Classical and Quantum 3 and 4-Sieves to Solve SVP with Low Memory

André Chailloux, Johanna Loyer

Classical and quantum 3 and 4-sieves to solve SVP with low memory

André Chailloux and Johanna Loyer

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

Abstract. The Shortest Vector Problem (SVP) is at the foundation of lattice-based cryptography. The fastest known method to solve SVP in dimension d is by lattice sieving, which runs in time $2^{td+o(d)}$ with $2^{md+o(d)}$ memory for constants $t, m \in \Theta(1)$. Searching reduced vectors in the sieve is a problem reduced to the configuration problem, *i.e.* searching k vectors satisfying given constraints on their pairwise scalar products.
In this work, we present a framework for k-sieve algorithms: we filter the input list of lattice vectors using a code structure modified from [Bec+16] to get lists centred around k codewords summing to the null-vector. Then, we solve a simpler instance of the configuration problem in the k filtered lists. Based on this framework, we describe classical sieves for $k = 3$ and 4 that introduce new time-memory trade-offs. We also use the k-Lists algorithm [Kir+19] inside our framework, and this improves the time for $k = 3$ and gives new trade-offs for $k = 4$.

Keywords. Shortest Vector Problem (SVP), Lattice sieving, Locality-sensitive filtering (LSF), Configuration problem.

1 Introduction

The Shortest Vector Problem (SVP) is a central problem in lattice-based cryptography. For a given d-dimensional lattice, SVP asks to find a shortest non-zero vector in the lattice. This problem admits a variety of derived problems such as SIS, LWE and their modular or ring versions, on which several of the (believed to be) quantum-resistant cryptographic protocols rely, such as Dilithium [Duc+19] and Kyber [Bos+18] both winners of the NIST standardization process. It is therefore crucial to estimate the hardness of these constructions both in classical and quantum models.

There are two main families of algorithms for SVP: those based on enumeration [FP85; Kan83; Poh81] and those based on sieving [NV08; MV10]. The former does not have good asymptotic running but requires a small amount of memory and has good performances in practice. The latter has much better asymptotic running time — especially with heuristics that improve the analysis of these algorithms — but requires a very large amount of memory, sometimes

as much as the running time. Despite these strong memory requirements, the current best algorithms for SVP in practice are based on sieving methods[1].

The rough idea of sieving algorithms is the following: we start from a list of lattice points of large norm and sum them in order to find shorter lattice points, and repeat until we find a short vector. However, the number of points we have to start with is very large, which explains the large memory requirements. For example, in 2-sieve algorithms, where we start from a list of N points and build shorter points by summing pairs, we need to take $N = 2^{0.2075d+o(d)}$ in order to be able to build enough shorter lattice points.

Reducing this memory requirement makes the attack more materially practical, and a way to do it is by the k-sieve introduced in [BLS16] and then improved by [HK17; HKL18; Kir+19]. The idea is to sum k lattice points instead of pairs at each sieving step in order to find shorter ones. This decreases the number N of lattice points that we need at each step to find the same number N of shorter lattice points. However, this will drastically increase the time to perform the sieving step. For instance, a naive exhaustive search of each k-tuple takes time $O(N^k)$, and the fact that N is smaller does not outweigh this increased exponent, see Table 1.

k	2	3	4	5	6
$\frac{\log_2(N)}{d}$	0.2075	0.1887	0.1724	0.1587	0.1473
$\frac{\log_2(N^k)}{d}$	0.4150	0.5661	0.6896	0.7935	0.8838

Table 1: N is the number of points needed for k-sieving. N^k is the running time of a naive exhaustive search for k-sieving.

There are two main ideas that significantly improve the complexity of k-sieving algorithms:

- For 2-sieving, perform locality-sensitive filtering (LSF). The idea is to regroup lattice points into filters for which pairs are more likely to be reducible than random pairs. The most efficient known way to perform LSF for 2-sieve is to construct a code that behaves as a random code but which is efficiently decodable. In [Bec+16], the authors use a random product code to achieve this. Then, filters will correspond to all lattice points which will be close to a given code point.
- For $k > 2$, one can replace the reducibility constraint $\|\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k\| \leq R$ (starting from vectors of norm R) with constraints of the form $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle \leq C_{i,j}$ for some well-chosen $C_{i,j}$. This is known as the configuration problem. The main advantage is that we now only have constraints on pairs of points instead of k-tuples and we can use much more efficient algorithms, including the LSF idea presented above.

[1] See https://www.latticechallenge.org/svp-challenge/

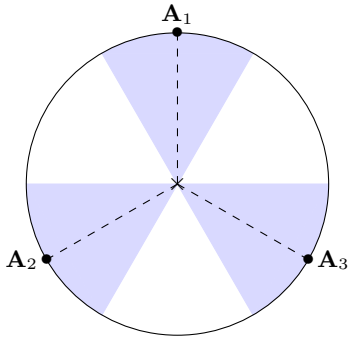


Fig. 1: $\mathbf{A}_1 + \mathbf{A}_2 + \mathbf{A}_3 = \vec{0}$ with $\langle \mathbf{A}_i | \mathbf{A}_j \rangle = -\frac{1}{2}$ for $i \neq j$.

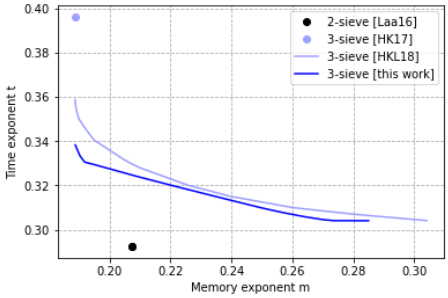
In this work, we add an extra idea that gives better time-memory trade-offs for 3-sieve and 4-sieve algorithms, both in the classical and quantum regimes, and introduce the notion of LSF tailored for k-sieving. In previous algorithms, we first start from a configuration problem and then use LSF only on pairs of points. Here, we first use LSF in a way to construct many lists of lattice points $L_1, \ldots, L_k$ such that k-tuples $(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k) \in L_1 \times \cdots \times L_k$ are more likely to reduce than random k-tuples.

We illustrate briefly this idea with an example: in 3-sieve algorithms, we look for triplets of lattice points $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3)$ each of norm R such that $\|\vec{\boldsymbol{x}}_1 + \vec{\boldsymbol{x}}_2 + \vec{\boldsymbol{x}}_3\| \leq R$. Instead of directly translating this condition into a configuration problem, we first perform a filtering step as follows: consider a list-decodable code that contains many triples of words $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$ such that $\mathbf{A}_1 + \mathbf{A}_2 + \mathbf{A}_3 = \vec{0}$ (see Figure 1). If we consider lists of lattice points L_1, L_2, L_3 such that each L_i contains lattice points close to $\mathbf{A}_i$, then triplets of points $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3) \in L_1 \times L_2 \times L_3$ of norm R are more likely to satisfy $\|\vec{\boldsymbol{x}}_1 + \vec{\boldsymbol{x}}_2 + \vec{\boldsymbol{x}}_3\| \leq R$ than random triplets. We then use known algorithms on configuration search on these triplets of lists to find reducible triplets.

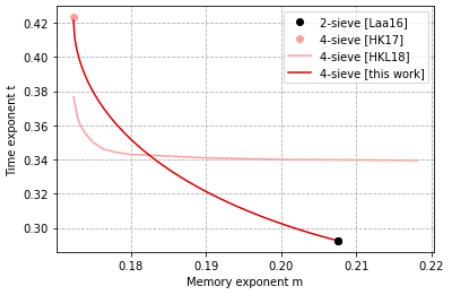
We summarize our contributions below.

- We show how to extend the construction of random product codes of [Bec+16] as a means of performing LSF tailored for k-sieving. Our code will also be efficiently decodable and the set of codewords can be partitioned into sets $\{\mathbf{A}_1, \ldots, \mathbf{A}_k\}$ each of size k such that $\mathbf{A}_1 + \cdots + \mathbf{A}_k = \vec{0}$.
- We analyze classical and quantum algorithms for 3 and 4-sieving using this k-sieve tailored filtering, and get improved time-memory trade-offs for these algorithms.

We first analyze our results for classical algorithms (see Figure 2). For the 3-sieve, our algorithm performs better in the minimal memory regime. In the main text, we also present other time-memory trade-offs. However, when we do not restrict memory, we obtain the same running time $2^{0.3041d+o(d)}$ as in [HKL18] and our method does not give improvements here. For 4-sieve algorithms, the



(a) Classical 3-sieve (Theorem 3).

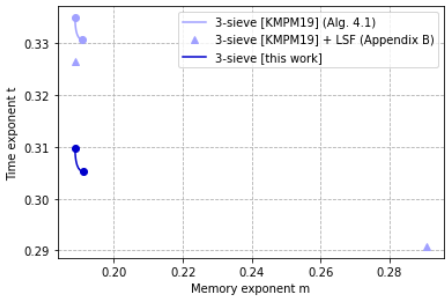


(b) Classical 4-sieve (Theorem 4).

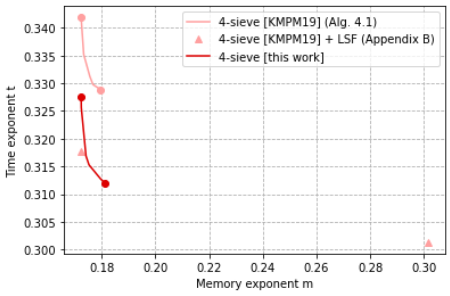
Fig. 2: Time $T = 2^{td+o(d)}$ for classical algorithms as a function of available memory $2^{md+o(d)} = 2^M$.

situation is a little different. We use a different algorithm than the ones studied in previous work. We essentially combine sequentially two 2-sieve algorithms. However, we first perform our tailored LSF on 4-tuples of points to speed up this process. As Figure 2 shows, this algorithm does not perform well in the minimal memory regime ($M = 2^{0.1723d+o(d)}$) but then works much better for slightly larger memories, outperforming our 3-sieve algorithm and also the best previously known running time for 4-sieve, which used more memory.

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



(a) Quantum 3-sieve (Theorem 5).



(b) Quantum 4-sieve (Theorem 6).

Fig. 3: Time $T = 2^{td+o(d)}$ for quantum 3 and 4-sieves algorithms as a function of available memory $2^{md+o(d)} = 2^M$.

In the quantum setting (see Figure 3), we use the same algorithms as in [Kir+19] so the comparison can be made more directly. Our algorithm uses our tailored filtering and then applies Algorithm 4.1 of [Kir+19], which is not the best algorithm for the configuration problem for low values of k. What we show is that this algorithm benefits from this prefiltering. The results should be compared with the state-of-the-art Algorithm B.2 of [Kir+19]. However, only the extremities of the trade-offs of their algorithm were given, represented by triangles on the graphs of Figure 3. For $k = 3$ in the minimal memory regime $M = 2^{0.1887d+o(d)}$, we achieve time $T = 2^{0.3098d+o(d)}$ improving the time $T = 2^{0.3266d+o(d)}$ of Algorithm B.2 in [Kir+19]. For $k = 4$, our algorithm does not work well for the lowest memory regime but gives a new interesting time-memory trade-off.

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

Outline of the article. Section 2 introduces the helpful preliminaries on quantum computing and lattice sieving. Section 3 presents the new code structure for the filtering step tailored for k-sieving. Then, we present a framework to solve SVP by sieving in Section 4, and describe some instances within this framework in the classical model in Section 5 and in the quantum model in Section 6.

2 Preliminaries

Notations. We designate the elements of $\mathbb{R}^d$ as vectors as well as points. The norm considered in this work is Euclidean and is denoted by $\|\cdot\|$. We denote by $\mathcal{S}^{d-1} = \{\vec{\boldsymbol{x}} \in \mathbb{R}^d : \|\vec{\boldsymbol{x}}\| = 1\}$ the d-dimensional unit sphere. For two vectors $\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2 \in \mathbb{R}^d$, $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle$ denotes their scalar product and $\theta(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) = \arccos(\frac{\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle}{\|\vec{\boldsymbol{x}}_1\| \|\vec{\boldsymbol{x}}_2\|})$ denotes their angle. We use the notation $\widetilde{\mathcal{O}}$ to denote running times $T = \widetilde{\mathcal{O}}\left(2^{cd}\right)$, which ignores sub-exponential factors in d.

2.1 Quantum computing

QRAM model. The Quantum Random Access Memory (QRAM) is an operation added to the quantum circuit model. We consider here only quantum-accessible classical memory (sometimes denoted as QRACM in the literature). Consider N classical registers $x_1, ..., x_N \in \{0,1\}^d$ stored in memory. A QRAM operation consists of applying the following unitary

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

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

Proposition 1 (Grover's algorithm [Gro96]). *We are given QRAM access to a list $L = \{x_1, ...x_\ell\}$. We consider a function $f : L \to \{0, 1\}$, associated to its unitary $O_f : |x\rangle |y\rangle \to |x\rangle |y \oplus f(x)\rangle$, such that there are t elements $x_i \in L$ said "marked" verifying $f(x_i) = 1$. The value t is not necessarily known. There exists a quantum algorithm, called Grover's algorithm, that returns a marked element with probability greater than $1/2$ using $\mathcal{O}(\sqrt{|L|/t})$ calls to O_f. Classically, this problem cannot be solved with a better average complexity than $\Theta(|L|/t)$ queries in the black box model.*

Proposition 2 (Quantum Amplitude amplification [Bra+02]). *Let $\mathcal{A}$ be an algorithm without measurements that finds a solution $x \in L$ such that $f(x) = 1$ with a success probability p. Quantum amplitude amplification returns a solution with probability $1/2$ using $\mathcal{O}(1/\sqrt{p})$ calls to O_f.*

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

Proposition 3. *Grover's algorithm (resp. Quantum Amplitude Amplification) described above can have a success of $1 - 2^{-\eta}$ with $\mathcal{O}(\eta\sqrt{|L|/t})$ (resp. $\mathcal{O}(\eta/\sqrt{p})$) calls to O_f.*

2.2 Lattice sieving

Definition 1 (Lattice). *Given a basis $B = (\vec{\boldsymbol{b}}_1, ...\vec{\boldsymbol{b}}_m) \in \mathbb{R}^d$ of linearly independent vectors, the lattice generated by B is defined as $\mathcal{L}(B) = \left\{\sum_{i=1}^m z_i \vec{\boldsymbol{b}}_i, z_i \in \mathbb{Z}\right\}$. For simplicity, we work with lattices of full rank, i.e. $d = m$.*

Definition 2 (Shortest Vector Problem). *Given a basis B, the Shortest Vector Problem (SVP) asks to find a shortest non-zero vector of $\mathcal{L}(B)$. By Minkowski's theorem, the Euclidean norm of a shortest vector of $\mathcal{L}(B)$ is upper-bounded by $\sqrt{d} \cdot \det(B)^{1/d}$.*

Sieving algorithms. The sieving algorithms, introduced by [NV08], are a class of heuristic algorithms that solves SVP. Given a list of lattice vectors of norm at most R and a reducing factor $\gamma < 1$, a sieving step will return a list of lattice vectors of norm at most γ. To obtain these reduced vectors, it computes all the differences between pairs of vectors of the input list and fills the output list with those which are of norm at most γ. Then, it iteratively builds lists of shorter lattice vectors by applying this sieve step. The first list of lattice vectors can be sampled with Klein's algorithm [Kle00] for example. Because the norms of the list vectors reduce with a factor by $\gamma < 1$ at each sieve step, the output list will hopefully contain a non-zero shortest lattice vector after a polynomial number of iterations.

We present here two simplifications of notation. First, we will only consider the case $R = 1$. Indeed, all the algorithms we consider will be independent of R. Also, in practice, we have $\gamma \approx 1$ (typically $\gamma = 1 - \frac{1}{\text{poly}(d)}$). For simplicity of

Algorithm 1 NV-sieve step

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

Algorithm 2 Solve SVP by the sieving method

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

notations, we will fix $\gamma = 1$ which doesn't change the overall analysis of these algorithms.

There is also another family of sieving algorithms called Gauss-sieve. These have worse asymptotic time-memory trade-offs so we don't analyze them in this work since we are interested in the best asymptotic time-memory trade-offs. However, they are usually faster in practice and it is an interesting follow-up to look at how our new ideas for sieving apply in this setting.

The sieving algorithms stand under the following heuristic.

Heuristic 1. Lattice points behave like uniform points.

Notice that uniform vectors of norm at most 1 are with a high probability of norm close to 1. So we will consider that the lattice vectors are lying on the sphere $\mathcal{S}^{d-1}$. The relevance of this heuristic has been studied in [NV08]. It becomes invalid when the vectors become short, but in this case, we can assume we have solved SVP. In practice, the faster algorithms to solve SVP rely on this heuristic.

Tuple-sieve and k-List algorithms. The NV-sieve described above can be generalized to the k-sieve.

Definition 3 (Approximate k-Lists problem). *Given k lists $L_1 ..., L_k$ of equal exponential (in d) size N and whose elements are i.i.d. uniformly chosen vectors from $\mathcal{S}^{d-1}$, the approximate k-List problem is to find N k-tuples $(\vec{x}_1, \ldots, \vec{x}_k) \in L_1 \times \cdots \times L_k$ satisfying $\|\vec{x}_1 + \cdots + \vec{x}_k\| \leq 1$.*

Volume of spherical cap / Hypercone filter. We define the spherical cap of center $\vec{s}$ and angle α as follows:

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

In order to compute the complexity of sieving algorithms, we will need the following value.

Proposition 4. *([Bec+16], Lemma 2.1) For an angle $\alpha \in [0, \pi/2]$ and a vector $\vec{\boldsymbol{v}} \in \mathcal{S}^{d-1}$, the ratio of the volume of a spherical cap $\mathcal{H}_{\vec{\boldsymbol{v}},\alpha}$ to the volume of the sphere $\mathcal{S}^{d-1}$ is*

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

2.3 Configurations

In the approximate k-Lists problem, we have the condition $\|\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k\| \leq 1$. Notice that we can rewrite

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

This means that the condition on $\|\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k\|$ can be verified if some constraints on the $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle$ are verified. This motivates the following definition:

Definition 4 (Configuration). *The configuration C of k points $\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k \in \mathcal{S}^{d-1}$ is the Gram matrix of the $\vec{\boldsymbol{x}}_i$'s, i.e. $C_{i,j} = \langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle$.*

A configuration is said balanced when for $i \neq j$, $C_{i,j} = -1/k$ and $C_{i,i} = 1$. In this case, the tuple points will form together the summits of a regular polyhedron inscribed in the sphere.

For $I \subset [k]$, we denote by $C[I]$ the $|I| \times |I|$ submatrix of C obtained by restricting C to the rows and columned whose indexes are in I.

Definition 5 (Configuration problem). *Let $k \in \mathbb{N}$ and $\epsilon > 0$. Suppose we are given a target configuration $C \in \mathbb{R}^{k \times k}$. Given lists $L_1, \ldots, L_k$ all of exponential (in d) size $|L|$ whose elements are i.i.d. uniform from $\mathcal{S}^{d-1}$, the configuration problem consists of finding a $1 - o(1)$ fraction of all solutions, where a solution is a k-tuple $(\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k)$ with $\vec{\boldsymbol{x}}_i \in L_i$ such that $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle \leq C_{i,j}$ for all i, j.*

[HK17] showed that the approximate k-Lists problem (Definition 3) can be reduced to the configuration problem.

Proposition 5 ([HK17]). *The probability that a k-tuple of i.i.d. uniformly random points on $\mathcal{S}^{d-1}$ satisfies a given configuration $C \in \mathbb{R}^{k \times k}$ is $\det(C)^{d/2}$.*

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

$$|L| = \widetilde{\mathcal{O}}\left(\left(\frac{1}{\det(C)}\right)^{\frac{d}{2(k-1)}}\right). \tag{1}$$

For a fixed k, the minimum value of $|L|$ is reached when the configuration C is balanced. In particular, for a balanced configuration with $k = 2$ we require $(4/3)^{d/2} = 2^{0.2075d+o(d)}$ points ; for $k = 3$, $(27/16)^{d/4} = 2^{0.1887d+o(d)}$ points and for $k = 4$, $(256/125)^{d/6} = 2^{0.1724d+o(d)}$ points. See Table 1 for the other values of k. We see that this value decreases with k, and that is the main interest of the k-sieve: to reduce the minimum memory we require to solve SVP. Considering non-balanced configurations leads to not reaching the lower bound for memory, but it can allow decreasing time by adding a little memory. This is useful to get better time-memory trade-offs.

Proposition 6 (Size of the filtered lists $L_i(x_j)$ given $C_{i,j}$ [Kir+19]). *We are given a configuration $C \in \mathbb{R}^{k\times k}$ and lists $L_1, \dots L_k \subset \mathcal{S}^{d-1}$ each of size $|L_j|$. For $\vec{\boldsymbol{x}}_1, \dots, \vec{\boldsymbol{x}}_i \in \mathcal{S}^{d-1}$, we denote*

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

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

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

And in particular,

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

3 Code structure and filtering

3.1 Locality Sensitive Filtering

Random Product Code (RPC). We assume $d = m \cdot b$, for $m = O(\text{polylog}(d))$ 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 $\mathfrak{C}$ of parameters $[d, m, B]$ on subsets of $\mathbb{R}^d$ and of size B^m is defined as a code of the form

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

where Q is a uniformly random rotation over $\mathbb{R}^d$ and the subcodes $\mathfrak{C}_1, \dots, \mathfrak{C}_m$ are sets of B vectors, sampled uniformly and independently random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$, so that codewords are points of the sphere S^{d-1}. We can have a full description of $\mathfrak{C}$ by storing mB points corresponding to the codewords of $\mathfrak{C}_1, \dots, \mathfrak{C}_m$ and by storing the rotation Q. When the context is clear, $\mathfrak{C}$ will correspond to the description of the code or to the set of codewords.

The code points of $\mathfrak{C}$ behave like random points of the sphere $\mathcal{S}^{d-1}$. This was argued in [Bec+16], see for instance Lemma 5.1 and Appendix C therein. Random product codes can be easily decoded in some parameter range, as the following proposition shows.

Proposition 7 ([Bec+16]). *Let $\mathfrak{C}$ be a random product code of parameters (d, m, B) with $m = log(d)$ and $B^m = N^{O(1)}$. For any $\vec{\boldsymbol{x}} \in \mathcal{S}^{d-1}$ and angle α, one can compute $\mathcal{H}_{\vec{\boldsymbol{x}},\alpha} \cap \mathfrak{C}$ in time $N^{o(1)} \cdot |\mathcal{H}_{\vec{\boldsymbol{x}},\alpha} \cap \mathfrak{C}|$, where $\mathcal{H}_{\vec{\boldsymbol{x}},\alpha}$ is defined in Proposition 4.*

Definition 6 (Hypercone filter). *Given a center $\mathbf{A}_i^j \in \mathcal{S}^{d-1}$ and an angle α, the filter $f_{\mathbf{A}_i^j,\alpha}$ is the set of all points in $\mathcal{S}^{d-1}$ of angle at most α with $\mathbf{A}_i^j$.*

Random Product Codes are useful tools to search reducing pairs of lattice vectors in a sieving step (Algorithm 1). Indeed, given a list of lattice vectors on $\mathcal{S}^{d-1}$ and a Random Product Code $\mathfrak{C}$, we can efficiently compute for a given vector all its nearest codewords. In this way, we construct lists that contain vectors from the input list close to the codewords. Each codeword is considered the center of a filter. Then, we can search in the same filter two vectors $\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2$ such that $\vec{\boldsymbol{x}}_1 - \vec{\boldsymbol{x}}_2$ is reduced. By adding Locality Sensitive Filtering using Random Product Codes [Bec+16; Laa16] in a sieving step, this provides the actual best algorithms to solve SVP. The sieve with LSF reaches time $2^{0.292d+o(d)}$ in the classical model [Bec+16], in time $2^{0.257d+o(d)}$ in the quantum model [CL21; Hei21; Bon+22].

However, a k-sieve structure for $k > 2$ searches k-tuples such that $\vec{\boldsymbol{x}}_1 + \cdots + \vec{\boldsymbol{x}}_k$ is reduced. Then, searching within one unique filter does not permit to quickly find a solution without having to check a lot of non-reducing elements. So we will slightly modify the construction of the random product code in order to take into account a configuration.

k-Random Product Code In order to describe our k-Random Product Code construction, we start with the case $k = 3$. Instead of constructing fully random codes $\mathfrak{C}_1, \ldots, \mathfrak{C}_m$, we will construct random codes $\mathfrak{C}_i$ which have the following property:

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

More formally, we assume $d = m \cdot b$, for $m = O(\text{polylog}(d))$ 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 with triangles $\mathfrak{C}$ of parameters $[d, m, B]$ on subsets of $\mathbb{R}^d$ and of size B^m is defined as a code of the form

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

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

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

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

   The code $\mathfrak{C}$ is then the set of points $\{\mathbf{A}_1^1, \mathbf{A}_2^1, \mathbf{A}_3^1, \ldots, \mathbf{A}_1^{B/3}, \mathbf{A}_2^{B/3}, \mathbf{A}_3^{B/3}\}$.

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

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

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

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

1. Sample B/k random vectors $\mathbf{A}_1^1, \ldots, \mathbf{A}_1^{B/k}$ sampled uniformly at random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$.
2. For each $j \in [B/k]$, pick a random vector $\mathbf{A}_2^j$ sampled uniformly at random over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$ with the condition that $\langle \mathbf{A}_1^j | \mathbf{A}_2^j \rangle = -\frac{1}{(k-1)m}$. Then, for $i \in [|2, k-1|]$, pick random vectors $\mathbf{A}_i^j$ such that for each previous $i' \in [i]$, $\langle \mathbf{A}_i^j | \mathbf{A}_{i'}^j \rangle = -\frac{1}{(k-1)m}$.
3. For each $j \in [B/k]$, let $\mathbf{A}_k^j$ be the unique point on the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$ such that $\sum_{i=1}^{k} \mathbf{A}_i^j = \vec{0}$. The code $\mathfrak{C}$ is then the set of points $\{\mathbf{A}_i^j\}_{i\in[k], j\in[B/3]}$.

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

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

For each $j \in [B/k]$, we actually take $\langle \mathbf{A}_i^j | \mathbf{A}_{i'}^j \rangle = -1/(k-1)$ for $i \neq i'$, because this balanced configuration optimizes the number of k-tuples whose vectors are respectively close to the centers $\mathbf{A}_i^j$ (See Proposition 5).

Proposition 8. *Let $\mathfrak{C}$ be a random product code with triangles of parameters $[d, m, B]$ with $m = log(d)$ and $B^m = N^{O(1)}$. For any $\vec{\boldsymbol{x}} \in \mathcal{S}^{d-1}$ and angle α, one can compute $\mathcal{H}_{\vec{\boldsymbol{x}},\alpha} \cap \mathfrak{C}$ in time $N^{o(1)} \cdot |\mathcal{H}_{\vec{\boldsymbol{x}},\alpha} \cap \mathfrak{C}|$.*

Proof. The decoding algorithm of Proposition 7 presented in [Bec+16] uses only the product structure of the code and not how the codes $\mathfrak{C}_1, \ldots, \mathfrak{C}_m$ are constructed. The same algorithm will therefore also efficiently decode random product codes with triangles. □

Definition 7 (Tuple-filter). *Let $\mathfrak{C}$ be a k-RPC with codewords $(\mathbf{A}_i^j)$ for $i \in [k]$ and $j \in [|\mathfrak{C}|/k]$ such that $\forall j \in [|\mathfrak{C}|/k], \sum_{i=1}^{k} \mathbf{A}_i^j = \vec{0}$. Given angle α, we call a tuple-filter $(f_{\alpha,\mathbf{A}_1^j}, \ldots, f_{\alpha,\mathbf{A}_k^j})$, with $f_{\alpha,\mathbf{A}_i^j}$ filters (see Definition 6).*

3.2 Residual vectors in filter.

Proposition 9 ([HKL18], Lemma 3). *We are given vectors i.i.d. uniformly random over $\mathcal{S}^{d-1}$ and a filter of center $\mathbf{A}$ and angle α. Then the vectors of angle at most α with $\mathbf{A}$ are i.i.d. uniformly random over the border of the filter. Their residual vectors are i.i.d. uniformly random over the (d-1)-dimensional sphere $\{\vec{\boldsymbol{y}} \in \mathbb{R}^d : \|\vec{\boldsymbol{y}}\| = 1, \theta(\vec{\boldsymbol{y}}, \mathbf{A}_i) = \alpha\}$.*

Then for a random $\vec{\boldsymbol{x}} \in \mathcal{S}^{d-1}$ of angle α with a center of filter $\mathbf{A}$, we can write for some $\vec{\boldsymbol{y}} \perp \mathbf{A}$,

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

We call $\vec{\boldsymbol{y}}$ the residual vector of $\vec{\boldsymbol{x}}$ on the filter of center $\mathbf{A}$ and angle α.

We are given a list L of lattice vectors assumed to be i.i.d. uniformly random over $\mathcal{S}^{d-1}$. We choose an angle α and sample a k-RPC $\mathfrak{C}$ having $1/\mathcal{V}(\alpha)$ codewords. Going through the $\vec{\boldsymbol{x}}$'s in the list L, we decode $\vec{\boldsymbol{x}}$ to its nearest unique codeword $\mathbf{A} \in \mathfrak{C}$. This step, called prefiltering, separates L into disjoint sublists, each one of size $N \cdot \mathcal{V}(\alpha)$. We focus on only one chosen tuple of filters of centers $\mathbf{A}_1, \ldots, \mathbf{A}_k$ respectively associated to the lists $L_1, \ldots, L_k$. By Proposition 9, with high probability the angle between any $\vec{\boldsymbol{x}} \in L_i$ and $\mathbf{A}_i$ is α.

While filling the list L_i with the $\vec{\boldsymbol{x}}$, we fill in parallel a list R_i with their residual vectors $\vec{\boldsymbol{y}}$ in the filter of center $\mathbf{A}_i$. Note that the points in L_i are i.i.d. uniformly random over the (d-1)-dimensional sphere $\{\vec{\boldsymbol{y}} \in \mathbb{R}^d : \|\vec{\boldsymbol{y}}\| = 1, \theta(\vec{\boldsymbol{y}}, \mathbf{A}_i) = \alpha\}$. See Figure 4 for illustration.

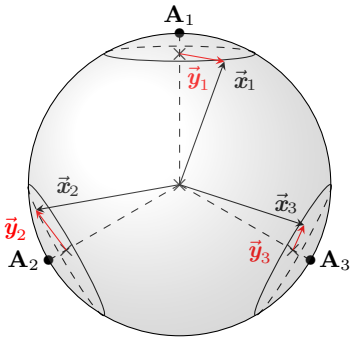


Fig. 4: List vectors $\vec{\boldsymbol{x}}_i \in L_i$ in their filters of centers $\mathbf{A}_i$ and their respective residual vectors $\vec{\boldsymbol{y}}_i \in R_i$, where for $i \neq j$, $\langle \mathbf{A}_i | \mathbf{A}_j \rangle = -\frac{1}{2}$.

Proposition 10. *Using the above notations for the lists L_i's and R_i's, a k-tuple $\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k \in L_1 \times \cdots \times L_k$ is reducing iff. their residual vectors $\vec{\boldsymbol{y}}_1, \ldots, \vec{\boldsymbol{y}}_k \in R_1 \times \cdots \times R_k$ satisfy*

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

Proof. By Proposition 9, we can consider that for $i \neq j$, $\langle \mathbf{A}_i | \vec{\boldsymbol{y}}_j \rangle = 0$. And for $i \neq j$, $\vec{\boldsymbol{x}}_i \in L_i$ and $\vec{\boldsymbol{x}}_j \in L_j$, we obtain:

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

Then we have

$$\left\| \sum_{i=1}^{k} \vec{\boldsymbol{x}}_i \right\|^2 = \left\| \sum_{i=1}^{k} \sin(\alpha) \vec{\boldsymbol{y}}_i \right\|^2 = k \sin^2(\alpha) + 2 \sin^2(\alpha) \left(\sum_{1 \leqslant i < j \leqslant k} \langle \vec{\boldsymbol{y}}_i | \vec{\boldsymbol{y}}_j \rangle \right). \quad (3)$$

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

Lemma 1. *Let be a configuration $C \in \mathbb{R}^{k \times k}$ and an angle α. If a k-tuple $\vec{\boldsymbol{x}}_1, \ldots, \vec{\boldsymbol{x}}_k$ satisfies C then their residual vectors $\vec{\boldsymbol{y}}_1, \ldots, \vec{\boldsymbol{y}}_k$ on a filter of angle α satisfies the configuration $C'(\alpha)$ with for $i \neq j$,*

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

Proof. As written in the previous proof, for $i \neq j$ we have $\langle \vec{\boldsymbol{x}}_i | \vec{\boldsymbol{x}}_j \rangle = \cos^2(\alpha) \langle \mathbf{A}_i | \mathbf{A}_j \rangle + \sin^2(\alpha) \langle \vec{\boldsymbol{y}}_i | \vec{\boldsymbol{y}}_j \rangle$. The configuration C gives constraints over the $\vec{\boldsymbol{x}}_i$'s, and $C'(\alpha)$ over the $\vec{\boldsymbol{y}}_i$'s ; and $\langle \mathbf{A}_i | \mathbf{A}_j \rangle$ is fixed at $-1/(k-1)$. □

If we consider a balanced configuration C for the $\vec{\boldsymbol{x}}_i$'s, then we have for $i \neq j$, $C_{i,j} = -1/k$ all equal. For residual vectors on a filter of angle α, this also implies $C'_{i,j}(\alpha)$ all equal for $i \neq j$. There are $C'_{i,j}$ at number $\sum_{i=1}^{k-1} i = k \cdot (k-1)/2$. Thus for $i \neq j$ we will have $C'_{i,j}(\alpha) = \frac{2}{k \cdot (k-1)} \cdot I_k(\alpha)$, with $I_k(\alpha)$ as defined in Proposition 10.

4 Framework

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

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

Parameters. The algorithm takes into its input an angle α, and a configuration $C \in \mathbb{R}^{k \times k}$ that defines constraints over the lattice vectors. We will discuss later how to choose them optimally. From α and C, we can compute the configuration $C'(\alpha)$ over the residual vectors on filters of angle α using Lemma 1. From C, we also know the number of vectors we require to achieve the sieve step, which gives by Proposition 5 the minimum memory requirement $|L| = \widetilde{\mathcal{O}}(\det(C)^{-\frac{d}{2(k-1)}})$.

Algorithm 3 Framework for our new k-sieves

Require: List L of lattice vectors of norm at most R ; reducing factor $\gamma < 1$.
Parameters: $k \in \mathbb{N}$; angle $\alpha \in (0, \pi/2]$; target configuration C.
Ensure: List L_{out} of lattice vectors of norm at most γR.
$L_{out} = \emptyset$
while $|L_{out}| < |L|$ **do** ▷ $\text{NbRep}_{\alpha,C}$ repeats
Sample a k-RPC code $\mathfrak{C}$ having $k \cdot 1/\mathcal{V}(\alpha)$ codewords
$L_i^j = \emptyset$ **for** $i \in [k], j \in [|\mathfrak{C}|/k]$
for each $\vec{x} \in L$ **do**
$\mathbf{A}_i^j \leftarrow \textbf{Decode}(\vec{x}, \mathfrak{C})$ ▷ Algorithm from Proposition 7.
$\vec{y} \leftarrow 1/\sin(\alpha) \cdot \left(\vec{x} - \cos(\alpha)\mathbf{A}_i^j\right)$ ▷ Residual vector of $\vec{x}$ in the filter of center $\mathbf{A}_i^j$
$L_i^j \leftarrow L_i^j \cup \{\vec{x}\}$; $R_i^j \leftarrow R_i^j \cup \{\vec{y}\}$
for each tuple-filter numbered $j \in [|\mathfrak{C}|/k]$ **do**
$Sol_{\vec{y}} \leftarrow \textbf{FindAllSolutionsWithinFilter}\left((R_i^j)_i, C'(\alpha)\right)$ ▷ Find all $(\vec{y}_i)_i \in R_1^j \times \cdots \times R_k^j$ satisfying $C'(\alpha)$
$Sol_{\Sigma\vec{x}} \leftarrow \left\{\sum_{i=1}^k \vec{x}_i : (\vec{y}_i)_i \in Sol_{\vec{y}}\right\}$ ▷ $\vec{x}_i \in L_i^j$ and $\vec{y}_i \in R_i^j$ share the same index in their respective lists
$L_{out} \leftarrow L_{out} \cup Sol_{\Sigma\vec{x}}$
return L_{out}

1. Prefiltering. We start by sampling a k-RPC $\mathfrak{C}$ (Defined in Part 3.1) of size $k \cdot 1/\mathcal{V}(\alpha)$. Its codewords are denoted $(\mathbf{A}_i^j)_{i,j}$ for $i \in [k]$ and $j \in [|\mathfrak{C}|/k]$ (we suppose these values are integers by simplicity). For a fixed $j \in [|\mathfrak{C}|/k]$ and for $i_1 \neq i_2 \in [k]$ we have $\langle \mathbf{A}_{i_1}^j | \mathbf{A}_{i_2}^j \rangle = -\frac{1}{k-1}$, that implies $\sum_{i=1}^k \mathbf{A}_i^j = \vec{0}$.

Once the code is sampled, we can start the so-called prefiltering step. For each vector $\vec{x} \in L$, we efficiently compute its nearest codeword in $\mathfrak{C}$ using the algorithm from Proposition 8. If it returns center $\mathbf{A}_i^j$, then we add $\vec{x}$ to its associated list L_i^j. We also compute $\vec{x}$'s residual vector $\vec{y} = 1/\sin(\alpha) \cdot \left(\vec{x} - \cos(\alpha)\mathbf{A}_i^j\right)$ (by Proposition 9) and we add it to list R_i^j. Given a residual vector in R_i^j, we will be able to recover its corresponding vector in L_i^j by just looking at the same index.

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

$$\text{NbFilters} := |\mathfrak{C}|/k = \mathcal{O}\left(\frac{1}{\mathcal{V}(\alpha)}\right). \tag{4}$$

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

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

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

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

Lemma 2. *With the same notations as before and for fixed $j \in [|\mathfrak{C}|/k]$, the expected number of tuples in the tuple-filter associated with the lists $R_1 \times \cdots \times R_k$ satisfying configuration $C'(\alpha)$ is on average*

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

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

Any subroutine with these inputs and outputs may suit the framework. For example, in the case $k = 2$, [CL21] describes a 2-sieve under this framework, where the subroutine uses quantum random walks to find the reducing pairs of vectors. We denote the time complexity of the subroutine **FindAllSolutionsWithinFilter** with parameters α and C by $T(\mathbf{FAS}_{C'(\alpha)})$.

3. Number of repeats. After searching all the solutions within every tuple-filters, we expect to find the following number of solutions:

$$|Sol_{all}| = |L|^k \cdot det(C)^{d/2}. \tag{5}$$

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

Lemma 3. *The number of repetitions in the while loop is*

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

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

Theorem 2. *Let $\alpha \in (0, \pi/2]$ be an angle and a configuration $C \in \mathbb{R}^{k \times k}$, and $C'(\alpha)$ the configuration on the residual vectors (See Lemma 1). Given an algorithm that solves the configuration problem $C'(\alpha)$ for k lists in time $T(\mathbf{FAS}_{C'(\alpha)})$, Algorithm 3 solves SVP in time*

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

where $\mathrm{NbRep}_{\alpha,C}$ *is given by Lemma 3 and* $\mathrm{NbFilters}_\alpha = \mathcal{O}(\frac{1}{\mathcal{V}(\alpha)})$ *by Equation 4.*

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

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

5 Classical sieving

We present here our 3-sieve and 4-sieve classical algorithms. In both cases, we use Theorem 2 so the only thing to explicit is the inner algorithm running in time $T(\mathbf{FAS}_{C'(\alpha)})$ as well as the parameters C and α. Actually, in both cases, the inner algorithm will use a classical 2-sieve algorithm so we first give formulas for the configuration problem with $k = 2$.

5.1 Classical 2-sieve

We present here the best-known algorithm for classical 2-sieve. While these are known results, we will need this analysis for our 3-sieve and 4-sieve algorithms. We can actually derive the best-known algorithms (in terms of asymptotic time and memory) from our framework.

Proposition 11. *Take $k = 2$, lists L_1, L_2 of random points of norm 1 with $|L_1| = |L_2|$, a target configuration $C = \begin{pmatrix} 1 & C_{12} \\ C_{12} & 1 \end{pmatrix}$. Let α st. $\mathcal{V}(\alpha) = \frac{1}{|L_1|}$.*

Algorithm 3 with parameter α constructs a list L_{out} of pairs of points $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2)$ with $(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \in L_1 \times L_2$ st. $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle \leq C_{12}$, in time T using memory M st.

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

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

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

$$T = \mathcal{O}\left(NbRep_{\alpha, C_{12}} \cdot \left(|L_1| + \frac{1}{\mathcal{V}(\alpha)} T(\mathbf{FAS}_{C'_{12}(\alpha)})\right)\right). \tag{6}$$

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

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

We take α st. $\mathcal{V}(\alpha) = \frac{1}{|L_1|}$ so Equation 6 becomes $T = \mathcal{O}\left(NbRep_{\alpha, C_{12}} \cdot |L_1|\right)$. Finally, from Lemma 3, we have

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

which allows us to conclude that

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

□

As a special case, we can take $|L_1| = |L_2| = 2^{0.2075d}$, $C_{12} = -1/3$ which gives $L_{out} = |L_1|$, $T = 2^{0.292d}$ and $M = |L_1|$ which is the best-known algorithm asymptotically.

5.2 Classical 3-sieve

So we now consider the case of $k = 3$. Our inner algorithms will construct the following intermediate lists:

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

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

Algorithm 4 FindAllSolutionsWithinFilter classical 3-sieve

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

Complexity of Algorithm 4.

Construction of the lists L_{12} and L_{13}. As a direct consequence of Proposition 11, we have:

Lemma 4. *Let T_{12} (resp. T_{13}) be the time to compute L_{12} (resp. L_{13}). Let α such that $|L_1| = 1/\mathcal{V}(\alpha)$. We have*

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

Construction of the lists $L_{23}(\vec{\boldsymbol{x}}_1)$. For a fixed $\vec{\boldsymbol{x}}_1$, notice that the lists $L_2(\vec{\boldsymbol{x}}_1)$ and $L_3(\vec{\boldsymbol{x}}_1)$ do not contain points uniformly distributed on the sphere since they have an inner-product constraint with $\vec{\boldsymbol{x}}_1$ so we cannot apply Proposition 11 directly. Fix $\vec{\boldsymbol{x}}_1 \in L_1$ and let $\vec{\boldsymbol{x}}_2 \in L_2$ and $\vec{\boldsymbol{x}}_3 \in L_3$. For simplicity of calculations, we consider the case where $\langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle = C_{12}, \langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_3 \rangle = C_{13}$ and $\langle \vec{\boldsymbol{x}}_2 | \vec{\boldsymbol{x}}_3 \rangle = C_{23}$. This approximation is justified from Heuristic 1. So we write

$$\vec{\boldsymbol{x}}_2 = C_{12}\vec{\boldsymbol{x}}_1 + \sqrt{1 - C_{12}^2}\vec{\boldsymbol{y}}_2 \quad ; \quad \vec{\boldsymbol{x}}_3 = C_{13}\vec{\boldsymbol{x}}_1 + \sqrt{1 - C_{13}^2}\vec{\boldsymbol{y}}_3 \tag{7}$$

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

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

which implies

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

We can now use Proposition 11 to give the running time of computing $R_{23}(\vec{\boldsymbol{x}}_1)$, which gives the running time $T_{23}(\vec{\boldsymbol{x}}_1)$ of computing $L_{23}(\vec{\boldsymbol{x}}_3)$. Let $Y = \begin{pmatrix} 1 & Y_{23} \\ Y_{23} & 1 \end{pmatrix}$ and let α" st. $\mathcal{V}(\alpha') = \frac{1}{|L_2(\vec{\boldsymbol{x}}_1)|}$. We have

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

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

$$T_{23} = |L_1| T_{23}(\vec{\boldsymbol{x}}_1) = \mathcal{O}\left(|L_1| NbRep_{\alpha',Y} \cdot |L_2(\vec{\boldsymbol{x}}_1)|\right) \tag{8}$$

$$= |L_1||L_2(\vec{\boldsymbol{x}}_1)|^2 \frac{\left(1 - Y_{23}^2\right)^{d/2}}{\left(1 - Y'_{23}(\alpha')^2\right)^{d/2}} \tag{9}$$

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

Proposition 12. *Let $|L_1|$ a list size and C a 3×3 configuration matrix with negative non-diagonal entries. Let $|L_2(\vec{\boldsymbol{x}}_1)| = |L_1|(1 - C_{12}^2)^{d/2}$. Let α' st. $\mathcal{V}(\alpha') = \frac{1}{|L_2(\vec{\boldsymbol{x}}_1)|}$. Algorithm 4 solves $\mathbf{FAS}_3^c(|L_1|, C)$ in time $T_{12} + T_{13} + T_{23}$ with*

$$T_{12} = T_{23} = \mathcal{O}\left(|L_1|^2(1 - C_{12}^2)^{d/2}\right)$$

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

where

$$Y_{23} = \frac{1}{\sqrt{1 - C_{12}^2}\sqrt{1 - C_{13}^2}} \cdot (C_{23} + C_{12}C_{13})$$

$$Y'_{23}(\alpha) = \frac{1}{\sin^2(\alpha)}\left(Y_{23} + \cos^2(\alpha')\right).$$

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

Complexity of the classical 3-sieve. The above was the analysis of the classical 3-sieve after the first filtering. We now apply Theorem 2 with $k = 3$ in order to obtain the running of our classical 3-sieve algorithm within our framework.

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

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

and

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

where $|L_1| = |L| \cdot \mathcal{V}(\alpha)$, $|L_{12}|, |L_{123}|$ can be taken from Proposition 6 and $\mathbf{FAS}_3^c(|L_1|, C) = T_{12} + T_{13} + T_{123}$ where each T_{12}, T_{13}, T_{123} can be taken from Proposition 12.

Proposition 13. *There exists a classical algorithm for SVP using 3-sieve that runs in time $2^{0.338d+o(d)}$ and uses memory $2^{0.1887d+o(d)}$.*

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

$$NbRep_{\alpha,C} = 2^{0.070d} \quad ; \quad \frac{1}{\mathcal{V}(\alpha)} = 2^{0.055d} \quad ; \quad T\left(\mathbf{FAS}_3^c(|L_1|, C')\right) = 2^{0.213d}$$

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

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

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

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

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

Table 2: Time complexity of our classical 3-sieving algorithm for a fixed memory constraint. α is the optimal angle used in the first prefiltering. Also see Figure 2a for a plot corresponding to this algorithm.

5.3 Classical 4-sieve

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

1. Construct $L_{12} = \{(\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2) \in L_1 \times L_2 : \langle \vec{\boldsymbol{x}}_1 | \vec{\boldsymbol{x}}_2 \rangle \leq C_{12}\}$ and $L_{34} = \{(\vec{\boldsymbol{x}}_3, \vec{\boldsymbol{x}}_4) \in L_3 \times L_4 : \langle \vec{\boldsymbol{x}}_3 | \vec{\boldsymbol{x}}_4 \rangle \leq C_{34}\}$.
2. Construct $L_{1234} = \{((\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2), (\vec{\boldsymbol{x}}_3, \vec{\boldsymbol{x}}_4)) \in L_{12} \times L_{34} : (\vec{\boldsymbol{x}}_1, \vec{\boldsymbol{x}}_2, \vec{\boldsymbol{x}}_3, \vec{\boldsymbol{x}}_4)$ satisfies configuration $C\}$.

Using these lists, we consider the following algorithm:

Algorithm 5 FindAllSolutionsWithinFilter classical 4-sieve

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

We then use the above algorithm as the **FindAllSolutionsWithinFilter** subroutine in Algorithm 3 to describe our entire algorithm for 4-sieve. The algorithm presented here is usually inefficient in memory because the lists L_{12} and L_{13} are large. However, thanks to our initial α-filtering, we start from smaller lists L_1, L_2, L_3, L_4 so the intermediate lists will be small as well.

Complexity of Algorithm 5.

Lemma 5. *Let T_{12} be the time to compute L_{12} (which is also the time to compute L_{34} by symmetry). Let α st. $\mathcal{V}(\alpha) = \frac{1}{|L_1|}$. Then*

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

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

$$|L_{12}| = |L_1|^2 \cdot \left(1 - C_{12}^2\right)^{d/2} \tag{10}$$

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

Lemma 6. *Let $\vec{z}_{12} \in L_{12}$ and $\vec{z}_{34} \leq L_{34}$. If $Angle(\vec{z}_{12}, \vec{z}_{34}) = \arccos\left(\frac{\sin^2(\alpha)}{2R'^2} - 1\right)$ then $\|\vec{z}_{12} + \vec{z}_{34}\|^2 \leq \sin^2(\alpha)$.*

Proof. We write

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

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

Lemma 7. *Let T_{1234} be the time to compute L_{1234}. Let $Y = \frac{r_0}{4+4C_{12}} - 1$. Let α' st. $\mathcal{V}(\alpha') = \frac{1}{|L_{12}|}$. We have*

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

with $Y'(\alpha') = \frac{1}{\sin^2(\alpha)}\left(T + \cos^2(\alpha')\right)$.

By combining the above 2 propositions, we have

Theorem 4. *Algorithm 5 runs in time $T = 2T_{12} + T_{1234}$ where T_{12} and T_{1234} can be taken respectively from Lemma 5 and Lemma 7.*

To conclude, we can plug this theorem again in Theorem 2 to get our results. Recall that we work with 4-tuples of residual vectors after an initial α-filtering so we look for 4-tuples of residual points $(\vec{y}_1, \vec{y}_2, \vec{y}_3, \vec{y}_4)$ st. $\|\vec{y}_1 + \vec{y}_2 + \vec{y}_3 + \vec{y}_4\| \leq \frac{1}{\sin(\alpha)}$ (see Equation 3). This means we take $r_0 = \frac{1}{\sin(\alpha)}$. Regarding memory requirements, we have that the memory M of our algorithm satisfies $M = \max\{|L_1|, |L_{12}|, |L_{1234}|\}$.

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

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

Table 3: Time complexity of our classical 4-sieving algorithm for a fixed memory constraint. α is the optimal angle used in the prefiltering. Also see Figure 2b for a plot corresponding to this algorithm.

6 Quantum sieving

We now study how our framework impacts quantum algorithms. In the quantum setting, Theorem 2 also applied and once again, we only need to describe the running time and amount of memory used for the inner configuration problem.

The input lists L_i are stored classically and are assumed to be quantumly accessible, *i.e.* for any given list L, we can efficiently construct the uniform superposition over all its elements $|\psi_L\rangle := \frac{1}{\sqrt{|L|}} \sum_\ell |\ell\rangle |L[\ell]\rangle$. In the following, we will not necessarily write the first register for simplicity[2].

6.1 Quantum 3-sieve

For the **FindAllSolutionsWithinFilter** quantum subroutine in case $k = 3$, we start with classical lists L_1, L_2, L_3 that are quantumly accessible. The algorithm outputs a list containing all triples in $L_1 \times L_2 \times L_3$ satisfying a given target configuration C.

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

As a reminder (See Proposition 6), given a configuration C, we use the following notation: for $i, j \in [k]$, $i \neq j$ and $\vec{\boldsymbol{y}}_j \in L_j$,

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

Algorithm 6 FindAllSolutionsWithinFilter quantum 3-sieve

Require: lists L_1, L_2, L_3 of vectors i.i.d. in $\mathcal{S}^{d-1}$ with $|L_1| = |L_2| = |L_3|$; a target configuration $C \in \mathbb{R}^{3\times 3}$.

Ensure: list L_{out} containing all 3-triples in $L_1 \times L_2 \times L_3$ satisfying configuration C.

$L_{out} := \emptyset$

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

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

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

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

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

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

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

return L_{out}

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

Complexity of Algorithm 6. We first analyse the complexity to find one solution during one single iteration from the while-loop.

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

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

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

It only keeps in the quantum superposition the elements $\vec{\boldsymbol{y}}_2 \in L_2$ such that $\langle \vec{\boldsymbol{y}}_1 | \vec{\boldsymbol{y}}_2 \rangle \leq C_{12}$ for each superposed $\vec{\boldsymbol{y}}_1$ from the first register. So the state ends up with a quantum superposition of all pairs in $L_1 \times L_2$ eligible to form the beginning of a triple-solution. This application of Grover's algorithm takes time $T_2 = \sqrt{\frac{|L_2|}{|L_2(\vec{\boldsymbol{y}}_1)|}} = (1 - C_{12})^{-d/4}$ by Proposition 6.

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

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

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

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

Lemma 8. *The operation $\mathcal{A} : |0\rangle |0\rangle |0\rangle \to |\psi_{L_1}\rangle \left|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\right\rangle \left|\psi_{L_3(\vec{\boldsymbol{y}}_1)}\right\rangle$ is repeated T_{AA} times inside the amplitude amplification to construct state $|\psi_{Sol}\rangle$ (with probability $1 - o(1)$), where*

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

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

Finally, the probability to measure a solution is thus $p = |Sol|/|L_1| \cdot \frac{1}{L_2(\vec{\boldsymbol{y}}_1)} \frac{1}{L_3(\vec{\boldsymbol{y}}_1)}$. By Theorem 3, the number of iterations of amplitude amplification is $\mathcal{O}\left(1/\sqrt{p}\right)$, hence the top line. The bottom line is obtained by expressing the sizes of $L_2(\vec{\boldsymbol{y}}_1)$ and $L_3(\vec{\boldsymbol{y}}_1)$ using Proposition 6. □

Subroutine complexity.

Proposition 14 (FindAllSolutionsWithinFilter quantum 3-sieve). *Let $|L_1|$ a list size and C a 3×3 configuration matrix with negative non-diagonal entries. Algorithm 6 solves $\mathbf{FAS}_3^q(|L_1|, C)$ in time $|Sol| \cdot (T_2 + T_3) \cdot T_{AA}$ where*

$$|Sol| = |L_1|^3 \cdot \det(C)^{d/2}$$

$$T_2 = (1 - C_{12})^{-d/4} \quad ; \quad T_3 = (1 - C_{13})^{-d/4}$$

$$T_{AA} = \mathcal{O}\left(\sqrt{|L_i^3|/|Sol|} \cdot (1 - C_{12})^{d/4} (1 - C_{13})^{d/4}\right)$$

After simplification, the time complexity of Algorithm 6 can be written

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

This algorithm uses classical memory $|L_1|$ and quantum memory poly(d) qubits.

Complexity of the quantum 3-sieve. The above was the analysis of the algorithm we use as the subroutine **FindAllSolutionsWithinFilter** in Algorithm 3 for quantum 3-sieve. The lists given in input of Algorithm 6 are then the lists of residual vectors R_1, R_2, R_3, which are of size $|R_1| = |L_1| = |L| \cdot \mathcal{V}(\alpha)$; and it return residual vectors that satisfy the target configuration $C'(\alpha)$. Using Theorem 2 in the case $k = 3$, we recover the overall time complexity of our quantum 3-sieve algorithm.

Theorem 5. *There is a quantum algorithm with parameter α that solves the 3-sieve problem for a configuration $C \in \mathbb{R}^{3\times 3}$ and lists of size $|L|$, that runs in time*

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

where $|L_1| = |L| \cdot \mathcal{V}(\alpha)$ and $T(\mathbf{FAS}_3^q(|L_1|, C'(\alpha)))$ given by Proposition 14. This algorithm uses quantum-accessible classical memory $M = |L|$ and quantum memory poly(d).

Minimal memory parameters.

Proposition 15. *There is a quantum algorithm that solves SVP in dimension d using 3-sieve that runs in time $T = 2^{0.3098d+o(d)}$, quantum-accessible classical memory $M = 2^{0.1887d+o(d)}$ and poly(d) quantum memory.*

Proof. We take a balanced configuration C with $C_{12} = C_{13} = C_{23} = -1/3$, $\alpha = 1.2343$rad and $|L| = 2^{0.1887d} = M$. We apply Proposition 5: We write $C'_{12}(\alpha) = C'_{13}(\alpha) = C'_{23}(\alpha) \approx -0.31$ and $|L_1| = |L| \cdot \mathcal{V}(\alpha) = 2^{0.1055d}$. We have

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

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

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

$\log_2(\text{Memory})/d$	0.1887	0.189	0.190	0.1907
$\log_2(\text{Time})/d$	0.3098	0.3073	0.3056	0.3053
α (rad)	1.2346	1.2341	1.2336	1.2331

Table 4: Time complexity of our quantum 3-sieving algorithm for a fixed memory constraint. α is the optimal angle used in the prefiltering. Also see Figure 3a for a plot corresponding to this algorithm.

6.2 Quantum 4-sieve

This algorithm and its analysis are very similar to Algorithm 6. As previously, we first analyse the complexity to find one solution during one single iteration from the while-loop.

Complexity of Algorithm 7.

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

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

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

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

Algorithm 7 FindAllSolutionsWithinFilter quantum 4-sieve

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

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

$L_{out} := \emptyset$

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

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

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

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

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

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

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

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

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

return L_{out}

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

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

Performing this Grover's algorithm takes time $T_3 = \sqrt{\frac{|L_3|}{|L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)|}}$. Proposition 6 gives $|L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)| = |L_3| \cdot \left(\frac{\det(C[1,2,3])}{\det(C[1,2])}\right)^{d/2}$. Note that these notations for partial configurations are given in Definition 4. So we can rewrite $T_3 = \left(\frac{\det(C[1,2,3])}{\det(C[1,2])}\right)^{-d/4}$.

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

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

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

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

Lemma 9. *The operation* $|0\rangle|0\rangle|0\rangle|0\rangle \to |\psi_{L_1}\rangle \left|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\right\rangle \left|\psi_{L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\right\rangle \left|\psi_{L_4(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\right\rangle$ *is repeated* T_{AA} *times inside the amplitude amplification to construct state* $|\psi_{Sol}\rangle$ *(with probability* $1 - o(1)$*), where*

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

where notation $C[I]$ *with a set of indexes* I *was introduced in Definition 4.*

Proof. The reasoning is the same as for the proof of Lemma 8. Performing a measurement of state $|\psi_{L_1}\rangle \left|\psi_{L_2(\vec{\boldsymbol{y}}_1)}\right\rangle \left|\psi_{L_3(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\right\rangle \left|\psi_{L_4(\vec{\boldsymbol{y}}_1,\vec{\boldsymbol{y}}_2)}\right\rangle$ gives a 4-tuple solution $(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3, \vec{\boldsymbol{y}}_4)$ with some probability p, we are going to specify. The probability of measuring a good $\vec{\boldsymbol{y}}_1$ is $|Sol|/|L_1|$. Then given a $\vec{\boldsymbol{y}}_1$, a triple $(\vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3, \vec{\boldsymbol{y}}_4)$ forms the solution together with $\vec{\boldsymbol{y}}_1$ with probability $1/(|L_2(\vec{\boldsymbol{y}}_1)| \cdot |L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)| \cdot |L_4(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)|)$.

Finally, the probability of success to measure a solution is thus $p = \frac{|Sol|}{|L_1|} \cdot 1/\left(|L_2(\vec{\boldsymbol{y}}_1)| \cdot |L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)| \cdot |L_4(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)|\right)$. By Theorem 3, the number of iterations of amplitude amplification is $\mathcal{O}\left(1/\sqrt{p}\right)$, hence the top line. The bottom line is obtained by expressing the sizes of $L_2(\vec{\boldsymbol{y}}_1)$, $L_3(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)$ and $L_4(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2)$ using Proposition 6. □

Measurement gives a 4-tuple $(\vec{\boldsymbol{y}}_1, \vec{\boldsymbol{y}}_2, \vec{\boldsymbol{y}}_3, \vec{\boldsymbol{y}}_4)$ solution to the configuration problem. We need to repeat this whole process until we find all the solutions at number $|Sol|$.

Notice that the same operations are performed over L_3 and over L_4, which implies that an optimal configuration will necessarily respect the symmetry $C_{13} = C_{14}$ and $C_{23} = C_{24}$.

In the end, this subroutine **FindAllSolutionsWithinFilter** runs in time

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

and this leads to the following theorem.

Proposition 16. *Given lists* $L_1, L_2, L_3, L_4 \subset \mathcal{S}^{d-1}$ *of same size* $|L_i|$ *with i.i.d. uniformly random vectors, and a configuration* $C \in \mathbb{R}^{4\times 4}$ *with* $C_{13} = C_{14}$ *and* $C_{23} = C_{24}$*, there exists an algorithm that finds all the* $|Sol|$ *4-tuples in* $L_1 \times L_2 \times L_3 \times L_4$ *satisfying configuration* C *in time*

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

Complexity of the quantum 4-sieve. The above was the analysis of the quantum 4-sieve after the prefiltering. We use this algorithm as the subroutine in our framework for $k = 4$. Using Theorem 2 in this case, we recover the overall time complexity of our quantum 4-sieve algorithm.

Theorem 6. *There is a quantum algorithm with parameter α that solves the 3-sieve problem for a configuration C and lists of size $|L|$ that runs in time T with*

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

and uses quantum-accessible classical memory $M = |L|$ and quantum memory poly(d), and where $|L_1| = |L| \cdot \mathcal{V}(\alpha)$ and $T(\boldsymbol{FAS}_4^q(|L_1|, C'(\alpha)))$ given by Proposition 16.

Minimal memory parameters.

Proposition 17. *There is a quantum algorithm that solves SVP in dimension d using 4-sieve that runs in time $T = 2^{0.3276d+o(d)}$ using quantum-accessible classical memory $M = 2^{0.1724d+o(d)}$ and quantum memory poly(d).*

Proof. We take a balanced configuration C with $C_{i,j} = -1/4$ for $i \neq j$, $\alpha \approx 1.3131$rad and $|L| = 2^{0.1724d} = M$. We apply Theorem 6: We write $C'_{i,j} \approx -0.244$ for $i \neq j$ and $|L_1| = |L| \cdot \mathcal{V}(\alpha) = 2^{0.124d}$. We have

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

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

Time-optimizing parameters.

Proposition 18. *There exists an algorithm that solves SVP in dimension d in time $T = 2^{0.3120d+o(d)}$ using quantum-accessible classical memory $M = 2^{0.1813d+o(d)}$ and quantum memory poly(d).*

Proof. We take a configuration C with $C_{12} \approx -0.3859$, $C_{13} = C_{14} \approx -0.2294$, $C_{23} = C_{24} \approx -0.2297$ and $C_{34} \approx -0.1998$. We take $\alpha \approx 1.313$rad and $|L| = 2^{0.1813d} = M$. We apply Proposition 6: We write $C'_{12} \approx -0.3859$, $C'_{13} = C'_{14} \approx -0.2210$, $C'_{23} = C'_{24} \approx -0.2215$ and $C'_{34} \approx -0.1892$. We set $|L_1| = |L| \cdot \mathcal{V}(\alpha) = 2^{0.1259d}$. We have

$$\mathrm{NbRep}_{\alpha,C} = 2^{0.1254d} \quad ; \quad \frac{1}{\mathcal{V}(\alpha)} = 2^{0.0554d} \quad ; \quad \mathbf{FAS}_4^q(|L_1|, C') = 2^{0.1312d}.$$

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

$\log_2(\text{Memory})/d$	0.1724	0.175	0.180	0.1813
$\log_2(\text{Time})/d$	0.3276	0.3153	0.3127	0.3120

Table 5: Time complexity of our quantum 4-sieving algorithm for a fixed memory constraint. For the prefiltering, we have an optimal $\alpha \approx 1.3131$rad. Also see Figure 3b for a plot corresponding to this algorithm.

Code used for our results. All our results have been obtained using SageMath and the code is available on `https://github.com/johanna-loyer/3-4-sieve`.

References

[Bec+16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. "New directions in nearest neighbor searching with applications to lattice sieving". In: *Proc. of the 2016 Annual ACM-SIAM Symposium on Discrete Algorithms* (2016).

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. "Tuple lattice sieving". In: *LMS J. Comput. Math.* (2016), pp. 146–162. URL: `https://www.cambridge.org/core/services/aop-cambridge-core/content/view/C1CE6384DEC54330AEFB2A4D38190094/S1461157016000292a.pdf/tuple_lattice_sieving.pdf`.

[Bon+22] Xavier Bonnetain, André Chailloux, André Schrottenloher, and Yixin Shen. "Finding many Collisions via Reusable Quantum Walks". In: (2022). URL: `https://doi.org/10.48550/arXiv.2205.14023`.

[Bos+18] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J.M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM". In: *IEEE* (2018). URL: `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8406610`.

[Bra+02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. "Quantum amplitude amplification and estimation". In: *Quantum Computation and Quantum Information: A Millennium Volume* (2002), 305:53–74. URL: `https://arxiv.org/pdf/quant-ph/0005055.pdf`.

[CL21] André Chailloux and Johanna Loyer. "Lattice sieving via quantum random walk". In: *ASIACRYPT* (2021). URL: `https://eprint.iacr.org/2021/570.pdf`.

[Duc+19] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. "CRYSTALS-Dilithium, Algorithm Specifications and Supporting Documentation". In: *NIST* (2019). URL: `https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf`.

[FP85] Ulrich Fincke and Michael Pohst. "Improved methods for calculating vectors of short length in a lattice". In: *Mathematics of Computation* 44(170) (1985), pp. 463–471. URL: `http://www.jstor.org/stable/2007966`.

[Gro96] Lov Grover. "A fast quantum mechanical algorithm for database search". In: *Proc.28th Annual ACM Symposium on the Theory of Computing STOC* (1996), pp. 212–219. URL: https://dl.acm.org/doi/pdf/10.1145/237814.237866.

[Hei21] Max Heiser*. "Improved Quantum Hypercone Locality Sensitive Filtering in Lattice Sieving". In: *preprint* (2021). URL: https://eprint.iacr.org/2021/1295.pdf.

[HK17] Gottfried Herold and Elena Kirshanova. "Improved algorithms for the approximate k-list problem in Euclidean norm". In: *Public-Key Cryptography* (2017), pp. 16–40. URL: https://link.springer.com/chapter/10.1007/978-3-662-54365-8_2.

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. "Speed-ups and time–memory trade-offs for tuple lattice sieving". In: *IACR International Workshop on Public Key Cryptography* (2018), pp. 407–436. URL: https://eprint.iacr.org/2017/1228.pdf.

[Kan83] R. Kannan. "Improved algorithms for integer programming and related lattice problems". In: *Proceedings of the 15th Symposium on the Theory of Computing (STOC), ACM Press* (1983), pp. 99–108.

[Kir+19] 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". In: *ASIACRYPT* (2019). URL: https://eprint.iacr.org/2019/1016.pdf.

[Kle00] Philip Klein. "Finding the closest lattice vector when it's unusually close". In: *SODA* (2000), pp. 937–941. URL: http://128.148.32.110/research/pubs/pdfs/2000/Klein-2000-FCL.pdf.

[Laa16] Thijs Laarhoven. "Search problems in cryptography, From fingerprinting to lattice sieving". PhD thesis. Eindhoven University of Technology, 2016. URL: https://research.tue.nl/files/14673128/20160216_Laarhoven.pdf.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. "Faster exponential time algorithms for the shortest vector problem". In: *SODA* (2010), pp. 1468–1480. URL: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973075.119.

[NV08] P.Q. Nguyen and T. Vidick. "Sieve algorithms for the shortest vector problem are practical". In: *J. Math. Crypt. 2* (2008), pp. 181–207. URL: https://doi.org/10.1515/JMC.2008.009.

[Poh81] Michael E. Pohst. "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications". In: *ACM SIGSAMBulletin* 15(1) (1981), pp. 37–44.