Lattice sieving via quantum random walks

André Chailloux and Johanna Loyer

Inria de Paris, EPI COSMIQ,
andre.chailloux@inria.fr ‖ johanna.loyer@inria.fr

Abstract. Lattice-based cryptography is one of the leading proposals for post-quantum cryptography. The Shortest Vector Problem (SVP) is arguably the most important problem for the cryptanalysis of lattice-based cryptography, and many lattice-based schemes have security claims based on its hardness. The best quantum algorithm for the SVP is due to Laarhoven [Laa16] and runs in (heuristic) time $2^{0.2653d+o(d)}$. In this article, we present an improvement over Laarhoven's result and present an algorithm that has a (heuristic) running time of $2^{0.2570d+o(d)}$ where d is the lattice dimension. We also present time-memory trade-offs where we quantify the amount of quantum memory and quantum random access memory of our algorithm. The core idea is to replace Grover's algorithm used in [Laa16] in a key part of the sieving algorithm by a quantum random walk in which we add a layer of local sensitive filtering.

1 Introduction

Lattice-based cryptography is one of the most appealing modern public-key cryptography. It has worst case to average case reductions [Ajt96], efficient schemes and allows more advanced primitives such as fully homomorphic encryption [Gen09]. Another important aspect is that lattice based problems are believed to be hard even for quantum computers. Lattice-based cryptography is therefore at the forefront of post-quantum cryptography, especially in the NIST post-quantum standardization process. It is therefore very important to put a large effort on quantum cryptanalysis and to understand the quantum hardness of lattice problems in order to increase our trust in these post-quantum solutions.

For a given lattice $\mathcal{L}$, the Shortest Vector Problem (SVP) asks to find a short vector of this lattice. Solving the SVP is arguably the most important problem for the cryptanalysis of lattice-based cryptography. Additionally to its own importance, it is used as a subroutine in the BKZ algorithm, which is often the best attack on lattice-based schemes. There are two main families of algorithms for SVP: enumeration algorithms which are asymptotically slow but have small memory requirements, and sieving algorithm which have the best asymptotic complexities but have large memory requirements. For finding very small vectors, which is required by the BKZ algorithm, sieving algorithms are currently the most efficient algorithms despite their large memory requirements.

Indeed, in the SVP challenge, the 10 top performances are done by sieving algorithms and the current record solves SVP for $d = 180$[1].

For a lattice $\mathcal{L}$ of dimension d, sieving algorithms solve SVP classically in time $2^{0.292d+o(d)}$ (with a heuristic analysis) using the local filtering technique introduced in [BDGL16]. Laarhoven presented a quantum equivalent of this algorithm that runs in time $2^{0.265d+o(d)}$ while using as much space as in the classical setting, namely $2^{0.208d+o(d)}$. The BKZ algorithm is the most efficient known attack against all lattice-based schemes which were chosen at the third round of NIST standardization process[2]. These two exponents are used for determining the number of bits of security in all these schemes hence improving the time exponent for SVP has direct implications on the security claims of these schemes.

Related work. Heuristic sieving algorithms were first introduced by Nguyen and Vidick [NV08] that presented an algorithm running in time $2^{0.415d+o(d)}$ and using $2^{0.2075d}$ memory. A more efficient sieve in practice but with the same asymptotic running time was presented in [MV10]. Then, there has been improvements by considering k-sieve algorithms [WLTB11, ZPH14, Laa16]. Also, several works showed how to use nearest neighbor search to improve sieving algorithms [LdW15, Laa15, BL16]. The best algorithm [BDGL16] runs in time $2^{0.292d+o(d)}$ and uses locality-sensitive filtering.

In the quantum setting, quantum analogues of the main algorithms for sieving were studied [LMvdP15, Laa16]. The best algorithm runs in time $2^{0.265d+o(d)}$ and is the quantum analogue of [BDGL16]. There has been two more recent works on quantum sieving algorithms. First, quantum variants of the k-sieve were studied in [KMPM19], giving interesting time-space trade-off and a recent article [AGPS20] studied more practical speedups of these quantum algorithms, *i.e.* when do these gains in the exponent actually translate to quantum speedups.

Contributions. In this article, we study and improve the asymptotic complexity of quantum sieving algorithm for SVP. This is the first improvement on the asymptotic running time of quantum sieving algorithms since the work of Laarhoven [Laa16][3].

It is not *a priori* clear how to use quantum random walks to adapt the algorithm from [BDGL16]. This algorithm is divided into a pre-processing phase and a query phase. In this query phase, we have several points that are in a filter F, which means here that there are close to a specific point. We are then given a new point $\vec{v}$ and we want to know whether there exists a point $\vec{w} \in F$ such

[1] The SVP challenge can be accessed here https://www.latticechallenge.org/svp-challenge.

[2] At this stage, there are 3 encryption schemes / key encapsulation mechanisms: KYBER, NTRU and SABER as well as two signature schemes: DILITHIUM and FALCON.

[3] We are talking here only about the asymptotic running time, there are other metrics of interest that have been covered in [KMPM19, AGPS20] where there were some improvements.

that $\|\vec{v} \pm \vec{w}\|$ is smaller than $\min\{\|\vec{v}\|, \|\vec{w}\|\}$[4]. Then we do not know how to do better here than Grover's algorithm, which takes time $\sqrt{|F|}$. On the other hand, if instead of this query framework, we start from a filter F and we want to find all the pairs $\vec{v}, \vec{w}$, then we can apply a quantum random walk.

Even within this framework, there are many ways of constructing quantum random walks and most of them do not give speedups over [Laa16]. What we show is that by adding proper additional information in the vertices of the random walk, in particular by adding another layer of filters within the vertices of the graph on which we perform the quantum walk, we can actually get some improvement over Grover's algorithm and achieve our speedups.

We now state our results. We present here not only the running time but also the amount of classical memory, quantum memory and quantum RAM (QRAM) operations required.

Our main theorem is an improvement of the best asymptotic quantum heuristic running time for the SVP bringing down the asymptotic running time from $2^{0.2653d+o(d)}$ to $2^{0.2570d+o(d)}$. Our results are in the QRAM model where QRAM operations can be done efficiently. Notice that Laarhoven's result is in this model so our result are directly comparable to his.

Theorem 1. *There exists a quantum algorithm using quantum random walks that solves the SVP on dimension d which heuristically solves SVP on dimension d in time $2^{0.2570d+o(d)}$, uses QRAM of maximum size $2^{0.0767d}$, a quantum memory of size $2^{0.0495d}$ and a classical memory of size $\mathsf{poly}(d) \cdot 2^{0.2075d}$.*

We can see that additionally to improving the best asymptotic running time, this algorithm uses much less quantum resources (both quantum memory and quantum RAM) than its running time which makes it fairly practical. We also present two trade-offs: a quantum memory-time trade-off and a QRAM-time trade-off. For a fixed amount of quantum memory, our algorithm performs as follows.

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

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

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

With this theorem, we obtain for $M = 0$ the quantum running time of Laarhoven's quantum algorithm and, for $M = 0.0495$, the result of Theorem 1.

We now present our second trade-off theorem where we fix the amount of QRAM.

[4] We remain a bit imprecise and informal here as we haven't properly described sieving algorithms yet.

Theorem 3 (Trade-off for fixed QRAM). *There exists a quantum algorithm using quantum random walks that solves SVP on dimension d which for a parameter $M' \in [0, 0.0767]$ heuristically runs in time $2^{\tau_{M'}d+o(d)}$, uses QRAM of maximum size $2^{\gamma_{M'}d}$, a quantum memory of size $2^{\mu_{M'}d}$ and uses a classical memory of size $\mathsf{poly}(d) \cdot 2^{0.2075d}$ where*

$$\tau_{M'} \in 0.2925 - 0.4647M' - [0; 6 \cdot 10^{-4}] \quad ; \quad \gamma_{M'} = M'$$

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

With this theorem, we obtain for $M' = 0$, the best classical exponent of [BDGL16] (we can actually show the algorithm uses no quantum resources in this case). For $M' = 0.0577$, we retrieve Laarhoven's quantum exponent and for $M' = 0.0767$, we get Theorem 1.

This theorem can also be helpful if we want to optimize other performance measures. For example, it has been argued that having efficient QRAM operations is too strong and that performing a QRAM operation should require time at least $r^{1/3}$ where r is the number of QRAM registers. This means we want to minimize the quantity $\lambda = \tau_{M'} + \frac{1}{3}\max\{\gamma_{M'}, \mu_{M'}\}$[5]. We also show some mild improvements in this metric: the previous best known bound was $\lambda = 0.2849$ [Laa16, AGPS20] while using our theorem, we can retrieve the previous results by taking $M' = 0.0577$ and slightly improve it by taking $M' = 0.0767$ to obtain $\lambda = 0.2824$.

Organisation of the paper. In section 2, we present preliminaries on Quantum computing. In Section 3, we then present sieving algorithm, as well as useful statements on lattices. In Section 4, we present the framework we use for sieving algorithm that we use and perform a first study of its time complexity. Next, we present in Section 5 the quantum walk that will allow our time improvements and in Section 5.4 the numerical values we achieve and the space-time trade-offs. We perform a final discussion in Section 8 and talk about parallelization of our algorithm as well as possible improvements.

2 Quantum computing preliminaries

2.1 Quantum circuits.

We consider here quantum circuits consisting of 1 and 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 time 1[6]. We use the textbook gate model where the running time of a quantum circuit is just the number of gates used. The

[5] We want to minimize $2^{\tau_{M'}d} \cdot \left(2^{\gamma_{M'}d} + 2^{\mu_{M'}d}\right)^{1/3}$ which asymptotically is equivalent to minimizing $\tau_{M'} + \frac{1}{3}\max\{\gamma_{M'}, \mu_{M'}\}$.

[6] We are only interested in 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 (see [NC00], Appendix 3).

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 also call this quantity quantum memory.

When we will know much more precisely how 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 quantum figures of merit, such as quantum memory or Quantum Random Access Memory usage.

2.2 Quantum Random Access Memory.

Quantum Random Access Memory (denoted hereafter QRAM) is a type of quantum operation which is not captured by the circuit model. Consider N registers $x_1, \ldots, x_N \in \{0,1\}^d$ stored in memory. A QRAM operation consists of applying the following unitary

$$U_{\mathrm{QRAM}} : |i\rangle|y\rangle \to |i\rangle|x_i \oplus y\rangle.$$

We say that we are in the QRAM model if the above unitary can be constructed efficiently, typically in time $O(d + \log(N))$. We can distinguish two different types of QRAM: QRACM, where the registers $x_1, \ldots, x_N$ are stored in some classical memory and QRAQM where the unitary U_{QRAM} can be applied on fully quantum registers. The former being of course easier to achieve than the latter.

QRAM operations are theoretically allowed by the laws of quantum mechanics and there are some proposals for building efficiently QRAM operations, such as [GLM08], even though its robustness has been challenged in [AGJO$^+$15]. The truth is that it very premature to know whether QRAM operations will be efficiently available in quantum computers. This would definitely require a major hardware breakthrough but as does quantum computing in general.

While our results are mainly in the QRAM model, we will also discuss other metrics where the cost of a QRAM operation is not logarithmic in N but has a cost of N^x for a constant x.

2.3 Grover algorithm.

One formulation of Grover's search problem [Gro96] is the following. We are given a list of data $x_1, ..., x_r$, with $x_i \in E$. Given a function $f : E \to \{0,1\}$, the goal is to find an i such that $f(x_i) = 1$, and to output "no solution" if there are no such i. Let $\mathrm{Sol} = \{i \in [r] : f(x_i) = 1\}$.

Classically, we cannot solve this problem with a better average complexity than $\Theta(\frac{r}{\mathrm{Sol}})$ queries, which is done by examining random x_i one by one until we find one whose image is 1 through f. Quantum computing allows a better complexity. Grover's algorithm solves this search problem in $O\left(\sqrt{\frac{r}{\mathrm{Sol}}}\right)$ queries to f. Applying Grover's algorithm this way requires efficient QRAM access to the data $x_1, \ldots, x_r$.

2.4 Quantum random walks.

We present here briefly quantum random walks (QRW). There are several variants of QRW and we will use the MNRS framework, first presented in [MNRS11].

We start from a 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 $\forall x \in V, (x, x) \notin E$ and the graph will be undirected so $(x, y) \in E \Rightarrow (y, x) \in E$. Let also $N(x) = \{y : (x, y) \in E\}$ be the set of neighbors of x. We have a set $M \subseteq V$ of marked elements and the goal of a QRW is to find $v \in M$.

Let $\varepsilon = \frac{|M|}{|V|}$ be the fraction of marked vertices and let δ be the spectral gap of G[7]. For any vertex x, we define $|p_x\rangle = \sum_{y \in N(x)} \frac{1}{\sqrt{|N(x)|}} |y\rangle$. We also define $|U\rangle = \frac{1}{\sqrt{|V|}} \sum_{x \in V} |x\rangle|p_x\rangle$. We now define the following quantities:

- SETUP cost $\mathcal{S}$: the SETUP cost $\mathcal{S}$ is the cost of constructing $|U\rangle$.
- UPDATE cost $\mathcal{U}$: here, it is the cost of constructing the unitary

$$U_{\text{UPDATE}} : |x\rangle|0\rangle \to |x\rangle|p_x\rangle.$$

- CHECK cost $\mathcal{C}$: it is the cost of computing the function $f_{\text{CHECK}} : V \to \{0, 1\}$ where $f_{\text{CHECK}}(v) = 1 \Leftrightarrow v \in M$.

Proposition 1. *[MNRS11] There exists a quantum random walk algorithm that finds a marked element $v \in M$ in time*

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

In order to compute the update cost, we can actually compute the classical running time of going from one vertex to another *i.e.* starting from a vertex x and constructing a vertex y for a random neighbor $y \in N(x)$. Then, we can use this procedure in quantum superposition to construct the unitary U_{UPDATE}. We refer to [Amb07, MNRS11, dW19] for more details on these QRW.

Quantum random walks on the Johnson graph. A very standard graph on which we can perform QRW is the Johnson graph $J(n, r)$. Each vertex v consists of r different (unordered) points $x_1, \ldots, x_r \in [n]$ as well as some additional data $D(v)$ that depends on the QRW we want to perform.

$v = (x_1, \ldots, x_r, D(v))$ and $v' = (x'_1, \ldots, x'_r, D(v'))$ form an edge in $J(n, r)$ iff. we can go from $(x_1, \ldots, x_r)$ to $(x'_1, \ldots, x'_r)$ by removing exactly one value and then adding one value. The Johnson graph $J(n, r)$ has spectral gap $\delta = \frac{n}{r(n-r)} \approx \frac{1}{r}$ when $r \ll n$ [dW19].

The additional data $D(v)$ here is used to reduce the checking time $\mathcal{C}$ with the drawback that it will increase the update time $\mathcal{U}$. Johnson graphs were often used, for example when trying to solve the element distinctness problem [Amb07], but also for the subset-sum problem [BJLM13, HM18, BBSS20] or for code-based problems [KT17].

[7] For a regular graph, if $\lambda_1 > \cdots > \lambda_{|V|}$ are the eigenvalues of the normalized adjacency matrix of G, then $\delta = \lambda_1 - \max_{i=2\ldots n} |\lambda_i|$.

Quantum data structures. A time analysis of quantum random 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 QRAQM that allows in particular insertion and deletion in $O(\log(n))$ time where n is the database size while maintaining this database in quantum superposition. Another paper on quantum algorithm for the subset problem using quantum random walks [BJLM13] 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 much QRAQM registers as the number of registers to store the whole database and this running time holds only in the QRAM model. In our work, we will use such a quantum data structure and refer to the above two papers for explicit details on how to construct such quantum data structures.

3 Lattice preliminaries

Notations. The norm $\|\cdot\|$ we use throughout this paper is the Euclidian norm, so for a vector $\vec{v} = (v_1, \ldots, v_d) \in \mathbb{R}^d$, $\|\vec{v}\| = \sqrt{\sum_{i=1}^{d} v_i^2}$. The inner product of $\vec{v} = (v_1, \ldots, v_d)$ and $\vec{w} = (w_1, \ldots, w_d)$ is $\langle \vec{v}, \vec{w} \rangle := \sum_{i=1}^{d} v_i w_i$. The non-oriented angle between $\vec{v}$ and $\vec{w}$ is denoted $\theta(\vec{v}, \vec{w}) := \arccos\left(\frac{\langle \vec{v}, \vec{w} \rangle}{\|\vec{v}\|\|\vec{w}\|}\right)$. We denote the d-dimensional sphere of radius R by $\mathcal{S}_R^{d-1} := \{\vec{v} \in \mathbb{R}^d : \|\vec{v}\| = R\}$, and $S^{d-1} := S_1^{d-1}$. Throughout the paper, for a known integer d, we will write $N := (\sqrt{4/3})^d$.

Lattices. The d-dimensional lattice $\mathcal{L} \subset \mathbb{R}^m$ generated by the basis $B = (b_1, ..., b_n)$ with $\forall i, b_i \in \mathbb{R}^m$ is the set of all integer linear combinations of its basis vectors: $\mathcal{L}(B) = \left\{ \sum_{i=1}^{d} \lambda_i \; b_i, \; \lambda_i \in \mathbb{Z} \right\}$.

Shortest Vector Problem. Given a basis of a lattice $\mathcal{L}$, the Shortest Vector Problem (SVP) asks to find a non-zero vector in $\mathcal{L}$ of minimal norm. SVP is known to be NP-hard [Ajt98]. This problem and its derivatives (SIS, LWE) have been used in several public-key cryptosystems, specifically as candidate for quantum-resistant cryptography [DKL$^+$19, FHK$^+$19, CDH$^+$19]. Thereby, one of the most important ways to know their security and choose parameters is to estimate the computational hardness of the best SVP-solving algorithms.

Sieving algorithms.

SVP solving methods. The algorithm LLL [LLL82] returns a reduced basis of a lattice in a polynomial time. However it is not sufficient to solve SVP. All the fastest known algorithms to solve SVP run in exponential time. A first method is enumeration [Kan83], that solves deterministically SVP using low space but in super-exponential time in the lattice dimension d.

Another method, which will interest us in this article, is lattice sieving [NV08, MV10]. They are heuristic algorithms that probably solve SVP in time and space $2^{\Omega(d)}$. To this day, the best complexity for sieving in the QRAM model is obtained by quantum hypercone LSF [Laa16] in $2^{0.2653d+o(d)}$ time and $2^{0.2075d+o(d)}$ space. Another algorithm [KMPM19] uses k-lists to solve SVP in $2^{0.2989d+o(d)}$ time and $2^{0.1395d+o(d)}$ space.

The NV-sieve. The NV-sieve [NV08] is a heuristic algorithm. It starts with a list of lattice vectors, that we can consider of norm at most 1 by normalization. Given this list and a constant $\gamma < 1$, the NV-sieve returns a list of lattice vectors of norm at most γ. It iteratively builds lists of shorter lattice vectors by applying a sieve. This sieve step consists in computing all the sums (plus and minus) of two list vectors, and fills the output list with those which have norm at most γ. For γ tending to 1, two vectors form a reducing pair - *i.e.* their sum is of norm at most γ - iff. they are of angle at most $\pi/3$. The first list of lattice vectors can be sampled with Klein's algorithm [Kle00] for example. A list size of $N^{1+o(1)} = (\sqrt{4/3})^{d+o(d)}$ suffices to have about one reducing vector in the list for each list vector, as stated in [NV08]. Because of the norms of the list vectors reduces with a factor by $\gamma < 1$ at each application of the algorithm, the output list will hopefully contain a non-zero shortest lattice vector after a polynomial number of application of the NV-sieve.

NNS and application to lattice sieving. A logic improvement of this algorithm is to use Neighbor Nearest Search (NNS) [IM98] 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. Used in the NV-sieve, the preprocessing partitions the input list in several buckets of lattice points, each bucket being associated with a hash function. The algorithm will only sum vectors from a same bucket, which are near to each other, instead of trying all pairs of vectors.

Locality-sensitive hashing (LSH). A method to solve NNS is locality-sensitive hashing (LSH) [IM98]. An LSH function is a hash-function that have high probability to collide for two elements if they are close, and a low one if they are far. Several categories of LSH functions exists: hyperplane LSH [Cha02], hypercone or spherical LSH [AINR14, AR15] and cross-polytope LSH [TT07].

Locality-sensitive filtering (LSF).

Locality-sensitive filtering (LSF). More recently, [BDGL16] improved NNS solving by introducing locality-sensitive filtering (LSF). LSF functions, called filters, map a vector $\vec{v}$ to a boolean value: 1 if $\vec{v}$ survives the filter, and 0 otherwise. They act similarly to LSH but only few vectors survive the filter.

These filters are instantiated by hypercone filters, characterized by a vector $\vec{s}$ and an angle $\alpha \in [0, \pi/2]$. For a filter f of center $\vec{s}$ and angle α, the vector $\vec{v}$ survives f iff $\theta(\vec{v}, \vec{s}) \leqslant \alpha$. In this case, the filter f is said relevant for $\vec{v}$. The

set of the vectors from a list that survives a filter f is called its bucket, and is denoted $f_\alpha(\vec{s})$. More formally, we define the spherical cap of center $\vec{v}$ and angle α as follows:

$$\mathcal{H}_{\vec{v},\alpha} := \{\vec{x} \in \mathcal{S}^{d-1} \mid \theta(\vec{x}, \vec{v}) \leqslant \alpha\}$$

and $\vec{v}$ survives the filter $f_\alpha(\vec{s})$ iff. $\vec{v} \in \mathcal{H}_{\vec{s},\alpha}$.

Proposition 2. *[MV10] For an angle $\alpha \in [0, \pi/2]$ and $\vec{v} \in \mathcal{S}^{d-1}$, the ratio of the volume of a spherical cap $\mathcal{H}_{\vec{v},\alpha}$ to the volume of the sphere $\mathcal{S}^{d-1}$ is*

$$\mathcal{V}_d(\alpha) := \mathsf{poly}(d) \cdot \sin^d(\alpha).$$

Proposition 3. *[BDGL16] For an angle $\alpha \in [0, \pi/2]$ and two vectors $\vec{v}, \vec{w} \in \mathcal{S}^{d-1}$ such that $\theta(\vec{v}, \vec{w}) = \theta$, the ratio of the volume of a wedge $\mathcal{H}_{\vec{v},\alpha} \cap \mathcal{H}_{\vec{w},\alpha}$ to the volume of the sphere $\mathcal{S}^{d-1}$ is*

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

Random product codes (RPC). LSF method from [BDGL16] uses a decoding oracle that returns relevant filters for a given vector. A random-product code (RPC) that admits a fast list-decoding algorithm is sampled over the sphere $\mathcal{S}^{d-1}$. The filters are determined by its code words, and the oracle is its decoding algorithm.

We assume $d = m \cdot b$, for $m = O(\mathsf{polylog}(d))$ and a block size b. The vectors in $\mathbb{R}^d$ will be identified with tuples of m vectors in $\mathbb{R}^b$. A random product code C of parameters $[d, m, B]$ on subsets of $\mathbb{R}^d$ and of size 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 $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$, so that codewords are points of the sphere S^{d-1}. We can have a full description of C by storing mB points corresponding to the codewords of $C_1, \ldots, C_m$ and by storing the rotation Q. When the context is clear, C will correspond to the description of the code or to the set of codewords. Random product codes can be easily decoded in some parameter range:

Proposition 4 ([BDGL16]). *Let C be a random product code of parameters $[d, m, B]$ with $m = \log(d)$ and $B^m = N^{O(1)}$. For any $\vec{v} \in S^{d-1}$ and $\alpha \in [0, \pi/2]$, one can compute $\mathcal{H}_{\vec{v},\alpha} \cap C$ in time $N^{o(1)} \cdot |\mathcal{H}_{\vec{v},\alpha} \cap C|$.*

We can now present the NV-sieve with LSF.

The NV-sieve with LSF. Let $\alpha \in [\pi/3, \pi/2]$ be an angle. The NV-sieve with LSF [BDGL16] takes as input a list of lattice vectors lying on $\mathcal{S}^{d-1}$ and a constant $\gamma < 1$. This algorithm runs in two phases. First, during the processing, it samples a random-product code C on the sphere, whose words give the α-filters. The

algorithm decodes half of the vectors of the list to get their nearest α-filters, and then add the vectors to the buckets associated to their α-filters.

Secondly, there is the queries phase. For each vector $\vec{v}$ from the other half of the list, the algorithm computes the α-filters of $\vec{v}$, and for each vector $\vec{w}$ having a common α-filter with $\vec{v}$, the algorithm checks whereas $\|\vec{v} - \vec{w}\| \leqslant \gamma$. If it is the case, $\vec{v} - \vec{w}$ is added to the output list. This algorithm solves SVP time[8] $2^{0.292d+o(d)}$, and in space $2^{0.208d+o(d)}$ with its space-efficient version [Laa16]. Applying a Grover search instead of testing each candidate in the filter gives the quantum NV-sieve with LSF [Laa16], which run in same space and in time $2^{0.265d+o(d)}$.

Probabilistic argument. If we consider any vector $\vec{w} \in \mathcal{S}^{d-1}$ and N^{ρ_0} random points $\vec{s}_1, \ldots, \vec{s}_{N^{\rho_0}}$ in $\mathcal{H}_{\vec{v},\alpha}$ for $\rho_0 := \frac{\mathcal{V}_d(\alpha)}{\mathcal{W}_d(\alpha,\theta)}$; then this proposition implies that there exists, with constant probability, an $i \in [N^{\rho_0}]$ such that $\vec{s}_i \in \mathcal{H}_{\vec{w},\alpha}$.

Consider a set $S = \vec{s}_1, \ldots, \vec{s}_M$ points taken from the uniform distribution on the sphere S^{d-1} and $\vec{v}$ another point randomly chosen on the sphere. Fix also an angle $\alpha \in (0, \pi/2)$. We have the following statements:

Proposition 5. $\forall i \in [M],\ \Pr[\vec{v} \in \mathcal{H}_{\vec{s}_i,\alpha}] = \Pr[\vec{s}_i \in \mathcal{H}_{\vec{v},\alpha}] = \mathcal{V}_d(\alpha)$.

Proof. Immediate by definition of $\mathcal{V}_d(\alpha)$ considering that both $\vec{v}$ and $\vec{s}_i$ are uniform random points on the sphere. □

From the above proposition, we immediately have that $\mathbb{E}[|S \cap \mathcal{H}_{\vec{v},\alpha}|] = M\mathcal{V}_d(\alpha)$. We now present a standard concentration bound for this quantity.

Proposition 6. *Assume we have $M\mathcal{V}_d(\alpha) = N^x$ with $x > 0$ an absolute constant. Then*

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

Proof. As before, let X_i be the random variable which is equal to 1 if $\vec{s}_i \in \mathcal{H}_{\vec{v},\alpha}$ 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 $|S \cap \mathcal{H}_{\vec{v},\alpha}|$. A direct application of the multiplicative Chernoff bound gives

$$\Pr[Y \geq 2N^x] \leq e^{-\frac{N^x}{3}}$$

which is the desired result. □

4 General framework for sieving algorithms using LSF

We present here a general framework for sieving algorithms using LSF. We present here one sieving step where we start from a list L of $N' = N^{1+o(1)}$ lattice vectors of norm 1 and output N' lattice vectors of norm $\gamma < 1$. Sieving algorithms for SVP then consists of applying this subroutine $\mathsf{poly}(d)$ times (where

[8] We consider here the values of the NV-sieve, which have asymptotically better space requirements even though this sieve is less efficient in practice.

we renormalize the vectors at each step) to find at the end a small vector. We can actually take γ very close to 1 at each iteration, and we refer for example to [NV08] for more details. This framework will encompass the best classical and quantum sieving algorithms.

Algorithm 1 Sieving algorithms using LSF with parameter c

Input: a list L of $N' = N^{1+o(1)}$ lattice vectors of norm 1, a constant $\gamma < 1$ and parameter $c \in (0, 1)$.
Output: a list L' of N' lattice vectors of norm at most γ.
Algorithm:

$L' := \{\}$ (empty list)
while $|L'| \leq N'$ **do**
  Sample a random product code C of parameter $[d, \log(d), N^{\frac{1-c}{\log(d)}}]$. Let $\vec{s}_1, \ldots, \vec{s}_{N^{1-c}}$ be the code points of C and let $\alpha \in [\pi/3, \pi/2]$ st. $\mathcal{V}_d(\alpha) = \frac{1}{N^{1-c}}$.
  for $\vec{v}$ in L **do**
    Add $\vec{v}$ to its α-filter's buckets $f_\alpha(\vec{s}_i)$
  for each $i \in [N^{1-c}]$ **do**
    $S \leftarrow$ **FindAllSolutions**$(f_\alpha(\vec{s}_i), \gamma)$
    $L' := L' \cup S$
return L'

The **FindAllSolutions**$(f_\alpha(\vec{s}_i), \gamma)$ subroutine starts from a list of vectors $\vec{x}_1, \ldots, \vec{x}_{N^c} \in f_\alpha(\vec{s}_i)$ and outputs all vectors of the form $\vec{x}_i \pm \vec{x}_j$ (with $i \neq j$) of norm less than γ. We want to find asymptotically all the solutions and not strictly all of them. Let's say here we want to output half of them. Sometimes, there are no solutions so the algorithm outputs an empty list.

4.1 Analysis of the above algorithm

Heuristics and simplifying assumptions. We first present the heuristic arguments and simplifying assumptions we use for our analysis.

1. The input lattice points behave like random points on the sphere $\mathcal{S}^{d-1}$. The relevance of this heuristic has been studied and confirmed in a few papers starting from the initial NV-sieve [NV08].
2. The code points of C behave like random points of the sphere $\mathcal{S}^{d-1}$. This was argued in [BDGL16], see for instance Lemma 5.1 and Appendix C therein.
3. We assume that a random point in $f_\alpha(\vec{s}_i)$ is on the border of the filter, *i.e.* that it can be written $\vec{x} = \cos(\alpha)\vec{s}_i + \sin(\alpha)\vec{y}$ with $\vec{y} \perp \vec{s}_i$ and of norm 1. As we argue below, this will be approximately true with very high probability.

In order to argue point 3, notice that for any angle $\alpha \in (\pi/4, \pi/2)$ and $\varepsilon > 0$, we have $\mathcal{V}_d(\alpha) \gg \mathcal{V}_d(\alpha - \epsilon)$. Indeed, for an angle $\epsilon > 0$, $\mathcal{V}_d(\alpha - \epsilon) = \sin^d(\alpha - \epsilon) =$

$\mathcal{V}_d(\alpha) \cdot (\epsilon')^d$ with $\epsilon' = \cos\epsilon - \frac{\sin\epsilon\cos\alpha}{\sin\alpha} < 1$ for $\alpha > \epsilon$. 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 with very high probability, points in $f_\alpha(\vec{s}_i)$ lie at the border of the cap and hence justifies point 3.

Completion. We start from a list L of N' points. The heuristic states that each point in L is modeled as a random point on the sphere S^{d-1} so each pair of points $\vec{x}, \vec{x}' \in L$ reduces with probability $\mathcal{V}_d(\pi/3) = \frac{1}{N}$. Since there are $\frac{N'(N'-1)}{2}$ pairs of points in L, we have on average $\frac{N'(N'-1)}{2N}$ pairs in L that are reducible. We can take for example $N' = 6N$ to ensure that there are on average $\approx 3N'$ pairs. Therefore, each time we find a random reducible pair, with probability at least $\frac{3N'-|L'|}{3N'} \geq 2/3$, it wasn't already in the list L'.

Throughout the rest of the paper

Time analysis.

Condition of reduction of vectors. Consider two random vectors $\vec{x}_0$ and $\vec{x}_1$ that are in the same α-filter's bucket of center $\vec{s}$. We write

$$\vec{x}_0 = \cos(\alpha)\vec{s} + \sin(\alpha)\vec{y}_0 \tag{1}$$
$$\vec{x}_1 = \cos(\alpha)\vec{s} + \sin(\alpha)\vec{y}_1 \tag{2}$$

with $\vec{y}_0, \vec{y}_1$ of norm 1 both orthogonal to $\vec{s}$. The vectors $\vec{y}_0$ and $\vec{y}_1$ are called residual vectors and if $\vec{x}_0$ and $\vec{x}_1$ are random vectors of $f_\alpha(\vec{s})$ then $\vec{y}_0, \vec{y}_1$ are random vectors in the sphere of dimension $d-2$ of vectors of norm 1 orthogonal to $\vec{s}$.

Proposition 7. *Using the notations just above, we have*

$$\theta(\vec{y}_0, \vec{y}_1) \leqslant 2\arcsin\big(\frac{1}{2\sin(\alpha)}\big) \iff \theta(\vec{x}_0, \vec{x}_1) \leqslant \frac{\pi}{3}.$$

Proof. We denote for simplicity $\theta_y := \theta(\vec{y}_0, \vec{y}_1)$. By subtracting Equation 2 from Equation 1 and then by squaring, we have

$$\begin{aligned}
\|\vec{x}_0 - \vec{x}_1\|^2 \leqslant 1 &\Leftrightarrow \sin^2(\alpha)\|\vec{y}_0 - \vec{y}_1\|^2 \leqslant 1 \\
&\Leftrightarrow \sin^2(\alpha)(2 - 2\cos(\theta_y)) \leqslant 1 \\
&\Leftrightarrow \cos(\theta_y) \geqslant 1 - \frac{1}{2\sin^2(\alpha)} \\
&\Leftrightarrow \theta_y \leqslant \arccos\big(1 - \frac{1}{2\sin^2(\alpha)}\big) = 2\arcsin\big(\frac{1}{2\sin(\alpha)}\big), \text{ true for } \alpha \leqslant \pi/2.
\end{aligned}$$

□

Corollary 1. *Let $\alpha \in [\pi/3, \pi/2]$ and a random pair of vectors in a same α-filter's bucket. The probability that the pair is reducing is equal to $\mathcal{V}_{d-1}(\theta^*_\alpha)$, with*

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

Notice that we have $\mathcal{V}_{d-1}$ because we work with residual vectors (orthogonal to $\vec{s}$) but since $\mathcal{V}_d$ and $\mathcal{V}_{d-1}$ are asymptotically equivalent, we will keep writing $\mathcal{V}_d(\theta^*_\alpha)$ everywhere for simplicity. From the above corollary, we have that for an α-filter that has N^c points randomly distributed in this filter, the expected number of reducing pairs is $N^{2c} \cdot \mathcal{V}_{d-1}(\theta^*_\alpha)$.

Proposition 8. *Consider Algorithm 1 with parameter $c \in [0,1]$ and associated angle $\alpha \in [\pi/3, \pi/2]$ satisfying $\mathcal{V}_d(\alpha) = N^{-(1-c)}$. Let ζ such that $N^\zeta = N^{2c} \cdot \mathcal{V}_{d-1}(\theta^*_\alpha)$. The above algorithm runs in time $T = \mathrm{NB_{REP}} \cdot (\mathrm{INIT} + \mathrm{FAS})$ where*

$$\mathrm{NB_{REP}} = \max\{1, N^{c-\zeta+o(1)}\} \quad ; \quad \mathrm{INIT} = N^{1+o(1)} \quad ; \quad \mathrm{FAS} = N^{1-c}\mathrm{FAS}_1$$

where FAS_1 the running time of a single call to the **FindAllSolutions** *subroutine.*

Proof. We first analyze the two **for** loops. INIT is the running time of the first loop. For each point $\vec{v} \in L$, we need to compute $\mathcal{H}_{\vec{v},\alpha} \cap C$ and update the corresponding buckets $f_\alpha(\vec{s}_i)$. We have $|C| = N^{1-c}$ and we chose α such that $\mathcal{V}_d(\alpha) = N^{-(1-c)}$, so the expected value of $|\mathcal{H}_{\vec{v},\alpha} \cap C|$ is 1. For each point $\vec{v}$, we can compute $\mathcal{H}_{\vec{v},\alpha} \cap C$ in time $N^{o(1)}|\mathcal{H}_{\vec{v},\alpha}|$ using Proposition 4. From there, we can conclude that we compute the filter for the N' points in time $\mathrm{INIT} = N^{1+o(1)}$.

The second loop runs in time $\mathrm{FAS} = N^{1-c}\mathrm{FAS}_1$ by definition. After this loop, the average number of solutions found is N^ζ for each call to **FindAllSolutions** so $N^{1-c+\zeta}$ in total (notice that we can have $\zeta < 0$, which means that we can find on average much less that one solution for each call of **FindAllSolutions**). We run the **while** loop until we find N' solutions so we must repeat this process $\mathrm{NB_{REP}} = \max\{1, N^{1-(1-c+\zeta)+o(1)}\} = \max\{1, N^{c-\zeta+o(1)}\}$ times. □

This formulation of sieving algorithms is easy to analyze. Notice that the above running time depends only on c (since α can be derived from c and ζ can be derived from c, α) and on the **FindAllSolutions** subroutine. We now retrieve the best known classical and quantum sieving algorithm in this framework.

Best classical algorithm. In order to retrieve the time exponent of [BDGL16], we take $c \to 0$, which implies $\alpha \to \pi/3$. We can compute $\theta^*_{\pi/3} \approx 1.23\mathrm{rad} \approx 70.53°$ and $\zeta = -0.4094$. In this case, we have $\mathrm{FAS}_1 = O(1)$. From the above proposition, we get a total running time of $T = N^{1.4094+o(1)} = 2^{0.2925d+o(d)}$.

Best quantum algorithm. In order to retrieve the time exponent of [Laa16], we take $c = 0.2782$. This value actually corresponds to the case where $\zeta = 0$, so we have on average one solution per α-filter. For the **FindAllSolutions** subroutine, we can apply Grover's algorithm on pairs of vectors in the filter to find this

solution in time $\sqrt{N^{2c}} = N^c$ (there are N^{2c} pairs) so $\mathrm{FAS}_1 = N^c$. Putting this together, we obtain $T = N^{1+c+o(1)} = N^{1.2782+o(1)} = 2^{0.2653d+o(d)}$.

In the next section, we show how to improve the above quantum algorithm. Our main idea is to replace Grover's algorithm used in the **FindAllSolutions** subroutine with a quantum random walk. In the next section, we present the most natural quantum walk which is done over a Johnson graph and where a vertex is marked if the points of a vertex contain a reducible pair, in a similar way than for element distinctness. We then show in a later section how this random walk can be improved by relaxing the condition on marked vertices.

5 Quantum random walk for the FindAllSolutions subroutine: a first attempt

5.1 Constructing the graph

We start from an unordered list $\vec{x}_1, \ldots, \vec{x}_{N^c}$ of distinct points in a filter $f_\alpha(\vec{s})$ with α satisfying $\mathcal{V}_d(\alpha) = \frac{1}{N^{1-c}}$. Let L_x be this list of $\vec{x}_i$. For each $i \in [N^c]$, we 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 $\vec{s}$. Recall from Proposition 7 that a pair $(\vec{x}_i, \vec{x}_j)$ is reducible iff. $\theta(\vec{y}_i, \vec{y}_j) = \theta^*_\alpha = 2\arcsin(\frac{1}{2\sin(\alpha)})$. We will work only on the residual vectors $\vec{y}_i$ and present the quantum random walk that finds pairs $\vec{y}_i, \vec{y}_j$ such that $\theta(\vec{y}_i, \vec{y}_j) = \theta^*_\alpha$ more efficiently than with Grover's algorithm. Let $L_y = \vec{y}_1, \ldots, \vec{y}_{N^c}$ be the list of all residual vectors.

The quantum walk has two extra parameters $c_1 \in [0, c]$ and $c_2 \in [0, c_1]$. From these two parameters, let $\beta \in [\pi/3, \pi/2]$ st. $\mathcal{V}_d(\beta) = N^{c_2 - c_1}$ and ρ_0 st. $N^{\rho_0} = \frac{\mathcal{V}_d(\beta)}{\mathcal{W}_d(\beta, \theta^*_\alpha)}$. We start by sampling a random product code $\mathcal{C}_2$ with parameters $[(d-1), \log(d-1), N^{\frac{\rho_0 + c_1 - c_2}{\log(d-1)}}]$ which has therefore $N^{\rho_0 + c_1 - c_2} = \frac{1}{\mathcal{W}_d(\beta, \theta^*_\alpha)}$ points denoted $\vec{t}_1, \ldots, \vec{t}_{N^{\rho_0 + c_1 - c_2}}$. We perform our quantum random walk on a graph $G = (V, E)$ where each vertex $v \in V$ contains:

- An unordered list $L^v_y = \vec{y}_1, \ldots, \vec{y}_{N^{c_1}}$ of distinct points taken from L_y.
- For each $\vec{t}_i \in \mathcal{C}_2$, we store the list of elements of $J^v(\vec{t}_i) := f_\beta(\vec{t}_i) \cap L^v_y$. For each $\vec{t}_i$, we do this using a quantum data structure that stores $J^v(\vec{t}_i)$ where we can add and delete efficiently in quantum superposition. This can be done with QRAM. Notice that we have on average

$$|J^v(\vec{t}_i)| = N^{c_1} \cdot \mathcal{V}_d(\beta) = N^{c_2},$$

  and we need to store in total $|\mathcal{C}_2| \cdot N^{c_2} = N^{c_1 + \rho_0}$ 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 consists of the above vertices for all possible lists L^v_y. We have $(v, w) \in E$ if we can go from L^v_y to L^w_y by changing exactly one value. In

order words

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

This means the graph G is exactly a Johnson graph $J(N^c, N^{c_1})$ where each vertex also has some additional information as we described above. 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 $(\vec{x}_i, \vec{x}_j)$.

Condition for a vertex to be marked. We define the following subsets of vertices . We first define the set M_0 vertices for which there exists a pair of points which is reducible.

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

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 as marked vertices subsets of M_0 but for which the update can be done more efficiently, but losing only a small fraction of the marked vertices. For each $J^v(\vec{t}_i)$, we define $\widetilde{J}^v(\vec{t}_i)$ which consists of the first $2N^{c_2}$ elements of $J^v(\vec{t}_i)$[9] and if $|J^v(\vec{t}_i)| \leq 2N^{c_2}$, we have $\widetilde{J}^v(\vec{t}_i) = J^v(\vec{t}_i)$. We define the set of marked elements M as follows:

$$M := \{v \in V : \exists \vec{t} \in \mathcal{C}_2, \exists \vec{y}_i, \vec{y}_j \neq \vec{y}_i \in \widetilde{J}^v(\vec{t}), \text{ st. } \theta(\vec{y}_i, \vec{y}_j) \leq \theta_\alpha^*\}.$$

The reason for using such a condition for marked vertices is that when we will perform an update, hence removing a point $\vec{y}_{old}$ from a vertex and adding a point $\vec{y}_{new}$, we will just need to look at the points in $\widetilde{J}^v(\vec{t})$ for $\vec{t} \in f_\beta(\vec{y}_{new}) \cap \mathcal{C}_2$ which can be done faster than by looking at all the points of the vertex. If we used $J^v(\vec{t})$ instead of $\widetilde{J}^v(\vec{t})$ 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 require to bound the update for any pair of adjacent vertices[10]. Also notice that each vertex still contain the sets $J^v(\vec{t}_i)$ (from which one can easily compute $\widetilde{J}^v(\vec{t}_i)$).

5.2 Time analysis of the quantum random walk on this graph

We are now ready to analyze our quantum random walk, and compute its different parameters. Throughout our analysis, we define $K(\vec{y}_i) := f_\beta(\vec{y}_i) \cap \mathcal{C}_2$ and we have on average

$$|K(\vec{y}_i)| = N^{\rho_0 + c_1 - c_2} \cdot \mathcal{V}_d(\beta) = N^{\rho_0}.$$

[9] We consider an global ordering of elements of L_y, for example with respect to their index, and $J^v(\vec{t}_i)$ consists of the $2N^{c_2}$ elements of $J^v(\vec{t}_i)$ which are the smallest with respect to this ordering.

[10] This problem arises in several quantum random walk algorithms, for example for quantum subset-sum algorithms. One solution is to use a heuristic that essentially claims that we can use the average running time of the update cost instead of the worst case. In our case, we don't need this heuristic as we manage to bound the update cost in the worst case. We refer to [BBSS20] for an interesting discussion on the topic.

Using Proposition 6, we have for each i,

$$\Pr[|K(\vec{y}_i)|] > 2N^{\rho}] \leq e^{-\frac{N^{\rho_0}}{3}} \tag{3}$$

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

$$\Pr[\forall i \in [N^c],\ |K(\vec{y}_i)| \leq 2N^{\rho}] \geq 1 - N^c e^{-\frac{N^{\rho_0}}{3}} = 1 - o(1). \tag{4}$$

So for a fixed α-filter, we have with high probability that each $|K(\vec{y}_i)|$ is bounded by $2N^{\rho_0}$ and we assume we are in this case. The sets $K(\vec{y}_i)$ can hence be constructed in time $N^{\rho_0+o(1)}$ using the decoding procedure (Proposition 4) for $\mathcal{C}_2$.

Setup cost. In order to construct a full vertex v from a list $L_y^v = \vec{y}_1, \ldots, \vec{y}_{N^{c_1}}$, the main cost is to construct the lists $J^v(\vec{t}_i) = f_\beta(\vec{t}_i) \cap L_y^v$. To do this, we start from empty lists $J^v(\vec{t}_i)$. For each $\vec{y}_i \in L_y^v$, we construct the list $K(\vec{y}_i) = f_\beta(\vec{y}_i) \cap \mathcal{C}_2$ and for each codeword $\vec{t}_j \in K(\vec{y}_i)$, we add $\vec{y}_i$ in $J^v(\vec{t}_i)$.

This takes time $N^{c_1} \cdot N^{\rho_0+o(1)}$. We can perform a uniform superposition of the vertices by performing the above procedure in quantum superposition. This can also be done in $N^{c_1} \cdot N^{\rho_0+o(1)}$ since we use a quantum data structure that performs these insertions in $J^v(\vec{t}_i)$ efficiently. So in conclusion,

$$\mathcal{S} = N^{c_1+\rho_0+o(1)}.$$

Update cost. We show here how to go from a vertex v with associated list L_y^v to a vertex w with $L_y^w = \left(L_y^v \backslash \{\vec{y}_{old}\}\right) \cup \{\vec{y}_{new}\}$. We start from a vertex v so we also have the lists $J^v(\vec{t}_i) = f_\beta(\vec{t}_i) \cap L_y^v$.

In order to construct the lists $J^w(\vec{t}_i)$, we first construct $K(\vec{y}_{old}) = f_\beta(\vec{y}_{old}) \cap \mathcal{C}_2$ and for each $\vec{t}_i$ in this set, we remove $\vec{y}_{old}$ from $J^v(\vec{t}_i)$. Then, we construct $K(\vec{y}_{new})$ and for each $\vec{t}_i$ in this set, we add $\vec{y}_{new}$ to $J^v(\vec{t}_i)$, thus obtaining all the $J^w(\vec{t}_i)$. 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. These operations take $N^{\rho_0+o(1)}$ deletions and insertions, which can be done efficiently.

If v was marked and $\vec{y}_{old}$ 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 $\vec{y}_{new}$ doesn't make it marked. So we need to check whether there exists $\vec{y'} \neq \vec{y}_{new}$ such that

$$\exists \vec{t} \in \mathcal{C}_2, \vec{y}_{new}, \vec{y_0} \in \widetilde{J}^w(\vec{t}) \text{ and } (\vec{y}_{new}, \vec{y}_0) \text{ are reducible .}$$

If such a point $\vec{y}_0$ exists, it necessarily lies in the set $\cup_{\vec{t} \in K(\vec{y}_{new})} \widetilde{J}^v(\vec{t})$ which is of size at most $2N^{\rho} \cdot 2N^{c_2} = 4N^{\rho_0+c_2}$. We perform a Grover search on this set to determine whether there exists a $\vec{y}_0 \in \cup_{\vec{t} \in \mathcal{C}_2} \widetilde{J}^v(\vec{t})$ that reduces with $\vec{y}_{new}$, and this takes time $N^{\frac{\rho_0+c_1+o(1)}{2}}$. In conclusion, we have that the average update time is

$$\mathcal{U} = N^{\rho_0+o(1)} + N^{\frac{\rho_0+c_2+o(1)}{2}} \leq N^{\max\{\rho_0, \frac{\rho_0+c_2}{2}\}+o(1)}.$$

Checking cost. Each vertex has a bit that says whether it is marked or not so we have

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

Computing the fraction of marked vertices Epsilon. We prove here the following proposition

Proposition 9. $\varepsilon \geq \Theta\left(\min\left\{N^{2c_1}\mathcal{V}_d(\beta), 1\right\}\right).$

Proof. We consider a random vertex in the graph and lower bound the probability that it is marked. A sufficient condition for a vertex v to be marked is if it satisfies the following 2 events :

- E_1 : $\exists \vec{t} \in \mathcal{C}_2, \exists \vec{y}_i, \vec{y}_j \neq \vec{y}_i \in J^v(\vec{t}),$ st. $\theta(\vec{y}_i, \vec{y}_j) \leq \theta^*_\alpha$.
- E_2 : $\forall \vec{t} \in \mathcal{C}_2, |J^v(\vec{t})| \leq 2N^{c_2}$.

The second property implies that $\forall \vec{t} \in \mathcal{C}_2,\ J^v(\vec{t}) = \widetilde{J}^v(\vec{t})$ and in that case, the first property implies that v is marked. We now bound the probability of each event

Lemma 1. $\Pr[E_1] \geq \Theta\left(\min\left\{N^{2c_1}\mathcal{V}_d(\beta), 1\right\}\right).$

Proof. For a fixed pair $\vec{y}_i, \vec{y}_j \neq \vec{y}_i \in L^v_y$, we have $\Pr[\theta(\vec{y}_i, \vec{y}_j) \leq \theta^*_\alpha] = \mathcal{V}_d(\theta^*_\alpha)$. Since there are $\Theta(N^{2c_1})$ such pairs, if we define the event E_0 as: $\exists \vec{y}_i, \vec{y}_j \neq \vec{y}_i \in L^v_y,$ st. $\theta(\vec{y}_i, \vec{y}_j) \leq \theta^*_\alpha$, we have

$$\Pr[E_0] \geq \Theta\left(\min\left\{N^{2c_1}\mathcal{V}_d(\beta), 1\right\}\right).$$

Now we assume E_0 holds and we try to compute the probability that E_1 is true conditioned on E_0. So we assume E_0 and let $\vec{y}_i, \vec{y}_j \neq \vec{y}_i \in L^v_y,$ st. $\theta(\vec{y}_i, \vec{y}_j) \leq \theta^*_\alpha$. For each code point $\vec{t} \in \mathcal{C}_2$, we have

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

Therefore, we have

$$\Pr[\exists \vec{t} \in \mathcal{C}_2,\ \vec{y}_i, \vec{y}_j \in J^v(\vec{t})] = 1 - (1 - \mathcal{W}_d(\beta, \theta^*_\alpha))^{|C_2|}. \tag{5}$$

Since $|\mathcal{C}_2| = \frac{1}{\mathcal{W}_d(\beta,\theta^*_\alpha)}$, we can conclude

$$\Pr[E_1|E_0] \geq \Pr[\exists \vec{t} \in \mathcal{C}_2,\ \vec{y}_i, \vec{y}_j \in J^v(\vec{t})] = 1 - (1 - \mathcal{W}_d(\beta, \theta^*_\alpha))^{|C_2|} \geq \Theta(1),$$

which implies $\Pr[E_1] \geq \Pr[E_1|E_0] \cdot \Pr[E_0] \geq \Theta\left(\max\left\{N^{2c_1}\mathcal{V}_d(\beta), 1\right\}\right).$ □

Lemma 2. $\Pr[E_2] \geq 1 - |C_2|e^{-\frac{N^{c_2}}{3}}.$

Proof. For each $\vec{t} \in \mathcal{C}_2$, we have using Proposition 6 that $\Pr[|J^v(\vec{t})| \leq 2N^{c_2}] \geq 1 - e^{-\frac{N^{c_2}}{3}}$. Using a union bound, we have

$$\Pr[\forall \vec{t} \in \mathcal{C}_2, |J^v(\vec{t})| \leq 2N^{c_2}] \geq 1 - |C_2|e^{-\frac{N^{c_2}}{3}}.$$

□

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

$$\begin{aligned}\varepsilon \geq \Pr[E_1 \wedge E_2] &\geq \Pr[E_1] + \Pr[E_2] - 1 \\ &\geq \Theta\left(\max\left\{N^{2c_1}\mathcal{V}_d(\beta), 1\right\}\right) - |C_2|e^{-\frac{Nc_2}{3}} \\ &\geq \Theta\left(\max\left\{N^{2c_1}\mathcal{V}_d(\beta), 1\right\}\right)\end{aligned}$$

The last inequality comes from the fact that $|C_2|e^{-\frac{Nc_2}{3}}$ is vanishing doubly exponentially in d (N is exponential in d) so it is negligible compared to the first term and is absorbed by the $\Theta(\cdot)$. □

Computing the spectral gap Delta. We are in a $J(N^c, N^{c_1})$ Johnson graph so we have

$$\delta \approx N^{-c_1}.$$

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

$$\begin{aligned}T_1 &= \mathcal{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\,\mathcal{U} + \mathcal{C}\right) \\ &= N^{c_1+\rho_0} + \frac{1}{\max\{1, N^{c_1}\sqrt{\mathcal{V}_d(\theta_\alpha^*)}\}}\left(N^{\max\{\rho_0, \frac{\rho_0+c_2}{2}\}+\frac{c_1}{2}}\right)\end{aligned}$$

In this running time, we can find one marked vertex with high probability if it exists. We repeat this quantum random walk until we find $\max\{\frac{N^\zeta}{2}, 1\}$ solutions.

Algorithm for the FindAllSolutions procedure

Pick a random product code $\mathcal{C}_2$.
while the number of solutions found is $< \frac{N^\zeta}{2}$:
    Run our QRW to find a solution and add it to the list of solutions if it hasn't been found.

For $\zeta > 0$, there are N^ζ different solutions that can be found in each α-filter. Each time we find a solution, since the list of solutions found is $< \frac{N^\zeta}{2}$. Therefore, the probability that each solutions found by the QRW is new is at least $\frac{1}{2}$. We have therefore

$$\mathrm{FAS}_1 = \max\{N^\zeta, 1\} \cdot T_1.$$

If $\zeta > 0$, our algorithm finds $\Theta(N^\zeta)$ solutions in time $N^\zeta T_1$ and if $\zeta \leq 0$, our algorithm finds 1 solution in time T_1 with probability $\Theta(N^{-\zeta})$.

5.3 Memory analysis

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

Memory requirements of the quantum random walk. Each vertex v of the graph stores all the $J^v(\vec{t_i})$ which together take space $N^{c_1+\rho_0}$. We need to store a superposition of vertices so we need $N^{c_1+\rho_0}$ quantum registers and we need that same amount of QRAM because we perform insertions and deletions in the database in quantum superposition. All the operations require QRAM access to the whole list L_y which is classically stored and is of size N^c. Therefore, we also require N^c QRAM.

5.4 Optimal parameters for this quantum random walk

Our algorithm takes in argument three parameters: $c \in [0,1]$, $c_1 \leq c$ and $c_2 \leq c_1$ from which we can express all the other variables we use: α, θ^*_α, β, ρ_0 and ζ. We recall these expressions as they are scattered throughout the previous sections:

- α: angle in $[\pi/3, \pi/2]$ that satisfies $\mathcal{V}_d(\alpha) = \frac{1}{N^{1-c}}$.
- $\theta^*_\alpha = 2\arcsin(\frac{1}{2\sin(\alpha)})$.
- β: angle in $[\pi/3, \pi/2]$ that satisfies $\mathcal{V}_d(\beta) = \frac{1}{N^{c_1-c_2}}$.
- ρ_0: non-negative real number such that $N^{\rho_0} = \frac{\mathcal{V}_d(\beta)}{\mathcal{W}_d(\beta,\theta^*_\alpha)}$.
- ζ: real number such that $N^\zeta = N^{2c}\mathcal{V}_d(\theta^*_\alpha)$.

Plugging the value of FAS_1 from the end of Section 5.2 in Proposition 8, we find that the total running time of our quantum sieving algorithm with parameters c, c_1, c_2 is

$$T = N^{c-\zeta}\left(N + N^{1-c}\max\{N^\zeta, 1\}\left(N^{c_1+\rho_0} + \frac{1}{\max\{1, N^{c_1}\sqrt{\mathcal{V}_d(\theta^*_\alpha)}\}}\left(N^{\max\{\rho_0, \frac{\rho_0+c_2}{2}\}+\frac{c_1}{2}}\right)\right)\right).$$

We ran a numerical optimization over c, c_1, c_2 to get our optimal running time, summed up in the following theorem.

Proposition 10. *Our algorithm with parameters*

$$c \approx 0.3300 \quad ; \quad c_1 \approx 0.1952 \quad ; \quad c_2 \approx 0.0603$$

heuristically solves SVP on dimension d in time $T = N^{1.2555+o(1)} = 2^{0.2605d+o(d)}$, uses QRAMM of maximum size $N^{0.3300+o(1)} = 2^{0.0685d+o(d)}$, a quantum memory of size $N^{0.2555+o(1)} = 2^{0.0530d+o(d)}$ and uses a classical memory of size $N^{1+o(1)} = 2^{0.2075d+o(d)}$.

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

$$\alpha \approx 1.1388\text{rad} \approx 65.25^\circ; \quad \theta^*_\alpha \approx 1.1661\text{rad} \approx 66.46^\circ; \quad \beta \approx 1.3745\text{rad} \approx 78.75^\circ$$

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

As well as the quantum walk parameters:

$$\mathcal{S} = N^{c_1+\rho_0} = N^{0.2555}; \quad \mathcal{U} = N^{\rho_0} = N^{0.0603}; \quad \mathcal{C} = 0; \quad \varepsilon = \delta = N^{-c_1} = N^{-0.1952}.$$

The equality $\rho_0 = c_2$ allows to balance the time of the two operations during the update step. With these parameters we also obtain $\mathcal{S} = \mathcal{U}/\sqrt{\epsilon\ \delta} = N^{c_1+\rho_0} = N^{0.2555d}$, which balances the overall time complexity.

Notice that with these parameters, we can rewrite T as

$$T = N^{c-\zeta}\left(N + N^{1-c+\zeta+c_1+\rho_0}\right) = N^{1+c-\zeta} + N^{1+c_1+\rho_0}.$$

Also, we have $c_1 + \rho_0 = c - \zeta$, which equalizes the random walk step with the initialization step. From our previous analysis, the amount of required QRAM is N^c and the amount of quantum memory needed is $N^{c_1+\rho_0}$.

6 Quantum random walk for the FindAllSolutions subroutine: an improved quantum random walk

We now add a variable $\rho \in (0, \rho_0]$ that will replace the choice of ρ_0 above. ρ_0 was chosen 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 $J^v(\vec{t})$ for $\vec{t} \in \mathcal{C}_2$. However, we can relax this and only mark a small fraction of these vertices. This will reduce the fraction of marked vertices, which makes it harder to find a solution, but having a smaller ρ will reduce the running time of our quantum random walk.

The construction is exactly the same as in the previous section just that we replace ρ_0 with ρ. This implies that $|C_2| = N^{\rho+c_1-c_2}$. We can perform the same analysis as above

Time analysis of this QRW 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 when if we have less marked vertices, then there on average more than one marked vertex, so our algorithm at least finds one solution with a constant probability.

The analysis walk is exactly the same than in Section 5.2, each repetition of the quantum random walk takes time T_1 with

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

with

$$\begin{aligned}&\mathcal{S} = N^{c_1+\rho}, \quad \mathcal{U} = N^{\max\{\rho, \frac{\rho+c_2}{2}\}+o(1)}, \quad \mathcal{C} = 1,\\ &\varepsilon = N^{2c_1}N^{\rho-\rho_0}\mathcal{V}_d(\theta^*_\alpha), \quad \delta = N^{-c1}.\end{aligned}$$

The only thing maybe to develop is the computation of ε. We perform the same analysis as above but with $|C_2| = N^{\rho+c_1-c_2}$. This means that Equation 5 of Lemma 1 becomes

$$\begin{aligned}\Pr[\exists \vec{t} \in \mathcal{C}_2,\ \vec{y}_i, \vec{y}_j \in J^v(\vec{t})] &= 1 - (1 - \mathcal{W}_d(\beta, \theta^*_\alpha))^{|C_2|}\\ &\geq |C_2|\mathcal{W}_d(\beta, \theta^*_\alpha) = N^{\rho-\rho_0}.\end{aligned}$$

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 $\mathcal{C}_2$. The algorithm becomes

Algorithm from Section 6 with parameter ρ

while the total number of solutions found is $< \frac{N^\zeta}{2}$:
  Pick a random product code $\mathcal{C}_2$.
  while the number of solutions found is $< \frac{N^{\zeta+\rho-\rho_0}}{2}$ with this $\mathcal{C}_2$:
    Run our QRW with ρ to find a new solution.

With this procedure, we also find $\Theta(N^\zeta)$ solutions in time $N^\zeta T_1$ and $\mathrm{FAS}_1 = N^\zeta T_1$ (Recall that we are in the case $\zeta \geq \zeta + \rho - \rho_0 > 0$). Actually, Optimal parameters will be when $c_2 = 0$ and $\rho \to 0$.

6.1 Analysis of the above algorithm

This change implies that some reducing pairs are missed. For the quantum random walk complexity, this only change 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 $\epsilon = N^{2c_1}\mathcal{V}_d(\theta^*_\alpha) \cdot N^{\rho_0-\rho}$.

However, this increasing is compensated by the reducing of the costs of the setup ($N^{c_1+\rho+o(1)}$) and the update ($2N^{\max\{\rho,\frac{\rho+c_2}{2}\}+o(1)}$).

A numerical optimisation over ρ, c, c_1 and c_2 leads to the following theorem.

Theorem 4 (Theorem 1 restated). *Our algorithm with a free ρ with parameters*

$$\rho \to 0 \quad ; \quad c \approx 0.3696 \quad ; \quad c_1 \approx 0.2384 \quad ; \quad c_2 = 0$$

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

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

$$\alpha \approx 1.1514 \text{ rad}; \quad \theta^*_\alpha \approx 1.1586 \text{ rad}; \quad \beta \approx 1.1112 \text{ rad}; \quad \zeta \approx 0.1313.$$

As well as the quantum walk parameters:

$$\mathcal{S} = N^{c_1+\rho} = N^{0.2384}; \quad \mathcal{U} = N^\rho = N^{o(1)}; \quad \mathcal{C} = 0; \quad \varepsilon = \delta = N^{-c_1} = N^{-0.2384}.$$

With these parameters, 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 10, we

reach the equality $S = U/\sqrt{\epsilon\delta}$, 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 T as

$$T = N^{c-\zeta}\left(N + N^{1-c+\zeta+c_1+\rho}\right) = N^{1+c-\zeta} + N^{1+c_1+\rho}.$$

With our optimal parameters, we have $\rho = 0$ and $c - \zeta = c_1$, which equalizes the random walk step with the initialization step. From our previous analysis, the amount of required QRAM is N^c and the amount of quantum memory needed is N^{c_1}.

7 Space-time trade-offs

By varying the values c, c_1, c_2 and ρ, we can obtain trade-offs between QRAM and time, and between quantum memory and time. All the following results come from numerical observations.

7.1 Trade-off for fixed quantum memory.

We computed the minimized time if we add the constraint that the quantum memory must not exceed 2^{Md}. For a chosen fixed M, the quantum memory is denoted is $2^{\mu_M d} = 2^{Md}$ and the corresponding minimal time by $2^{\tau_M d}$. The variation of M also impacts the required QRAM to run the algorithm, that we denote by $2^{\gamma_M d}$.

So we get a trade-off between time and quantum memory in Figure 1, and the evolution of QRAM in function of M for a minimal time is in Figure 2.

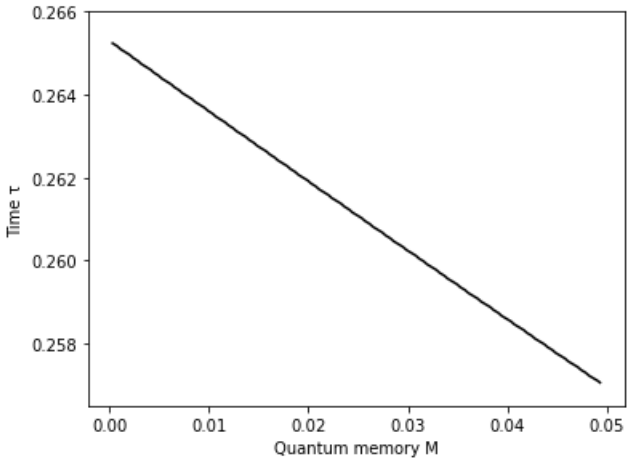

Fig. 1: Quantum memory-time trade-off.

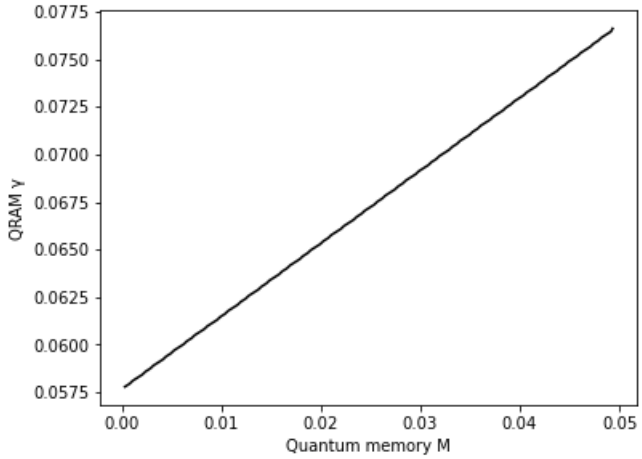

Fig. 2: QRAM in function of available quantum memory for minimized time.

For more than $2^{0.0495d}$ 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 1 and γ_M from Figure 2 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 expression $0.2653 - 0.1670M$. The difference between τ_M and its affine approximation does not exceed $4 \cdot 10^{-5}$. By the same way, the difference between γ_M and its affine average function of expression $0.0578 + 0.3829M$ is inferior to $2 \cdot 10^{-4}$. All this is summarized in the following theorem.

Theorem 5 (Trade-off for fixed quantum memory). *There exists a quantum algorithm using quantum random walks that solves SVP on dimension d which for a parameter $M \in [0, 0.0495]$ heuristically runs in time $2^{\tau_M d + o(d)}$, uses QRAM of maximum size $2^{\gamma_M d}$, a quantum memory of size $2^{\mu_M d}$ and a classical memory of size $2^{0.2075d}$ where*

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

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

In the informal formulation of this theorem, we used the symbols $\lessapprox$ and $\gtrapprox$ that refers to these hidden small values.

7.2 Trade-off for fixed QRAM.

We also get a trade-off between QRAM and time. For a chosen fixed M', the QRAM is denoted by $2^{\gamma_{M'} d} = 2^{M' d}$, and the corresponding minimal time by

$2^{\tau_{M'}d}$. The required quantum memory is denoted $2^{\mu_{M'}d}$. Note that $2^{\mu_{M'}d}$ is the also the amount of the required quantum QRAM called "QRAQM".

This gives a trade-off between time and QRAM in the figure 3, and the evolution of quantum memory in function of M' is in the figure 4.

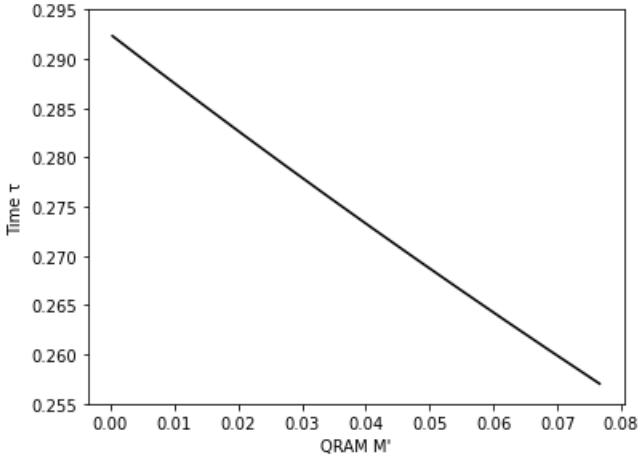

Fig. 3: QRAM-time trade-off.

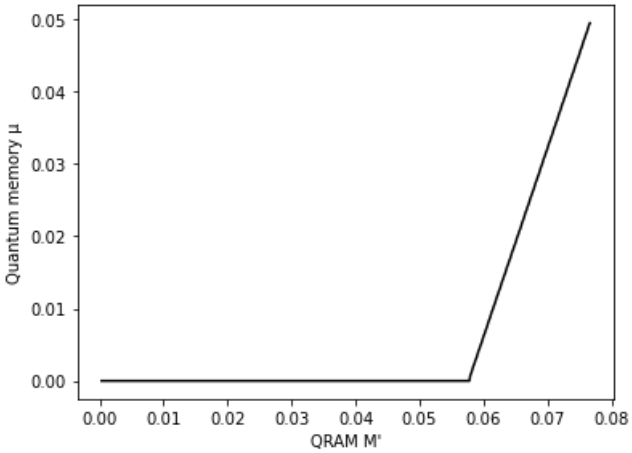

Fig. 4: Quantum memory in function of available QRAM for minimized time.

For more than $2^{0.0767d}$ QRAM, increasing it does not improve the time complexity.

The difference between the function $\tau_{M'}$ and its average affine function of expression $0.2926 - 0.4647 \cdot M'$ does not exceed $6 \cdot 10^{-4}$. This affine function is a 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 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 6 (Trade-off for fixed QRAM). *There exists a quantum algorithm using quantum random walks that solves SVP on dimension d which for a parameter $M' \in [0, 0.0767]$ heuristically runs in time $2^{\tau_{M'} d + o(d)}$, uses QRAM of maximum size $\mathsf{poly}(d) \cdot 2^{\gamma_{M'} d}$, a quantum memory of size $\mathsf{poly}(d) \cdot 2^{\mu_{M'} d}$ and uses a classical memory of size $\mathsf{poly}(d) \cdot 2^{0.2075d}$ where*

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

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

Finally, we present a table with a few values that presents some of the above trade-offs.

Time $\tau_{M'}$	**0.2925**	0.2827	0.2733	**0.2653**	0.2621	0.2598	**0.2570**
QRAM $\gamma_{M'}$	**0**	0.02	0.04	**0.0578**	0.065	0.070	**0.0767**
Q. memory $\mu_{M'}$	0	0	0	**0**	0.0190	0.0324	**0.0495**
Comment	[BDGL16] alg.			[Laa16] alg.			Thm 1.

Fig. 5: Time, QRAM and quantum memory values for our algorithm.

8 Discussion

Impact on lattice-based cryptography. Going from a running time of $2^{0.2653d + o(d)}$ to $2^{0.2570d + o(d)}$ slightly reduces the security claims based on the analysis of the SVP (usually via the BKZ algorithm). For example, if one claims 128 bits of security using the above exponent then one must reduce this claim to 124 bits of quantum security. This of course can usually be fixed with a slight increase of the parameters but cannot be ignored if one wants to have the same security claims as before.

Parallelization. On thing we haven't talked about in this article is whether our algorithm paralellizes well. Algorithm 1 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 $\mathsf{poly}(d)$. Then, there are N

separate instances of **FindAllSolutions** which can be also perfectly parallelized and each one also takes time $\mathsf{poly}(d)$ when $c \to 0$. The **while** loop is repeated $N^{-\zeta} = N^{0.409d}$ times so the total running time (here depth) is $N^{0.409d+o(d)}$ with a classical circuit of width N. Such a result already surpasses the result from [BDGL16] that achieves depth $N^{1/2}$ with a quantum circuit of width N using parallel Grover search.

In the quantum setting, our algorithm parallelizes also quite well. If we consider our optimal parameters ($c = 0.3696$) with a similar reasoning, our algorithm will parallelize perfectly with N^{1-c} processors (so that there is exactly one for each call to **FindAllSolutions** *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 didn't achieve better than a depth of $N^{0.409+o(d)}$ which is the classical parallelization. This is also the case if we use Grover's algorithm as in [Laa16] for the **FindAllSolutions** and we use parallel Grover search as in [BDGL16] so best known (classical or quantum) algorithm with lowest depth that uses a circuit of width N is the classical parallel algorithm described above.

Acknowledgments and paths for improvements

The authors want to thank Simon Apers for helpful discussions about quantum random walks, in particular about the fact that there are no better generic algorithms for finding k different marked than to run the whole random walk (including the setup) $O(k)$ times. There could however be a smarter way to do this in our setting which would improve the overall complexity of our algorithm. Another possible improvement would be to embed the local sensitivity property in the graph on which we perform the random walk instead of working on the Johnson graph.

References

[AGJO+15] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 17(12):123010, Dec 2015.

[AGPS20] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 583–613, Cham, 2020. Springer International Publishing.

[AINR14] Alexandr Andoni, Piotr Indyk, Huy Lê Nguyên, and Ilya Razenshteyn. Beyond locality-sensitive hashing. *SODA*, page 1018–1028, 2014.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery.

[Ajt98] Miklos Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). *30th Annual ACM Symposium on Theory of Computing Proceedings*, pages 10 – 19, 1998.

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

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. *STOC*, page 793–801, 2015.

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666. Springer, 2020.

[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.

[BJLM13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, volume 7932 of *Lecture Notes in Computer Science*, pages 16–33. Springer, 2013.

[BL16] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope lsh. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2016*, pages 3–23, Cham, 2016. Springer International Publishing.

[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*, page 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, 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.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169?178, New York, NY, USA, 2009. Association for Computing Machinery.

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[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.

[HM18] Alexander Helm and Alexander May. Subset sum quantumly in 1.17^{n}. In Stacey Jeffery, editor, *13th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2018, July 16-18, 2018, Sydney, Australia*, volume 111 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

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

[Kan83] R. Kannan. Improved algorithms for integer programmingand related lattice problems. *Proceedings of the 15thSymposium on the Theory of Computing (STOC), ACM Press*, pages 99 – 108, 1983.

[Kle00] Philip Klein. Finding the closest lattice vector when it's unusually close. *SODA*, page 937–941, 2000.

[KMPM19] Elena Kirshanova, Erik Martensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. *ASIACRYPT*, 2019.

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2017.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 3–22, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[Laa16] Thijs Laarhoven. *Search problems in cryptography, From fingerprinting to lattice sieving*. PhD thesis, Eindhoven University of Technology, 2016.

[LdW15] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In Kristin Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology – LATINCRYPT 2015*, pages 101–118, Cham, 2015. Springer International Publishing.

[LLL82] A.K. Lenstra, H.W. Lenstra, and L. Lovasz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, pages 513–534, 1982.

[LMvdP15] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Des. Codes Cryptogr.*, 77(2-3):375–400, 2015.

[MNRS11] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM J. Comput.*, 40(1):142–164, 2011.

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

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, New York, NY, USA, 2000.

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

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

[WLTB11] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved nguyen-vidick heuristic sieve algorithm for shortest vector problem. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, page 1–9, New York, NY, USA, 2011. Association for Computing Machinery.

[ZPH14] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the shortest vector problem. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, pages 29–47, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.