# Sieving Algorithms for the Shortest Vector Problem Internship Report

Bogdan Ursu, supervised by Prof. Serge Vaudenay, LASEC EPFL

01/02/2016 - 30/06/2016

## General context

A (full-rank) lattice of dimension $n$ is a discrete subgroup of $\mathbb{R}^n$. Alternately, a lattice can be regarded from a geometrical point of view as the intersection points of a $n$-dimensional grid. Due to their combinatorial properties, lattices have been used to solve a multitude of important problems such as integer programming or polynomial factorization. They are also employed in designing cryptographic primitives seen as secure even against quantum computers, an important area of current research in cryptography.

## Problem studied

Finding the intersection point in the grid which is closest to the origin is a problem known as the shortest vector problem (SVP) and is perhaps the most famous one in the theory of lattices. This is mainly because the security of a large variety of cryptosystems, digital signatures and hash functions depends on its hardness. From a complexity theory perspective, Ajtai et al. proved in [4] that recovering the shortest non-zero vector with respect to the euclidean norm is NP-hard, under some special type of randomized reductions. An even stronger result is due to Micciancio in [22], in which NP-hardness is shown even for certain constant approximation factors. In cryptography, one of the main advantages of using approximate versions of the shortest vector problem up to polynomial factors is given by average-case hardness guarantees, as proven in [3].

**Known algorithms** Notwithstanding the NP-hardness result of the problem, an even more interesting topic relates to designing the algorithms necessary for solving it. From a security perspective, these algorithms are relevant as they indicate minimal thresholds for the security parameters of cryptographic primitives based on the shortest vector problem (SVP). One of the first provable algorithms is Kannan Enumeration, which runs in time $2^{\Omega(n \log(n))}$ and polynomial space with respect to the dimension $n$ of the lattice. Algorithms that work in $2^{O(n)}$ time and space include the Voronoi cell algorithm [23], the Discrete Gaussian Sampling [2] and sieving-based algorithms [5, 25]. Nevertheless, in practice, heuristic modifications of the aforementioned algorithms often yield better results. The record dimension reached in the SVP challenge [13] is $146$, held by a heuristic variant of Schnorr's Random Sampling Reduction.

Historically, the first sieving heuristic proposed is the NV-Sieve [26], followed by the experimentally faster GaussSieve [25]. Unfortunately, no theoretical time complexity bounds exist for the later. Further improvements of the NV-Sieve and GaussSieve algorithms have been put forward recently, using locality sensitive hash functions (LSH). Table 1 gives more precise information on the known provable algorithms for the shortest vector problem, along with practical heuristics. Presently, sieving algorithms and their heuristic variants for solving the SVP reach dimension $116$ in the SVP challenge [13]. Due to their competitiveness, the primary objective of this internship is the study of existing sieving-based algorithms.

## Contributions

This report surveys some of the most important provable algorithms and heuristics of the sieving family for solving the shortest vector problem, along with its approximation variant, which will be defined formally later. Firstly, we give more details on the existing proofs and explain why we can use values of the $\epsilon$ parameter which are greater than the length of the shortest non-zero vector. Secondly, our findings include a better adaptation of the AKS algorithm for approximate SVP. In fact, our new analysis of this adaptation improves on the previously known time and space

Table 1: Time and space complexities of known SVP solvers. Except for the overlattice sieve, the heuristics enumerated all belong to the sieving family.

| Algorithm | Time | Memory | |
|---|---|---|---|
| Kannan Enumeration [15] | $2^{\frac{n\log(n)}{2\epsilon}+o(n)}$ | poly(n) | Proven |
| Voronoi Cell [24] | $2^{2n}$ | $2^{n}$ | Proven |
| AKS [5, 26, 25] | $2^{3.397n+o(n)}$ | $2^{1.984n+o(n)}$ | Proven |
| AKS-Birthday [14] | $2^{2.64791n+o(n)}$ | $2^{1.32396n+o(n)}$ | Proven |
| ListSieve-Birthday [27] | $2^{2.465n+o(n)}$ | $2^{1.233n+o(n)}$ | Proven |
| Discrete Gaussian [2] | $2^{n+o(n)}$ | $2^{n+o(n)}$ | Proven |
| GaussSieve [25] | ? | $2^{0.2075n+o(n)}$ | Heuristic |
| NV-Sieve [26] | $2^{0.415n+o(n)}$ | $2^{0.2075n+o(n)}$ | Heuristic |
| Levelled NV-Sieve [34] | $2^{0.3778n+o(n)}$ | $2^{0.2833n+o(n)}$ | Heuristic |
| Overlattice Sieve [9] | $2^{0.3774n+o(n)}$ | $2^{0.2925n+o(n)}$ | Heuristic |
| LSH NV-Sieve [8] | $2^{0.292n+o(n)}$ | $2^{0.292n+o(n)}$ | Heuristic |

complexity results for solving approximate-SVP. The idea of this new adaptation is to use less centers during the sieving procedure, which we achieve by relaxing the previous requirements on their properties, while preserving correctness. An article to describe specifically these new results is currently in the works. The second part of the report is dedicated to the survey of several state of the art sieving heuristics in detail, along with our attempts at improving them.

## Arguments supporting its validity

A first argument is that our new analysis of the AKS for solving approximate-SVP improves on both time and space complexity bounds for any approximation factor larger than $3.37$. Using the conceptually simpler AKS algorithm for approximate-SVP also seems to suggest that, for asymptotically large approximation factors, provable bounds on these sieving algorithms are only tied to known bounds on the kissing constant. The kissing constant in dimension $n$ stems from the theory of sphere packings and represents the maximal number of vectors in $\mathbb{R}^n$, for which all the common angles are at least $60°$.

Concerning our survey of sieving heuristics, we provide the reader with an unified approach and we also give more precise details. As a first example, Heuristic 3 is implicitly used in [26], but we give an explicit description along with a case in which the heuristic does not hold. A second example is related to the LSH Sieving in Lemma 14, where it is explained why small errors in the false positive probability lead only to a constant increase in the list sizes.

## Summary and future work

After giving several basic theoretical results on lattices in Section 1, we move on to the description of the first provably correct sieving algorithm which we denote by AKS in Section 2. We adapt this algorithm for solving the approximate shortest vector problem in Section 3. The second part of this work describes known state of the art results relating to heuristic sieving algorithms in Section 4 and the known enhancements of these heuristics using locality sensitive hash functions (LSH) in Section 5. We also describe a possible idea for improvement using more than two layers of hash functions. It is unclear how to mathematically optimize the complexity in these cases, therefore we leave this topic for future work. Another promising direction is to apply LSH to the tuple sieves described in the very recent article of [7]. In particular, the naive application of the LSH will yield an immediate improvement, but it remains to be seen precisely if it is possible to obtain even better results. On a different line of thought, we may also ask whether it is possible to combine the Discrete Gaussian Sampler of [2] with the NV-Sieve. Such an achievement would yield a provable algorithm. Indeed, it may be possible to obtain covering results for the corona when the vectors follow a discrete Gaussian distribution. Unfortunately, the main difficulty would be to control the distribution of lattice vectors outputted by each execution of the sieve.

# 1 Background and Notation

This section comprises several fundamental definitions and results related to basic probability theory, lattices and $n$-dimensional geometry in $\mathbb{R}^n$.

**Lemma 1** (**A corollary of the Chernoff Bound**). *Let $X_1, ..., X_n$ be independent random Bernoulli variables which take values in $\{0, 1\}$. We define $X \stackrel{def}{=} \sum_{i=1}^{n} X_i$ and $0 < \delta < 1$. Then, it holds that $X$ is not much smaller than $E[X]$, with exponentially small tail:*

$$P(X \geq (1-\delta)E[X]) \geq 1 - \frac{1}{e^{\delta^2 E[X]/2}}.$$

**Hyperballs and Coronas** By $B_n(\mathbf{x}, R)$ we denote the $n$-dimensional ball of radius $R$ centered in $\mathbf{x}$. When the ball is centered in the origin we will use the shorthand notation $B_n(R)$. For $0 < \gamma < 1$, we also define the corona $C_n(\gamma, R) \stackrel{def}{=} \{\mathbf{x} \in \mathbb{R}^n | \gamma R \leq \|\mathbf{x}\| \leq R\}$, with $C_n(\gamma)$ as shorthand notation when the radius is 1. In the following, let us give a formula for the volume of a hyperball, depending on whether its dimension is even or odd:

$$vol(B_{2k}(R)) = \frac{\pi^k}{k!} R^{2k} \qquad vol(B_{2k+1}(R)) = \frac{2(k!)(4\pi)^k}{(2k+1)!} R^{2k+1}$$

**Hyperspherical caps** From now on, we denote by $\theta(\mathbf{x}, \mathbf{y})$ the common angle between the vectors $\mathbf{x}$ and $\mathbf{y}$, which is always between 0 and $\pi$. We define a hyperspherical cap of dimension $n$ and height $h$ in $B_n(R)$ as:

$$S_n(\mathbf{x}, h, R) \stackrel{def}{=} \{\mathbf{y} \in B_n(R) : \|\mathbf{y}\| = R \text{ and } \theta(\mathbf{x}, \mathbf{y}) \leq \arccos\Big(1 - \frac{h}{R}\Big)\}.$$

$S_n(\mathbf{x}, h, R)$ is therefore a portion of the sphere which is centered in $\mathbf{x}$ and for which every composing vector is at an angle at most $\arccos\Big(1 - \frac{h}{R}\Big)$ from $\mathbf{x}$. Similarly, it is possible to define a spherical cap in terms of the maximum angle made by a composing vector with the centering vector: $S_n(\mathbf{x}, \varphi, R) \stackrel{def}{=} \{\mathbf{y} \in B_n(R) : \|\mathbf{y}\| = R \text{ and } \theta(\mathbf{x}, \mathbf{y}) \leq \varphi\}$.

Remark the abuse of notation, in the following we will use the appropriate definition depending on whether the second parameter is an angle or not. The next lemma provides an estimate on the relative volume of a hyperspherical cap with respect to the volume of its hypersphere.

**Lemma 2.** *[8, Lemma 2.1][25, see also Lemma 4.1]: Let $\mathbf{x}$ be a unit vector and consider the hyperspherical cap $S_n(\mathbf{x}, h, R)$ of height $h$ of a hyperball of radius $R$. Then the ratio of its volume with respect to the volume of the $B_n(R)$ hyperball satisfies:*

$$\frac{vol(S_n(\mathbf{x}, h, R))}{vol(B_n(R))} = poly(n)\Big(1 - \Big(1 - \frac{h}{R}\Big)^2\Big)^{\frac{n}{2}}.$$

*The expression $\frac{vol(S_n(\mathbf{x}, \varphi, R))}{vol(B_n(R))} = poly(n)\sin(\varphi)^n$ for the spherical cap expressed in terms of the angle $\varphi$ follows directly.*

**Definition 1** (**Lattices and the shortest vector**). *A lattice $\mathcal{L}$ of dimension $n$ is a discrete subgroup of $\mathbb{R}^n$, or equivalently, the set $\mathcal{L}(\mathbf{b_1}, ..., \mathbf{b_n}) = \{\sum_{i=1}^{n} x_i \mathbf{b_i} : x_i \in \mathbb{Z}\}$ of all integral combinations of n linearly independent vectors $\mathbf{b_1}, ..., \mathbf{b_n}$ in $\mathbb{R}^n$. The vectors $\mathbf{b_1}, ..., \mathbf{b_n}$ will be referred to as the basis of the lattice. We denote by $\lambda_1(\mathcal{L})$ the length of the shortest non-zero vector in $\mathcal{L}$.*

**Definition 2** (**Fundamental parallelepiped**). *Let $\mathcal{L}$ be a lattice obtained from the basis $\mathbf{b_1}, ..., \mathbf{b_n}$. When the basis $\mathbf{b_1}, ..., \mathbf{b_n}$ forms the columns of a basis matrix $B$, the fundamental parallelepiped $P(B)$ of the lattice is defined as: $P(B) \stackrel{def}{=} \{B\mathbf{x} : \mathbf{x} \in [0, 1)^n\}$.*

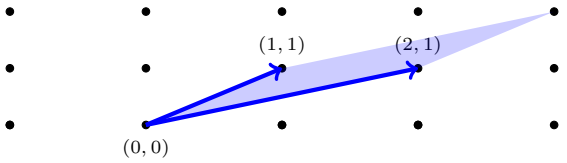


Figure 1: A basis of $\mathbb{Z}^2$. The shaded area is the fundamental parallelepiped.[28]

**Definition 3** (**Shortest vector problem - SVP**). *Given a basis* $\mathbf{b_1}, ..., \mathbf{b_n}$ *of a lattice* $\mathcal{L}$*, find a shortest non-zero vector of* $\mathcal{L}$*.*

**Definition 4** ($\mu$**-Approximate SVP -** $\mu$**SVP**). *Given a basis* $\mathbf{b_1}, ..., \mathbf{b_n}$ *of a lattice* $\mathcal{L}$*, find a non-zero lattice vector of length at most* $\mu\lambda_1(\mathcal{L})$*.*

In the following sections, it will be necessary to motivate at some point that the number of lattice vectors in a hyperball of a specific radius is large enough. In order to accomplish this we state the very well-known Gaussian Heuristic for approximating the number of lattice vectors in a specific region in the hyperspace. We opt to use the statement from [14]:

**Heuristic 1** (**Gaussian Heuristic**). *[14]: Given an arbitrary lattice* $\mathcal{L}$ *along with its basis* $B$*, if* $S$ *is a measurable subset of* $Span(B)$*, then the following approximation holds:*

$$|\mathcal{L} \cap S| \approx \frac{vol(S)}{det(\mathcal{L})},$$

*where the determinant* $det(\mathcal{L})$ *of lattice* $\mathcal{L}$ *may be defined as the volume of the fundamental parallelepiped* $\det(\mathcal{L}) = vol(P(B))$*.*

**Definition 5** (**Gram-Schmidt orthogonalization**). *Given a real basis* $\mathbf{b_1}, ..., \mathbf{b_n}$ *of* $\mathbb{R}^n$*, the Gram-Schmidt orthogonalization of* $\mathbf{b_1}, ..., \mathbf{b_n}$ *is defined by* $\mathbf{b_i^*} = \mathbf{b_i} - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b_j^*}$*, where* $\mu_{i,j} \overset{def}{=} \frac{\langle \mathbf{b_i}, \mathbf{b_j^*} \rangle}{\langle \mathbf{b_j^*}, \mathbf{b_j^*} \rangle}$ *[29].*

**The LLL algorithm** One of the most famous algorithms for solving approximate SVP is the LLL algorithm [29]. This algorithm starts with an integral input basis and a parameter $\delta$ and outputs a $\delta - LLL$ reduced basis $\mathbf{b_1}, ..., \mathbf{b_n}$ which satisfies the two following properties:

1. The Gram-Schmidt coefficients are bounded in absolute value by $\frac{1}{2}$, i.e. $|\mu_{i,j}| \leq \frac{1}{2}$ for $i = \overline{1, n}$ and $j < i$.
2. The Gram-Schmidt orthogonalized basis vectors do not decrease too fast, i.e. $\|\mathbf{b_{i+1}^*}\|^2 \geq (\delta - \mu_{i+1,i}^2)\|\mathbf{b_i^*}\|^2, i = \overline{1, n-1}$.

The property that bears the most interest in the following sections is that the LLL algorithm with $\delta = \frac{3}{4}$ outputs in polynomial time a basis for which $\|\mathbf{b_1}\| \leq 2^{(n-1)/2}\lambda_1(\mathcal{L}(B))$. Thus, it provides an approximation of $\lambda_1(\mathcal{L}(B))$ up to an exponential factor [29].

**Lemma 3.** *[30, 14] Let* $A$ *be an algorithm that solves the SVP in a given lattice* $\mathcal{L} \subseteq \mathbb{R}^n$ *when it receives as input a hint* $\lambda$ *such that* $\lambda_1(\mathcal{L}) \leq \lambda \leq (1 + \frac{1}{n})\lambda_1(\mathcal{L})$*. Then it is possible to design an algorithm* $A'$ *which uses* $A$ *as a subroutine to solve SVP without being given the hint* $\lambda$*, in time greater by a factor* $f = O(n^2)$*.*

We use this lemma to improve the clarity of the exposition, in the sense that the SVP algorithms presented in the following will always assume that the hint is given as input. In fact, what happens when proving this lemma is that algorithm $A'$ first runs the LLL algorithm to obtain an approximation $\nu$ of $\lambda_1(\mathcal{L})$ up to a $2^{(n-1)/2}$ factor. Then it guesses a value $i$ in the set $\{0, ..., -\frac{n-1}{2}\log_{(1+\frac{1}{n})} 2\}$ and computes the hint $\lambda = \nu(1 + \frac{1}{n})^i$. As there are $O(n^2)$ possible values for $i$, this means that the time increases by a factor in $O(n^2)$. For more details, the interested reader is referred to [30, 14].

## 2 The AKS sieving algorithm

The first provable lattice sieving algorithm has been proposed by [5], and has been subsequently improved in a number of papers [25, 26, 14]. The main idea of the algorithm is to sample a great number of lattice vectors and then to reduce their norm by pairwise reductions until a shortest vector is recovered. This process of successively reducing vectors is called sieving. For some sub-unitary real $\gamma$ and some parameter $\epsilon = O(\lambda_1(\mathcal{L}))$, the algorithm will start with vectors in a hyperball $B_n(R)$ and will then obtain vectors in the hyperballs $B_n(\gamma R + \epsilon)$, $B_n(\gamma^2 R + \gamma\epsilon + \epsilon)$ and so on. However, one of the main difficulties to this approach lies in evaluating the distribution of lattice vectors. Instead, our perspective of the lattice is "blurred" by adding to each lattice vector $\mathbf{v}$ a small real perturbation $\mathbf{x}$, which will mean that depending on the size of $\mathbf{x}$, the result $\mathbf{v} + \mathbf{x}$ could have been obtained from other lattice vectors as well. The algorithm will keep pairs of lattice vectors and their perturbed variants and will sieve only based on information from the later. Then, once it finished sieving, it will use the former in order to recover the shortest vector. The first step of the algorithm will be to apply LLL reduction, due to the fact that a good approximation of $\lambda_1(\mathcal{L})$ is required. The details for computing this approximation, denoted by $\lambda$, are explained in the proof of Lemma 3. This presentation follows the general outline from [14]. In the following we first describe the sampling and the sieving procedures.

### Sampling

Rather than first sampling lattice vectors and then adding the perturbation, the sampling procedure generates pairs by first choosing a perturbation of small norm and then by computing, using an approach similar to Babai's rounding [6], the approximate closest lattice vector to $-\mathbf{x}$. Whilst in Babai rounding each real coordinate is approximated to its nearest integer, in this case the algorithm uses the floor function instead. To give an example of how does the modulo operation work, the vector $(4.2, 3.1, ..., 2.7) \mod P(B)$ will result in $(0.2, 0.1, ..., 0.7)$. In Sample Algorithm 1, the description of the sampling algorithm is given as it has been used in [5, 26].

**Algorithm 1** [5]:Sample

**Input:** $B = \{\mathbf{b_1}, ..., \mathbf{b_n}\}$ and perturbation $\mathbf{x}$
**Output:** A lattice vector $\mathbf{v}$ and its perturbed variant $\mathbf{v} + \mathbf{x}$.
1: Compute $\mathbf{v} = -\mathbf{x} + (\mathbf{x} \mod P(B))$.
2: **return** pair $(\mathbf{v}, \mathbf{v} + \mathbf{x})$.

### Sieving

The sieving procedure receives as input a list of pairs in $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in \mathcal{L} \cap B_n(R)$. Then, it performs reductions so that for the outputted pairs, the norms of the second components are in a ball of even smaller radius $\gamma R + \epsilon$, where $\gamma$ is a positive sub-unitary real and $\epsilon$ is a parameter which is linear in $\lambda_1(\mathcal{L})$. During the process, a certain number of pairs will not be reducible but will be used instead for the reduction of other pairs. These pairs are usually called centers. Before the start of the procedure none of the pairs are designated as centers. The initial list is processed sequentially: if a pair $(\mathbf{v}, \mathbf{v} + \mathbf{x})$ is geometrically close to a pair $(\mathbf{c}, \mathbf{c} + \mathbf{x}')$, that has been labelled as a center, the pair is reduced by computing two vector differences and outputting the pair $(\mathbf{v} - \mathbf{c}, (\mathbf{v} + \mathbf{x}) - \mathbf{c})$. Otherwise, if $(\mathbf{v}, \mathbf{v} + \mathbf{x})$ is not close to any center, it will be designated as a center itself. Geometrically close here means that the difference between the second component of the two pairs is smaller than $\gamma R$.

Remark that the algorithm is written in such a way as to highlight the fact that the initial perturbations are preserved by the sieve, which is a technical detail needed for proving that we output a shortest non-zero lattice vector with high probability. Also, the sieving routine only

considers the second component of each pair, thus it only concerns itself with the perturbed lattice vectors and not the lattice vectors themselves. The formal description is given in Sieving Algorithm 2.

**Algorithm 2** [5]:Sieving

**Input:** $R$, $0 < \gamma < 1$ and a list $List$ of vector pairs $(\mathbf{v}, \mathbf{v}+\mathbf{x})$ in $\mathcal{L} \times B_n(R)$.
**Output:** Another list $List'$ of vector pairs $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in \mathcal{L} \times B_n(\gamma R + \epsilon)$. ▷ here $\epsilon$ denotes the maximal norm of $\mathbf{x}$

$Centers \leftarrow \emptyset$.
$List' \leftarrow \emptyset$.
**for** $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in List$ **do**
    **if** there is no pair $(\mathbf{v}', \mathbf{v}'+\mathbf{x}') \in Centers$ such that $\|\mathbf{v}'+\mathbf{x}'-(\mathbf{v}+\mathbf{x})\| \leq \gamma R$ **then**:
        Add $(\mathbf{v}, \mathbf{v}+\mathbf{x})$ to $Centers$.
    **else** add $(\mathbf{v}-\mathbf{v}', (\mathbf{v}+\mathbf{x})-\mathbf{v}')$ to $List'$.
    **end if**
**end for**
**return** $List'$.

## The main algorithm

The input to the main algorithm is first and foremost an integral lattice basis $B = \{\mathbf{b_1}, ..., \mathbf{b_n}\}$, along with a hint $\lambda$ which is a good approximation of $\lambda_1(\mathcal{L})$ as in Lemma 3. This is followed by $\gamma$ which determines how much do we shrink the radius at each execution of the sieve and $\epsilon_0$ which is a parameter used for computing $\epsilon$ as $\epsilon = \epsilon_0 \lambda$. The initial radius $R$ is incrementally decreased to $\gamma R + \epsilon$, $\gamma^2 R + \gamma\epsilon + \epsilon$ and so on, converging towards a small radius $R_0$, where $R_0 \stackrel{def}{=} \frac{\epsilon}{1-\gamma}$. Parameter $\epsilon'$ will dictate when is the radius close enough to $R_0$, namely the algorithm will stop once the radius is smaller than $R_0 + \epsilon'$. More details on the number of steps necessary to accomplish this will be given in the proof of correctness. Finally, input $N$ determines the initial number of sampled pairs. The full description of the algorithm is given in Algorithm 3.

**Algorithm 3** [5]:AKS

**Input:** Integer Lattice Basis $B = \{\mathbf{b_1}, ..., \mathbf{b_n}\}$, $\epsilon_0, \epsilon', \gamma$, $N$ and hint $\lambda$.
**Output:** A shortest non-zero vector of $B$.

$\epsilon \leftarrow \epsilon_0 \times \lambda$.
$List_0 \leftarrow \emptyset$.
**for** $i = 1$ to $N$ **do**
    Draw $\mathbf{x}$ uniformly at random from $B_n(\epsilon)$.
    Add Sample Algorithm $1(B, \mathbf{x})$ to $List_0$.
**end for**
$R \leftarrow n \times max_i \|\mathbf{b_i}\| + \epsilon$.
$j \leftarrow 1$.
**while** $R > (1+\epsilon')\frac{\epsilon}{1-\gamma}$ **do**
    $List_j \leftarrow SievingAlgorithm\ 2(List_{j-1}, R, \gamma)$.
    $R \leftarrow \gamma R + \epsilon$.
    $j \leftarrow j + 1$
**end while**
Consider all pairs $(\mathbf{v}, \mathbf{v}+\mathbf{x_1}), (\mathbf{w}, \mathbf{w}+\mathbf{x_2})$ in $List_k$ and return minimal non-zero difference $\mathbf{v}-\mathbf{w}$.

## Correctness of the AKS algorithm

After the sampling phase, the hope would be that the initial radius $R$ is larger than $\lambda_1(\mathcal{L})$ by at most an exponential factor in $n$ (i.e. $R = 2^{O(n)}\lambda_1(\mathcal{L})$). This would allow for reaching the last possible radius $\frac{\epsilon}{1-\gamma} + \epsilon' = O(\lambda_1(\mathcal{L}))$ in a polynomial number of steps. This condition is indeed ensured thanks to the following lemma:

**Lemma 4.** *[26, Lemma 3.3] Let $(\mathbf{b_1}, ..., \mathbf{b_n})$ be an LLL-reduced basis of a lattice $\mathcal{L}$. If $\mathbf{s}$ is a shortest vector of $\mathcal{L}$ then there exists an index $i \in \{1, .., n\}$ such that $\mathbf{s}$ belongs to the lattice spanned by $(\mathbf{b_1}, ..., \mathbf{b_i})$ and $\|\mathbf{b_j}\| \leq 2^{O(n)}\lambda_1(\mathcal{L})$ for every $j = \overline{1, i}$.*

Therefore, upon applying the LLL algorithm it is possible to safely remove all the basis vectors of norm bigger than $2^{O(n)}\lambda_1(\mathcal{L})$ without losing the shortest vector in the lattice. Subsequently, as perturbed vectors are of the form $\mathbf{x} \mod P(B)$, by the triangle inequality all perturbed vectors are of norm smaller than $n \max_j \|\mathbf{b_j}\| + \epsilon$, which is the value the algorithm chooses as the initial radius. It follows that the initial radius is larger by an exponential factor than $\lambda_1(\mathcal{L})$, i.e. $R = 2^{O(n)}\lambda_1(\mathcal{L})$. Using this result, it is possible to bound the number of sieve executions by a polynomial in $n$:

**Number of sieve executions and choice of $\epsilon'$** Denote the number of sieving steps by a variable $k$, which will be upper-bounded in the following. Let $R$ denote the initial radius from which the AKS algorithm starts sieving. When radius $R_0 \stackrel{def}{=} \frac{\epsilon}{1-\gamma}$ is reached, it is not possible to reduce the radius anymore. Nonetheless, $R_0$ is achieved only as the number of sieving executions reaches infinity, since after $k$ iterations the reached radius is $\gamma^k R + \frac{1-\gamma^k}{1-\gamma}\epsilon = R_0 + \gamma^k(R - R_0)$. This quantity is equal to $R_0$ only when $k \to \infty$. Instead, consider $\epsilon' = \frac{1}{n}$ and ask that $R_0 + \gamma^k(R - R_0) = (1 + \epsilon')R_0$. Then $k = \lceil \log_\gamma(\frac{R_0}{n(R-R_0)}) \rceil$. Parameter $\epsilon_0$ is not dependent on $n$, which along with the fact that $\lambda_1(\mathcal{L}) \leq \lambda \leq (1 + \frac{1}{n})\lambda_1(\mathcal{L})$ implies that $\epsilon = \epsilon_0\lambda$ is in $\Theta(\lambda_1(\mathcal{L}))$. Since $\gamma$ is not dependent on $n$, this means that $R_0 = \Theta(\lambda_1(\mathcal{L}))$ as well. Now, it is known that using Lemma 4 as described above it is guaranteed to obtain $R = 2^{O(n)}\lambda_1(\mathcal{L})$, which means that $k = \log_{1/\gamma}(n2^{O(n)}) = O(n)$.

It is possible to return now to the description of the proof of correctness. Some explanations regarding the last steps of the algorithm are also necessary. If $R_0$ is the last radius considered for perturbed vectors, let $r_0$ be the last radius for the corresponding lattice vectors obtained, which means that $r_0 = R_0 + \epsilon$. Once a small enough radius has been reached, the goal is to obtain sufficient lattice vectors as to saturate the $n$-dimensional ball of radius $r_0 = O(\lambda_1(\mathcal{L}))$. As all these pairs will be geometrically close, taking their pairwise difference in line 14 would hopefully yield the shortest vector in the lattice. But how to ensure that not all pairs will be used for reduction in Sieving Algorithm 2? For this, it is necessary to introduce first a result concerning the kissing constant. The kissing constant is related to the maximal number of vectors in $\mathbb{R}^n$ such that every common angle is lower bounded by a specific value (normally 60°):

**Theorem 1 (Kabatiansky and Levenshtein).** *[25] : Let $\theta_0$ be an angle such that $0 < \theta_0 < 63°$ and let $S \subseteq \mathbb{R}^n$ be a set of vectors such that for every distinct $v_i, v_j \in S$, $\theta(v_i, v_j) > \theta_0$. When $n \to \infty$, it holds that $|S| \leq 2^{c \times n}$, where:*

$$c = -\frac{1}{2}\log(1 - \cos(\theta_0)) - 0.099.$$

Using this theorem, it is possible to bound the number of pairs which are labelled as centers at each iteration of the sieve and therefore lost, by first noticing that the distance between the perturbed vectors needs to be at least $\gamma R$. From [14], $\theta_0 \approx_{n\to\infty} \arccos(1 - \frac{\gamma^2}{2})$, which in turn makes it possible to upper bound the minimal angle $\theta_0$ by 60°, falling under the conditions of Theorem 1 and obtaining the following lemma:

**Lemma 5** (**Number of centers**). *[14, Lemma 7.1]: Let $S \subseteq B_n(R)$ such that the distance between any two vectors of $S$ is at least $\gamma R$, with $0 < \gamma < 1$. Then $|S| \leq N_T \stackrel{def}{=} 2^{c_t n + o(n)}$, where:*

$$c_t = -\log_2(\gamma) + 0.401.$$

At this point, it is certainly possible to choose the initial $N$ as something around $n^3 N_T + 1$ to ensure that after all the sieving steps at least one lattice vector in $O(\lambda_1(\mathcal{L}))$ is obtained, but there are no guarantees that this lattice vector is non-zero. To formally prove that this is the case, it is necessary to use some special types of vector pairs which are denoted as good. These are those pairs whose perturbation satisfies the following condition:

**Definition 6** (**Good perturbations and good pairs**). *Let us define $T_s \stackrel{def}{=} B_n(0, \epsilon) \cap B_n(-\mathbf{s}, \epsilon)$ and $T_{-s} \stackrel{def}{=} B_n(0, \epsilon) \cap B_n(\mathbf{s}, \epsilon)$, where $\mathbf{s}$ is a shortest vector in $L$. Then a pair $(\mathbf{v}, \mathbf{v} + \mathbf{x})$ is defined to be good when $\mathbf{x} \in (T_s \cup T_{-s}) \setminus (T_s \cap T_{-s})$.*

When the perturbations are drawn uniformly at random from the $n$-dimensional ball of radius $\epsilon = \epsilon_0 \lambda$, the following lemma provides a lower bound on the probability that a perturbation and by extension a pair is good. The formulation for the case when $\epsilon_0 \leq 1$ can be found in [26], whilst the case $\epsilon_0 \leq \frac{1}{2}$ is not interesting, as there would be no good perturbations. Values of $\epsilon_0 > 1$ have been used before in [32, 33], what we do in addition is to give an explicit formula for this case.

**Lemma 6.** *Consider $\epsilon_0 > \frac{1}{2}$ and $\lambda$, where $\lambda_1(\mathcal{L}) \leq \lambda \leq (1 + \frac{1}{n})\lambda_1(\mathcal{L})$. For any perturbation $\mathbf{x}$ drawn uniformly at random from $B_n(\epsilon_0 \lambda)$, the probability that $\mathbf{x}$ is good is given by:*

$$P_{x \xleftarrow{\$} B_n(\epsilon_0 \lambda)}(\mathbf{x} \in (T_s \cup T_{-s}) \setminus (T_s \cap T_{-s})) \geq \frac{1}{N_G},$$

*where $N_G \stackrel{def}{=} 2^{-c_g n + o(n)}$ and*

$$c_g = \begin{cases} -\frac{1}{2}\log_2(1 - \frac{1}{4\epsilon_0^2}), \text{if } \epsilon_0 \lambda \leq \lambda_1(\mathcal{L}) \\ -\frac{1}{2}\log_2(1 - \frac{1}{4\epsilon_0^2}) - \frac{1}{n}\log_2\left(1 - \left(\frac{\epsilon_0^2 - 1}{4\epsilon_0^2 - 1}\right)^{\frac{n}{2}}\right), \text{if } \epsilon_0 \lambda > \lambda_1(\mathcal{L}). \end{cases}$$

*Proof.* First recall that from Lemma 3, the hint $\lambda$ satisfies $\lambda_1(\mathcal{L}) \leq \lambda \leq (1 + \frac{1}{n})(\lambda_1(\mathcal{L}))$. First consider $\lambda = \lambda_1(\mathcal{L})$. When $\epsilon_0 \leq 1$, it is sufficient to use Lemma 2 for height $h = (\epsilon_0 - \frac{1}{2})\lambda$ and radius $\epsilon = \epsilon_0 \lambda$, as the two spherical caps corresponding to $T_s$ and $T_{-s}$ do not intersect, as in Figure 2. We obtain that the relative volume is $poly(n)\left(1 - \left(1 - \frac{(\epsilon_0 - \frac{1}{2})\lambda_1(\mathcal{L})}{\epsilon_0 \lambda_1(\mathcal{L})}\right)^2\right)^{\frac{n}{2}} = poly(n)\left(1 - \frac{1}{4\epsilon_0^2}\right)^{\frac{n}{2}} = 2^{\frac{n}{2}\log_2\left(1 - \frac{1}{4\epsilon_0^2}\right) + o(n)}$, which is the expression we wanted.

Nevertheless, for $\epsilon_0 > 1$, it is necessary to take into account the intersection which is composed of two spherical caps of height $(\epsilon_0 - 1)\lambda$, and we are left with: $poly(n)((1 - \frac{1}{4\epsilon_0^2})^{\frac{n}{2}} - (1 - \frac{1}{\epsilon_0^2})^{\frac{n}{2}})$. Using the fact that $\log(a + b) = \log(a) + \log(1 + \frac{b}{a})$, the result follows.

Now, let us see what happens when $\lambda_1(\mathcal{L}) \leq \lambda$. The error is at most $\frac{1}{n}$, which results in hyperspheres $B_n(\epsilon_0 \lambda)$ of volumes larger than the volume of $B_n(\epsilon_0 \lambda_1(\mathcal{L}))$ by a factor $(1 + \frac{1}{n})^n$. Thus, this difference belongs to $2^{o(n)}$ and will be disregarded. □

Because the expression $\frac{1}{n}\log_2\left(1 - \left(\frac{\epsilon_0^2 - 1}{4\epsilon_0^2 - 1}\right)^{\frac{n}{2}}\right)$ is in $o(1)$ as $n$ goes to infinity, it could be disregarded as well and it is possible to use the formulation for $\epsilon_0 \lambda \leq \lambda_1(\mathcal{L})$ in the case when $\epsilon_0 \lambda > \lambda_1(\mathcal{L})$.

Returning to the correctness of the AKS, the idea of the proof is to show that at the end of all the sieving steps, as far as the good pairs are concerned, the probability of obtaining a collision

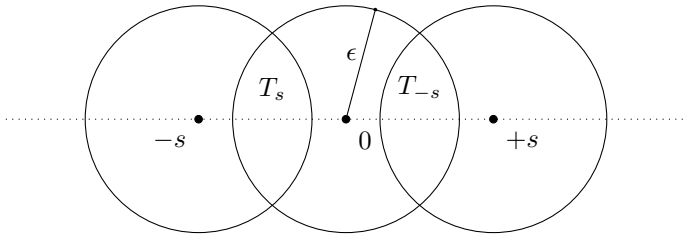


Figure 2: Good perturbations in the case where $\epsilon < \lambda_1(\mathcal{L})$

on the lattice vectors of two pairs (i.e. two pairs $(\mathbf{v}, \mathbf{v} + \mathbf{x_1}), (\mathbf{v}, \mathbf{v} + \mathbf{x_2})$) is at least $\frac{1}{2}$ times the probability of recovering the shortest vector when differences are computed. Therefore, we need one more result which would allow us to estimate the cardinality of the set in which we want to obtain a collision, for the ball of radius $r_0\lambda$:

**Proposition 1.** *[14, Lemma 7.2]: Given a lattice $L \in \mathbb{R}^n$ and $r_0 \geq \frac{1}{\gamma}$, it holds that $|\mathcal{L} \cap B_n(r_0\lambda)| \leq N_B \stackrel{def}{=} 2^{c_b n + o(n)}$, where $c_b = \log_2(r_0) + 0.401$.*

This result follows directly from Lemma 5 with values $R = r_0\lambda$ and $\gamma = \frac{1}{r_0}$. It is possible now to describe the proof of correctness, through the following theorem, where $N_T$, $N_G$ and $N_B$ are defined as in Lemmas 5 and 6 and Proposition 1.:

**Theorem 2.** *[26, 14, 30] The AKS algorithm succeeds in finding a shortest non-zero vector with probability at least $\frac{1}{2} - \xi$ when run with $N \geq \frac{1}{1-\frac{1}{n}} N_G \times (n^2 \times N_T + N_B + 1)$, $0 < \gamma < 1$, $\epsilon_0 > \frac{1}{2}$, $\epsilon' = \frac{1}{n}$, $\lambda_1(\mathcal{L}) \leq \lambda \leq (1 + \frac{1}{n})\lambda_1(\mathcal{L})$ and for $n \to \infty$, where $\xi$ is of the form $\frac{1}{2^{\theta(n)}}$.*

*Proof.* Recall the definition of a good pair from Definition 6 and let $X$ denote the number of good pairs sampled at the beginning of the sieve. We consider the worst case in which the probability that a pair is good is exactly $\frac{1}{N_G}$. It follows from Lemma 1 with $\delta = \frac{1}{n}$ that:

$$P(X \geq n^2 \times N_T + N_B + 1) \geq 1 - \frac{1}{e^{\frac{N}{2N_G n^2}}}.$$

Therefore, the number of good pairs sampled at the beginning of the algorithm is greater than $n^2 \times N_T + N_B + 1$ with probability exponentially close to 1. According to Lemma 5, at each execution of the sieve there is a loss of at most $N_T$ pairs. It has been shown in a previous paragraph that for $\epsilon' = \frac{1}{n}$, the number of executions of the sieve sub-algorithm is in $O(n)$. Thus, the number of lost pairs is $o(n^2 \times N_T)$ pairs, which means that before line 14 there are $N_B + 1$ good vector pairs.

Now it remains to show that the probability of a collision in the ball $B_n(r_0\lambda)$ is at most half the probability of obtaining the shortest vector by using a slight modification of the AKS algorithm. The idea is to consider $AKS_2$, an identical algorithm with the exception that in Sample Algorithm 1 the algorithm applies only once, with probability $\frac{1}{2}$, a function $\tau$ on all perturbations $\mathbf{x} \in B_n(\mathbf{0}, \epsilon)$. The artifice of applying the function only with a $\frac{1}{2}$ probability is a tossing argument introduced by [5]. The function $\tau$ is defined as:

$$\tau(\mathbf{x}) = \begin{cases} \mathbf{x} + \mathbf{s}, \text{if } \mathbf{x} \in T_s \setminus (T_s \cap T_{-s}), \text{where } \mathbf{s} \text{ will denote the shortest vector in } L \\ \mathbf{x} - \mathbf{s}, \text{if } \mathbf{x} \in T_{-s} \setminus (T_s \cap T_{-s}) \\ \mathbf{x}, \text{otherwise.} \end{cases}$$

Unsurprisingly, this function looks strange and might actually be uncomputable in polynomial time. Also, it might appear that the proof employs a circular argument. In fact, it is irrelevant

what is the complexity of $\tau$, since in the following it will be shown that the probability of a lattice vector being outputted by $AKS_2$ or $AKS$ is the same. There are several properties to consider at this point:

1. In Sample Algorithm 1, pairs $(\mathbf{v}, \mathbf{v}+\mathbf{x})$ are generated by sampling $\mathbf{x}$ uniformly from a small hyperball. Then, the lattice vector $\mathbf{v}$ is computed as $\mathbf{v} = -\mathbf{x} + (\mathbf{x} \mod P(B))$. Recall that when sieving, the sub-procedure makes its decisions only based on perturbed vectors. In fact, these remain the same even if function $\tau$ is applied, $\mathbf{v}+\mathbf{x} = \mathbf{v}+\tau(\mathbf{x})$ for all $\mathbf{x} \in B_n(\mathbf{0}, \epsilon)$. Therefore, the $\tau$ function has no impact on the perturbed vectors obtained after sampling.
2. Perturbations are being maintained throughout the algorithm. Moreover, as the sets $T_s$ and $T_{-s}$ have equal volumes, it follows that the $\tau$ function maintains the same distribution of perturbations as in the original $AKS$, that is to say that $\tau(\mathbf{x})$ is also uniformly distributed in $B_n(\epsilon)$. This along with the fact that sieving does not use lattice vectors translates to the same distribution of the output pairs. Nevertheless, it cannot be said that the initial AKS and the modified variant have the same control flow, this is true for the first execution of the sieve, but not for the others.
3. To conclude, first recall that with probability almost 1, at the end of the algorithm there are $N_B$ pairs left for which the perturbations are good. From Proposition 1, it is known that at least two pairs correspond to the same lattice vector $\mathbf{v}$. This means that in $AKS_2$, with probability $\frac{1}{2}$, one pair had an application of $\tau$ on it and the other did not. Namely, what is obtained is two lattice vectors $\mathbf{v}$ and $\mathbf{v} \pm \mathbf{s}$. This means that the $AKS_2$ algorithm recovers the shortest vector $\mathbf{s}$ by taking pairwise differences. Since $AKS_2$ has the same output distributions as $AKS_1$, this means that with probability $\frac{1}{2}$, the $AKS$ algorithm also recovers the shortest vector $s$.

□

In order to ensure a higher success probability, it is possible to run the AKS algorithm a polynomial number of times $p(n)$, resulting in a success probability larger than a threshold which is roughly $1 - \frac{1}{2^{p(n)}}$, a probability exponentially close to 1. The complexity of the algorithm remains the same, as the polynomial number of rounds is absorbed into the $o(n)$ term in the exponent.

### Complexity analysis

According to [14], the space complexity is given by the initial list size, and is therefore upper bounded by $2^{c_{space}n+o(n)}$, where $c_{space} = c_g + \max(c_t, c_b)$, while the time complexity for the sieve is no more than $2^{(c_{space}+c_t)n+o(n)}$. In order to minimize the complexity of the pairwise comparisons one must resort to the trick of removing all pairs corresponding to the same lattice vector, which takes time $2^{c_{space}+c_b}$.

When a small enough radius is reached, the algorithm performs a comparison of all pairwise differences of the remaining vector pairs, which means at most $2^{2c_bn+o(n)}$ operations. Thus, $c_{time} = \max(c_{space} + c_t, c_{space} + c_b, 2c_b)$. Surprisingly, attempting to optimize will yield that $2c_b$ will be the dominating term.

Now, since the last radius for the perturbed vectors is $\frac{\epsilon}{1-\gamma}$, this means that $r_0$ must be $\epsilon_0(1 + \frac{1}{1-\gamma})$. Optimizing over the time complexity for $\epsilon_0 > \frac{1}{2}$ yields $c_{time} = 3.34418$ with $\epsilon_0 = 0.701$ and $\gamma = 0.585$. The main reference for this analysis is [14].

## 3 Adapting the AKS for solving $\mu$SVP

In this section we describe a natural adaptation of the $AKS$ sieve for finding approximations of the shortest vector. This course of action has first been explored for modifications of $ListSieve$, as

proposed in [21] and [33]. The $ListSieve$ is another sieving algorithm which has lower time and space complexity compared to the $AKS$, and its fastest variant uses the birthday paradox and has been introduced in [27] and [14]. Nevertheless, we show that when the goal is to recover only an approximation of the shortest vector up to a factor $\mu$, adapting the $AKS$ is more natural and results in an algorithm that has both lower time and lower space complexity than adaptations of the $ListSieve$. Current findings suggest that designing a provable sieving algorithm for finding an approximation of the shortest vector results in time and space complexities only limited by the known bounds [12] on the kissing number, as given in Theorem 1. Indeed, for asymptotically large approximation factors, both the algorithms based on the $ListSieve$ and our improvement are time-bounded by $2^{0.802n+o(n)}$.

What changes with respect to the original outline of the $AKS$ is the introduction of a different sieving procedure based on the description of the $AKS - Birthday$ from [14]. The idea of this modification is to preselect the pairs that are to be used as centers right after the sampling phase, and to keep two separate lists of vectors, as in the $ListSieve - Birthday$ algorithm. One list will only be used for reductions, while the other will provide a sufficiently large set of reduced pairs at the end of the sieving steps, so that using Lemma 6 we obtain at least one good pair. The new idea here is that, by separating the pairs used for reductions from the pairs among which we search for the approximate shortest vector, it is not necessary anymore to require that the pairs used as centers contain good perturbations. This idea could also be used to improve the $AKS-Birthday$ described in [14], but unfortunately the resulting improved algorithm does not have a better time complexity than the $ListSieve - Birthday$. We proceed by giving a detailed description of the modified sieving procedure used first in $AKS - Birthday$, followed by a description of the modified $AKS$, its proof of correctness and a comparison with previous results.

## Modified Sieving procedure

In Sieving Algorithm 4, we give a full description of the sieving procedure described in [14] for the $AKS - Birthday$ algorithm. In particular, throughout our algorithm we maintain two lists of vectors instead of just one, we first preselect a number of pairs, denoted as $N_C$, which are taken from a list $C$ and we add them to a new set, call it $Centers$. For each $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in (C \cup S) \setminus Centers$ we attempt to reduce the pair by finding a center pair $(\mathbf{c}, \mathbf{c}+\mathbf{x}')$ such that the difference $(\mathbf{v}+\mathbf{x})-(\mathbf{c}+\mathbf{x}')$ is smaller than $\gamma R$ and we keep these differences for the next iteration of the sieve.

**Algorithm 4** (Adapted from [14]):Sieving

**Input:** Lists $C$, $S$, $Centers$ included in $\mathcal{L} \times B_n(R)$, along with $R$, $0 < \gamma < 1$.
**Output:** Lists $C'$ and $S'$ of vector pairs $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in L \times B_n(\gamma R + \epsilon)$. ▷ every perturbation $\mathbf{x}$ satisfies $\|\mathbf{x}\| \leq \epsilon$.

$C' = S' = \emptyset$.
**for** $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in C$ **do**
  **if** there is a pair $(\mathbf{v}', \mathbf{v}'+\mathbf{x}') \in Centers$ such that $\|\mathbf{v}'+\mathbf{x}'-(\mathbf{v}+\mathbf{x})\| \leq \gamma R$ **then**
    Add $(\mathbf{v}-\mathbf{v}', (\mathbf{v}+\mathbf{x})-\mathbf{v}')$ to $C'$.
  **end if**
**end for**
**for** $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in S$ **do**
  **if** there exists $(\mathbf{v}', \mathbf{v}'+\mathbf{x}') \in Centers$ such that $\|\mathbf{v}'+\mathbf{x}'-(\mathbf{v}+\mathbf{x})\| \leq \gamma R$ **then**
    Add $(\mathbf{v}-\mathbf{v}', (\mathbf{v}+\mathbf{x})-\mathbf{v}')$ to $S'$.
  **end if**
**end for**
**return** $C'$, $S'$.

## The main algorithm

For recovering the shortest vector in the lattice up to an approximation factor $\mu$, we use the outline of the original $AKS$ algorithm and we modify it in Algorithm 5 to allow for the incorporation of the modified sieving procedure explained in the previous paragraph. The algorithm starts with a great number of pairs $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in L \cap B_n(R)$, where the second component is bounded by an initial radius $R$. Then, it reduces the norm of all elements repeatedly until it is left with many pairs in the hyperball of small radius $R_0$, which will have to be close to $\mu$. At each step $j$, the set $C_j$ will provide the centers which will be used to reduce both itself and the set $S_j$, obtaining sets $C_{j+1}$ and $S_{j+1}$. At the end of all the runs of the sieving procedure, we only look at vectors in the last set $S_{j-1}$ and output the shortest non-zero one.

**Algorithm 5** $\mu AKS$

**Input:** Integer Lattice Basis $B = \{\mathbf{b_1}, ..., \mathbf{b_n}\}$, $\epsilon_0, 0 < \gamma < 1$, $N_1$, $N_2$, $\mu$, $R_0$ and $\lambda$.
**Output:** A $\mu-$approximation of the shortest non-zero vector of $\mathcal{L}(B)$.

$C_0 = S_0 = \emptyset$.
$\epsilon \leftarrow \epsilon_0 \times \lambda$
**for** $i = 1$ to $N_1$ **do**
    Draw $\mathbf{x}$ uniformly at random from $B_n(\epsilon)$.
    Add Sample Algorithm $1(B, \mathbf{x})$ to $C_0$.
**end for**
**for** $i = 1$ to $N_2$ **do**
    Draw $\mathbf{x}$ uniformly at random from $B_n(\epsilon)$.
    Add Sample Algorithm $1(B, \mathbf{x})$ to $S_0$.
**end for**
$R \leftarrow n \times max_i \|\mathbf{b_i}\| + \epsilon$.
$j \leftarrow 1$.
**while** $R > R_0$ **do**
    $Centers \leftarrow \emptyset$.
    Add first $N_C$ pairs of $C_{j-1}$ to $Centers$.
    $(C_j, S_j) \leftarrow$Sieving Algorithm $4(C_{j-1} \setminus Centers, S_{j-1}, Centers, R, \gamma)$.
    $R \leftarrow \gamma R + \epsilon$.
    $j \leftarrow j + 1$.
**end while**
Consider all $(\mathbf{v}, \mathbf{v}+\mathbf{x}) \in S_{j-1}$ and output shortest non-zero $\mathbf{v}$.

## Correctness of the $\mu$AKS

First, it is important to choose $N_C$ appropriately such that the number of lost pairs is not too great. The only other difference with the original $AKS$ will be that now we require that at the end of the sieving steps we obtain at least one good pair. We are ready now to state our main result, the proof of which is very similar to the proof for the $AKS - Birthday$ algorithm as described in [14].

**Theorem 3.** *When run with $\frac{1}{2} < \epsilon_0$, $0 < \gamma < 1$, $N_1 = n^4 N_T$, $N_2 = (\frac{n}{n-1})^2 \times n^4 \times N_G$, $N_C = n^2 N_T$, $\lambda_1 \leq \lambda \leq \lambda_1(1+\frac{1}{n})$ and $R_0 \geq \frac{\epsilon_0 \lambda}{1-\gamma}(1+\frac{1}{n})$, the $\mu AKS$ algorithm succeeds in finding a $\mu-$approximation of the shortest non-zero vector with probability greater than $\frac{1}{2} - \xi$, for $n \to \infty$, where $\xi = \frac{1}{2^{\theta(n)}}$ and $\mu\lambda$ is greater or equal than $R_0 + \epsilon_0 \lambda$.*

*Proof.* First recall from the analysis of the AKS that the radius for the perturbed vectors converges towards $\frac{\epsilon}{1-\gamma}$. In order to ensure that the number of sieve operations is in $O(n)$, we ask that the last radius $R_0$ is greater than $\frac{\epsilon}{1-\gamma}$ by at least $\frac{1}{n}$. The norm of the lattice vectors obtained will therefore be bounded by $R_0 + \epsilon$, which is why we ask that $\mu\lambda \geq R_0 + \epsilon$.

There is a crucial difference between the sieving procedure of the AKS and the modified sieving procedure of the $\mu AKS$. In the first case all the pairs not labelled as centers will be reduced and there will be at most $N_T$ centers. In the second case however, since the centers are preselected, they will not ensure a perfect covering for the list of pairs we want to reduce. This means that some pairs will not be labelled as centers and will still be at a distance greater than $\gamma R$ than all the centers, so they are also lost. Let us call these pairs exterior pairs. In the following, we estimate the number of exterior pairs using the approach used in [14] for $AKS - Birthday$.

Assume for the moment that we are applying the sieving procedure of the AKS on the $N_C$ pairs that have been set aside in line 15. Let $p_i$ be the probability that the $i^{th}$ pair is irreducible, namely that it is far away from all previous pairs. This would mean that pair $i$ will be added to $Centers$. From Lemma 5, we know that the number of irreducible pairs will be smaller than $N_T$, which means that $\sum_{i=1}^{N_C} p_i \leq N_T$. Also, as the sieving procedure ensures that $Centers$ keeps growing, $p_i$ will be decreasing. Thus we have that $N_C \times p_{N_C} \leq N_T$ and then $p_{N_C} \leq \frac{1}{n^2}$.

Let us return to the $\mu$AKS algorithm and the modified sieving procedure. Now we look at the pairs which we want to reduce and which will not be among the first $n^2 N_C$ pairs set aside as centers. The probability $p'$ of a pair being exterior is smaller than $p_{N_C+1}$, as there are more centers than in the scenario mentioned in the previous paragraph. It is worth pointing out an observation from [14], that all the vectors from the last list $S$ are independently and identically distributed, which would allow us to use Lemma 1 to bound the number of exterior pairs.

For simplicity, in the following we assume that the number of sieving steps (number of entries in the while loop of line 13) is exactly $n$ rather than $O(n)$. The probability that a pair is not exterior after all the executions of the sieve is greater than $(1 - \frac{1}{n^2})^n$. Therefore, the probability $p''$ that a pair is exterior when sieving is completed is less than $1 - (1 - \frac{1}{n^2})^n \approx \frac{1}{n}$.

Let us consider the worst case scenario in which $p'' = \frac{1}{n}$. Consider $X_i$ to be a random variable equal to 1 if pair $i$ is not exterior and to 0 if pair $i$ is exterior. Also, let $X \stackrel{def}{=} \sum_{i=1}^{N_2} X_i$, $N_2' \stackrel{def}{=} (\frac{n}{n-1})^2 \times n^3 \times N_G$ and $N_2'' = \frac{n^3}{1-\frac{1}{n}} N_G$. In fact, $N_2' = E[X]$ and $N_2'' = (1 - \frac{1}{n})E[X]$. From Lemma 1 with $\delta = \frac{1}{n}$, it follows that:

$$P(X \geq N_2'') \geq 1 - \frac{1}{e^{\frac{N_2'}{2n^2}}} \geq 1 - \frac{1}{e^n}.$$

This means that with probability exponentially close to 1, the algorithm will obtain after all the sieving steps at least $N_2''$ pairs. In the following we are interested in how many pairs out of $N_2''$ are actually good, in the sense of Definition 6. For this, first let $Y$ be a random variable denoting the number of good pairs out of the $N_2''$ pairs obtained after sieving. Using Lemma 6 and Lemma 1 with $\delta = \frac{1}{n}$, we have that:

$$P(Y \geq n^3) \geq 1 - \frac{1}{e^{\frac{N_2''}{2N_G n^2}}} = 1 - \frac{1}{e^{\frac{n^2}{2(n-1)}}}.$$

So we obtain at least one good pair with probability exponentially close to 1. To conclude, we use the tossing argument from the proof of Theorem 2 once more to construct algorithm $\mu AKS_2$. As the lattice vectors outputted by $\mu AKS_2$ follow the same distribution as $\mu AKS$, the probability to output a zero vector is at most double the probability of outputting the shortest vector. Otherwise, we obtain a good approximation of the shortest vector.

□

Our most important observation in the construction of our algorithm has been that, since the perturbations associated with centers are irrevocably lost, it is not absolutely necessary that we use only good pairs as centers. To our knowledge, while center preselection has been described before in [14], the idea of not restricting ourselves to good center pairs is novel. The proof of correctness is dependent on the pre-selection of the pairs we use for reduction and unlike previous

approaches to the AKS [14, 25] we process them separately from the pairs on which we apply the tossing argument.

## Complexity analysis

Using the correctness analysis before, as it is sufficient to obtain just one good pair at the end of the sieving steps, we take $N_1 = n^4 N_T = 2^{c_t n + o(n)}$, $N_2 = (\frac{n}{n-1})^3 \times n^3 \times N_G = 2^{c_g n + o(n)}$ and $N_C = 2^{c_t n + o(n)}$. Let us denote the space complexity by $2^{c_{space} n + o(n)}$ and the time complexity by $2^{c_{time} n + o(n)}$. The space complexity is $N_{space} = N_1 + N_2$, which means that $c_{space} = \max(c_t, c_g)$. The most expensive operation is the sieving procedure, which requires time $N_{space} N_C$. Therefore we obtain $c_{time} = c_{space} + c_t$. To obtain the expression of the overall complexity we use the fact that the radius we converge to during AKS sieving is $\epsilon(1 + \frac{1}{1-\gamma})$ which means that $\mu$ must be bigger than $\epsilon_0(1 + \frac{1}{1-\gamma})$. The function $\epsilon_0(1 + \frac{1}{1-\gamma})$ is increasing with respect to both its arguments and $\gamma$ and $\epsilon_0$ must also be as large as possible to have a small number of centers and a big probability of sampling good pairs. Therefore, we need to require that $\mu = \epsilon_0(1 + \frac{1}{1-\gamma})$. Plugging the expressions from Lemma 5 and Lemma 6 we obtain the following result:

**Lemma 7.** *The $\mu AKS$ algorithm with parameters as in Theorem 3 solves the approximate shortest vector problem in time complexity bounded by $2^{c_{time} n + o(n)}$ and space complexity bounded by $2^{c_{space} n + o(n)}$, where:*

$$c_{time} = \max\Big[ -\frac{1}{2}\log_2\Big(1 - \frac{1}{4\epsilon_0^2}\Big), \log_2\Big(\frac{\mu - \epsilon_0}{\mu - 2\epsilon_0}\Big) + 0.401\Big] + \log_2\Big(\frac{\mu - \epsilon_0}{\mu - 2\epsilon_0}\Big) + 0.401 \text{ and}$$

$$c_{space} = \max\Big[ -\frac{1}{2}\log_2\Big(1 - \frac{1}{4\epsilon_0^2}\Big), \log_2\Big(\frac{\mu - \epsilon_0}{\mu - 2\epsilon_0}\Big) + 0.401\Big], \text{where } \mu > 2\epsilon_0.$$

In the expressions above, term $\log_2\left(\frac{\mu-\epsilon_0}{\mu-2\epsilon_0}\right)$ corresponds to the exponent describing the number of centers $c_t$ and thus to $\log(\frac{1}{\gamma})$. The minimal time complexity is reached when we have the following equality, we omit giving a verbose expression just in terms of $\epsilon_0$:

$$-\frac{1}{2}\log_2\Big(1 - \frac{1}{4\epsilon_0^2}\Big) = \log_2\Big(\frac{\mu - \epsilon_0}{\mu - 2\epsilon_0}\Big) + 0.401.$$

Although this last equation is solvable in the general case by treating either a quadratic or a cubic equation (using for the cubic the Cardano's method), we will omit giving a verbose expression. Another remark is that for asymptotically large $\mu$, an optimal value for $\epsilon_0$ is $\epsilon_0 \approx 0.765663$, while $c_{time} \approx 0.802$.

## Comparison with previous results

The effect of relaxing the search for a minimal difference of lattice vectors has been applied before on modifications of the ListSieve-Birthday algorithm. A first proposal has been put forward in [21, 33], of complexity:

**Lemma 8.** ([33, Theorem 6]) *The optimal time complexity of the $\mu ListSieve - Birthday$ (introduced in [33] as Algorithm 2) is $2^{c_{time} n + o(n)}$, where $c_{time} = 0.802 - 1.5\log_2(1 - \mu^{-2/3})$. The corresponding space complexity is $2^{c_{space} n + o(n)}$, where $c_{space} = 0.401 - 0.5\log_2(1 - \mu^{-2/3})$ and $\epsilon_0 = \frac{\sqrt[3]{\mu}}{2}$.*

A second proposal appears in [21], where the authors impose an additional restriction on the lattice vectors in the list constructed by the ListSieve algorithm. Namely, they ask that the distance between every two lattice vectors be greater than $\mu\lambda$. The complexity of this adaptation of the ListSieve is given by the following result:

Table 2: Comparison of time complexity bounds for $\mu ListSieve - Birthday$ variants and $\mu AKS$. A table entry $c_{time}$ indicates a time complexity of $2^{c_{time}n+o(n)}$, when $n \to \infty$.

| | Modified ListSieve[21] | | | $\mu$ListSieve-Birthday[33] | | $\mu$AKS | | |
|---|---|---|---|---|---|---|---|---|
| $\mu$ | $\epsilon_0$ | $\gamma$ | $c_{time}$ | $\epsilon_0$ | $c_{time}$ | $\epsilon_0$ | $\gamma$ | $c_{time}$ |
| 2.71 | 0.824335 | 0.621497 | 1.99758 | 2.697097 | 2.36552 | 0.595838 | 0.718168 | 1.75721 |
| 3.61 | 0.881461 | 0.761989 | 1.77978 | 0.767018 | 1.99933 | 0.624387 | 0.790868 | 1.47898 |
| 8 | 1.08595 | 0.311067 | 1.36434 | 1 | 1.42456 | 0.687324 | 0.90601 | 1.0868 |
| 15 | 1.30208 | 0.294289 | 1.16723 | 1.2331 | 1.19071 | 0.719597 | 0.94961 | 0.951187 |
| 100 | 2.35691 | 0.755867 | 0.903217 | 32.15789 | 0.904852 | 0.757947 | 0.992363 | 0.824121 |

Table 3: Comparison of corresponding space complexity bounds for $\mu ListSieve - Birthday$ variants and $\mu AKS$. A table entry $c_{space}$ indicates a space complexity of $2^{c_{space}n+o(n)}$, when $n \to \infty$.

| | Modified ListSieve[21] | | | $\mu$ListSieve-Birthday[33] | | $\mu$AKS | | |
|---|---|---|---|---|---|---|---|---|
| $\mu$ | $\epsilon_0$ | $\gamma$ | $c_{space}$ | $\epsilon_0$ | $c_{space}$ | $\epsilon_0$ | $\gamma$ | $c_{space}$ |
| 2.71 | 0.824335 | 0.621497 | 0.833343 | 2.697097 | 0.922173 | 0.595838 | 0.718168 | 0.878607 |
| 3.61 | 0.881461 | 0.761989 | 0.749856 | 0.767018 | 0.800109 | 0.624387 | 0.790868 | 0.739491 |
| 8 | 1.08595 | 0.311067 | 0.59624 | 1 | 0.608519 | 0.687324 | 0.90601 | 0.543402 |
| 15 | 1.30208 | 0.294289 | 0.526077 | 1.2331 | 0.53057 | 0.719597 | 0.94961 | 0.475594 |
| 100 | 2.35691 | 0.755867 | 0.435 | 32.15789 | 0.435284 | 0.757947 | 0.992363 | 0.41206 |

**Lemma 9.** ([21]) *The time and space complexities of the algorithm referred to as Modified Algorithm in [21] are $2^{c_{time}n+o(n)}$ and $2^{c_{space}n+o(n)}$, respectively. Expressions $c_{time}$ and $c_{space}$ are defined as $c_{time} = \max(2c_l + c_g, 2c_g)$, $c_{space} = \max(c_l, c_g)$, with $c_g$ as defined in Lemma 6 and $c_l = \log_2(\frac{\sqrt{\epsilon^2+\mu^2}+\epsilon}{\mu}) + 0.401$.*

Experimentally, we have verified up to approximation factor $2^{16}$ that this adaptation of the AKS algorithm has a lower time complexity than both approximation variants of the ListSieve-Birthday, in Table 2 we compare the first values, keeping the same approximation factors as given in [21, 33]. It should be noted that a trivial adaptation of the AKS does not perform better compared to the ListSieve-Birthday. For approximation factors in $\omega(1)$, even taking $\epsilon_0 = 1 - \frac{1}{2^\mu}$ will yield that the complexity of the ListSieve-Birthday adaptation is in $o(\mu AKS\ complexity)$.

In order to compare the space complexity, first we observe that our algorithm starts outperforming the more efficient version of the ListSieve-Birthday variants at around $\mu \approx 3.37$ and we have experimentally verified that this behaviour holds up to a $2^{16}$ approximation factor. In Table 3 we compare the complexities for the same small values of $\mu$.

The ListSieve-Birthday algorithm seems to be more efficient because in the list of vectors we use for reduction we keep only vectors of norm greater than a threshold $r_0\lambda$, which when optimized is around $3.01\lambda$. Nevertheless, the proof of correctness for the ListSieve-Birthday requires that the shrinking factor has to be chosen as $1 - \frac{1}{n}$. We have no such restrictions on the algorithm we propose, and this is why we achieve a better time complexity when considering the approximate version of the shortest vector problem.

## Quantum Search

Most cryptographic primitives based on lattices are normally regarded as secure with respect to quantum computers. Nevertheless, it remains relevant to see whether quantum computers can employ speed-ups of the known classical algorithms solving the SVP in our case. In particular, the Grover quantum search algorithm can be used to speed up sieving algorithms, as pointed out in [20]. The algorithm considers a list $L$ of size $N$ along with a function $f : L \to \{0, 1\}$

such that the set $f^{-1}(1)$ is small. When the memory model considered is a RAM memory, which is also quantumly addressable, the search requires $O(\sqrt{N})$ operations, as opposed to $O(N)$ in the classical setting. In the quantum setting, reducing our list of centers is done in $2^{c_{space}+c_t/2}$, which unsurprisingly yields, for asymptotically large approximations factors, a time complexity of $2^{0.602n+o(n)}$, just as the adaptations of $ListSieve - Birthday$. Naturally, as the quantum search algorithm brings only a speedup in the exponent of $\frac{1}{4}$, it follows that $\mu AKS$ will still have a smaller time and space complexity when compared to $ListSieve$.

# 4 A heuristic version of the AKS - The NV-Sieve

The rest of this report is concerned with the analysis of heuristic algorithms currently used to solve the shortest vector problem. The fastest sieving algorithms in practice according to [13] are modifications of the GaussSieve [25], which is itself based on the ListSieve. The GaussSieve is a very simple algorithm which maintains a list of lattice vectors, initially empty, and then starts sampling lattice vectors which it uses to reduce vectors in the list. Moreover, every time a vector in the list gets reduced it will be treated as a new vector and it will be used once more for reducing the entire list, until no more reductions are possible. While much faster in practice than the sieving heuristic we are going to focus on from now on, only bounds on its space complexity are provable. This means that, from a theoretical point of view, we will focus our analysis on another sieve, due to P. Q. Nguyen and T. Vidick and referred to as the NV-Sieve, first proposed in [26]. As experimentally the GaussSieve is faster than the NV-Sieve, it is conjectured that it has a smaller theoretical complexity as well. This remains an important outstanding problem in the field.

Moving on to our high level description of the NV-Sieve, a first observation is that in the sieving procedure of the $AKS$ algorithm, it is possible to ignore reducing the vectors of norm smaller than $\gamma R$ without affecting the correctness or the complexity analysis. The heuristic of the NV-Sieve is that the lattice vectors targeted for reduction, and which reside in the corona $C_n(\gamma, R) \stackrel{def}{=} \{\mathbf{x} \in \mathbb{R}^n | \gamma R \leq \|\mathbf{x}\| \leq R\}$, are uniformly distributed. Then, using perturbed lattice vectors becomes unnecessary and only lattice vectors are being employed. Moreover, this suggests that the $\epsilon$ loss in the radius reduction does not occur anymore.

The algorithm is not identical to the AKS. The sieving and the sampling subroutines obviously change, but so does the stopping condition. Concerning vector sampling, any algorithm that outputs approximate closest lattice vectors suffices, as long as the approximation is not too large. In practice, the sampling is performed using Klein's randomized algorithm for finding an approximation of the closest lattice vector to any real vector [16]. For brevity, in the following it is assumed that the sampling works in the same fashion without outputting the perturbations.

## Sieving

Apart from the fact that vectors of norm smaller than $\gamma R$ are not reduced anymore, the sieving procedure resembles the one used for approximate SVP. The pairs which are used for reducing other pairs, which were denoted as centers, are once again preselected. In a slight departure from the on-the-fly algorithm described in [26], the algorithm described in the following is due to [17]. While in [17], this selection is randomized, at least theoretically it is possible to use a deterministic procedure and simply label a specific portion of lattice vectors in the beginning of the wider list as centers. Another detail already given in the previous paragraph is that lattice vectors of norm smaller than $\gamma R$ will not be reduced. The full procedure is described more formally in Sieving Algorithm 6.

**Algorithm 6** [26]:Sieving

**Input:** Radius $R$, sets $List$ and $Centers$ of lattice vectors of norms smaller than $R$, $0 < \gamma < 1$.
**Output:** Another list $List'$ of lattice vectors, this time of norms smaller than $\gamma R$.

$List' \leftarrow \emptyset$.
**for** $\mathbf{v} \in List$ **do**
  **if** $\|\mathbf{v}\| \geq \gamma R$ **then**
    **if** there is $\mathbf{c} \in Centers$ such that $\|\mathbf{c} - \mathbf{v}\| \leq \gamma R$ **then**
      Add $\mathbf{c} - \mathbf{v}$ to $List'$.
    **end if**
  **else** add vector $\mathbf{v}$ to $List'$.
  **end if**
**end for**

## The main algorithm

The main algorithm is given in NV-Sieve 7. The main idea is to simplify the workings of the previous algorithms, just sample a large number of lattice vectors and reduce them repeatedly until it is not possible to reduce any longer. Nevertheless, the initial number of vectors sampled will be significantly smaller than in the AKS and the proof of correctness will be based on heuristics. The fundamental heuristic underlying the NV-Sieve will be that somehow, the lattice vectors used by the algorithm will always follow a uniform distribution.

While it remains very similar to the original AKS, there are some differences. Firstly, as the proof of correctness is done on the basis of heuristics, it is not necessary to use perturbations any more. The algorithm will halt its execution once it runs out of lattice vectors and it cannot perform reductions any longer. Moreover, it will always discard zero vectors as they bear no practical relevance. As in the previous sections, $N$ will represent the initial number of sampled lattice vectors and $N_C$ will denote the number of centers selected at each step.

**Algorithm 7** [26]:NV-Sieve

**Input:** Integer Lattice Basis $B = \{\mathbf{b_1}, ..., \mathbf{b_n}\}$, $\gamma$, $N$, $N_C$
**Output:** A shortest non-zero vector of $B$.

Sample $N$ lattice vectors using Klein's algorithm [16] and add them to $List_0$.
Initialize $i \leftarrow 0$.
$R \leftarrow max_{\mathbf{v} \in List_0} \|\mathbf{v}\|$.
**while** $List_i$ is not empty **do**
  $Centers \leftarrow \emptyset$.
  Add first $N_C$ vectors from $List_i$ to $Centers$.
  $List_{i+1} \leftarrow SievingAlgorithm\ 6(R, List_i \setminus Centers, Centers, \gamma)$.
  $R \leftarrow \gamma R$.
  Remove all zero vectors from $List_{i+1}$.
  $i \leftarrow i + 1$.
**end while**
**return** $argmin_{non-zero\ \mathbf{x} \in List_{i-1}} \|\mathbf{x}\|$.

## Analysis of the NV-Sieve

One of the main problems encountered by sieving algorithms is the inability to draw lattice vectors according to some convenient distribution and then ensure, at the same time, that this distribution is preserved after applying the sieving steps. Nevertheless, the practical experiments carried out in [26] suggest that the assumption that lattice vectors in sieving algorithms remain

distributed uniformly in the corona we sieve from is not without merit. As such, the following heuristic has been put forward:

**Heuristic 2.** *[26]: Consider a random integer lattice basis $B$ and an arbitrary radius $R \in \mathbb{R}$ with $R \geq \frac{4}{3}\lambda_1(\mathcal{L}(B))$. Then all **list** vectors $\mathbf{x} \in List_i$, such that $\gamma R \leq \|\mathbf{x}\| \leq R$ are uniformly distributed in the corona $C_n(\gamma)$, before and after every run of the sieving sub-algorithm.*

It could be argued that when $R = \frac{4}{3}\lambda_1(\mathcal{L}(B))$ there are not sufficiently many lattice vectors of norm smaller than $R$. We will show, using Heuristic 1, that there are exponentially many lattice vectors in the ball of radius $B_n(\frac{4}{3}\lambda_1(\mathcal{L}))$. As we know that the hyperball of radius $\lambda_1(\mathcal{L})$ contains at least the shortest vector, from this heuristic we have that $\frac{vol(B_n(\lambda_1(\mathcal{L})))}{det(\mathcal{L})} \geq 1$. But the volume of the hyperball of radius $\frac{4}{3}\lambda_1(\mathcal{L}(B))$ is larger by a factor of $\left(\frac{4}{3}\right)^n$ than the hyperball of radius $\lambda_1(\mathcal{L})$. This means that there should be at least $\left(\frac{4}{3}\right)^n$ lattice vectors in $B_n(\frac{4}{3}\lambda_1(\mathcal{L}))$. Remark that the constant $\frac{4}{3}$ here can be modified, its choice is to provide an unified approach with the $\frac{4}{3}$ approximation factor which will result in the following from Heuristic 3.

**Overview of the analysis** Let us summarise the approach taken in [26] to justify the correctness of the NV-Sieve algorithm. First, they estimate the maximal number of lattice vectors needed to cover the corona $C_n(\gamma, R) \stackrel{def}{=} \{\mathbf{x} \in \mathbb{R}^n | \gamma R \leq \|\mathbf{x}\| \leq R\}$ completely. This means that they estimate the maximal cardinality $\zeta$ of a set of real vectors $S \subseteq C_n(\gamma, R)$, such that any two vectors in the set are at distance at least $\gamma R$ from each other. Then, it will be argued just as in the proof of $\mu AKS$ that preselecting a number of centers larger than $\zeta$ by a polynomial factor still covers the corona satisfactorily. Afterwards, they approximate the number of lattice vectors lost through collisions in line 9 of the NV-Sieve algorithm.

**Estimating the number of centers** As in the proof of the AKS algorithm, a fundamental question for all sieving algorithms is whether it is possible to bound the maximal number of centers. The constraint that needs to be imposed is that there will exist a sub-list of vectors such that any lattice vector of big norm (greater than $\gamma R$) is within $\gamma R$ distance from at least one vector in the sub-list.

**Lemma 10.** *[26, Adapted from Lemma 4.1.1]: Consider $0 < \gamma < 1$, some radius $R$, and a set $S \subseteq C_n(\gamma, R)$ such that the difference between every two vectors in $S$ is at least $\gamma R$. Then $poly(n)(\frac{4}{3})^{\frac{n}{2}} \leq |S| \leq N_{lower}(\varphi)$, where $N_{lower}(\varphi) \stackrel{def}{=} poly(n)\frac{1}{\sin^n(\varphi)}$ and for which:*

$$\varphi = \begin{cases} \arccos(\frac{1}{2\gamma}), \text{ if } \gamma \leq \gamma_0 \stackrel{def}{=} \frac{1}{2}(\sqrt{5}-1) \\ \arccos(1-\frac{\gamma^2}{2}), \text{ if } \gamma > \gamma_0. \end{cases}$$

*Proof.* Start by considering an arbitrary $\mathbf{x} \in C_n(\gamma)$. Now analyse the minimal $\varphi_{min}$ and maximal $\varphi_{max}$ angle $\mathbf{x}$ makes with any $\mathbf{y} \in C_n(\gamma) \cap B_n(\mathbf{x}, \gamma)$ by observing that $S_n(\mathbf{x}, \varphi_{min}) \cap C_n(\gamma) \subseteq B_n(\mathbf{x}, \gamma) \cap C_n(\gamma) \subseteq S_n(\mathbf{x}, \varphi_{max}) \cap C_n(\gamma)$. It should be noted that $\frac{vol(S_n(\mathbf{x},\varphi) \setminus C_n(\gamma))}{vol(S_n(\mathbf{x},\varphi) \cap C_n(\gamma))} = (\frac{1}{\gamma})^n =_{n\to\infty} 0$, result obtained by using Lemma 2 and rescaling the radius by $\gamma$. This means that for any hyperspherical cap, the relative volume of its intersection with $C_n(\gamma)$ can be approximated by the relative volume of the entire hyperspherical cap.

Then notice that $\frac{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \gamma^2}{2\|\mathbf{x}\|\|\mathbf{y}\|}$ has no stationary points for the considered domain. In particular, this means that it is only needed to search at the boundaries of the domain, which finally implies that $\varphi_{max} = \frac{\pi}{3}$, while $\cos(\varphi_{min}) = \max(1 - \frac{\gamma^2}{2}, \frac{1}{2\gamma})$. At this stage, applying Lemma 2 yields the result, as the maximal number of centers will be bounded by the inverse of the volume

of the spherical cap of minimum angle with respect to the entire $n$-dimensional hyperball. The lower bound on $|S|$ follows from Lemma 2 for angle $\frac{\pi}{3}$. □

**Estimating the number of collisions** Is is also necessary to show that the number of vectors lost through collisions in line 9 of the main algorithm is not too great. First, let us state an estimate of the maximal number of lattice vectors in a ball of radius $R$:

**Lemma 11.** *[26, Lemma 3.2]: Given a lattice $\mathcal{L} \in \mathbb{R}^n$ and a radius $R \in \mathbb{R}^*$, it holds that $|\mathcal{L} \cap B_n(R)| \leq 2^{c_R n}$, where the constant $c_R = \log_2(1 + \frac{2R}{\lambda_1(\mathcal{L})})$.*

This result is proven by a volume argument. Placing a ball of radius $\frac{\lambda_1(\mathcal{L})}{2}$ around each lattice vector ensures that any two balls are disjoint. Moreover, since they are all included in $B_n(R + \frac{\lambda_1(\mathcal{L})}{2})$, it is possible to bound the maximal number of lattice vectors by $\frac{vol(B_n(R+\lambda_1(\mathcal{L})/2))}{vol(B_n(\lambda_1(\mathcal{L})/2))} = \left(\frac{R+\lambda_1(\mathcal{L})/2}{\lambda_1(\mathcal{L})/2}\right)^n$.

Now that we have an upper bound on the maximal number of lattice vectors it is possible to proceed to the following result, which is another formulation of the widely used birthday paradox:

**Lemma 12.** *[26, Lemma 4.4]: By making $p$ draws with replacement from a set of cardinality $N$, the average number of different elements picked is $N - N(1 - \frac{1}{N})^p$. Thus, the average number of vectors lost by collision is $p - N + N(1 - \frac{1}{N})^p \approx \frac{p^2}{2N}$, negligible if $p \ll \sqrt{N}$.*

Under the assumption of Heuristic 2, this can provide us with a bound on the number of lattice vectors we expect to lose through collisions at each step. More precisely, we can assume that at each iteration of the sieve, we are somehow re-sampling from $\mathcal{L} \cap B_n(R)$. From Lemma 11, even though the bound is not tight, according to [26], it is possible in practice to resort to the following approximation:

**Heuristic 3.** *For arbitrary $L$, the number of lattice vectors of norm at most $R$ follows the rough approximation $|\mathcal{L} \cap B_n(R)| \approx (\frac{R}{\lambda_1(\mathcal{L})})^n$.* [1]

First, let us show that the volume of the corona dominates the inner part that is left. Take for example $n = 2j$. This is without loss of generality, as the odd dimension case can be treated identically. Then, the volume of a corona whose thickness is $\gamma R$ will be $\frac{\pi^j}{j!}(R^{2j} - \gamma^{2j} R^{2j}) = \frac{\pi^j}{j!} R^{2j}(1 - \gamma^{2j}) \approx_{n\to\infty} \frac{\pi^j}{j!} R^{2j}$, since $\gamma < 1$. Then it follows that $|\mathcal{L} \cap B_n(R)| \approx |\mathcal{L} \cap C_n(\gamma, R)|$. We need to know for what values of $R$ sampling $poly(n)\left(\frac{4}{3}\right)^{\frac{n}{2}}$ vectors from $\mathcal{L} \cap B_n(R)$ will become problematic by inducing too many collisions. From the lemma, this happens when we want to sample around $\sqrt{|\mathcal{L} \cap B_n(R)|} \approx (\frac{R}{\lambda_1(\mathcal{L})})^{n/2}$ elements from the $B_n(R)$ ball. If the initial list size is $poly(n)\left(\frac{4}{3}\right)^{\frac{n}{2}}$, we will encounter problems only when $\frac{R}{\lambda_1} \approx \frac{4}{3}$. Therefore, the collisions become significant only once we already have a $\frac{4}{3}$ approximation of $\lambda$. Note that it would be interesting to be able to lower-bound tightly $|\mathcal{L} \cap B_n(R)|$, nonetheless this does not seem to be an easy problem. It is possible now to give the correctness result under the assumption that the heuristic on the distribution of lattice vectors and the estimate on their number are correct.

**Theorem 4.** *[26] Assuming that Heuristic 2 and Heuristic 3 hold, the $NV - Sieve$ algorithm with $N \geq O(n^2 \times N_{lower}(\varphi))$, $N_C = n \times N_{lower}(\varphi)$ and $0 < \gamma < 1$, recovers at least a $\frac{4}{3}$ approximation of*

[1] One counter-example is to consider an orthogonal basis of vectors ordered by norm, with $\|b_2\| = 2^n \lambda_1(\mathcal{L})$, as anything bigger can be safely removed. Applying the $LLL - reduction$ will leave the basis unchanged. Writing $v = \sum_{i=1}^n a_i b_i$, we know that $|a_1| \leq \frac{R}{\lambda_1(\mathcal{L})}$ and $|a_i| \leq \frac{R}{2^n \lambda_1(\mathcal{L})}$, $i = \overline{2,n}$. An upper-bound on our number of vectors is $(\frac{2R}{\lambda_1(\mathcal{L})})(\frac{R}{2^{n-1}\lambda_1(\mathcal{L})})^{n-1}$, which is much smaller than $\left(\frac{R}{\lambda_1(\mathcal{L})}\right)^n$.

*$\lambda_1(\mathcal{L})$ with probability at least $\frac{1}{2}$, for sufficiently large $n$. $N_{lower}(\varphi)$ and $\varphi$ depend on the choice of $\gamma$ and are defined in Lemma 10.*

In order to prove this lemma, first notice that the number of executions of the sieve is at most $\lceil log_\gamma(\frac{\lambda}{R})\rceil \approx O(n)$. Therefore, it is sufficient to consider an initial list size sufficiently large to allow the removal of $N_C$ centers at each execution of the sieve, namely $N \geq O(n \times N_C)$. Nonetheless, it still remains to show that even when preselecting the centers, a good coverability of the corona can be reached with a number of centers larger only by a polynomial factor than $N_{lower}$. Indeed, let us consider $N_C = nN_{lower}$. The expected fraction of the corona not covered by a center is $(1 - \frac{1}{N_{lower}})$, thus the expected fraction of the corona not covered by $N_C$ centers is $(1 - \frac{1}{N_{lower}})^{nN_{lower}} = \frac{1}{e^n}$, which is in turn smaller than $\frac{1}{N_{lower}}$ since $N_{lower} < e^n$. Then the expected number of points not covered is $(N - N_C)\frac{1}{N} < 1$.

The first natural question that arises is whether it is reasonable to consider the upperbound on the number of centers from Lemma 10 instead of a smaller number. Indeed, the following result shows that the sieve needs, up to a polynomial factor, at least $N_{lower}(\varphi)$ centers in order to ensure that the coronas considered at each step are satisfactorily covered:

**Lemma 13 (Tightness).** *[26, Lemma 4.1.2]: Let $List \subseteq \mathcal{L} \cap B_n(R)$, with $|List|$ denoted as $N$. When $N < 4\sqrt{\frac{\pi}{2n}}\left(\frac{4}{3}\right)^{\frac{n}{2}}$, the expected number of points in $List$ that are at distance at least $\gamma R$ from all other points in $List$ is at least $(1 - \frac{1}{n})N$.*

This result provides a lower bound for the size of the list at each step. If the cardinality falls below this value we expect the algorithm to run out of vectors in a sub-linear number of sieve executions, which would not reduce the radius to a small enough value close to $\lambda_1(\mathcal{L})$.

## Complexity analysis of the NV-Sieve

Now that enough results were given to ensure that the algorithm functions correctly if the heuristic assumptions hold, its complexity should be evaluated in order to compare it to the theoretical results on the $AKS$ algorithm from Section 2. The choice of $\gamma$ influences the angle of the covering spherical caps and it will be detailed afterwards.

The space complexity of the sieve is given by the initial number $N$ of lattice points sampled, similarly with the AKS. Choosing a shortest vector at the end and the elimination of zero vectors is linear in $N$, while the most costly is the sieving step, which can be upperbounded by $N^2$. Minimizing the complexity may therefore be reduced to finding a minimal expression for $N = N_{lower}(\varphi)$, which can be expanded using the formula $\sin(\arccos(x)) = \sqrt{1-x^2}$ and Lemma 10 to:

$$N = \begin{cases} poly(n)\left(\frac{1}{1-\frac{1}{4\gamma^2}}\right)^{n/2}, \text{ if } \gamma \leq \gamma_0 \overset{def}{=} \frac{1}{2}(\sqrt{5}-1) \\ poly(n)\left(\frac{1}{\gamma^2(1-\frac{\gamma^2}{4})}\right)^{n/2}, \text{ if } \gamma > \gamma_0. \end{cases}$$

Both functions of $N$ have their minimum in $\left(\frac{4}{3}\right)^{\frac{n}{2}}$ for $\gamma \to 1$, which simplifies the computations as there is no need to treat the two expressions separately. For this theoretical analysis, taking $\gamma = 1 - \frac{1}{n}$ as in Algorithm 3 yields $N = 2^{n/2\log(4/3)+o(n)} = 2^{0.207519n+o(n)}$, after several more intricate computations which are omitted here. In conclusion, since the overall complexity is quadratic with respect to $N$, it follows that the complexity of the algorithm is $2^{0.415037n+o(n)}$. For practical details on the parameters the interested reader is referred to [26].

# 5 Speed-ups of the NV-Sieve using angular hashing

One of the ideas that have been proposed upon the apparition of the NV-Sieve is multi-levelled sieving. The first proposals consisted in two [32] or three [34] levels of centers. These techniques improve the time complexity of the NV-Sieve to $2^{0.3778n+o(n)}$, albeit at an increase in space complexity. Presently, the complexity of the 4-levelled sieve and beyond are conjectured in [18] to follow the time-space complexity trade-off curve from the overlattice sieving algorithm proposed in [9]. Moreover, the overlattice sieve itself is outperformed by another class of state-of-the art techniques, on which we have focused our efforts and which will be described next.

In the original paper in which the NV-Sieve was put forward [26], it is suggested that the quadratic complexity of the sieving step might be further improved upon. The idea of the optimization is to use nearest neighbour techniques to identify more easily a center at distance at most $\gamma R$ for any lattice vector targeted for reduction. Identifying near neighbours is done using special hash families that are locality sensitive. Namely, they will partition the $n$-dimensional corona $C_n(\gamma)$ into several regions and with high probability two vectors will collide with respect to the hash function when they belong to the same region. This technique is referred to as locality-sensitive hashing (LSH). Nguyen and Vidick attempted to identify close neighbours using Euclidean-based hash families and did not reach positive results. Much later, in [17], a hash family which provides a significant improvement to the NV-Sieve has been proposed, which used hash functions based on the angular rather than Euclidean distance. Moreover, the reason for which this line of research is more promising, when compared to the overlattice or to the levelled sieving is that these hashing techniques can be equally applied on the GaussSieve [25]. Recall that the GaussSieve is considered to be one of the fastest algorithms at the moment, outperforming the NV-Sieve in practice. Therefore, any practical improvements of the GaussSieve are of much interest.

To date, a large variety of different hash functions for finding near neighbours have been proposed, [19, 10] as well as derived techniques [8]. The application of all these techniques on top of the NV-Sieve or GaussSieve is almost identical, which is why this report will focus on the first paper that introduced this speedup [17].

In the nearest neighbour problem, we are given a query vector, along with a list of vectors in a vector space and a distance function. Then, it is asked to find a vector in the list which is closest to the query vector with respect to the distance function. In the context of sieving algorithms, this problem may be relaxed from finding the closest neighbour of a given query to locating instead a vector which is close enough such that by taking the difference, a shorter lattice vector is obtained. Thus, it is sufficient to consider a relaxed version of the nearest neighbour problem, its approximate version:

**Definition 7 (Approximate Nearest-Neighbour Problem).** *Given a list of real vectors $L \subseteq \mathbb{R}^n$ and a query $q \in \mathbb{R}^n$ along with distance parameter $p$ and distance function $d$, output a close neighbour $w \in L$ s.t. $d(q, w) \leq p$.*

In the following, this report describes the application of locality-sensitive hash functions based on random hyperplanes on the NV-Sieve. The main reference for this section is [17].

## Locality-sensitive hash functions

The main idea of finding near neighbours in vector spaces has already been described in the introduction of this section. Namely, the hyperspace containing the list of vectors that need to be reduced will be partitioned into different regions using hash functions, and when two vectors collide with respect to the hash function they will be likely to fall in the same region. These hash functions are called locality sensitive and are defined as in the following:

**Definition 8 ($(\mathbf{r_1}, \mathbf{r_2}, \mathbf{p_1}, \mathbf{p_2})$-sensitive hash families).** *A family of hash functions $\mathcal{H}$ is said to be a*

*$(r_1, r_2, p_1, p_2)$-sensitive hash family with respect to a distance function $d$ and a collision relation $R$ if for all $\mathbf{x}, \mathbf{y}$ in the domain of the function, it holds that:*

1. *if $d(\mathbf{x}, \mathbf{y}) \leq r_1$ then $P_{h \in \mathcal{H}}(h(\mathbf{x}) R\, h(\mathbf{y})) \geq p_1$.*
2. *if $d(\mathbf{x}, \mathbf{y}) \geq r_2$ then $P_{h \in \mathcal{H}}(h(\mathbf{x}) R\, h(\mathbf{y})) \leq p_2$.*

The collision relation $R$ is initially defined to be the equality of the hash digests. If the distance between any two vectors is smaller than threshold $r_1$, the probability of collision will be higher than some probability $p_1$, while if the vectors are far-away, the probability of collision will be smaller than some probability $p_2$. In order for the hash family to be useful, it is necessary that $p_2 \leq p_1$. This difference between these two probabilities can be amplified, through the following process:

**Amplification** By applying a combination of $AND$ or $OR$ compositions, it is possible to increase the gap between the two probabilities, in the case when $p_1 > p_2$:

- AND composition
  From a $(r_1, r_2, p_1, p_2)$-sensitive hash family $\mathcal{H}$, construct a $(r_1, r_2, p_1^k, p_2^k)$-sensitive hash family $\mathcal{H}'$ by considering $k$ independent hash functions $h_1, ..., h_k \in \mathcal{H}$ and defining $h'(\mathbf{a}) = (h_1(\mathbf{a}), ..., h_k(\mathbf{a})), \forall \mathbf{a} \in A$.

  It remains to define the collision relation, $h'(\mathbf{a})$ collides with $h'(\mathbf{b})$ if and only if $h_i(\mathbf{a}) R\, h_i(\mathbf{b})$ for all $i = \overline{1, k}$. Even though this type of composition increases the gap between the probabilities, it will also decrease their values which will mean that not even near neighbours will collide often enough. This is why it is necessary to combine this type of composition with the following one:
- OR composition
  From a $(r_1, r_2, p_1, p_2)$-sensitive hash family $\mathcal{H}$, construct a $(r_1, r_2, 1-(1-p_1)^t, 1-(1-p_2)^t)$-sensitive hash family $\mathcal{H}'$ by taking $t$ independent hash functions $h_1, ..., h_k \in \mathcal{H}$ and defining the collision relation as $h'(\mathbf{a}) R\, h'(\mathbf{b})$ if and only if $h_i(\mathbf{a}) R\, h_i(\mathbf{b})$ for at least one $i = \overline{1, t}$. The behaviour of this construction contrasts the preceding one, as it will increase the values of both probabilities but it will also decrease the existing gap between them.

## Angular hashing

In the following, a particular hash family is defined and subsequently applied on the sieving procedure using the description from [17]. First, let us start by specifying our distance measure $d(\mathbf{v}, \mathbf{w})$ for two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ as their common angle $\theta = cos^{-1}\Big(\frac{\langle \mathbf{v}, \mathbf{w} \rangle}{\|\mathbf{v}\| \|\mathbf{w}\|}\Big)$. Note that in order to satisfy all the axioms of a distance function it is actually needed to consider the normalized angular distance. Then, let us define the Charikar family of hash functions, introduced in [11] as $\mathcal{H} = \{h_{\mathbf{a}} : \mathbf{a} \in \mathbb{R}^n, \|\mathbf{a}\| = 1\}$, where each hash function actually distinguishes between the two areas delineated by the hyperplane for which $\mathbf{a}$ is a normal vector:

$$h_a(\mathbf{v}) = \begin{cases} 1, \text{if } \langle \mathbf{a}, \mathbf{v} \rangle \geq 0 \\ 0, \text{if } \langle \mathbf{a}, \mathbf{v} \rangle < 0. \end{cases}$$

Considering the collision relation $R$ to be the equality, it is also possible to explicitly state the probability of collision for two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, according to the following remark:

**Remark 1.** *[17]:When $\mathbf{a}$ is drawn at random from the unit sphere, for any $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, it holds that:*

$$P[h_a(\mathbf{v}) = h_a(\mathbf{w})] = 1 - \frac{\theta(\mathbf{v}, \mathbf{w})}{\pi}.$$

To prove this result one simply needs to take into account the probability that the projection of a unit vector uniformly sampled, into the hyperplane specified by $\mathbf{v}, \mathbf{w}$, falls in the circular sector defined by $\frac{\mathbf{v}}{\|\mathbf{v}\|}, \frac{\mathbf{w}}{\|\mathbf{w}\|}$.

## Sieving using locality sensitive hash functions

In this paragraph, the euclidean distance used in the NV-Sieve will be expressed in terms of the angular distance, which will allow for a reformulation of the sieving procedure, first due to [17]. This reformulation will consider the angular distance rather than the euclidean one for determining whether a vector can be reduced with a specific center. In fact, in the proof of Lemma 10, it has been proven that the maximal angle of the covering spherical caps is $\frac{\pi}{3}$, which leads to the following remark:

**Remark 2.** *For every radius $R$, it holds that:*

$$\|\mathbf{v}-\mathbf{w}\| \leq \gamma R \text{ implies that } \theta(\mathbf{v},\mathbf{w}) \leq \frac{\pi}{3}, \text{ for any } \mathbf{v},\mathbf{w} \in \{\mathbf{x} \in \mathbb{R}^n : \gamma R \leq \|\mathbf{x}\| \leq R\}.$$

In order to be able to say something about the converse, first notice that $\theta(\mathbf{v},\mathbf{w}) \leq \gamma R$ implies that $\|\mathbf{v}-\mathbf{w}\| \leq R$. This means that in the limiting case of $\gamma \to 1$, the converse of Remark 2 also holds and finding vectors that are at a distance smaller than $\gamma R$ is equivalent to finding vectors that are at an angle smaller than $\frac{\pi}{3}$. Taking again into account that $\gamma = 1 - \frac{1}{n}$ and $n$ is large, the corona $\{\mathbf{x} \in \mathbb{R}^n | \gamma R \leq \|\mathbf{x}\| \leq R\}$ resembles geometrically the hypersphere of radius $R$. Therefore, an equivalent statement of Heuristic 2 may be introduced:

**Heuristic 4.** *[17]: Consider a random integer lattice basis $B$ and an arbitrary radius $R \in \mathbb{R}$ with $R \geq \frac{4}{3}\lambda_1(\mathcal{L}(B))$. Then for any input or output $List$ of Sieving Algorithm 8, the distribution of angles between pairs of vectors $\mathbf{x},\mathbf{y} \in List$, such that $\gamma R \leq \|\mathbf{x}\|, \|\mathbf{y}\| \leq R$, follows the distribution of angles of uniformly random vectors on the unit sphere.*

This heuristic assumption means that we cannot work with the definition for locality sensitive functions normally used in the literature, and described in the previous section. This is due to our goal of increasing the probability of collision for vectors at angles smaller than $\frac{\pi}{3}$ and decreasing it for vectors at angles greater than $\frac{\pi}{3}$, which would leave us in the case where $p_1 = p_2$. For this reason, the $p_2$ and $r_2$ parameters will be essentially ignored and another work-around to the non-existence of a gap will be employed.

In [17], the amplification procedure is done using AND compositions of arity $k$ followed by one OR composition of arity $t$. Assuming that the optimal choice of the $k, t$ values is given, it is possible to modify the sieving procedure of the NV-Sieve, obtaining Sieving Algorithm 8. As far as the collision relation is concerned, the algorithm uses $t$ hash functions which are $k - wise$ compositions of basic angular hash functions, $h_1^i, ..., h_k^i$ for $i = \overline{1,t}$. Two vectors $\mathbf{v}$ and $\mathbf{w}$ will collide when the tuples $(h_1^i(\mathbf{v}), ..., h_k^i(\mathbf{v})) = (h_1^i(\mathbf{w}), ..., h_k^i(\mathbf{w}))$ are equal on every component for at least one $i = \overline{1,t}$. In particular, in order to detect collisions, the algorithm will use $t$ hash-tables and two vectors will collide when they are assigned to the same hash bucket in at least one of the $t$ hash-tables.

**Algorithm 8** [17]:Sieving

**Input:** Radius $R$, two sets $List$ and $Centers$ of lattice vectors of norms smaller than $R$ along with a shrinking factor $0 < \gamma < 1$.

**Output:** Another list $List'$ of lattice vectors, this time of norms smaller than $\gamma R$.

1: Construct the first $k \times t$ basic $(\frac{\pi}{3}, \frac{\pi}{3}, \frac{2}{3}, \frac{2}{3})$-sensitive functions by drawing random vectors from the unit sphere.
2: Perform $t$ AND-compositions to obtain $h_{AND_1}, ..., h_{AND_t}$ $(\frac{\pi}{3}, \frac{\pi}{3}, (\frac{2}{3})^k, (\frac{2}{3})^k)$-sensitive functions and what is abstractedly an OR composition of arity $t$.
3: Initialize $t$ hash-tables $T_1, ..., T_t$. ▷ each with potentially $2^k$ hash buckets.
4: **for** all $v$ in $Centers$ **do**
5: for $i = 1$ to $t$ add $\mathbf{v}$ to bucket $T_i[h_{AND_i}(\mathbf{v})]$.
6: **end for**
7: $List' \leftarrow \emptyset$.
8: **for** $\mathbf{v} \in List$ **do**
9: **if** $\|\mathbf{v}\| < \gamma R$ **then**
10: add vector $\mathbf{v}$ to $List'$.
11: **else**
12: **for** $\mathbf{c} \in Candidates = \cup_{i=1}^{t} T_i[h_{AND_i}(\mathbf{v})]$ **do**
13: **if** $\|\mathbf{c} - \mathbf{v}\| \leq \gamma R$ **then**
14: add $\mathbf{c} - \mathbf{v}$ to $List'$ and continue to the next $\mathbf{v} \in List$.
15: **end if**
16: **end for**
17: **end if**
18: **end for**
19: **return** $List'$.

## Complexity of LSH Sieving

Recall that the most expensive step in the NV-Sieve was the sieving, which had a quadratic complexity in the initial list size $N$, while every other step was linear. The estimates for the global complexity of the algorithm when the sieving procedure is replaced by Sieving Algorithm 8 are analysed below:

1. The time used for hashing/ space needed in line 4 of Sieving Algorithm 8 is:

$$O(N_C t k) \times O(n \log(n) 2^{O(\log^*(n))}).$$

The second term denotes the complexity of multiplication used when computing scalar products. Since it is exponentially smaller than $N$, we will omit it henceforth.

2. Given a specific $\mathbf{v} \in \mathcal{L}$, the time required when searching for a nearby center is $O(tk + N_C \times P(\text{collision for distant vectors}))$.

3. Overall complexity is $O(N \times (tk + N_C \times P(\text{collision for distant vectors}) + 1))$.

So far, the presentation was at an abstract level and it remains the same regardless of the hash functions being used or even if they are not hash functions at all, but filters as in [8]. It remains to choose values for $p_1, p_2, r_1$ and $r_2$. For $r_1$, it follows from Remark 2 that it can be chosen to be $\frac{\pi}{3}$, which implies $p_1 = \frac{2}{3}$. As mentioned before, assuming Heuristic 4 makes parameters $r_2$ and $p_2$ superfluous and are discarded. In order to ensure that the correctness properties of the NV-Sieve are preserved, it is necessary to show that the following two properties hold for the particular $t$ and $k$ values used by the algorithm:

**Property 1.** *[17, Adapted from Equation 11]: The average probability of collision for any $\mathbf{x}, \mathbf{y}$ near vectors (a vector pair which leads to a reduction as their common angle $\theta(\mathbf{x}, \mathbf{y})$ is less or equal to $\frac{\pi}{3}$) is indeed exponentially close to 1. What is therefore asked is that*

$$P_1 \stackrel{def}{=} E\Big[1 - \Big(1 - \Big(1 - \frac{\theta}{\pi}\Big)^k\Big)^t \mid \theta \leq \frac{\pi}{3}\Big] \geq 1 - error, \text{ where } error = \frac{1}{2^n}.$$

**Property 2.** *[17]: The average probability of collision for any $\mathbf{x}, \mathbf{y}$ two distant vectors (they are irreducible, thus their common angle $\theta(\mathbf{x}, \mathbf{y})$ is strictly greater than $\frac{\pi}{3}$) is exponentially small w.r.t. $n$, which would imply that the algorithm does not have to process too many false candidates. We consider the list size $N = 2^{O(n)}$, namely it is asked that the upper bound is $N^{-\alpha}$, for some constant $\alpha$ with $0 < \alpha < 1$. This is equivalent to requiring that*

$$P_2 \stackrel{def}{=} E\Big[1 - \Big(1 - \Big(1 - \frac{\theta}{\pi}\Big)^k\Big)^t \mid \theta > \frac{\pi}{3}\Big] \leq N^{-\alpha}.$$

First let us analyse why is Property 1 needed. Recall that in Section 4 it has been shown that the NV-Sieve is optimal when $N$ and $N_C$ are of the form $\big(\frac{4}{3}\big)^{\frac{n}{2}+o(n)}$. Using LSH hashing invariably results in a number of possible reductions being skipped because the close vectors did not collide in the hash tables. The following result shows how to ensure that a sufficient number of lattice vectors reduce at each iteration of the sieve:

**Lemma 14.** *Consider the modified Sieving Algorithm 8 where $k$ and $t$ are chosen in such a way that the average error probability of two close vectors not colliding is $\frac{1}{2^n}$. Then its output list has approximately the same size as the output of the procedure without hashing, as in Sieving Algorithm 6. Moreover, with probability that tends to 1 we have at most a polynomial number of irreducible lattice vectors that are lost during all the executions of the sieving sub-algorithm.*

*Proof.* In fact, ensuring that all the vectors which are not centers reduce compels us to increase the size of the list of centers, which in turn results in an increase in the size of the list to sieve as well. Start by examining what happens only at one execution of the sieve, increasing the input size $|List|$ by a factor $f \stackrel{def}{=} \frac{1}{1-error}$ yields that the expected number of reduced vectors outputted is $|List|$. We use Markov's inequality and we obtain that the probability to lose more than a polynomial fraction of lattice vectors $q(n)$ at one step is bounded from above by $\frac{error}{q(n)}$. Then, when $p(n)$ denotes the number of steps, the probability to lose not more than a polynomial number of vectors overall is greater than $(1 - \frac{error}{q(n)})^{p(n)} =_{n\to\infty} 1$ if we choose $q(n) = p(n)$. To conclude, we know that we need to apply the growth $p(n)$ times, nevertheless $f^{p(n)} = (\frac{1}{1-error})^{p(n)}$ collapses to 1 as well, as $n$ goes to infinity. □

In contrast, ensuring that Property 2 holds is quite natural if one attempts to reduce the number of possibly bad candidates recovered using LSH. In the following let us focus on parameters $k$ and $t$, whose optimization has the two following goals:

1. Minimize overall complexity

$$O(N \times (tk + N_C \times P_2)).$$

2. While maximizing the output of a sieve execution:

$$N' = \Theta((N - N_C) \times P_1).$$

Here, $P_1$ is the average probability that near vectors collide and $P_2$ is the average probability that distant vectors collide. For brevity, let us define the collision probability for vectors at angle $\theta$ as $f(\theta) \stackrel{def}{=} 1 - (1 - (1 - \frac{\theta}{\pi})^k)^t$. This is a decreasing function, therefore it holds that $P_1 \geq f(\frac{\pi}{3})$. A

pertinent question is whether it makes sense to use different parameters $k$ and $t$ at each execution of the sieve, but this is not the case as the input sizes to each call of the sieving procedure are of exponential size with respect to $n$ and they differ only by polynomial factors, which means that we are dealing with essentially the same case during the execution of each sieving procedure.

#### Maximizing $\mathbf{N'}$, the number of reduced vectors

We want that almost all close vectors collide with probability of error exponentially small, thus we can start by ensuring that $P_1 \to 1$. We have that $P_1 \geq (1-(1-p_1^k)^t)$, with $p_1 = \frac{2}{3}$. This inequality is tight, as angles lower than $\frac{\pi}{3}$ are concentrated near $\frac{\pi}{3}$. So, we need that $(1-(1-p_1^k)^t) \to 1$. It is a reasonable assumption to consider that $k, t$ are functions depending on $n$, and then three cases are distinguished:

- Case 1: $t = \Theta((\frac{1}{p_1})^k)$, i.e. $t = c((\frac{1}{p_1})^k)$ for some real positive constant $c$. Then $\lim_{n\to\infty}(1-(1-p_1^k)^t) = 1 - \frac{1}{e^c}$, which is not close enough to 1.
- Case 2: $t \ll ((\frac{1}{p_1})^k)$. Asking that $P_1$ goes to 1 is equivalent to requiring that $(1-p_1^k)^t$ goes to 0, which is again equivalent to $t\log(1-p_1^k) \to -\infty$.

  Now, when $0 < x < 0.5828$, we know that $-\frac{3x}{2} < \log(1-x) < \frac{3x}{2}$ from [1, 4.1.35]. When $x = p_1^k$, we will certainly have $t\log(1-p_1^k) \geq t \times -\frac{3}{2}p_1^k$, but we want that the first quantity goes to $-\infty$, which will happen when $\frac{t}{(\frac{1}{p_1})^k} \to \infty$. This suggests that $t$ has to be exponential in $k$, and $k \ll \log_{1/p_1}(t)$, which contradicts the initial assumption. This means that we are left with the following case:
- Case 3: $t \gg ((\frac{1}{p_1})^k)$, then indeed $(1-(\frac{1}{p_1})^k)^t$ goes to 0. Thus, $k \ll \log_{1/p_1}(t)$. Now, in order to ensure Property 1, we need to require that $(1-(1-p_1^k)^t) \geq 1 - error$, with error equal to $\frac{1}{2^n}$. Taking $k = \log_{1/p_1}(t) - \log_{1/p_1}(\ln(\frac{1}{error}))$ ensures that indeed $k$ is much smaller than the logarithm of $t$. Moreover, we also satisfy Property 1, as:

$$(1-(1-p_1^k)^t) = 1 - \left(1 - p_1^{\log_{1/p_1}(t) - \log_{1/p_1}(\ln(\frac{1}{error}))}\right)^t = \left(1 - \frac{\ln(1/error)}{t}\right)^t.$$

  Using, as in [17] that $1-x < e^{-x}$, it naturally follows that the expression above is larger than $1 - e^{-\ln(1/error)} = 1 - error$. Having such a small error will allow us through Lemma 14 to say that the initial list size does not have to increase when compared to the initial list size of the NV-Sieve. As a remark, using polynomially small error will imply an increase in the initial list size by $e$. It is possible now to conclude this paragraph and move on to the next goal.

#### Minimizing the overall complexity $\mathbf{O(N \times (tk + N_C \times P_2))}$

Our goal is to obtain a sub-quadratic complexity in $N$, this means that we need:

1. $tk \leq N$ and since $t$ is much bigger than $k$ we mainly need that $t < N$.
2. $NN_CP_2 < NN_C$. Since $N_C$ differs only by a polynomial factor from $N$ we can ask that $N^2P_2 < N^2$. To simplify computations, it is necessary to define an optimization factor $0 < \alpha < 1$ such that $P_2 = N^{-\alpha}$.

Again, the possible values of the parameters must be analysed case by case. First, consider $t \gg 2^k$. Then $P_2 \to 1$, which is far too large. In fact, in order to be able to upper bound $P_2$, $t$ must be much smaller than $(1-\frac{\theta}{\pi})^{-k}$ for most angles $\theta$ in $[\frac{\pi}{3}, \pi]$, except possibly for angles in a small neighbourhood of $\frac{\pi}{3}$. Let us call the diameter of this neighbourhood $2\psi$ and defer the analysis for angles in this small set. First, we need to analyse the probability density function of angles following the distribution from Heuristic 4:

**Lemma 15.** *[17, Lemma 4] Under the assumption that Heuristic 4 holds, the probability density function $g(\theta)$ of angles between pairs of list vectors satisfies:*

$$g(\theta) = poly(n)2^{n\log_2(\sin(\theta))+o(n)}.$$

For brevity reasons, we do not restate the proof of this lemma. Back to the optimization, recall that $f(\theta) \stackrel{def}{=} 1-(1-(1-\frac{\theta}{\pi})^k)^t$. Then, $P_2 = E(f(\theta) \mid \theta > \frac{\pi}{3}) = \frac{\int_{\pi/3}^{\pi} g(\theta)f(\theta)d\theta}{\int_{\pi/3}^{\pi} g(\theta)d\theta}$. It would be convenient to have that $t$ is much smaller than $(1-\frac{\theta}{\pi})^{-k}$, as this would allow us to approximate $f(\theta)$ by $t(1-\frac{\theta}{\pi})^k$. However, this is not true for all the angles, but only for angles significantly bigger than $\frac{\pi}{3}$. So we limit the angles for which the approximation does not hold in a small neighbourhood of radius $\psi$ around $\frac{\pi}{3}$, obtaining that:

$$P_2 = \frac{1}{\int_{\frac{\pi}{3}}^{\pi} g(\theta)d\theta}\Big[\underbrace{\int_{\frac{\pi}{3}}^{\frac{\pi}{3}+\psi} g(\theta)f(\theta)d\theta}_{J} + \underbrace{\int_{\frac{\pi}{3}+\psi}^{\frac{\pi}{2}} g(\theta)f(\theta)d\theta}_{I}\Big].$$

We describe now the strategy employed by [17] to find the optimal $\alpha$ for which $P_2 \leq N^{-\alpha}$. Note first that $\int_{\frac{\pi}{3}}^{\pi} g(\theta)d\theta = 1 - \int_0^{\frac{\pi}{3}} g(\theta)d\theta \geq 1 - \frac{\pi}{3}\big(\frac{\sqrt{3}}{2}\big)^{n+o(n)} \to 1$. Therefore, analysing $P_2$ is equivalent to looking only at the $I$ and $J$ integrals. Secondly, they show that for small enough $\psi$, it holds that $J \leq N^{-1}$. Then, unimpeded by the $J$ integral, they compute the optimal $\alpha$ for which $I \leq N^{-\alpha}$. The separation of $P_2$ into two integrals is the work around the non-existence of the gap between the $r_1$ and $r_2$ in our locality sensitive hash function.

### Bounding the I integral

As previously mentioned, we want that $t$ is much smaller than $(1-\frac{\theta}{\pi})^{-k}$ for angles $\theta$ larger than $\frac{\pi}{3}+\psi$. An appropriate choice for $\psi$ would then be $\frac{\pi}{\sqrt{n}}$. Indeed, $t$ is much smaller than $(\frac{2}{3}-\frac{1}{\sqrt{n}})^{-k}$. This is due to our choice of $k$, as for $k = \log_{1/p_1}(t) - \log_{1/p_1}(error)$, it holds that:

$$\lim_{n\to\infty} \frac{(\frac{2}{3}-\frac{1}{\sqrt{n}})^{-k}}{t} = +\infty.$$

It is interesting to note that a smaller neighbourhood of radius, say $\frac{1}{n}$, would not ensure that the equation above holds. Proceed now to bounding the $I$ integral itself. Since $t$ is much smaller than $(\frac{2}{3}-\frac{1}{\sqrt{n}})^{-k}$, it is possible to use the approximation $1-(1-(1-\frac{\theta}{\pi})^k)^t \approx t(1-\frac{\theta}{\pi})^k$, which would mean that:

$$I = \int_{\pi/3+\psi}^{\pi} g(\theta)f(\theta)d\theta \approx poly(n)\int_{\pi/3+\psi}^{\pi} 2^{n\log_2(\sin(\theta))} t\Big(1-\frac{\theta}{\pi}\Big)^k d\theta.$$

This is in turn smaller than

$$\int_{\pi/3+\psi}^{\pi} 2^{n\log_2(\sin(\theta))+k\log_2(1-\theta/\pi)+\log_2(t)+o(n)}d\theta.$$

For brevity we write $c_n \stackrel{def}{=} \log_2(N)/n$ and $c_t \stackrel{def}{=} \log_2(t)/n$ and we define parameter $\alpha$ as in the following:

$$\alpha \stackrel{def}{=} -\frac{c_t n + \max_{\theta\in(\frac{\pi}{3},\pi]}\Big[n\log_2(\sin(\theta)) + k\log_2(1-\theta/\pi)\Big] + o(n)}{c_n n}. \tag{1}$$

Using the dependency between $k = \frac{c_t n}{\log_2(1/p_1)} - \log_{1/p_1}(n\ln(2))$, the following expression is obtained:

$$\alpha \stackrel{def}{=} -\frac{c_t + \max_{\theta\in(\frac{\pi}{3},\pi]}\Big[\log_2(\sin(\theta)) + c_t\log_{1/p_1}(1-\theta/\pi)\Big]}{c_n} + o(1). \tag{2}$$

This completes the bounding of the $I$ integral by $N^{-\alpha}$. We proceed now to bounding the $J$ integral.

### Bounding the J integral

First notice that $g(\theta) \leq g(\frac{\pi}{3} + \psi)$, and we obtain:

$$J = \int_{\pi/3}^{\frac{\pi}{3}+\psi} g(\theta) f(\theta) d\theta \leq \psi g\Big(\frac{\pi}{3} + \psi\Big).$$

Using a Taylor expansion of $\sin(\theta)$ in $\frac{\pi}{3}$ yields $\sin(x) = \sqrt{\frac{3}{4}} + O(x - \frac{\pi}{3})$ and therefore $\sin(\frac{\pi}{3} + \psi) = \sqrt{\frac{3}{4}}(1 + O(\psi))$ for subunitary $\psi$. When $\psi = \frac{\pi}{\sqrt{n}}$ for instance, the term $(1 + O(\frac{1}{\sqrt{n}}))^n$ can be expressed as $2^{o(n)}$. Therefore, J is upperbounded by $2^{-\frac{n}{2}\log(4/3)+o(n)}$. Now, recall that from the complexity analysis of Section 4, $N$ and $N_C$ must belong to $2^{\frac{n}{2}\log(4/3)+o(n)}$, which means that the integral is upperbounded by $\frac{1}{N}$. This is lower than $N^{-\alpha}$ computed above.

### Computing the optimal complexity

Please recall that:

$$P_2 = \frac{1}{\int_{\frac{\pi}{3}}^{\pi} g(\theta) d\theta} \Big[I + J\Big].$$

First, we have already shown that $\int_{\frac{\pi}{3}}^{\pi} g(\theta) d\theta \geq 1 - \frac{\pi}{3}\big(\frac{\sqrt{3}}{2}\big)^{n+o(n)}$. As the choice of $N$ is $\Big(\frac{4}{3}\Big)^{\frac{n}{2}+o(n)}$, this implies that $\frac{1}{\int_{\frac{\pi}{3}}^{\pi} g(\theta) d\theta} \leq \frac{N}{N-1}$. Also, from the previous two paragraphs, we know that the $I$ and $J$ integrals are both smaller than $N^{-\alpha}$, with $\alpha$ defined as in Equation 2. This means that we obtain $P_2 = E(f(\theta)|\theta > \frac{\pi}{3}) \leq \frac{2N}{N-1} N^{-\alpha}$. The $o(n)$ factor in the definition of $N$ gives us the leeway to conclude that $P_2 \leq N^{-\alpha}$. Let us recall the complexity of our algorithm, since both $N$ and $N_C$ are of the form $2^{c_n n+o(n)}$, it follows that:

1. Time for hashing/ Space needed in step line 4: $\tilde{O}(Nt) = 2^{(c_n+c_t)n+o(n)}$, the actual time of computing just one basic hash will be reflected in $o(n)$.
2. Time for searching $\tilde{O}(NN_C P_2 + 1) = 2^{n(c_n+(1-\alpha)c_n)+o(n)}$, because $N$ and $N_C$ differ by a polynomial factor corresponding to the number of sieve iterations.
3. Overall time: $2^{c_n+max\{c_t,(1-\alpha)c_n\}+o(n)}$.

To compute the minimal complexity of the algorithm, we use the value of $\alpha$ from Equation 2 and we minimize numerically with respect to $c_t \in [0, c_n)$ the expression $c_n + \max(c_t, (1-\alpha)c_n)$ for $c_n = 0.2075$. We obtain that the exponent for the time complexity is equal to $0.33653n$ when $c_t = 0.12903$, $k = \log_{1/p_1}(t) - \log_{1/p_1}(n/1.4427) \approx \log_{1/p_1}(t) = 0.22058n$ and $\alpha = 0.37816$.

Actually, the previous computations plugged into the proof of theorem 4 make up the proof of the following theorem, due to [17]:

**Theorem 5.** *[17, Theorem 1]: As $n \to \infty$, and under the assumptions of Heuristics 3 and 4, the NV-Sieve algorithm using Sieving Algorithm 8 solves the $\frac{4}{3}$ approximate-shortest vector problem in time and space $2^{0.33653n+o(n)}$ with parameters $N = poly(n)\Big(\frac{4}{3}\Big)^{\frac{n}{2}} = 2^{0.2075n+o(n)}$, $t = 2^{0.12903n+o(n)}$ and $k \approx 0.22058n$.*

**Computing values of t and k using numerical methods** In order to verify whether the values computed above are indeed optimal, as the analysis suggests, we have used different numerical strategies to determine appropriate values of $t$ and $k$ for moderately large dimensions. Since the domain of the $t$ parameter is exponential, the principal approach has been to use the NMinimize function from Wolfram Mathematica for different values of $t$, with $t$ growing exponentially, but

this results in overall complexities which are much larger than theoretical predictions, due to the asymptotic behaviour of the solution. The optimization methods which have been used are the Nelder Mead, the Differential Evolution and the Random Search algorithms offered by the Mathematica language [31].

## Different configurations of AND and OR gates

While successive applications of the same type of gate clearly can be collapsed into one level, it is not clear whether utilizing other configurations than the ones used in the existing in the literature yields better results. A first approach would be to show that more levels can be expressed in two levels but presently, this remains unproven. In the following, we state the probabilities for the remaining case of two levels as well as the three levelled case. In order to simplify the presentation, we assume the presence of a gap in list angles between $\frac{\pi}{3}$ and $\frac{\pi}{2}$. In particular, the parameters must be chosen in such a way such that the following relations hold:

**An OR gate followed by an AND gate** Let $t$ be the arity of all the OR gates, which are followed by an AND gate of arity $k$. Then, the following two expressions must be bounded:

1. $E\big((1-(\frac{\theta}{\pi})^t)^k \mid \theta \leq \frac{\pi}{3}\big) \to 1$.
2. $E\big((1-(\frac{\theta}{\pi})^t)^k \mid \theta > \frac{\pi}{3}\big) \leq N^{-\alpha}$.

First, it is required that $(1-\frac{1}{3^t})^k \to 1$, which is equivalent to $\frac{k}{3^t} \to 0$. Thus it follows that $k \ll 3^t$. A reasonable choice for $k$ is $3^{t-\log(n)}$ as it will ensure that $(1-(\frac{1}{3})^t)^k \to 1$.

For the second condition, it is possible to define $f(\theta) \stackrel{def}{=} (1-(\frac{\theta}{\pi})^t)^k$, which is decreasing. For simplification we assume a gap in angles between $\frac{\pi}{3}$ to $\frac{\pi}{2}$, and we only need to analyse $(1-\frac{1}{2^t})^k$. But,

$$\left(1-\frac{1}{2^t}\right)^k = 2^{k\log(1-\frac{1}{2^t})} \approx 2^{-k\frac{1}{2^t}}.$$

This means that $\alpha$ must be defined as $\alpha \stackrel{def}{=} \frac{k}{2^t c_n n} = 3^{t(1-\log_3(2))-\log(n)}\frac{1}{c_n n}$. Unfortunately, unlike the case $OR$ followed by $AND$, it does not seem to be possible to express $\alpha$ as a function that does not depend on $n$. The same problem persists if $\alpha$ is expressed in terms of $k$, namely $\alpha = \frac{k^{(1-\log_3(2))}}{c_n n^2}$.

**Three levels, starting with AND** We denote the arities of the AND gates as $k_1$ and $k_2$ and the arity of the OR gate as $t$:

1. $E\big([1-(1-(1-\frac{\theta}{\pi})^{k_1})^t]^{k_2} \mid \theta \leq \frac{\pi}{3}\big) \to 1$.
2. $E\big([1-(1-(1-\frac{\theta}{\pi})^{k_1})^t]^{k_2} \mid \theta > \frac{\pi}{3}\big) \leq N^{-\alpha}$.

In order to satisfy the first condition it is sufficient to consider $k_2 = \frac{(1-(1-\frac{\theta}{\pi})^{k_1})^{-t}}{n}$. Then, the second expression can be written as $1-k_2(1-(1-\frac{\theta}{\pi})^{k_1})^t$. This means that $2^{\log_2(1-k_2(1-(1-\frac{\theta}{\pi})^{k_1})^t)} \leq 2^{-\alpha c_n n}$, but the second expression in the logarithm is not small, in fact it needs to be close to 1, so we cannot approximate it easily.

**Three levels, starting with OR** We denote the arities of the OR gates as $t_1$ and $t_2$ and the arity of the AND gate as $k$:

1. $E\big((1-[1-(1-(\frac{\theta}{\pi})^{t_1})^k]^{t_2}) \mid \theta \leq \frac{\pi}{3}\big) \to 1$.

2. $E\big((1-[1-(1-(\frac{\theta}{\pi})^{t_1})^k]^{t_2}) \mid \theta > \frac{\pi}{3}\big) \leq N^{-\alpha}$.

In order to satisfy the first condition it is enough to consider $t_2 = \frac{(1-(\frac{\theta}{\pi})^{t_1})^{-k}}{n}$. For the second one we assume a gap in the distribution of angles between $\frac{\pi}{3}$ and $\frac{\pi}{2}$. The second expression could then be transformed into $t_2(1-\frac{1}{2^{t_1}})^k = 2^{\log(t_2)-\frac{k}{2^{t_1}}}$, which means that $\alpha$ should be defined as $\frac{\log(t_2)-(\frac{k}{2^{t_1}})}{c_n n}$.

The main difficulty in computing the optimal complexity for the gate configurations aforementioned is that these cases seem to differ significantly from the AND-then-OR case presented first and used in [17]. The difficulty in the cases where the first gate is an OR stems from the fact that the term corresponding to the dimension $n$ does not vanish in the expressions we have computed for $\alpha$. In parallel, for the configurations that start with an AND gate, it is not clear how to balance the three parameters involved and we leave the treatment of these supplementary gate configurations as an open question.

# 6 Conclusion and future work

This report summarizes a great part of the work carried out during my internship. The approach we took to our problem was mainly theoretical, which is why we also focused our attention on studying the provable sieving algorithms in the literature and not exclusively the heuristics. Other than the improved analysis for approximate-SVP which will hopefully constitute a future publication, we have also analysed in details the mathematical fundamentals of state of the art sieving heuristics, along with our idea for extending the amplification procedure of the LSH functions. Another notable idea explored was whether the AKS algorithm itself could be improved using LSH. Apart from the fact that any provable result using such hash functions requires some known distribution on the lattice vectors being sampled (the AKS algorithm itself does not assume anything on said distribution), it is also difficult to design an LSH function for the hyperball rather than the hypersphere. For brevity reasons and to avoid redundancy, we do not expand on the summary or the contributions, they have already been detailed in the corresponding paragraphs in the introduction to this report.

# 7 Acknowledgements

I would like to thank my supervisor, Prof. Vaudenay, for the tremendous help and patience bestowed on me over the course of my internship, especially for his great attention to detail and mathematical insight. I am also very appreciative of the great support offered me by Divesh Aggarwal and Anja Becker, for the discussions on the fundamentals of lattice theory and for always posing me with new problems to think about. I am grateful as well to Thijs Laarhoven for his timely answers on various aspects pertaining to the locality sensitive hashing improvements. My thanks also go to Sonia Bogos for proofreading this report and to the entire LASEC laboratory in general for the interesting seminar sessions and for making me feel so welcome.

# Appendix A References


[1] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Dover Publications, 2012. https://books.google.ch/books?id=KiPCAgAAQBAJ.

[2] Divesh Aggarwal, Daniel Dadush, and Noah Stephens-Davidowitz. Solving the closest vector problem in $2^n$ time - the discrete gaussian strikes again! *CoRR*, 2015. http://arxiv.org/abs/1504.01995.

[3] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC (Symposium on Theory of Computing)*, pages 99–108, New York, NY, USA, 1996. ACM. http://doi.acm.org/10.1145/237814.237838.

[4] Miklós Ajtai. The Shortest Vector Problem in L2 is NP-hard for randomized reductions (extended abstract). In *STOC (Symposium on Theory of Computing)*, pages 10–19, New York, NY, USA, 1998. ACM. http://doi.acm.org/10.1145/276698.276705.

[5] Miklos Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC (Symposium on Theory of Computing)*, pages 266–275. ACM, July 2001. https://dl.acm.org/citation.cfm?id=380857.

[6] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. http://dx.doi.org/10.1007/BF02579403.

[7] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. Cryptology ePrint Archive, Report 2016/713, 2016. http://eprint.iacr.org/2016/713.

[8] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24. http://epubs.siam.org/doi/abs/10.1137/1.9781611974331.ch2.

[9] Anja Becker, Nicolas Gama, and Antoine Joux. A sieve algorithm based on overlattices. *LMS Journal of Computation and Mathematics*, 17:49–70, 2014. http://journals.cambridge.org/article_S1461157014000229.

[10] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In *AFRICACRYPT*, pages 3–23, 2016. http://dx.doi.org/10.1007/978-3-319-31517-1_1.

[11] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC (Symposium on Theory of Computing)*, pages 380–388, 2002. http://doi.acm.org/10.1145/509907.509965.

[12] V. I. Levenshtein G. A. Kabatiansky. On bounds for packings on a sphere and in space. *Probl. Peredachi Inf.*, 14(1):3–25, 1978. http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=ppi&paperid=1518&option_lang=eng.

[13] N. Gama and P. Baumann. SVP challenge, 2016. https://latticechallenge.org/svp-challenge/.

[14] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. *Coding and Cryptology: Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings*, chapter Algorithms for the Shortest and Closest Lattice Vector Problems. Springer Berlin Heidelberg, 2011. http://dx.doi.org/10.1007/978-3-642-20901-7_10.

[15] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *CRYPTO*, pages 170–186, 2007. http://dx.doi.org/10.1007/978-3-540-74143-5_10.

[16] Philip Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 937–941, 2000. http://dl.acm.org/citation.cfm?id=338219.338661.

[17] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO 2015*, pages 3–22. Springer Berlin Heidelberg, 2015. http://dx.doi.org/10.1007/978-3-662-47989-6_1.

[18] Thijs Laarhoven. Search problems in cryptography : from fingerprinting to lattice sieving (phd thesis). *Eindhoven : Technische Universiteit Eindhoven*, 2016. http://repository.tue.nl/837539.

[19] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *LATINCRYPT*, pages 101–118, 2015. http://dx.doi.org/10.1007/978-3-319-22174-8_6.

[20] Thijs Laarhoven, Michele Mosca, and Joop Pol. Finding shortest lattice vectors faster using quantum search. *Des. Codes Cryptography*, 77(2-3):375–400, December 2015. http://dx.doi.org/10.1007/s10623-015-0067-5.

[21] Mingjie Liu, Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. Shortest lattice vectors in the presence of gaps. Cryptology ePrint Archive, Report 2011/139, 2011. http://eprint.iacr.org/.

[22] Daniele Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, March 2001. http://cseweb.ucsd.edu/~daniele/papers/SVP.html.

[23] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC (Symposium on Theory of Computing)*, pages 351–358, New York, NY, USA, 2010. ACM. http://doi.acm.org/10.1145/1806689.1806739.

[24] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC (Symposium on Theory of Computing)*, pages 351–358, 2010. https://dl.acm.org/citation.cfm?id=1806739&CFID=799946185&CFTOKEN=37031878.

[25] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA*, pages 1468–1480. ACM/SIAM, 2010. http://epubs.siam.org/doi/abs/10.1137/1.9781611973075.119.

[26] P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, pages 181–207, July 2008. ftp://ftp.di.ens.fr/pub/users/pnguyen/JoMC08.pdf.

[27] Xavier Pujol and Damien Stehle. Solving the shortest lattice vector problem in time $2^{2.465n}$. Cryptology ePrint Archive, Report 2009/605, 2009. https://eprint.iacr.org/2009/605.

[28] Oded Regev. Lecture 1- Introduction. *Lecture notes on lattices in computer science*, 2004. https://www.cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/introduction.pdf.

[29] Oded Regev. Lecture 2 - LLL algorithm. *Lecture notes on lattices in computer science*, 2004. https://www.cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/lll.pdf.

[30] Oded Regev. Lecture 8 - a $2^{O(n)}$ time algorithm for SVP. *Lecture notes on lattices in computer science*, 2004. http://www.cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/svpalg.pdf.

[31] Wolfram Research, Inc. Mathematica reference for NMinimize, 2016. https://reference.wolfram.com/language/ref/NMinimize.html.

[32] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In *ASIACCS*, pages 1–9, 2011. http://doi.acm.org/10.1145/1966913.1966915.

[33] Wei Wei, Mingjie Liu, and Xiaoyun Wang. Finding shortest lattice vectors in the presence of gaps. In *Topics in Cryptology — CT-RSA 2015*, pages 239–257. Springer International Publishing, April 2015. http://dx.doi.org/10.1007/978-3-319-16715-2_13.

[34] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the shortest vector problem. In *SAC*, pages 29–47, 2014. http://dx.doi.org/10.1007/978-3-662-43414-7_2.