\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j) \leq \theta_\alpha^*$.
- E_2 : $\forall \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta, |J^v(\vec{\boldsymbol{t}})| \leq 2N^{c_\beta}$.

The second property implies that $\forall \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta,\ J^v(\vec{\boldsymbol{t}}) = \widetilde{J}^v(\vec{\boldsymbol{t}})$ and in that case, the first property implies that v is marked. We now bound the probability of each event in Lemmas 4.4 and 4.5. □

Lemma 4.4. $\Pr[E_1] \geq \Theta\left(\min\left\{N^{2c_V}\mathcal{V}_n(\theta_\alpha^*), 1\right\}\right)$.

Proof. For a fixed pair $\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \neq \vec{\boldsymbol{y}}_i \in L_y^v$, we have $\Pr[\theta(\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j) \leq \theta_\alpha^*] = \mathcal{V}_n(\theta_\alpha^*)$. Since there are $\Theta(N^{2c_V})$ such pairs, if we define the event E_0 as: $\exists \vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \neq \vec{\boldsymbol{y}}_i \in L_y^v,$ st. $\theta(\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j) \leq \theta_\alpha^*$, we have

$$\Pr[E_0] \geq \Theta\left(\min\left\{N^{2c_V}\mathcal{V}_n(\theta_\alpha^*), 1\right\}\right).$$

Now we assume E_0 holds and we try to compute the probability that E_1 is truly conditioned on E_0. So we assume E_0 and let $\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \neq \vec{\boldsymbol{y}}_i \in L_y^v,$ st. $\theta(\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j) \leq \theta_\alpha^*$. For each code point $\vec{\boldsymbol{t}} \in \mathfrak{C}_\beta$, we have

$$\Pr[\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \in J^v(\vec{\boldsymbol{t}})] = \Pr[\vec{\boldsymbol{t}} \in \mathcal{H}_{\vec{\boldsymbol{y}}_i,\beta} \cap \mathcal{H}_{\vec{\boldsymbol{y}}_j,\beta}] = \mathcal{W}_n(\beta, \theta_\alpha^*).$$

Therefore, we have

$$\Pr[\exists \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta,\ \vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \in J^v(\vec{\boldsymbol{t}})] = 1 - (1 - \mathcal{W}_n(\beta, \theta_\alpha^*))^{|\mathfrak{C}_\beta|}. \tag{4.3}$$

Since $|\mathfrak{C}_\beta| = 1/\mathcal{W}_n(\beta, \theta_\alpha^*)$, we can conclude

$$\Pr[E_1|E_0] \geq \Pr[\exists \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta,\ \vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \in J^v(\vec{\boldsymbol{t}})] = 1 - (1 - \mathcal{W}_n(\beta, \theta_\alpha^*))^{|\mathfrak{C}_\beta|} \geq \Theta(1),$$

which implies $\Pr[E_1] \geq \Pr[E_1|E_0] \cdot \Pr[E_0] \geq \Theta\left(\max\left\{N^{2c_V}\mathcal{V}_n(\theta_\alpha^*), 1\right\}\right)$. □

Lemma 4.5. $\Pr[E_2] \geq 1 - |\mathfrak{C}_\beta| e^{-\frac{N^{c_\beta}}{3}}$.

Proof. For each $\vec{\boldsymbol{t}} \in \mathfrak{C}_\beta$, we have using Lemma 4.2 that $\Pr[|J^v(\vec{\boldsymbol{t}})| \leq 2N^{c_\beta}] \geq 1 - e^{-\frac{N^{c_\beta}}{3}}$. Using a union bound, we have

$$\Pr[\forall \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta, |J^v(\vec{\boldsymbol{t}})| \leq 2N^{c_\beta}] \geq 1 - |\mathfrak{C}_\beta| e^{-\frac{N^{c_\beta}}{3}}.$$

□

We can now finish the proof of our Proposition. We have

$$\begin{aligned}
\epsilon \geq \Pr[E_1 \wedge E_2] &\geq \Pr[E_1] + \Pr[E_2] - 1 \\
&\geq \Theta\left(\max\left\{N^{2c_V}\mathcal{V}_n(\theta_\alpha^*), 1\right\}\right) - |\mathfrak{C}_\beta| e^{-\frac{N^{c_\beta}}{3}} \\
&\geq \Theta\left(\max\left\{N^{2c_V}\mathcal{V}_n(\theta_\alpha^*), 1\right\}\right)
\end{aligned}$$

The last inequality comes from the fact that $|\mathfrak{C}_\beta| \cdot e^{-N^{c_\beta}/3}$ is vanishing doubly exponentially in n (N is exponential in n) so it is negligible compared to the first term and is absorbed by the $\Theta(\cdot)$.

Spectral gap δ. We are in a $J(N^{c_\alpha}, N^{c_V})$ Johnson graph so we have $\delta \approx N^{-c_V}$.

Running time of the quantum walk. The running time T_1 of the quantum walk is (omitting the $o(1)$ terms and the $\mathcal{O}(\cdot)$ notations)

$$\begin{aligned} T_1 &= \mathcal{S} + \frac{1}{\sqrt{\epsilon}}\left(\frac{\mathcal{U}}{\sqrt{\delta}} + \mathcal{C}\right) \\ &= N^{c_V+\rho_0} + \frac{1}{\max\left\{1, N^{c_V}\sqrt{\mathcal{V}_n(\theta^*_\alpha)}\right\}}\left(N^{\max\left\{\rho_0, \frac{\rho_0+c_\beta}{2}\right\}+\frac{c_V}{2}}\right) \end{aligned}$$

In this running time, we can find one marked vertex with a high probability if it exists. We repeat this quantum walk until we find asymptotically all the solutions, whose expected number is $\max\left\{\frac{N^\zeta}{2}, 1\right\}$.

Algorithm 10 FindAllReducing procedure

Require: Set of vectors $f_\alpha(\vec{\boldsymbol{s}})$
Ensure: List L_{out} with N^ζ pairwise-reduced vectors
1: $L_{out} = \{\}$
2: Compute the list of residual vectors $L_y := \{\vec{\boldsymbol{y}}_i = (\vec{\boldsymbol{x}}_i - \cos(\alpha)\vec{\boldsymbol{s}})/\sin(\alpha), \vec{\boldsymbol{x}}_i \in f_\alpha(\vec{\boldsymbol{s}})\}$
3: Pick a random product code $\mathfrak{C}_\beta$.
4: **while** $L_{out} < N^\zeta$ **do**
5: Run our quantum walk to find a reduced pair $\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \in L_y$.
6: Recover the corresponding lattice vectors $\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{x}}_j \in f_\alpha(\vec{\boldsymbol{s}})$ ▷ Share indices i, j in their respective lists
7: **if** $\vec{\boldsymbol{x}}_i - \vec{\boldsymbol{x}}_j \notin L_{out}$ **then** add $\vec{\boldsymbol{x}}_i - \vec{\boldsymbol{x}}_j$ to L_{out}.
8: **return** L_{out}

For $\zeta > 0$, there are N^ζ different solutions that can be found in each α-filter. Each iteration finds a solution, so this algorithm finds a list of solutions of asymptotic size N^ζ in time

$$T(\text{FAR}) = \max\{N^\zeta, 1\} \cdot T_1.$$

If $\zeta > 0$, our algorithm finds $\Theta(N^\zeta)$ solutions in time $N^\zeta T_1$ and if $\zeta \leq 0$, our algorithm finds one solution in time T_1 with probability $\Theta(N^{-\zeta})$.

Classical memory. We have to store at the same time in classic memory the N list vectors of size n, and the buckets of the α-filters. Each vector is in $N^{o(1)}$ α-filter, so our algorithm requires a classical space of size $N^{1+o(1)}$.

QRAM requirements of the quantum walk. Each vertex v in the graph stores all the $J^v(\vec{\boldsymbol{t}}_i)$ which together take space $N^{c_V+\rho_0}$. We store a superposition of vertices so we need $N^{c_V+\rho_0}$ quantum registers and the same amount of QRAQM because we perform insertions and deletions in the database in quantum superposition. All the operations require QRACM access to the whole list L_y which is classically stored and is of size N^{c_α}. Therefore, we also require N^{c_α} QRACM.

4.3.3. Optimal parameters

Our algorithm takes in argument three parameters:

- $c_\alpha \in [0, 1]$, N^{c_α} is the number of vectors per α-filter,
- $c_V \in [0, c_\alpha]$, N^{c_V} is the number of vectors per vertex,
- $c_\beta \in [0, c_V]$, N^{c_β} is the number of vectors per β-filter.

From these three parameters, we can express all the other variables we use, whose we recall their expressions as they are scattered throughout the previous sections:

- $\alpha \in [\pi/3, \pi/2]$ that satisfies $\mathcal{V}_n(\alpha) = \frac{1}{N^{1-c_\alpha}}$. α is the angle of the first layer of filtering (code $\mathfrak{C}$).
- $\theta^*_\alpha = 2\arcsin\left(\frac{1}{2\sin(\alpha)}\right)$. θ^*_α is the target angle for pairs of residual vectors, given by Proposition 3.18.
- $\beta \in [\pi/3, \pi/2]$ that satisfies $\mathcal{V}_n(\beta) = \frac{1}{N^{c_V - c_\beta}}$. β is the angle of the second layer of filtering (code $\mathfrak{C}_\beta$).
- $\rho_0 \geq 1$ such that $N^{\rho_0} = \frac{\mathcal{V}_n(\beta)}{\mathcal{W}_n(\beta, \theta^*_\alpha)}$. N^{ρ_0} is the number of β-filters for which a vertex vector has to be inserted in not to miss any solution collision.
- $\zeta \geq 0$ such that $N^\zeta = N^{2c_\alpha} \cdot \mathcal{V}_n(\theta^*_\alpha)$. N^ζ is the average number of solutions found per call to FindAllReducing.

Plugging the value of $T(\text{FAR})$ from the end of Section 4.3.2 in Proposition 4.1, we find that the total running time of our quantum sieving algorithm with parameters c, c_V, c_β is

$$T = N^{c_\alpha - \zeta}\left(N + N^{1-c_\alpha}\max\{N^\zeta, 1\}\left(N^{c_V+\rho_0} + \frac{N^{\max\left\{\rho_0, \frac{\rho_0+c_\beta}{2}\right\}+\frac{c_V}{2}}}{\max\left\{1, N^{c_V}\sqrt{\mathcal{V}_n(\theta^*_\alpha)}\right\}}\right)\right).$$

We ran a numerical optimization over c_α, c_V, c_β to get our optimal running time, summed up in the following theorem.

Proposition 4.6. *Our algorithm with parameters*

$$c_\alpha \approx 0.3301 \quad , \quad c_V \approx 0.1952 \quad , \quad c_\beta \approx 0.0603$$

heuristically solves SVP on dimension n in time $T = N^{1.2555+o(1)} = 2^{0.2605n+o(n)}$, uses a classical memory of size $N^{1+o(1)} = 2^{0.2075n+o(n)}$, a quantum memory of size $N^{0.2555+o(1)} = 2^{0.0530n+o(n)}$, QRACM of size $N^{0.3301+o(1)} = 2^{0.0685n+o(n)}$, and QRAQM of size $N^{0.2555+o(1)} = 2^{0.0530n+o(n)}$.

With these parameters, we obtain the values of the other parameters:

$$\alpha \approx 1.1388 \text{ rad}, \quad \theta^*_\alpha \approx 1.1661 \text{ rad}, \quad \beta \approx 1.3745 \text{ rad}$$

$$\rho_0 \approx 0.0603, \quad \zeta \approx 0.0745.$$

As well as the quantum walk costs:

$$\mathcal{S} = N^{c_V+\rho_0} = N^{0.2555}, \quad \mathcal{U} = N^{\rho_0} = N^{0.0603}, \quad \mathcal{C} = 1, \quad \epsilon = \delta = N^{-c_V} = N^{-0.1952}.$$

The equality $\rho_0 = c_\beta$ allows to balance the time of the two operations during the update step. With these parameters we also obtain $\mathcal{S} = \mathcal{U}/\sqrt{\epsilon\,\delta} = N^{c_V+\rho_0} = N^{0.2555d}$, which balances the overall time complexity. Notice that with these parameters, we can rewrite the time expression as

$$T = N^{c_\alpha-\zeta}\left(N + N^{1-c_\alpha+\zeta+c_V+\rho_0}\right) = N^{1+c_\alpha-\zeta} + N^{1+c_V+\rho_0}.$$

Also, having $c_V + \rho_0 = c_\alpha - \zeta$ equalizes the walk cost with the initialization cost. From our previous analysis, we require QRACM of size N^{c_α} and quantum memory and QRAQM of size $N^{c_V+\rho_0}$.

4.4. Adding sparsification

For the second layer of filtering, each point is inserted in N^{ρ_0} β-filters $\vec{t}_i \in \mathfrak{C}_\beta$. The value ρ_0 was fixed in order to make sure that if a pair $\vec{y}_i, \vec{y}_j$ exists in a vertex v, then it will appear on one of the $J^v(\vec{t}_i)$ for $\vec{t}_i \in \mathfrak{C}_\beta$. However, we can relax this and only mark a small fraction of these vertices, by taking a fourth parameter ρ that will replace the choice of ρ_0 above. This will reduce the probability for a vertex to be marked, as we miss solutions, but having a smaller ρ will reduce the overall running time of our quantum random walk.

The construction is exactly the same as in the previous section, except we replace ρ_0 with a freely chosen $\rho \in (0, \rho_0]$. This implies that $|\mathfrak{C}_\beta| = N^{\rho+c_V-c_\beta}$. We can perform the same analysis as above.

Time analysis of this quantum walk in the regime $\zeta + \rho - \rho_0 > 0$. We consider the regime where $\zeta + \rho - \rho_0 > 0$ and $\rho \in (0, \rho_0]$ (in particular $\zeta > 0$, since $\rho_0 > 0$). This regime ensures that even if we have fewer marked vertices, then there are on average more than one marked vertex, so our algorithm finds at least one solution with a constant probability.

The analysis of the walk is exactly the same as in Section 4.3.2, each repetition of the quantum random walk takes time T_1 with

$$T_1 = \mathcal{S} + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\,\mathcal{U} + \mathcal{C}\right)$$

with

$$\mathcal{S} = N^{c_V+\rho}, \quad \mathcal{U} = N^{\max\{\rho, \frac{\rho+c_\beta}{2}\}+o(1)}, \quad \mathcal{C} = 1, \quad \epsilon = N^{2c_V} N^{\rho-\rho_0} \mathcal{V}_n(\theta_\alpha^*), \quad \delta = N^{-c_V}.$$

The only thing to develop is the computation of ϵ. We perform the same analysis as above but with $|\mathfrak{C}_\beta| = N^{\rho+c_V-c_\beta}$. This means that Equation 4.3 of Lemma 4.4 becomes

$$\begin{aligned}\Pr[\exists \vec{\boldsymbol{t}} \in \mathfrak{C}_\beta :\ \vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \in J^v(\vec{\boldsymbol{t}})] &= 1 - (1 - \mathcal{W}_n(\beta, \theta_\alpha^*))^{|\mathfrak{C}_\beta|} \\ &\geq |\mathfrak{C}_\beta| \cdot \mathcal{W}_n(\beta, \theta_\alpha^*) = N^{\rho-\rho_0}.\end{aligned}$$

which gives the extra term $N^{\rho-\rho_0}$ in ϵ. Another issue is that now, we can only extract $N^{\zeta+\rho-\rho_0}$ solutions each time we construct the graph, we have therefore to repeat this procedure to find $\frac{N^{\zeta+\rho-\rho_0}}{2}$ solutions with this graph and then repeat the procedure with a new code $\mathfrak{C}_\beta$. The algorithm becomes:

Algorithm 11 FindAllReducing procedure with free parameter ρ

Require: Set of vectors $f_\alpha(\vec{\boldsymbol{s}})$
Ensure: List L_{out} of half of the all pairwise reduced vectors in $f_\alpha(\vec{\boldsymbol{s}})$
1: $L_{out} = \{\}$
2: Compute the set of residual vectors $f'_\alpha(\vec{\boldsymbol{s}}) := \{\vec{\boldsymbol{y}}_i = (\vec{\boldsymbol{x}}_i - \cos(\alpha)\vec{\boldsymbol{s}})/\sin(\alpha), \vec{\boldsymbol{x}}_i \in f_\alpha(\vec{\boldsymbol{s}})\}$
3: Pick a random product code $\mathfrak{C}_\beta$.
4: **while** $L_{out} < \frac{N^\zeta}{2}$ **do**
5: Run our quantum walk with free ρ on set $f'_\alpha(\vec{\boldsymbol{s}})$ to find a reduced pair $\vec{\boldsymbol{y}}_i, \vec{\boldsymbol{y}}_j \in f_{\vec{\boldsymbol{s}},\alpha}$.
6: Recover the corresponding lattice vectors $\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{x}}_j \in f_\alpha(\vec{\boldsymbol{s}})$ ▷ Share indices i, j in their respective lists
7: **if** $\vec{\boldsymbol{x}}_i - \vec{\boldsymbol{x}}_j \notin L_{out}$ **then** add $\vec{\boldsymbol{x}}_i - \vec{\boldsymbol{x}}_j$ to L_{out}.
8: **return** L_{out}

With this procedure, we also find $\Theta(N^\zeta)$ solutions in time $N^\zeta T_1$ and $\mathrm{FAR}_1 = N^\zeta T_1$ (Recall that we are in the case $\zeta \geq \zeta + \rho - \rho_0 > 0$). Actually, optimal parameters will be when $c_\beta = 0$ and $\rho \to 0$.

Complexity analysis

This change implies that some reducing pairs are missed. For the quantum random walk complexity, this only changes the probability, denoted ϵ, so that a vertex is marked. Indeed, it is equal to the one so that there happens a collision between two vectors through a filter, which is no longer equal to the existence of a reducing pair within the vertex. Indeed, to have a collision, there is the supplementary condition of both vectors of a reducing pair are inserted in the same filter, which is of probability $N^{\rho_0-\rho}$. So we get a higher value of $\epsilon = N^{2c_V}\mathcal{V}_n(\theta_\alpha^*) \cdot N^{\rho_0-\rho}$.

However, this increase is compensated by the reducing of the costs of the setup ($N^{c_V+\rho+o(1)}$) and the update ($2N^{\max\{\rho, \frac{\rho+c_\beta}{2}\}+o(1)}$).

A numerical optimisation over ρ, c_α, c_V and c_β leads to the following theorem.

Theorem 4.7. *Our algorithm with a free ρ with parameters*

$$\rho \to 0 \quad , \quad c_\alpha \approx 0.3696 \quad , \quad c_V \approx 0.2384 \quad , \quad c_\beta = 0$$

heuristically solves SVP on dimension n in time $T = N^{1.2384+o(1)} = 2^{0.2570n+o(n)}$, uses QRAM of maximum size $N^{0.3696} = 2^{0.0767n}$, a quantum memory of size $N^{0.2384} = 2^{0.0495n}$ and uses a classical memory of size $N^{1+o(1)} = 2^{0.2075n+o(n)}$.

With these parameters, we obtain the values of the other parameters:

$$\alpha \approx 1.1514 \text{ rad}, \quad \theta^*_\alpha \approx 1.3104 \text{ rad}, \quad \beta \approx 1.1112 \text{ rad}, \quad \zeta \approx 0.1313.$$

As well as the quantum walk costs:

$$\mathcal{S} = N^{c_V+\rho} = N^{0.2384}, \quad \mathcal{U} = N^\rho = N^{o(1)}, \quad \mathcal{C} = 1, \quad \epsilon = \delta = N^{-c_V} = N^{-0.2384}.$$

We also have $\rho_0 = 0.107$ so we are in the regime where $\zeta + \rho - \rho_0 > 0$. As in the previous time complexity stated in Theorem 4.6, we reach the equality $\mathcal{S} = \mathcal{U}/\sqrt{\epsilon\delta}$, which allows to balance the time of the two steps of the quantum random walk: the setup and the search itself.

Notice that with these parameters, we can rewrite the time

$$T = N^{c_\alpha-\zeta}\left(N + N^{1-c_\alpha+\zeta+c_V+\rho}\right) = N^{1+c_\alpha-\zeta} + N^{1+c_V+\rho}.$$

With our optimal parameters, we have $\rho = 0$ and $c_\alpha - \zeta = c_V$, which equalizes the random walk step with the initialization step. From our previous analysis, the amount of required QRAM is N^{c_α} and the amount of quantum memory needed is N^{c_V}.

4.5. Space-time tradeoffs

By varying the values c_α, c_V, c_β and ρ, we can obtain tradeoffs between QRAM and time, and between quantum memory and time. All the following results come from numerical observations based on the formulas from our complexity analysis of time, quantum memory and QRAM.

4.5.1. Tradeoff for fixed quantum memory.

We computed the minimized time when we add the constraint that the quantum memory must not exceed 2^{Mn}. For a chosen fixed M, the quantum memory is denoted is $2^{\mu_M n} = 2^{Mn}$ and the corresponding minimal time by $2^{\tau_M n}$. The variation of M also impacts the required QRAM to run the algorithm, which we denote by $2^{\gamma_M n}$. We get a tradeoff between time and quantum memory in Figure 4.2a, and the evolution of QRAM in function of M for a minimal time is in Figure 4.2b.

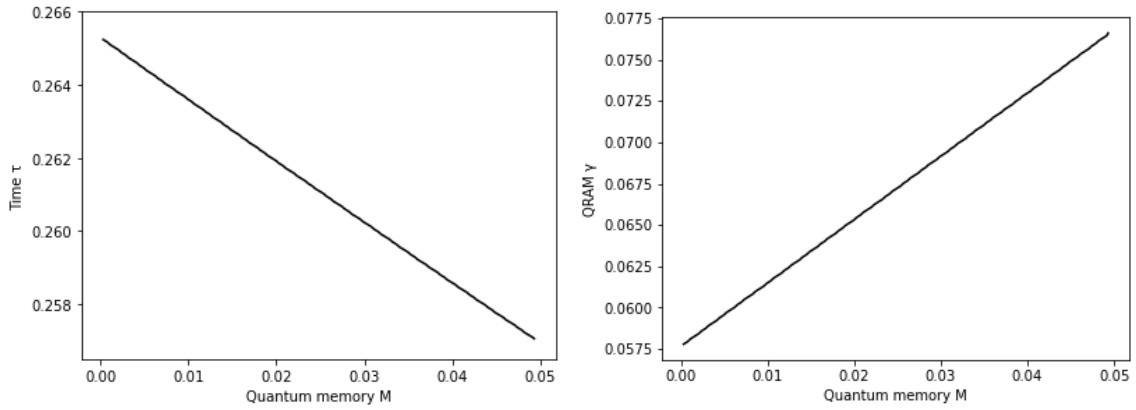
For more than $2^{0.0495n}$ quantum memory, increasing it does not improve the time complexity anymore. An important fact is that for a fixed M the corresponding value τ_M from figure 4.2*a* and γ_M from Figure 4.2b can be achieved simultaneously with the same algorithm.

We observe that from $M = 0$ to 0.0495 these curves are very close to affine. Indeed, the function that passes through the two extremities points is of the expression $0.2653 - 0.1670M$. The difference between τ_M and its affine approximation does not exceed $4 \cdot 10^{-5}$. In the same way, the difference between γ_M and its affine average function of the expression $0.0578 + 0.3829M$ is inferior to $2 \cdot 10^{-4}$. All this is summarized in the following theorem.

Theorem 4.8 (Tradeoff for fixed quantum memory)**.** *There exists a quantum algorithm using quantum random walks that solves SVP on dimension n which for a parameter $M \in [0, 0.0495]$ heuristically runs in time $2^{\tau_M n+o(n)}$, uses QRAM of maximum size $2^{\gamma_M n}$, a quantum memory of size $2^{\mu_M n}$ and a classical memory of size $2^{0.2075n}$ where*

$$\tau_M \in 0.2653 - 0.1670M + [-2 \cdot 10^{-5}; 4 \cdot 10^{-5}]$$

$$\gamma_M \in 0.0578 + 0.3829M - [0; 2 \cdot 10^{-4}] \quad ; \quad \mu_M = M.$$

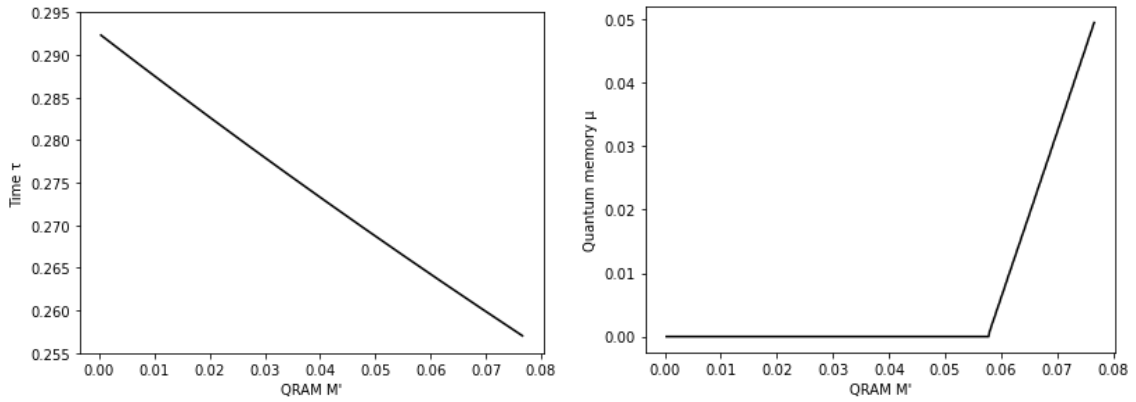


(a) Time in function of available quantum memory. (b) QRAM in function of available quantum memory for minimized time.

Figure 4.2: Tradeoff for fixed quantum memory.

4.5.2. Tradeoff for fixed QRAM.

We also get a tradeoff between QRAM and time. For a chosen fixed M', the QRAM is denoted by $2^{\gamma_{M'}n} = 2^{M'n}$, and the corresponding minimal time by $2^{\tau_{M'}n}$. The required quantum memory is denoted $2^{\mu_{M'}n}$. Note that $2^{\mu_{M'}n}$ is also the amount of the required quantum QRAM. This provides a tradeoff between time and QRAM in figure 4.3a, and the evolution of quantum memory in function of M' is in figure 4.3b.



(a) Time in function of available QRACM. (b) Quantum memory in function of available QRACM for minimized time.

Figure 4.3: Tradeoff for fixed QRACM.

For more than $2^{0.0767n}$ QRACM, increasing it does not improve the time complexity. The difference between the function $\tau_{M'}$ and its average affine function of the expression $0.2926 - 0.4647 \cdot M'$ does not exceed $6 \cdot 10^{-4}$. This affine function is an upper bound of $\tau_{M'}$. From $M' = 0$ to 0.0579 the function $\gamma_{M'}$ is at 0. Then, it is close to the affine function of the expression $2.6356(M' - 0.0579)$. So $\gamma_{M'}$ can be approximated by $\max\{2.6356(M' - 0.0579), 0\}$, and the difference between $\gamma_{M'}$ and this approximation does not exceed $9 \cdot 10^{-4}$. All this is summarized in the following theorem.

Theorem 4.9 (Tradeoff for fixed QRACM)**.** *There exists a quantum algorithm using quantum random walks that solves SVP on dimension n which for a parameter* $M' \in [0, 0.0767]$ *heuristically runs in time* $2^{\tau_{M'}n+o(n)}$*, uses QRACM of size* $\mathrm{poly}(d) \cdot 2^{\gamma_{M'}n}$*, a quantum memory of size* $\mathrm{poly}(n) \cdot 2^{\mu_{M'}n}$ *and uses a classical memory*

of size $\mathrm{poly}(d) \cdot 2^{0.2075n}$ *where*

$$\tau_{M'} \in 0.2927 - 0.4647M' - [0; 6 \cdot 10^{-4}] \quad ; \quad \gamma_{M'} = M'$$

$$\mu_{M'} \in \max\{2.6356(M' - 0.0579), 0\} + [0; 9 \cdot 10^{-4}].$$

Best lattice sieves within our framework

Finally, the following table presents some values of the combined above tradeoffs.

Time $\tau_{M'}$	**0.2925**	0.2827	0.2733	**0.2653**	0.2621	0.2598	**0.2570**
QRACM $\gamma_{M'}$	0	0.02	0.04	**0.0578**	0.065	0.070	**0.0767**
QRAQM $\mu_{M'}$	0	0	0	**0**	0.0190	0.0324	**0.0495**
Comment	[Bec+16] alg.			[Laa15] alg.			Thm 4.7.

Table 4.1: Time, QRAM and quantum memory complexities for our algorithm.

The left-most column complexities are obtained when we choose parameters c_α, c_V, c_β such that no QRAM nor quantum memory are used, and that the time is optimized under this constraint. In this case, we can argue that they are exactly at 0, not only $\mathcal{O}(\mathrm{poly}(n))$. With such a setting, the algorithm fits in the classical model. And we exactly recover the complexity of the best classical lattice sieve [Bec+16]. Then, if we authorize the use of QRAM and $\mathrm{poly}(n)$ qubits, we then recover the previous best quantum algorithm [Laa15], whose results are displayed in the fourth column. The last column repeats the result of our Theorem 4.7 where the parameters are fixed to optimize the time.

4.6. Reusable quantum walks

At the end of this work, we wondered if there could be a more thoughtful way to find k different marked vertices than to run the whole random walk $\mathcal{O}(k)$ times. The authors of [Bon+23] found a way that gives a slight improvement in the complexity of our algorithm. We summarized their results in Section 2.3.3, and here is an application. The time complexity of our FindAllReducing subroutine, by Theorem 4.7, is

$$\mathrm{FAR}_1 = N^{\rho_0} \cdot N^{\zeta - \rho_0} \left(\mathcal{S} + \frac{1}{\sqrt{\epsilon}} \left(\frac{\mathcal{U}}{\sqrt{\delta}} + \mathcal{C} \right) \right). \tag{4.4}$$

With reusable quantum walks, the setup of cost $\mathcal{S}$ can only be run one time, that leads to a time complexity in

$$\mathrm{FAR}_1 = N^{\rho_0} \left(\mathcal{S} + \frac{N^{\zeta - \rho_0}}{\sqrt{\epsilon}} \left(\frac{\mathcal{U}}{\sqrt{\delta}} + \mathcal{C} \right) \right). \tag{4.5}$$

Theorem 4.10. *[Bon+23, Appendix C] There exists a quantum algorithm that heuristically solves SVP in time* $2^{0.2563n + o(n)}$.

Proof. Redoing the numerical optimization gives parameters

$$c_\alpha \approx 0.3875 \quad ; \quad c_V \approx 0.27,$$

which gives $\zeta \approx 0.1568$ and $\rho_0 \approx 0.1214$. The costs of the quantum walk become

$$\mathcal{S} \approx N^{0.27} \quad ; \quad \epsilon \approx N^{-0.2} \quad ; \quad \delta \approx N^{-0.27} \quad ; \quad \mathcal{U} = \mathcal{C} = 1.$$

This gives $\mathrm{FAR}_1 \approx N^{0.27}$. Using these parameters in Equation 4.5 gives that the total running time to solve SVP is in $N^{1.2347} = 2^{0.2563n + o(n)}$. □

4.7. Discussion

Impact on lattice-based cryptography. Going from a running time of $2^{0.2653n+o(n)}$ to $2^{0.2563n+o(n)}$ reduces the security claims based on the analysis of the SVP (usually via the BKZ algorithm). For example, if one claims 128 bits of quantum security using the above exponent then one must reduce this claim to 124 bits of quantum security. With the previous claimed security level $2^{128} = 2^{0.265n}$, we have $n \approx 483$, so with the new best exponent we get $2^{0.257n} = 2^{124}$. This can usually be fixed with a slight increase in the parameters but cannot be ignored if one wants to have the same security claims as before.

Parallelization. One thing we have not talked about in this article is whether our algorithm parallelizes well. Algorithm 9 seems to parallelize very well, and we argue that it is indeed the case.

For this algorithm, the best classical algorithm takes $c \to 0$. In this case, placing each $\vec{v} \in L$ in its corresponding α-filters can be done in parallel, and with N processors (or N width) it can be done in time $\mathrm{poly}(n)$. Then, there are N separate instances of FindAllReducing which can be also perfectly parallelized and each one also takes time $\mathrm{poly}(n)$ when $c \to 0$. The while loop is repeated $N^{-\zeta} = N^{0.409n}$ times so the total running time (here depth) is $N^{0.409n+o(n)}$ with a classical circuit of width N. Such a result surpasses already the result from [Bec+16] that achieves depth $N^{1/2}$ with a quantum circuit of width N using parallel Grover search.

In the quantum setting, our algorithm also parallelizes quite well. If we consider our optimal parameters ($c = 0.3696$) with similar reasoning, our algorithm will parallelize perfectly with N^{1-c} processors (so that there is exactly one for each call to FindAllReducing *i.e.* for the quantum random walk). Unfortunately, after that, we do not know how to parallelize well within the quantum walk. When we consider circuits of width N, our optimizations did not achieve better than a depth of $N^{0.409+o(1)}$ which is the classical parallelization. This is also the case if we use Grover's algorithm as in [Laa15] for the FindAllReducing and we use parallel Grover search as in [Bec+16] so best known (classical or quantum) algorithm with lower depth that uses a circuit of width N is the classical parallel algorithm described above.

5. k-Sieves with tailored k-RPC filtering

The work in this chapter has been published in PQCrypto 2023 [CL23] and is a joint work with André Chailloux.

5.1. Overview

***k*-Sieve.** Reducing the memory requirement would make the attack more materially practical, especially when it comes to quantum memory which is very limiting for implementations. A way to reduce memory is by the k-sieve introduced in [BLS16] and then improved by [HK17; HKL18; Kir+19]. The idea is to sum k lattice points instead of pairs at each sieving step in order to find shorter ones. This decreases the number N of lattice points that we need at each step to find the same number N of shorter lattice points. However, this will drastically increase the time to perform the sieving step. The goal is to keep relatively low memory but also try to limit the time.

Configurations. Two main ideas have significantly improved the complexity of k-sieving algorithms: pairwise LSF [Bec+16; HKL18] (explained in Section 3.3) and configurations (Definition 3.26). For $k > 2$, one can replace the reducibility constraint $\|\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k\| \leq 1$ (starting from vectors of norm 1) with constraints of the form $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle \leq C_{i,j}$ for some well-chosen $C_{i,j}$. This is known as the configuration problem. The main advantage is that now we only have constraints on pairs of points instead of on k-tuples, and we can use much more efficient algorithms performing for example pairwise LSF. Searching for unbalanced target configurations increases the memory but in counterpart can reduce the running time.

Contributions. In this chapter, we introduce a new filtering technique tailored for k-sieving and use it to describe new sieving algorithms whose time-memory tradeoffs improve the state-of-the-art in some regimes. We show how to extend the construction of random product codes of [Bec+16] as a means of performing LSF tailored for k-sieving. Our code will also be efficiently decodable and its codewords can be partitioned into subsets $\{\mathbf{A}_1, \ldots, \mathbf{A}_k\}$ each of size k such that $\mathbf{A}_1 + \cdots + \mathbf{A}_k = \vec{\mathbf{0}}$.

While previous k-sieve algorithms start from a configuration problem and then use pairwise LSF; our framework performs the following: we filter the input list of lattice vectors using our tailored code structure to get lists $L_1, \ldots, L_k$ respectively centered around k codewords $\mathbf{A}_1, \ldots, \mathbf{A}_k$ summing to the null-vector. Then, we solve a simpler instance of the configuration problem in the k filtered lists. The k-tuples $(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k) \in L_1 \times \cdots \times L_k$ are more likely to reduce than random k-tuples. Based on this framework, we describe classical sieves for $k = 3$ and 4 that introduce improved time-memory tradeoffs in some regimes. We use the k-Lists algorithm [Kir+19] inside our framework, and this improves the time for $k = 3$ and gives new tradeoffs for $k = 4$. All our quantum algorithms also have the advantage of requiring only a polynomial number of qubits (in the lattice dimension n).

Please refer to figures 5.7 for space-time tradeoffs of classical algorithms and 5.8 for quantum ones.

Outline. Section 5.2 presents a new code structure for the filtering step tailored for k-sieving and as an application we present a new framework to solve SVP. We describe some instances within this framework in the classical model in Section 5.3 and in the quantum model in Section 5.4. We finally discuss the results in Section 5.5. The SageMath code used for the numerical results of this chapter is available here: `https://github.com/johanna-loyer/3-4-sieve`.

5.2. k-RPC and Framework of the k-sieve with tailored filtering

5.2.1. Filtering with k-Tuple Random Product Codes

We recall Definition 3.12 of a Random Product Codes (RPC). We assume $n = m \cdot b$, for $m = \mathcal{O}(\text{polylog}(n))$ and a block size b. The vectors in $\mathbb{R}^n$ are identified with tuples of m vectors in $\mathbb{R}^b$. A random product code $\mathfrak{C}$ of parameters (n, m, B) on subsets of $\mathbb{R}^n$ and of size B^m is defined as a code of the form

$$\mathfrak{C} = Q \cdot (\mathfrak{C}_1 \times \mathfrak{C}_2 \times \cdots \times \mathfrak{C}_m)$$

where Q is a uniformly random rotation over $\mathbb{R}^n$ and the subcodes $\mathfrak{C}_1, \ldots, \mathfrak{C}_m$ are sets of B vectors, sampled uniformly and independently random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$, so that codewords are points of the sphere $\mathcal{S}^{n-1}$. Claim 3.14 states that the code points of a random product code $\mathfrak{C}$ behave like random points of the sphere $\mathcal{S}^{n-1}$. Random product codes can be easily decoded with the algorithm given in Proposition 3.13. So given a point $\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}$, an angle $\alpha \in (0, \pi/2)$ and a random product code $\mathfrak{C}$, one can efficiently compute the set of the M nearest codewords in $\mathfrak{C}$ to $\vec{\boldsymbol{x}}$, and this takes a time proportional to M.

Each codeword of the RPC constitutes the center of a *filter* (See Definition 3.15). For the 2-sieve with Locality Sensitive Filtering, one inserts each list vector into its nearest filters, and then for each vector one searches a reducing one within its filters. It provides the current best algorithms to solve SVP both classically [Bec+16] (Algorithm 6) and quantumly as we saw in the previous chapter (Algorithm 9). However, the k-sieve for $k > 2$ searches k-tuples such that $\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k$ is reduced. So, searching within one unique filter does not permit one to quickly find a solution without having to check a lot of non-reducing elements. So we will slightly modify the construction of the random product code to take into account a configuration.

***k*-Tuple Random Product Code (k-RPC).** We start with the case $k = 3$ to describe our k-RPC construction. Instead of constructing fully random codes $\mathfrak{C}_1, \ldots, \mathfrak{C}_m$, we will construct random codes $\mathfrak{C}_i$ with the following property:

$$\forall \mathbf{A}_1 \in \mathfrak{C}_i, \exists \mathbf{A}_2, \mathbf{A}_3 \in \mathfrak{C}_i \ st. \ \mathbf{A}_1 + \mathbf{A}_2 + \mathbf{A}_3 = \vec{\mathbf{0}}.$$

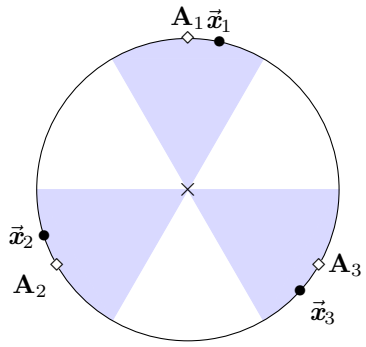


Figure 5.1: Each k-RPC codeword belongs to a k-tuple $(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ that sums to $\vec{\mathbf{0}}$. Each codeword $\mathbf{A}_i$ is the center of a filter $f_{\mathbf{A}_i}$, that will allow during a sieving step to search a reducing tuple $(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k)$ within the tuple-filter $f_{\mathbf{A}_1} \times \cdots \times f_{\mathbf{A}_k}$.

We assume $n = m \cdot b$ for $m = \mathcal{O}(\text{polylog}(n))$ and a block size b. The vectors in $\mathbb{R}^n$ will be identified with tuples of m vectors in $\mathbb{R}^b$. An k-Tuple Random Product Code $\mathfrak{C}$ of parameters (n, m, B) is defined as a code of the form

$$\mathfrak{C} = Q \cdot (\mathfrak{C}_1 \times \mathfrak{C}_2 \times \cdots \times \mathfrak{C}_m)\,.$$

where Q is a uniformly random rotation over $\mathbb{R}^n$ and the subcodes $\mathfrak{C}_1, \ldots, \mathfrak{C}_m$ are each constructed as follows:

1. Pick $B/3$ random vectors $\mathbf{A}_1^1, \ldots, \mathbf{A}_1^{B/3}$ sampled uniformly at random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$.
2. For each $i \in [B/3]$, pick a random vector $\mathbf{A}_2^j$ sampled uniformly at random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$ with the condition $\langle \mathbf{A}_1^j | \mathbf{A}_2^j \rangle = -\frac{1}{2m}$.

3. For each $i \in [B/3]$, let $\mathbf{A}_3^j$ be the unique point on the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$ st. $\mathbf{A}_1^j + \mathbf{A}_2^j + \mathbf{A}_3^j = \vec{\mathbf{0}}$. The code $\mathfrak{C}$ is then the set of points $\{\mathbf{A}_1^1, \mathbf{A}_2^1, \mathbf{A}_3^1, \ldots, \mathbf{A}_1^{B/3}, \mathbf{A}_2^{B/3}, \mathbf{A}_3^{B/3}\}$.

Notice that $\mathbf{A}_3^j = -(\mathbf{A}_1^j + \mathbf{A}_2^j)$ is of the correct norm $\frac{1}{\sqrt{m}}$. Indeed,

$$\|\mathbf{A}_3^j\|^2 = \| - (\mathbf{A}_1^j + \mathbf{A}_2^j)\|^2 = \|\mathbf{A}_1^j\|^2 + \|\mathbf{A}_2^j\|^2 + 2\langle \mathbf{A}_2^j | \mathbf{A}_1^j \rangle = \frac{2}{m} - \frac{2}{2m} = \frac{1}{m}.$$

We can generalize this construction for any constant k to get a k-RPC of codewords $\{\mathbf{A}_i^j\}_{i\in[k], j\in[B/3]}$ such that

$$\forall \mathbf{A}_1 \in \mathfrak{C}_i, \exists \mathbf{A}_2, \ldots \mathbf{A}_k \in \mathfrak{C}_i \ st. \ \sum_{i=1}^{k} \mathbf{A}_i = \vec{\mathbf{0}}.$$

1. Pick B/k random vectors $\mathbf{A}_1^1, \ldots, \mathbf{A}_1^{B/k}$ sampled uniformly at random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$.

2. For each $j \in [B/k]$, pick a random vector $\mathbf{A}_2^j$ sampled uniformly at random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$ with the condition $\langle \mathbf{A}_1^j | \mathbf{A}_2^j \rangle = -\frac{1}{(k-1)m}$. Then, for $i \in [|2, k-1|]$, pick random vectors $\mathbf{A}_i^j$ such that for each previous $i' \in [i]$, $\langle \mathbf{A}_i^j | \mathbf{A}_{i'}^j \rangle = -\frac{1}{(k-1)m}$.

3. For each $j \in [B/k]$, let $\mathbf{A}_k^j$ be the unique point on the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$ such that $\sum_{i=1}^{k} \mathbf{A}_i^j = \vec{\mathbf{0}}$. The code $\mathfrak{C}$ is then the set of points $\{\mathbf{A}_i^j\}_{i\in[k], j\in[B/3]}$.

As before, we can check that $\mathbf{A}_k^j = -\sum_{j=1}^{k-1} \mathbf{A}_i^j$ is of the correct norm $\frac{1}{\sqrt{m}}$:

$$\begin{aligned}\|\mathbf{A}_k^j\|^2 &= \sum_{j=1}^{k-1} \|\mathbf{A}_i^j\|^2 + \sum_{i=1}^{k-1} \sum_{\substack{i'=1 \\ i' \neq i}}^{k-1} \langle \mathbf{A}_i^j | \mathbf{A}_{i'}^j \rangle \\ &= \frac{k-1}{m} + (k-1)(k-2) \cdot \frac{-1}{(k-1)m} = \frac{k-1}{m} - \frac{k-2}{m} = \frac{1}{m}.\end{aligned}$$

For each $j \in [B/k]$, we actually take $\langle \mathbf{A}_i^j | \mathbf{A}_{i'}^j \rangle = -1/(k-1)$ for $i \neq i'$, because this balanced configuration optimizes the number of k-tuples whose vectors are respectively close to the centers $\mathbf{A}_i^j$ (See Proposition 3.28).

Proposition 5.1. *Let $\mathfrak{C}$ be a random product code with triangles of parameters (n, m, B) with $m = \log(n)$ and $B^m = N^{O(1)}$. For any $\vec{\boldsymbol{x}} \in \mathcal{S}^{n-2}$ and angle α, one can compute $\mathcal{H}_{\vec{\boldsymbol{x}},\alpha} \cap \mathfrak{C}$ in time $N^{o(1)} \cdot |\mathcal{H}_{\vec{\boldsymbol{x}},\alpha} \cap \mathfrak{C}|$.*

Proof. The decoding algorithm of Proposition 3.13 uses only the product structure of the code and not how the codes $\mathfrak{C}_1, \ldots, \mathfrak{C}_m$ are constructed. The same algorithm will therefore also efficiently decode k-tuple random product codes. □

We recall Definition 3.15: a hypercone filter $f_{\boldsymbol{s}}$ of center $\boldsymbol{s}$ and angle α is a set that can be filled with vectors of angle at most α with $\boldsymbol{s}$.

Definition 5.2 (Tuple-filter)**.** Let $\mathfrak{C}$ be a k-RPC with codewords $(\mathbf{A}_i^j)$ for $i \in [k]$ and $j \in [|\mathfrak{C}|/k]$ such that $\forall j \in [|\mathfrak{C}|/k], \sum_{i=1}^{k} \mathbf{A}_i^j = \vec{\mathbf{0}}$. Given angle α and some j, we call $(f_{\mathbf{A}_1^j}, \ldots, f_{\mathbf{A}_k^j})$ a tuple-filter, where each $f_{\mathbf{A}_i^j}$ is a filter.

***k*-RPC filtering.** We are given a list L of lattice vectors assumed to be *i.i.d.* uniformly random over $\mathcal{S}^{n-1}$. We choose an angle α and sample a k-RPC $\mathfrak{C}$ containing $1/\mathcal{V}(\alpha)$ codewords, each one being the center of a filter. Choosing the code size at $1/\mathcal{V}(\alpha)$ makes that any point $\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}$ is on average at angle α to its nearest codeword. For each vector in the list L, we decode it to its nearest unique codeword $\mathbf{A} \in \mathfrak{C}$. This step, called prefiltering, separates L into disjoint sublists, each one of size $|L| \cdot \mathcal{V}(\alpha)$. We focus on only one

tuple of filters of centers $\mathbf{A}_1, \dots, \mathbf{A}_k$ respectively associated to the lists $L_1, \dots, L_k$, that are empty at the beginning. By Claim 3.16, the angle between any $\vec{\boldsymbol{x}} \in L_i$ and $\mathbf{A}_i$ is α with high probability. We simplify by considering that it is always the case. So for $\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1}$ of angle α with a center of filter $\mathbf{A}$, we can write for some $\vec{\boldsymbol{y}} \perp \mathbf{A}$,

$$\vec{\boldsymbol{x}} = \cos(\alpha)\mathbf{A} + \sin(\alpha)\vec{\boldsymbol{y}}.$$

We call $\vec{\boldsymbol{y}}$ the residual vector of $\vec{\boldsymbol{x}}$ on the filter of center $\mathbf{A}$ and angle α. While filling the list L_i with the $\vec{\boldsymbol{x}}$, we fill in parallel a list R_i with their residual vectors $\vec{\boldsymbol{y}}$ in its filter of center $\mathbf{A}_i$. Note that the points in L_i are *i.i.d.* uniformly random over the (n-1)-dimensional sphere $\{\vec{\boldsymbol{x}} \in \mathbb{R}^n : \|\vec{\boldsymbol{x}}\| = 1, \theta(\vec{\boldsymbol{x}}, \mathbf{A}_i) = \alpha\}$, which is isometric to the sphere $\mathcal{S}^{n-2}$ on which the residual vectors are *i.i.d.* uniformly random.

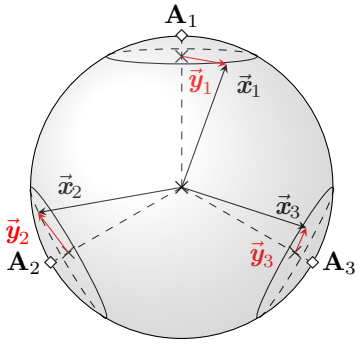


Figure 5.2: List vectors $\vec{\boldsymbol{x}}_i \in L_i$ in their filters of centers $\mathbf{A}_i$ and angle α and their respective residual vectors $\vec{\boldsymbol{y}}_i \in R_i$. Attention: For the sake of readability, red arrows represent the non-normalized vectors $\sin(\alpha)\vec{\boldsymbol{y}}_i$. (Also see Figure 3.6 that illustrates Claim 3.16). Please keep in mind that this scheme is in dimension $n = 3$ and that in high dimensions the volume ratios are exacerbated.

The following lemmas give us the equivalence between the problem of searching a tuple of list vectors $\vec{\boldsymbol{x}}$'s satisfying a configuration C and searching a tuple of residual vectors $\vec{\boldsymbol{y}}$'s satisfying a configuration C', depending on C and the angle α of the filters.

Lemma 5.3. *Using the above notations for the lists L_i's and R_i's, a k-tuple $(\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_k) \in L_1 \times \cdots \times L_k$ is reducing iff. their residual vectors $(\vec{\boldsymbol{y}}_1, \dots, \vec{\boldsymbol{y}}_k) \in R_1 \times \cdots \times R_k$ satisfy*

$$\sum_{1 \leqslant i < j \leqslant k} \langle \vec{\boldsymbol{y}}_i | \vec{\boldsymbol{y}}_j \rangle \leqslant \frac{1 - k\cos^2(\alpha)}{2\sin^2(\alpha)} := I_k(\alpha). \tag{5.1}$$

Proof. For $\vec{\boldsymbol{x}}_i \in L_i$, we have $\vec{\boldsymbol{x}}_i = \cos(\alpha)\mathbf{A}_i + \sin(\alpha)\vec{\boldsymbol{y}}_i$. Claim 3.16 ensures the randomness of the points $\vec{\boldsymbol{y}}_i$ in the sphere $\mathcal{S}^{n-2}$, so we have for $i \neq j$, $\langle \mathbf{A}_i | \vec{\boldsymbol{y}}_j \rangle \approx 0$ with high probability. We consider the equality for simplicity. For $i \neq j$, $\vec{\boldsymbol{x}}_i \in L_i$ and $\vec{\boldsymbol{x}}_j \in L_j$, we obtain:

$$\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle = \cos^2(\alpha)\langle \mathbf{A}_i | \mathbf{A}_j \rangle + \sin^2(\alpha)\langle \vec{\boldsymbol{y}}_i | \vec{\boldsymbol{y}}_j \rangle. \tag{5.2}$$

Then we have

$$\begin{aligned} \left\| \sum_{i=1}^{k} \vec{\boldsymbol{x}}_i \right\|^2 &= \left\| \sum_{i=1}^{k} \cos(\alpha)\mathbf{A}_i + \sin(\alpha)\vec{\boldsymbol{y}}_i \right\|^2 = \left\| \sum_{i=1}^{k} \sin(\alpha)\vec{\boldsymbol{y}}_i \right\|^2 \quad \text{as } \sum_{i=1}^{k} \mathbf{A}_i = \vec{\mathbf{0}} \\ &= k\sin^2(\alpha) + 2\sin^2(\alpha) \left(\sum_{1 \leqslant i < j \leqslant k} \langle \vec{\boldsymbol{y}}_i | \vec{\boldsymbol{y}}_j \rangle \right). \end{aligned} \tag{5.3}$$

In the case the tuple $(\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_k)$ is reducing, we have $\left\| \sum_{i=1}^{k} \vec{\boldsymbol{x}}_i \right\|^2 \leqslant 1$, hence the wanted result. Notice also that we can translate the norm condition $\left\| \sum_{i=1}^{k} \vec{\boldsymbol{x}}_i \right\|^2 \leqslant 1$ directly into a norm condition of the residual vectors $\left\| \sum_{i=1}^{k} \vec{\boldsymbol{y}}_i \right\|^2 \leqslant \frac{1}{\sin^2(\alpha)}$. □

Lemma 5.4. *Let $C \in \mathbb{R}^{k\times k}$ be a configuration, and α an angle. If a k-tuple $\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k$ satisfies C then their residual vectors $\vec{\boldsymbol{y}}_1, \ldots, \vec{\boldsymbol{y}}_k$ on a filter of angle α satisfies the configuration $C'(\alpha)$ with for $i \neq j$,*

$$C'_{i,j}(\alpha) = -\frac{1}{\sin^2(\alpha)}\left(C_{i,j} + \frac{\cos^2(\alpha)}{k-1}\right)$$

Proof. As already justified in the previous proof (Equation 5.2), for $i \neq j$ we have $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle = \cos^2(\alpha)\langle \mathbf{A}_i | \mathbf{A}_j \rangle + \sin^2(\alpha)\langle \vec{\boldsymbol{y}}_i | \vec{\boldsymbol{y}}_j \rangle$. The configuration C gives constraints over the scalar products $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle$, and $C'(\alpha)$ over the scalar products $\langle \vec{\boldsymbol{y}}_i | \vec{\boldsymbol{y}}_j \rangle$; and $\langle \mathbf{A}_i | \mathbf{A}_j \rangle$ is fixed at $-1/(k-1)$. Rewriting the above equation with it gives $C_{i,j} = \cos^2(\alpha) \cdot \frac{-1}{k-1} + \sin^2(\alpha) C'_{i,j}$. Hence the result. □

If we consider a balanced configuration C for the $\vec{\boldsymbol{x}}_i$'s, then we have for $i \neq j$, $C_{i,j} = -1/k$ all equal. For residual vectors on a filter of angle α, this also implies $C'_{i,j}(\alpha)$ all equal for $i \neq j$. There are $C'_{i,j}$ at number $\sum_{i=1}^{k-1} i = k \cdot (k-1)/2$. Thus for $i \neq j$ we will have $C'_{i,j}(\alpha) = \frac{2}{k\cdot(k-1)} \cdot I_k(\alpha)$, with $I_k(\alpha)$ as defined in Lemma 5.3.

As $I_k(\alpha) \geqslant -1$, the constraints over the vectors $\vec{\boldsymbol{y}}_i$'s are relaxed in comparison with the input vectors $\vec{\boldsymbol{x}}_i$. This allows more flexibility to choose a configuration to reduce the running time, and this is the intuition behind why the prefiltering step allows finding solutions more easily.

5.2.2. Framework adapted for the k-sieve

The idea behind the framework of our sieving algorithms remains the same as in Chapter 4 but is generalized for the k-sieve algorithm for any $k \geq 2$:

1. Prefilter the list vectors with a k-RPC,
2. Search all reduced tuples within each filter,
3. Repeat steps 1. and 2. until all the reduced points are found.

Algorithm 12 k-Sieve Framework with LSF prefiltering

Require: List L of lattice vectors of norm at most 1 ; reducing factor $\gamma < 1$.
Parameters: $k \in \mathbb{N}$; angle $\alpha \in (0, \pi/2]$; target configuration C.
Ensure: List L_{out} of lattice vectors of norm at most γ.
$L_{out} = \emptyset$
while $|L_{out}| < |L|$ **do** ▷ $\mathrm{NbRep}_{\alpha,C}$ repeats
  Sample a k-RPC code $\mathfrak{C}$ of size $k \cdot 1/\mathcal{V}(\alpha)$ ▷ Its codewords will be denoted $\mathbf{A}_i^j$
  Initialize the lists $L_i^j = \emptyset$ **for** $i \in [k], j \in [|\mathfrak{C}|/k]$
  for each $\vec{\boldsymbol{x}} \in L$ **do**
    $\mathbf{A}_i^j \leftarrow \mathbf{Decode}(\vec{\boldsymbol{x}}, \mathfrak{C})$ ▷ Algorithm from Proposition 3.13.
    $\vec{\boldsymbol{y}} \leftarrow \left(\vec{\boldsymbol{x}} - \cos(\alpha)\mathbf{A}_i^j\right)/\sin(\alpha)$ ▷ Residual vector of $\vec{\boldsymbol{x}}$ in the filter of center $\mathbf{A}_i^j$
    $L_i^j \leftarrow L_i^j \cup \{\vec{\boldsymbol{x}}\}$; $R_i^j \leftarrow R_i^j \cup \{\vec{\boldsymbol{y}}\}$
  for each tuple-filter numbered $j \in [|\mathfrak{C}|/k]$ **do**
    $Sol_{\vec{\boldsymbol{y}}} \leftarrow \mathsf{FindAllReducing}\left((R_i^j)_i, C'(\alpha)\right)$ ▷ Find all $(\vec{\boldsymbol{y}}_i)_i \in R_1^j \times \cdots \times R_k^j$ satisfying $C'(\alpha)$
    $Sol_{\Sigma\vec{\boldsymbol{x}}} \leftarrow \left\{\sum_{i=1}^k \vec{\boldsymbol{x}}_i : (\vec{\boldsymbol{y}}_i)_i \in Sol_{\vec{\boldsymbol{y}}}\right\}$ ▷ $\vec{\boldsymbol{x}}_i \in L_i^j$ and $\vec{\boldsymbol{y}}_i \in R_i^j$ share the same index i
    $L_{out} \leftarrow L_{out} \cup Sol_{\Sigma\vec{\boldsymbol{x}}}$
return L_{out}

Complexity analysis of the framework algorithm 12

Heuristic and simplifying assumptions. We remind the reader that the complexity analysis of the algorithms presented in this chapter relies on the following assumptions:

- **Heuristic 3.8**. The input lattice points are uniformly randomly distributed on the sphere $\mathcal{S}^{n-1} := \{\vec{\boldsymbol{x}} \in \mathbb{R}^n : \|\vec{\boldsymbol{x}}\| = 1\}$.
- **Claim 3.14**. The points of a random product code are indistinguishable from random independent points in $\mathcal{S}^{n-1}$.
- **Claim 3.16**. Given a point $\vec{\boldsymbol{s}} \in \mathcal{S}^{n-1}$, we assume that a random vector $\vec{\boldsymbol{x}}$ of angle at most α with $\vec{\boldsymbol{s}}$ is exactly at angle α, and then can be decomposed $\vec{\boldsymbol{x}} = \cos(\alpha)\vec{\boldsymbol{s}} + \sin(\alpha)\vec{\boldsymbol{y}}$ with $\vec{\boldsymbol{y}} \perp \vec{\boldsymbol{s}}$ and $\|\vec{\boldsymbol{y}}\| = 1$. The residual vector $\vec{\boldsymbol{y}}$ is random uniform in $\mathcal{S}^{n-2}$ an orthogonal complement to $\vec{\boldsymbol{s}}$.

Proposition 3.9 has given the volume of the spherical cap of center $\boldsymbol{s} \in \mathcal{S}^{n-1}$, which is $\mathcal{V}_n(\alpha) := \{\vec{\boldsymbol{x}} \in \mathcal{S}^{n-1} \ : \ \theta(\boldsymbol{s}, \vec{\boldsymbol{x}}) \leq \alpha\} = \mathrm{poly}(n) \cdot \sin^{n/2}(\alpha)$. Notice that when we work with residual vectors in $\mathcal{S}^{n-2}$ the sphere in $\mathbb{R}^{n-1}$, we would write $\mathcal{V}_{n-1}$. But since $\mathcal{V}_n$ and $\mathcal{V}_{n-1}$ are asymptotically equivalent, we will just be writing $\mathcal{V}$ for simplicity of notations.

1. Prefiltering (lines 4-9). We start by sampling a k-RPC $\mathfrak{C}$ (Defined in Part 5.2.1) of size $k \cdot 1/\mathcal{V}(\alpha)$. Its codewords are denoted $(\mathbf{A}_i^j)_{i,j}$ for $i \in [k]$ and $j \in [|\mathfrak{C}|/k]$ (we suppose these values are integers by simplicity). For a fixed $j \in [|\mathfrak{C}|/k]$ and for $i_1 \neq i_2 \in [k]$ we have $\langle \mathbf{A}_{i_1}^j | \mathbf{A}_{i_2}^j \rangle = -\frac{1}{k-1}$, that implies $\sum_{i=1}^k \mathbf{A}_i^j = \vec{\mathbf{0}}$.

Once the code is sampled, we can start the so-called prefiltering step. For each vector $\vec{\boldsymbol{x}} \in L$, we efficiently compute its nearest codeword in $\mathfrak{C}$ using the algorithm from Proposition 5.1. If it returns center $\mathbf{A}_i^j$, then we add $\vec{\boldsymbol{x}}$ to its associated list L_i^j. We also compute $\vec{\boldsymbol{x}}$'s residual vector $\vec{\boldsymbol{y}} = \left(\vec{\boldsymbol{x}} - \cos(\alpha)\mathbf{A}_i^j\right)/\sin(\alpha)$ (by Claim 3.16) and we add it to list R_i^j. Given a residual vector in R_i^j, we will be able to recover its corresponding vector in L_i^j by just looking at the same index.

There are tuple-filters $(\mathbf{A}_i^j)_{i \in [k]}$ at number

$$\mathrm{NbFilters} := |\mathfrak{C}|/k = \mathcal{O}\left(\frac{1}{\mathcal{V}(\alpha)}\right). \tag{5.4}$$

As we compute the nearest filter in amortized time $\mathcal{O}(1)$ for each vector in L, the prefiltering step takes time $|L|$.

2. Find all solutions within a tuple-filter (lines 10-13). We started with a list L and we wanted to solve a configuration problem, and after the prefiltering step, we can consider easier instances of the configuration problem on the sublists of L. The subroutine FindAllReducing solves one of these instances at a time, and we run it over each of the $1/\mathcal{V}(\alpha)$ tuple-filters.

Let's fix some $j \in [|\mathfrak{C}|/k]$ and consider the instance of a configuration problem on the k lists $(R_i^j)_i$ with configuration $C'(\alpha)$. The subroutine then has to find all the k-tuples within $R_1^j \times \cdots \times R_k^j$ that satisfies the configuration $C'(\alpha)$. As we focus on only one filter at a time, in the following we will no longer write the j in exponent to lighten the notations.

The number of solutions the subroutine has to return is given by the following lemma.

Lemma 5.5. *With the same notations as before and for fixed $j \in [|\mathfrak{C}|/k]$, the expected number of tuples in the tuple-filter associated with the lists $R_1 \times \cdots \times R_k$ satisfying configuration $C'(\alpha)$ is on average*

$$|Sol_f| = \mathcal{O}\left(|R_1|^k \cdot \det(C'(\alpha))^{n/2}\right) = \mathcal{O}\left(|L|^k \mathcal{V}(\alpha)^k \cdot \det\left(C'(\alpha)\right)^{n/2}\right).$$

Proof. There are $|R_1|^k$ tuples in $R_1 \times ... \times R_k$ as the lists are all of same size $|R_1| = |L| \cdot \mathcal{V}(\alpha)$. Any tuple $(\vec{\boldsymbol{y}}_1, ..., \vec{\boldsymbol{y}}_k)$ from this set has probability $\det(C'(\alpha))^{n/2}$ to satisfy configuration $C'(\alpha)$. Hence the expected number of tuples satisfying $C'(\alpha)$. □

Any subroutine with these inputs and outputs may suit the framework. For example, in the case $k = 2$, we described in Section 4 a 2-sieve fitting this framework, where the subroutine uses quantum random walks to find the reducing pairs of vectors. We denote the time complexity of the subroutine FindAllReducing with parameters α and C by $T(\mathrm{FAR}_{C'(\alpha)})$.

3. Number of repeats (while loop 3). After searching all the solutions within every tuple-filters, by Theorem 3.28 we expect to find the following number of solutions:

$$|Sol_{all}| = |L|^k \cdot \det(C)^{n/2}. \tag{5.5}$$

To complete the sieve step, we are required to find $|L|$ reduced lattice vectors. Thus steps 1. and 2. have to be repeated until enough solutions have been found. The missed solutions are the ones such that a part of the solution is in one tuple-filter and the rest is in another. By doing a new prefiltering, it changes the partitions of the sphere, and this allows us to find some of these missing solutions.

Lemma 5.6. *The number of repetitions in the while loop is*

$$\begin{aligned} NbRep_{\alpha,C} &= \mathcal{O}\left(\max\left\{1, \frac{|Sol_{all}|}{|Sol_f| \cdot NbFilters}\right\}\right) \\ &= \mathcal{O}\left(\max\left\{1, \frac{|L_1|^k \det(C)^{n/2}}{|L_1|^k \mathcal{V}^k(\alpha) \det\left(C'(\alpha)\right)^{n/2} \cdot \frac{1}{\mathcal{V}(\alpha)}}\right\}\right) \\ &= \mathcal{O}\left(\max\left\{1, \frac{\det(C)^{n/2}}{\mathcal{V}^{k-1}(\alpha) \det\left(C'(\alpha)\right)^{n/2}}\right\}\right). \end{aligned}$$

The overall time complexity of an algorithm based on this framework is given in the following theorem.

Theorem 5.7. *Let $\alpha \in (0, \pi/2]$ be an angle and a configuration $C \in \mathbb{R}^{k\times k}$, and $C'(\alpha)$ the configuration on the residual vectors (See Lemma 5.4). Given an algorithm that solves the configuration problem $C'(\alpha)$ for k lists in time $T(\mathrm{FAR}_{C'(\alpha)})$, Algorithm 12 solves SVP in time*

$$T(k\text{-}sieve) := \ \mathrm{NbRep}_{\alpha,C} \cdot \left(|L| + \mathrm{NbFilters}_\alpha \cdot T(\mathrm{FAR}_{C'(\alpha)})\right)$$

where $\mathrm{NbRep}_{\alpha,C}$ *is given by Lemma 5.6 and* $\mathrm{NbFilters}_\alpha = \mathcal{O}\left(\frac{1}{\mathcal{V}(\alpha)}\right)$ *by Equation 5.4.*

The above theorem is the main technical contribution of our work. The main novelty is the angle α which can be freely chosen. Taking an angle $\alpha = \pi/2$ means that we do not perform any tailored LSF.

Optimization of the parameters. The $C_{i,j}$'s are parameters to optimize to get the minimal overall time of the k-sieve, and they obey the constraints on memory and reduceness of the tuples. We also require that the inner algorithm for solving the configuration problem with $C'(\alpha)$ uses at most memory M. There is also the prefiltering angle $\alpha \in (0, \pi/2]$ that has to be optimized. In the next sections, we will present algorithms that fit in the framework 12 and for each one we will specify the optimal values for C and α we have obtained by numerical optimization. The code is available on `https://github.com/johanna-loyer/3-4-sieve`.

5.3. Classical k-sieves

We use Theorem 5.7 to know the overall complexity of the k-sieve, so the only thing to explicit is the inner algorithm running in time $T(\mathrm{FAR}_{C'(\alpha)})$ as well as the parameters C and α. This subroutine has to solve a configuration problem in the input lists $L_1, \ldots, L_k$ for the configuration $C = (C_{i,j})_{i,j\in[k]} \in \mathbb{R}^{k\times k}$. The subroutine has to find a $1 - o(1)$ fraction of the k-tuples $(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k) \in L_1 \times \cdots \times L_k$ such that $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle \leq C_{i,j}$ for all $i \neq j$ for some $\epsilon > 0$. By Lemma 5.4, we can solve this problem by solving $R_1, \ldots, R_k$ for configuration $C'(\alpha)$. We present here our 3-sieve and 4-sieve classical algorithms. Actually, in both cases, the inner algorithm will use a classical 2-sieve algorithm so we first give formulas for the configuration problem with $k = 2$.

5.3.1. Classical 2-sieve

We present here the best-known algorithm for classical 2-sieve. While these are known results, we will need this analysis for our 3-sieve and 4-sieve algorithms.

Proposition 5.8. *Take $k = 2$, lists L_1, L_2 of random points of norm 1 with $|L_1| = |L_2|$, a target configuration $C = \begin{pmatrix} 1 & C_{12} \\ C_{12} & 1 \end{pmatrix}$. Let α st. $\mathcal{V}(\alpha) = \frac{1}{|L_1|}$. Algorithm 12 with parameter α constructs a list L_{out} of pairs of points $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2)$ with $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \in L_1 \times L_2$ such that $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle \leq C_{12}$, in time T using memory M where*

$$\begin{aligned} |L_{out}| &= \mathcal{O}\left(|L_1|^2 \det(C)^{n/2}\right) = \mathcal{O}\left(|L_1|^2 \left(1 - C_{12}^2\right)^{n/2}\right) \\ T &= \mathcal{O}\left(|L_1|^2 \frac{\det(C)^{n/2}}{\det\left(C'(\alpha)\right)^{n/2}}\right) = \mathcal{O}\left(|L_1|^2 \frac{(1 - C_{12}^2)^{n/2}}{(1 - C'_{12}(\alpha)^2)^{n/2}}\right). \\ M &= \mathcal{O}\left(\max\{|L_1|, |L_{out}|\}\right) \end{aligned}$$

where recall that $C'_{12}(\alpha) = \frac{1}{\sin^2(\alpha)} \cdot \left(C_{12} + \cos^2(\alpha)\right)$. Notice that $|L_{out}|$ corresponds asymptotically to all the pairs $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \in L_1 \times L_2$ such that $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle \leq C_{12}$ so we find here asymptotically all solutions.

Proof. We use Theorem 5.7 with $k = 2$ and some parameter α to get

$$T = \mathcal{O}\left(\mathrm{NbRep}_{\alpha, C_{12}} \cdot \left(|L_1| + \frac{1}{\mathcal{V}(\alpha)} \cdot T(\mathrm{FAR}_{C'_{12}(\alpha)}) \right) \right). \tag{5.6}$$

Recall that here, $\mathrm{FAR}_{C'_{12}(\alpha)}$ computes the running time of finding all solution pairs with inner product smaller than $C'_{12}(\alpha)$ when starting with lists of size $|R_1| = |L_1|\mathcal{V}(\alpha)$. We perform an exhaustive search on the pairs of points to find all solutions so

$$T(\mathrm{FAR}_{C'_{12}(\alpha)}) = \mathcal{O}\left(\max\{1, |L_1|^2 \mathcal{V}^2(\alpha)\}\right).$$

We take α such that $\mathcal{V}(\alpha) = \frac{1}{|L_1|}$ so Equation 5.6 becomes $T = \mathcal{O}\left(\mathrm{NbRep}_{\alpha, C_{12}} \cdot |L_1| \right)$. Finally, from Lemma 5.6, we have

$$\mathrm{NbRep}_{\alpha, C_{12}} = \mathcal{O}\left(\max\left\{ 1, \frac{\det(C)^{n/2}}{\mathcal{V}^{k-1}(\alpha) \det\left(C'(\alpha)\right)^{n/2}} \right\} \right) = |L_1| \cdot \frac{\det(C)^{n/2}}{\det\left(C'(\alpha)\right)^{n/2}},$$

which allows us to conclude that

$$T = \mathcal{O}\left(|L_1|^2 \frac{\det(C)^{n/2}}{\det\left(C'(\alpha)\right)^{n/2}} \right).$$

□

As a special case, we can take $|L_1| = |L_2| = 2^{0.2075n}$, $C_{12} = -1/2$ which gives $L_{out} = |L_1|$, $T = 2^{0.292n}$ and $M = |L_1|$. This is the exact same complexity as [Bec+16], the best known classical algorithm asymptotically, that actually fits our framework.

5.3.2. Classical 3-sieve

We now consider the case of $k = 3$. Our subroutine will construct the following intermediate lists:

1. Construct $L_{12} = \{(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \in L_1 \times L_2 : \langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle \leq C_{12}\}$ and $L_{13} = \{(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_3) \in L_1 \times L_3 : \langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_3 \rangle \leq C_{23}\}$.
2. For each $\vec{\boldsymbol{x}}_1 \in L_1$, let $L_{12}(\vec{\boldsymbol{x}}_1) = \{\vec{\boldsymbol{x}}_2 \in L_2 : (\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \in L_{12}\}$ and $L_{13}(\vec{\boldsymbol{x}}_1) = \{\vec{\boldsymbol{x}}_3 \in L_3 : (\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_3) \in L_{13}\}$.
3. For each $\vec{\boldsymbol{x}}_1 \in L_1$, compute $L_{123}(\vec{\boldsymbol{x}}_1) = \{(\vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3) \in L_{12}(\vec{\boldsymbol{x}}_1) \times L_{13}(\vec{\boldsymbol{x}}_1) : \langle \vec{\boldsymbol{x}}_2 | \vec{\boldsymbol{x}}_3 \rangle \leq C_{23}\}$. For each $\vec{\boldsymbol{x}}_1 \in L_1$, triples $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3)$ are solutions when $(\vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3) \in L_{123}(\vec{\boldsymbol{x}}_1)$.

Now that we defined all intermediate lists, we can write the algorithm we use for solving the inner configuration problem with $k = 3$.

Algorithm 13 FindAllReducing classical 3-sieve

Require: lists L_1, L_2, L_3 of vectors i.i.d. in $\mathcal{S}^{n-1}$ with $|L_1| = |L_2| = |L_3|$; target configuration $C \in \mathbb{R}^{3\times 3}$.
Ensure: list L_{out} of all 3-tuples in $L_1 \times L_2 \times L_3$ satisfying configuration C.
$L_{out} := \emptyset$.
construct L_{12} and L_{13} using a 2-sieve algorithm with angle parameter α', from which you can recover lists $L_{12}(\vec{\boldsymbol{x}}_1)$ and $L_{13}(\vec{\boldsymbol{x}}_1)$
for each $\vec{\boldsymbol{x}}_1 \in L_1$:
compute $L_{123}(\vec{\boldsymbol{x}}_1)$ using a 2-sieve algorithm with angle parameter α''
for each $(\vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3) \in L_{123}(\vec{\boldsymbol{x}}_1)$, do $L_{out} := L_{out} \cup \{(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3)\}$.
return L_{out}

Complexity of Algorithm 13.

Construction of the lists L_{12} and L_{13}. As a direct consequence of Proposition 5.8, we have:

Lemma 5.9. *Let T_{12} (resp. T_{13}) be the time to compute L_{12} (resp. L_{13}). Let α such that $|L_1| = 1/\mathcal{V}(\alpha)$. We have*

$$T_{12} = \mathcal{O}\left(|L_1|^2 \frac{\left(1-C_{12}^2\right)^{n/2}}{\left(1-C'_{12}(\alpha)^2\right)^{n/2}}\right)$$
$$T_{13} = \mathcal{O}\left(|L_1|^2 \frac{\left(1-C_{13}^2\right)^{n/2}}{\left(1-C'_{13}(\alpha)^2\right)^{n/2}}\right)$$

Construction of the lists $L_{23}(\vec{\boldsymbol{x}}_1)$. For a fixed $\vec{\boldsymbol{x}}_1$, notice that the lists $L_2(\vec{\boldsymbol{x}}_1)$ and $L_3(\vec{\boldsymbol{x}}_1)$ do not contain points uniformly distributed on the sphere since they have an inner-product constraint with $\vec{\boldsymbol{x}}_1$ so we cannot apply Proposition 5.8 directly. Fix $\vec{\boldsymbol{x}}_1 \in L_1$ and let $\vec{\boldsymbol{x}}_2 \in L_2$ and $\vec{\boldsymbol{x}}_3 \in L_3$. For simplicity of calculations, we consider the case where $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle = C_{12}, \langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_3 \rangle = C_{13}$ and $\langle \vec{\boldsymbol{x}}_2 | \vec{\boldsymbol{x}}_3 \rangle = C_{23}$. This approximation is justified from Heuristic 3.8. So we write

$$\vec{\boldsymbol{x}}_2 = C_{12}\vec{\boldsymbol{x}}_1 + \sqrt{1-C_{12}^2}\vec{\boldsymbol{y}}_2 \quad ; \quad \vec{\boldsymbol{x}}_3 = C_{13}\vec{\boldsymbol{x}}_1 + \sqrt{1-C_{13}^2}\vec{\boldsymbol{y}}_3 \tag{5.7}$$

where $\vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3$ are orthogonal to $\vec{\boldsymbol{x}}_1$ and of norm 1. Also, if $\vec{\boldsymbol{x}}_2$ (resp. $\vec{\boldsymbol{x}}_3$) is a random vector satisfying $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle = C_{12}$ (resp. $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_3 \rangle = C_{13}$) then $\vec{\boldsymbol{y}}_2$ (resp. $\vec{\boldsymbol{y}}_3$) is a random unit vector. Let $Y_{23} := \langle \vec{\boldsymbol{y}}_2 | \vec{\boldsymbol{y}}_3 \rangle$. We have

$$\langle \vec{\boldsymbol{x}}_2 | \vec{\boldsymbol{x}}_3 \rangle = C_{12}C_{13} + \sqrt{1-C_{12}^2}\sqrt{1-C_{13}^2}Y_{23}$$

which implies

$$Y_{23} = \frac{C_{23} - C_{12}C_{13}}{\sqrt{1-C_{12}^2}\sqrt{1-C_{13}^2}}.$$

We can now use Proposition 5.8,which gives the running time $T_{23}(\vec{\boldsymbol{x}}_1)$ of computing $L_{23}(\vec{\boldsymbol{x}}_3)$. Let $Y = \begin{pmatrix} 1 & Y_{23} \\ Y_{23} & 1 \end{pmatrix}$ and let α' such that $\mathcal{V}(\alpha') = \frac{1}{|L_2(\vec{\boldsymbol{x}}_1)|}$. We have

$$T_{23}(\vec{\boldsymbol{x}}_1) = \mathcal{O}\left(\ \mathrm{NbRep}_{\alpha',Y} \cdot |L_2(\vec{\boldsymbol{x}}_1)|\right).$$

Now, let T_{23} be the running of computing all the lists $L_{23}(\vec{\boldsymbol{x}}_1)$ since the number of $\vec{\boldsymbol{x}}_1$ is $|L_1|$, we have

$$T_{23} = |L_1| \cdot T_{23}(\vec{\boldsymbol{x}}_1) = \mathcal{O}\left(|L_1| \cdot\ \mathrm{NbRep}_{\alpha',Y} \cdot |L_2(\vec{\boldsymbol{x}}_1)|\right) \tag{5.8}$$
$$= \mathcal{O}\left(|L_1| \cdot |L_2(\vec{\boldsymbol{x}}_1)|^2 \cdot \frac{\left(1-Y_{23}^2\right)^{n/2}}{\left(1-Y'_{23}(\alpha')^2\right)^{n/2}}\right) \tag{5.9}$$

with $Y'_{23}(\alpha) = \frac{1}{\sin^2(\alpha)}\left(Y_{23} + \cos^2(\alpha')\right)$. Putting everything together, we have the following

Proposition 5.10. *Let $|L_1|$ a list size and C a 3×3 configuration matrix with negative non-diagonal entries. Let $|L_2(\vec{\boldsymbol{x}}_1)| = |L_1|(1-C_{12}^2)^{n/2}$. Let α' such that $\mathcal{V}(\alpha') = \frac{1}{|L_2(\vec{\boldsymbol{x}}_1)|}$. Algorithm 13 solves $\mathrm{FAR}_3^c(|L_1|, C)$ in time $T_{12}+T_{13}+T_{23}$ with*

$$T_{12} = T_{23} = \mathcal{O}\left(|L_1|^2(1-C_{12}^2)^{n/2}\right)$$
$$T_{123} = |L_1||L_2(\vec{\boldsymbol{x}}_1)|^2\frac{\left(1-Y_{23}^2\right)^{n/2}}{(1-Y'_{23}(\alpha')^2)^{n/2}}$$

where

$$Y_{23} = \frac{1}{\sqrt{1-C_{12}^2}\sqrt{1-C_{13}^2}}\cdot(C_{23}+C_{12}C_{13})$$
$$Y'_{23}(\alpha) = \frac{1}{\sin^2(\alpha)}\left(Y_{23}+\cos^2(\alpha')\right).$$

Let $|L_{12}| = |L_1|^2\left(1-C_{12}^2\right)^{n/2}$. This algorithm uses memory $M = \max\{|L_1|, |L_{12}|\}$.

Complexity of the classical 3-sieve.

The above was the analysis of the classical 3-sieve after the first filtering. We now apply Theorem 5.7 with $k=3$ in order to obtain the running of our classical 3-sieve algorithm within our framework (Algorithm 12).

Theorem 5.11. *There is a classical algorithm with parameter α that solves the 3-sieve problem for a configuration C and lists of size $|L|$ that runs in time T and that uses memory M with*

$$T = \mathcal{O}\left(\ \mathrm{NbRep}_{\alpha,C}\cdot\left(|L| + \frac{1}{\mathcal{V}(\alpha)}\cdot T\left(\mathrm{FAR}_3^c\left(|L_1|, C'(\alpha)\right)\right)\right)\right),$$

and

$$M = \max\{|L|, |L_1|, |L_{12}|, |L_{123}|\}.$$

where $|L_1| = |L|\cdot\mathcal{V}(\alpha)$, $|L_{12}|, |L_{123}|$ can be taken from Proposition 3.29 and $\mathrm{FAR}_3^c(|L_1|, C) = T_{12}+T_{13}+T_{123}$ where each T_{12}, T_{13}, T_{123} can be taken from Proposition 5.10.

Proposition 5.12. *There exists a classical algorithm for SVP using 3-sieve that runs in time $2^{0.338n+o(n)}$ and uses memory $2^{0.1887n+o(n)}$.*

Proof. Take the above proposition with a configuration matrix C such that $C_{12} = C_{13} = C_{23} = -\frac{1}{3}$, $\alpha = 1.2954rad$ and $|L| = 2^{0.1887n}$. We apply Proposition 5.11; We write $C'_{12}(\alpha) = C'_{13}(\alpha) = C'_{23}(\alpha) \approx -0.32$ and $|L_1| = |L|\cdot\mathcal{V}(\alpha) = 2^{0.133n}$. We have, (omitting $o(n)$ factors in the exponent)

$$\mathrm{NbRep}_{\alpha,C} = 2^{0.070n} \quad ; \quad \frac{1}{\mathcal{V}(\alpha)} = 2^{0.055n} \quad ; \quad T\left(\mathrm{FAR}_3^c(|L_1|, C')\right) = 2^{0.213n}$$

Putting everything together, we indeed have a running time of $2^{0.070n}\cdot 2^{0.055n}\cdot 2^{0.213n} = 2^{0.338n}$. The memory $M = \max\{|L|, |L_{out}|, |L_{int}|\}$ where L_{int} is the intermediate list used in $\mathrm{FAR}_3^c(|L_1|, C'(\alpha))$. We have

$$|L_{int}| = |L_1|^2\left(1-C'_{12}(\alpha)^2\right)^{n/2} = 2^{0.1887n}.$$

This implies that the memory used is $M = 2^{0.1887n}$. □

Space-time tradeoff. We also extend this algorithm where we fix the available memory to something more than the minimal memory $2^{0.1887n}$. We present here a list of points that we obtain, showing the general behaviour of our algorithm:

$\frac{1}{n}\log_2$(Memory)	0.1887	0.19	0.2	0.2075	0.22	0.24	0.26	0.272	0.286
$\frac{1}{n}\log_2$(Time)	0.338	0.334	0.328	0.325	0.320	0.313	0.307	0.304	0.304
Angle α (rad)	1.2954	1.305	1.329	1.346	1.366	1.408	1.470	$\pi/2$	$\pi/2$

Table 5.1: Time complexity of our classical 3-sieving algorithm for a fixed memory constraint. α is the optimal angle used in the first prefiltering. Also see Figure 5.3 for a plot corresponding to this algorithm.

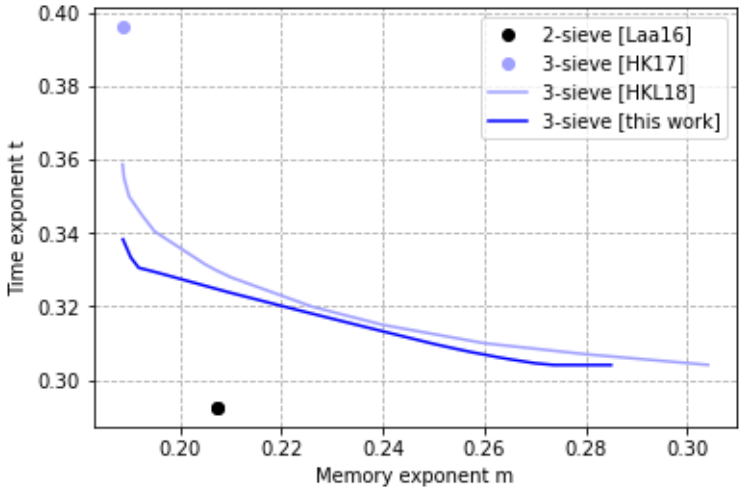


Figure 5.3: Time $T = 2^{tn+o(n)}$ for classical 3-sieves as a function of available memory $2^{mn+o(n)} = 2^M$. (Theorem 5.11).

5.3.3. Classical 4-sieve

We now consider the case $k = 4$. For our inner algorithms, we start with 4 lists L_1, L_2, L_3, L_4. There are actually several strategies of merging the lists. Here we choose to perform the following merges:

1. Construct $L_{12} = \{(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \in L_1 \times L_2 : \langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle \leq C_{12}\}$ and $L_{34} = \{(\vec{\boldsymbol{x}}_3, \vec{\boldsymbol{x}}_4) \in L_3 \times L_4 : \langle \vec{\boldsymbol{x}}_3 | \vec{\boldsymbol{x}}_4 \rangle \leq C_{34}\}$.

2. Construct $L_{1234} = \{((\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2), (\vec{\boldsymbol{x}}_3, \vec{\boldsymbol{x}}_4)) \in L_{12} \times L_{34} : (\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3, \vec{\boldsymbol{x}}_4)$ satisfies configuration $C\}$.

Using these lists, we consider the following algorithm:

Algorithm 14 FindAllReducing classical 4-sieve

Require: lists L_1, L_2, L_3, L_4 of vectors i.i.d. in $\mathcal{S}^{n-1}$ with $|L_1| = |L_2| = |L_3| = |L_4|$; target configuration $C \in \mathbb{R}^{4\times 4}$ with $C_{12} = C_{34}$ and $C_{13} = C_{14} = C_{23} = C_{24}$.
Ensure: list L_{out} of all 4-tuples $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3, \vec{\boldsymbol{x}}_4) \in L_1 \times L_2 \times L_3 \times L_4$ satisfying configuration C.
Construct L_{12} and L_{34} using our classical 2-sieve algorithm.
Start from L_{12} and L_{34} and use our classical 2-sieve algorithm to compute L_{1234}.
return L_{1234}.

We then use the above algorithm as the FindAllReducing subroutine in Algorithm 12 to describe our entire algorithm for 4-sieve. The algorithm presented here is usually inefficient in memory because the lists L_{12} and L_{13} are large. However, thanks to our initial α-filtering, we start from smaller lists L_1, L_2, L_3, L_4 so the intermediate lists will be small as well.

Complexity of Algorithm 14.

Lemma 5.13. *Let T_{12} be the time to compute L_{12} (which is also the time to compute L_{34} by symmetry). Let α such that $\mathcal{V}(\alpha) = \frac{1}{|L_1|}$. Then*

$$T_{12} = \mathcal{O}\left(|L_1|^2 \frac{\left(1-C_{12}^2\right)^{n/2}}{\left(1-C'_{12}(\alpha)^2\right)^{n/2}}\right)$$

This comes directly from the analysis of our simplified 2-sieve algorithm (Proposition 5.8). The size of the intermediate lists L_{12} and L_{34} is then

$$|L_{12}| = |L_1|^2 \cdot \left(1-C_{12}^2\right)^{n/2} \tag{5.10}$$

We now look at the time to compute L_{1234}. Elements of L_{12} are of squared norm $R^2 = 2 + 2C_{12}$, using $\|\vec{\boldsymbol{x}}_1 + \vec{\boldsymbol{x}}_2\|^2 = \|\vec{\boldsymbol{x}}_1\|^2 + \|\vec{\boldsymbol{x}}_2\|^2 + 2\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle$.

Lemma 5.14. *Let $\vec{\boldsymbol{z}}_{12} \in L_{12}$ and $\vec{\boldsymbol{z}}_{34} \in L_{34}$. If $\theta(\vec{\boldsymbol{z}}_{12}, \vec{\boldsymbol{z}}_{34}) = \arccos\left(\frac{\sin^2(\alpha)}{2R^2} - 1\right)$ then $\|\vec{\boldsymbol{z}}_{12} + \vec{\boldsymbol{z}}_{34}\|^2 \leq \sin^2(\alpha)$.*

Proof. We write

$$\|\vec{\boldsymbol{z}}_{12} + \vec{\boldsymbol{z}}_{34}\|^2 = \|\vec{\boldsymbol{z}}_{12}\|^2 + \|\vec{\boldsymbol{z}}_{34}\|^2 + 2\langle \vec{\boldsymbol{z}}_{12} | \vec{\boldsymbol{z}}_{34} \rangle = 2R^2 + 2\langle \vec{\boldsymbol{z}}_{12} | \vec{\boldsymbol{z}}_{34} \rangle$$

By taking $\langle \vec{\boldsymbol{z}}_{12} | \vec{\boldsymbol{z}}_{34} \rangle = R^2 \left(\frac{r_0^2}{2R^2} - 1\right)$, we obtain indeed $\|\vec{\boldsymbol{z}}_{12} + \vec{\boldsymbol{z}}_{34}\|^2 \leq \left(-\frac{1}{\sin(\alpha)}\right)^2$. □

Lemma 5.15. *Let T_{1234} be the time to compute L_{1234}. Let $Y = -\frac{1}{\sin(\alpha)} \cdot \frac{1}{4+4C_{12}} - 1$. Let α' such that $\mathcal{V}(\alpha') = \frac{1}{|L_{12}|}$. We have*

$$T_{1234} = \mathcal{O}\left(|L_{12}|^2 \frac{\left(1-Y^2\right)^{n/2}}{\left(1-Y'(\alpha')^2\right)^{n/2}}\right),$$

with $Y'(\alpha') = \frac{1}{\sin^2(\alpha)}\left(Y + \cos^2(\alpha')\right)$.

By combining the above 2 propositions, we have

Theorem 5.16. *Algorithm 14 runs in time $T = 2T_{12} + T_{1234}$ where T_{12} and T_{1234} can be taken respectively from Lemma 5.13 and Lemma 5.15.*

To conclude, we can plug this theorem again in Theorem 5.7 to get our results. Recall that we work with 4-tuples of residual vectors after an initial α-filtering so we look for 4-tuples of residual points $(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3, \vec{\boldsymbol{y}}_4)$ st. $\|\vec{\boldsymbol{y}}_1 + \vec{\boldsymbol{y}}_2 + \vec{\boldsymbol{y}}_3 + \vec{\boldsymbol{y}}_4\| \leq \frac{1}{\sin(\alpha)}$ (see Equation 5.3). Regarding memory requirements, we have that the memory M of our algorithm satisfies $M = \max\{|L_1|, |L_{12}|, |L_{1234}|\}$.

This algorithm gives a smooth space-time tradeoff from low memory to the point where the memory is $2^{0.0275n}$ and the time is $2^{0.292n}$, corresponding precisely to the complexity of the 2-sieve algorithm (and indeed corresponds to the case where our 4-sieve algorithm performs independently two 2-sieve algorithms). When looking at the minimal memory setting, so $M = 2^{0.1724n}$, this algorithm performs poorly, as the time is $2^{0.418n}$. However, when looking at intermediate memory requirements, there are some ranges when the algorithm performs quite well. For example, when taking $M = 2^{0.1887n}$, this algorithm performs better than the 3-sieve classical algorithm we presented before, as we can see in Figure 5.7. We put below a list of values of interest. As in the previous case, the less memory we are allowed, the more it is interesting to perform a tailored prefiltering step. We put below a list of values and the corresponding angle α used in the prefiltering step.

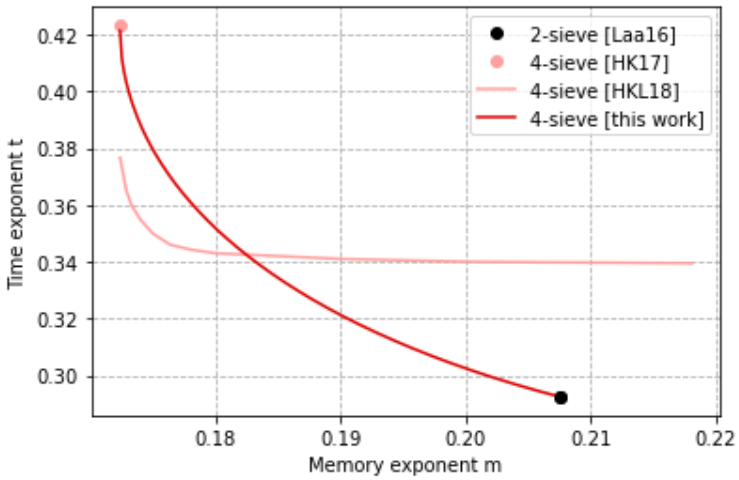


Figure 5.4: Time $T = 2^{tn+o(n)}$ for classical 4-sieves as a function of available memory $2^{mn+o(n)} = 2^M$. (Theorem 5.16).

$\log_2(\text{Memory})/n$	0.1724	0.175	0.18	0.1887	0.193	0.198	0.203	0.2075
$\log_2(\text{Time})/n$	0.418	0.380	0.352	0.324	0.315	0.306	0.298	0.2925
Angle α (rad)	1.278	1.315	1.350	1.401	1.425	1.457	1.494	$\pi/2$

Table 5.2: Time complexity of our classical 4-sieving algorithm for a fixed memory constraint. α is the optimal angle used in the prefiltering. Also see Figure 5.4 for a plot corresponding to this algorithm.

5.4. Quantum k-sieves

We now study quantum algorithms within our framework. In the quantum setting, we still use Theorem 5.7 and once again, we only need to describe the running time and amount of memory used for the subroutine. Here the input lists L_i are stored classically and are assumed to be quantumly accessible, *i.e.* for any given list L, we can efficiently construct the uniform superposition over all its elements $|\psi_L\rangle := \frac{1}{\sqrt{|L|}} \sum_\ell |\ell\rangle|L[\ell]\rangle$. In the following, we will not necessarily write the first register for simplicity[1].

5.4.1. Quantum 3-sieve

In the case $k = 3$, the FindAllReducing quantum subroutine starts with classical lists L_1, L_2, L_3 that are quantumly accessible, and it outputs a list containing all triples in $L_1 \times L_2 \times L_3$ satisfying a given target configuration C.

To find one solution, our algorithm constructs a uniform quantum superposition over all triples and then applies two Grover's algorithms in order to get a quantum superposition of candidate solutions. This whole process is then repeated inside an amplitude amplification to get a superposition over the solutions, that we measure, and we repeat this whole process until we have found all the solutions.

As a reminder, (See Proposition 3.29), given a configuration C, we use the following notation: for $i, j \in [k]$, $i \neq j$ and $\vec{\boldsymbol{y}}_j \in L_j$,

$$L_i(\vec{\boldsymbol{y}}_j) := \{\vec{\boldsymbol{y}}_i \in L_i : \langle \vec{\boldsymbol{y}}_i | \vec{\boldsymbol{y}}_j \rangle \leq C_{i,j}\}.$$

[1]This simplification was already done in [Kir+19]. At no point do we use the fact that we do not have the first register, this is just for simplicity of notations.

Algorithm 15 FindAllReducing quantum 3-sieve

Require: lists L_1, L_2, L_3 of vectors i.i.d. in $\mathcal{S}^{n-1}$ with $|L_1| = |L_2| = |L_3|$; a target configuration $C \in \mathbb{R}^{3\times 3}$.
Ensure: list L_{out} containing all 3-triples in $L_1 \times L_2 \times L_3$ satisfying configuration C.
  $L_{out} := \emptyset$
  while $|L_{out}| < |Sol|$ **do**
    Construct state $|\psi_{L_1}\rangle|\psi_{L_2}\rangle|\psi_{L_3}\rangle$
    Apply **Grover** on the second register to get state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3}\rangle$
    Apply **Grover** on the third register to get state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1)}\rangle$
    Apply Amplitude Amplification to get state $|\psi_{Sol}\rangle$, the uniform superposition of all solutions
    Take a measurement and get some $(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3)$
    if $(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3)$ satisfies configuration C **then** add it to L_{out}
  return L_{out}

Complexity of Algorithm 15.

We first analyse the complexity to find one solution during one single iteration from the while-loop.

Initialization. We assume that lists L_1, L_2 and L_3 of i.i.d. random points are classically stored and quantumly accessible. So the state $|\psi_{L_1}\rangle|\psi_{L_2}\rangle|\psi_{L_3}\rangle$ can be constructed efficiently.

Grover on the second register. The algorithm then applies Grover's algorithm on the second register such that the two first registers become

$$|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle = \frac{1}{\sqrt{|L_1|}}\frac{1}{\sqrt{|L_2(\vec{\boldsymbol{y}}_1)|}}\sum_{\vec{\boldsymbol{y}}_1\in L_1}\sum_{\vec{\boldsymbol{y}}_2\in L_2(\vec{\boldsymbol{y}}_1)}|\vec{\boldsymbol{y}}_1\rangle|\vec{\boldsymbol{y}}_2\rangle.$$

It only keeps in the quantum superposition the elements $\vec{\boldsymbol{y}}_2 \in L_2$ such that $\langle\vec{\boldsymbol{y}}_1|\vec{\boldsymbol{y}}_2\rangle \le C_{12}$ for each superposed $\vec{\boldsymbol{y}}_1$ from the first register. So the state ends up with a quantum superposition of all pairs in $L_1 \times L_2$ eligible to form the beginning of a triple-solution. This application of Grover's algorithm takes time $T_2 = \sqrt{\frac{|L_2|}{|L_2(\vec{\boldsymbol{y}}_1)|}} = (1 - C_{12}^2)^{-n/4}$ by Proposition 3.29.

Grover on the third register. Similarly, we also apply Grover's algorithm on the third register to get the state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1)}\rangle$ equal to

$$\frac{1}{\sqrt{|L_1|}}\frac{1}{\sqrt{|L_2(\vec{\boldsymbol{y}}_1)|}}\frac{1}{\sqrt{|L_3(\vec{\boldsymbol{y}}_1)|}}\sum_{\vec{\boldsymbol{y}}_1\in L_1}\sum_{\vec{\boldsymbol{y}}_2\in L_2(\vec{\boldsymbol{y}}_1)}\sum_{\vec{\boldsymbol{y}}_3\in L_3(\vec{\boldsymbol{y}}_1)}|\vec{\boldsymbol{y}}_1\rangle|\vec{\boldsymbol{y}}_2\rangle|\vec{\boldsymbol{y}}_3\rangle$$

in time $T_3 = \sqrt{\frac{|L_3|}{|L_3(\vec{\boldsymbol{y}}_1)|}} = (1 - C_{13}^2)^{-n/4}$. The sizes $|L_2(\vec{\boldsymbol{y}}_1)|$ and $|L_3(\vec{\boldsymbol{y}}_1)|$ do not depend on the choice of $\vec{\boldsymbol{y}}_1$, that is why we can write their corresponding normalizing factors before the sum over the $\vec{\boldsymbol{y}}_1$'s.

Amplitude amplification. The goal is now to construct a uniform quantum superposition over all elements of the set of solutions $Sol := \{(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3) \in L_1 \times L_2 \times L_3 \text{ satisfying } C\}$, by applying a quantum amplitude amplification. Let $\mathcal{A}$ be unitary that maps $|0\rangle|0\rangle|0\rangle$ to the state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1)}\rangle$ constructed so far.

Lemma 5.17. *The operation $\mathcal{A} : |0\rangle|0\rangle|0\rangle \to |\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1)}\rangle$ is repeated T_{AA} times inside the amplitude amplification to construct state $|\psi_{Sol}\rangle$ (with probability $1 - o(1)$), where*

$$\begin{aligned} T_{AA} &= \mathcal{O}\left(\sqrt{|L_1|/|Sol|}\cdot\sqrt{|L_2(\vec{\boldsymbol{y}}_1)|}\sqrt{|L_3(\vec{\boldsymbol{y}}_1)|}\right) \\ &= \mathcal{O}\left(\sqrt{|L_i^3|/|Sol|}\cdot(1 - C_{12}^2)^{n/4}(1 - C_{13}^2)^{n/4}\right). \end{aligned}$$

Proof. Performing a measurement of state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1)}\rangle$ gives a triplet solution $(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3)$ with some probability p, we are going to specify. There are $|L_1|$ possible $\vec{\boldsymbol{y}}_1$ and $|Sol|$ "good" ones belonging to a solution, so the probability of measuring a good $\vec{\boldsymbol{y}}_1$ is $|Sol|/|L_1|$. Then given a $\vec{\boldsymbol{y}}_1$, a pair $(\vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3) \in L_2(\vec{\boldsymbol{y}}_1) \times L_3(\vec{\boldsymbol{y}}_1)$ forms the solution together with $\vec{\boldsymbol{y}}_1$ with probability $\frac{1}{|L_2(\vec{\boldsymbol{y}}_1)|}\frac{1}{|L_3(\vec{\boldsymbol{y}}_1)|}$.

Finally, the probability to measure a solution is thus $p = |Sol|/|L_1| \cdot \frac{1}{L_2(\vec{\boldsymbol{y}}_1)}\frac{1}{L_3(\vec{\boldsymbol{y}}_1)}$. By Theorem 2.10, the number of iterations of amplitude amplification is $\mathcal{O}\left(1/\sqrt{p}\right)$, hence the top line. The bottom line is obtained by expressing the sizes of $L_2(\vec{\boldsymbol{y}}_1)$ and $L_3(\vec{\boldsymbol{y}}_1)$ using Proposition 3.29. □

Subroutine complexity.

Proposition 5.18 (FindAllReducing quantum 3-sieve)**.** *Let $|L_1|$ a list size and C a 3×3 configuration matrix with negative non-diagonal entries. Algorithm 15 solves $\mathrm{FAR}_3^q(|L_1|, C)$ in time $|Sol| \cdot (T_2 + T_3) \cdot T_{AA}$ where*

$$|Sol| = |L_1|^3 \cdot \det(C)^{n/2}$$
$$T_2 = (1 - C_{12}^2)^{-n/4} \quad ; \quad T_3 = (1 - C_{13}^2)^{-n/4}$$
$$T_{AA} = \mathcal{O}\left(\sqrt{|L_i^3|/|Sol|} \cdot (1 - C_{12}^2)^{n/4}(1 - C_{13}^2)^{n/4}\right)$$

After simplification, the time complexity of Algorithm 15 can be written

$$T(\mathrm{FAR}_3^q(|L_1|, C)) = \sqrt{|L_i|^3 \cdot |Sol|} \cdot \left((1 - C_{12}^2)^{n/2} + (1 - C_{13}^2)^{n/2}\right).$$

This algorithm uses classical memory $|L_1|$ and quantum memory $\mathrm{poly}(n)$ *qubits.*

Complexity of the quantum 3-sieve.

The above was the analysis of the algorithm we use as the subroutine FindAllReducing in Algorithm 12 for quantum 3-sieve. The lists given in input of Algorithm 15 are then the lists of residual vectors R_1, R_2, R_3, which are of size $|R_1| = |L_1| = |L| \cdot \mathcal{V}(\alpha)$; and it return residual vectors that satisfy the target configuration $C'(\alpha)$. Using Theorem 5.7 in the case $k = 3$ gives the overall time complexity of our quantum 3-sieve algorithm.

Theorem 5.19. *There is a quantum algorithm with parameter α that solves the 3-sieve problem for a configuration $C \in \mathbb{R}^{3\times 3}$ and lists of size $|L|$, that runs in time*

$$T = \mathcal{O}\left(\ \mathrm{NbRep}_{\alpha,C}\left(|L| + \frac{1}{\mathcal{V}(\alpha)} \cdot T(\mathrm{FAR}_3^q(|L_1|, C'(\alpha)))\right)\right)$$

where $|L_1| = |L| \cdot \mathcal{V}(\alpha)$ and $T(\mathrm{FAR}_3^q(|L_1|, C'(\alpha)))$ given by Proposition 5.18. This algorithm uses quantum-accessible classical memory $M = |L|$ and quantum memory poly(d).

Minimal memory parameters.

Proposition 5.20. *There is a quantum algorithm that solves SVP in dimension n using 3-sieve that runs in time $T = 2^{0.3098d+o(n)}$, quantum-accessible classical memory $M = 2^{0.1887n+o(n)}$ and* $\mathrm{poly}(n)$ *quantum memory.*

Proof. We take a balanced configuration C with $C_{12} = C_{13} = C_{23} = -1/3$, $\alpha = 1.2343$rad and $|L| = 2^{0.1887n} = M$. We apply Proposition 5.19: We write $C'_{12}(\alpha) = C'_{13}(\alpha) = C'_{23}(\alpha) \approx -0.31$ and $|L_1| = |L| \cdot \mathcal{V}(\alpha) = 2^{0.1055n}$. We have

$$\mathrm{NbRep}_{\alpha,C} = 2^{0.1055n} \quad ; \quad \frac{1}{\mathcal{V}(\alpha)} = 2^{0.0832n} \quad ;$$
$$T(\mathbf{FAS}_3^q(|L_1|, C')) = 2^{0.1210n}.$$

Putting everything together, we have a running time of
$2^{0.1055n} \cdot 2^{0.0832n} \cdot 2^{0.1210n} = 2^{0.3098}$. □

Space-time tradeoffs. We also extend this algorithm where we fix the available memory to something more than the minimal memory $2^{0.1887n}$.

$\log_2(\text{Memory})/n$	0.1887	0.189	0.190	0.1904
$\log_2(\text{Time})/n$	0.3069	0.3050	0.3040	0.3039
α (rad)	1.2153	1.2191	1.2255	1.2233

Table 5.3: Time complexity of our quantum 3-sieving algorithm for a fixed memory constraint. α is the optimal angle used in the prefiltering. Also see Figure 5.5 for a plot corresponding to this algorithm.

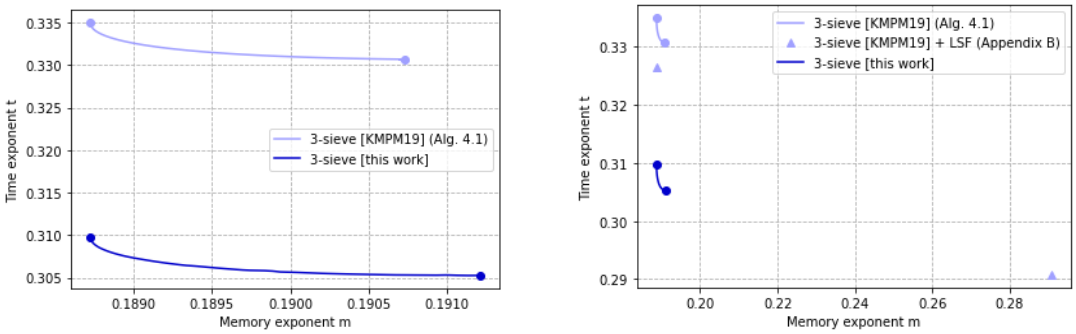


Figure 5.5: Space-time tradeoffs for quantum 3-sieves algorithms with time $2^{tn+o(n)}$ as a function of available memory $2^{mn+o(n)}$ (Theorem 5.19). The left graphic shows the comparison between the quantum BLS and our improved version adding k-RPC prefiltering. The right graphic adds the extremities of the tradeoff of [Kir+19, Appendix B], the quantum hybrid with pairwise filtering.

5.4.2. Quantum 4-sieve

This algorithm and its analysis are very similar to Algorithm 15. As previously, we first analyze the complexity to find one solution during one single iteration from the while-loop.

Complexity of Algorithm 16.

Initialization. Lists L_i for $i = 1, 2, 3, 4$ are assumed stored classically and quantumly accessible, so we can construct the state $|\psi_{L_1}\rangle|\psi_{L_2}\rangle|\psi_{L_3}\rangle|\psi_{L_4}\rangle$.

Grover on the second register. The algorithm applies Grover's algorithm over the second register such that the two first registers become

$$|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle = \frac{1}{\sqrt{|L_1|}}\frac{1}{\sqrt{|L_2(\vec{\boldsymbol{y}}_1)|}}\sum_{\vec{\boldsymbol{y}}_1\in L_1}\sum_{\vec{\boldsymbol{y}}_2\in L_2(\vec{\boldsymbol{y}}_1)}|\vec{\boldsymbol{y}}_1\rangle|\vec{\boldsymbol{y}}_2\rangle,$$

which takes time $T_2 = \sqrt{\frac{|L_2|}{|L_2(\vec{\boldsymbol{y}}_1)|}} = (1 - C_{12}^2)^{-n/4}$.

Grover on the third register. Another Grover's algorithm is then performed over the third register $|\psi_{L_3}\rangle$ such that it becomes the quantum superposition over all elements of $L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)$, for $\vec{\boldsymbol{y}}_1 \in L_1$ and $\vec{\boldsymbol{y}}_2 \in L_2(\vec{\boldsymbol{y}}_1)$ being elements in quantum superposition in the two first registers. Let $Z = |L_1| \cdot |L_2(\vec{\boldsymbol{y}}_1)| \cdot |L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)|$. The three first registers then become the state

Algorithm 16 FindAllReducing quantum 4-sieve

Require: lists L_1, L_2, L_3, L_4 of vectors i.i.d. in $\mathcal{S}^{n-1}$ with $|L_1| = |L_2| = |L_3| = |L_4|$; a target configuration $C \in \mathbb{R}^{4\times 4}$.

Ensure: list L_{out} containing all 4-triples in $L_1 \times L_2 \times L_3 \times L_4$ satisfying configuration C.

$L_{out} := \emptyset$

while $|L_{out}| < |Sol|$ **do**

Construct $|\psi_{L_1}\rangle|\psi_{L_2}\rangle|\psi_{L_3}\rangle|\psi_{L_4}\rangle$

Apply **Grover** on the second register to get state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3}\rangle|\psi_{L_4}\rangle$

Apply **Grover** on the third register to get state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\rangle|\psi_{L_4}\rangle$

Apply **Grover** on the fourth register to get state:

$$|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\rangle|\psi_{L_4(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\rangle$$

Apply Amplitude Amplification to get state $|\psi_{Sol}\rangle$, the uniform superposition of all solutions

Take a measurement and get some $(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3, \vec{\boldsymbol{y}}_4)$

if $(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3, \vec{\boldsymbol{y}}_4)$ satisfies configuration C **then** add it to L_{out}

return L_{out}

$$|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\rangle = \frac{1}{\sqrt{Z}} \sum_{\vec{\boldsymbol{y}}_1 \in L_1} \sum_{\vec{\boldsymbol{y}}_2 \in L_2(\vec{\boldsymbol{y}}_1)} \sum_{\vec{\boldsymbol{y}}_3 \in L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)} |\vec{\boldsymbol{y}}_1\rangle|\vec{\boldsymbol{y}}_2\rangle|\vec{\boldsymbol{y}}_3\rangle.$$

Performing this Grover's algorithm takes time $T_3 = \sqrt{\frac{|L_3|}{|L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)|}}$. Proposition 3.29 gives $|L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)| = |L_3| \cdot \left(\frac{\det(C[1,2,3])}{\det(C[1,2])}\right)^{n/2}$. Note that these notations for partial configurations are given in Definition 3.26. So we can rewrite $T_3 = \left(\frac{\det(C[1,2,3])}{\det(C[1,2])}\right)^{-n/4}$.

Grover on the fourth register. Analogously to what was done over the third register, we perform Grover's algorithm over the fourth one $|\psi_{L_4}\rangle$. For $Z' = |L_1| \cdot |L_2(\vec{\boldsymbol{y}}_1)| \cdot |L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)| \cdot |L_4(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)|$, this operation allows to construct the state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\rangle|\psi_{L_4(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\rangle$ equal to

$$\frac{1}{\sqrt{Z'}} \sum_{\vec{\boldsymbol{y}}_1 \in L_1} \sum_{\vec{\boldsymbol{y}}_2 \in L_2(\vec{\boldsymbol{y}}_1)} \sum_{\vec{\boldsymbol{y}}_3 \in L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)} \sum_{\vec{\boldsymbol{y}}_4 \in L_4(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)} |\vec{\boldsymbol{y}}_1\rangle|\vec{\boldsymbol{y}}_2\rangle|\vec{\boldsymbol{y}}_3\rangle|\vec{\boldsymbol{y}}_4\rangle.$$

This takes time $T_4 = \left(\frac{\det(C[1,2,4])}{\det(C[1,2])}\right)^{-n/4}$.

Amplitude amplification. We then want to construct a uniform quantum superposition over all elements of the set of solutions $Sol := \{(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3, \vec{\boldsymbol{y}}_4) \in L_1 \times L_2 \times L_3 \times L_4$ satisfying $C\}$, by applying a quantum amplitude amplification.

Lemma 5.21. *The operation $|0\rangle|0\rangle|0\rangle|0\rangle \to |\psi_{L_1}\rangle|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\rangle|\psi_{L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\rangle|\psi_{L_4(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\rangle$ is repeated T_{AA} times inside the amplitude amplification to construct state $|\psi_{Sol}\rangle$ (with probability $1 - o(1)$), where*

$$\begin{aligned} T_{AA} &= \sqrt{\frac{|L_1|}{|Sol|}}\sqrt{|L_2(\vec{\boldsymbol{y}}_1)|}\sqrt{|L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)|}\sqrt{|L_4(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)|} \\ &= \frac{|L_i|^2}{\sqrt{|Sol|}} \cdot (1 - C_{12}^2)^{n/4} \cdot \left(\frac{\det(C[1,2,3])}{\det(C[1,2])}\right)^{n/4} \cdot \left(\frac{\det(C[1,2,4])}{\det(C[1,2])}\right)^{n/4} \end{aligned}$$

where notation $C[I]$ with a set of indexes I was introduced in Definition 3.26.

Proof. The reasoning is the same as for the proof of Lemma 5.17. Performing a measurement of state $|\psi_{L_1}\rangle|\psi_{L_2(\vec{y}_1)}\rangle|\psi_{L_3(\vec{y}_1,\vec{y}_2)}\rangle|\psi_{L_4(\vec{y}_1,\vec{y}_2)}\rangle$ gives a 4-tuple solution $(\vec{y}_1,\vec{y}_2,\vec{y}_3,\vec{y}_4)$ with some probability p, we are going to specify. The probability of measuring a good $\vec{y}_1$ is $|Sol|/|L_1|$. Then given a $\vec{y}_1$, a triple $(\vec{y}_2,\vec{y}_3,\vec{y}_4)$ forms the solution together with $\vec{y}_1$ with probability $1/(|L_2(\vec{y}_1)|\cdot|L_3(\vec{y}_1,\vec{y}_2)|\cdot|L_4(\vec{y}_1,\vec{y}_2)|)$. Finally, the probability of success to measure a solution is thus $p=\frac{|Sol|}{|L_1|}\cdot 1/\left(|L_2(\vec{y}_1)|\cdot|L_3(\vec{y}_1,\vec{y}_2)|\cdot|L_4(\vec{y}_1,\vec{y}_2)|\right)$. By Theorem 2.10, the number of iterations of amplitude amplification is $\mathcal{O}\left(1/\sqrt{p}\right)$, hence the top line. The bottom line is obtained by expressing the sizes of $L_2(\vec{y}_1)$, $L_3(\vec{y}_1,\vec{y}_2)$ and $L_4(\vec{y}_1,\vec{y}_2)$ using Proposition 3.29. □

Measurement gives a 4-tuple $(\vec{y}_1,\vec{y}_2,\vec{y}_3,\vec{y}_4)$ solution to the configuration problem. We need to repeat this whole process until we find all the solutions at number $|Sol|$. Notice that the same operations are performed over L_3 and over L_4, which implies that an optimal configuration will necessarily respect the symmetry $C_{13}=C_{14}$ and $C_{23}=C_{24}$. In the end, this subroutine FindAllReducing runs in time

$$T(\mathrm{FAR}_4^q) \quad = \quad |Sol|\cdot(T_2+T_3+T_4)\cdot T_{AA},$$

and this leads to the following theorem.

Proposition 5.22. *Given lists $L_1,L_2,L_3,L_4\subset\mathcal{S}^{n-1}$ of same size $|L_i|$ with i.i.d. uniformly random vectors, and a configuration $C\in\mathbb{R}^{4\times4}$ with $C_{13}=C_{14}$ and $C_{23}=C_{24}$, there exists an algorithm that finds all the $|Sol|$ 4-tuples in $L_1\times L_2\times L_3\times L_4$ satisfying configuration C in time*

$$T(\mathrm{FAR}_4^q)=|L_i|^2\sqrt{|Sol|}\left(\left(\frac{1}{1-C_{12}^2}\right)^{n/2}+\left(\frac{\det(C[1,2,3])}{1-C_{12}^2}\right)^{n/4}\right).$$

Complexity of the quantum 4-sieve.

The above was the analysis of the quantum 4-sieve after the prefiltering. We use this algorithm as the subroutine in our framework for $k=4$. Using Theorem 5.7 in this case, we recover the overall time complexity of our quantum 4-sieve algorithm.

Theorem 5.23. *There is a quantum algorithm with parameter α that solves the 3-sieve problem for a configuration C and lists of size $|L|$ that runs in time T with*

$$T=\mathcal{O}\left(\ \mathrm{NbRep}_{\alpha,C}\left(|L|+\frac{1}{\mathcal{V}(\alpha)}T(\mathrm{FAR}_4^q(|L_1|,C'(\alpha)))\right)\right)$$

and uses quantum-accessible classical memory $M=|L|$ and quantum memory poly(d), and where $|L_1|=|L|\cdot\mathcal{V}(\alpha)$ and $T(\mathrm{FAR}_4^q(|L_1|,C'(\alpha)))$ given by Proposition 5.22.

Minimal memory parameters.

Proposition 5.24. *There is a quantum algorithm that solves SVP in dimension n using 4-sieve that runs in time $T=2^{0.3276n+o(n)}$ using quantum-accessible classical memory $M=2^{0.1724n+o(n)}$ and quantum memory* poly(n).

Proof. We take a balanced configuration C with $C_{i,j}=-1/4$ for $i\neq j$, $\alpha\approx1.3131$rad and $|L|=2^{0.1724n}=M$. We apply Theorem 5.23: We write $C'_{i,j}\approx-0.244$ for $i\neq j$ and $|L_1|=|L|\cdot\mathcal{V}(\alpha)=2^{0.124n}$. We have

$$\mathrm{NbRep}_{\alpha,C}=2^{0.1069n}\quad;\quad\frac{1}{\mathcal{V}(\alpha)}=2^{0.0484n}\quad;\quad T(\mathrm{FAR}_4^q(|L_1|,C'))=2^{0.1722n}.$$

Putting everything together, we have a running time of $2^{0.1069n}\cdot2^{0.0484n}\cdot2^{0.1722n}=2^{0.3276n}$. □

Time-optimizing parameters.

Proposition 5.25. *There exists an algorithm that solves SVP in dimension n in time $T = 2^{0.3120n+o(n)}$ using quantum-accessible classical memory $M = 2^{0.1813n+o(n)}$ and quantum memory* $\mathrm{poly}(n)$.

Proof. We take a configuration C with $C_{12} \approx -0.386$, $C_{13} = C_{14} \approx -0.229$, $C_{23} = C_{24} \approx -0.230$ and $C_{34} \approx -0.200$. We take $\alpha \approx 1.313$rad and $|L| = 2^{0.1813n} = M$. We apply Proposition 5.23: We write $C'_{12} \approx -0.386$, $C'_{13} = C'_{14} \approx -0.229$, $C'_{23} = C'_{24} \approx -0.224$ and $C'_{34} \approx -0.189$. We set $|L_1| = |L| \cdot \mathcal{V}(\alpha) = 2^{0.1259n}$. We have

$$\mathrm{NbRep}_{\alpha,C} = 2^{0.1254} \quad ; \quad \frac{1}{\mathcal{V}(\alpha)} = 2^{0.0554n};$$

$$\mathrm{FAR}_4^q(|L_1|, C') = 2^{0.1312n}.$$

Putting everything together, we have a running time of $2^{0.1254n} \cdot 2^{0.0554n} \cdot 2^{0.1312n} = 2^{0.3120n}$. □

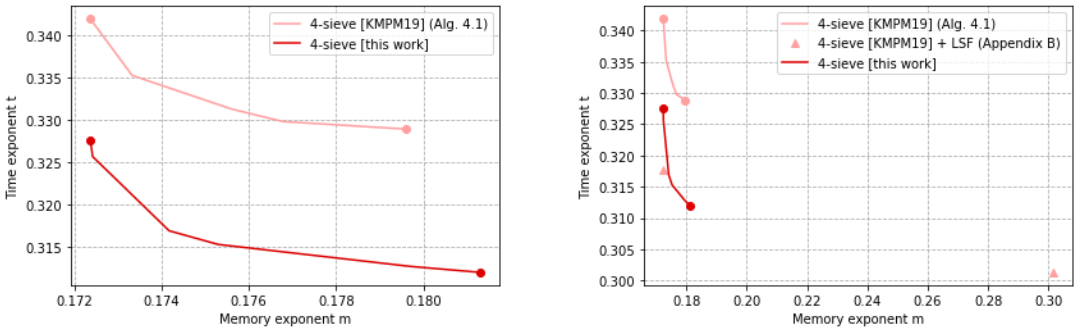


Figure 5.6: Space-time tradeoffs for quantum 4-sieves algorithms with time $2^{tn+o(n)}$ as a function of available memory $2^{mn+o(n)}$ (Theorem 5.23). The left graphic shows the comparison between the quantum BLS and our improved version adding k-RPC prefiltering. The right graphic adds the extremities of the tradeoff of [Kir+19, Appendix B], the quantum hybrid with pairwise filtering.

5.5. Discussion

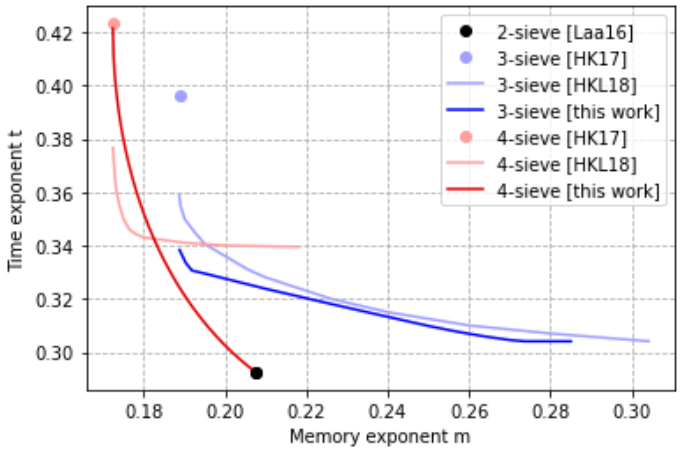


Figure 5.7: Time $2^{tn+o(n)}$ for classical k-sieves as a function of available memory $2^{mn+o(n)}$.

	Parameters setting	t	m	α	Reference
$k=3$	Min. memory + tailored LSF	0.338	0.1887	1.2954	Sec. 5.3.2
	Min. memory + LSF	0.3588	0.1887	-	[HKL18]
	Min. memory	0.3962	0.1887	-	[HK17]
	No memory constraint + LSF	0.3041	0.3041	-	[HKL18]
$k=4$	Memory-opt. config. + LSF	0.3766	0.1724	-	[HKL18]
	Memory-opt, tailored LSF	0.418	0.1724	1.278	Section 5.3.3
	Memory-opt. config.	0.424	0.1724	-	[HK17]
	Fixed memory, tailored LSF	0.3224	0.1887	1.401	Section 5.3.3
	Time-opt. config. + LSF	0.3419	0.2181	-	[HKL18]

Table 5.4: Classical 3 and 4-sieves, corresponds to the graph in Figure 5.7.

We first analyze our results for classical algorithms (see Figure 5.7). For the 3-sieve, our algorithm performs better in the minimal memory regime. However, when we do not restrict memory, we obtain the same running time $2^{0.3041n+o(n)}$ as in [HKL18] and our method does not give improvements here. For 4-sieve algorithms, the situation is a little different. We use a different approach than the ones studied in previous work. We essentially combine sequentially two 2-sieve algorithms. However, we first perform our tailored LSF on 4-tuples of points to speed up this process. As Figure 5.7 shows, this algorithm does not perform well in the minimal memory regime ($M = 2^{0.1723n+o(n)}$) but then works much better for slightly larger memories, outperforming our 3-sieve algorithm and also the best previously known running time for 4-sieve, which used more memory.

We must notice however that it is hard to make direct comparisons with previous work in the classical setting as those are mainly done for Gauss-sieve and we present results for NV-sieve which has better space-time tradeoffs asymptotically. However, our results do show that tailored LSF significantly improves the algorithms we study, and we leave it as future work to extend this idea to the Gauss sieve.

In the quantum setting (see Figure 5.8), we use the same algorithms as in [Kir+19] so the comparison can be made more directly. Our algorithm uses our tailored filtering and then applies Algorithm 4.1 of [Kir+19], which is not the best algorithm for the configuration problem for low values of k. What we show is that this algorithm benefits from this prefiltering. The results should be compared with the state-of-the-art Algorithm B.2 of [Kir+19]. However, only the extremities of the tradeoffs of their algorithm were given, represented by

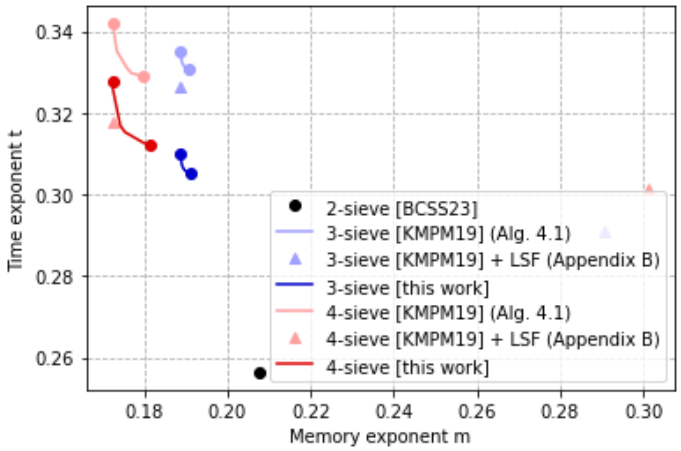


Figure 5.8: Time $2^{tn+o(n)}$ for quantum k-sieves as a function of available memory $2^{mn+o(n)}$.

	Parameters setting	t	m	α	Reference
$k=3$	Memory-opt, tailored LSF	0.3069	0.1887	1.2153	Sec. 5.4.1
	Memory-opt. config.	0.3349	0.1887	-	[Kir+19], Alg. 4.1
	Memory-opt. config. + LSF	0.3266	0.1887	-	[Kir+19], Alg. B.2
	Time-opt. config.	0.3306	0.1907	-	[Kir+19], Alg. 4.1 and 4.2
	Time-opt, tailored LSF	0.3039	0.1904	1.2233	Sec. 5.4.1
	Time-opt. config. + LSF	0.2908	0.2908	-	[Kir+19], Alg. B.2
$k=4$	Memory-opt, tailored LSF	0.3276	0.1724	1.3131	Sec. 5.4.2
	Memory-opt. config.	0.3419	0.1724	-	[Kir+19], Alg. 4.1
	Memory-opt. config. + LSF	0.3178	0.1724	-	[Kir+19], Alg. B.2
	Time-opt. config.	0.3289	0.1796	-	[Kir+19], Alg. 4.1
	Time-opt. config.	0.3197	0.1731	-	[Kir+19], Alg. 4.2
	Time-opt, tailored LSF	0.313	0.1800	1.2887	Sec. 5.4.2
	Time-opt. config. + LSF	0.3013	0.3013	-	[Kir+19], Alg. B.2

Table 5.5: Quantum 3 and 4-sieves, corresponds to the graph in Figure 5.8.

triangles on the graphs of Figure 5.8. For $k=3$ in the minimal memory regime $M=2^{0.1887n+o(n)}$, we achieve time $T=2^{0.3069n+o(n)}$ improving the time $T=2^{0.3266n+o(n)}$ of Algorithm B.2 in [Kir+19]. For $k=4$, our algorithm does not work well for the lowest memory regime but gives a new interesting space-time tradeoff.

Notice that as in previous algorithms cited here, our quantum algorithms require quantumly accessible classical memory (QRACM) and poly(n) qubits.

The goal of the construction of k-RPC and applications on low k-sieves was to check if this kind of filtering had an interest. It was believed before that doing two layers of filtering was useless, and we showed that on the contrary, the k-sieves benefit from it. Here again, the conditional bound of [KL21] cannot apply as our framework expands the k-sieve algorithms to be considered. With the algorithms proposed in previous sections, taking higher k diminishes the advantages of k-RPC filtering, until making it disappear for high k. But our algorithm structure remains a basic approach, and there are a lot of different other ways to mix k-RPC filtering, pairwise BDGL filtering, and merging the lists.

6. Security analysis of Wave

This chapter is based on the work [Loy23] which is currently a preprint, and took part in a joint submission [Ban+23] to the NIST with Gustavo Banegas, Kévin Carrier, André Chailloux, Alain Couvreur, Thomas Debris-Alazard, Philippe Gaborit Pierre Karpman, Ruben Niederhagen, Nicolas Sendrier, Benjamin Smith, and Jean-Pierre Tillich.

6.1. Overview

Code-based cryptography. Codes were originally introduced by Hamming [Ham50] to correct errors caused by noisy data transmission or unstable storage. The idea is to encode the data by adding some redundancy. Then, to access the data, one uses a decoding algorithm that allows one to recover the initial data even if it has been partially altered. There is however a limit to the number of errors that a decoding algorithm can correct.

The differences with a lattice lie in the definition space ($\mathbb{R}^n$ for lattices; $\mathbb{F}_q^n$ for codes) and the metric (Euclidian distance for lattices; Hamming, Lee, rank, etc., for codes...). Previously in this thesis (RPC in Definition 3.12), we saw how to use codes as a tool to solve lattice problems. In this chapter, we study a proper code problem: the Decoding Problem (DP). The Decoding Problem is NP-hard in the worst case [BMT78] and is believed to be hard for a random code, thus cryptographic schemes can rely on its hardness. The first code-based encryption scheme is the McEliece scheme [McE78], which is already resistant to quantum attacks like Shor's, but has very large keys. The goal for designers is to choose a code that allows good performances (fast decoding and small key sizes), but whose structure is not too obvious to the attacker so that the Decoding Problem with this code remains computationally hard.

In the previous NIST call for post-quantum encryption schemes [NIS22], Classic McEliece [Ber+22] was a finalist but was not standardized because of its large public key size, despite its strong and most conservative security. From this batch of schemes, let us also cite BIKE [Ara+21], based on a binary version of NTRU, a lattice problem that becomes a code-based one.

In 2023, NIST's call for digital signature schemes saw once again several code-based schemes emerge. FuLeeca [Rit+23] is based on Lee metric, LESS [Bal+23a] and MEDS [Cho+23b] are based on code equivalence problems; but different security vulnerabilities have been quickly found for these three last schemes. CROSS [Bal+23b] and SDitH [Agu+23] rely on a restricted Decoding Problem, and MIRA [Ara+23b], MiRitH [Adj+23] and RYDE [Ara+23a] use the rank metric version of DP. Finally, PERK [Aar+23] is based on the permuted kernel problem, a specific instance of the code equivalence problem. Two of the code-based submissions are based on a rather new code structure $(U, U+V)$, with modified Reed-Muller code for Enhanced pqsigRM [Cho+23a] and ternary for Wave [Ban+23].

Contributions. For each of the four best known attacks on Wave, we do a complete time complexity analysis and provide explicit expressions as functions of Wave parameters. So the claimed security level can easily be updated with new sets of parameters using our formulas. We then apply our theorems to the Round-1 parameter selection, whose results are summarized in Table 6.3. We describe a quantum smoothed Wagner's algorithm based on the combined approaches of [Sen11], [Bri+20], and [CDE21], and our new algorithm provides an improved message attack on Wave.

Outline. We first recall in Section 6.2 basic information about code cryptographic problems particularly in the particular case of Wave, the ISD framework, and list merging. Section 6.3 presents key attacks based on ISD and Dumer's algorithm. Then Section 6.4 presents message attacks based on ISD and Wagner's algorithm. In Section 6.5, we conclude and comment on the obtained results. The SageMath code used for the numerical results of this chapter is available here: `https://github.com/johanna-loyer/WaveISDcryptanalysis.git`.

6.2. Code-based cryptography

Notations.

$\mathbb{F}_q$ denotes a q-ary finite field. Vectors are in raw notation, written in bold and their coordinates are in plain, with $\boldsymbol{x} = (x_i)_i$. The weight considered in this chapter is the Hamming weight denoted $|\boldsymbol{x}| := |\{i : x_i \neq 0\}|$. For a vector $\boldsymbol{x} = (x_0, \dots, x_{n-1})$, we denote by $\boldsymbol{x}_{|_{[i,j]}}$ the vector $(x_i, \dots, x_{j-1})$ restricted on coordinates i to $j-1$. For a matrix $\mathbf{H}$ we denote by $\mathbf{H}^\top$ its transpose. For $\boldsymbol{x} = (x_0, \dots, x_{n-1}) \in \mathbb{F}_q^n$ and $\mathbf{M} = (M_{i,j})_{0 \leq i < r, 0 \leq j < n-1} \in \mathbb{F}_q^{r \times n}$, we define $\boldsymbol{x} \star \mathbf{M} := (x_j M_{i,j})_{0 \leq i < r, 0 \leq j < n}$ their row-wise star product.

6.2.1. Code problems and Wave

Definition 6.1 (Code $[n,k]_q$)**.** A linear code C of length n and dimension k over $\mathbb{F}_q$ is a k-dimensional subspace of $\mathbb{F}_q^n$. The elements of C are called *codewords*. The *rate* of C is defined as k/n. A matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ verifying $C = \{\mathbf{xG} \mid \mathbf{x} \in \mathbb{F}_q^k\}$ is called a generator matrix of C, and a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ verifying $C = \{\mathbf{y} \in \mathbb{F}_q^n \mid \mathbf{yH}^\top = \mathbf{0}\}$ is called a parity check matrix of C. For any $\mathbf{y} \in \mathbb{F}_q^n$, the vector $\mathbf{yH}^\top$ is called the syndrome of $\mathbf{y}$ (relatively to $\mathbf{H}$). The dual code of C is $C^\perp = \{\mathbf{xH} \mid \mathbf{x} \in \mathbb{F}_q^{n-k}\}$.

Problem 6.2 (Decoding Problem $-$ $\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w}$)**.** Given a parity check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$, a syndrome $\boldsymbol{s} \in \mathbb{F}_q^{n-k}$ and a target weight $w \in [|0, n|]$, find a vector $\boldsymbol{e} \in \mathbb{F}_q^n$ such that $|\boldsymbol{e}| = w$ and $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$.

The problem $\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w}$ is hard on average for $\mathbf{H}$ uniformly distributed in $\mathbb{F}_q^{(n-k) \times n}$ and $\boldsymbol{s} = \boldsymbol{e}\mathbf{H}^\top$ with $\boldsymbol{e}$ a uniform vector in $\mathbb{F}_q^n$ of weight $|\boldsymbol{e}| = w$. The best known algorithms have a polynomial complexity when $\frac{q-1}{q}(n-k) \leq w < k + \frac{q-1}{q}(n-k)$, and exponential otherwise. Notice that to find a codeword with a given target weight, one can solve an instance of $\mathsf{DP}_{\mathbf{H},\boldsymbol{s}=\mathbf{0},w}$. This problem is believed to be as hard as $\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w}$ with an arbitrary $\boldsymbol{s}$.

Proposition 6.3. *For a uniformly random matrix $\boldsymbol{H} \in \mathbb{F}_3^{(n-k) \times n}$, we expect the solutions to the $\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w}$ problem to be on average $\binom{n}{w} \frac{2^w}{3^{n-k}}$.*

Proof. There are $\binom{n}{w} 2^w$ words of length n and weight w in $\mathbb{F}_3$. For some $\boldsymbol{e} \in \mathbb{F}_3^n$ and $\mathbf{H} \in \mathbb{F}_3^{(n-k) \times n}$, the vector $\boldsymbol{e}\mathbf{H}^\top$ has 3^{n-k} possible values, so the probability that for a given $\boldsymbol{e}$ it gives the correct one is $\frac{1}{3^{n-k}}$. □

Remark. There does not necessarily exist a solution to generic instances of the DP problem. But in the Wave settings, there is always at least one solution on average, and there are even exponentially many ones.

Let us introduce some notions about code specific to the Wave signature scheme [DST19].

Definition 6.4 (Generalized ternary $(U, U+V)$-code)**.** We consider integers n, k, k_U, k_V with n even such that $n > k > 0$, $k = k_U + k_V$, $0 < k_U < n/2$ and $0 < k_V < n/2$. For i from 0 to $n/2$, let $\mathbf{a} = (a_i)_i$, $\mathbf{b} = (b_i)_i$, $\mathbf{c} = (c_i)_i$ and $\mathbf{d} = (d_i)_i$ denote vectors in $\mathbb{F}_3^{n/2}$ such that $\forall i \in [0, n/2]$, $a_i c_i \neq 0$ and $a_i d_i - b_i c_i \neq 0$.

The ternary linear codes U (resp. V) are of length $n/2$ and dimension k_U (resp. k_V) and admits generator matrix $\mathbf{G}_U$ and parity check matrix $\mathbf{H}_U \in \mathbb{F}_3^{(n/2-k_U) \times n/2}$ (resp. $\mathbf{G}_V$ and $\mathbf{H}_V \in \mathbb{F}_3^{(n/2-k_V) \times n/2}$). Then, the generalized ternary $(U, U+V)$-code C associated to $(\mathbf{H}_U, \mathbf{H}_V, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ has the following parity check matrix

$$\mathbf{H} = \begin{pmatrix} \mathbf{d} \star \mathbf{H}_U & -\mathbf{b} \star \mathbf{H}_U \\ -\mathbf{c} \star \mathbf{H}_V & \mathbf{a} \star \mathbf{H}_V \end{pmatrix}.$$

The dual of code associated to $(\mathbf{H}_U, \mathbf{H}_V, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ is a $(U, U+V)$-code associated to $(\mathbf{G}_U, \mathbf{G}_V, -\mathbf{c}, \mathbf{d}, \mathbf{a}, -\mathbf{b})$.

Definition 6.5 (Type-U and type-V codewords)**.** We consider a generalized ternary $(U, U+V)$-code C and retake the above notations. Given a chosen target weight t, we call a type-U codeword in C a word $\mathbf{u} \in U$

of form $\vec{\boldsymbol{u}} = (\mathbf{a} \star \mathbf{u} \| \mathbf{c} \star \mathbf{u})$ and of weight $|\mathbf{u}| = t/2$. And a type-V codeword in C is a word $\mathbf{v} \in V$ of form $\mathbf{v} = (\mathbf{b} \star \mathbf{v} \| \mathbf{d} \star \mathbf{v})$ and of weight $|\mathbf{v}| = t/2$.

Wave uses a permuted generalized ternary $(U, U+V)$-code of length n and dimension k admitting a parity check matrix $\mathbf{H} \in \mathbb{F}_3^{(n-k)\times n}$ that constitutes the public key. Are also fixed a weight w, and dimensions k_U for code U and k_V for V. The signature of a message m by Wave is an $\boldsymbol{e} \in \mathbb{F}_3^n$ such that $|\boldsymbol{e}| = w$ and $\boldsymbol{e}\mathbf{H}^\top = h(m) \in \mathbb{F}_3^{(n-k)}$, where h is a hash function. A signer with their secret key U, V can use them to efficiently compute such a $\boldsymbol{e}$ to sign their message m.

Proposition 6.6 ([Sen23] p.6)**.** *Consider a generalized ternary $(U, U+V)$-code C whose code U has dimension k_U and V dimension k_V. For a target weight t, we expect the number of type-U codewords of C to be on average $\binom{n/2}{t/2}\frac{2^{t/2}}{3^{n/2-k_U}}$, and the number of type-V codewords of C to be on average $\binom{n/2}{t/2}\frac{2^{t/2}}{3^{n/2-k_V}}$.*

Key attacks. From the proposition just above, for some values of weight t the number of type-U codewords is higher than those expected for a random code, given by Proposition 6.3. Then, one can use this fact to exhibit a type-U or type-V codeword, that provides a distinguisher of the Wave public key from the uniform, namely solving the DWK_{n,k_U,k_V} problem. This is the goal of *key attacks* on Wave. Notice that one can run this attack directly on the $(U, U+V)$-code but also on its dual code. We draw the attention of the reader to the work of [Sen23] for further details on type-U and type-V words.

Problem 6.7 (The Distinguishing Wave Keys Problem DWK_{n,k_U,k_V})**.** Given $\mathbf{H} \in \mathbb{F}_3^{(n-(k_U+k_V))\times n}$, decide whether $\mathbf{H}$ has been chosen uniformly at random or among parity-check matrices of permuted generalized $(U, U+V)$-codes where U has dimension k_U, and V dimension k_V.

Message attacks. Another way to attack Wave is by forging a message-signature pair $(\boldsymbol{e}, \boldsymbol{s}) \in \mathbb{F}_3^n \times \mathbb{F}_3^{n-k}$ such that $|\boldsymbol{e}| = w$ and $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$. This consists in solving the $\mathsf{DOOM}_{n,k,w}$ problem, which is hard if $\mathsf{DP}_{n,k,w}$ is hard. This problem was introduced in [JJ02] for $\mathbb{F}_2$ and [Sen11] presented an approach for solving it.

Problem 6.8 (Decoding One Out of Many $\mathsf{DOOM}_{n,k,w}$)**.** Given an arbitrary large list S of syndromes in $\mathbb{F}_q^{n-k}$, a parity check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ and a target weight w, find $\boldsymbol{s} \in S$ and $\boldsymbol{e} \in \mathbb{F}_q^n$ such that $|\boldsymbol{e}| = w$ and $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$.

6.2.2. Information Set Decoding (ISD) Framework

Attacks on the Decoding Problem are commonly[1] based on the Information Set Decoding (ISD) framework that received several refinements. Introduced by Prange [Pra62], the idea is to pick a random subset of indices, that gives a submatrix $\mathbf{H}''$ and subsyndrome $\boldsymbol{s}''$, compute the unique $\boldsymbol{e}''$ such that $\boldsymbol{e}''\mathbf{H}'' = \boldsymbol{s}''$, and repeat the process until it forms a complete $\boldsymbol{e}$ with good weight $|\boldsymbol{e}| = w$. Stern and Dumer [Ste88; Dum91] improved it by taking advantage of the Generalized Birthday Paradox, and Schroeppel and Shamir [SS81] extended this idea by using Wagner's approach [Wag02]. We use here a framework similar to [FS09], which uses the parity check matrix instead of the generator matrix. Another variant uses representation techniques [MMT11; Bec+12], but this refinement only provides a very slight gain in the Wave setting as shown in [Bri+20]. With nearest-neighbour techniques [MO15; BM17; BM18; Car+22] the gain in asymptotic factors is compensated by a high overhead. For these reasons, we will not deal with these techniques in this thesis and restrict our cryptanalysis to algorithms [Pra62; Dum91; SS81]. We refer the reader to [Deb23] for a more detailed introduction to code-based theory and ISD algorithms, and more specifically for the successive generalizations of the ISD algorithms you may look at [Eti23].

To solve the problem $\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w}$, the idea behind ISD is to rewrite $\mathbf{H}$ into a systematic form and then to solve an easier instance $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$, where $\mathbf{H}'' \in \mathbb{F}_q^{(k+\ell)\times\ell}$ and $\boldsymbol{s}'' \in \mathbb{F}_q^\ell$ for parameters ℓ the length of the $\boldsymbol{s}''$

[1] A recent paper [Car+22] presented a way to make the statistical decoding [Jab01] perform better than ISD algorithms in some regimes. Except for this algorithm, all the known attacks on DP for the sixty last years were based on the ISD framework.

and p the target weight of $\boldsymbol{e}''$. We need to find many solutions $\boldsymbol{e}'' \in \mathbb{F}_q^{k+\ell}$ to the subproblem $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$ to hope to get one of them that gives a complete solution $\boldsymbol{e} = (\boldsymbol{e}'\|\boldsymbol{e}'') \in \mathbb{F}_3^n$ to the $\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w}$ problem.

In order to analyse the complexity of the ISD framework, we will need the following lemma.

Lemma 6.9. *Let $\boldsymbol{e} \in \mathbb{F}_3^n$ be a vector of weight w and parameters ℓ, p. Let us define the subvectors $\boldsymbol{e}' \in \mathbb{F}_3^{n-k-\ell}$ and $\boldsymbol{e}'' \in \mathbb{F}_3^{k+\ell}$ such that $\boldsymbol{e} = (\boldsymbol{e}'\|\boldsymbol{e}'')$. We say that $\boldsymbol{e}$ is well cut if $|\boldsymbol{e}'| = w - p$ and $|\boldsymbol{e}''| = p$. The probability that a random $\boldsymbol{e} \in \mathbb{F}_3^n$ of weight w is well cut is*

$$PrGoodCut = \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{w-p}}{\binom{n}{w}}. \tag{6.1}$$

Classical ISD algorithm.

Theorem 6.10. *We are given a classical algorithm that finds $NbSolFound$ solutions to $\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p}$ in time $T_{\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p}}$, among the $NbSol(\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p})$ total solutions to $\mathsf{DP}_{k+\ell,\ell,p}$. $PrGoodCut$ is defined as in Proposition 6.9, and $NbSol(\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w})$ denotes the number of solutions to the $\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w}$ problem. Then the classical Information Set Decoding framework (Algorithm 17) solves $\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w}$ in time*

$$T_{DP} = \max\left\{ T_{\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p}}, \frac{T_{\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p}}}{NbSol(\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w}) \cdot PrGoodCut \cdot \frac{NbSolFound}{NbSol(\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p})}} \right\}.$$

The term on the left inside the max is the complexity when only one iteration of Steps 1-4 in the ISD suffices, while the term on the right is the one for several iterations.

Algorithm 17 Classical ISD

Input: $\mathbf{H}_0 \in \mathbb{F}_3^{n\times(n-k)}$, syndrome $\boldsymbol{s} \in \mathbb{F}_3^{n-k}$, weight w.
Parameters $\ell \in [0, n-k]$ and $p \in [\max\{0, w-(n-k-\ell)\}, \min\{w, k+\ell\}]$
Output: $\boldsymbol{e} \in \mathbb{F}_3^n$ such that $\boldsymbol{e}\mathbf{H}_0^\top = \boldsymbol{s}$ and $|\boldsymbol{e}| = w$.

1: Pick a **random permutation of columns** π and apply $\mathbf{H} \leftarrow \pi(\mathbf{H}_0)$
2: Apply a **partial Gaussian Elimination** on $\mathbf{H}$ to transform it into a systematic form $\mathbf{H} = \left(\begin{array}{c|c} I_{n-k-\ell} & \mathbf{H}' \\ \hline \mathbf{0} & \mathbf{H}'' \end{array}\right) \in \mathbb{F}_q^{n\times(n-k)}$ where $\mathbf{H}' \in \mathbb{F}_q^{(k+\ell)\times(n-k-\ell)}$ and $\mathbf{H}'' \in \mathbb{F}_q^{(k+\ell)\times\ell}$, and a syndrome $\boldsymbol{s} = (\boldsymbol{s}'\|\boldsymbol{s}'') \in \mathbb{F}_q^{n-k}$ with $\boldsymbol{s}' \in \mathbb{F}_q^{n-k-\ell}$ and $\boldsymbol{s}'' \in \mathbb{F}_q^{\ell}$.
3: **Solve the subproblem** $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$: Construct a list of vectors $(\boldsymbol{e}'', \boldsymbol{e}''\mathbf{H}''^\top) \in \mathbb{F}_q^{k+\ell} \times \mathbb{F}_q^{\ell}$ such that $|\boldsymbol{e}''| = p$ and $\boldsymbol{e}''\mathbf{H}''^\top = \boldsymbol{s}''$.
4: **Test step**. For each $\boldsymbol{e}''$ found during Step 3, recover the complete vector $\boldsymbol{e} = (\boldsymbol{e}'\|\boldsymbol{e}'')$ that satisfies $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$, and check if $|\boldsymbol{e}| = w$.
5: **Repeat** Steps 1-4 until Step 4 succeeds and gives a $\boldsymbol{e} \in \mathbb{F}_3^n$ such that $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$ and $|\boldsymbol{e}| = w$.
6: **return** $\boldsymbol{e}_0 = \pi^{-1}(\boldsymbol{e})$. It verifies $\boldsymbol{e}_0\mathbf{H}_0^\top = \boldsymbol{s}$ and $|\boldsymbol{e}_0| = w$.

Proof. We use the same notations as above.

Steps 1-2. Applying a random permutation of columns and a partial Gaussian elimination on $\mathbf{H}$ can be done in polynomial time.

Step 3. This step takes time $T_{\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}}$ that depends on the choice of the subroutine. It returns $NbSolFound$ solutions to the $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$ subproblem.

Step 4 From an $\boldsymbol{e}'' \in \mathbb{F}_3^{k+\ell}$ such that $\boldsymbol{e}''\mathbf{H}''^\top = \boldsymbol{s}''$, one can efficiently compute $\boldsymbol{e}' = \boldsymbol{s}' - \boldsymbol{e}''\mathbf{H}'^\top \in \mathbb{F}_3^{n-k-\ell}$. The vector $\boldsymbol{e} = (\boldsymbol{e}'\|\boldsymbol{e}'')$ then satisfies $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$. There are $NbSolFound$ solutions that need to be checked for weight. The check time is dominated in the sum by the time of Step 3.

Step 5. Suppose there is a precise solution $\boldsymbol{e}$ that we want to find where $\boldsymbol{e} = (\boldsymbol{e}'\|\boldsymbol{e}'')$ with $\boldsymbol{e}' \in \mathbb{F}_3^{k+\ell}$ and $\boldsymbol{e}'' \in \mathbb{F}_3^{n-k-\ell}$. The probability that $\boldsymbol{e}$ is "well cut" *i.e.* for $\boldsymbol{e}$ there is $|\boldsymbol{e}'| = w - p$ and $|\boldsymbol{e}''| = p$, is given

by Lemma 6.9. Then, supposing $\boldsymbol{e}$ is well cut, one iteration of steps (1-4) returns a list containing a fraction $\frac{NbSolFound}{NbSol(\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p})}$ of the solutions to the $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$ subproblem. As there are $NbSol(\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w})$ such solutions $\boldsymbol{e}$, the probability that one iteration returns a solution is

$$PrFindSol = \min\left\{1, NbSol(\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w}) \cdot PrGoodCut \cdot \frac{NbSolFound}{NbSol(\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p})}\right\}. \tag{6.2}$$

To get a solution with probability $1-o(1)$, one has to repeat steps (1-4) a number $1/PrFindSol$ of iterations. Once we have found a solution $\boldsymbol{e}$ such that $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$, then $\pi^{-1}(\boldsymbol{e})\mathbf{H}_0^\top$, and this achieves the algorithm.

Putting everything together gives the result. □

Quantum ISD algorithm.

Notations. We recall that given a quantumly accessible list L, $\mathrm{ind}_L(\boldsymbol{x})$ denotes the index of element $\boldsymbol{x}$ in the list L. The quantum superposition of a list L is the quantum state $|\psi_L\rangle = \frac{1}{\sqrt{|L|}}\sum_{\boldsymbol{x}\in L}|\mathrm{ind}_L(\boldsymbol{x})\rangle|\boldsymbol{x}\rangle$ (See Definition 2.6).

Theorem 6.11. *We are given an algorithm that constructs a quantum superposition of $NbSolFound$ solutions to $\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p}$ in time $T_{\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p}}$, among the $NbSol(\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p})$ total solutions to this subproblem. $PrGoodCut$ is defined in Proposition 6.9, and $NbSol(\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w})$ denotes the number of solutions to $\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w}$. Then, Algorithm 18 solves $\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w}$ in quantum time*

$$T_{DP} = \max\left\{T_{\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p}}, \frac{T_{\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p}}}{\sqrt{NbSol(\mathsf{DP}_{\boldsymbol{H},\boldsymbol{s},w}) \cdot PrGoodCut \cdot \frac{NbSolFound}{NbSol(\mathsf{DP}_{\boldsymbol{H}'',\boldsymbol{s}'',p})}}}\right\}.$$

Algorithm 18 Quantum ISD

Input: $\mathbf{H}_0 \in \mathbb{F}_3^{n\times(n-k)}$, syndrome $\boldsymbol{s} \in \mathbb{F}_3^{n-k}$, weight w.
Parameters $\ell \in [0, n-k]$ and $p \in [\max\{0, w-(n-k-\ell)\}, \min\{w, k+\ell\}]$
Output: $\boldsymbol{e}_0 \in \mathbb{F}_3^n$ such that $\boldsymbol{e}_0\mathbf{H}_0^\top = \boldsymbol{s}$ and $|\boldsymbol{e}_0| = w$.

1: Pick a **random permutation of columns** π and apply $\mathbf{H} \leftarrow \pi(\mathbf{H}_0)$
2: Apply a **partial Gaussian Elimination** on $\mathbf{H}$ to transform it into a systematic form $\mathbf{H} = \left(\begin{array}{c|c} I_{n-k-\ell} & \mathbf{H}' \\ \hline \mathbf{0} & \mathbf{H}'' \end{array}\right) \in \mathbb{F}_q^{n\times(n-k)}$ where $\mathbf{H}' \in \mathbb{F}_q^{(k+\ell)\times(n-k-\ell)}$ and $\mathbf{H}'' \in \mathbb{F}_q^{(k+\ell)\times\ell}$, and a syndrome $\boldsymbol{s} = (\boldsymbol{s}'\|\boldsymbol{s}'') \in \mathbb{F}_q^{n-k}$ with $\boldsymbol{s}' \in \mathbb{F}_q^{n-k-\ell}$ and $\boldsymbol{s}'' \in \mathbb{F}_q^{\ell}$.
3: **Solve the subproblem** $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$: Apply the procedure that constructs in quantum superposition a list L of $\boldsymbol{e}'' \in \mathbb{F}_3^{k+\ell}$ such that $|\boldsymbol{e}''| = p$ and $\boldsymbol{e}''\mathbf{H}''^\top = \boldsymbol{s}''$, *i.e.*

$$|0\rangle \to \frac{1}{\sqrt{|L|}} \sum_{(\boldsymbol{e}'',\boldsymbol{y})\in L} |\mathrm{ind}_L(\boldsymbol{e}'')\rangle|\boldsymbol{e}''\rangle,$$

where $\mathrm{ind}_L(\boldsymbol{e}'')$ is the index of the tuple $(\boldsymbol{e}'', \boldsymbol{y} = \boldsymbol{e}''\mathbf{H}''^\top)$ in list L.
4: **Test step**. From a vector $\boldsymbol{e}''$ we can compute the complete candidate solution $\boldsymbol{e} \in \mathbb{F}_3^n$ such that $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$, so there exists a procedure

$$|0\rangle \to |\Psi\rangle := \frac{1}{\sqrt{|L|}} \sum_{\substack{(\boldsymbol{e}'',\boldsymbol{y})\in L \\ \boldsymbol{e}=(\boldsymbol{s}'-\boldsymbol{e}''\mathbf{H}'^\top\|\boldsymbol{e}'')}} |\mathrm{ind}_L(\boldsymbol{e}'')\rangle|\boldsymbol{e}\rangle.$$

The vectors $\boldsymbol{e}$ in the second register then satisfy $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$. Apply **Grover**, iterating on the operation $|0\rangle \to |\Psi\rangle$, to only keep the $\boldsymbol{e}$'s in the superposition which are of weight w.
5: Apply a **Amplitude Amplification** on steps 1-4 to find a good permutation π in Step 1 with high probability.
6: **Measure** $\boldsymbol{e}$. Return $\boldsymbol{e}_0 = \pi^{-1}(\boldsymbol{e})$. It satisfies $\boldsymbol{e}_0\mathbf{H}_0^\top = \boldsymbol{s}$ and $|\boldsymbol{e}_0| = w$.

Proof. **Steps 1,2.** These steps do not change from the classical version and are efficiently done.

Step 3. This operation takes time $T_{\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}}$ that depends on the choice of the subroutine. It returns a quantum superposition $|\psi_L\rangle$ over $|L| = NbSolFound$ solutions to the $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$ subproblem. All the $\boldsymbol{y}$ in tuples in L in this state are equal to $\boldsymbol{y} = \boldsymbol{s}''$ as it is an output condition of the subroutine. So we can discard this last register that can now be considered classical.

Step 4. After discarding the classical register $|\boldsymbol{s}''\rangle$, we add a zero quantum register to $|\psi_L\rangle$ to get the state

$$\frac{1}{\sqrt{|L|}} \sum_{(\boldsymbol{e}'',\boldsymbol{y})\in L} |\mathrm{ind}_L(\boldsymbol{e}'')\rangle|0\rangle|\boldsymbol{e}''\rangle$$

where $\mathrm{ind}_L(\boldsymbol{e}'')$ is the index of the tuple $(\boldsymbol{e}'', \boldsymbol{e}''\mathbf{H}''^\top)$ in list L.

We apply the efficient quantum circuit $|0\rangle|\boldsymbol{e}''\rangle \mapsto |\boldsymbol{s}' - \mathbf{H}'\boldsymbol{e}''\rangle|\boldsymbol{e}''\rangle$ on its two last registers to get the state

$$|\Psi\rangle := \frac{1}{\sqrt{|L|}} \sum_{\substack{(\boldsymbol{e}'',\boldsymbol{y})\in L \\ \boldsymbol{e}'=\boldsymbol{s}'-\mathbf{H}'\boldsymbol{e}''}} |\mathrm{ind}_L(\boldsymbol{e}'')\rangle|\boldsymbol{e}'\rangle|\boldsymbol{e}''\rangle = \frac{1}{\sqrt{|L|}} \sum_{\substack{(\boldsymbol{e}'',\boldsymbol{y})\in L \\ \boldsymbol{e}=(\boldsymbol{s}'-\mathbf{H}'\boldsymbol{e}''\|\boldsymbol{e}'')}} |\mathrm{ind}_L(\boldsymbol{e}'')\rangle|\boldsymbol{e}\rangle.$$

This state is a uniform quantum superposition over candidate solutions $\boldsymbol{e} \in \mathbb{F}_3^n$ that satisfy $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$. We need to only keep those that are of good weight w, so we apply a Grover search [Gro96] that iterates the procedure $|0\rangle \to |\Psi\rangle$. Tts check function takes $\boldsymbol{e}$ and returns 1 if $|\boldsymbol{e}| = w$, and 0 otherwise. It returns the

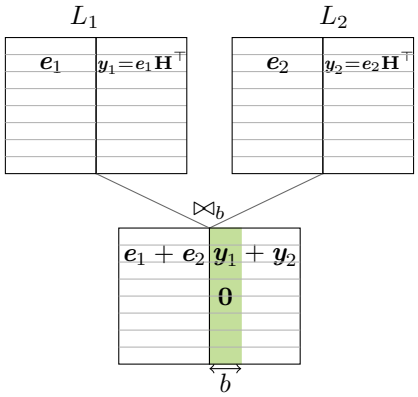


Figure 6.1: Merging lists L_1 and L_2 on the b first coordinates.

following quantum superposition over solutions to $\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w}$

$$\frac{1}{\sqrt{Z}} \sum_{\substack{(\boldsymbol{e}'',\boldsymbol{y})\in L \\ e=(\boldsymbol{s}'-\mathbf{H}'\boldsymbol{e}''\|\boldsymbol{e}'') \\ |e|=w}} |\mathrm{ind}_L(\boldsymbol{e}'')\rangle|\boldsymbol{e}\rangle,$$

where Z is the number of such solutions $\boldsymbol{e}$. This requires at most $\sqrt{|L|} = \sqrt{NbSolFound}$ Grover iterations.

Step 5. Suppose there is a precise solution $\boldsymbol{e}$ that we want to find where $\boldsymbol{e} = (\boldsymbol{e}'\|\boldsymbol{e}'')$ with $\boldsymbol{e}' \in \mathbb{F}_3^{k+\ell}$ and $\boldsymbol{e}'' \in \mathbb{F}_3^{n-k-\ell}$. Lemma 6.9 gives the probability that $\boldsymbol{e}$ is "well cut", *i.e.* $|\boldsymbol{e}'| = w - p$ and $|\boldsymbol{e}''| = p$. Then, supposing $\boldsymbol{e}$ is well cut, one iteration of steps (1-4) returns a list in quantum superposition containing a fraction $\frac{NbSolFound}{NbSol(\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p})}$ of the solutions to the $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$ subproblem. One iteration of Steps 1-4 returns a solution with probability $PrFindSol$ whose expression is in Equation 6.2. To get a solution with a probability close to 1, we apply an amplitude amplification on this process, which takes $1/\sqrt{PrFindSol}$ iterations. Then we measure and find a solution to $\mathsf{DP}_{\mathbf{H},\boldsymbol{s},w}$. □

DOOM variant of the ISD.

[Sen11] presented an approach for solving more efficiently the DOOM problem. Instead of having only one syndrome $\boldsymbol{s}$ in the input of the ISD frameworks 17 and 18, the adversary takes an arbitrarily large list S of syndromes. At the end of the algorithm, the adversary wins if they get a pair $(\boldsymbol{e}_0, \boldsymbol{s}) \in \mathbb{F}_3^n \times S$ such that $\boldsymbol{e}_0\mathbf{H}_0 = \boldsymbol{s}$. The subroutine of the third step is also adapted in function: it takes in input a list S'' of the restricted syndromes $\boldsymbol{s}''$, and outputs solutions $(\boldsymbol{e}'', \boldsymbol{s}'') \in \mathbb{F}_3^{k+\ell} \times \mathbb{F}_3^{\ell}$ to the subproblem, where $\boldsymbol{s}''$ are restrictions of the $\boldsymbol{s} \in S$ on their ℓ last coordinates. The time complexity of this variant stays the same as the one given in Theorem 6.10 for classical and in Theorem 6.11 for quantum. This approach will be applied and explained in more detail in Section 6.4 in the context of message attacks on Wave.

6.2.3. List merging

Subroutines within the ISD algorithms will make great use of list merging. Merging two lists L_1 and L_2 on the b first coordinates means constructing the following list.

$$L_1 \bowtie_b L_2 := \left\{(\boldsymbol{e}_1 + \boldsymbol{e}_2, \boldsymbol{y}_1 + \boldsymbol{y}_2) \ : \ (\boldsymbol{e}_1, \boldsymbol{y}_1) \in L_1,\ (\boldsymbol{e}_2, \boldsymbol{y}_2) \in L_2,\ (\boldsymbol{y}_1 + \boldsymbol{y}_2)_{|_{[0;b]}} = \mathbf{0}\right\} \tag{6.3}$$

Size of the merged list.

For lists L_1 and L_2 randomly sampled in $\mathbb{F}_3^n \times \mathbb{F}_3^\ell$, their merged list is of expected size $|L_1 \bowtie_b L_2| = \frac{|L_1|\cdot|L_2|}{3^b}$ on average. Then for lists L_1, L_2 already merged so that their vectors have already their b_0 first coordinates at zero, we have on average for $b \geq b_0$,

$$|L_1 \bowtie_b L_2| = \frac{|L_1| \cdot |L_2|}{3^{b-b_0}}. \tag{6.4}$$

Classical merging.

We want to construct the merged list $L = L_1 \bowtie_b L_2$. We sort L_1 by lexicographic order according to its second tuple elements $\boldsymbol{y}_1$, which takes time $|L_1|\cdot\log(|L_1|)$. Then, for each $(\boldsymbol{e}_2, \boldsymbol{y}_2) \in E_2$, we search $(\boldsymbol{e}_1, \boldsymbol{y}_1) \in L_1$ such that $\boldsymbol{y}_1 + \boldsymbol{y}_2$ values $\mathbf{0}$ on its b first coordinates. Thanks to the sorting, for each $\boldsymbol{e}_2$ one can find a solution in L_2 (if it exists) in time $\log|L_1|$ by dichotomic search. For each collision found on $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$, we add $(\boldsymbol{e}_1 + \boldsymbol{e}_2, \boldsymbol{y}_1 + \boldsymbol{y}_2)$ to L. So the classical merging takes time $(|L_1| + |L_2|)\cdot\log|L_1|$. Hence the following lemma.

Lemma 6.12. *Given lists L_1 and L_2, one can construct the list $L_1 \bowtie_b L_2$ for an arbitrary b in time $\widetilde{\mathcal{O}}(|L_1|, |L_2|, |L_1 \bowtie_b L_2|)$.*

Quantum merging.

We are given a list L_1 classically stored and assumed quantumly accessible, and a procedure $|0\rangle \to |\psi_{L_2}\rangle$ that returns in time T the uniform quantum superposition on the list L_2,

$$|\psi_{L_2}\rangle = \frac{1}{\sqrt{|L_2|}} \sum_{(\boldsymbol{e}_2,\boldsymbol{y}_2)\in L_2} |\mathrm{ind}_{L_2}(\boldsymbol{e}_2)\rangle|\boldsymbol{e}_2\rangle|\boldsymbol{y}_2\rangle. \tag{6.5}$$

We sort L_1 in the lexicographic order according to its second tuple elements $\boldsymbol{y}_1$, which takes time $|L_1| \cdot \log(|L_1|)$. We define the following function:

$$match_{L_1}(\boldsymbol{e}_2, \boldsymbol{y}_2) = \begin{cases} (\boldsymbol{e}_1, \boldsymbol{y}_1) \in L_1 \text{ such that } (\boldsymbol{y}_1 + \boldsymbol{y}_2)_{|[0:b]} = \mathbf{0} \text{ if it exists,} \\ \perp \text{ otherwise.} \end{cases}$$

If several such $(\boldsymbol{e}_1, \boldsymbol{y}_1)$'s match, the function will arbitrarily return the first one by lexicographic order. However, if lists L_1, L_2 are random and there is $|L_1| \leq |L_2|$, then there will be on average at most one such tuple in L_1. So we make this assumption by simplicity from here[2].

The function $match_{L_1}$ is efficiently implementable by performing a dichotomic search as L_1 is sorted and assumed quantumly accessible. We then apply this circuit on $|\psi_{L_2}\rangle$ using an auxiliary register:

$$\frac{1}{\sqrt{|L_2|}} \sum_{(\boldsymbol{e}_2,\boldsymbol{y}_2)\in L_2} |\mathrm{ind}_{L_2}(\boldsymbol{e}_2)\rangle|match_{L_1}(\boldsymbol{e}_2, \boldsymbol{y}_2)\rangle|\boldsymbol{e}_2\rangle|\boldsymbol{y}_2\rangle.$$

While the classical merging ran a for loop on L_2 to check, in the quantum model we replace it with Grover's search [Gro96] that iterates on the procedure $|0\rangle \to |\psi_{L_2}\rangle$. We define the Grover check function as follows: Given $(\boldsymbol{e}_2, \boldsymbol{y}_2)$, it returns 1 if $match_{L_1}(\boldsymbol{e}_2, \boldsymbol{y}_2) \neq \perp$, and 0 else. Applying Grover requires at most $\sqrt{|L_2|}$ iterations, and returns the following state, where the non-$\perp$ elements are removed from the superposition.

$$\frac{1}{\sqrt{|L_1 \bowtie_b L_2|}} \sum_{\substack{(\boldsymbol{e}_2,\boldsymbol{y}_2)\in L_2 \\ match_{L_1}(\boldsymbol{e}_2,\boldsymbol{y}_2)=(\boldsymbol{e}_1,\boldsymbol{y}_1)\neq\perp}} |\mathrm{ind}_{L_2}(\boldsymbol{e}_2)\rangle|\boldsymbol{e}_1\rangle|\boldsymbol{y}_1\rangle|\boldsymbol{e}_2\rangle|\boldsymbol{y}_2\rangle.$$

[2]When we will apply quantum merging further in this work, we will manipulate random lists L_1, L_2 such that $|L_1|^2 = |L_2|$, so there will be at most one solution with very high probability. This allows us to consider that this quantum merging process constructs a quantum superposition over the list $L_1 \bowtie_b L_2$ without missing any element.

And finally, by simply summing, swapping and reassembling the registers, we get the state

$$\frac{1}{\sqrt{|L_1 \bowtie_b L_2|}} \sum_{\substack{(\boldsymbol{e}_2,\boldsymbol{y}_2)\in L_2 \\ match_{L_1}(\boldsymbol{e}_2,\boldsymbol{y}_2)=(\boldsymbol{e}_1,\boldsymbol{y}_1)\neq\perp}} |\mathrm{ind}_{L_2}(\boldsymbol{e}_2)\rangle|\boldsymbol{e}_1+\boldsymbol{e}_2\rangle|\boldsymbol{y}_1+\boldsymbol{y}_2\rangle|\boldsymbol{e}_2,\boldsymbol{y}_2\rangle,$$

where the last register $|\boldsymbol{e}_2, \boldsymbol{y}_2\rangle$ cannot be discarded because of the requirement of the reversibility of the process, but it will not be used anymore. The previous state then can be rewritten

$$|\psi_{L_1\bowtie_b L_2}\rangle|\mathrm{Aux}\rangle := \frac{1}{\sqrt{|L_1 \bowtie_b L_2|}} \sum_{\substack{(\boldsymbol{e},\boldsymbol{y})\in L_1\bowtie_b L_2 \\ \boldsymbol{e}=\boldsymbol{e}_1+\boldsymbol{e}_2,\boldsymbol{e}_1\in L_1,\boldsymbol{e}_2\in L_2}} |\mathrm{ind}_{L_2}(\boldsymbol{e}_2)\rangle|\boldsymbol{e}\rangle|\boldsymbol{y}\rangle|\mathrm{Aux}\rangle. \tag{6.6}$$

This whole process takes time $(|L_1| + T\sqrt{|L_2|}) \cdot \log|L_1|$.

Lemma 6.13. *Given a list L_1 classically stored and quantumly accessible, and a procedure that returns the quantum state $|\psi_{L_2}\rangle$ (Eq. 6.5) in time T, for $|L_1| \leq |L_2|$, there exists a quantum algorithm that returns the state $|\psi_{L_1\bowtie_b L_2}\rangle$ (Eq. 6.6) for an arbitrary b in time $\widetilde{\mathcal{O}}(|L_1|, T\sqrt{|L_2|})$.*

6.3. Key attacks on DWK

We are given a public $(U, U+V)$-code with generator matrix $\mathbf{G} \in \mathbb{F}_3^{(k_U+k_V)\times n}$ and parity check matrix $\mathbf{H} \in \mathbb{F}_3^{(n-(k_U+k_V))\times n}$. The point of this attack is to solve the Distinguishing Wave Keys Problem 6.7, which can be done by finding a type-U or type-V word $\boldsymbol{e}$ of weight t in the public code or its dual. [Sen23] pointed out that type-U words outnumber type-V ones, so the attacker can restrain their search to only type-U words as they are easier to find. The parameter t can be chosen as the attacker wants under the condition that the number of such words has to be higher than in a random code. The former condition, by combining Propositions 6.3 and 6.6, is equivalent to

$$3^{n-2\cdot k_V} > \binom{n}{t} 2^t. \tag{6.7}$$

So the time complexity of this key attack is the minimum between the time of solving the problems $\mathsf{DP}_{\mathbf{H},\mathbf{0},t}$ and $\mathsf{DP}_{\mathbf{G},\mathbf{0},t'}$, respectively to find a type-U word in the public code and in its dual, with t and t' are freely chosen such that they respect Equation 6.7. In this section, we present how to solve $\mathsf{DP}_{\mathbf{H},\mathbf{0},t}$ and these algorithms can directly be adapted to the dual version.

6.3.1. Classical key attack

The best known classical key attack on Wave is due to [Sen23], who applies Dumer's algorithm [Dum91] within the ISD framework [FS09]. We start by constructing the following lists.

$$\begin{aligned} E_1 &:= \{(\boldsymbol{x}_1 \parallel \mathbf{0}^{\frac{k+\ell}{2}}) \mid \boldsymbol{x}_1 \in \mathbb{F}_3^{\frac{k+\ell}{2}}, |\boldsymbol{x}_1| = p/2\} \quad ; \quad L_1 := \{(\boldsymbol{e}_1'', \boldsymbol{e}_1''\mathbf{H}''^\top) : \boldsymbol{e}_1'' \in E_1\} \\ E_2 &:= \{(\mathbf{0}^{\frac{k+\ell}{2}} \parallel \boldsymbol{x}_2) \mid \boldsymbol{x}_2 \in \mathbb{F}_3^{\frac{k+\ell}{2}}, |\boldsymbol{x}_2| = p/2\} \quad ; \quad L_2 := \{(\boldsymbol{e}_2'', \boldsymbol{e}_2''\mathbf{H}''^\top) : \boldsymbol{e}_2'' \in E_2\} \end{aligned} \tag{6.8}$$

Both these initial lists are of size

$$\binom{(k+\ell)/2}{p/2} 2^{p/2} = \widetilde{\mathcal{O}}\left(\binom{k+\ell}{p}^{1/2} 2^{p/2}\right). \tag{6.9}$$

We apply classical merging from Lemma 6.12 on the lists L_1 and L_2 to get the merged list $L_1 \bowtie_\ell L_2$ filled with elements of form $(\boldsymbol{e}'', \boldsymbol{e}''\mathbf{H}''^\top) = (\boldsymbol{e}'', \mathbf{0})$. We can see in Equation 6.8 that vectors $\boldsymbol{e}_1'' \in E_1$ and $\boldsymbol{e}_2'' \in E_2$

have disjoint sets of non-zero coordinates. Therefore, their sum $\boldsymbol{e}''$ that we get through the merged list is in form $\boldsymbol{e}'' = \boldsymbol{e}''_1 + \boldsymbol{e}''_2 = (\boldsymbol{x}_1 \| \boldsymbol{x}_2)$ with $|\boldsymbol{x}_1| = |\boldsymbol{x}_2| = p/2$. So all the $\boldsymbol{e}''$ for $(\boldsymbol{e}'', \boldsymbol{e}''\mathbf{H}'^{\top}) \in L$ obtained are of weight p by construction, which ensures the correctness of the algorithm. Using this process as a subroutine within the ISD framework leads to the following theorem.

Theorem 6.14. *We are given a generalized ternary $(U, U+V)$-code C of dimensions (n, k, k_U, k_V). Fix ISD parameters ℓ, p, and a target weight t. There exists a classical algorithm that solves the DWK_{n,k_U,k_V} problem for code C in time*

$$T = \max\left\{ \binom{k+\ell}{p}^{1/2} 2^{p/2}, \frac{3^{n/2-k_U}\binom{n}{t}^{1/2}}{2^{(t-p)/2}\binom{k+\ell}{p}^{1/2}\binom{n-k-\ell}{t-p}} \right\}.$$

Proof. L_1 and L_2 are of the same size given by Equation 6.9. Constructing the initial lists takes time $\mathcal{O}(|L_1|)$ and merging $L_1 \bowtie_\ell L_2$ takes time $\widetilde{\mathcal{O}}(|L_1|)$ by Lemma 6.12. So Dumer's subroutine runs in time

$$T_{\mathsf{DP}_{\mathbf{H}'',s'',p}} = \widetilde{\mathcal{O}}(|L_1|) = \widetilde{\mathcal{O}}\left(\binom{k+\ell}{p}^{1/2} 2^{p/2}\right).$$

The merged list is of expected size $|L_1|\cdot|L_2|\cdot 3^{-\ell} = \widetilde{\mathcal{O}}\left(\binom{k+\ell}{p} 2^p \cdot 3^{-\ell}\right)$, by Equation 6.4. Proposition 6.3 gives the number of solutions to the DP subproblem: $NbSol(\mathsf{DP}_{\mathbf{H}'',\mathbf{0},p}) = \binom{k+\ell}{p} 2^p$. And Proposition 6.6 gives the number of solutions to the DP problem in the $(U, U+V)$-code: $NbSol(\mathsf{DP}_{\mathbf{H},\mathbf{0},t}) = \binom{n/2}{t/2} \frac{2^{t/2}}{3^{n/2-k_V}}$.

Applying Theorem 6.10 with these amounts gives the time complexity of the ISD framework with a Dumer subroutine. Simplifying the expression directly gives the result. □

Remark. We have observed that Wagner algorithm does not perform better than Dumer in this setting, *i.e.* one does not get profit from taking additional levels in the merging tree. The reason is that the condition in Equation 6.7 forces the target weight t to remain quite small. And this impacts the number of vectors one can generate with this weight, which is low in comparison to those with higher weights (as we are in ternary). Additional merging levels are useful when there are sufficiently many vectors, which is not the case here, but it will have an advantage in a different setting, for the message attacks, as we will see in Section 6.4.

Numerical results.

The time complexity is optimal when both initial lists are of maximal size, *i.e.* by fixing p such that $\binom{k+\ell}{p}^{1/2} 2^{p/2} = 3^\ell$. Parameters ℓ and t are obtained by numerical optimization to minimize the time complexity of the attack. As a result, we obtain as optimal parameters $l \approx 0.01$, $t \approx 0.21$ and $p \approx 0.003$. With these values, the ISD algorithm with a Dumer subroutine solves DWK_{n,k_U,k_V} in time $2^{0.0161n+o(n)}$ *i.e.* 2^{138} for the set of Wave parameters (I); in time $2^{0.0165n+o(n)}$ *i.e.* 2^{206} for set (III); and in time $2^{0.0167n+o(n)}$ *i.e.* 2^{274} for (V).

The sets of Wave parameters (I), (III) and (V) can be found in the following table.

With this choice of parameters, the algorithm finds $|L|$ solutions in time $|L|$, so in amortized time $O(1)$ per solution. The $o(n)$ terms above encapsulate the hidden polynomial terms, as our analysis only focused on the asymptotic complexity. These results are summarized in the first column of Table 6.3.

6.3.2. Quantum key attack

The quantum key attack algorithm has a very similar structure to the classical one. We use the quantum version of the ISD framework (Algorithm 18), which performs a quantum Amplitude Amplification (Theorem 2.9

Table 6.1: Sets of Wave parameters as selected in [Ban+23] (NIST submission, round 1) and the corresponding required security levels in the number of bits.

	Classic	Quantum	n	k	w	k_U
(I)	128	64	8576	4288	7668	2966
(III)	192	96	12544	6272	11226	4335
(V)	256	128	16512	8256	14784	5704

[Bra+02]) instead of a classical while loop. This makes a quadratic gain over the number of iterations of the algorithm. Within the ISD framework, we also replace the classical Dumer subroutine with its quantum merging variant.

Let us define the following lists. Note that the list L_2 does not need to be classically written at any moment of the algorithm.

$$\begin{aligned} E_1 &:= \{(\boldsymbol{x}_1 \parallel \mathbf{0}^{\frac{2(k+\ell)}{3}}) \mid \boldsymbol{x}_1 \in \mathbb{F}_3^{\frac{k+\ell}{3}}, |\boldsymbol{x}_1| = p/3\} \ ; \ L_1 := \{(\boldsymbol{e}_1'', \boldsymbol{e}_1''\mathbf{H}''^{\top}) : \boldsymbol{e}_1'' \in E_1\} \\ E_2 &:= \{(\mathbf{0}^{\frac{k+\ell}{3}} \parallel \boldsymbol{x}_2) \mid \boldsymbol{x}_2 \in \mathbb{F}_3^{\frac{2(k+\ell)}{3}}, |\boldsymbol{x}_2| = 2p/3\} \ ; \ L_2 := \{(\boldsymbol{e}_2'', \boldsymbol{e}_2''\mathbf{H}''^{\top}) : \boldsymbol{e}_2'' \in E_2\} \end{aligned} \tag{6.10}$$

The lists are no longer of equal size. Indeed, to balance the running times, the list in quantum superposition is taken quadratically larger than the classical one:

$$|L_1| = \widetilde{\mathcal{O}}\left(\binom{k+\ell}{p}^{1/3} 2^{p/3}\right) \quad \text{and} \quad |L_2| = |L_1|^2. \tag{6.11}$$

The algorithm starts by classically constructing the list L_1. It also constructs the uniform quantum superposition over the elements of L_2: $|\psi_{L_2}\rangle = \frac{1}{\sqrt{|L_2|}} \sum_{(\boldsymbol{e}_2, \boldsymbol{y}_2) \in L_2} |\mathrm{ind}_{L_2}(\boldsymbol{e}_2)\rangle |\boldsymbol{e}_2\rangle |\boldsymbol{y}_2\rangle$, where $\mathrm{ind}_{L_2}(\boldsymbol{e}_2)$ is the index of the tuple $(\boldsymbol{e}_2, \boldsymbol{y})$ in the list L_2. We apply a quantum merging (Lemma 6.13) on L_1 and $|\psi_{L_2}\rangle$ to get the state $|\psi_{L_1 \bowtie_\ell L_2}\rangle$, which contains the quantum superposition of all the $\boldsymbol{e}'' = (\boldsymbol{e}_1'' + \boldsymbol{e}_2'')$ for $\boldsymbol{e}_1'' \in E_1$ and $\boldsymbol{e}_2'' \in E_2$ such that $\boldsymbol{e}''\mathbf{H}''^{\top} = \mathbf{0}$ Please look at Equation 6.6 for an explicit expression of this quantum state. As by construction $\boldsymbol{e}_1'' \in E_1$ and $\boldsymbol{e}_1'' \in E_1$ have disjoint set of non-zero coordinates, then $\boldsymbol{e}''$ is of weight $|\boldsymbol{e}''| = |\boldsymbol{e}_1''| + |\boldsymbol{e}_2''| = p/3 + 2/3 = p$. So we end up with a quantum superposition of $|L_1 \bowtie_\ell L_2|$ solutions to the $\mathsf{DP}_{\mathbf{H}'',\mathbf{0},p}$ subproblem. Using this as a subroutine within the ISD framework leads to the following theorem.

Theorem 6.15. *Let us fix parameters ℓ, p, t and set $k := k_U + k_V$. There exists a quantum algorithm under the QRAM model assumption that solves DWK_{n,k_U,k_V} in time*

$$T = \max\left\{ \binom{k+\ell}{p}^{1/2} 2^{p/2}, \frac{3^{n/4-k_U/2}\binom{n}{t}^{1/4}}{2^{t/4-p/2}\binom{n-k-\ell}{t-p}^{1/2}} \right\}.$$

Proof. Sizes of lists L_1 and L_2 are given in Equation 6.11 just above. Constructing the initial classical list takes time $|L_1|$, and constructing the initial quantum state $|\psi_{L_2}\rangle$ can be done in efficient time by a Quantum Fourier Transform and then applying the circuit $|\boldsymbol{e}_2''\rangle|0\rangle \mapsto |\boldsymbol{e}_2''\rangle|\boldsymbol{e}_2''\mathbf{H}''^{\top}\rangle$.

The quantum merging takes time $|L_1| + \sqrt{|L_2|}$ by Lemma 6.13. On average we can expect $|L_1 \bowtie_\ell L_2| = \binom{k+\ell}{p} 2^p \cdot 3^{-\ell} := NbSolFound$ by Equation 6.4. This is also equal, up to a polynomial factor, to $NbSol(\mathsf{DP}_{\mathbf{H}'',\mathbf{0},p})$ the number of solutions to the DP subproblem, By Proposition 6.3. And Proposition 6.6 gives the number of solutions to the DP problem in the $(U, U+V)$-code which is $NbSol(\mathsf{DP}_{\mathbf{H},\mathbf{0},t}) = \binom{n/2}{t/2} \frac{2^{t/2}}{3^{n/2-k_U}}$. Plugging these values into the Theorem 6.11 with the same notations provides the result. □

Numerical results

The time complexity is optimal when the list L_2 is of maximal size, so when p is fixed such that $\binom{k+\ell}{p}^{1/3} 2^{p/3} = 3^\ell$. Parameters ℓ and t are obtained by numerical optimization to minimize the time complexity of the attack. We give here the optimal ISD parameters and the associated time complexities for each set of parameters of Wave given in Table 6.1.

The time complexity is optimal for $t \approx 0.21$, $l \approx 0.0052$ and $p \approx 0.0024$. The ISD algorithm with a quantum Dumer subroutine solves DWK_{n,k_U,k_V} for the set of Wave parameters (I) in time $2^{0.0094n+o(n)}$ *i.e.* 2^{80} bits of quantum security; for the set (III) in time $2^{0.0096n+o(n)}$ *i.e.* 2^{120}; and for the set (V) in time $2^{0.0097n+o(n)}$ *i.e.* 2^{160}. The slight differences in the time exponents come from the fact that the dimensions k_U, k_V are not exactly linear in n. These results are summarized in the second column of Table 6.3.

6.4. Message attacks on DOOM

This attack consists in forging a signature by solving the problem $\mathsf{DOOM}_{n,k,w}$ (Problem 6.8) that we remind here: Given a list S of syndromes in $\mathbb{F}_3^{n-k}$ and a matrix $\mathbf{H}$, find $\boldsymbol{e} \in \mathbb{F}_3^n$ and $\boldsymbol{s} \in S$ such that $\boldsymbol{e}\mathbf{H}^\top = \boldsymbol{s}$. Once again we use the ISD framework, but here we use Wagner algorithm [SS81] as a subroutine instead of just Dumer's [Dum91].

6.4.1. Classical message attack

The best known classical message attack algorithm is the smoothed Wagner algorithm from [Bri+20] based on the approach from [Sen11] to solve DOOM. Choose parameter a the tree depth of Wagner algorithm. Wagner algorithm [SS81] can be seen as a generalisation of Dumer [Dum91], where taking Wagner with $a = 1$ exactly describes Dumer's algorithm.

First lists.

We start by constructing the first-level lists $L_1^{(0)}, \dots, L_{2^a-1}^{(0)}$ of size $|L_i^{(0)}| = 3^{\ell/a}$, where for $i = 1$ to $2^a - 1$ we sample

$$E_i^{(0)} \subseteq \left\{ (\mathbf{0}^{\frac{k+\ell}{2^a}} \| \dots \| \mathbf{0}^{\frac{k+\ell}{2^{a-1}}} \| \underbrace{\boldsymbol{x}}_{i\text{th block}} \| \mathbf{0}^{\frac{k+\ell}{2^{a-1}}} \| \dots \| \mathbf{0}^{\frac{k+\ell}{2^{a-1}}}) \mid \boldsymbol{x} \in \mathbb{F}_3^{\frac{k+\ell}{2^{a-1}}}, |\boldsymbol{x}| = \frac{p}{2^a - 1} \right\}.$$

$$L_i^{(0)} := \left\{ ((\boldsymbol{e}'', \mathbf{0}), \boldsymbol{e}''\mathbf{H}''^\top) : \boldsymbol{e}'' \in E_i^{(0)} \right\} \tag{6.12}$$

And with the DOOM approach, the last list is filled with $3^{\ell/a}$ syndromes restricted on their ℓ last coordinates

$$L_{2^a}^{(0)} \subseteq \left\{ ((\mathbf{0}, \boldsymbol{s}''), -\boldsymbol{s}'') : \boldsymbol{s} = (\boldsymbol{s}' \| \boldsymbol{s}'') \in S, \boldsymbol{s}' \in \mathbb{F}_3^{n-k-\ell}, \boldsymbol{s}'' \in \mathbb{F}_3^\ell \right\}. \tag{6.13}$$

The aim to store elements in the form $((\boldsymbol{e}'', \boldsymbol{s}'), \boldsymbol{e}'\mathbf{H}^\top - \boldsymbol{s}'')$ is to merge them on their last elements and get at the end some for which $\boldsymbol{e}''\mathbf{H}^\top - \boldsymbol{s}'' = \mathbf{0}$ and be able to recover the corresponding $\boldsymbol{e}''$ and $\boldsymbol{s}''$. To be formal, let us precise the tuple addition $(\boldsymbol{e}_1, \boldsymbol{s}_1) + (\boldsymbol{e}_2, \boldsymbol{s}_2) = (\boldsymbol{e}_1 + \boldsymbol{e}_2, \boldsymbol{s}_1 + \boldsymbol{s}_2)$.

Notice that we need the list size to be lower than the number of words of weight p we can generate from $\mathbb{F}_3^{\frac{k+\ell}{2^a-1}}$, *i.e.* we require

$$3^{\ell/a} \leq \binom{(k+\ell)/(2^a-1)}{p/(2^a-1)} 2^{p/(2^a-1)}.$$

Actually, as [Bri+20] has already shown and that we recover in our numerical optimizations, the optimal choice for p in high weight w is to take it at the maximum, *i.e.* $p = k + \ell$. Then by rewriting the condition

on a with this value of p gives this simplified formula:

$$3^{\ell/a} \leq 2^{\frac{k+\ell}{2^a-1}}. \tag{6.14}$$

Merging tree.

For Wagner algorithm, we consider a binary merging tree with at the first level the lists $L_1^{(0)}, \dots, L_{2^a-1}^{(0)}$ and $L_{2^a}^{(0)}$ defined in Equations 6.12 and 6.13. To pass from the j-th level to the $(j+1)$-th we merge pairwise lists using Lemma 6.12, for odd i:

$$L_{(i+1)/2}^{(j+1)} := L_i^{(j)} \bowtie_{(j+1)\ell/a} L_{i+1}^{(j)} \tag{6.15}$$

By construction, every $((\boldsymbol{e}'', \boldsymbol{s}''), \boldsymbol{y}) \in L_{(i+1)/2}^{(j)}$ will satisfy $\boldsymbol{y} = \boldsymbol{e}''\mathbf{H}''^\top - \boldsymbol{s}''$ and $|\boldsymbol{e}''| = j\frac{p}{2^a-1}$.

At each floor, the size of this newly merged list is $|L_{(i+1)/2}^{(j+1)}| = \frac{|L_i^{(j)}| \cdot |L_{i+1}^{(j)}|}{3^{\ell/a}}$, so by recurrence it remains constant at $3^{\ell/a}$ on average. Please refer to Figure 6.2 to visualize the merging process.

At the end of Wagner algorithm, we get a final list $L_1^{(a)}$ filled with tuples in form $(\boldsymbol{e}'', \boldsymbol{s}'', \boldsymbol{e}''\mathbf{H}^\top - \boldsymbol{s}'' = \mathbf{0})$, and by construction, we have $|\boldsymbol{e}''| = p$. At each level in the merging tree, the list sizes are $3^{\ell/a}$ on average, so at the end we find as many solutions $(\boldsymbol{e}'', \boldsymbol{s}'')$ to the $\mathsf{DP}_{\mathbf{H}'', \boldsymbol{s}'', p}$ subproblem.

Algorithm 19 Classical Wagner algorithm for DOOM [Bri+20]

Input: $\mathbf{H}'' \in \mathbb{F}_3^{(k+\ell)\times\ell}$, a list S of target syndromes $\boldsymbol{s}_1'', \dots, \boldsymbol{s}_{3^{\ell/a}}'' \in \mathbb{F}_3^\ell$
length ℓ, target weight p, tree depth a.
Output: List of $(\boldsymbol{e}'', \boldsymbol{s}'') \in \mathbb{F}_3^{k+\ell} \times S$ such that $|\boldsymbol{e}''| = p$ and $\boldsymbol{e}''\mathbf{H}''^\top = \boldsymbol{s}''$

1: Sample lists $E_i^{(0)}$, $L_i^{(0)}$ for $i = 1$ to 2^a using Equation 6.12.
2: **for** $j = 0$ to $a-1$ **do**
3: **for** $i = 1$ to $2^{(a-j)}$ **do**
4: Merge $L_{(i+1)/2}^{(j+1)} = L_i^{(j)} \bowtie_{(j+1)\ell/a} L_{i+1}^{(j)}$.
5: **return** $L_1^{(a)}$

Proposition 6.16. *Let us fix parameters ℓ, p, a such that $3^{\ell/a} \leq 2^{\frac{k+\ell}{2^a-1}}$ (See Equation 6.14). There exists a classical algorithm that solves $\mathsf{DOOM}_{n,k,w}$ in time*

$$T = \max\left\{3^{\ell/a}, \frac{3^{n-k-\ell}}{2^{w-p}\binom{n-k-\ell}{w-p}}\right\}.$$

Proof. By Lemma 6.12, each merging step takes time $3^{\ell/a}$, so Wagner's subroutine 19 takes time $T_{\mathsf{DP}_{\mathbf{H}'', \boldsymbol{s}'', p}} = 3^{\ell/a}$ to find $NbSolFound = 3^{\ell/a}$ solutions. By Proposition 6.3, the solutions to the DP problem are at number $NbSol(\mathsf{DP}_{\mathbf{H}, \boldsymbol{s}, w}) = \binom{n}{w}\frac{2^w}{3^{n-k}}$, and the solutions to the subproblem are at number $NbSol(\mathsf{DP}_{\mathbf{H}'', \boldsymbol{s}'', p}) = \binom{k+\ell}{p}2^p$. We apply the classical ISD algorithm 17 with a Wagner subroutine, and the Theorem 6.10 with these values directly conducts to the result. □

Smoothing

The discreteness of integer parameter a makes the time complexity of Wagner algorithm evolve by stairs, which is not optimal for the majority of its points. [Bri+20] introduced a smoothed Wagner algorithm, whose idea is to start with longer lists and a stricter first merging. The lists $L_i^{(0)}$ are merged pairwise on m bits

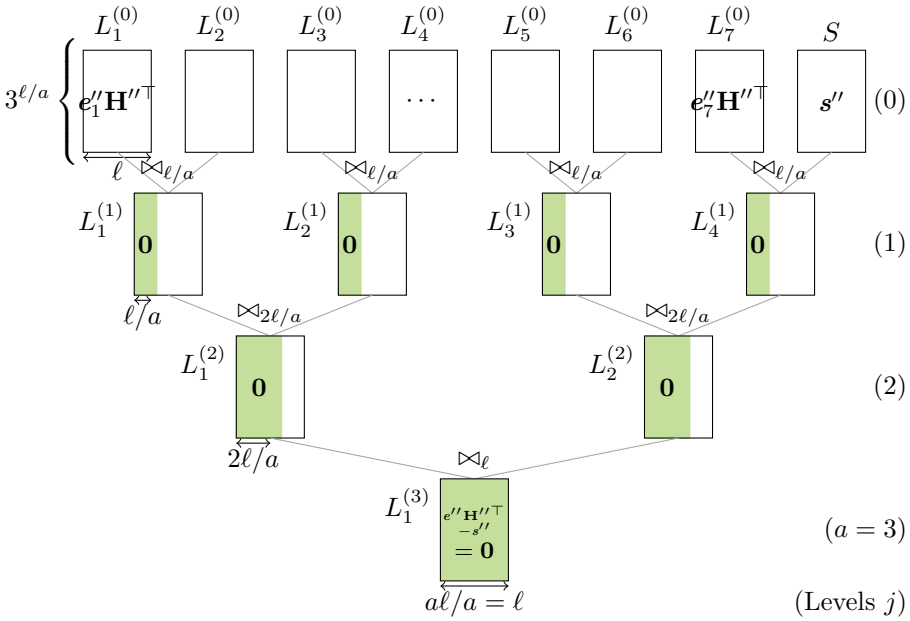


Figure 6.2: Wagner subroutine for $a = 3$. There are $2^a - 1 = 7$ initial lists of $\boldsymbol{e}''\mathbf{H}''^\top$, plus the syndromes list S. At each level, the lists are merged on ℓ/a more coordinates, until the final list $L_1^{(a)}$ filled with elements in the form $((\boldsymbol{e}'', \boldsymbol{s}''), \mathbf{0})$, where pairs $(\boldsymbol{e}'', \boldsymbol{s}'')$ are solutions to the $\mathsf{DOOM}_{k+\ell,k,p}$ subproblem.

such that these merged lists are of size 2^λ for well-chosen m and λ. From there we merge on $\frac{\lambda}{\log_2 3}$ more coordinates at each level, until merging on all the ℓ coordinates.

Algorithm 20 Classical smoothed Wagner algorithm for DOOM [Bri+20]

Input: $\mathbf{H}'' \in \mathbb{F}_3^{(k+\ell)\times\ell}$, target syndromes $\boldsymbol{s}_1'', \ldots, \boldsymbol{s}_{3^{\ell/a}} \in \mathbb{F}_3^\ell$
length ℓ, target weight p, tree depth a.
Output: List of $(\boldsymbol{e}'', \boldsymbol{s}'') \in \mathbb{F}_3^{k+\ell} \times S$ such that $|\boldsymbol{e}''| = p$ and $\boldsymbol{e}''\mathbf{H}''^\top = \boldsymbol{s}''$

Compute λ and m using Equations 6.16 and 6.18.
Sample lists $E_i^{(0)}$, $L_i^{(0)}$ for $i = 1$ to $2^a - 1$, and $L_{2^a}^{(0)}$ using Equation 6.12.
for $i = 1$ to 2^a **do**
    Merge $L_{(i+1)/2}^{(1)} = L_i^{(0)} \bowtie_m L_{i+1}^{(0)}$
for $j = 1$ to $a - 1$ **do**
    for $i = 1$ to $2^{(a-j)}$ **do**
        Merge $L_{(i+1)/2}^{(j+1)} = L_i^{(j)} \bowtie_{m+(j+1)\lambda} L_{i+1}^{(j)}$
return $L_1^{(a)}$

Proposition 6.17. *Let a be the largest integer such that $3^{\ell/a} < 2^{(k+\ell)/(2^a-1)}$. If $a \geq 3$, the classical smoothed Wagner algorithm can find 2^λ solutions to $\mathsf{DP}_{\mathbf{H}'',\boldsymbol{s}'',p}$ in time $\mathcal{O}(2^\lambda)$ with*

$$\lambda = \frac{1}{a-2}\left(\ell\log(3) - 2\cdot\frac{k+\ell}{2^a-1}\right). \tag{6.16}$$

Proof. We restate the proof from [Bri+20] adapted in the context of DOOM (it only changes 2^a to $2^a - 1$ in the formulae).

We are given parameters k and ℓ, and we fix a at the largest integer such that $3^{\ell/a} < 2^{\frac{k+\ell}{2^a-1}}$ to respect the requirement stated in Equation 6.14, and we suppose that $a \geq 3$. At the first level in the tree, we take

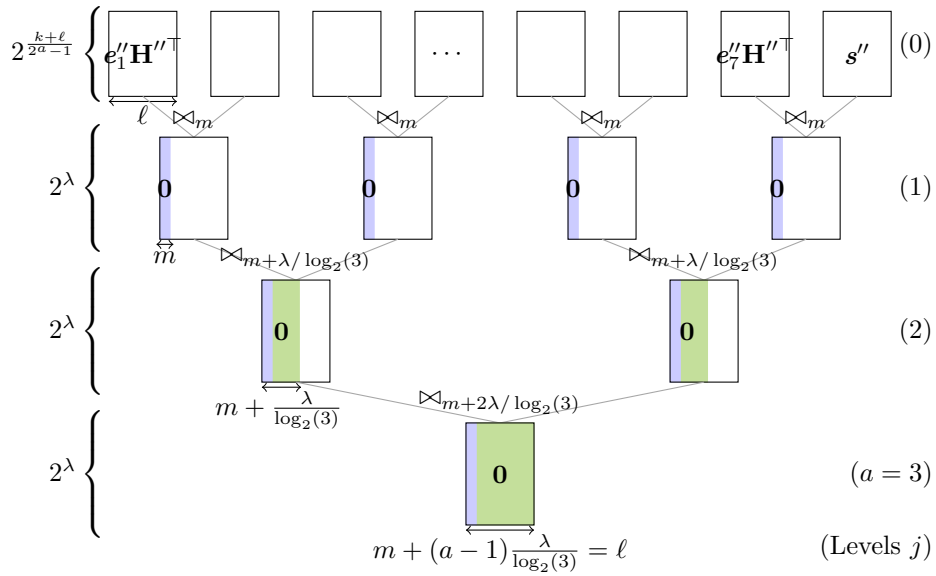


Figure 6.3: Smoothed Wagner algorithm for $a = 3$. The first merging is operated on a small number of coordinates m, and then we merge on $\lambda/\log_2 3$ more coordinates at each level.

lists $L_i^{(0)}$ with the maximum possible size $|L_i^{(0)}| = 2^{\frac{k+\ell}{2^a-1}}$. We firstly merge on $m \leq \ell/a$ coordinates (Steps 2-4 in Algorithm 20). In order to obtain lists of size 2^λ at the second level, we have to choose m such that

$$\frac{\left(2^{\frac{k+\ell}{2^a-1}}\right)^2}{3^m} = 2^\lambda \quad i.e. \quad \lambda = \frac{2(k+\ell)}{2^a-1} - m\log_2 3 \tag{6.17}$$

The $(a-1)$ next merging steps are designed such that merging two lists of size 2^λ gives a new list of size 2^λ, which means that we merge on $\lambda/\log_2 3$ coordinates. In the final list, we have to put a constraint on all coordinates, therefore λ and m have to verify:

$$m + (a-1)\frac{\lambda}{\log_2 3} = \ell. \tag{6.18}$$

By combining Equations 6.17 and 6.18, We get the expression of λ as given in the statement of the proposition, and $m = \frac{1}{a-2}\left(\frac{2(k+\ell)(a-3)}{\log_2 3(2^a-1)} - \ell\right)$. The order a is chosen to be the largest integer such that $3^{\ell/(a-1)} < 2^{\frac{k+\ell}{2^a-1}}$, so λ and m are positive and $2^\lambda \leq 2^{\frac{k+\ell}{2^a-1}}$. □

Theorem 6.18. *There exists a classical algorithm that solves* $\mathsf{DOOM}_{n,k,w}$ *in time*

$$T = \max\left\{\left(\frac{3^\ell}{2^{\frac{k+\ell}{2^a-1}}}\right)^{\frac{1}{a-2}}, \frac{3^{n-k-\ell}}{2^{w-p}\binom{n-k-\ell}{w-p}}\right\}.$$

The left term in the max is improved in comparison with Proposition 6.16 for ISD with non-smoothed Wagner. This corresponds to the case of a single iteration of the ISD algorithm.

Proof. By Proposition 6.16, the classical smoothed Wagner algorithm takes times $2^\lambda = \left(\frac{3^\ell}{2^{\frac{k+\ell}{2^a-1}}}\right)^{\frac{1}{a-2}}$. There are $NbSol(\mathsf{DP}_{\mathbf{H},s,w}) = \binom{n}{w}\frac{2^w}{3^{n-k}}$ solutions for a random code. The time complexity of the ISD classical algorithm 17 with smoothed Wagner subroutine is given by Theorem 6.10 that directly conducts to the result. □

Numerical results.

As we said before, taking $p = k+\ell$ is optimal. Parameters ℓ and a are then chosen by numerical optimization. The optimal a in this setting is here $a = 5$ and $\ell \approx 0.05$. The optimal values of ℓ may slightly vary in function of Wave parameters due to the fact that they are not exactly linear.

Without smoothing. The ISD algorithm with Wagner's subroutine with the set of Wave parameters (I) solves DWK_{n,k_U,k_V} in time $2^{0.0153n+o(n)}$ *i.e.* 2^{130}. For set (III) it solves it in time $2^{0.0156n+o(n)}$ *i.e.* 2^{196}, and for set (V) in time $2^{0.0158n+o(n)}$ *i.e.* 2^{261}.

With smoothing. The ISD algorithm with smoothed Wagner's subroutine for set (I) solves DWK_{n,k_U,k_V} in time $2^{0.0151n+o(n)}$ *i.e.* 2^{129}. For set (III) it solves it in time $2^{0.0155n+o(n)}$ *i.e.* 2^{194}, and for set (V) in time $2^{0.0157n+o(n)}$ *i.e.* 2^{258}.

We see that the smoothing slightly improves the message attack on Wave and grabs a few security bits. The results with the smoothing are summarized in the third column of Table 6.3.

Previous work [FS09] considered the tree depth a as a float instead of an integer, in order to give a complexity approximation of a smoothed Wagner algorithm. If we optimize the time complexity of the non-smoothed Wagner from Proposition 6.16 with a allowed to be a float, the difference is of only one or two bits of security less in comparison with the analysis of the smoothed Wagner algorithm from [Bri+20]. Indeed, for set (I), the number of security bits is 128, for (III) it is 192, and for (V), 256. So considering a float a provides a tight lower bound in this setting.

6.4.2. Quantum message attack

Notations. We recall that given a quantumly accessible list L, $\mathrm{ind}_L(\boldsymbol{x})$ denotes the index of element $\boldsymbol{x}$ in the list L. The quantum superposition of a list L is the quantum state $|\psi_L\rangle = \frac{1}{\sqrt{|L|}} \sum_{\boldsymbol{x}\in L} |\mathrm{ind}_L(\boldsymbol{x})\rangle|\boldsymbol{x}\rangle$ (See Definition 2.6).

For the quantum message attack, we combine DOOM approach from [Sen11], quantum Wagner algorithm of [CDE21] and smoothing from [Bri+20]. The merging tree has the same structure as in the smoothed classical algorithm presented in the previous section. Quantum mergings (see Lemma 6.13) are performed on the right-most side of the tree, and classical mergings (see Lemma 6.12) are performed everywhere else.

Algorithm 21 Quantum Wagner algorithm for DOOM

Input: $\mathbf{H}'' \in \mathbb{F}_3^{(k+\ell)\times\ell}$, list S of target syndromes in $\mathbb{F}_3^\ell$
length ℓ, target weight p, tree depth a.
Output: List in quantum superposition of $(\boldsymbol{e}'', \boldsymbol{s}'') \in \mathbb{F}_3^{k+\ell} \times S$ such that $|\boldsymbol{e}''| = p$ and $\boldsymbol{e}''\mathbf{H}''^\top = \boldsymbol{s}''$

1: Sample lists $L_i^{(0)}$ for $i = 1$ to $2^a - 1$ using Equations 6.12 and 6.13.
2: Construct state $|\psi_{L_{2^a}^{(0)}}\rangle$, quantum superposition of $3^{2\ell/a}$ syndromes $\boldsymbol{s}'' \in \mathbb{F}_3^\ell$
3: **for** $j = 0$ to $a-1$ **do**
4: **for** $i = 1$ to $2^{(a-j)} - 1$ **do**
5: Classically merge $L_{(i+1)/2}^{(j+1)} = L_i^{(j)} \bowtie_{(j+1)\ell/a} L_{i+1}^{(j)}$
6: Quantumly merge $L_{(2^{a-j}+1)/2}^{(j+1)} = L_{2^{a-j}-1}^{(j)} \bowtie_{(j+1)\ell/a} L_{2^{a-j}}^{(j)}$
7: **return** $|\psi_{L_1^{(a)}}\rangle$

Proposition 6.19. *We are given n, k, w. Let fix parameters ℓ, p and a such that $3^{\ell/a} \leq 2^{\frac{k+\ell}{2^a-1}}$. There exists*

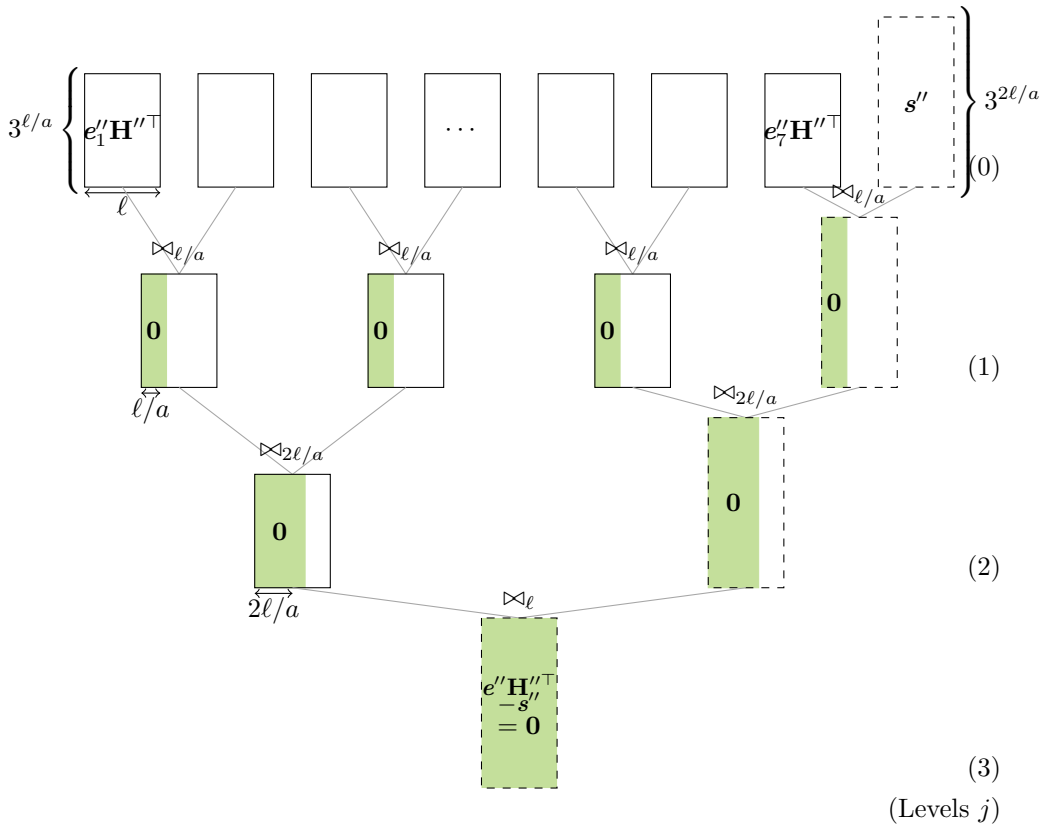


Figure 6.4: Quantum Wagner algorithm (Proposition 6.19 and Algorithm 21). Dashed-line boxes represent lists that are not classically constructed but of which we have an algorithm that constructs a quantum superposition of the elements.

a quantum algorithm that solves $\mathsf{DOOM}_{n,k,w}$ *in time*

$$T = \max\left\{ 3^{\ell/a}, \sqrt{\frac{3^{n-k-\ell}}{2^{w-p}\binom{n-k-\ell}{w-p}}} \right\}.$$

Proof. Quantum Wagner algorithm does the classical merges in time $3^{\ell/a}$, and the quantum ones in time $\frac{3^{\ell/a}\times\sqrt{3^{2\ell/a}}}{3^{\ell/a}} = 3^{\ell/a}$. So the whole Wagner algorithm takes time $T_{\mathsf{DP}_{\mathbf{H}'',s'',p}} = 3^{\ell/a}$ Proposition 6.3 gives the number of solutions to the DP problem $NbSol(\mathsf{DP}_{\mathbf{H},s,w}) = \binom{n}{w}\frac{2^w}{3^{n-k}}$, and to the DP subproblem $NbSol(\mathsf{DP}_{\mathbf{H}'',s'',p}) = \binom{k+\ell}{p}2^p\cdot 3^{-\ell}$. Applying Theorem 6.10 with these amounts gives the time complexity of the ISD algorithm with Wagner algorithm as a subroutine, and this directly leads to the result. □

Algorithm 22 Quantum smoothed Wagner algorithm for DOOM

Input: $\mathbf{H}'' \in \mathbb{F}_3^{(k+\ell)\times\ell}$, list S of target syndromes in $\mathbb{F}_3^\ell$
length ℓ, target weight p, tree depth a.
Output: List in quantum superposition of $(\boldsymbol{e}'', \boldsymbol{s}'') \in \mathbb{F}_3^{k+\ell} \times S$ such that $|\boldsymbol{e}''| = p$ and $\boldsymbol{e}''\mathbf{H}''^\top = \boldsymbol{s}''$

1: Sample lists $L_i^{(0)}$ for $i = 1$ to $2^a - 1$ using Equations 6.12 and 6.13.
2: Compute λ and m using Theorem 6.20.
3: Construct state $|\psi_{L_{2^a}^{(0)}}\rangle$, quantum superposition of $3^{2\ell/a}$ syndromes $\boldsymbol{s}'' \in \mathbb{F}_3^\ell$
4: **for** $i = 1$ to $2^a - 2$ **do**
5: Classically merge $L_{(i+1)/2}^{(1)} = L_i^{(0)} \bowtie_m L_{i+1}^{(0)}$
6: Quantumly merge $L_{2^{a-1}}^{(1)} = L_{2^a-1}^{(0)} \bowtie_m L_{2^a}^{(0)}$
7: **for** $j = 1$ to $a - 1$ **do**
8: **for** $i = 1$ to $2^{(a-j)} - 1$ **do**
9: Classically merge $L_{(i+1)/2}^{(j+1)} = L_i^{(j)} \bowtie_{m+(j+1)\lambda} L_{i+1}^{(j)}$
10: Quantumly merge $L_{(2^{a-j}+1)/2}^{(j+1)} = L_{2^{a-j}-1}^{(j)} \bowtie_{m+(j+1)\lambda} L_{2^{a-j}}^{(j)}$
11: **return** $|\psi_{L_1^{(a)}}\rangle$

Theorem 6.20. *We are given* n, k, w*. Let fix parameters* ℓ*,* p *and* $a \geq 3$ *such that* $3^{\ell/a} \leq 2^{\frac{k+\ell}{2^a-1}}$*. There exists a quantum algorithm that solves* $\mathsf{DOOM}_{n,k,w}$ *in time*

$$T = \max\left\{ \left(\frac{3^\ell}{2^{\frac{k+\ell}{2^a-1}}}\right)^{\frac{1}{a-2}}, \sqrt{\frac{3^{n-k-\ell}}{2^{w-p}\binom{n-k-\ell}{w-p}}} \right\}.$$

Proof. The logic is the same as in the classical smoothed Wagner algorithm, but the optimal list sizes obey a different balance. The order a is chosen at the largest integer such that $3^{\ell/a} < 2^{\frac{k+\ell}{2^a-1}}$ to respect the condition set in the Equation 6.14. The classical lists $L_i^{(0)}$ for $i = 1$ to $2^a - 1$ are chosen of maximal size $2^{\frac{k+\ell}{2^a-1}} =: 2^\gamma$. The list $L_{2^a}^{(0)}$ in quantum superposition is of size $2^{\gamma'}$. The classical list $L_i^{(j)}$ for $j > 0$ and $0 \leq i < 2^a$ are of size 2^λ, and the quantum list for levels ($j > 0$) are of size $2^{2\lambda}$. The classical merging from level (0) to level (1) is done on the m first elements.

Now we need to choose γ, γ', λ and m. The classical merging from level (0) to level (1) puts the constraint $\lambda = 2\gamma - m\log_2 3$. And the quantum merging requires $2\lambda = \gamma' + \gamma - m\log_2 3$. So we can deduce from the above that $\lambda = \gamma' - \gamma$. For the classical part of the merging tree for level (1) and more, the constraints

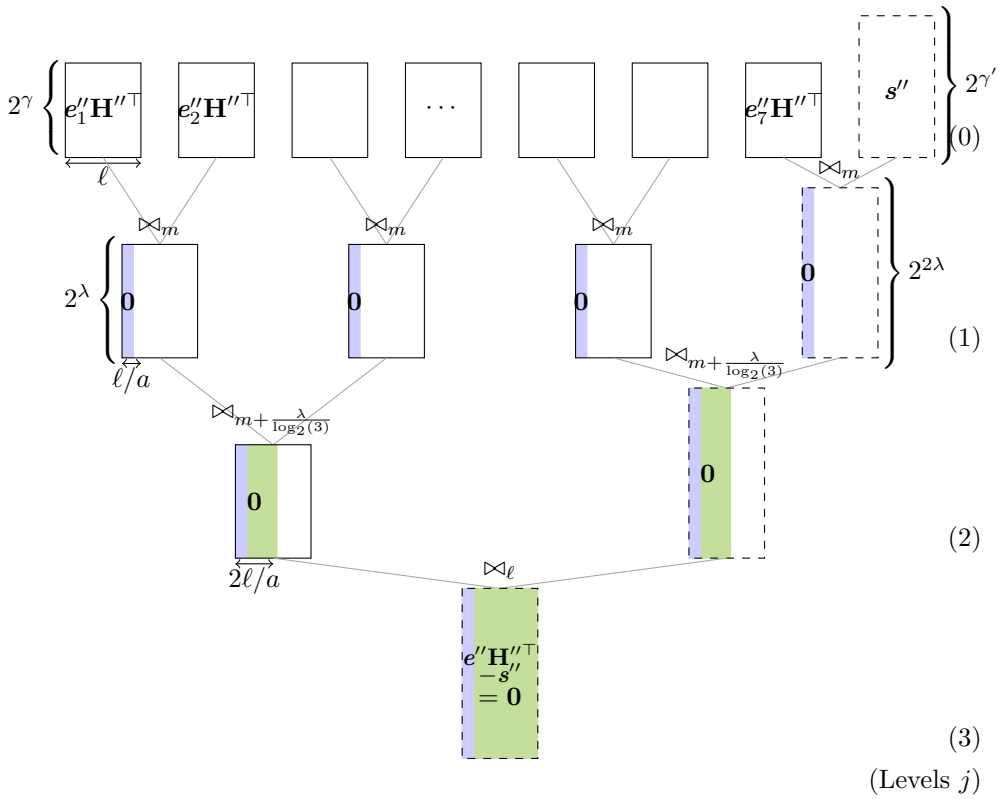


Figure 6.5: Quantum smoothed Wagner algorithm (Proposition 6.20 and Algorithm 22). Dashed-line boxes represent lists that are not classically constructed but of which we have an algorithm that constructs a quantum superposition of the elements.

on λ and m remain the same as in Proposition 6.17 so their expressions are already given (respectively) by Equations 6.16 and 6.18, where $\lambda = \frac{1}{a-2}\left(\ell \log(3) - \frac{2(k+\ell)}{2^a-1}\right)$.

The first-level classical merges take time 2^γ, the first quantum merges take time $2^{\max\{\gamma, \frac{\lambda+\gamma}{2}\}}$, and all the other merges take time 2^λ, which dominates as $\lambda \geq \gamma$. So the quantum smoothed Wagner subroutine takes time $T_{\mathsf{DP}_{\mathbf{H}'',s'',p}} = 2^\lambda$ to construct a list in quantum superposition with 2^{2^λ} solutions to the $\mathsf{DP}_{\mathbf{H}'',s'',p}$ subproblem.

Proposition 6.3 gives the number of solutions to the DP problem $NbSol(\mathsf{DP}_{\mathbf{H},s,w}) = \binom{n}{w}\frac{2^w}{3^{n-k}}$, and to the DP subproblem $NbSol(\mathsf{DP}_{\mathbf{H}'',s'',p}) = \binom{k+\ell}{p} 2^p \cdot 3^{-\ell}$. Theorem 6.11 with these amounts gives the time complexity of the quantum ISD algorithm with smoothed Wagner algorithm as a subroutine, and this directly leads to the result. □

Numerical results.

As said before, taking $p = k + \ell$ is optimal. Parameters ℓ and a are then chosen by numerical optimization.

Without smoothing. Taking $l \approx 0.032$ and $a = 6$ is optimal. The quantum ISD algorithm with quantum Wagner's subroutine with the set of Wave parameters (I) solves DWK_{n,k_U,k_V} in time $2^{0.0093n+o(n)}$ *i.e.* 2^{79}. For set (III) it solves it in time $2^{0.0096n+o(n)}$ *i.e.* 2^{120}, and for set (V) in time $2^{0.0098n+o(n)}$ *i.e.* 2^{161}.

With smoothing. Taking $l \approx 0.034$ and $a = 6$ is optimal. The quantum ISD algorithm with smoothed quantum Wagner's subroutine with the set of Wave parameters (I) solves DWK_{n,k_U,k_V} in time $2^{0.0091n+o(n)}$ *i.e.* 2^{78}. For set (III) it solves it in time $2^{0.0094n+o(n)}$ *i.e.* 2^{117}, and for set (V) in time $2^{0.0095n+o(n)}$ *i.e.* 2^{156}.

These results are summarized in the fourth column of Table 6.3.

Table 6.2: Number of quantum security bits for message attacks.

Algorithm	(I)	(III)	(V)
ISD + Wagner [CDE21]	79	120	161
Our estimation in [Ban+23]	77	117	157
ISD + Smoothed Wagner (Thm. 6.20)	78	117	156

We see that quantum Wagner algorithm benefits from smoothing, by respectively decreasing the security by respectively 1, 3 and 5 security bits for sets (I), (III) and (V). And we correct the claimed quantum security level of [Ban+23], whose estimation did not rely on the analysis of an explicitly described algorithm, which is now formalized by our theorem 6.20. The estimation only differs by plus or minus one security bit, and the slight underestimation in case (V) maintains the security level far from the required threshold at 128 security bits.

6.5. Discussion

Table 6.3 summarizes Wave security against all the attacks studied in this work.

The Wave parameters (recalled in Table 6.1) were chosen such that the time of both the classical attacks, on key and on message, are superior and the closest to the required number of security bits. Sendrier [Sen23] explained the process to deduce the optimal parameters from the tradeoff between these two classical attacks, as each one gives opposite constraints on the parameters.

Table 6.3 reveals a visible gap between the minimal number of required security bits and the ones obtained for key attacks. This is a consequence of the discreteness of the Wave parameters that prevent exactly reaching

Table 6.3: Security levels of Wave instances. λ bits of security indicate that the most efficient known attacks require a time 2^λ to execute.

	Classical		Quantum	
Setting	Key attack Thm. 6.14	Message attack Thm. 6.18	Key attack Thm. 6.15	Message attack Thm. 6.20
(I)	138	129	80	78
(III)	206	194	120	117
(V)	274	258	160	156

the minimum number of security bits for both attacks. They may exist solutions to smooth this tradeoff to gain on the size of the key, and we let it at an open question.

For message attacks, the parameters in [Ban+23] were chosen using estimations, not based on an explicitly described algorithm. Our analysis in Section 6.4 shows that there exists a classical algorithm that gets close time complexity to the lower bound from [FS09] but does not reach it. For quantum message attacks, our algorithm recovered the estimated security level with a slight difference of one security bit.

We also notice that the key attack benefits more from the quantum setting than the message attacks. The reason is that Grover's algorithm has a stronger impact when there is only one search on a large range, as in Dumer's algorithm, instead of on several fragmented small ones as happens in Wagner algorithm. Notice also that quantum security is not a limiting factor. A quadratic gain, which is the best that we can get from simply applying Grover, is not even reached. Moreover, the time analysis of the quantum attacks was done without considering the extra time of QRAM operations. Future practical implementations of these attacks then can be way more demanding in running time.

Therefore the classical attacks are then what determines the security of Wave, and hence the choice of its optimal parameters. All the best known attacks at this day rely on the Decoding Problem, which is a well-studied problem. It is however an open problem to determine if there exists a better key-distinguishing attack that uses the structure of the $(U, U + V)$-code, potentially by avoiding going through the Decoding Problem.

7. Conclusion: Open problems

As each answer raised new questions during this thesis, please find below some open problems and leads for further work.

Lattice sieving

What complexities do we obtain by applying both filtering techniques from [Bec+16] and [CL23] to k-sieves? In Chapter 5, we introduced the k-RPC filtering, a new LSF technique tailored for the k-sieve. A natural extension is to combine k-RPC prefiltering and then search close vectors through usual [Bec+16] pairwise filtering. Such an algorithm will certainly give better time-memory tradeoffs than both algorithms on their own. We can in addition consider other techniques: unbalanced configurations [Kir+19] and sparsification as it was done in [CL21].

What are the optimal merging trees of classical k-sieves? In Chapter 5 we presented classical algorithms for the 3 and 4-sieves, and it happened that for our two algorithms, the best strategies to merge the lists differed. If we want to generalize the k-sieve algorithm for higher k, we have to understand what is the optimal merging strategy depending on k. We have also to find which layout of the filtering layers gives the best complexities.

In practice, how do the new k-sieves perform? It would be interesting to see how practical Gauss-Sieve benefits from k-RPC in practice by implementing algorithms 13 and 14.

How much do k-sieves benefit from quantum walks? As sieving algorithms rely on a collision problem, incorporating quantum walks in sieving algorithms is very promising. Indeed, in Chapter 4 we saw that the time exponent of the SVP-solver was decreased from $2^{0.265n}$ to $2^{0.257n}$ in the quantum model with assumed efficient QRAM. So, the k-sieve might also benefit from quantum walks. Such an algorithm for $k = 3$ for example may give an even better time exponent than the current best attack on SVP.

Do quantum algorithms exist to solve SVP in exponential time without QRAM? All SVP-solving algorithms running in super-exponential time stand in either the classical model or the quantum circuit model with QRAM. Quantum enumeration does not require QRAM, but runs in super-exponential time. No algorithm has been found that gets profit from the quantum circuit model without QRAM, and still runs in exponential time. As the quantum circuit model is the one that will happen with the highest probability and in the closest future, it can be interesting to look at what can be done without QRAM, or at least without assuming efficient QRAM. A lead would be not to access lattice vectors from a database, but to construct a quantum sampling of vectors by adapting Klein's algorithm [Kle00] to be usable by Grover's search [Gro96], and then find collisions through the approach proposed in [JS20].

Can quantum filtering improve sieving algorithms? [Hei21] introduced a way to do quantumly the search of the nearest filters, which was previously done classically including in quantum algorithms. Our attempt to use their technique in our quantum walk algorithm from Chapter 4 was not fruitful. However, it may give something interesting to use it within the k-sieve framework presented in Chapter 5.

Codes

How lattice techniques can be adapted to codes? Codes are another important lead for post-quantum cryptography. They have strong links with lattices due to their close structures, however only a few transfers of techniques have been done between these two research fields. Algorithms for the Decoding Problem can benefit from filtering and sieving techniques, as suggested by the results of [GJN23]. The follow-up would be to explore the code-sieving algorithms in the classical model, in the quantum model with QRACM (using Grover's search [Gro96]) and with QRAQM (using quantum walks [Mag+11]).

Can message attacks on Wave be improved? Considering an unbalanced merging tree may lead to a speed up, as [Sch22] found for the XOR problem. In this case, smoothing will be different and may have a higher impact on the time complexity. Quantum walks [KT17] may also be interesting to solve the decoding problem in this regime.

Can key attacks on Wave be improved? In Chapter 6 we presented attacks on the Wave scheme. Quantum attacks may be improved by the quantum ISD framework [Kir18] and nearest-neighbor techniques [MMT11; Bec+12]. Using ternary $(U, U+V)$ codes in digital signatures is very recent. Our work is only the second one to estimate the quantum security of Wave, with previously [CDE21]. To be confident in this scheme, it really needs to be more looked at to check if it does resist attacks. In particular, does there exist a key-distinguisher that does not go through the Decoding Problem by exploiting the particular structure of the $(U, U+V)$ code?

Bibliography

[Aar+23] Najwa Aaraj, Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Victor Dyseryn, Andre Esser, Philippe Gaborit, Mukul Kulkarni, Victor Mateu, Marco Palumbi, Lucas Perin, and Jean-Pierre Tillich. "PERK". In: NIST PQC (2023). URL: https://pqc-perk.org/assets/downloads/PERK_specifications.pdf (cit. on p. 69).

[Adj+23] Gora Adj, Luis Rivera-Zamarripa1, Javier Verbel, Emanuele Bellini, Stefano Barbero, Andre Esser, Carlo Sanna, and Floyd Zweydinger. "MiRitH (MinRank in the Head)". In: NIST PQC (2023). URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/MiRitH_spec-web.pdf (cit. on p. 69).

[AG11] S. Arora and R. Ge. "New algorithms for learning in presence of errors". In: ICALP (2011). URL: https://users.cs.duke.edu/~rongge/LPSN.pdf (cit. on p. 21).

[Agg+15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. "Solving the shortest vector problem in 2^n time via discrete Gaussian sampling". In: STOC (2015). URL: https://dl.acm.org/doi/10.1145/2746539.2746606 (cit. on pp. 21, 22).

[Agg+21] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. "Improved (provable) algorithms for the shortest vector problem via bounded distance decoding". In: STACS (2021). URL: https://arxiv.org/abs/2002.07955 (cit. on p. 22).

[Agu+23] Carlos Aguilar, Melchor, Thibauld Feneuil, Nicolas Gama, Shay Gueron, James Howe, David Joseph, Antoine Joux, Edoardo Persichetti, Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue. "The Syndrome Decoding in the Head (SD-in-the-Head) Signature Scheme". In: NIST PQC (2023). URL: https://sdith.org/docs/sdith-v1.0.pdf (cit. on p. 69).

[Ajt96a] M. Ajtai. "Generating hard instances of lattice problems". In: STOC (1996). URL: https://dl.acm.org/doi/pdf/10.1145/237814.237838 (cit. on p. 20).

[Ajt96b] M. Ajtai. "Generating Hard Instances of Lattice Problems (Extended Abstract)". In: STOC. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 99–108. ISBN: 0897917855. URL: https://doi.org/10.1145/237814.237838 (cit. on p. 20).

[Ajt99] M. Ajtai. "Generating hard instance of the short basis problem". In: ICALP (1999). URL: https://courses.grainger.illinois.edu/cs598dk/sp2021/Files/ajtai99.pdf (cit. on p. 19).

[Alb+19] Martin Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, and Marc Stevens. "The general sieve kernel and new records in lattice reduction". In: EUROCRYPT (2019). URL: https://eprint.iacr.org/2019/089.pdf (cit. on pp. 19, 21).

[Alb+20] Martin R. Albrecht, Shi Bai, Pierre-Alain Fouque, Paul Kirchner, Damien Stehlé, and Weiqiang Wen. "Faster Enumeration-based Lattice Reduction: Root Hermite Factor $k^{1/(2k)}$ in Time $k^{k/8+o(k)}$". In: (2020). URL: https://eprint.iacr.org/2020/707 (cit. on p. 21).

[All+23] Jonathan Allcock, Jinge Bao, Joao F. Doriguelloand, Alessandro Luongo, and Miklos Santha. "Constant-depth circuits for Uniformly Controlled Gates and Boolean functions with application to quantum memory circuits". In: arXiv (2023). URL: https://arxiv.org/pdf/2308.08539.pdf (cit. on p. 12).

[Amb07] Andris Ambainis. "Quantum Walk Algorithm for Element Distinctness". In: SIAM J. Comput. 37.1 (2007), pp. 210–239. URL: https://doi.org/10.1137/S0097539705447311 (cit. on p. 18).

[And+14] Alexandr Andoni, Piotr Indyk, Huy Lê Nguyên, and Ilya Razenshteyn. "Beyond locality-sensitive hashing". In: SODA (2014), pp. 1018–1028. URL: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973402.76 (cit. on p. 25).

[ANS18] Y. Aono, P.Q. Nguyen, and Y. Shen. "Quantum Lattice Enumeration and Tweaking Discrete Pruning". In: ASIACRYPT (2018). URL: https://inria.hal.science/hal-01870620/document (cit. on p. 21).

[AR15] Alexandr Andoni and Ilya Razenshteyn. "Optimal data-dependent hashing for approximate near neighbors". In: STOC (2015), pp. 793–801. URL: https://dl.acm.org/doi/pdf/10.1145/2746539.2746553 (cit. on p. 25).

[Ara+21] N Aragon, P. L. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Ghosh, S. Gueron, T. Güneysu, C. Aguilar Melchor, R. Misoczki, E. Persichetti, J. Richter-Brockmann, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor. "BIKE - Bit Flipping Key Encapsulation". In: NIST PQC (2021). URL: https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.10.1.pdf (cit. on p. 69).

[Ara+23a] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Antoine Joux, Matthieu Rivain, Jean-Pierre Tillich, and Adrien Vin*c* cotte. "RYDE specifications". In: NIST PQC (2023). URL: https://pqc-ryde.org/assets/downloads/RYDE_Specifications.pdf (cit. on p. 69).

[Ara+23b] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, Matthieu Rivain, and Jean-Pierre Tillich. "MIRA Specifications". In: NIST PQC (2023). URL: https://pqc-mira.org/assets/downloads/mira_spec.pdf (cit. on p. 69).

[Aru+15] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. "On the robustness of bucket brigade quantum RAM". In: New J. of Physics (2015). URL: http://dx.doi.org/10.1088/1367-2630/17/12/123010 (cit. on p. 12).

[Bal+23a] Marco Baldi, Alessandro Barenghi, Luke Beckwith, George Mason University, Jean-Fran*c* cois Biasse, Andre Esser, Kris Gaj, Kamyar Mohajerani, Gerardo Pelosi, Edoardo Persichetti, Markku-Juhani O. Saarinen, Paolo Santini, and Robert Wallace. "LESS: Linear Equivalence Signature Scheme". In: NIST PQC (2023). URL: https://www.less-project.com/LESS-2023-06-01.pdf (cit. on p. 69).

[Bal+23b] Marco Baldi, Alessandro Barenghi, Sebastian Bitzer, Patrick Karl, Felice Manganiello, Alessio Pavoni, Gerardo Pelosi, Paolo Santini, Jonas Schupp, Freeman Slaughter, Antonia Wachter-Zeh, and Violetta Weger. "CROSS: Codes and Restricted Objects Signature Scheme". In: NIST PQC (2023). URL: https://www.cross-crypto.com/CROSS_Specification.pdf (cit. on p. 69).

[Ban+23] Gustavo Banegas, Kévin Carrier, André Chailloux, Alain Couvreur, Thomas Debris-Alazard, Philippe Gaborit, Pierre Karpman, Johanna Loyer, Ruben Niederhagen, Nicolas Sendrier, Benjamin Smith, and Jean-Pierre Tillich. "Wave Support Documentation". In: NIST PQC (2023). URL: https://wave-sign.org/wave_documentation.pdf (cit. on pp. 6, 69, 79, 88, 89).

[Bec+12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. "Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding". In: EUROCRYPT. Lecture Notes in Computer Science. Springer, 2012. URL: https://eprint.iacr.org/2012/026.pdf (cit. on pp. 71, 92).

[Bec+16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. "New directions in nearest neighbor searching with applications to lattice sieving". In: ACM-SIAM Discrete Algorithms (2016). URL: https://doi.org/10.1137/1.9781611974331.ch2 (cit. on pp. 5, 23, 25, 27–29, 33, 45–48, 54, 91).

[Ber+13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. "Quantum Algorithms for the Subset-Sum Problem". In: PQCrypto. Ed. by Philippe Gaborit. Vol. 7932. Lecture Notes in Computer Science. Springer, 2013, pp. 16–33. URL: https://doi.org/10.1007/978-3-642-38616-9%5C_2 (cit. on p. 18).

[Ber+19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. "The SPHINCS+ Signature Framework". In: SIGSAC (2019). URL: https://dl.acm.org/doi/pdf/10.1145/3319535.3363229 (cit. on p. 4).

[Ber+22] Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. "Classic McEliece: conservative code-based cryptography: cryptosystem specification". In: NIST PQC (2022). URL: https://classic.mceliece.org/mceliece-spec-20221023.pdf (cit. on p. 69).

[Ber09] Daniel J. Bernstein. "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?" In: SHARCS (2009). URL: http://repository.tue.nl/663426 (cit. on p. 12).

[BJP06] J. P. Buhler, H. W. Lenstra Jr., and Carl Pomerance. "Factoring integers with the number field sieve". In: LNM (2006). URL: https://link.springer.com/chapter/10.1007/BFb0091539 (cit. on p. 11).

[BKW03] A. Blum, A. Kalai, and H. Wasserman. "Noise-tolerant learning, the parity problem, and the statistical query model". In: J. ACM (2003). URL: https://dl.acm.org/doi/pdf/10.1145/792538.792543 (cit. on p. 21).

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. "Tuple lattice sieving". In: LMS J. Comput. Math. (2016), pp. 146–162. URL: https://www.cambridge.org/core/services/aop-cambridge-core/content/view/C1CE6384DEC54330AEFB2A4D38190094/S1461157016000292a.pdf/tuple_lattice_sieving.pdf (cit. on pp. 29, 32, 47).

[BM17] Leif Both and Alexander May. "Optimizing BJMM with Nearest Neighbors: Full Decoding in $2^{2/21n}$ and McEliece Security". In: WCC. Sept. 2017. URL: http://wcc2017.suai.ru/Proceedings%7B%5C_%7DWCC2017.zip (cit. on p. 71).

[BM18] Leif Both and Alexander May. "Decoding Linear Codes with High Error Rate and Its Impact for LPN Security". In: PQCrypto. Ed. by Tanja Lange and Rainer Steinwandt. Vol. 10786. Lecture Notes in Computer Science. Fort Lauderdale, FL, USA: Springer, Apr. 2018, pp. 25–46. URL: https://doi.org/10.1007/978-3-319-79063-3%7B%5C_%7D2 (cit. on p. 71).

[BMT78] Elwyn R. Berlekamp, Robert J. McEliece, and Henk Van Tilborg. "On the Inherent Intractability of Certain Coding Problems". In: IEEE Transactions on Information Theory (1978). URL: https://ieeexplore.ieee.org/document/1055873 (cit. on p. 69).

[Bon+23] Xavier Bonnetain, André Chailloux, André Schrottenloher, and Yixin Shen. "Finding many Collisions via Reusable Quantum Walks". In: EUROCRYPT (2023). URL: https://arxiv.org/abs/2205.14023 (cit. on pp. 17, 45).

[Bos+18] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J.M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM". In: IEEE (2018). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8406610 (cit. on pp. 4, 21).

[Boy+98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. "Tight bounds on quantum searching". In: Progress of Physics (1998). URL: https://onlinelibrary.wiley.com/doi/pdf/10.1002/(SICI)1521-3978(199806)46:4/5%3C493::AID-PROP493%3E3.0.CO;2-P (cit. on p. 14).

[Bra+02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. "Quantum amplitude amplification and estimation". In: QCQI (2002), 305:53–74. URL: https://arxiv.org/abs/quant-ph/0005055 (cit. on pp. 15, 79).

[Bri+20] Rémi Bricout, André Chailloux, Thomas Debris-Alazard, and Matthieu Lequesne. "Ternary Syndrome Decoding with Large Weight". In: SAC (2020). URL: https://arxiv.org/pdf/1903.07464.pdf (cit. on pp. 6, 69, 71, 80–82, 84).

[Bud+20] Alessandro Budroni, Qian Guo, Thomas Johansson, Erik Mårtensson, and Paul Stankovski Wagner. "Making the BKW Algorithm Practical for LWE". In: Indocrypt (2020). URL: https://bora.uib.no/bora-xmlui/bitstream/handle/11250/2756518/2020-1467.pdf (cit. on p. 21).

[Car+22] Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. "Statistical Decoding 2.0: Reducing Decoding to LPN". In: ASIACRYPT. Lecture Notes in Computer Science. Springer, 2022. URL: https://eprint.iacr.org/2022/1000 (cit. on p. 71).

[CDE21] André Chailloux, Thomas Debris-Alazard, and Simona Etinski. "Classical and Quantum algorithms for generic Syndrome Decoding problems and applications to the Lee metric". In: PQCrypto (2021). URL: https://arxiv.org/pdf/2104.12810.pdf (cit. on pp. 6, 69, 84, 88, 92).

[CGK98] Isaac L. Chuang, Neil Gershenfeld, and Mark Kubinec. "Experimental Implementation of Fast Quantum Searching". In: Physical review letters (1998). URL: https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.80.3408 (cit. on p. 3).

[Cha02] Moses S. Charikar. "Similarity estimation techniques from rounding algorithms". In: STOC (2002), pp. 380–388 (cit. on p. 25).

[Cho+23a] Jinkyu Cho, Jong-Seon No, Yongwoo Lee, Young-Sik Kim, and Zahyun Koo. "Enhanced pqsigRM: Code-Based Digital Signature Scheme with Short Signature and Fast Verification for Post-Quantum Cryptography". In: NIST PQC (2023). URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/Enhanced-pqsigRM-spec-web.pdf (cit. on p. 69).

[Cho+23b] Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Lars Ran, Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. "MEDS: Matrix Equivalence Digital Signature". In: NIST PQC (2023). URL: https://www.meds-pqc.org/spec/MEDS-2023-07-26.pdf (cit. on p. 69).

[CL21] André Chailloux and Johanna Loyer. "Lattice sieving via quantum random walk". In: ASIACRYPT (2021). URL: https://eprint.iacr.org/2021/570.pdf (cit. on pp. 5, 33, 91).

[CL23] André Chailloux and Johanna Loyer. "Classical and quantum 3 and 4-sieves to solve SVP with low memory". In: PQCrypto (2023). URL: https://eprint.iacr.org/2023/200.pdf (cit. on pp. 6, 47, 91).

[Coc73] Clifford Cocks. "A note on 'non-secret encryption'". In: GCHQ declassified files (1973). URL: https://web.archive.org/web/20180928121748/https://www.gchq.gov.uk/sites/default/files/document_files/Cliff%20Cocks%20paper%2019731120.pdf (cit. on p. 3).

[Deb23] Thomas Debris-Alazard. "Code-based Cryptography: Lecture Notes". In: ENS Lyon (2023). URL: https://arxiv.org/pdf/2304.03541.pdf (cit. on p. 71).

[DH76] W. Diffie and M. E. Hellman. "New directions in cryptography". In: IEEE Trans. I.T. (1976). URL: https://dl.acm.org/doi/abs/10.1145/3549993.3550007 (cit. on pp. 3, 11).

[DLW20] Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger. "Sieve, Enumerate, Slice, and Lift: Hybrid Lattice Algorithms for SVP via CVPP". In: (2020). URL: https://eprint.iacr.org/2020/487 (cit. on p. 25).

[DST19] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. "Wave: A New Family of Trapdoor One-Way Preimage Sampleable Functions Based on Codes". In: ASIACRYPT (2019). URL: https://arxiv.org/pdf/1810.07554.pdf (cit. on p. 70).

[Duc+19] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. "CRYSTALS-Dilithium, Algorithm Specifications and Supporting Documentation". In: NIST (2019) (cit. on pp. 4, 21, 25).

[Duc17] Léo Ducas. "Shortest Vector from Lattice Sieving: a Few Dimensions for Free". In: EUROCRYPT (2017). URL: https://eprint.iacr.org/2017/999.pdf (cit. on p. 25).

[Dum91] Ilya Dumer. "On minimum distance decoding of linear codes". In: Workshop Inform. Theory (1991), pp. 50–52 (cit. on pp. 71, 77, 80).

[DW22] Léo Ducas and Wessel van Woerden. "On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography". In: EUROCRYPT (2022). URL: https://eprint.iacr.org/2021/1332.pdf (cit. on p. 20).

[Eti23] Simona Etinski. "Generalized Syndrome Decoding Problem and its Application to Post-Quantum Cryptography". In: PhD Thesis (2023) (cit. on p. 71).

[Fey82] Richard Feynman. "Simulating Physics with Computers". In: International Journal of Theoretical Physics 21.6&7 (1982), pp. 467–488. URL: citeseer.ist.psu.edu/feynman82simulating.html (cit. on p. 3).

[Fou+18] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. "Falcon: Fast-Fourier Lattice-basedCompact Signatures over NTRU". In: NIST (2018). URL: https://www.di.ens.fr/~prest/Publications/falcon.pdf (cit. on pp. 4, 21).

[FS09] Matthieu Finiasz and Nicolas Sendrier. "Security Bounds for the Design of Code-based Cryptosystems". In: ASIACRYPT. Ed. by M. Matsui. Vol. 5912. Lecture Notes in Computer Science. Springer, 2009, pp. 88–105. URL: https://eprint.iacr.org/2009/414.pdf (cit. on pp. 6, 71, 77, 84, 89).

[FS86] A. Fiat and A. Shamir. "How to Prove Yourself: Practical Solution to Identification and Signature Problems". In: CRYPTO (1986). URL: https://link.springer.com/chapter/10.1007/3-540-47721-7_12 (cit. on p. 21).

[Gen09] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices". In: STOC. 2009. URL: https://dl.acm.org/doi/pdf/10.1145/1536414.1536440 (cit. on p. 21).

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. "Public-key cryptosystems from lattice reduction problems". In: CRYPTO (1997). URL: https://dspace.mit.edu/bitstream/handle/1721.1/149835/MIT-LCS-TR-703.pdf?sequence=1&isAllowed=y (cit. on p. 20).

[GJN23] Qian Guo, Thomas Johansson, and Vu Nguyen. "A New Sieving-Style Information-Set Decoding Algorithm". In: ePrint Archive (2023). URL: https://eprint.iacr.org/2023/247 (cit. on p. 92).

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Quantum Random Access Memory". In: Phys. Rev. Lett. 100 (16 2008), p. 160501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.100.160501 (cit. on p. 12).

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. "Trapdoors for hard lattices and new cryptographic constructions". In: STOC. ACM. 2008, pp. 197–206. URL: https://dl.acm.org/doi/pdf/10.1145/1374376.1374407 (cit. on p. 21).

[GR04] Lov K. Grover and Terry Rudolph. "How significant are the known collision and element distinctness quantum algorithms". In: Quantum Information and Computation (2004). URL: http://dl.acm.org/citation.cfm?id=2011617.2011622 (cit. on p. 12).

[Gro96] Lov Grover. "A fast quantum mechanical algorithm for database search". In: STOC (1996), pp. 212–219. URL: https://dl.acm.org/doi/pdf/10.1145/237814.237866 (cit. on pp. 13, 14, 74, 76, 91, 92).

[Ham50] Richard Hamming. "Error detecting and error correcting codes". In: Bell System Technical Journal (1950). URL: https://ieeexplore.ieee.org/document/6772729 (cit. on p. 69).

[Hei21] Max Heiser. "Improved Quantum Hypercone Locality Sensitive Filtering in Lattice Sieving". In: preprint (2021). URL: https://eprint.iacr.org/2021/1295.pdf (cit. on pp. 29, 91).

[HK17] Gottfried Herold and Elena Kirshanova. "Improved algorithms for the approximate k-list problem in Euclidean norm". In: PKC (2017), pp. 16–40. URL: https://eprint.iacr.org/2017/017.pdf (cit. on pp. 6, 31, 32, 47, 66).

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. "Speed-ups and time–memory tradeoffs for tuple lattice sieving". In: PKC (2018), pp. 407–436. URL: https://eprint.iacr.org/2017/1228.pdf (cit. on pp. 29, 32, 47, 66).

[HPS01] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. "NSS: An NTRU lattice-based signature scheme". In: EUROCRYPT (2001). URL: https://www.researchgate.net/profile/Jill-Pipher/publication/2321399_NSS_The_NTRU_Signature_Scheme/links/00b7d51f119190a3f3000000/NSS-The-NTRU-Signature-Scheme.pdf (cit. on p. 21).

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. "NTRU: A ring-based public key cryptosystem". In: ANTS (1998). URL: http://csclub.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring21/michaela/files/Hoffstein97.pdf (cit. on pp. 20, 21).

[IM98] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: To- wards removing the curse of dimensionality". In: STOC (1998), pp. 604–613 (cit. on p. 25).

[Jab01] Abdulrahman Al Jabri. "A statistical decoding algorithm for general linear block codes". In: LNCS (2001). URL: https://link.springer.com/content/pdf/10.1007/3-540-45325-3_1.pdf?pdf=inline%20link (cit. on p. 71).

[JJ02] Thomas Johansson and Fredrik Jönsson. "On the complexity of some cryptographic problems based on the general decoding problem". In: IEEE Transactions on Information Theory (2002). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1035119 (cit. on p. 71).

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: IJIS (2001). URL: https://link.springer.com/content/pdf/10.1007/s102070100002.pdf (cit. on p. 3).

[JR23] Samuel Jaques and Arthur R. Rattew. "QRAM: A Survey and Critique". In: preprint (2023). URL: https://arxiv.org/pdf/2305.10310.pdf (cit. on p. 12).

[JS20] Samuel Jaques and André Schrottenloher. "Low-gate Quantum Golden Collision Finding". In: SAC (2020). URL: https://eprint.iacr.org/2020/424.pdf (cit. on pp. 12, 91).

[Kan83] Ravi Kannan. "Improved algorithms for integer programming and related lattice problems". In: STOC (1983), pp. 99–108. URL: https://dl.acm.org/doi/pdf/10.1145/800061.808749 (cit. on p. 21).

[Kir+19] Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. "Quantum algorithms for the approximate k-list problem and their application to lattice sieving". In: ASIACRYPT (2019). URL: https://eprint.iacr.org/2019/1016.pdf (cit. on pp. 6, 31, 32, 47, 59, 62, 65–67, 91).

[Kir18] Elena Kirshanova. "Improved Quantum Information Set Decoding". In: PQCrypto. Ed. by Tanja Lange and Rainer Steinwandt. Vol. 10786. Lecture Notes in Computer Science. Springer, 2018, pp. 507–527. URL: https://doi.org/10.1007/978-3-319-79063-3%5C_24 (cit. on p. 92).

[KL21] Elena Kirshanova and Thijs Laarhoven. "Lower bounds on lattice sieving and information set decoding". In: CRYPTO (2021). URL: https://eprint.iacr.org/2021/785.pdf (cit. on pp. 5, 6, 29, 67).

[Kle00] Philip Klein. "Finding the closest lattice vector when it's unusually close". In: SODA (2000), pp. 937–941. URL: http://128.148.32.110/research/pubs/pdfs/2000/Klein-2000-FCL.pdf (cit. on pp. 22, 91).

[KS01] M. Ajtai R. Kumar and D. Sivakumar. "A sieve algorithm for the shortest lattice vector problem". In: STOC (2001). URL: https://dl.acm.org/doi/pdf/10.1145/380752.380857 (cit. on p. 21).

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. "Quantum Information Set Decoding Algorithms". In: PQCrypto. 2017. URL: https://arxiv.org/pdf/1703.00263.pdf (cit. on p. 92).

[Laa15] Thijs Laarhoven. "Search problems in cryptography, From fingerprinting to lattice sieving". PhD thesis. Eindhoven University of Technology, 2015. URL: https://thijs.com/docs/phd-final.pdf (cit. on pp. 5, 27–29, 33, 36, 45, 46).

[Lab19] Quintessance Labs. "Breaking RSA Encryption - an Update on the State-of-the-Art". In: (2019). URL: https://www.quintessencelabs.com/blog/breaking-rsa-encryption-update-state-art (cit. on p. 3).

[LLL82] A.K. Lenstra, H.W. Lenstra, and L. Lovász. "Factoring polynomials with rational coefficients". In: Math. Ann. (1982). URL: https://infoscience.epfl.ch/record/164484/files/nscan4.PDF (cit. on p. 21).

[LM18] Thijs Laarhoven and Artur Mariano. "Progressive lattice sieving". In: (2018). URL: https://eprint.iacr.org/2018/079 (cit. on p. 25).

[Loy23] Johanna Loyer. "Quantum security analysis of Wave". In: ePrint (2023). URL: https://eprint.iacr.org/2023/1263 (cit. on pp. 6, 69).

[Lyu09] Vadim Lyubashevsky. "Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures". In: ASIACRYPT (2009). URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3ad556181c59e22dd8138e43102b7daab7b1546f (cit. on p. 21).

[Mag+11] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. "Search via Quantum Walk". In: SIAM J. Comput. 40.1 (2011), pp. 142–164. URL: https://doi.org/10.1137/090745854 (cit. on pp. 5, 16, 17, 38, 92).

[McE78] Robert J. McEliece. "A public-key cryptosystem based on algebraic coding theory". In: DSN (1978). URL: https://ntrs.nasa.gov/api/citations/19780016269/downloads/19780016269.pdf#page=123 (cit. on p. 69).

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. "Decoding random linear codes in $O(2^{0.054n})$". In: ASIACRYPT. 2011. URL: https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/ac11_decoding.pdf (cit. on pp. 71, 92).

[MO15] Alexander May and Ilya Ozerov. "On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes". In: EUROCRYPT. Ed. by E. Oswald and M. Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 203–228. URL: https://www.crypto.ruhr-uni-bochum.de/imperia/md/content/may/paper/codes.pdf (cit. on p. 71).

[MV10] Daniele Micciancio and Panagiotis Voulgaris. "Faster exponential time algorithms for the shortest vector problem". In: SODA (2010), pp. 1468–1480. URL: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973075.119 (cit. on pp. 21, 24).

[NC00] Michael A. Nielsen and Isaac L. Chuang. QCQI. New York, NY, USA: Cambridge University Press, 2000. ISBN: 0-521-63503-9 (cit. on p. 9).

[NIS22] NIST. "PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates". In: NIST (2022). URL: https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4 (cit. on p. 69).

[NIS23] NIST. "Post-Quantum Cryptography: Digital Signature Schemes". In: NIST (2023). URL: https://csrc.nist.gov/projects/pqc-dig-sig/standardization/call-for-proposals (cit. on p. 21).

[NIS91] NIST. "Digital Signature Standard". In: FIPS Publication 186 (1991). URL: http://csrc.nist.gov/fips (cit. on p. 11).

[NR09] Phong Q. Nguyen and Oded Regev. "Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures". In: J. Cryptology (2009). URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2f1bc5ba253cf4973bc414f1ad1967c2016ddfef (cit. on p. 21).

[NV08] P.Q. Nguyen and T. Vidick. "Sieve algorithms for the shortest vector problem are practical". In: J. Math. Crypt. 2 (2008), pp. 181–207. URL: https://doi.org/10.1515/JMC.2008.009 (cit. on pp. 5, 21–24).

[Pra62] Eugene Prange. "The use of information sets in decoding cyclic codes". In: IRE Trans. IT 8.5 (1962), pp. 5–9. URL: http://dx.doi.org/10.1109/TIT.1962.1057777 (cit. on pp. 6, 71).

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: STOC (2005), pp. 84–93. URL: https://dl.acm.org/doi/pdf/10.1145/1568318.1568324 (cit. on p. 20).

[Reg09] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: J. ACM (2009). URL: https://dl.acm.org/doi/pdf/10.1145/1568318.1568324 (cit. on pp. 19, 21).

[RG14] Steven Rich and Barton Gellman. "NSA seeks to build quantum computer that could crack most types of encryption". In: Washington Post (2014). URL: https://www.washingtonpost.com/world/national-security/nsa-seeks-to-build-quantum-computer-that-could-crack-most-types-of-encryption/2014/01/02/8fff297e-7195-11e3-8def-a33011492df2_story.html (cit. on p. 4).

[Rit+23] Stefan Ritterhoff, Sebastian Bitzer, Patrick Karl, Georg Maringer, Thomas Schamberger, Jonas Schupp, Georg Sigl, Antonia Wachter-Zeh, and Violetta Weger. "FuLeeca: Algorithm Specifications and Supporting Documentation". In: NIST PQC (2023). URL: https://www.ce.cit.tum.de/fileadmin/w00cgn/lnt/_my_direct_uploads/FuLeeca.pdf (cit. on p. 69).

[RSA77] Ronald Rivest, Adi Shamir, and Leonard Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: (1977). URL: http://doi.acm.org/10.1145/359340.359342 (cit. on pp. 3, 11).

[Sch+] M. Schneider, N. Gama, P. Baumann, and L. Nobach. "Lattice Challenge". In: (). URL: https://latticechallenge.org/ (cit. on pp. 19, 21, 25).

[Sch22] André Schrottenloher. "Improved Quantum Algorithms for the k-XOR Problem". In: SAC (2022). URL: https://eprint.iacr.org/2021/407.pdf (cit. on p. 92).

[SE94] C.-P. Schnorr and M. Euchner. "Lattice basis reduction: improved practical algorithms and solving subset sum problems". In: Math. Programming (1994). URL: https://link.springer.com/content/pdf/10.1007/BF01581144.pdf (cit. on p. 21).

[Sen11] Nicolas Sendrier. "Decoding one out of many". In: PQCrypto (2011). URL: https://eprint.iacr.org/2011/367.pdf (cit. on pp. 6, 69, 71, 75, 80, 84).

[Sen23] Nicolas Sendrier. "Wave Parameter Selection". In: PQCrypto (2023). URL: https://eprint.iacr.org/2023/588.pdf (cit. on pp. 71, 77, 88).

[Sho94] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: FOCS. 1994, pp. 124–134. URL: https://arxiv.org/pdf/quant-ph/9508027.pdf (cit. on pp. 3, 10, 11).

[Sin99] Simon Singh. The Code Book. Doubleday, 1999, pp. 279–292. URL: https://cryptome.org/ukpk-alt.htm (cit. on p. 3).

[SS81] Richard Schroeppel and Adi Shamir. "A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems". In: SIAM (1981). URL: https://epubs.siam.org/doi/epdf/10.1137/0210033 (cit. on pp. 6, 71, 80).

[Ste88] Jacques Stern. "A method for finding codewords of small weight". In: Coding Theory. 1988. URL: https://link.springer.com/chapter/10.1007/BFb0019850 (cit. on p. 71).

[TT07] Kengo Terasawa and Yuzuru Tanaka. "Spherical LSH for approximate nearest neighbor search on unit hypersphere". In: WADS (2007), pp. 27–38. URL: https://link.springer.com/chapter/10.1007/978-3-540-73951-7_4 (cit. on p. 25).

[Wag02] David Wagner. "A generalized birthday problem". In: CRYPTO (2002). URL: https://www.enseignement.polytechnique.fr/informatique/profs/Francois.Morain/Master1/Crypto/projects/Wagner02.pdf (cit. on p. 71).

[Wol23] Ronald de Wolf. Quantum Computing: Lecture Notes. 2023. URL: https://homepages.cwi.nl/~rdewolf/qcnotes.pdf (cit. on p. 18).