Quantum cryptanalysis on euclidean lattices

Johanna Loyer

123 avenue Albert Thomas
87000 Limoges

2 rue Simone Iff
75012 Paris

Internship report

Quantum cryptanalysis on euclidean lattices

March - July 2020

Johanna LOYER

Master 2 in Mathematics CRYPTIS, University of Limoges

Under the supervision of

André CHAILLOUX

Researcher in the COSMIQ project-team, Inria Paris

Contents

Remerciements

Je voudrais tout d'abord adresser toute ma gratitude à mon tuteur de stage, André Chailloux, pour l'encadrement et la pédagogie dont il a fait preuve, et ce malgré les conditions très particulières dans lesquelles se sont déroulées ce stage. Je le remercie également pour la confiance qu'il m'a accordée en acceptant que je poursuive en thèse avec lui. J'ai hâte de travailler sur ces fameuses pistes cyclables quantiques.

Je tiens à remercier mes collègues de l'équipe-projet COSMIQ pour leur accueil et les échanges passionnants que nous avons eus. Merci aussi de m'avoir fait réaliser grâce à leur engagement qu'il y a aussi des aspects humains et même politiques qui entourent la cryptologie.

Je tiens aussi à remercier les professeurs du Master Cryptis de l'Université de Limoges, qui m'ont fourni les connaissances et les méthodes, ainsi que le fait de pousser à la curiosité, qui ont été nécessaires à la réussite de mes études.

Merci à ma famille d'avoir transformé ce confinement en une période agréable mais productive. Et merci à ma famille de cœur pour leur présence quoi qu'il arrive.

Je remercie tous mes amis de la promo Math Cryptis qui ont fait de ces deux ans les deux meilleures années de ma vie, aussi bien intellectuellement que relationnellement. Cela va me manquer ces rendez-vous jeux de société, ces soirées à la colloc et ces week-ends de révisions/projets sur fond de musiques des années 2000.

Je remercie Jérôme d'avoir supporté que je lui parle d'hypercones et de quantique à longueur de journée. Remarque, il me l'a bien rendu avec son passage à l'échelle des VE et autres calculs d'efficacité de recharge par bornes.

Je souhaite enfin remercier M. Biondi, ancien professeur de SI, qui m'a encouragée à ne pas baisser les bras en prépa et à continuer à croire en mes rêves de travailler dans la recherche. (Certes, avec des mots bien à lui.) C'est chose faite, et c'est en partie grâce à son discours.

Notations

The following notations are used in this paper.

- If not precised, v_i stands for the i-th coordinate of the vector $\boldsymbol{v}$.
- We will several times amalgamate the sets $\mathbb{Z}_{2^n} \approx \{0,1\}^n$.
- We denote $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.
- We will several times amalgamate the notions of vectors in $\mathbb{R}^d$ and points in $\mathbb{R}^d$.
- We use $\widetilde{O}(n) := O(\text{polylog}(n)) = O(\log(n)^k)$ for some k, with O the Landau notation.
- For a vector $\boldsymbol{v}$, we call the associated normalized vector: $\boldsymbol{v}/\|\boldsymbol{v}\|$.

1. Introduction

With the possible emergence of quantum computers in the near future, developing cryptography that is resistant to it is one of the challenges that will have to be addressed. In 2015 the National Institute of Standards and Technology (NIST) launched an international competition in the purpose of standardizing quantum resistant protocols of encryption, signature and key exchange. The security of the candidate cryptosystems are based on problems on error decoding codes, multivariate, hashing functions, and lattices. Several finalists [DKL$^+$19, CDH$^+$19, FHK$^+$19] announced on the 23th July 2020 are lattice-based protocols. Therefore it is necessary to deeply study their problems to verify if their difficulty is sufficient against both classical and quantum computers.

From March to July 2020, I did my final internship under the supervision of André Chailloux, researcher in quantum cryptography. It took place at the Inria, a public research institute in computer science and applied mathematics. The project-team I worked with was COSMIQ (ex-SECRET), the acronym standing for COde-based, SyMmetric cryptology, and Quantum Information. Their research work concerns the design and analysis of cryptographic algorithms, classically and quantumly. It mixes fundamental and practical aspects of information protection: cryptanalysis, design of algorithms and implementation.

My internship mostly consisted in acquiring a solid basis of knowledge in quantum computing. The purpose was to start my doctorate with an already constituted bibliography in solving-SVP sieving algorithms, and be able to apply to them quantum speed-ups. Towards the end of my internship, I started to follow a lead to improve the complexity of an algorithm solving the shortest-vector problem on lattices. It has not yet succeeded, so I will pursue this idea during my Ph.D.

This report will present the knowledge I have acquired during these five months. The first part is about Quantum Computing. I will expose the fundamental principles, the quantum model, an attack on actual main cryptosystems (RSA, ECC, Diffie-Hellman), and finally present the most important algorithm for us : Grover's algorithm. In the second part, Sieve Attack Algorithms on Lattices, I will present some cryptographic problems, and algorithms which solve one of them in particular (SVP) by the sieve method, and then present several improvement of its complexities.

2. Quantum Computing

Classical computing consists in storing and processing information coded in form of states of electronic components. In the same way, we speak about quantum computing when data are coded by states of quantum particles. In this case, classical physics no longer stands and the quantum physics theory takes over. That allows us to exploit the particular properties of these particles to do things that were proven impossible for a classical algorithm.

2.1 Fundamental principles

We are going to begin with a comparison with classical physics to understand the important differences between the two physics models. In classical computing, a bit have the value 0 or 1, but only one of these two states at the same time. It can change from one to another, but its state is always unique at an instant t. For quantum particles, the elementary piece of information is called **qubit** and its state can be a **superposition** of 0 and 1 at the same time. When we perform a **measurement** of the particle, there are probabilities of finding the states 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 have measured.

However, probabilities alone are not sufficient to completely explain some physical behaviors of a quantum particle, for example the interference effect. Thus, we use **amplitudes** to characterize its state. Physically this notion is related to the wave function of the particle, and it is translated by a probability when we measure. For a **qubit** at a state denoted $|\psi\rangle$, we can fully describe this state by its amplitudes $\alpha, \beta \in \mathbb{C}$:

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

Here, the value $|\alpha|^2$ (resp. $|\beta|^2$) is the probability of finding 0 (resp. 1) when we perform a measurement. For this, we must have $|\alpha|^2 + |\beta|^2 = 1$.

If we manipulate n qubits, we can rename for simplicity the states $|0\rangle, |1\rangle, ...|2^n - 1\rangle$ instead of $|0...00\rangle$, $|0...01\rangle$, ..., $|1...11\rangle$, these last ones are the binary writing of the first. A register of n qubits can be in any superposition in this form:

$$\alpha_0|0\rangle + \alpha_1|1\rangle + \cdots + \alpha_{2^n-1}|2^n - 1\rangle \qquad \text{with} \sum_{j=0}^{2^n-1} |\alpha_j|^2 = 1.$$

Another important property of quantum particle is the **entanglement**. When two particles are said entangled, it means that if we measure one of them, the state of the other acts as if we also measured it. That implies that we can act on one particle and cause immediate consequence on the other.

For example, let's consider the 2-qubit register $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. If we perform a measurement of any of the two qubits, we obtain with the same probability 0 or 1. But if we then measure the second, one will necessarily get the same result as the first one.

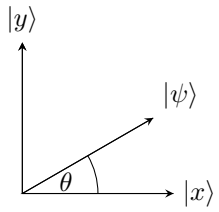
With that basic knowledge about quantum physics, it is sufficient to understand and create quantum algorithms. It is not uncommon for researchers to work on quantum computing without knowing much more about all the subtleties of quantum mechanics.

2.2 Quantum circuit model

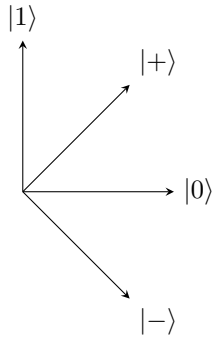
A quantum circuit generalizes the idea of classical circuits. The AND, OR and NOT gates are replaced by elementary quantum gates. A model well defined allows us to know which tools we can use to then construct more complex algorithms.

2.2.1 Measurement

First we are going to formalize the measurement operation for amplitudes in $\mathbb{R}$. For a qubit at state $|\psi\rangle$, we can measure it by taking an orthogonal unitary basis $(|x\rangle, |y\rangle)$. Denote θ the angle between $|x\rangle$ and $|\psi\rangle$ seen as vectors. The probability of having as result $|x\rangle$ by performing a measurement of $|\psi\rangle$ is $\cos^2(\theta)$, and for a result $|y\rangle$ is $\sin^2(\theta)$.



(a) Any basis $(|x\rangle, |y\rangle)$.



(b) The Hadamard basis: $(|+\rangle, |-\rangle)$.

In particular, we set the Hadamard basis (figure above) with $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, which will have several applications further.

When we do not precise an other basis for performing a measurement, it will be by default $(|0\rangle, |1\rangle)$.

2.2.2 Quantum gates

As qubits have a vector form, we can apply to them functions represented in form of unitary matrices. Note that for one qubit (a vector of 2 complex elements), the dimension of a matrix is 2×2.

Definition 2.1 (Pauli matrices)**.**

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

$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ Bitflip: exchanges the amplitudes of $|0\rangle$ and $|1\rangle$ (equivalent to NOT gate)

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

$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ Phaseflip: changes the sign before $|1\rangle$

From these 2×2 single-qubit matrices, we want to construct matrices which act on several qubits. To do this operation, we first need to define the Kronecker product, i.e. the tensor product of matrices.

Definition 2.2 (Kronecker product)**.** If A an $m \times n$ matrix and B a $p \times q$ matrix, then the Kronecker product $A \otimes B$ is the $pm \times qn$ matrix:

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

For example: for $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ and $B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$,

$$A \otimes B = \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix}$$

For U a matrix and $n \in \mathbb{N}$, we will denote $U^{\otimes n} = \underbrace{U \otimes \cdots \otimes U}_{\text{n}}$.

The n-qubits Pauli matrices are obtained by the tensor product of n of the 2×2 Pauli matrices. Since we have 4 possibilities for each of the n tensor factors, we count 4^n n-qubits Pauli matrices of dimension $2^n \times 2^n$.

Theorem 2.3. *[dW19] Every complex 2×2 matrix A can be written as a linear combination of Pauli matrices:*

$$A = \lambda_0 I + \lambda_1 X + \lambda_2 Y + \lambda_3 Z,$$

where $\lambda_i \in \mathbb{C}$. Every $2^n \times 2^n$ matrix A can be written uniquely as a linear combination of the 4^n n-qubits Pauli matrices of dimension $2^n \times 2^n$.

We now define other matrices convenient to use.

$$R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \qquad \text{Phase gate: } R_\theta \text{ rotates the phase of } |1\rangle \text{ by an angle } \theta.$$

This conducts to: $R_\theta|0\rangle = |0\rangle$, $R_\theta|1\rangle = e^{i\theta}|1\rangle$.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad \text{Hadamard transform}$$

If we apply H to a state $|0\rangle$ and then measure, we have equal probability of observing $|0\rangle$ or $|1\rangle$. Idem for $|1\rangle$. However, if we apply H to $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, we obtain $|0\rangle$. Mathematically, it can be easily be verified by calculus ($H = H^{-1}$), and physically it is due to the interference effect.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \text{Controlled NOT}$$

If the first qubit (control) values 1, the second (target) is opposed. The sub-matrix $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ (NOT operation) can be generalised by any 2×2 unitary matrix.

Theorem 2.4. *[dW19] With H, $CNOT$ and $T = R_{\pi/4}$, we can construct an approximation of any possible quantum circuit.*

Example We can construct a circuit to turn two input qubits into an entangled state with only two gates: H and $CNOT$.

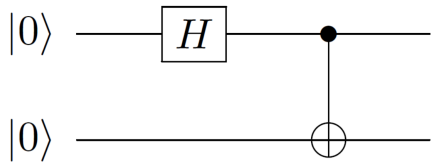


Figure 2.2: Entanglement circuit. (Illustration from [dW19])

Indeed, after the Hadamard gate, the first qubit is in the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, which has as many chances of being measured at $|0\rangle$ or $|1\rangle$. Then we apply the $CNOT$ gate on the two qubits. If the first is measured in the state $|0\rangle$, the $CNOT$ gate has no action, so the two qubits are in the state $|0\rangle$. If the first is measured in the state $|1\rangle$, the $CNOT$ gate flips the second bit which becomes $|1\rangle$. Thus in both cases we get the same strictly random state for the two qubits.

2.2.3 Queries to oracles

Definition 2.5. Let $|x\rangle = |x_0, ..., x_{n-1}\rangle$ be a register of n qubits. For $i \in [0, n-1]$ and $b \in \{0, 1\}$, we call a query oracle:

$$O_x : |i, b\rangle \mapsto |i, b \oplus x_i\rangle$$

In simpler terms, we dispose in one hand the register $|x\rangle$ containing data. In the other we have two qubits. We set the first at the value i for which we are interested to know x_i, and the other bit b we will write onto. After applying O_x seen as a black-box, the second qubit is now at the state x_i.

In particular, for $|b\rangle = |-\rangle$ from the Hadamard basis, we have $O_x : |i, -\rangle \mapsto (-1)^{x_i}|i, -\rangle$.

For a function f that we suppose we know how to compute, we can generalize to any oracle:

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

Note that O_f entangles the two qubits. Indeed, if we measured the second and obtain the state $|f(a)\rangle$ and by measuring the first then we get $|a\rangle$, and reciprocally.

2.2.4 Quantum Fourier Transform

Definition 2.6 (Quantum Fourier Transform - QFT)**.** Let be $N = 2^n$ and $|k\rangle$ a register of n qubits. We define the quantum Fourier transform:

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

In its matrix form, denoting $\omega := e^{2i\pi/N}$:

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

Note that $F_2 = H$. And we also note that $F_N|0^n\rangle = H^{\otimes n}|0^n\rangle$.

Theorem 2.7. *[Sho94] The QFT is feasible in $O(n \cdot \log(n))$ gates H and controlled-$R_{2\pi/2^m}$ for $1 \leqslant m \leqslant n$.*

Elementary bricks to construct algorithms

According to the quantum circuit model, we have these tools at our disposal:

- any normalized linear combination or tensor product of I, X, Y, Z and $H, R_\theta, CNOT$,
- query oracles for obtaining information about a qubit in a register,
- the Quantum Fourier Transform to put the register in a superposed state,
- measurements.

For computing the complexity of an algorithm, depending on the case we can count the number of gates we need or focus on the number of queries to an oracle.

2.3 Attack on RSA, ECC and Diffie-Hellman

2.3.1 Shor's Algorithm

This algorithm is not directly in the domain of research I will work in, but it is a historically important one. Published in 1994, Shor's algorithm [Sho94] can factor a number N in a polynomial time: $O(\log(N)^2)$, and therefore breaks the RSA cryptosystem. For comparison, the actual best classical factoring algorithm is about $O(2^{\sqrt{\log(N)}})$.

Definition 2.8 (Period finding problem)**.** Let be a function $f : \mathbb{N} \to \{0, ..., N-1\}$ with $r \in \{0, ..., N-1\}$ such that $f(a) = f(b) \mod r \Leftrightarrow a = b \mod r$. Find r the period of f.

Theorem 2.9. *The factorisation problem* FACT *is reduced to the period finding problem.*

Proof. Let be N an non prime odd integer and $x \in \{2, ..., N-1\}$ chosen randomly. If $\gcd(x, N) > 1$, the proof is done: we have found a non trivial factor of N. Else, x and N are co-primes, that we will suppose from now.

Let be the sequence (in $\mathbb{Z}_N^*$) : 1, x, x^2, ... This sequence admit a period $r \in [|0, N-1|]$, such that $x^r = 1$. r can also be called the order of the element x.

With a probability greater than $1/2$, r is even and neither $(x^{r/2} + 1)$ nor $(x^{r/2} - 1)$ are multiples of N. Else, we change of x until it is the case. We have:

$$\begin{aligned} & x^r && \equiv 1 \mod N \\ \Rightarrow \quad & x^{r/2} - 1 && \equiv 0 \mod N \\ \Rightarrow \quad & (x^{r/2} + 1)(x^{r/2} - 1) && \equiv 0 \mod N \\ \Rightarrow \quad & (x^{r/2} + 1)(x^{r/2} - 1) && = kN \text{ for } k > 0 \end{aligned}$$

In short, $(x^{r/2} + 1)$ and $(x^{r/2} - 1)$ have good chances to have a non-trivial common factor with N. If it does not work, we change x again until we find. Thus, if we know how to find a period in an efficient way, we also know how to factorize in an efficient way. □

Main idea. The algorithm put a first register in an uniform state over all its qubits by applying a QFT. Then it queries to an oracle to get in the second register the images by the function for which we are looking for its period. Each image is entangled with its antecedent. By applying the QFT on the first register again and then perform a measurement of the first register, we get a value that allow to recover the period.

Let $q = 2^\ell$. Again, for simplicity we will confuse the sets $\{0, ..., q-1\}$ and $\{0,1\}^\ell$, the elements of the second set are seen as the binary writing of an element of the first. We dispose of two gates to construct the associated quantum circuit:

- the Quantum Fourier transform $F_q : |a\rangle \to \frac{1}{\sqrt{q}} \sum_{b=0}^{q-1} (e^{2i\pi/q})^{ab} |b\rangle$
- and a black-box $O_f : |a\rangle|0^n\rangle \to |a\rangle|f(a)\rangle$.

Algorithm 1 Shor

Require: two registers: $|0^\ell\rangle|0^n\rangle$.
Ensure: the period r of f

1. Apply F_q to the first register. ▷ We get an uniform superposition : $\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|0^n\rangle$
2. Apply O_f on all the qubits. ▷ We get the state : $\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|f(a)\rangle$. Note that all the qubits of the first register are now entangled with the second (i.e. measuring the value of ones can force the state of the others).
3. Measure the qubits of the second register. ▷ The value of $f(s)$ is obtained for a s to be determined. Let m be the number of a's such that $f(a) = f(s)$, that is to say $a = s \mod r$. We can write $a = jr+s$ with $0 \leqslant j \leqslant m$. After this measurement, so we get the state: $\frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} |jr + s\rangle|f(s)\rangle$.
   We can ignore the second register from there, because it is frozen in the state $|f(s)\rangle$.
4. Apply again F_q to the first register. ▷ We get (it is sufficient to write the calculations):

$$\frac{1}{\sqrt{qm}} \sum_{b=0}^{q-1} e^{2i\pi \frac{sb}{q}} \Big(\sum_{j=0}^{m-1} e^{2i\pi \frac{jrb}{q}} \Big) |b\rangle$$

5. Measure the first register giving $|b\rangle$.
6. Recover r from b ▷ See the reasoning below.

We denote the result of the measure: $|b\rangle$, from which we will find r. What are the $|b\rangle$ that have the highest probability of being measured? There are two cases to consider:

Simple case. $r \mid q$

Thus $0 \leqslant r < q$ and $m = q/r$. We have $e^{2i\pi rb/q} = 1$ iff $rb/q \in \mathbb{N}$ iff b is a multiple of q/r.

The amplitude (i.e. the coefficient before a state in the sum above) of a such b so is:

$$\frac{1}{\sqrt{qm}} e^{2i\pi \frac{sb}{q}} \Big(\sum_{j=0}^{m-1} e^{2i\pi \frac{jrb}{q}} \Big) = \frac{1}{\sqrt{qm}} 1 \cdot \Big(\sum_{j=0}^{m-1} 1 \Big) = \frac{m}{\sqrt{qm}} = \sqrt{\frac{m}{q}}$$

By squaring its module, we get the probability that the result of the measurement is the state $|b\rangle$, which is $m/q = 1/r$. As we know that the register is in an uniform state (each $|b\rangle$ has the same probability to be measured), we deduce that the number of such b is r. By measuring the first register, so we find randomly one of these b multiples of q/r. We denote $b = cq/r$ with $0 \leqslant c < r$. So we have:

$$b/q = c/r$$

We know b (measured) and q (in the problem statement). If c and r are co-primes, we find them easily by simplifying the fraction b/q to make it irreducible, and the proof is over.

If they have a common factor, we restart the algorithm until finding c co-prime with r. According to a theorem ([HW79] theorem 328), there are $O\big(\frac{r}{log(log(r))}\big)$ numbers smaller than r which are co-primes with r, hence the probability of finding such a c which is in $O\big(\frac{1}{log(log(r))}\big)$. And thus we start again the algorithm $O(log(log(N)))$ times before finding the wanted result.

Difficult case. $r \nmid q$

It is the case that happens most often because $q = 2^\ell$.

We will not detail here. In summary, if q is chosen sufficiently high, the gap between b/q and c/r is low ($\leqslant 1/2q$). With calculations of continuous fractions, we can find c and r.

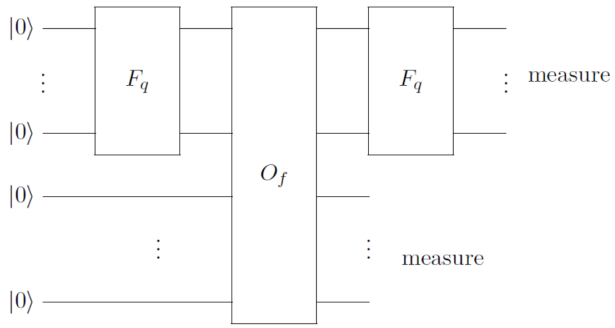


Figure 2.3: Shor's algorithm circuit. (Illustration from [dW19])

2.3.2 Hidden Subgroup Problem

As we will see soon, the Hidden Subgroup Problem (HSP) is a generalization of the period-finding problem. Even if classical HSP-solving algorithms run in exponential complexities, quantum ones are very efficient.

Definition 2.10 (Coset)**.** Let G be a group and $H \leqslant G$. The cosets are the gH for a $g \in G$.

Definition 2.11 (Hidden Subgroup Problem - HSP)**.** Let be G a group, a finite set S, and a function $f : G \to S$. Find $H \leqslant G$ such that:

1. f constant in each coset gH, and
2. f distinct on two different cosets : $f(g) = f(g') \Leftrightarrow gH = g'H$.

Remark 2.12. The period finding and discrete logarithm problems are HSP instances.

Indeed, for the period finding problem, we consider $G = \mathbb{Z}_{\phi(N)}$ (with $\phi(N)$ the Euler's phi function) and $H = \langle r \rangle$. For f a function of period r, f is constant on $r + H$ and distinct on different cosets. Thus finding $H = \langle r \rangle$ leads to know r.

For the discrete logarithm, given γ a generator of a set S ($N = |S|$) and $A \in S$, the problem is to find the unique $a \in \mathbb{N}$ such that $\gamma^a = A$. We consider $G = \mathbb{Z}_N \times \mathbb{Z}_N$ and $H = \langle (a, 1) \rangle$. We have $f : G \to S$ with $f(x, y)\gamma^x A^{-y}$ that we can show it is constant on H and distinct on different cosets. Finding $H = \langle (a, 1) \rangle$ leads to know a.

Therefore, constructing an algorithm which solves efficiently HSP breaks RSA, ECC and Diffie-Hellman.

Algorithm 2 Solving HSP if G is abelian

Require: Two registers $|O^{|G|}\rangle|0^{|S|}\rangle$
Ensure: the hidden subgroup g
Apply the QFT ▷ we get an uniform superposition $|g\rangle|0\rangle$
Apply O_f ▷ we get the state $|g\rangle|f(g)\rangle$
Measure the second register ▷ we obtain $f(s)$
Apply the QFT on the first register
Measure the first register ▷ we obtain g

Complexity. This algorithm runs in time $O(\sqrt{N})$ with N the number of the subgroups of G.

2.4 Grover's Algorithm

Definition 2.13 (Grover's search problem)**.** For $N = 2^n$, we are given an arbitrary $x = (x_1, ..., x_N) \in \{0,1\}^N$. The goal is to find an i such that $x_i = 1$, and to output "no solution" if there are no such i. We assume that we know the number t of such i, or at least an approximation of t.

This problem has first been presented as a search in a un-ordered database. In the original article [Gro96], Grover fixed $t = 1$, which is generalized here.

Classically, it is not possible to get a better complexity than $\Theta(N)$ queries, by examining each x_i one by one until we find one which values 1. Grover's algorithm solves the search problem in $O(\sqrt{N})$ queries, and with $O(\sqrt{N}\log(N))$ other gates.

Main idea. The algorithm separates "good" ($x_i = 1$) and "bad" ($x_i = 0$) indices i and increases step by step the amplitudes of the "good" state. The action of the algorithm can be understood thanks to geometric arguments which we will present.

First, we set:

$$\theta := \text{Arcsin}\Big(\sqrt{\frac{t}{N}}\Big) = \text{Arccos}(\sqrt{\frac{N-t}{N}}\Big)$$

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

$$|U\rangle := \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle = \sqrt{\frac{N-t}{t}}|B\rangle + \sqrt{\frac{t}{N}}|G\rangle$$

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

Geometrically, we see that θ represents the angle between the states $|B\rangle$ and $|U\rangle$. Indeed, $\frac{t}{N}$ is the probability of finding a good solution at the beginning of the algorithm, i.e. getting $|G\rangle$ by a measurement. So $\sqrt{t/N}$ is the amplitude of $|G\rangle$ at the beginning. After a measure, we have the relation $\Pr(|G\rangle) = \sin(\theta)$, so the associate angle has indeed the expression $\theta = \text{Arcsin}(\sqrt{t/N})$.

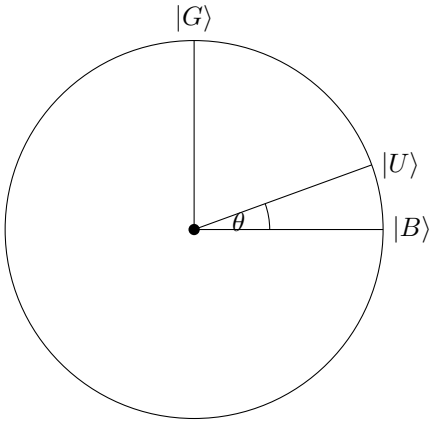


Figure 2.4: Geometrical representation of the state of the N-qubit register.

So, by a basic rule of trigonometry, we can write the following relation:

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

To run the algorithm, we have these gates at our disposal:

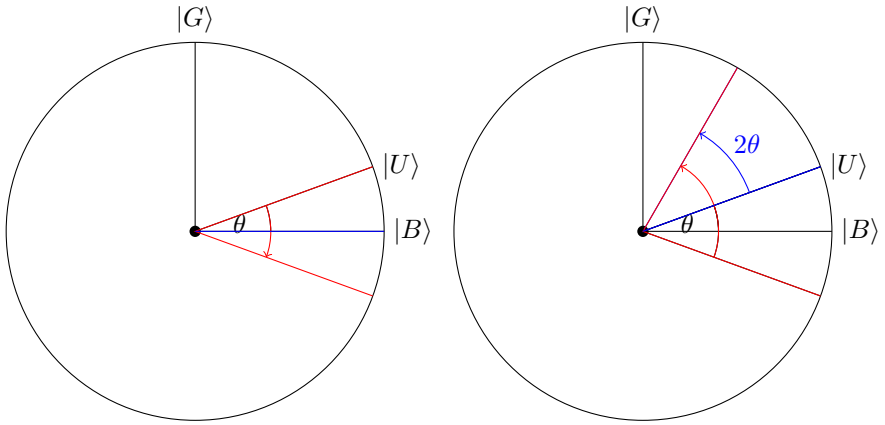
- H the Hadamard gate, with $H^{\otimes n}$ that acts like F_N over $|0^n\rangle$,
- the query oracle $O_{x,\pm}|i\rangle = (-1)^{x_i}|i\rangle$,
- $R|i\rangle = \begin{cases} -|i\rangle & \text{if } |i\rangle \neq |0^n\rangle \\ |0^n\rangle & \text{else.} \end{cases}$, which is realizable in $O(n)$ elementary gates.

Proposition 2.14. *$O_{x,\pm}$ acts as a reflection through $|B\rangle$, and $H^{\otimes n}RH^{\otimes n}$ acts as a reflection through $|U\rangle$.*

Proof. $O_{x,\pm}|i\rangle = (-1)^{x_i}|i\rangle$ 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 it is by definition 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 $\mathcal{G} = H^{\otimes n}RH^{\otimes n}O_{x,\pm}$. The algorithm starts with $|U\rangle$ and at each iteration of the loop, we move forward of the angle 2θ. We note that each application of $\mathcal{G}$ performs only one query to the oracle $O_{x,\pm}$.



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

Algorithm 3 Grover

Require: a register $|0^N\rangle$, the number of solutions t (or its approximation)
Ensure: an index i such that $x_i = 1$, or "no solution"

Apply $H^{\otimes n}$ on the entire register. ▷ We get $H^{\otimes n}|0^n\rangle = |U\rangle$
for $k \simeq \frac{\pi}{4\theta} - \frac{1}{2}$ iterations **do**
  Apply $O_{x,\pm}$ ▷ Reflection through $|B\rangle$
  Apply $H^{\otimes n}RH^{\otimes n}$ ▷ Reflection through $|U\rangle$
Measure the first register and verify if the resulting i is a solution.

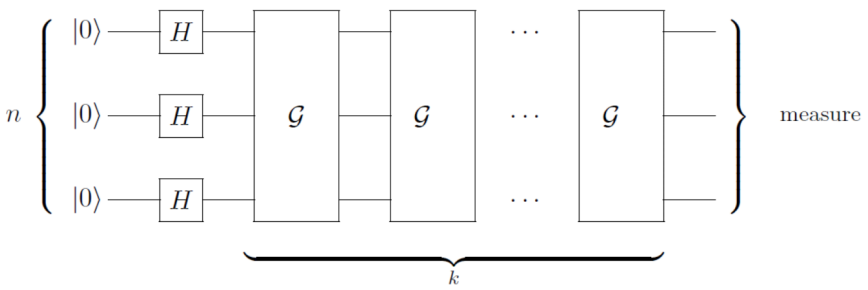


Figure 2.6: Circuit of the algorithm with k Grover iterations. (Illustration from [dW19])

Now we have to choose the setting of k, the number of applications 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 $(2k+1)\theta = \frac{\pi}{2}$, so we set $k \simeq \frac{\pi}{4\theta} - \frac{1}{2}$.

Here is a generalization of Grover's algorithm described in [vM10].

Definition 2.15 (Amplitude amplification)**.** Given a function $f : \mathbb{Z}^n \to \{0,1\}$, we want to solve $f(z) = 1$ with a probability of error $\epsilon < 1/2$.

We dispose of:

- H the Hadamard gate,
- $O_f|z\rangle = (-1)^{f(z)}|z\rangle$,
- $\mathcal{A}$ an algorithm which has a probability p of finding a solution.

The principle of Grover's algorithm is repeated, by taking $\theta = \text{Arcsin}(\sqrt{p})$ and $|U\rangle = \mathcal{A}|0\rangle$.

Algorithm 4 Amplitude amplification

Require: a register $|0^N\rangle$, the probability p of success of $\mathcal{A}$.
Ensure: z such that $f(z) = 1$, or "no solution"
  Apply $H^{\otimes n}$ on the entire register ▷ we get $H^{\otimes n}|0^n\rangle = |U\rangle$
  for $O(1/\sqrt{p})$ iterations **do**
    Apply O_f ▷ Reflection through $|B\rangle$
    Apply $\mathcal{A}R\mathcal{A}^{-1}$ ▷ Reflection through $|U\rangle$
  Measure the first register
  If the resulting z is a solution, return z. Else, return "no solution".

This algorithm has a complexity in $O(1/\sqrt{p})$ queries of f.
For Grover for example, we have taken $O_f = O_{x,\pm}$, $\mathcal{A} = H^{\otimes n}$, and $p = t/N$.

3. Sieve Attack Algorithms on Lattices

In this part, we will present algorithms that attack lattice-based cryptosystems by solving the cryptographic problem of the shortest-vector (SVP). Currently, the two main practical methods to solve SVP are enumeration and sieving according to the recent records for the lattice-challenge [SGBN10].

For a dimension d, enumeration has a low space complexity, but a time complexity super-exponential in the dimension d. Sieving runs with a space complexity of $2^{\theta(d)}$ and in single exponential time $2^{\theta(d)}$ but the hidden constants in the exponents are relatively big. We will only focus on sieving algorithms here.

3.1 Lattice-based cryptographic problems

In this part, we will introduce some definitions about lattices and several cryptographic problems based on them.

Definition 3.1 (Lattice)**.** The lattice $\mathcal{L} \subset \mathbb{R}^m$ generated by a basis $\mathbf{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n)$ of linearly independants $\mathbf{b}_i \in \mathbb{R}^m$ is:

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

$\mathcal{L}$ is said full rank iff $n = m$. By default, we will consider the lattices full rank in the following.

Definition 3.2 (Successive minima λ_i)**.** Let $\text{Ball}(r, n)$ be the ball of dimension n and radius r. For $1 \leqslant i \leqslant m$,

$$\lambda_i(\mathcal{L}) := \inf \left\{ r > 0 \mid \dim\left(\mathcal{L} \cap \text{Ball}(r, n)\right) \geqslant i \right\}$$

So we have $\lambda_1(\mathcal{L}) \leqslant \lambda_2(\mathcal{L}) \leqslant ... \leqslant \lambda_m(\mathcal{L})$, and in particular $\lambda_1 = \inf_{\boldsymbol{y} \in \mathcal{L} \setminus \{0\}} \|\boldsymbol{y}\|$ which is the length of the shortest vector of $\mathcal{L}$.

Any secure cryptosystem has to rely on NP-hard problems. We will here define several lattice-based cryptographic problems, and then show why the first one, SVP, interest us particularly.

Definition 3.3 (SVP)**.** Find some $\boldsymbol{v} \in \mathcal{L}$ such that $\|\boldsymbol{v}\| = \lambda_1(\mathcal{L})$.

α-SVP (approximate SVP): find some non-zero $\boldsymbol{v} \in \mathcal{L}$ such that $\|\boldsymbol{v}\| \leqslant \alpha\lambda_1(\mathcal{L})$.

Definition 3.4 (CVP)**.** Given a target vector $\boldsymbol{t} \in \text{Span}(\mathbf{B})$. Find some $\boldsymbol{v} \in \mathcal{L}$ such that $\|\boldsymbol{v} - \boldsymbol{t}\|$ is minimized, i.e. $\|\boldsymbol{v} - \boldsymbol{t}\| = d(\mathcal{L}, \boldsymbol{t}) = \min_{\mathbf{u} \in \mathcal{L}} d(\mathbf{u}, \boldsymbol{t})$.

α-CVP (Relative): find some $\boldsymbol{v} \in \mathcal{L}$ such that $\|\boldsymbol{v} - \boldsymbol{t}\| \leqslant \alpha.d(\mathcal{L}, \boldsymbol{t})$ (when α increases, the hardness decreases.)

r-AbsCVP (Absolute): find some $\boldsymbol{v} \in \mathcal{L}$ such that $\|\boldsymbol{v} - \boldsymbol{t}\| \leqslant r$

Definition 3.5 (BDD - Bounded Distance Decoding)**.** Given $\boldsymbol{t} \in \text{Span}(\mathbf{B})$. Let $r \leqslant \lambda_1(\mathcal{L})/2$ and $\alpha \leqslant 1/2$.

α-BDD (relative): promised $d(\mathcal{L}, \mathbf{t}) \leqslant \alpha\lambda_1$. Find $\boldsymbol{v} \in \mathcal{L}$ (unique) such that $\|\boldsymbol{v} - \boldsymbol{t}\| \leqslant \alpha.\lambda_1$.

r-AbsBDD (absolute): promised $d(\mathcal{L}, \mathbf{t}) \leqslant r$. Find $\boldsymbol{v} \in \mathcal{L}$ (unique) such that $\|\boldsymbol{v} - \boldsymbol{t}\| \leqslant r$.

The previous problem can be seen as a decoding problem. That illustrates the strong link between lattice-based and code-based cryptography.

Definition 3.6 (Subset Sum Problem)**.** Given a set of integers, find a non-empty subset whose sum is zero.

Definition 3.7 (SIS - Short Integer Solution)**.** Given a matrix $A \in \mathbb{Z}_q^{n \times m}$, find a non-zero vector $\boldsymbol{v} \in \mathbb{Z}^m$ such that $A\boldsymbol{v} = 0$ and $\|\boldsymbol{v}\| \leqslant \beta$.

Definition 3.8 (LWE)**.** Given some couples $(a_i, a_i.s + e_i)$ with $a_i \in \mathbb{Z}_q^n$ and a small error $e_i \in \mathbb{Z}$, find the vector $s \in \mathbb{Z}_q^n$.

Among the NIST competition finalists, Dilithium [DKL+19] is based on modular variants of SIS and LWE, namely: MSIS, SelfTargetSIS and MLWE. The two other lattice-based finalists, Falcon [FHK+19] and NTRU [CDH+19], are based on the NTRU problem [HPS98]. All these problems are proven reduced to the Shortest Vector Problem. Their hardness depends on SVP, so that is why we mainly focus on this problem in particular.

3.2 Sieving algorithms

3.2.1 The Nguyen-Vidick Sieve

Main idea. The NV-sieve [NV08] starts with a list L of long lattice vectors and iteratively builds lists of shorter lattice vectors L' by applying a sieve to L. After poly(d) applications of the sieve, we hope to get a list containing a non-zero shortest lattice vector.

The input L can be obtained by Klein's algorithm [Kle00]. It samples vectors from a discrete Gaussian over the lattice with a large variance, i.e. a relatively high probability of sampling long vectors.

Algorithm 5 The Nguyen-Vidick sieve

Require: a list L of n vectors of norm $\leqslant R$; a sieve factor γ such that $2/3 < \gamma < 1$
Ensure: the list L' of n vectors of norm $\leqslant \gamma R$
  $L' :=$ empty list
  sample $C \subset L \cap \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| \geqslant \gamma R\}$ of poly$(d) \cdot n$
  for each $\boldsymbol{v} \in L \backslash C$ **do**
    if $\|\boldsymbol{v}\| \leqslant \gamma R$ **then**
      add $\boldsymbol{v}$ to the list L'
    else
      for each $\boldsymbol{w} \in C$ **do**
        if $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma R$ **then**
          add $\boldsymbol{v} - \boldsymbol{w}$ to the list L'
          continue the loop for over "$\boldsymbol{v} \in L \backslash C$"
  return L'

Assumption 3.9. *(validated by experiments [NV08]) The angle $\theta(\boldsymbol{v}, \boldsymbol{w})$ between two list vectors $\boldsymbol{v}, \boldsymbol{w} \in L$ follows the same distribution as the distribution of angles $\theta(\boldsymbol{v}, \boldsymbol{w})$ between vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$ drawn at random from the unit sphere.*

It is important to notice that for all the algorithms we will present, there are no proven complexities but only results based on heuristics. It is thus primordial to choose well the heuristic we use and experimentally check them.

Setting of N the number of vectors

We follow the reasoning of [NV08], using the heuristic Assumption 3.9 in the case $\gamma \to 1$. We want to know how many vectors in L we need at average to cover the d-dimensional ball of center 0^d and radius 1 (normalized for any radius R). We take a list vector of length R, represented by a point on this sphere and

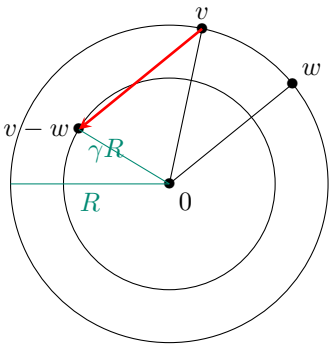


Figure 3.1: For $\boldsymbol{v}$ and $\boldsymbol{w}$ of length R in L, if $\boldsymbol{v} - \boldsymbol{w}$ has a length of γR or less we add it to L'.

consider the ball of radius 1 ($\gamma \to 1$) around this point. The fraction of the sphere occupied by this ball is $\sin^d(\pi/3) = \sqrt{3/4}^d$. About $\sqrt{4/3}^d$ points on the sphere are therefore required to cover the whole sphere, so we set $N = 4/3^{d/2+o(d)}$.

Complexity. The time complexity is dominated by comparing almost every pair of vectors in L in each sieving step. This leads to a time complexity in $O(n^2)$ for each sieve step, which is applied poly(d) times. Therefore the NV-sieve heuristically solves SVP in time $O(n^2) \approx 2^{0.415d+o(d)}$ and in space $O(n) \approx 2^{0.208d+o(d)}$.

3.2.2 The GaussSieve

Main idea. The GaussSieve [MV10] starts with a short list of vectors, and iteratively builds a longer and longer list of lattice vectors, occasionally reducing the lengths of list vectors in the process, until at some point this list L' contains a shortest vector.

Reduce $\boldsymbol{v}$ with $\boldsymbol{w}$: **if** $\|\boldsymbol{v} \pm \boldsymbol{w}\| < \|\boldsymbol{v}\|$ **then** $\boldsymbol{v} := \boldsymbol{v} \pm \boldsymbol{w}$.

Algorithm 6 GaussSieve

Require: a lattice $\mathcal{L}$
Ensure: a list L containing a shortest vector
  L' := empty list
  S := empty stack
  while $\boldsymbol{v}$ is not a shortest vector **do**
    get a vector $\boldsymbol{v}$ from the stack S (or sample new one)
    for each $\boldsymbol{w} \in L'$ **do**
      reduce $\boldsymbol{v}$ with $\boldsymbol{w}$
      reduce $\boldsymbol{w}$ with $\boldsymbol{v}$
      if $\boldsymbol{w}$ has changed **then** remove $\boldsymbol{w}$ from the list L'
        add $\boldsymbol{w}$ to the stack S (unless $\boldsymbol{w} = 0$)
    if $\boldsymbol{v}$ has changed **then**
      add $\boldsymbol{v}$ to the stack S (unless $\boldsymbol{v} = 0$)
    else add $\boldsymbol{v}$ to the list L'
  return L'

Complexity. According to [MV10], there is no known upper bound of the time complexity, but the experiments showed a time complexity in time $2^{0.415d+o(d)}$ and space $2^{0.208d+o(d)}$.

The GaussSieve seems to be more efficient than the NV-sieve in practice.

3.3 Leveled sieving

The 1-level sieve

The 1-level is exactly the NV-sieve algorithm. We can use the notations w_1 and C_1 instead for more consistency with the following.

The 2-level sieve

Main idea. It is a generalization of the 1-level one, with two level of centers C_1 (of radius γ_1) and C_2 (γ_2). It partitions the space to reduce the search space.

We choose these parameters such that $\gamma_2 < 1 < \gamma_1 < \sqrt{2}\gamma_2$. γ_1 is chosen larger than in the 1-level sieve, each of the vectors in C_1 will now cover a larger part of the space.

Algorithm 7 The 2-level sieve

Require: a list L of $(4/3)^{d/2+o(d)}$ vectors of norm $\leqslant R$
Ensure: the list L' of $(4/3)^{d/2+o(d)}$ vectors of norm $\gamma_2 \cdot \leqslant R$
  $L' :=$ empty list
  $C_1 := \{0\}$
  for each $\boldsymbol{v} \in L$ **do**
    if $\exists \boldsymbol{w}_1 \in C_1 : \|\boldsymbol{v} - \boldsymbol{w}_1\| \leqslant \gamma_1 \cdot R$ **then**
      if $\exists \boldsymbol{w}_2 \in C_2^{(\boldsymbol{w}_1)} : \|\boldsymbol{v} - \boldsymbol{w}_2\| \leqslant \gamma_2 \cdot R$ **then**
        add $\boldsymbol{v} - \boldsymbol{w}_2$ to the list L'
      else
        add $\boldsymbol{v}$ to the list $C_2^{(\boldsymbol{w}_1)}$
    else
      add $\boldsymbol{v}$ to the list C_1
      $C_2^{(\boldsymbol{v})} := \{\boldsymbol{v}\}$
  return L'

Complexity. We denote $n_1 = |C_1|$ and $n_2 = |C_2^{(\boldsymbol{w}_1)}|$. [WLTB11] showed that these values only depend of γ_1, γ_2 and d. The number of centers stored is $n_1 n_2$, so it is the space complexity. And for each of these vectors, we need to perform over C_1 and (potentially) another search over a list $C_2^{(\boldsymbol{w}_1)}$. Thus the time complexity is $n_1 n_2 (n_1 + n_2)$. Since, we can look for which values γ_1 and γ_2 minimize the complexities. The optimal γ_2 is 1, and the optimal γ_1 can be calculated at $x \approx 1.093$ a root in $(1, \sqrt{2})$ of the polynome $x^6 - 4x^4 + 4$.

From this, the 2-level sieve solves SVP in time $2^{0.3836d+o(d)}$ and space $2^{0.2557d+o(d)}$.

High-level sieving

Complexity. The k-level sieve has the same structure than the 2-level sieve, with centers sets $C_1, C_2, ..., C_k$ and parameters $\gamma_k < 1 < \gamma_{k-1} < ... < \gamma_1 < \sqrt{2}\gamma_k$.

According to [Laa15], for $k \geqslant 3$, the optimal parameters are $(\gamma_1, ..., \gamma_k) = (x^{k-1}, ..., x, 1)$ where $x \in [1, 2^{1/(2k-2)}]$ is a root of $x^{4k-2} - 4x^{2k} + 4$. Which this choice, time and space complexities are $2^{c_{time} n + o(n)}$ and $2^{c_{space} n + o(n)}$ where $c_{time} = (k+1)\log_2(x)$ and $c_{space} = k \cdot \log_2(x)$.

So these coefficients are lower and lower until 4 : $c_{time} = 0.3774$ and $c_{space} = 0.2925$. However, for $k > 4$, the exponents increase again.

Remark 3.10. It appears that for a same k the trade-off curves (space/time complexities) are exactly the same for the k-level sieve and the sieve based on overlattices [BGJ14] with a tower of k overlattices.

A lattice $\mathcal{L}'$ of dimension d such that $\mathcal{L} \subseteq \mathcal{L}'$ is called an overlattice of $\mathcal{L}$. The main idea of this other algorithm is to use a tower of k overlattices $\mathcal{L}_i$, where $\mathcal{L} = \mathcal{L}_0 \subseteq ... \subseteq \mathcal{L}_k$. In this tower, the lattice $\mathcal{L}_k$ is chosen at the bottom of the tower in a way that ensures that we can efficiently compute a sufficiently large set of short vectors in $\mathcal{L}_k$. Starting from this set of short vectors, we move from each lattice of a tower to the one above using summation of vectors while controlling the growth of norms.

3.4 Locality-sensitive hashing (LSH)

Main idea. We use locality-sensitive hash functions, which have more probability to have equal output hashes when the inputs are close vectors. Instead of reading all the list vectors to reduce a vector $\boldsymbol{v}$, we now partition the space and only consider those who have the same hash as $\boldsymbol{v}$, i.e. their difference has a high probability to have a lower length.

Note that these hash functions have completely the opposite properties that we are usually looking for when we use hash functions.

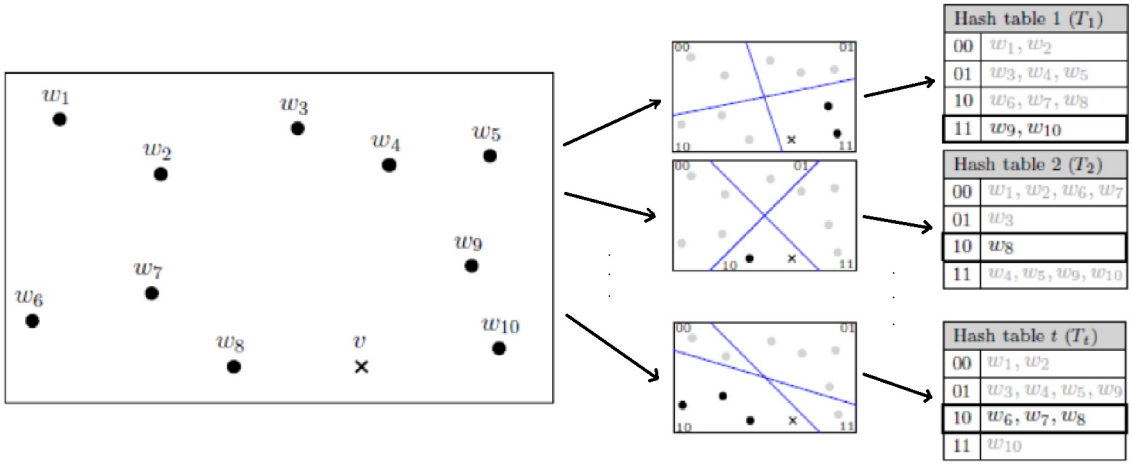


Figure 3.2: In this example, each of the t hash functions separates the space in four areas of different hashes. We compute the hash value of the target $\boldsymbol{v}$ through the hash function and then we look in their associated hash tables to see which vectors have the same hash. Those are candidates to be the closest point of $\boldsymbol{v}$: $\boldsymbol{w}_6, \boldsymbol{w}_7, \boldsymbol{w}_8, \boldsymbol{w}_9$ and $\boldsymbol{w}_{10}$. (Illustration from [Laa15])

Definition 3.11. A family $\mathcal{H} = \{h : \mathbb{R}^d \to \mathcal{U} \subseteq \mathbb{N}\}$ is called (r_1, r_2, p_1, p_2)-sensitive for a similarity measure D if for any $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d$,

- if $D(\boldsymbol{v}, \boldsymbol{w}) < r_1$, then $\Pr_{h \in \mathcal{H}}[h(\boldsymbol{v}) = h(\boldsymbol{w})] \geqslant p_1$ and
- if $D(\boldsymbol{v}, \boldsymbol{w}) > r_2$, then $\Pr_{h \in \mathcal{H}}[h(\boldsymbol{v}) = h(\boldsymbol{w})] \leqslant p_2$.

Ideally, $r_1 \approx r_2$ and $p_1 \approx 1 \gg p_2 \approx 0$. We can choose $D(\boldsymbol{v}, \boldsymbol{w}) := \theta(\boldsymbol{v}, \boldsymbol{w}) = \text{Arccos}\left(\frac{\boldsymbol{v}^T \boldsymbol{w}}{\|\boldsymbol{v}\| \cdot \|\boldsymbol{w}\|}\right)$.

Definition 3.12 (Combined hash function)**.** For $j \in [|1, t|]$, we will denote the combined hash function $h_j := (h_{1,j}, ..., h_{k,j})$. That means that two vectors have the same hash through h_j if and only if they have the same hash through $h_{i,j}$ for all i.

The parameters of the algorithms are:

- t the number of hash tables,
- k the number of hash functions combined together to construct each h_j.

If we do not precise other notations, we consider in all this part $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant t$.

Algorithm 8 NV-sieve with LSH

Require: a list of n vectors of norm $\leqslant R$ $\quad \triangleright\ n = (4/3)^{d/2+o(d)}$
Ensure: the list L' of n vectors of norm $\leqslant \gamma.R$
  $L' :=$ empty list
  sample C with $poly(d) \cdot n$ elements of $L \cap \{x \in \mathbb{R}^d : \|x\| \geqslant \gamma.R\}$
  $T_1, ..., T_t :=$ empty hash tables and sample $k.t$ random hash functions $h_{i,j} \in \mathcal{H}$
  for each $\boldsymbol{w} \in C$ **do**
    add $\boldsymbol{w}$ to the bucket $T_j[h_j(\boldsymbol{w})]$
  for each $\boldsymbol{v} \in L \backslash C$ **do**
    if $\|\boldsymbol{v}\| \leqslant \gamma R$ **then** add $\boldsymbol{v}$ to the list L'
    else
      $\mathcal{C} := \bigcup_{j=1}^{t} T_j[h_j(\boldsymbol{v})]$
      for each $\boldsymbol{w} \in \mathcal{C}$ **do**
        if $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma R$ **then**
          add $\boldsymbol{v} - \boldsymbol{w}$ to the list L'
          break (continue the loop over "$\boldsymbol{v} \in L \backslash C$")
  return L'

Algorithm 9 HashSieve - GaussSieve with LSH

Require: parameters k and t
Ensure: a list L containing a shortest vector
  $L :=$ empty list ; $S :=$ empty stack
  $T_1, ..., T_t :=$ empty hash tables and sample $k.t$ random hash functions $h_{i,j} \in \mathcal{H}$
  while $\boldsymbol{v}$ is not a shortest vector **do**
    get a vector $\boldsymbol{v}$ from the stack S (or sample new one)
    $\mathcal{C} := \bigcup_{j=1}^{t} T_j[h_j(\boldsymbol{v})]$
    for each $\boldsymbol{w} \in L$ **do**
      reduce $\boldsymbol{v}$ with $\boldsymbol{w}$
      reduce $\boldsymbol{w}$ with $\boldsymbol{v}$
      if $\boldsymbol{w}$ has changed **then**
        remove $\boldsymbol{w}$ from the list L
        remove $\boldsymbol{w}$ from all t hash tables T_j
        add $\boldsymbol{w}$ to the stack S (unless $\boldsymbol{w} = 0$)
    if $\boldsymbol{v}$ has changed **then**
      add $\boldsymbol{v}$ to the stack S (unless $\boldsymbol{v} = 0$)
      add $\boldsymbol{v}$ to all hash tables T_j
    else add $\boldsymbol{v}$ to the list L
  return L

The following lemma (inspired from [IM98]) gives an approximation for the values of parameters and the corresponding complexities of the sieve algorithms with LSH.

Assumption 3.13. *For all $\boldsymbol{v}, \boldsymbol{w} \in L$, if $\boldsymbol{v} - \boldsymbol{w} > \gamma R$ then $\boldsymbol{v}$ and $\boldsymbol{w}$ are orthogonal.*

Lemma 3.14. *Suppose there exists a (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{H}$. Let :*

$$\rho = \frac{\log p_1}{\log p_2}, \quad k = \frac{\log n}{\log(1/p_2)}, \quad t = O(n^\rho)$$

Then, assuming the heuristic 3.13, for any $\boldsymbol{v} \in \mathbb{R}^d$, with high probability we can find an element $\boldsymbol{w}^* \in L$ with $D(\boldsymbol{v}, \boldsymbol{w}^*) \leqslant r_1$ or correctly conclude that no element $\boldsymbol{w}^* \in L$ with $D(\boldsymbol{v}, \boldsymbol{w}^*) \leqslant r_2$ exists, with the following costs :

1. *Time for preprocessing the list: $\widetilde{O}(n^{1+\rho \log_{1/p_2}(n)})$*
2. *Space complexity of the preprocessed data: n vectors in $t = \widetilde{O}(n^\rho)$ hash tables: $\widetilde{O}(n^{1+\rho})$*
3. *Time for answering a query $\boldsymbol{v}$: $\widetilde{O}(n^\rho)$*
   Hash evaluations of the query vector $\boldsymbol{v}$: $t = \widetilde{O}(n^\rho)$
   Candidates to compare to the query vector $\boldsymbol{v}$: $\widetilde{O}(n^\rho)$

Since we need to perform these searches $\widetilde{O}(n)$ times, and we need to repeat the whole sieving procedure poly(d) times, the time complexity is in the order $\widetilde{O}(n^{1+\rho})$.

Proof. We have t combined hash functions $h_j = (h_{1,j}, ..., h_{t,j})$.

If $D(\boldsymbol{v}, \boldsymbol{w}) \leqslant r_1, \Pr[h_j(\boldsymbol{v}) = h_j(\boldsymbol{w})] \geqslant p_1^k = \frac{1}{t}$, because we want to have on average at least one vector $\boldsymbol{w}$ which collide with $\boldsymbol{v}$.

And if $D(\boldsymbol{v}, \boldsymbol{w}) \geqslant r_1, \Pr[h_j(\boldsymbol{v}) = h_j(\boldsymbol{w})] \leqslant p_2^k = \frac{1}{n}$, because we want that it happens only on average at most one time on n vectors. We obtain :

$$k.\log(p_2) = -\log(n) \text{ so } k = -\frac{\log(n)}{\log(p_2)} = \frac{\log(n)}{\log(1/p_2)}$$

$$\text{and } 1/t = p_1^k = p_1^{-\frac{\log(n)}{\log(p_2)}} \text{ so } t = n^{\log(p_1)/\log(p_2)} := n^\rho$$

□

The goal is thus to design the LSH family $\mathcal{H}$ so that $\rho = \frac{\log(p_1)}{\log(p_2)}$ is as small as possible.

In practice, the calculated complexity is below the true because the assumption 3.13 is too far from the reality, especially in moderate dimensions. To know the real optimal parameters and complexities for a chosen LSH family, it is necessary to calculate the probabilities of finding good and bad vectors in function of the definition of the LSH family.

Without using the Assumption 3.13, we say this about the bounds for the complexities:

Theorem 3.15. *The NV-sieve with LSH runs with complexities at most:*

1. *Time (hashing): $\widetilde{O}(n \cdot t)$*
2. *Space: $\widetilde{O}(n \cdot t)$*
3. *Time (query): $\widetilde{O}(n^2 \cdot p_2)$*

The next part shows examples of hash function families to run the two previous algorithms.

3.4.1 Hyperplane LSH

Definition 3.16 (Charikar hash family)**.** [Cha02] For a random vector $\boldsymbol{a}$ from a Gaussian distribution, we denote $\mathcal{H} := \{h_{\boldsymbol{a}} : \boldsymbol{a} \in \mathbb{R}^d, \|\boldsymbol{a}\| = 1\}$ and $h_{\boldsymbol{a}}(\boldsymbol{v}) := \begin{cases} 1 & \text{if } \boldsymbol{a}^T\boldsymbol{v} \geqslant 0 \\ 0 & \text{if } \boldsymbol{a}^T\boldsymbol{v} < 0. \end{cases}$

The vector $\boldsymbol{a}$ defines a hyperplane (for which $\boldsymbol{a}$ is a normal vector), and $h_{\boldsymbol{a}}$ maps the two half-spaces separated by this hyperplane respectively to the bits 0 and 1.

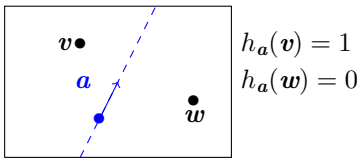


Figure 3.3: Example of a hyperplane hash function.

For another illustration, in the Figure 3.2 the LSH family was a hyperplane LSH with a combination (Definition 3.12) of $k = 2$ hash functions together.

Complexity. According to [Laa15], the NV-sieve with hyperplane LSH heuristically solves SVP in time $2^{0.3366d+o(d)}$ and space $2^{0.2075d+o(d)}$.

3.4.2 Polytope LSH

As a reminder: for a vector $\boldsymbol{v} \in \mathbb{R}^n$, $\|\boldsymbol{v}\|_1 = \sum_{i=1}^n |v_i|$. In ℓ_1, spheres are squares.

Definition 3.17 (Cross-polytope LSH)**.** [TT07] Let be $\boldsymbol{e}_i$ for $i = 1, ..., d$ the ith unit vector in $\mathbb{R}^d$. The d-dimensional cross-polytope is defined as the ℓ_1-unit sphere in $\mathbb{R}^d$ with as vertices $\pm\boldsymbol{e}_1, ..., \pm\boldsymbol{e}_d$.

The hash function associated to the cross-polytope maps a vector $\boldsymbol{v}$ to the nearest vertex to $\boldsymbol{v}$. More formally:

$$h(\boldsymbol{v}) := \text{sign}(v_{i^*})\boldsymbol{e}_{i^*} \text{ with } i^* = \max_{i \in [|1,d|]} |\boldsymbol{v}_i|.$$

Complexity. According to the analysis from [Laa15], the NV-sieve with cross-polytope LSH heuristically solves SVP in time $2^{0.2972d+o(d)}$ and space $2^{0.2075d+o(d)}$.

3.4.3 Hypercone LSH

Definition 3.18 (Spherical LSH)**.** [AINR14] We assume that all points in a data set L lie on the unit sphere $\mathcal{S}^{d-1} = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| = 1\}$. First, we sample $u = 2^{\Theta(\sqrt{d})}$ vectors $\mathbf{s}_1, ..., \mathbf{s}_u \in \mathbb{R}^d$ from a d-dimensional Gaussian distribution with average norm of the $\mathbf{s}_i$'s being 1. To each $\mathbf{s}_i$ from $i = 1$ to $u = 2^{\Theta(d)}$, we then associate a hash region :

$$H_i := \left\{\boldsymbol{x} \in \mathcal{S}^{d-1} : \langle \boldsymbol{x}, \mathbf{s}_i \rangle \geqslant \alpha\right\} \setminus \bigcup_{j=1}^{i-1} H_j$$

where $\alpha \in (0, 1)$. For instance, [Laa15] set $\alpha = d^{-1/4}$.

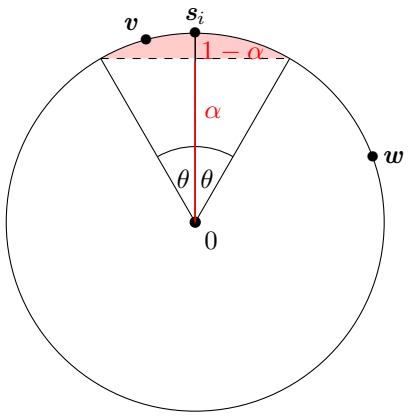


Figure 3.4: The red area represents H_i. For $h_i \in \mathcal{H}$ the function corresponding to $\boldsymbol{s}_i$, we have for instance here $h_i(\boldsymbol{v}) = 1$ and $h_i(\boldsymbol{w}) = 0$.

This indicates how to partition the unit sphere in regions. We now extend this method to all $\mathbb{R}^d$. As spherical hash regions correspond to spherical caps, the normalization of the vectors extends these hash regions to hypercones.

Definition 3.19 (Hypercone LSH)**.** Given a vector $\boldsymbol{v} \in \mathbb{R}^d$, the hypercone hash of a vector $\boldsymbol{v} \in \mathbb{R}^d$ is defined as the spherical hash of the normalized vector $\boldsymbol{v}/\|\boldsymbol{v}\| \in \mathcal{S}^{d-1}$. In other words, denoting the spherical hash family by $\mathcal{H}'$ and the hypercone hash family by $\mathcal{H}$, we have $h(\boldsymbol{v}) = h'(\boldsymbol{v}/\|\boldsymbol{v}\|)$ for $h \in \mathcal{H}, h' \in \mathcal{H}'$ and $\boldsymbol{v} \in \mathbb{R}^d$.

Complexity. The NV-Sieve with hypercone LSH heuristically solves SVP in both time and space $2^{0.2972d+o(d)}$.

Proof. The complete proof of this complexity is in the appendix. □

3.5 Hypercone Locality-sensitive filter (LSF)

A LSF family is a LSH family where a very small number of vectors goes in each hash-bucket. Having a small ρ ($\rho < \frac{1}{2c^2-1}$) gives access to a decoding oracle. (See Lemma 3.14 for a reminder of the definition of the parameter ρ.) We will call "filters" the hash functions from a such LSH family.

In this part, we will focus on LSF families based on the same principle of the hypercone LSH seen just above in the section 3.4.3. We will take a larger α, which has effect to reduce the number of vectors surviving one filter.

Definition 3.20 (Hypercone locality-sensitive filters)**.** A hypercone filter f is characterised by a unit vector $\boldsymbol{s}$ chosen randomly and a value $\alpha \in (0, 1)$. A normalized vector $\boldsymbol{w}$ survives f, i.e. $f(\boldsymbol{w}) = 1$ iff $\langle \boldsymbol{w}, \boldsymbol{s} \rangle \geqslant \alpha$. In this case, we will say $\boldsymbol{w}$ survives f.

Let μ be the canonical Lebesgue measure over $\mathbb{R}^d$. We denote the unit sphere by $\mathcal{S}^{d-1} := \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| = 1\}$ and the half-spaces by $\mathcal{H}_{\boldsymbol{v},\alpha} := \{\boldsymbol{x} \in \mathbb{R}^d : \langle \boldsymbol{v}, \boldsymbol{x} \rangle \geqslant \alpha\}$. For $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$ such that $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \cos(\theta)$ and $\alpha, \beta \in (0, 1)$ we denote spherical caps and wedges (Figure 3.5) by:

$$\mathcal{C}_{\boldsymbol{v},\alpha} := \mathcal{S}^{d-1} \cap \mathcal{H}_{\boldsymbol{v},\alpha}$$

$$\mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta} := \mathcal{S}^{d-1} \cap \mathcal{H}_{\boldsymbol{v},\alpha} \cap \mathcal{H}_{\boldsymbol{w},\beta}$$

Lemma 3.21. *[BDGL16] Given $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$ with $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \cos(\theta)$, the volume of spherical caps and wedges for a given large d is:*

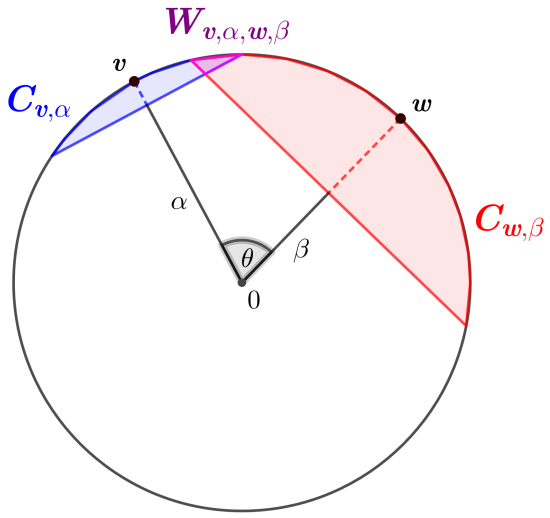


Figure 3.5: Representation on the unit sphere of caps, wedges, and parameters α and β.

$$\mathcal{C}(\alpha) := \frac{\mu(\mathcal{C}_{\boldsymbol{v},\alpha})}{\mu(\mathcal{S}^{d-1})} = poly(d) \cdot (1-\alpha^2)^{d/2}$$

$$\mathcal{W}(\alpha,\beta,\theta) := \frac{\mu(\mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta})}{\mu(\mathcal{S}^{d-1})} = poly(d) \cdot (1-\gamma^2)^{d/2}$$

$$with\ \gamma = \sqrt{\frac{\alpha^2+\beta^2-2\alpha\beta\cos(\theta)}{\sin^2(\theta)}}$$

Main idea. The purpose is to speed up the algorithms 8 and 9 (NV- and Gauss- sieves with LSH) by selecting the filters that are relevant for the input vectors, instead of sampling them totally randomly. The method proposed by [BDGL16] is to sample a code C that admit a fast decoding algorithm, which gives the list of candidate vectors for the reducing.

Definition 3.22 (Random product codes)**.** A random product code C of parameters $[d, m, B]$ on subsets of $\mathbb{R}^d$ and of size $M = B^m$ is defined as a code of the form

$$C = Q \cdot (C_1 \times C_2 \times \cdots C_m),$$

where Q is a uniformly random rotation over $\mathbb{R}^d$ and the subcodes $C_1, ..., C_m$ are sets of B vectors, sampled uniformly and independently random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$.

To be useful in our case, the code C has to be efficiently decodable and to behave as a real random code over the sphere, considering the probability of collision between two vectors.

Theorem 3.23. *[BDGL16] There exists an algorithm that, given the description $Q, C_1, ..., C_m$ of a random product code C of parameters $[d, m, B]$ and a target vector $\boldsymbol{t}$, returns the set $S = C \cap \mathcal{C}_{\boldsymbol{t},\alpha}$ in average time*

$$\mathcal{T}(M,\alpha) = O\Big(d \cdot B + m \cdot B\log(B) + m \cdot M \cdot \mathcal{C}_d(\alpha)\Big)$$

For a given $i \in [|1, m|]$, $c_{i,j}$ with $j \in [|1, B|]$ are the elements of C_i after the sort. We denote $d_{i,j} := \langle \mathbf{c}_{i,j}, \mathbf{t}_i \rangle$, with $\mathbf{t}_i$ the vector with all-zero except the ith position which is equal to t_i.

Algorithm 10 List decoding

Require: The description $Q, C_1, ..., C_m$ of the code C, a target vector $\boldsymbol{t} \in \mathbb{R}^d$ and $\alpha \in (0, 1)$.
Ensure: $L = C \cap \mathcal{C}_{\boldsymbol{t},\alpha}$ containing all α-close words to $\mathbf{t}$.

```
L := empty list
Sort each C_i by decreasing d_{i,j}
R_i := α − Σ_{k=i+1}^{m} d_{k,1}
for each j_1 ∈ [|1, B|] such that d_{i,j_2} ⩽ R_1 do
  for each j_2 ∈ [|1, B|] such that d_{i,j_2} ⩽ R_2 − d_{1,j_1} do
    ⋮
    for each j_m ∈ [|1, B|] such that d_{i,j_m} ⩽ R_m − d_{1,j_1} − ... − d_{m−1,j_{m−1}} do
      Add (c_{1,j_1}, ..., c_{m,j_m}) to L
return L
```

Proof. This algorithm has an overall running time which is the sum of the following three terms:

- mB dot-products of dimension b,
- $O(m \cdot B \log(B))$ operations during the sorting step,
- $O(m \cdot |S|)$ visits of nodes during the pruned enumeration, where $|S| = O(M \cdot \mathcal{C}_d(\alpha))$.

□

Thus, we can consider we have an oracle that allows us to compute efficiently relevant filters to the input list of vectors.

The algorithm has the following parameters:

- $t \in \mathbb{N}$: number of hash tables.
- $\alpha \in (0, 1)$: the query parameter for finding the relevant vectors of a given target vector. Increasing α decreases the number of queries to filters to find nearby vectors.
- $\beta \in (0, 1)$: the insertion parameter for finding all filters that a vector is inserted in. Increasing β decreases the number of vectors we store in buckets.

The parameter k is set at 1. So we do not longer combine hash functions together.

Algorithm 11 NV-sieve with LSF

Require: a list of n vectors of norm $\leqslant R$ $\quad \triangleright\ n = (4/3)^{d/2+o(d)}$
Ensure: the list L' of n vectors of norm $\leqslant \gamma.R$
  $L' :=$ empty list
  sample $\widehat{C}$ with $poly(d) \cdot n$ elements of $L \cap \{x \in \mathbb{R}^d : \|x\| \geqslant \gamma.R\}$ $\quad \triangleright$ Pay attention not confusing the code C and the set of centers $\widehat{C}$.
  sample m random subcodes $C_1, ..., C_m$
  for each $\boldsymbol{w} \in \widehat{C}$ **do**
    $\mathcal{F}_{\boldsymbol{w},\beta} :=$ set of all β-close filters to $\boldsymbol{w}$
    add $\boldsymbol{w}$ to all β-close filters $f \in \mathcal{F}_{\boldsymbol{w},\beta}$
  for each $\boldsymbol{v} \in L \backslash \widehat{C}$ **do**
    if $\|\boldsymbol{v}\| \leqslant \gamma R$ **then** add $\boldsymbol{v}$ to the list L'
    else
      $\mathcal{F}_{\boldsymbol{v},\alpha} :=$ set of all α-close filters to $\boldsymbol{v}$
      $\mathcal{C} := \bigcup_{f \in \mathcal{F}_{\boldsymbol{v},\alpha}} B_f$
      for each $\boldsymbol{w} \in \mathcal{C}$ **do**
        if $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma R$ **then**
          add $\boldsymbol{v} - \boldsymbol{w}$ to the list L'
          break (continue the loop over "$\boldsymbol{v} \in L \backslash \widehat{C}$")
  return L'

Theorem 3.24 (Costs of one query)**.** *Given n points uniformly distributed on the sphere, and t spherical filters with parameters α, β, the time to answer a query is:*

$$\mathcal{T}_{query} = \widetilde{O}(t \cdot \mathcal{C}(\alpha) \cdot [1 + n \cdot \mathcal{C}(\beta)])$$

Proof. First, to answer a query, we compute the $O(t \cdot \mathcal{C}(\alpha))$ relevant filters using list decodable random product codes.

We assume the vectors in L are uniformly distributed on the sphere, and all filters cover an equal portion of the sphere (Assumpion 3.9). So, each filter bucket will roughly have the same size.

In total each list vector has been inserted in $O(t \cdot \mathcal{C}(\beta))$ filters, leading to $O(n \cdot t \cdot \mathcal{C}(\beta))$ total entries in the filter database, and $O(n \cdot \mathcal{C}(\beta))$ vectors per filter.

We have the cost of computing relevant filters $O(t \cdot \mathcal{C}(\alpha))$ summed with the cost of comparing the vector to all other vectors in these filters $\widetilde{O}(t \cdot \mathcal{C}(\alpha) \cdot n \cdot \mathcal{C}(\beta))$. In total, we get that a query costs $\widetilde{O}(t \cdot \mathcal{C}(\alpha) \cdot [1 + n \cdot \mathcal{C}(\beta)])$. □

Setting of t the number of filters

We want to make sure that we do not miss nearby vectors at angle $\frac{\pi}{3}$, which are potential reducing vectors. The probability that two vectors at angle $\frac{\pi}{3}$ are found through a collision is $\widetilde{O}(t \cdot \mathcal{W}(\alpha, \beta, \frac{\pi}{3}))$, that we want to be close to 1. Thus:

$$t = \widetilde{O}\Big(\frac{1}{\mathcal{W}(\alpha, \beta, \frac{\pi}{3})}\Big)$$

Choice of the query and insertion parameters α and β

We keep $n = (3/4)^{d/2+o(d)}$, for the reason explained in section 3.2.

Time complexity. $\mathcal{T} = \mathcal{T}_{preprocessing} + \mathcal{T}_{queries}$. For the preprocessing, for each of the n input vectors we have compute the $t \cdot \mathcal{C}(\beta)$ relevant filters.

$$\begin{aligned}
\mathcal{T}_{preprocessing} &= \widetilde{O}\Big(n \cdot t \cdot \mathcal{C}(\beta)\Big) \\
&= \widetilde{O}\Big(\frac{n \cdot \mathcal{C}(\beta)}{\mathcal{W}(\alpha, \beta, \pi/3)}\Big) \\
&= \widetilde{O}\Big(\Big(\frac{4(1-\beta^2)}{3-4(\alpha^2+\beta^2-\alpha\beta)}\Big)^{d/2}\Big) \\
\mathcal{T}_{queries} &= \widetilde{O}(n \cdot \mathcal{T}_{query}) \\
&= \widetilde{O}\Big(n \cdot \frac{\mathcal{C}(\alpha) \cdot [1 + n \cdot \mathcal{C}(\beta)]}{\mathcal{W}(\alpha, \beta, \frac{\pi}{3})}\Big) \\
&= \widetilde{O}\Big(\Big(\frac{4(1-\alpha^2)}{3-4(\alpha^2+\beta^2-\alpha\beta)}\Big)^{d/2}\Big[1 + (4(1-\beta^2)/3)^{d/2}\Big]\Big)
\end{aligned}$$

Space complexity. We store n vectors and $t.n.\mathcal{C}(\beta)$ filters, represented also by vectors in $\mathbb{R}^d$.

$$\begin{aligned}
\mathcal{S} &= \widetilde{O}\Big(n + n \cdot t \cdot \mathcal{C}(\beta)\Big) \\
&= \widetilde{O}\Big(n + \frac{n \cdot \mathcal{C}(\beta)}{\mathcal{W}(\alpha, \beta, \pi/3)}\Big) \\
&= \widetilde{O}\Big((4/3)^{d/2} + \Big(\frac{4(1-\beta^2)}{3-4(\alpha^2+\beta^2-\alpha\beta)}\Big)^{d/2}\Big)
\end{aligned}$$

Complexity. The NV-sieve with hypercone LSF solves SVP in

- time and space $\mathcal{T} = \mathcal{S} = (3/2)^{d/2+o(d)} \simeq 2^{0.292d+o(d)}$ for $\alpha = \beta = \frac{1}{2}$,
- time $\mathcal{T} = (5/3^{d/2+o(d)}) \simeq 2^{0.292d+o(d)}$ and space $\mathcal{S} = (4/3)^{d/2+o(d)} \simeq 2^{0.368d+o(d)}$ for $\alpha = \frac{1}{4}$ and $\beta = \frac{1}{2}$.

By setting $\beta = \frac{1}{2}$ and varying α between these two values, we can get the optimal time complexity for a given space complexity.

3.6 Quantum speed-ups

In this part, we are going to look at the complexities we can hope in the quantum model. To do that, we apply Grover's algorithm (Amplitude amplification, Algorithm 4 in section 2.4) on the search phase for the candidate vectors.

As a reminder, Grover's algorithm returns $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = 1$ with a complexity of $O(\sqrt{n})$ instead of $O(n)$ in classical. However, this quadratic gain only concerns a search phase, not the whole sieves.

The Nguyen-Vidick and Gauss Sieves.

Given two lattice vectors $\boldsymbol{v}$ and $\boldsymbol{w}$, we define $f_{\boldsymbol{v}}(\boldsymbol{w}) := \begin{cases} 1 & \text{if } \|\boldsymbol{v}-\boldsymbol{w}\| \leqslant \gamma R \\ 0 & \text{else.} \end{cases}$

Algorithm 12 The Quantum NV-sieve

Require: a list L of n vectors of norm $\leqslant R$; a sieve factor γ such that $2/3 < \gamma < 1$
Ensure: the list L' of n vectors of norm $\leqslant \gamma R$
  $L' :=$ empty list
  sample $C \subset L \cap \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| \geqslant \gamma R\}$ of $\text{poly}(d) \cdot n$
  for each $\boldsymbol{v} \in L \backslash C$ **do**
    if $\|\boldsymbol{v}\| \leqslant \gamma R$ **then**
      add $\boldsymbol{v}$ to the list L'
    else
      $\boldsymbol{w} :=$ Grover$(f_{\boldsymbol{v}})$ ▷ We get a $\boldsymbol{w}$ such that $\|\boldsymbol{v}-\boldsymbol{w}\| \leqslant \gamma R$ (or "no solution").
      add $\boldsymbol{v}-\boldsymbol{w}$ to the list L'
  return L'

Complexity. [LMvdP13] For a given $\boldsymbol{v}$, the complexity of finding a reducing $\boldsymbol{w}$ (if it exists) is $\widetilde{O}(\sqrt{N})$. This leads to complexities in time $(4/3)^{\frac{3d/2+o(d)}{2}} \simeq 2^{0.311d+o(d)}$ and in space $(4/3)^{d/2+o(d)} \simeq 2^{0.208d+o(d)}$.

Classical time complexity was $2^{0.415d+o(d)}$ for the same space complexity. So the quantum NV-sieve is 25% faster than the classical one.

The GaussSieve obtains exactly the same speed-up.

Leveled and Overlattice Sieves.

These two methods give complexities not better than the quantum NV-sieve. In short, the reason is that they operates with many search subroutines which cannot be speeded up by simply applying Grover.

LSH and LSF.

Given two lattice vectors $\boldsymbol{v}$, $\boldsymbol{w}$ and a set of vectors $\mathcal{C}$, we define $f_{\boldsymbol{v},\mathcal{C}}(\boldsymbol{w}) := \begin{cases} 1 & \text{if } \boldsymbol{w} \in \mathcal{C} \textbf{ and } \|\boldsymbol{v}-\boldsymbol{w}\| \leqslant \gamma R \\ 0 & \text{else.} \end{cases}$

Algorithm 13 Quantum NV-sieve with LSH

Require: a list of n vectors of norm $\leqslant R$ $\quad \triangleright\ n = (4/3)^{d/2+o(d)}$
Ensure: the list L' of n vectors of norm $\leqslant \gamma.R$
  $L' :=$ empty list
  sample C with $poly(d) \cdot n$ elements of $L \cap \{x \in \mathbb{R}^d : \|x\| \geqslant \gamma.R\}$
  $T_1, ..., T_t :=$ empty hash tables and sample $k.t$ random hash functions $h_{i,j} \in \mathcal{H}$
  for each $\boldsymbol{w} \in C$ **do**
    add $\boldsymbol{w}$ to the bucket $T_j[h_j(\boldsymbol{w})]$
  for each $\boldsymbol{v} \in L \backslash C$ **do**
    if $\|\boldsymbol{v}\| \leqslant \gamma R$ **then** add $\boldsymbol{v}$ to the list L'
    else
      $\mathcal{C} := \bigcup_{j=1}^{t} T_j[h_j(\boldsymbol{v})]$
      $\boldsymbol{w} :=$ Grover$(f_{\boldsymbol{v},\mathcal{C}})$ $\quad \triangleright$ We get a $\boldsymbol{w} \in \mathcal{C}$ such that $\|\boldsymbol{v}-\boldsymbol{w}\| \leqslant \gamma R$ (or "no solution").
      add $\boldsymbol{v}-\boldsymbol{w}$ to the list L'

The classical time complexity of a search in the candidates list is $n^{1-\alpha}$. By applying Grover for this search, the time complexity is reduced to $\sqrt{n^{1-\alpha}}$. So the total number of comparisons is $\widetilde{O}(n^{(3-\alpha)/2})$ and the number of hashing remains at $\widetilde{O}(n \cdot t)$. The parameters k and t have then to be computed for each LSH family to get the optimal complexities of the algorithm in its quantum version.

Hyperplane LSH According to [Laa15], it solves SVP in time $2^{0.286d+o(d)}$ and space $2^{0.208d+o(d)}$, which is a speed-up of about 15%.

Cross-polytope LSH According to [BL15], it solves SVP in time $2^{0.268d+o(d)}$ and space $2^{0.208d+o(d)}$, which is a speed-up of about 10%.

Hypercone LSH According to [LMvdP13], it solves SVP in time $2^{0.268d+o(d)}$ and space $2^{0.208d+o(d)}$, which is a speed-up of about 10%.

Note that cross-polytope and hypercone LSH has the same complexities in both classical and quantum.

Hypercone LSF According to [Laa15], it solves SVP in time $2^{0.2975d+o(d)}$ and space $2^{0.208d+o(d)}$, which is a speed-up of about 9%.

4. Conclusion

Here is a recap of the complexities obtained for the heuristic algorithms heuristically solving SVP by sieving we presented in this report. Remember that these results are only under heuristic assumptions.

Algorithm	Classical complexity		Quantum complexity	
	c_{time}	c_{space}	c_{time}	c_{space}
NV-sieve [NV08]	0.415	0.208	0.311	0.208
GaussSieve [MV10]	0.415	0.208	0.311	0.208
2-level sieve [WLTB11]	0.384	0.256	0.311	0.208
3-level sieve [ZPH13]	0.3778	0.288	0.311	0.208
4- and high-level sieve [Laa15]	0.3774	0.293	0.311	0.208
Overlattice sieve [BGJ14]	0.3774	0.293	0.311	0.208
Hyperplane LSH [Laa15]	0.337	0.208	0.286	0.208
Hypercone LSH [Laa15]	0.298	0.208	0.268	0.208
Cross-polytope LSH [Laa15]	0.298	0.208	0.268	0.208
Hypercone LSF [BDGL16]	0.292	0.208	0.265	0.208

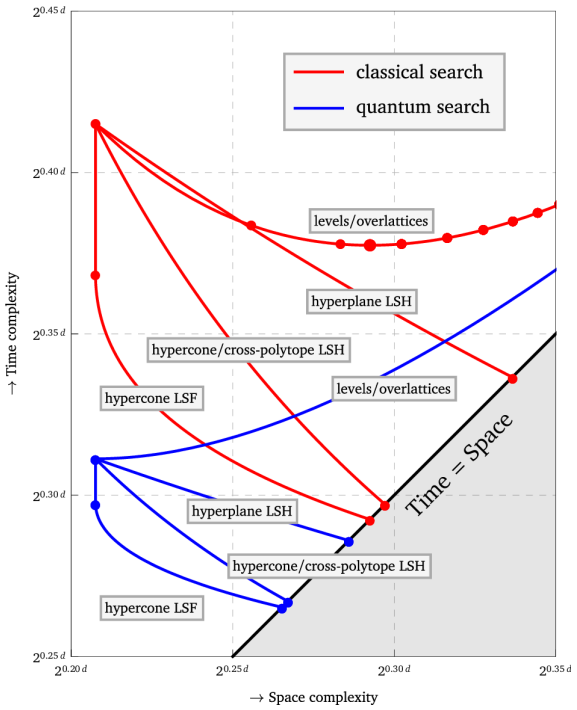


Figure 4.1: The space/time trade-offs curves of the sieving algorithms (Illustration from [Laa15]).

We see that the hypercone filtering is the actual most promising method, in both classical and quantum models. For the further researches, we can pursue several leads to try improving the complexity of solving SVP by sieving.

The first lead is to modify the NV-sieve with hypercone LSH (Algorithm 8) by cutting the dimension d in half. Concretely, for $x = (x_1, ..., x_d)$ a lattice vector, we define $x_L = (x_1, ..., x_{d/2}, 0, ..., 0)$ and $x_R = (0, ..., 0, x_{d/2+1}, ..., x_d)$. In the algorithm, instead of testing $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma R$, we test if $\|\boldsymbol{v}_L - \boldsymbol{w}_L\| \leqslant \gamma R/\sqrt{2}$ and $\|\boldsymbol{v}_G - \boldsymbol{w}_G\| \leqslant \gamma R/\sqrt{2}$. (The $1/\sqrt{2}$ are normalization factors.)

This method has the advantage to create a saturation of candidate vectors. That allows to recoverer the nearest and it works well with an application of Grover's algorithm.

Then, we could also analyse the NV-sieve with hypercone LSF (Algorithm 11) with this modification.

Secondly, we could take a larger the size of the input list of any NV-sieve based algorithm. The goal is again to get a saturation and find more quickly a reducing vector. There is rapidly a problem of exploding space complexity with a naive application of this method, but it could be overcome it if we do not save all the vectors at time, but only those which are in a delimited part of the sphere.

Thirdly, more generally, we can study other LSH families and see if they are more efficient. Indeed, the hypercone LSH family has good theoretical results but computing angles between vectors in high dimension is not the most practical. We also can focus on families specifically designed for quantum sieves.

5. References

[AINR14] Alexandr Andoni, Piotr Indyk, Huy Lê Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. *SODA*, page 1021, 2014.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. *Proc. of the 2016 Annual ACM-SIAM Symposium on Discrete Algorithms*, 2016.

[BGJ14] Anja Becker, Nicolas Gama, and Antoine Joux. A sieve algorithm based on overlattices. *LMS Journal of Computation and Mathematics. Volume 17, Issue A Algorithmic Number Theory Symposium XI*, pages 49 – 70, 2014.

[BL15] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope lsh. *Cryptology ePrint Archive, Report 2015/823*, pages 1 – 25, 2015.

[CDH+19] C. Chen, O. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, J.M. Schanck, P. Schwabe, W. Whyte, and Z. Zhang. Ntru. *Round-3 submission to the NIST pqc project*, 2019.

[Cha02] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. *STOC*, pages 380 – 388, 2002.

[DKL+19] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-dilithium, algorithm specifications and supporting documentation. *Round-3 submission to the NIST pqc project*, 2019.

[dW19] Ronald de Wolf. *Quantum Computing: Lecture Notes*. University of Amsterdam, 2019.

[FHK+19] P-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Falcon: Fast-fourier lattice-based compact signatures over NTRU. *Round-3 submission to the NIST pqc project*, 2019.

[Gro96] Lov Grover. A fast quantum mechanical algorithm for database search. *Proc.28th Annual ACM Symposium on the Theory of Computing STOC*, pages 212 – 219, 1996.

[HPS98] J. Hoffstein, J. Pipher, and J.H. Silverman. *NTRU: a ring-based public key cryptosystem.*, volume 1423. Buhler, J.P. (ed.) ANTS 1998. LNCS, 1998.

[HW79] G. H. Hardy and E. M. Wright. An introduction to the theory of numbers. *Oxford University Press*, 1979.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: To- wards removing the curse of dimensionality. *STOC*, pages 604 – 613, 1998.

[Kle00] Philip Klein. *Finding the closest lattice vector when it's unusually close*. SODA, 2000.

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

[LMvdP13] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Solving the shortest vector problem in lattices faster using quantum search. *PQCrypto*, pages 83 – 101, 2013.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. *SODA*, page 1468 – 1480, 2010.

[NV08] P.Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Math. Crypt. 2*, pages 181 – 207, 2008.

[SGBN10] M. Schneider, N. Gama, P. Baumann, and L. Nobach. Lattice challenge. *online at https://latticechallenge.org/*, since 2010.

[Sho94] Peter Shor. Algorithms for quantum computation: Discrete logarithm and factoring. *Proc. 35th Annual Symposium on Foundations of ComputerScience*, pages 124 – 134, 1994.

[TT07] Kengo Terasawa and Yuzuru Tanaka. Spherical lsh for approximate nearest neighbor search on unit hypersphere. *WADS*, pages 27 – 38, 2007.

[vM10] D. van Melkebeek. *Quantum Information Processing. Lecture 7: Query Lower Bounds*. University of Maryland, 2010.

[WLTB11] X. Wang, M. Liu, C. Tian, and J. Bi. Improved nguyen-vidick heuristic sieve algorithm for shortest vector problem. ASIACCS, pages 1 – 9, 2011.

[ZPH13] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the shortest vector problem. *SAC*, pages 29 – 47, 2013.

6. Appendix: Proof of the Complexity of the NV-sieve with hypercone LSH

We want to compute the complexity of the NV-sieve (Algorithm 8) with a hypercone LSH family $\mathcal{H}$ (Definition 3.19). This proof follows the one developed in [Laa15]. First we have to check if two nearby vectors collide with a constant probability $1-\epsilon$, then that a distant vector collide happens with a exponentially small probability, and finally we balance the parameters to get an optimal complexity.

The Lemma 3.1 from [AINR14] stated the following result:

Lemma 6.1. *Let $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d$ and let θ denote the angle between $\boldsymbol{v}$ and $\boldsymbol{w}$. Then for large d, the hypercone hash family $\mathcal{H}$ satisfies:*

$$p(\theta) := Pr_{h\in\mathcal{H}}[h(\boldsymbol{v}) = h(\boldsymbol{w})] = \exp\left[-\frac{\sqrt{d}}{2}\tan^2\left(\frac{\theta}{2}\right)(1+o(1))\right]$$

Lemma 6.2. *The probability that a reducing vector $\boldsymbol{w}$ is a candidate vector for $\boldsymbol{v}$ with $\theta = \theta(\boldsymbol{v}, \boldsymbol{w})$ is*

$$p^*(\theta) := 1 - (1 - p(\theta)^k)^t.$$

Proof. It suffices to explain the different probabilities. The probability that at least one vector at angle θ have k hashes equal to $\boldsymbol{v}$'s is $p(\theta)^k$. So the probability that there is no vector at angle θ with k same hashes is $1-p(\theta)^k$. So the probability that there is no vector at angle θ with k same hashes for t hash tables is $(1-p(\theta)^*)^t$. The fact that at least one vector at angle θ has k same hashes than $\boldsymbol{v}$ for t hash tables is equivalent to say that at least one vector at angle θ which reduces $\boldsymbol{v}$ is selected to be a candidate, and its probability is $1-(1-p(\theta)^k)^t$. □

Nearby vectors collide with constant probability

Lemma 6.3. *Let $\epsilon > 0$ and let $k = \frac{6.(\ln(t)-\ln(\ln(1/\epsilon))}{\sqrt{d}}$. Then the probability that nearby vectors collide in at least one of the hash tables is at least $1-\epsilon$.*

Proof. The probability that a reducing vector $\boldsymbol{w}$ is a candidate vector, given the angle $\theta = \theta(\boldsymbol{v}, \boldsymbol{w}) \in (0, \pi/3)$ is $p_1^* = \mathbb{E}_{\theta\in(0,\pi/3)}[p^*(\theta)]$. Since $p^*(\theta)$ is strictly decreasing in θ, we can obtain a lower bound by substituting $\theta = \frac{\pi}{3}$.

$$p_1^* \geqslant p^*\Big(\frac{\pi}{3}\Big) = 1 - \Big(1 - p\Big(\frac{\pi}{3}\Big)^k\Big)^t$$

We replace by its expression $p(\frac{\pi}{3}) = \exp\left[-\frac{\sqrt{d}}{2}\tan^2(\frac{\pi/3}{2})\right] = \exp(-\frac{\sqrt{d}}{6})$, and we choose $k = \frac{6.(\ln(t)-\ln(\ln(1/\epsilon))}{\sqrt{d}}$ for any $\epsilon > 0$.

$$
\begin{aligned}
p_1^* &\geqslant 1 - \Big(1 - \exp(-\frac{\sqrt{d}}{6}.\frac{6(\ln(t) - \ln\ln(1/\epsilon))}{\sqrt{d}})\Big)^t \\
&= 1 - \Big(1 - \exp(-\ln(t) + \ln\ \ln(1/\epsilon))\Big)^t \\
&= 1 - \Big(1 - \frac{\ln(1/\epsilon)}{t}\Big)^t \\
&\geqslant 1 - \epsilon \text{ because } \forall x, 1 - x \leqslant e^{-x}, \text{ with here } x = \frac{\ln(1/\epsilon)}{t}.
\end{aligned}
$$

ϵ is a constant which does not depend on d. This achieves the proof. □

Distant vectors collide with low probability

As a reminder,

Assumption 3.9. The angle $\theta(\boldsymbol{v}, \boldsymbol{w})$ between two list vectors $\boldsymbol{v}, \boldsymbol{w} \in L$ follows the same distribution as the distribution of angles $\theta(\boldsymbol{v}, \boldsymbol{w})$ between vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$ drawn at random from the unit sphere.

Lemma 6.4. *[Laa15] If Assumption 3.9 holds, then the probability density function $f(\theta)$ of angles between non-reducing vectors satisfies*

$$f(\theta) = \sqrt{\frac{2d}{\pi}}(\sin\theta)^{d-2}$$

Since Lemma 6.3 expresses k depending on t, this means that only t remains to be chosen. We write: $n = 2^{c_n \cdot d}$, $t = 2^{c_t \cdot d}$, and $\gamma_1 = \log_2(\sqrt{4/3}) \approx 0.2075$.

Lemma 6.5. *Let $c_n \geqslant \gamma_1$. Then, if Assumption 3.9 holds, for large d the probability of collisions between distant vectors is bounded by:*

$$p_2^* = \Pr_{h_{i,j} \subset \mathcal{H}}\left[\boldsymbol{v}, \boldsymbol{w} \textit{ collide} \mid \theta(\boldsymbol{v}, \boldsymbol{w}) > \frac{\pi}{3}\right] \leqslant n^{-\alpha + o(1)}$$

where $\alpha \in (0, 1)$ is defined as

$$\alpha = -\frac{1}{c_n}\Big[\max_{\theta \in (\frac{\pi}{3}, \frac{\pi}{2})}\Big\{\log_2\sin(\theta) - c_t\Big(3\tan^2(\frac{\theta}{2}) - 1\Big)\Big\}\Big]$$

Remark 6.6. The condition $c_n \geqslant \gamma_1$ is easily satisfiable with no additional cost caused by a too large n. Indeed, we have seen in Section 3.2 the explanation of the setting of c_n.

Proof. Two "bad" vectors has an angle $\theta \in (\frac{\pi}{3}, \frac{\pi}{2})$. Letting $f(\theta)$ denote the density of angles $\theta \in (\frac{\pi}{3}, \frac{\pi}{2})$, we have:

$$
\begin{aligned}
p_2^* &= \mathbb{E}_{\theta \in (\frac{\pi}{3}, \frac{\pi}{2})}[p^*(\theta)] = \int_{\pi/3}^{\pi/2} f(\theta)p^*(\theta)\ \mathrm{d}\theta \\
&= \int_{\pi/3}^{\pi/2} \Big(\sqrt{\frac{2d}{\pi}}(\sin\theta)^{d-2}(1 + o(1))\Big)\Big(1 - (1 - \exp\Big[\frac{k\sqrt{d}}{2}\tan^2(\frac{\theta}{2})(1 + o(1))\Big])^t\Big)\ \mathrm{d}\theta \\
&\qquad \text{by Lemma 6.4 for } f(\theta) \text{ and Lemma 6.1 for } p^*(\theta) \\
&\leqslant \int_{\pi/3}^{\pi/2} \sin(\theta)^d\Big(1 - (1 - \exp\Big[-3\ln(t)\tan^2(\frac{\theta}{2}))\Big])^t\Big)\ \mathrm{d}\theta \\
&\qquad \text{we omit the normalizing constant (negligible) for } f(\theta), \\
&\qquad \text{and } k \text{ given by Lemma 6.3.}
\end{aligned}
$$

For more practicity, let write $w(\theta) = [-3\ln(t)\tan^2(\theta/2)](1+o(1))$.

Note that for $\theta \in (\frac{\pi}{3}, \frac{\pi}{2})$ far from $\pi/3$, we have $w(\theta) << w(\pi/3) = -\ln(t)$, so $\Big(1-\exp(w(\theta))\Big)^t \approx 1 - t.\exp(w(\theta))$. However, the integration range include $\pi/3$. So we split the integration interval at $\pi/3+\delta$ where $\delta = \Theta(1/\sqrt{d})$.

$$\begin{aligned} p_2^* &= \int_{\pi/3}^{\pi/3+\delta} f(\theta)p^*(\theta)\ \mathrm{d}\theta + \int_{\pi/3+\delta}^{\pi/2} f(\theta)p^*(\theta)\ \mathrm{d}\theta \\ &:= I_1 + I_2 \end{aligned}$$

Bounding I_1. f is croissant in θ, thus for $\pi/3 \leqslant \theta \leqslant \pi/3+\delta$, we have $f(\theta) \leqslant f(\frac{\pi}{3}+\delta)$. And with $p^*(\theta) \leqslant 1$, we obtain

$$\begin{aligned} I_1 &\leqslant \mathrm{poly}(d)\sin(\frac{\pi}{3}+\delta)^d \\ &= \mathrm{poly}(d)(\sqrt{3}/2)^d(1+O(\delta))^d := 2^{-\gamma_1 d + o(d)} \end{aligned}$$

by the Taylor expansion of $\sin(x)$ for $x \approx \pi/3$ which gives: $\sin(\frac{\pi}{3}+\delta) = \frac{\sqrt{3}}{2}(1+O(\delta))$.

Bounding I_2. The choice of $\delta = \Theta(1/\sqrt{d})$ allows to consider that the $o(1)$-term in $w(\theta)$ can be neglected.

$$\begin{aligned} I_2 &= \int_{\pi/3+\delta}^{\pi/2} f(\theta)p^*(\theta)\ \mathrm{d}\theta \\ &\leqslant \int_{\pi/3+\delta}^{\pi/2} \sin(\theta)^d\Big(1-(1-\exp\Big[-3\ln(t)\tan^2(\theta/2)\Big])^t\Big)\ \mathrm{d}\theta \\ &= \int_{\pi/3+\delta}^{\pi/2} \sin(\theta)^d t.\exp\Big[-3t.\tan^2(\theta/2)(1+o(1))\Big]\ \mathrm{d}\theta \\ &\leqslant \int_{\pi/3}^{\pi/2} 2^{d.\log_2(\sin\theta) - (3\tan^2(\theta/2)-1)\log_2(t) + o(d)}\ \mathrm{d}\theta \\ &\leqslant \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})} \Big\{2^{d.\log_2(\sin\theta) - (3\tan^2(\theta/2)-1)\log_2(t) + o(d)}\Big\} \end{aligned}$$

Thus we obtain

$$\log_2(I_2) \leqslant \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})} \Big\{d.\log_2(\sin\theta) - (3\tan^2(\theta/2)-1)\log_2(t)\Big\} + o(d)$$

Bounding $p_2^* = I_1 + I_2$. We denote $c_{p_2^*} = \frac{\log_2(p_2^*)}{d}$ the exponent of p_2^*. Combining previous bounds, with the assumption that $c_n \geqslant \gamma_1$:

$$\begin{aligned} c_{p_2^*} &\leqslant \max\Big\{-\gamma_1, \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\{\log_2(\sin\theta) - c_t\Big(3\tan^2(\theta/2)-1\Big)\}\Big\} + o(1) \\ &\leqslant -\alpha c_n + o(1) \end{aligned}$$

By setting:

$$\alpha = -\frac{1}{c_n}\Big[\max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\Big\{\log_2(\sin\theta) - c_t\Big(3\tan^2(\theta/2)-1\Big)\Big\}\Big]$$

Indeed, we have $c_n \geqslant \alpha$ by assumption we did and $\alpha < 1$ by definition. So we also have $-\gamma_1 \leqslant -\alpha c_n + o(1)$. Thus $p_2^* \leqslant n^{-\alpha}$, that achieves the proof. □

Balancing the parameters and overall complexity

Theorem 6.7. *The NV-sieve with hypercone LSH heuristically solves SVP in time and space* $2^{0.2972d+o(d)}$ *using the parameters* $k = 0.3727\sqrt{d} + o(d)$ *and* $t = 2^{0.0896d+o(d)}$.

Proof. By the two previous lemmas, we have

1. The average probability that a reducing vector $\boldsymbol{w}$ collides with $\boldsymbol{v}$ in at least one of the t hash tables is at least constant in d:
$$p_1^* = \Pr_{h_{i,j}\in\mathcal{H}}\left[\boldsymbol{v}, \boldsymbol{w} \text{ collide } |\theta(\boldsymbol{v}, \boldsymbol{w}) \leqslant \frac{\pi}{3}\right] \geqslant 1 - \epsilon$$

2. The average probability that a non-reducing vector $\boldsymbol{w}$ collides with $\boldsymbol{v}$ in at least one of the t hash tables is exponentially small:
$$p_2^* = \Pr_{h_{i,j}\in\mathcal{H}}\left[\boldsymbol{v}, \boldsymbol{w} \text{ collide } |\theta(\boldsymbol{v}, \boldsymbol{w}) > \frac{\pi}{3}\right] \leqslant n^{-0.5681+o(1)}$$

3. The number of hash tables grows as $t = n^{0.4319+o(1)}$

The complexities of the algorithm are given by:

- time (hashing): $\widetilde{O}(n.t) = 2^{(c_n+c_t)d+o(d)}$
- time (searching): $\widetilde{O}(n^2.p_2^*) = 2^{c_n+(1-\alpha)c_n.d+o(d)}$
- time (overall): $2^{(c_n+\max\{c_t,(1-\alpha)c_n\})d+o(d)}$
- space: $\widetilde{O}(n.t) = 2^{(c_n+c_t)d+o(d)}$

We write the time and space complexities as $2^{c_{time}d+o(d)}$ and $2^{c_{space}d+o(d)}$. We have:

$$c_{time} = c_n + \max\{c_t, (1-\alpha)c_n\}, \qquad c_{space} = c_n + c_t$$

We search to minimize the overall time complexity, thus we balance the hashing and searching complexities, i.e. we solve $(1-\alpha)\gamma_1 = c_t$ numerically for c_t.

We obtain $c_t \approx 0.0896$. We denote θ^* the angle in the expression of α. This leads to:

$$\theta^* \approx 0.425, \qquad \alpha \approx 0.568, \qquad c_{time} \approx 0.297, \qquad c_{space} \approx 0.297.$$

With parameters $t = 2^{0.0896}$ and $k = \frac{6.\ln(t)}{\sqrt{d}}$, both time and space complexities are thus balanced at $2^{0.297d+o(d)}$. □