Algorithms for the Shortest and Closest Lattice Vector Problems

Corrected version – 02/01/2013

Guillaume Hanrot and Xavier Pujol and Damien Stehlé

Laboratoire LIP (U. Lyon, CNRS, ENS Lyon, INRIA, UCBL),
46 Allée d'Italie, 69364 Lyon Cedex 07, France.
{guillaume.hanrot,xavier.pujol,damien.stehle}@ens-lyon.fr

Abstract. We present the state of the art solvers of the Shortest and Closest Lattice Vector Problems in the Euclidean norm. We recall the three main families of algorithms for these problems, namely the algorithm by Micciancio and Voulgaris based on the Voronoi cell [STOC'10], the Monte-Carlo algorithms derived from the Ajtai, Kumar and Sivakumar algorithm [STOC'01] and the enumeration algorithms originally elaborated by Kannan [STOC'83] and Fincke and Pohst [EUROCAL'83]. We concentrate on the theoretical worst-case complexity bounds, but also consider some practical facets of these algorithms.

1 Introduction

The Shortest Lattice Vector Problem (SVP) consists in finding $\boldsymbol{x} \in \mathbb{Z}^n \setminus \mathbf{0}$ minimizing $\|B \cdot \boldsymbol{x}\|$, where $B \in \mathbb{Q}^{m \times n}$ is given as input. The Closest Lattice Vector Problem (CVP) consists in finding $\boldsymbol{x} \in \mathbb{Z}^n$ minimizing $\|B \cdot \boldsymbol{x} - \boldsymbol{t}\|$, where $B \in \mathbb{Q}^{m \times n}$ and $\boldsymbol{t} \in \mathbb{Q}^m$ are given as inputs.[1] In this survey, we will restrict ourselves to the Euclidean norm $\|\boldsymbol{y}\| = \sqrt{\sum_{i \leq m} y_i^2}$. These optimization problems admit simple geometric interpretations: SVP consists in finding a shortest non-zero vector in the Euclidean lattice $L[B] := \sum_{i \leq n} x_i \boldsymbol{b}_i$ spanned by the columns $(\boldsymbol{b}_i)_i$ of B, whereas CVP consists in finding a vector of $L[B]$ closest to the given target $\boldsymbol{t}$.

SVP and CVP have been investigated in mathematics for more than a century, with, among others, the works of Hermite [38], Korkine and Zolotarev [46], Minkowski [59] and Voronoi [81]. However, the algorithmic study of lattices took off rather lately, at the beginning of the 1980's. At

[1] Wlog we will assume that $\boldsymbol{t}$ belongs to the span of the columns of B, as otherwise it suffices to solve CVP for the orthogonal projection of $\boldsymbol{t}$ onto it.

that time, lattices happened to be a bottleneck in combinatorial optimization and in algorithmic number theory, and ground-breaking results were then obtained: A. Lenstra, H. Lenstra Jr. and L. Lovász proposed the first polynomial-time approximation algorithm for SVP [47], whereas P. van Emde Boas showed that the decisional variant of CVP, is NP-hard [19] (as well as the decisional variant of SVP for the infinity norm). Shortly afterwards, Fincke and Pohst [20, 21] and Kannan [42, 43] described the first SVP and CVP solvers. Following the genesis of lattice-based cryptography in the mid-1990's [4, 39, 27], whose security provably/heuristically relies on the hardness of variants of SVP and CVP, much effort was spent devising faster solvers. This resulted in the construction of a new type of SVP and CVP algorithms by Ajtai, Kumar and Sivakumar [5, 6]. More recently, yet another completely different algorithm was introduced by Micciancio and Voulgaris [57, 56].

SVP and CVP are common in many fields of computational mathematics and computer science. We have already mentioned combinatorial optimization and algorithmic number theory. We refer the interested reader to the surveys [17] and [48]. They are also very frequent in communications theory [60, 1, 34, 61]. The cryptographic role of SVP and CVP is twofold. In the last few years, a number of cryptographic primitives have been devised along with proofs of security under the assumption that there is no (probabilistic and sometimes quantum) polynomial-time algorithm for solving arbitrary instances of variants of SVP and CVP. Attempting to solve SVP and CVP allows to assess the validity of these assumptions. We refer to [55, 72] for recent accounts on lattice-based cryptography. On the other hand, the best known algorithm for breaking these cryptographic schemes as well as a number of other cryptographic functions such as some based on the knapsack problem [66] attempt to find short or close vectors from a relevant lattice, by reducing it. Lattice reduction consists in starting from a basis B and trying to improve its quality, traditionally measured by the orthogonality of its vectors. The most famous lattice reduction algorithm is probably LLL (see the book [64]). It runs in polynomial time but it only provides vectors that are no more than exponentially longer (in n) than the shortest ones. This worst-case behavior seems to also hold in practice [63] (up to a constant factor in the exponent). LLL may also be used to find lattice vectors relatively close to target vectors [7], but these vectors may be exponentially further from the target than the optimal solution(s). Schnorr's hierarchy [74] of reduction algorithms allows to achieve a continuum between LLL and exact SVP and CVP solvers. The best known theoretical variant (in terms of achieved basis quality for any

fixed computational cost) is due to Gama and Nguyen [23]. However, in practice, the heuristic and somewhat mysterious BKZ algorithm from [75] is used instead (see [24] for a detailed account on the practical behavior of BKZ). All known realizations of Schnorr's hierarchy (see the surveys [62, 73]) rely on an algorithm that solves SVP for smaller-dimensional lattices.

From a computational hardness perspective, the decisional variant of CVP is known to be NP hard [19], whereas the decisional variant of SVP is only known to be NP hard under randomized reductions [2] (see also the book [54]). This remains the case for the relaxed variants GapSVP_γ and GapCVP_γ for any constant $\gamma \geq 1$, where the optimal value of $\|B \cdot \boldsymbol{x}\|$ (resp. $\|B \cdot \boldsymbol{x} - \boldsymbol{t}\|$) is known to be either ≤ 1 or $\geq \gamma$ (see [44, 35]). When $\gamma = \Omega(\sqrt{n})$, GapSVP_γ and GapCVP_γ belong to NP $\cap$ coNP, and are thus unlikely to be NP hard [70]. Schnorr's hierarchy coupled with the SVP solver from [57, 56] allows one to solve GapSVP_γ and GapCVP_γ for $\gamma = k^{O(n/k)}$ in time $2^{O(k)}$. In particular, the relaxation factor $\gamma = 2^{c\frac{n \log\log n}{\log n}}$ can be achieved in polynomial time [74] for any constant $c > 0$. It is also worth noting that there is a dimension-preserving deterministic polynomial-time reduction from GapSVP_γ to GapCVP_γ for any γ (see [28]).

Three families of SVP and CVP solvers. There exist three main families of SVP and CVP solvers, which we compare in Table 1. Describing them is the purpose of the present survey.

The algorithm by Micciancio and Voulgaris [57, 56] aims at computing the Voronoi cell of the lattice, whose knowledge facilitates the tasks of solving SVP and CVP. This algorithm allows one to solve SVP and CVP deterministically, in time $\leq 2^{2n+o(n)}$ and space $\leq 2^{n+o(n)}$. We will describe this algorithm in Section 3.

Table 1. Comparing the three families of SVP and CVP solvers.

	Time complexity upper bound	Space complexity upper bound	Remarks
Sec. 3	$2^{2n+o(n)}$	$2^{n+o(n)}$	Deterministic
Sec. 4, SVP	$2^{2.465n+o(n)}$	$2^{1.325n+o(n)}$	Monte-Carlo
Sec. 4, CVP	$(2+1/\varepsilon)^{O(n)}$	$(2+1/\varepsilon)^{O(n)}$	Monte-Carlo solves $(1+\varepsilon)$-CVP only
Sec. 5, SVP	$n^{n/(2e)+o(n)}$	$\mathcal{P}oly(n)$	Deterministic
Sec. 5, CVP	$n^{n/2+o(n)}$	$\mathcal{P}oly(n)$	Deterministic

Singly exponential time complexity had already been achieved about 10 years before by Ajtai, Kumar and Sivakumar [5, 6], with an algorithm that consists in saturating the space with a cloud of (perturbed) lattice

points. But the saturation algorithms have at least three drawbacks: they are Monte Carlo (their success probability can be made exponentially close to 1, though), the CVP variants of these algorithms may only find vectors that are no more than $1 + \varepsilon$ times further away from the target than the optimal solution, for an arbitrary $\varepsilon > 0$ (the complexity grows when ε tends to 0), and their best known complexity upper bounds are higher than that of the Voronoi-based Micciancio-Voulgaris algorithm. The saturation-based Ajtai *et al.* SVP solver has been successively improved in [69, 65, 58, 68], and the currently best time complexity upper bound is $2^{2.465n+o(n)}$, with a space requirement bounded by $2^{1.325n+o(n)}$. Improvements on the Ajtai *et al.* CVP solver have been proposed by Blömer and Naewe [9]. We will describe the saturation-based SVP and CVP solvers in Section 4.

Before the algorithms of Ajtai *et al.*, the fastest SVP and CVP solvers relied on a deterministic procedure that enumerates all lattice vectors below a prescribed norm, or within a prescribed distance to a given target vector. This procedure uses the Gram-Schmidt orthogonalization of the input basis to recursively bound the integer coordinates of the candidate solutions. Enumeration-based SVP and CVP solvers were first described by Fincke and Pohst [20, 21] and Kannan [42, 43]. Kannan used it to propose solvers with bit-complexities $n^{O(n)}$. These were later refined by Helfrich [36], and their analyzes were improved in [32] who proved that the SVP (resp. CVP) solver has complexity $\leq n^{n/(2e)+o(n)}$ (resp. $\leq n^{n/2+o(n)}$). By refining a construction due to Ajtai [3], Hanrot and Stehlé [33] showed the existence of input bases for which Kannan's SVP algorithm performs $\geq n^{n/(2e)+o(n)}$ bit operations. We will describe these algorithms in Section 5.

Solving SVP and CVP in practice. The practicality of SVP solvers has attracted much more attention than their CVP counterparts, due to their importance in Schnorr's hierarchy [74] and their cryptanalytic applications. For currently handleable dimensions, the enumeration-based SVP solvers seem to outperform those of the other families. This statement requires clarification, as rigorous codes providing correctness guarantees can be accelerated significantly by allowing heuristics, which makes the comparison task more complex.

All the available implementations providing strong correctness guarantees (e.g., `fplll` [12] or the SVP solvers of Magma [10]) rely on the enumeration process. Several heuristics are known for the solvers of the saturation and enumeration families (at the moment, the Micciancio-Voulgaris Voronoi-based algorithm seems uncompetitive, and would require fur-

ther practical investigation). However, the heuristic implementations of the enumeration families, relying on tree pruning strategies [75, 76, 80, 25] seem to outperform the heuristic implementations of the saturation families [65, 58]. This observation has led to hardware implementations of the enumeration [37, 16].

It is hard to compare the soundness of the different heuristics used, but one way to compare the resulting codes is to see how they perform on actual inputs. However, checking the result is non-trivial, as there is no known way of rigorously doing so (apart from solving SVP once more). To circumvent this problem, the authors of the online SVP challenges [26] sampled SVP instances from some specific distribution [29] for which the expectancy of the minimum is known (via the Gaussian heuristic, see Section 5). If the vector found is no longer than slightly more than this expectancy, the challenge is considered solved. Another possibility, suggested in [24], consists in generating instances where the shortest vector is most likely the unique non-zero lattice vector whose norm is some known threshold (such as lattices corresponding to knapsacks [66]). Yet another strategy consists in looking at the reduction qualities achieved by using the SVP solvers within an implementation of Schnorr's hierarchy (typically BKZ). Comparisons are thus possible via lattice reduction challenges [49]. A drawback of this approach is that although the SVP solver is asymptotically the cost-dominating ingredient in reduction algorithms, there are other important factors, such as the number of times the SVP solver is called.

Related topics. Many problems on lattices are closely related to SVP and CVP. Sometimes but not always, the best algorithms for solving them are essentially the same as for solving SVP and CVP.

A variant of SVP consists in listing all shortest non-zero vectors, or counting them (their number is the so-called kissing number). The Theta series is a formal series whose coefficient of degree k is the number of vectors of squared norm equal to k (for all $k \geq 0$). The first coefficients of the Theta series of a lattice are typically computed with enumeration-based algorithms (see, e.g., [80]).

Given a lattice L, it may be desirable to compute the successive minima $\lambda_i(L) = \min\{r : \dim(\mathrm{span}(L \cap \mathcal{B}(r))) \geq i\}$, for all $i \leq n$, and linearly independent lattice vectors that reach them. Micciancio [53] showed how this problem can be reduced to CVP. Blömer and Naewe [9] have proposed a saturation-based algorithm that returns linearly independent vectors whose norms are not much greater than the minima.

The covering radius of a lattice is the maximal distance to that lattice of a vector of the spanned linear space. An algorithm relying on the Ajtai *et al.* CVP solver [31] provides a tight approximation to it. Also connected to CVP is the Bounded Distance Decoding problem (BDD): in BDD, the target vector is guaranteed to be within a distance $r \leq \lambda_1(L)/2$ of the lattice L. Several algorithms [45, 50] have been specifically designed for such CVP instances. BDD is of particular significance for MIMO communications [61] and cryptography [71]. We also refer to [51] for reductions of several lattice problems to and from BDD.

In this survey, we only consider the Euclidean norm. Recent works on solving SVP/CVP for the other ℓ_p norms include [9, 15, 18].

2 Some Background on Euclidean Lattices

A lattice is a discrete additive subgroup of some $\mathbb{R}^m$. Any such object can be represented as $\{B \cdot \boldsymbol{x}, \boldsymbol{x} \in \mathbb{Z}^n\}$, for some n and some full column rank matrix $B \in \mathbb{R}^{m \times n}$. For a given lattice L, some quantities, or invariants, do not depend on the particular choice of the basis. These include, among others: the embedding dimension m, the lattice rank n, the lattice determinant $\det L = \sqrt{\det(B^t \cdot B)}$, the lattice minimum $\lambda(L) = \min_{\boldsymbol{x} \in \mathbb{Z}^n \setminus \mathbf{0}} \|B \cdot \boldsymbol{x}\|$ and the covering radius $\mu(L) = \max_{\boldsymbol{t} \in \mathrm{span}(L)}(\mathrm{dist}(\boldsymbol{t}, L))$. We refer to [13, 30, 78, 52] for mathematical introductions to lattices. The most fundamental result on lattice invariants is Minkowski's theorem, which states that for all n, there exists a constant $\gamma_n \leq n$ such that for any rank n lattice L, we have $\lambda(L) \leq \sqrt{\gamma_n}(\det L)^{1/n}$.

When $L = \{B \cdot \boldsymbol{x}, \boldsymbol{x} \in \mathbb{R}^n\}$ with a full column rank matrix B, we say that the columns $(\boldsymbol{b}_i)_i$ of B form a basis of the lattice L. Any given lattice of rank ≥ 2 has infinitely many lattice bases: The columns of the full column rank matrices $B_1, B_2 \in \mathbb{R}^{m \times n}$ span the same lattice if and only if there exists a matrix $U \in \mathbb{Z}^{n \times n}$ of determinant ± 1 such that $B_2 = B_1 \cdot U$. Such a matrix U is called unimodular.

For computational statements, we will always consider rational lattices, given by rational bases (see [11] for the case of real-valued input bases). In the case of CVP, the target vector will also be rational. The complexities of the algorithms we will study are typically exponential with respect to n but only polynomial with respect to m and the bit-sizes of the entries of the input matrix. For the sake of simplicity, we will assume that the bit-size of the input is $\mathcal{P}oly(n)$.

The process of improving the quality of a basis by applying well-chosen unimodular matrices is generically called lattice reduction. The achieved

qualities, or reductions, are most often defined in terms of orthogonality. For a given basis $(\boldsymbol{b}_i)_{i\leq n}$, we define its Gram-Schmidt orthogonalization by $\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j<i} \mu_{i,j}\boldsymbol{b}_j^*$, where $\mu_{i,j} = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle / \|\boldsymbol{b}_j^*\|^2$. As the $\boldsymbol{b}_i^*$'s are orthogonal, we have $\lambda(L) \geq \min_{i\leq n} \|\boldsymbol{b}_i^*\|$, where L is the lattice spanned by the $\boldsymbol{b}_i$'s. We also have $\mu(L) \leq \sqrt{\sum_{i\leq n} \|\boldsymbol{b}_i^*\|^2}/2$ (see [7]).

A basis is said size-reduced if $|\mu_{i,j}| \leq 1/2$ for any $i > j$. A basis is said LLL-reduced if it is size-reduced and if $\|\boldsymbol{b}_i^*\| \geq \|\boldsymbol{b}_{i-1}^*\|/2$ for any $i < n$. A LLL-reduced basis of any rational lattice can be computed in polynomial time from any input basis [47]. However, its quality is only moderate: the quantity $\|\boldsymbol{b}_1\|/\lambda(L)$ may be bounded by 2^n. This upper bound seems reached in practice, up to a constant factor in the exponent [63].

On the other hand, the Hermite-Korkine-Zolotarev (HKZ) reduction is more expensive to compute (it is polynomial-time equivalent to solving SVP), but HKZ-reduced bases are much more orthogonal. A basis $B = (\boldsymbol{b}_i)_{i\leq n}$ is said HKZ-reduced if it is size-reduced, if $\|\boldsymbol{b}_1\| = \lambda(L(B))$ and if once projected orthogonally to $\boldsymbol{b}_1$ the vectors $\boldsymbol{b}_2, \ldots, \boldsymbol{b}_n$ are themselves HKZ-reduced. By applying Minkowski's theorem on the projected lattices, it is possible to show that any HKZ-reduced basis $(\boldsymbol{b}_i)_i$ satisfies $\|\boldsymbol{b}_1\|/\lambda(L) \leq \exp((\log^2 n)/4)$ (see [33]).

Schnorr's hierarchy of reductions and reduction algorithms [74] offers a compromise between LLL's efficiency and HKZ's quality. The principle is to consider small dimensional projected blocks: if $(\boldsymbol{b}_i)_{i\leq n}$ is the current lattice basis, the β-dimensional projected block starting at index k is $(\boldsymbol{b}_i^{(k)})_{k\leq i<k+\beta}$, where $\boldsymbol{b}_i^{(k)} = \boldsymbol{b}_i - \sum_{j<k} \mu_{i,k}\boldsymbol{b}_j^*$ is the projection of $\boldsymbol{b}_i$ orthogonally to the span of the first $k-1$ basis vectors. Within blocks, one performs HKZ-reductions (or calls to an SVP solver), and blocks are handled in a LLL-manner. This vague description has been instantiated in several precisely analyzed hierarchies of reduction algorithms [74, 22, 23] and is also the basis of the famous heuristic BKZ algorithm [75] implemented in NTL [77]. The best complexity/quality trade-off currently known [23] allows one to find a basis $(\boldsymbol{b}_i)_i$ such that $\|\boldsymbol{b}_1\|/\lambda(L) \leq (2\gamma_\beta)^{\frac{n-\beta}{\beta-1}}$ using calls to an SVP solver in dimension β. Note that the practical quality of BKZ-reduced bases has been investigated in [24], and that worst-case quality lower bounds for fixed block-sizes are known [3, 33]. Finally, in order to maximize $\|\boldsymbol{b}_n^*\|$ rather than minimize $\|\boldsymbol{b}_1\|$, one may reduce the dual basis B^{-t} (we refer to [69] for an introduction on the dual lattice), and apply the obtained unimodular transform to B. Thanks to Banaszczyk's transference theorem [8], the corresponding basis $(\boldsymbol{c}_i)_{i\leq n}$ of L satisfies $\mu(L)/\|\boldsymbol{c}_n^*\| \leq \gamma_n(2\gamma_\beta)^{\frac{n-\beta}{\beta-1}}$.

3 An SVP/CVP Solver Relying on the Voronoi Cell

The Micciancio-Voulgaris deterministic algorithm based on the Voronoi cell [57, 56] provides the SVP/CVP solver with the best known complexity upper bound: It terminates within $2^{2n+o(n)}$ operations while its space requirement is $\leq 2^{n+o(n)}$. From a practical perspective, this algorithm seems bound to remain slower than the guaranteed implementations of the enumeration-based solvers: On the one hand, its cost cannot be lower than 2^n, as this is the size of the representation of a generic Voronoi cell; On the other hand, although their time complexity bounds are asymptotically much higher than 2^n, the enumeration-based solvers still terminate relatively efficiently in dimensions n higher than 60. The asymptotic bounds suggest Voronoi-based solvers will eventually beat enumeration-based solvers, but the cut-off dimension seems to be out of reach with nowadays computational power. Also, for the moment, there is no known heuristic version of the Micciancio-Voulgaris algorithm which would allow to break the 2^n barrier.

The Voronoi cell $\mathcal{V}(L)$ of a lattice L is the set of vectors strictly closer to the origin than to any other lattice point:

$$\mathcal{V}(L) = \{\boldsymbol{x} : \forall \boldsymbol{b} \in L \setminus \mathbf{0}, \|\boldsymbol{b} - \boldsymbol{x}\| > \|\boldsymbol{x}\|\}\,. \tag{1}$$

We let $\overline{\mathcal{V}}(L)$ denote the topological closure of $\mathcal{V}(L)$, i.e., the set of points closer or at equal distance to the origin than to any other lattice point.

The definition (1) involves an infinite number of inequalities. In fact, there exists a minimal set $(\boldsymbol{v}_j)_{j\leq m}$ of vectors of L that suffices to define $\mathcal{V}(L)$: $\mathcal{V}(L) = \{\boldsymbol{x} : \forall j \leq m, \|\boldsymbol{v}_j - \boldsymbol{x}\| > \|\boldsymbol{x}\|\}$. Stated differently, the Voronoi cell is the interior of a polytope. We call these vectors the relevant vectors of L. Voronoi [81] displayed a strong link between the relevant vectors and the cosets of $L/2L$. A coset of $L/2L$ is of the shape $\{\sum_i (2x_i + e_i)\boldsymbol{b}_i, x_i \in \mathbb{Z}\}$, where $(\boldsymbol{b}_i)_i$ is a basis of L and the e_i's are fixed elements of $\{0, 1\}$. The vector $\boldsymbol{e}$ may be interpreted as the parity of the coset (note that it depends on the choice of the lattice basis $(\boldsymbol{b}_i)_i$).

Lemma 3.1 ([14, Th. 10, p. 477]). *A vector $\boldsymbol{v} \in L \setminus 2L$ is a relevant vector of the lattice L if $\pm\boldsymbol{v}$ are the unique minima (for the norm) of the coset $\boldsymbol{v} + 2L$. Consequently, there are $\leq 2(2^n - 1)$ relevant vectors.*

3.1 The Voronoi cell suffices for solving SVP and CVP

Assume we know the relevant vectors $(\boldsymbol{v}_i)_{i\leq m}$ of a lattice L. We now explain how to solve SVP and CVP by using this data.

To solve SVP in time $\mathcal{P}oly(n) \cdot 2^n$ from the $\boldsymbol{v}_i$'s, it suffices to observe that the shortest relevant vectors reach the lattice minimum.

Lemma 3.2. *If $\boldsymbol{s} \in L$ satisfies $\|\boldsymbol{s}\| = \lambda(L)$, then $\boldsymbol{s}$ is a relevant vector.*

Proof. The vector $\boldsymbol{s}$ cannot belong to $2L$ as otherwise $\boldsymbol{s}/2$ would be a shorter non-zero lattice vector. It remains to show that $\pm\boldsymbol{s}$ are the unique minima of the coset $\boldsymbol{s}+2L$. Assume that $\|\boldsymbol{s}+2\boldsymbol{b}\| = \|\boldsymbol{s}\|$ for some $\boldsymbol{b} \in L \setminus \boldsymbol{0}$. We have $\|\boldsymbol{s}\|^2 + 4\|\boldsymbol{b}\|^2 + 4\langle \boldsymbol{s}, \boldsymbol{b}\rangle = \|\boldsymbol{s}\|^2$, which leads to $\|\boldsymbol{b}\|^2 = -\langle \boldsymbol{s}, \boldsymbol{b}\rangle$. The Cauchy-Schwarz inequality provides $\|\boldsymbol{b}\| = \|\boldsymbol{s}\|$ if $\boldsymbol{b}$ and $\boldsymbol{s}$ are linearly dependent, and $\|\boldsymbol{b}\| < \|\boldsymbol{s}\|$ otherwise. In the first situation, the only way of ensuring that $\|\boldsymbol{s} + 2\boldsymbol{b}\| = \|\boldsymbol{s}\|$ is to take $\boldsymbol{b} = -\boldsymbol{s}$. In the other situation, we must have $\boldsymbol{b} = \boldsymbol{0}$, since $\boldsymbol{s}$ is a shortest non-zero vector of L. □

`Input` : The relevant vectors $(\boldsymbol{v}_i)_{i \leq N}$ and $\boldsymbol{t} \in 2\overline{\mathcal{V}}(L)$.
`Output` : A vector $\boldsymbol{t}' \in (\boldsymbol{t} + L) \cap \overline{\mathcal{V}}(L)$.
While $\boldsymbol{t} \notin \overline{\mathcal{V}}(L)$ do
    Find $i \leq N$ maximizing $\langle \boldsymbol{t}, \boldsymbol{v}_i\rangle / \|\boldsymbol{v}_i\|^2$, $\boldsymbol{t} \leftarrow \boldsymbol{t} - \boldsymbol{v}_i$.
Return $\boldsymbol{v}$.

Algorithm 1: The $\mathrm{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$ algorithm.

We now explain how to solve CVP, i.e., subtract from a given target vector a lattice vector so that the result belongs to the closed Voronoi cell. The engine of the algorithm is a sub-routine $\mathrm{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$ that solves CVP assuming that the target vector $\boldsymbol{t}$ belongs to $2 \cdot \overline{\mathcal{V}}(L)$. If such a routine is available, then CVP can be solved using a polynomial number of calls to it. First, note that any target vector $\boldsymbol{t}$ in the linear span of L belongs to $2^k \cdot \overline{\mathcal{V}}(L)$, with $k = \log_2(2\|\boldsymbol{t}\|/\lambda(L))$ (because $\mathcal{B}(\lambda(L)/2) \subseteq \overline{\mathcal{V}}(L)$). Now, the routine $\mathrm{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$ may be used to subtract vectors of L to any given target vector $\boldsymbol{t} \in 2^\ell \cdot \overline{\mathcal{V}}(L)$ so that the result belongs to $(\boldsymbol{t}+L) \cap 2^{\ell-1} \cdot \overline{\mathcal{V}}(L)$: It suffices to use $\mathrm{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$ with the lattice $2^{\ell-1}L$, whose closed Voronoi cell is $2^{\ell-1} \cdot \overline{\mathcal{V}}(L)$.

The sub-routine $\mathrm{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$, is an improvement of the Iterative Slicer from [79]. Given a target $\boldsymbol{t} \in 2 \cdot \overline{\mathcal{V}}$, it first determines if $\boldsymbol{t}$ belongs to $\overline{\mathcal{V}}(L)$, in which case it stops. Otherwise, it finds $x \in (1, 2]$ such that $\boldsymbol{t}$ is on the boundary of $x \cdot \overline{\mathcal{V}}$, as well as a facet of $x \cdot \overline{\mathcal{V}}$ containing $\boldsymbol{t}$. Geometrically, it means that for every facet of $\overline{\mathcal{V}}$ we construct a cone of apex $\boldsymbol{0}$ and base that facet. These cones cover the whole space (and their interiors are disjoint), and we determine a cone containing $\boldsymbol{t}$. In practice, this is obtained by finding a relevant vector $\boldsymbol{v}_i$ that maximizes $\langle \boldsymbol{t}, \boldsymbol{v}_i\rangle / \|\boldsymbol{v}_i\|^2$ (in which case $x = 2\langle \boldsymbol{t}, \boldsymbol{v}_i\rangle / \|\boldsymbol{v}_i\|^2$). Once $\boldsymbol{v}_i$ is found, the target $\boldsymbol{t}$ is updated to a new target vector $\boldsymbol{t} \leftarrow \boldsymbol{t} - \boldsymbol{v}_i$. This process is repeated until $\boldsymbol{t}$ eventually

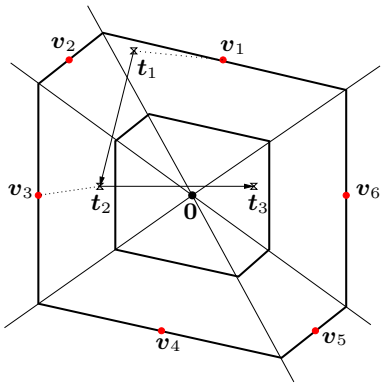

Fig. 1. Applying $\mathrm{CVP}_{2\overline{\mathcal{V}}\to\overline{\mathcal{V}}}$ to $\boldsymbol{t}_1$ with relevant vectors $(\boldsymbol{v}_i)_i$.

happens to belong to $\overline{\mathcal{V}}(L)$. Figure 1 provides an example of an execution of $\mathrm{CVP}_{2\overline{\mathcal{V}}\to\overline{\mathcal{V}}}$ in dimension 2. The following lemma states some important properties satisfied by the sequence of target vectors $(\boldsymbol{t}_k)_{k\geq 1}$ with $\boldsymbol{t}_1 = \boldsymbol{t}$, corresponding to the successive loop iterations.

Lemma 3.3. *Let $\boldsymbol{t}_1 \in 2\cdot\overline{\mathcal{V}}(L)$ and $(\boldsymbol{t}_k)_{k\geq 1}$ be the sequence obtained by applying the algorithm $\mathrm{CVP}_{2\overline{\mathcal{V}}\to\overline{\mathcal{V}}}$ to $\boldsymbol{t}_1$. Then:*

- *For any k, we have $\boldsymbol{t}_k \in 2\cdot\overline{\mathcal{V}}(L)$ and $\|\boldsymbol{t}_k\| > \|\boldsymbol{t}_{k+1}\|$,*
- *The execution terminates within $N \leq 2^n$ iterations and $\boldsymbol{t}_N \in \overline{\mathcal{V}}(L)$.*

Furthermore, if $\boldsymbol{t}_1 \in 2\cdot\mathcal{V}(L)$, then $\boldsymbol{t}_k \in 2\cdot\mathcal{V}(L)$ for all k.

Proof. The first claim derives from the observation that both $\boldsymbol{t}_k$ and $\boldsymbol{t}_k - x\boldsymbol{v}_{i(k)}$ belong to $x\cdot\overline{\mathcal{V}}(L) \subseteq 2\cdot\overline{\mathcal{V}}(L)$ (because $\frac{1}{x}\boldsymbol{t}_k$ belongs to both $\overline{\mathcal{V}}(L)$ and $\boldsymbol{v}_{i(k)} + \overline{\mathcal{V}}(L)$). By convexity, so does $\boldsymbol{t}_k - \boldsymbol{v}_{i(k)}$. For the second claim, note that by construction we have $\|\boldsymbol{t}_k - x\boldsymbol{v}_{i(k)}\| = \|\boldsymbol{t}_k\|$. This implies that $\|\boldsymbol{t}_k - \boldsymbol{v}_{i(k)}\|^2 = \|\boldsymbol{t}_k\|^2 - (x-1)\|\boldsymbol{v}_{i(k)}\|^2 < \|\boldsymbol{t}_k\|^2$. At this stage, we have proved that we have a sequence of vectors of $(\boldsymbol{t}_1 + L)\cap 2\cdot\overline{\mathcal{V}}(L)$ of strictly decreasing norms. Each such vector belongs to one of the 2^n cosets of $\boldsymbol{t}_1 + L$ modulo $2L$. We show by contradiction that such a coset cannot be visited twice. Let $k < \ell$ such that $\boldsymbol{t}_k = \boldsymbol{t}_\ell \bmod 2L$. Since both $\boldsymbol{t}_k$ and $\boldsymbol{t}_\ell$ belong to $\overline{\mathcal{V}}(2L)$, they must be shortest elements of their respective cosets $\boldsymbol{t}_k + 2L$ and $\boldsymbol{t}_\ell + 2L$. Since we assumed that these cosets are equal, the vectors $\boldsymbol{t}_k$ and $\boldsymbol{t}_\ell$ must have equal norms. This is in contradiction with $\|\boldsymbol{t}_k\| > \|\boldsymbol{t}_\ell\|$. □

This result allows us to bound the cost of sub-routine $\mathrm{CVP}_{2\overline{\mathcal{V}}\to\overline{\mathcal{V}}}$ by $\mathcal{P}oly(n)\cdot 2^{2n}$. Overall, once the relevant vectors of L are known, it is possible to solve CVP within $\mathcal{P}oly(n)\cdot 2^{2n}$ bit operations.

3.2 Computing the relevant vectors

In the SVP and CVP solvers of the previous subsection, the list of the relevant vectors of the lattice L under scope was assumed known. We now explain how to obtain such a list.

Let $(\boldsymbol{b}_i)_{i\leq n}$ be a basis of L that such that $\|\boldsymbol{b}_n^*\| \geq 2\mu(L)/\phi_n$, for some known $\phi_n = 2^{o(n)}$. This can be achieved by applying Schnorr's hierarchy with block-size $\omega(1)$ (see Section 2), on the dual basis of $(\boldsymbol{b}_i)_{i\leq n}$. The purpose of this pre-processing of the lattice basis will become clearer soon.

We assume we know the relevant vectors of $L^- := L[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}]$ (thanks to a recursive call in dimension $n-1$). A relevant vector of L is a strict length minimum for a coset of $L/2L$: it cannot be made smaller by subtracting from it an element of $2L$. In particular, it cannot be made smaller by subtracting from it a relevant vector of $2L^-$, i.e., twice a relevant vector of L^-. This implies that all the relevant vectors of L are contained in the interior of the cylinder of basis $2 \cdot \mathcal{V}(L^-)$ that is orthogonal to the span of L^-.

We now show that the relevant vectors of L cannot be arbitrarily far away from the span of L^-, thus allowing us to bound the search region. Let $\boldsymbol{t} = \sum_i e_i \boldsymbol{b}_i$ with $e_i \in \{0, 1\}$ for all i. We search for the relevant vector $\sum(e_i + 2x_i)\boldsymbol{b}_i \in L$ corresponding to the coset $\boldsymbol{t} + 2L$. This is a CVP instance for the target $\boldsymbol{t}$ and the lattice $2L$. We cannot use the CVP algorithm of the previous subsection directly, as it requires the knowledge of the relevant vectors of L, which we are currently trying to compute. Instead, we note that:

$$\mathrm{CVP}(\boldsymbol{t}, 2L) = \min_{\|\cdot\|} \left\{ \mathrm{CVP}(\boldsymbol{t} + 2x_n\boldsymbol{b}_n, 2L^-) : x_n \in \mathbb{Z} \right\}.$$

In fact, since $\boldsymbol{t}$ is within distance $2\mu(L)$ from $2L$ and $\|\boldsymbol{t} + 2x_n\boldsymbol{b}_n + 2\sum_{i<n} x_i\boldsymbol{b}_i\| \geq |e_n + 2x_n|\|\boldsymbol{b}_n^*\|$, we obtain that it suffices to consider the x_n's such that $|e_n + 2x_n| \leq 2\mu(L)/\|\boldsymbol{b}_n^*\|$. Thanks to the reducedness of the $\boldsymbol{b}_i$'s, it suffices to consider a bounded number of x_n's. This dimension reduction trick for CVP is inspired from the enumeration-based CVP solver (see Section 5). Overall, we have proved the following result.

Lemma 3.4. *Let $(\boldsymbol{b}_i)_i$ be a basis of a lattice L with $\|\boldsymbol{b}_n^*\| \geq 2\mu(L)/\phi_n$. Then the relevant vectors of L are contained in the set*

$$\bigcup_{|x_n|\leq\phi_n} \left[x_n\boldsymbol{b}_n^* + \left(x_n(\boldsymbol{b}_n - \boldsymbol{b}_n^*) + L^- \cap 2 \cdot \mathcal{V}(L^-) \right) \right],$$

where $L^- = L[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}]$.

This justifies Algorithm 2. Its cost is no more than $2\phi_n + 1 = 2^{o(n)}$ times the cost of enumerating $(\boldsymbol{t} + L^-) \cap 2 \cdot \mathcal{V}(L^-)$, for an arbitrary $\boldsymbol{t}$ in the span of the lattice L^-, whose relevant vectors are known.

```
Input  : A basis of a lattice L.
Output : The relevant vectors of L.
Find a basis (b_i)_{i≤n} of L such that ‖b_n^*‖ ≥ 2μ(L)/φ_n.
If n = 1, then Return ±b_1.
Compute the relevant vectors (v_i)_{i≤N} of L^-.
Cand ← ∅. For x_n = −φ_n to φ_n do
    (t_i)_i ← Enum_{2V}(x_n(b_n − b_n^*), L^-),
    Cand ← Cand ∪ {x_n b_n^* + t_i}_i.
Vor ← ∅. For any non-zero coset C of L/2L do
    If there are strict minima ±v in C ∩ Cand, then add them to Vor.
Return Vor.
```

Algorithm 2: Computing the relevant vectors.

We now explain how to draw the list of all the elements belonging to $(\boldsymbol{t} + L^-) \cap 2 \cdot \mathcal{V}(L^-)$. Algorithm $\texttt{Enum}_{2\mathcal{V}}$ first computes a shortest representative $\boldsymbol{s}$ of $\boldsymbol{t} + L^-$: If $\boldsymbol{t} = \boldsymbol{0}$, then we have $\boldsymbol{s} = \boldsymbol{0}$, and otherwise we may use the CVP algorithm from the last subsection (recall that we know the relevant vectors of L^-). We prove the correctness of $\texttt{Enum}_{2\mathcal{V}}$ only for the situation where $\boldsymbol{s}$ is the only shortest element in $\boldsymbol{t} + L^-$. Correctness also holds without this assumption and can be proved using results from [40] on the relevant vectors of a Voronoi cell [82]. A proof will appear in the final version of [57, 56].

The next step of $\texttt{Enum}_{2\mathcal{V}}$ consists in backtracking all the possible executions of $\text{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$ that start from an element of $(\boldsymbol{t} + L^-) \cap 2 \cdot \mathcal{V}(L^-)$: If $\boldsymbol{s}$ is the unique shortest element in $\boldsymbol{t} + L^-$, then any of these executions has to terminate with $\boldsymbol{s}$. Algorithm $\texttt{Enum}_{2\mathcal{V}}$ starts from a list of accessible vectors `Acc` consisting of $\boldsymbol{s}$ and an empty list of visited vectors `Vis`. At any iteration, it takes the shortest accessible vector $\boldsymbol{u}$, tags it as visited, and adds to the accessible list the $\boldsymbol{u} + \boldsymbol{v}_i$'s for all relevant vectors $\boldsymbol{v}_i$. Before proceeding to the next iteration, $\texttt{Enum}_{2\mathcal{V}}$ cleans the accessible list by deleting all vectors belonging to a coset of $\boldsymbol{t} + L^-$ mod $2L^-$ reached by the visited list, and keeping a shortest accessible vector for each remaining coset of $\boldsymbol{t} + L^-$ mod $2L^-$. The process stops when the accessible list is empty. The number of iterations is bounded by the number of cosets of $\boldsymbol{t} + L^-$ mod $2L^-$, i.e., termination occurs within 2^{n-1} iterations, allowing us to bound the cost by $\mathcal{P}oly(n) \cdot 2^{2n}$.

```
Input  : The relevant vectors (v_i)_{i≤N} of a lattice L⁻, and t.
Output : A superset of (t + L⁻) ∩ 2·V(L⁻).
Find a shortest element s of t + L⁻.
Acc ← {s}, Vis ← ∅. While Acc ≠ ∅ do
    Let u be a shortest element of Acc,
    Vis ← Vis ∪ {u}, Acc ← Acc ∪ {u + v_i}_i,
    For any coset C of t + L⁻ mod 2L⁻ do
        If C ∩ Vis ≠ ∅, then Acc ← Acc \ C,
        Else Acc ← (Acc \ C) ∪ min_{‖·‖}(Acc ∩ C).
Return Vis.
```

Algorithm 3: The $\texttt{Enum}_{2\mathcal{V}}$ algorithm.

Lemma 3.5. *Let L^- be a lattice and $\boldsymbol{t} \in \mathrm{span}(L^-)$. Suppose that there exists a unique shortest element $\boldsymbol{s}$ in $\boldsymbol{t} + L$. Then, at the end of the execution of $\texttt{Enum}_{2\mathcal{V}}$ for target $\boldsymbol{t}$ and lattice L^-, the set of visited vectors contains all elements of $\boldsymbol{t} + L^- \cap 2 \cdot \mathcal{V}(L^-)$.*

Proof. Suppose by contradiction that $\boldsymbol{u}_1 \in \boldsymbol{t} + L^- \cap 2 \cdot \mathcal{V}(L^-)$ does not belong to the final visited list. By Lemma 3.3, there exists a sequence $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N$ of vectors of $\boldsymbol{t} + L^- \cap 2 \cdot \mathcal{V}(L^-)$ with $\boldsymbol{u_N} = \boldsymbol{s}$ and, for all k, $\|\boldsymbol{u}_k\| > \|\boldsymbol{u}_{k+1}\|$ and any $\boldsymbol{u}_{k+1} - \boldsymbol{u}_k$ is a relevant vector of L^-. Since $\boldsymbol{s}$ belongs to the final visited list, there must exist a $k < N$ such that $\boldsymbol{u}_{k+1}$ is in the visited list, but not $\boldsymbol{u}_k$. When $\boldsymbol{u}_{k+1}$ was included in the visited list, the vector $\boldsymbol{u}_k$ was added in the accessible list, because $\boldsymbol{u}_{k+1} - \boldsymbol{u}_k$ is a relevant vector. Since $\boldsymbol{u}_k$ is the sole vector of its coset of $\boldsymbol{t} + L^- \bmod 2L^-$ that belongs to $\mathcal{V}(2L^-)$, it cannot have been discarded later. This contradicts the definition of k. □

To conclude, the cost of computing the relevant vectors for the n-dimensional lattice L is that of sufficiently reducing the input basis of L, computing the relevant vectors of L^- of dimension $n-1$ and then running $\texttt{Enum}_{2\mathcal{V}}$ no more than $2\phi_n + 1 = 2^{o(n)}$ times. This leads to the claimed $2^{2n+o(n)}$ complexity upper bound.

4 Saturating the Space

In this section, we describe the category of sieve algorithms for SVP and (approximate) CVP, which are Monte-Carlo probabilistic algorithms running in exponential time and space. Heuristic versions of these algorithms have been used to solve SVP up to dimension 63 [65, 58].

The mathematical property which all the sieve algorithms try to exploit is that there are boundably many points within a compact body

which are distant from one another. Thus, by saturating the space with (possibly perturbed) lattice points, one eventually finds close-by or identical vectors. In Euclidean norm, the proof of the saturation property leading the best known complexity bounds relies on the following result on sphere packings.

Theorem 4.1 ([41]). *Let $E \subseteq \mathbb{R}^n \backslash \{\mathbf{0}\}$. If there exists $\phi_0 > 0$ such that for any $\boldsymbol{u}, \boldsymbol{v} \in E$, the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$ is $\geq \phi_0$, then $|E| \leq 2^{cn+o(n)}$ with $c = -\frac{1}{2}\log_2\left[1 - \cos(\min(\phi_0, 62.99^\circ))\right] - 0.099$.*

4.1 The AKS Algorithm

The `AKS` algorithm was introduced by Ajtai, Kumar and Sivakumar in [5] as the first single-exponential time algorithm for SVP. However, no explicit time bound was given. In [69], Regev described a simpler version of this algorithm running in time $2^{16n+o(n)}$. The constant in the exponent was decreased from 16 to 5.9 by Nguyen and Vidick [65], 3.4 by Micciancio and Voulgaris [58] and 2.7 by Pujol and Stehlé [68].

`AKS` can be described as follows: Let $\gamma < 1$ be a constant. Let $\mathcal{S}$ be a set of N lattice vectors sampled in the ball of radius $R = 2^{O(n)} \cdot \lambda(L)$. If N is large enough, there exists a pair $(\boldsymbol{u}, \boldsymbol{v})$ of vectors such that $\|\boldsymbol{u} - \boldsymbol{v}\| \leq \gamma R$, so $\boldsymbol{u} - \boldsymbol{v}$ is a shorter vector of L. The main step of the algorithm, called sieve, consists in choosing $\mathcal{C} \subseteq \mathcal{S}$ such that $|\mathcal{C}|$ is not too large and for any $\boldsymbol{u} \in \mathcal{S} \setminus \mathcal{C}$, there exists $\boldsymbol{v} \in \mathcal{C}$ such that $\|\boldsymbol{u} - \boldsymbol{v}\| \leq \gamma R$ (see Figure 2). This is used to generate a set $\mathcal{S}' \subseteq L \cap \mathcal{B}(\gamma R)$ with $|\mathcal{S}'| = |\mathcal{S}| - |\mathcal{C}|$. The sieve can be applied a polynomial number of times to obtain lattice vectors whose norms are $\leq r_0 \lambda(L)$ for some constant r_0.

There is no known way of ensuring that the vectors in the final set are uniformly distributed, but a technical trick allows to ensure that any shortest non-zero vector of L can be written as the difference between two vectors in the final set. A perturbation randomly chosen in $\mathcal{B}(\xi\lambda(L))$ is added to each sampled lattice vector to obtain a perturbed vector (if $\lambda(L)$ is unknown, one may try polynomially many guesses). At any time, the algorithm keeps track of both lattice vectors and perturbed vectors, applying the same operations on them. Provided that $\xi > \frac{1}{2}$, a given perturbed vector might sometimes correspond to two different lattice vectors whose distance is $\lambda(L)$. The sieve function makes tests only on perturbed vectors so is unaware of the genuine lattice vectors. This can be used to prove that the probability of success is at most $2^{O(n)}$ times smaller than the probability of having the same lattice vector twice in the final set. The latter occurs when it contains $\geq |L \cap \mathcal{B}(r_0\lambda(L))|$ elements.

The algorithm is given below in the version of [65]. Perturbations are generated before lattice vectors and are used to compute them. In particular, if $\boldsymbol{x}$ and $\boldsymbol{y}$ are two perturbations such that $\boldsymbol{x} - \boldsymbol{y} \in L$, they will lead to the same lattice vector. Although this is might look as the most natural solution, this idea introduced by Regev [69] makes the proof simpler. In algorithm `NewPair`, the vector $(-\boldsymbol{x}) \bmod \mathcal{P}(B)$ is the vector whose coordinates with respect to B are the fractional parts of the coordinates of $-\boldsymbol{x}$ with respect to B.

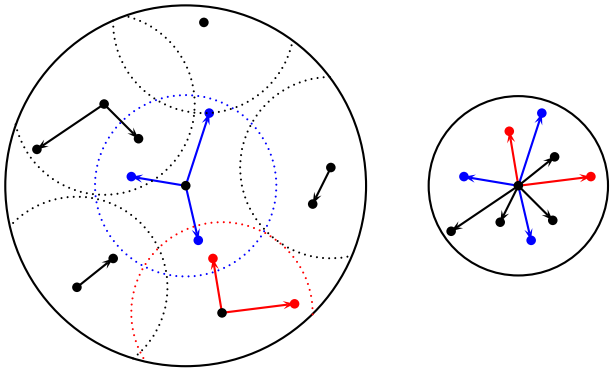

Fig. 2. The set $\mathcal{S}$ before and after one step of the sieve.

`Input` : A basis B and a perturbation $\boldsymbol{x}$.
`Output` : A lattice vector $\boldsymbol{u}$ and a perturbed vector $\boldsymbol{u}'$.
$\boldsymbol{u}' \leftarrow (-\boldsymbol{x}) \bmod \mathcal{P}(B)$, $\boldsymbol{u} \leftarrow \boldsymbol{u}' + \boldsymbol{x}$.
Return $(\boldsymbol{u}, \boldsymbol{u}')$.

Algorithm 4: The `NewPair` function.

`Input` : A basis B, $R > 0$, a set $\mathcal{T} \subseteq L \times \mathcal{B}(R)$, $\gamma > 0$.
`Output` : A list of pairs $\mathcal{T}'$.
$\mathcal{C} \leftarrow \emptyset$, $\mathcal{T}' \leftarrow \emptyset$.
For each pair $(\boldsymbol{t}, \boldsymbol{t}') \in \mathcal{T}$ do
    If $\exists (\boldsymbol{c}, \boldsymbol{c}') \in \mathcal{C}, \|\boldsymbol{t}' - \boldsymbol{c}'\| \leq \gamma R$, then add $(\boldsymbol{t} - \boldsymbol{c}, \boldsymbol{t}' - \boldsymbol{c})$ to $\mathcal{T}'$,
    Else add $(\boldsymbol{t}, \boldsymbol{t}')$ to $\mathcal{C}$.
Return $\mathcal{T}'$

Algorithm 5: The `Sieve` algorithm.

`Input` : A basis B, $\xi > \frac{1}{2}$, $\gamma > 0$, N, $\lambda \approx \lambda(L)$.
`Output` : A shortest non-zero vector of L.
$R \leftarrow n \cdot \max_i \|\boldsymbol{b}_i\| + \xi$, $\mathcal{T} \leftarrow \emptyset$.
For $i = 1$ to N do
    $\boldsymbol{x} \leftarrow$ random point uniformly chosen in $\mathcal{B}(\xi\lambda)$,
    Add $\texttt{NewPair}(B, \boldsymbol{x})$ to $\mathcal{T}$.
For $i = 1$ to $\left\lceil \log_\gamma \left(\frac{\xi\lambda}{nR(1-\gamma)} \right) \right\rceil$ do
    $\mathcal{T} \leftarrow \texttt{Sieve}(B, R, \mathcal{T})$, $R \leftarrow \gamma R + \xi\lambda$.
Remove pairs of $\mathcal{T}$ corresponding to the same lattice point.
Return the difference between two closest distinct lattice points in $\mathcal{T}$ (fail if they do not exist).

Algorithm 6: The `AKS` algorithm.

Theorem 4.2. *Let $(\boldsymbol{b}_i)_{i \leq n}$ be a basis of a lattice L such that $\max_i \|\boldsymbol{b}_i\| \leq 2^{3n}\lambda(L)$. Let $\xi = 0.676$, $\gamma = 0.496$, $N = 2^{1.984n}$ and $\lambda \in [\lambda(L), (1 + 1/n)\lambda(L))$. With probability exponentially close to 1, `AKS` returns a shortest vector of $L \setminus \mathbf{0}$. Moreover, it terminates in time $\leq \mathcal{P}oly(n)2^{3.397n}$ and space $\leq \mathcal{P}oly(n)2^{1.984n}$.*

It is possible to ensure the first assumption is fulfilled by LLL-reducing the basis B and removing vectors $\boldsymbol{b}_i$ such that $\|\boldsymbol{b}_i\| > 2^{2n}\|\boldsymbol{b}_1\|$. This does not change the shortest vectors (see [65, Le. 3.3]). Although $\lambda(L)$ is unknown, running the algorithm with $\lambda = \|\boldsymbol{b}_1\|(1 + 1/n)^{-i}$ for $i = 0, \ldots, \frac{n}{2\log_2\left(1+\frac{1}{n}\right)} = O(n^2)$ ensures that one value of λ will satisfy the condition on λ, by LLL-reducedness.

We describe the four main steps of the proof. More details are given in the appendix. For the whole proof, an arbitrary shortest vector $\boldsymbol{s}$ is fixed. Only the following subset of the sampled pairs is ever considered: the *good* pairs $(\boldsymbol{t}, \boldsymbol{t}')$ are those such that the perturbation $\boldsymbol{x} = \boldsymbol{t}' - \boldsymbol{t}$ belongs to $\mathcal{B}(\mathbf{0}, \xi\lambda) \cap \mathcal{B}(-\boldsymbol{s}, \xi\lambda)$. The first step consists in proving that a sampled pair is good with probability $\geq 2^{\frac{n}{2}\log_2\left(1-\frac{1}{4\xi^2}\right)+o(n)}$. This is done via elementary geometry arguments. The second step consists in bounding the number of vectors that are used as centers (and lost): There are $\leq 2^{(-\log_2\gamma+0.401)n+o(n)}$ centers. The third step consists in proving that, with high probability and at the end of the execution, the set $\mathcal{T}$ contains the same lattice vector twice. This is done by bounding the number of elements in $L \cap \mathcal{B}(r_0\lambda(L))$ which contains $\mathcal{T}$ by $2^{(\log_2 r_0+0.401)n+o(n))}$, where $r_0 \approx \xi(1 + \frac{1}{1-\gamma})$. The last two steps are based on the saturation phenomenon (Theorem 4.1). Finally, it can be shown that the probability of success is at least as large as the probability of collision (up to a constant factor). This implies that `AKS` succeeds with probability exponentially close to 1. Optimization on r_0 and ξ leads to the constants of Theorem 4.2.

Improving the complexity by using the birthday paradox. In the analysis of `AKS`, we use the fact that before removing identical lattice vectors in $\mathcal{T}$, the set contains two good pairs with the same lattice vector with high probability: This is because $\mathcal{T}$ contains $> N_B(n)$ lattice vectors corresponding to good pairs, and $\mathcal{T}$ is itself contained in a set of cardinality $\leq N_B(n)$. If the vectors were iid, no more than $O(\sqrt{N_B(n)})$ vectors would suffice. Although this may not hold for the version of `AKS` given above, the algorithm can be modified so that the iid-ness condition is fulfilled.

The change is as follows: At the start of the execution, we fix the set of vectors that will be used as centers at every step. Other vectors cannot be used as centers, so if at any sieving step, no close enough center is found, the vector under scope is lost. The probability of loss of each vector during the whole process can be made exponentially low by choosing the initial number of pairs carefully. These modifications can be implemented without incurring a significant cost increase (see the appendix for details). On the other hand, the number of vectors needed for the last step of the algorithm is decreased by an exponential factor. This leads to an algorithm that consumes time $\leq 2^{2.648n}$ and space $\leq 2^{1.325n}$.

Heuristic version of AKS. A heuristic version of `AKS` has been studied by Nguyen and Vidick [65]. The main modification consists in not using perturbations. To generate lattice vectors, random vectors are sampled using the randomized version of Babai's nearest plane algorithm [7] described in [45]. Under an assumption of uniform distribution of sieved points, they show that the space complexity of their algorithm is $\leq 2^{0.208n}$. In practice, the heuristic version solves SVP up to dimension 50. A refinement of this heuristic has been proposed by Wang *et al.* [83].

4.2 The ListSieve Algorithm

`ListSieve` was introduced by Micciancio and Voulgaris [58]. It can be described as a variant of `AKS` in which the order of the loop in the `Sieve` function and the second loop of `AKS` is reversed: in `ListSieve`, the outer loop is on vectors and the inner loop is on norm reduction steps. This makes the following improvement possible: for a given point, instead of using separate sets of centers for each step, all centers are put together in the same list. At a given step, any item in the list may be used provided that it decreases the norm of the vector by a factor $\geq 1/\gamma$. The loss of ξ between each step $R \leftarrow \gamma R + \xi$ of `AKS` does not occur in `ListSieve`,

making it possible to set γ polynomially close to 1 (e.g., $\gamma = 1 - 1/n$) to keep the list size as small as possible.

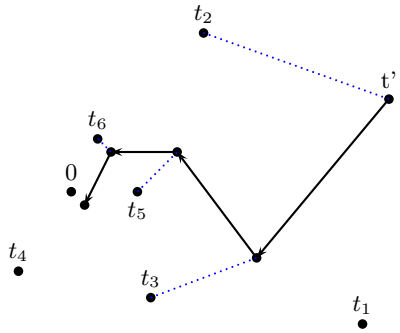

Fig. 3. Reduction of $\boldsymbol{t}'$ with the list $\{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_6\}$.

`Input` : A pair $(\boldsymbol{u}, \boldsymbol{u}')$ and a list $\mathcal{T} \subseteq L$.
`Output` : A reduced pair $(\boldsymbol{u}, \boldsymbol{u}')$.
While $\exists \boldsymbol{w} \in \mathcal{T}, \|\boldsymbol{u}' - \boldsymbol{w}\| < \gamma \|\boldsymbol{u}'\|$ do
    $(\boldsymbol{u}, \boldsymbol{u}') \leftarrow (\boldsymbol{u} - \boldsymbol{w}, \boldsymbol{u}' - \boldsymbol{w})$.
Return $(\boldsymbol{u}, \boldsymbol{u}')$.

Algorithm 7: The `Reduction` algorithm.

`Input` : A basis B, $\lambda \approx \lambda(L)$, $\xi > \frac{1}{2}$, N.
`Output` : A shortest non-zero vector of L.
Choose $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ independently and uniformly in $\mathcal{B}(\xi\lambda)$. $\mathcal{T} \leftarrow \{\boldsymbol{0}\}$.
For $i = 1$ to N do
    $(\boldsymbol{t}_i, \boldsymbol{t}_i') \leftarrow$ `Reduction`(`NewPair`$(B, \boldsymbol{x}_i), \mathcal{T})$,
    Add $\boldsymbol{t}_i$ to $\mathcal{T}$ unless it already belongs to it.
Find closest vectors $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ in $\mathcal{T}$ (fail if $|\mathcal{T}| = 1$).
Return $\boldsymbol{s}_1 - \boldsymbol{s}_2$.

Algorithm 8: The `ListSieve` algorithm.

`Input` : A basis B, $\lambda \approx \lambda(L)$, $\xi > \frac{1}{2}$, $r_0 > 2\xi$, N_1, N_2.
`Output` : A shortest non-zero vector of L.
Choose $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_1}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N_2})$ iid uniform in $\mathcal{B}(\xi\lambda)$. $\mathcal{T} \leftarrow \emptyset, \mathcal{U} \leftarrow \emptyset$.
For $i = 1$ to N_1 do
    $(\boldsymbol{t}_i, \boldsymbol{t}_i') \leftarrow$ `Reduction`(`NewPair`$(B, \boldsymbol{x}_i), \mathcal{T})$.
    If $\|\boldsymbol{t}_i\| \geq r_0\lambda$ then Add $\boldsymbol{t}_i$ to $\mathcal{T}$ unless it already belongs to it.
For $i = 1$ to N_2 do
    $(\boldsymbol{u}_i, \boldsymbol{u}_i') \leftarrow$ `Reduction`(`NewPair`$(B, \boldsymbol{y}_i), \mathcal{T})$,
    Add $\boldsymbol{t}_i$ to $\mathcal{U}$ unless it already belongs to it.
Find closest vectors $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ in $\mathcal{U}$ (fail if $|\mathcal{U}| = 1$).
Return $\boldsymbol{s}_1 - \boldsymbol{s}_2$.

Algorithm 9: The `ListSieveBirthday` algorithm.

Theorem 4.3. *Let $(\boldsymbol{b}_i)_{i\leq n}$ be a basis of a lattice L. If $\lambda \in [\lambda(L), (1 + 1/n)\lambda(L))$, $N = 2^{1.874n}$, $\xi = 0.685$ and n is sufficiently large, then* `ListSieve` *returns a shortest non-zero vector of L with probability $\geq \frac{1}{3}$, in time $\leq \mathcal{P}oly(n)2^{3.199n}$ and space $\leq \mathcal{P}oly(n)2^{1.325n}$.*

The proof of `ListSieve` is similar to that of `AKS`. The size of the list is bounded with Theorem 4.1, using two properties: any vector is reduced with respect to the previous vectors of the list and all vectors in the list belong to the lattice. Because of this, collisions occur when enough pairs are sampled. As in the proof of `AKS`, it can be proved that the probability of success is nearly as high as the probability that a collision occurs with a *good* vector. A detailed proof is given in appendix.

Improving the complexity by using the birthday paradox. In the original version of `ListSieve`, the vectors of the list may not be statistically independent. Dividing the algorithm into two phases makes it possible to sample iid vectors in a small ball $\mathcal{B}(r_0\lambda)$ (see Algorithm 9). In this version, the birthday paradox can be used to analyze the second phase, which leads to smaller time and space complexity bounds than for all versions of `AKS`.

Theorem 4.4. *Let $(\boldsymbol{b}_i)_{i\leq n}$ be a basis of a lattice L. If $\lambda \in [\lambda(L), (1 + 1/n)\lambda(L))$, $\xi = 0.9476$, $r_0 = 3.0169$, N_1 chosen uniformly in $[0, \lfloor 2^{1.3501n} \rfloor]$, $N_2 = 2^{1.2325n}$ and n is sufficiently large, then* `ListSieveBirthday` *returns a shortest non-zero vector of L with probability exponentially close to 1, in time $\leq \mathcal{P}oly(n)2^{2.465n}$ and space $\leq \mathcal{P}oly(n)2^{1.233n}$.*

During the first phase of the algorithm, very short vectors are thrown away. This allows to improves the bound on $|\mathcal{T}|$ from `ListSieve`. Indeed, because of perturbations, the lower bound for the angle between two vectors is worse for vectors of small norm. During the second phase, lattice vectors are reduced with respect to the list of vectors obtained during the first phase. As long as $|\mathcal{T}|$ is large enough, this should produce short lattice vectors (e.g., vectors in $\mathcal{B}(r_0\lambda)$). Unfortunately, it is unclear whether the probability for a vector to be short decreases when $|\mathcal{T}|$ increases. This is why $|\mathcal{T}|$ is randomized. The proof that the difference between two vectors in $\mathcal{U}$ is a shortest non-zero lattice vector is the same as for `AKS`. More details are given in appendix.

Heuristic version of ListSieve. In [58], Micciancio and Voulgaris give experimental results on a heuristic version of `ListSieve` that they call

GaussSieve. In GaussSieve, vectors are more reduced with respect to each other. As in the heuristic version of AKS, there are no perturbations: It is an open problem whether there is a way to guarantee that the algorithm succeeds. Thus, the stopping condition is chosen heuristically. However, a space complexity upper bound of $2^{0.401n}$ is proved (this corresponds to the best known upper bound on the so-called kissing number). According to the experiments from [58], GaussSieve can be used up to dimension 63 and outperforms the (guaranteed) deterministic enumeration algorithm from Section 5 without tree pruning.

4.3 Solving CVP

In 2002, Ajtai *et al.* [6] showed how to modify their SVP solver to solve CVP within any fixed factor $1+\varepsilon$ for $\varepsilon > 0$, in time and space $2^{O(n)}$. In [9], Blömer and Naewe showed that the sieve algorithms can be used to solve $(1+\varepsilon)$-CVP by reducing the latter to the "Generalized SVP" $(1+\varepsilon)$-GSVP, which, given a basis of a lattice L and a linear subspace M of $\mathbb{R}^m$, consists in finding a vector in $L \setminus M$ of norm $\leq (1+\varepsilon)\min_{\|\cdot\|}(L \setminus M)$. They proved that $(1+\varepsilon)$-GSVP can be solved by AKS just by changing the parameters and the last step, in which a condition is added: The difference between the two vectors must not be in M. As the shortest vector outside M can be much larger than $\lambda(L)$, it may not be possible to "fill" the ball $\mathcal{B}(r_0\lambda)$ corresponding to the end of the execution of AKS. The analysis of AKS does not carry over, but it is proved instead that AKS returns a vector outside of M of norm at most $r_0\lambda$. The approximation factor $1+\varepsilon$ is obtained by choosing γ and ξ so that $r_0 \leq 1+\varepsilon$. This is possible for any $\varepsilon > 0$, but at the expense of increasing the complexity bound (the exponent is polynomial in $\frac{1}{\varepsilon}$). The reduction from $(1+\varepsilon)$-CVP to $(1+\varepsilon)$-GSVP is based on Kannan's embedding technique [43, Se. 6]: For a lattice $L \subseteq \mathbb{R}^m$ and a target $\boldsymbol{t} \in \mathbb{R}^m$, we define L' as the lattice spanned by $(L \times \{0\}) \cup (\boldsymbol{t}, \gamma)$; provided that γ is large enough, if $(\boldsymbol{u}, x) \in \mathbb{R}^m \times \mathbb{R}$ is a shortest vector of $L' \setminus (\mathbb{R}^n \times \{0\})$, then $x = \pm\gamma$ and the solution to $\mathrm{CVP}(L, \boldsymbol{t})$ is $\pm\boldsymbol{u}$. More recently, Eisenbrand *et al.* [18] built upon [9] to decrease the cost dependence in ε.

5 Enumeration-Based Solvers

The oldest SVP/CVP solvers rely on an enumeration process, that, given a basis $(\boldsymbol{b}_i)_{i \leq n}$ of lattice L, a center $\boldsymbol{t}$ and a radius A, looks for all points of $L \cap \mathcal{B}(\boldsymbol{t}, A)$. It does so by enumerating all points of projections of L

orthogonally to basis vectors, that belong to hyperballs of smaller dimensions. In practice, a heuristic version of the enumeration based on pruning has been used to solve SVP for a generic-looking lattice of dimension 110, within a few days of computation [25, 26].

The main ingredient for the complexity analyzes of enumeration-based solvers consists in bounding and/or estimating the number of lattice points within a ball. Estimates are provided by the so-called *Gaussian heuristic*: if K is a measurable subset of the span of the n-dimensional lattice L, then $|K \cap L| \approx \mathrm{vol}(K)/\det(L)$ (where vol denotes the n-dimensional volume). For some choices of K compact and sufficiently large, or K or L sampled randomly, then rigorous versions of the Gaussian heuristic can be obtained. We will use the Gaussian heuristic mainly for balls, in which case we have $\mathrm{vol}\,\mathcal{B}_n(\boldsymbol{t}, A) = \frac{A^n \pi^{n/2}}{\Gamma(n/2+1)} \approx \frac{2^{O(n)} A^n}{n^{n/2}}$, for any $\boldsymbol{t}$ and A.

5.1 The `Enum` algorithm

`Enum` (Algorithm 10) enumerates $L \cap \mathcal{B}(\boldsymbol{t}, A)$ by using the triangular relationship between the basis $(\boldsymbol{b}_i)_{i \le n}$ of L and its Gram-Schmidt orthogonalization $(\boldsymbol{b}_i^*)_{i \le n}$. More specifically, it relies on the two following observations:

- If $\boldsymbol{x} = \sum_i x_i \boldsymbol{b}_i$ belongs to $L \cap \mathcal{B}(\boldsymbol{t}, A)$, then, for any $i \le n$, we have $\boldsymbol{x}^{(i)} \in L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, A)$, where $\boldsymbol{x}^{(i)}$, $L^{(i)}$ and $\boldsymbol{t}^{(i)}$ are the projections of $\boldsymbol{x}$, L and $\boldsymbol{t}$ orthogonally to the linear span of $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}$.
- Enumerating $L^{(n)} \cap \mathcal{B}(\boldsymbol{t}^{(n)}, A)$ is easy and once $L^{(i+1)} \cap \mathcal{B}(\boldsymbol{t}^{(i+1)}, A)$ is known, it is easy to enumerate $L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, A)$. Indeed: Assume that $\boldsymbol{x}^{(i)} \in L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, A)$; Write $\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i+1)} + (x_i + c_i)\boldsymbol{b}_i^*$ for some $x_i \in \mathbb{Z}$ and $c_i \in \mathbb{R}$. Once $\boldsymbol{x}^{(i+1)} \in L^{(i+1)} \cap \mathcal{B}(\boldsymbol{t}^{(i+1)}, A)$ is fixed, we must have

$$x_i \in \mathbb{Z} \cap \left[-c_i - \frac{\sqrt{A^2 - \|\boldsymbol{x}^{(i+1)}\|^2}}{\|\boldsymbol{b}_i^*\|}, -c_i + \frac{\sqrt{A^2 - \|\boldsymbol{x}^{(i+1)}\|^2}}{\|\boldsymbol{b}_i^*\|} \right] \quad (2)$$

During its execution, `Enum` considers all points in $L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, A)$, for $i = n, n-1, \ldots, 1$. An inherent drawback is that the complexity may be (significantly) more than $|L \cap \mathcal{B}(\boldsymbol{t}, A)|$. This is because it often happens that at some stage, an element of $L^{(i+1)} \cap \mathcal{B}(\boldsymbol{t}^{(i+1)}, A)$ has no descendant in $L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, A)$ (i.e., the interval in (2) contains no integer) : This corresponds to a "dead-end" in the enumeration tree.

`Input` : A basis $(\mathbf{b}_i)_{i\leq n}$ of a lattice L, $\boldsymbol{t} \in \mathrm{span}(L)$, $A > 0$.
`Output` : All vectors in $L \cap \mathcal{B}(\boldsymbol{t}, A)$.
Compute the $\mu_{i,j}$'s and $\|\boldsymbol{b}_i^*\|^2$'s.
Compute the t_i's such that $\mathbf{t} = \sum_i t_i \mathbf{b}_i^*$.
$S \leftarrow \emptyset, \boldsymbol{\ell} \leftarrow \mathbf{0}, \boldsymbol{x} \leftarrow \mathbf{0}$, $x_n \leftarrow \lceil t_n - A/\|\mathbf{b}_n^*\| \rceil$, $i \leftarrow n$.
While $i \leq n$ do
  $\ell_i \leftarrow (x_i - t_i + \sum_{j>i} x_j \mu_{j,i})^2 \|\mathbf{b}_i^*\|^2$,
  If $i = 1$ and $\sum_{1\leq j\leq n} \ell_j \leq A^2$ then
    $S \leftarrow S \cup \{\mathbf{x}\}$, $x_1 \leftarrow x_1 + 1$.
  If $i \neq 1$ and $\sum_{j\geq i} \ell_j \leq A^2$ then
    $i \leftarrow i - 1$, $x_i \leftarrow \left\lceil t_i - \sum_{j>i}(x_j \mu_{j,i}) - \sqrt{\frac{A^2 - \sum_{j>i} \ell_j}{\|\mathbf{b}_i^*\|^2}} \right\rceil$.
  If $\sum_{j\geq i} \ell_j > A$, then $i \leftarrow i + 1$, $x_i \leftarrow x_i + 1$.
Return S.

Algorithm 10: The `Enum` algorithm.

The cost of `Enum` can be bounded by $\sum_i |L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, A)|$, up to a small polynomial factor. The Gaussian heuristic leads to the approximation (for $i \leq n$):

$$|L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, A)| \approx \frac{2^{O(n)} A^{n-i+1}}{(n-i+1)^{\frac{n-i+1}{2}} \cdot \prod_{j=i}^{n} \|\boldsymbol{b}_j^*\|}.$$

Unfortunately, some of the involved balls are very small compared to their corresponding lattice $L^{(i)}$, and it seems hard to prove that the heuristic always holds. Though of mostly theoretical nature (because of the fuzzy $2^{O(n)}$ factor), the following result provides theoretical evidence towards the validity of the Gaussian heuristic in the present situation.

Theorem 5.1 ([32]). *The bit-complexity of* `Enum` *can be bounded from above by:*

$$2^{O(n)} \prod_{1\leq i\leq n} \max\left(1, \frac{A}{\sqrt{n}\|\boldsymbol{b}_i^*\|}\right) \leq 2^{O(n)} \max_{I\subseteq[1,n]} \frac{A^{|I|}}{\sqrt{n}^{|I|} \cdot \prod_{i\in I} \|\boldsymbol{b}_i^*\|}.$$

`Enum` may be used directly to solve SVP and CVP, once the bound A has been set. In the case of SVP, it may be derived from Minkowski's theorem, or from the current basis $(\mathbf{b}_i)_{i\leq n}$: For example, one may choose $A = \min(\min_i \|\mathbf{b}_i\|, \sqrt{n}(\det L)^{1/n})$. In the case of CVP, it may be derived from any bound on $\mu(L)$, such as $\sqrt{\sum_i \|\boldsymbol{b}_i^*\|^2}/2$. The bound may also be set heuristically using the Gaussian heuristic: The guess for A is then derived from $\mathrm{vol}\,\mathcal{B}(\boldsymbol{t}, A) \approx \det L$, and is increased if no solution is found. The bound A can also be decreased during the execution of `Enum`, every time a better solution is found, as the complexity increases sharply with A.

As written, the space required by `Enum` may be more than $\mathcal{P}oly(n)$. It can be made $\mathcal{P}oly(n)$ for the SVP and CVP applications, as only a single shortest/closest vector is required: the update of S in `Enum` should then be replaced by an update of the best solution found so far.

5.2 Reducing before enumerating

The cost estimates and upper bounds of `Enum` strongly depend on A and on the decrease of the $\|\boldsymbol{b}_i^*\|$'s. This observation suggests that the more reduced the basis $(\boldsymbol{b}_i)_i$, the lower the cost. Fincke and Pohst [20] initially used a LLL-reduced basis $(\boldsymbol{b}_i)_i$. For such a basis, we have $\|\boldsymbol{b}_{i+1}^*\| \geq \|\boldsymbol{b}_i^*\|/2$, which leads to a $2^{O(n^2)}$ complexity upper bound. Kannan [42] observed that the cost of `Enum` is so high that a much more aggressive pre-processing significantly lowers the total cost while negligibly contributing to it. In practice, one may use BKZ: This still leads to a $2^{O(n^2)}$ complexity bound for `Enum` when the block-size is fixed, but the $O(\cdot)$ constant decreases as the inverse of the block-size (up to factors of lower order). Kannan's theoretical algorithms are even more aggressive than that. His SVP algorithm is in fact an HKZ-reduction algorithm that calls itself recursively in lower dimensions to optimize the reduction before the calls to `Enum`.

The HKZ-reduction algorithm proceeds as follows (we refer to [36] for more details). Before calling `Enum` to find a shortest non-zero lattice vector, it quasi-HKZ-reduces the $\boldsymbol{b}_i$'s: at the moment `Enum` is called, the basis is such that once projected orthogonally to $\boldsymbol{b}_1$ it is HKZ-reduced, and $\|\boldsymbol{b}_2^*\| \geq \|\boldsymbol{b}_1\|/2$. To find such a basis, it suffices to:

- HKZ-reduce (in dimension $n-1$) the projections of the $\boldsymbol{b}_i$'s orthogonally to $\boldsymbol{b}_1$;
- extend them to a basis $(\boldsymbol{b}_i)_i$ of L, while maintaining that property and keeping the previous $\boldsymbol{b}_1$;
- HKZ-reduce $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ (and update the basis of L accordingly);
- and iterate the previous steps until quasi-HKZ-reduction is reached.

Then a shortest vector $\boldsymbol{b}$ of $L \setminus \mathbf{0}$ can be found with `Enum`, and extended into a basis of L (keeping $\boldsymbol{b}$ in first position). The HKZ-reduction algorithm is then called recursively on the projection of the basis orthogonally to $\boldsymbol{b}$, to get a reduced basis of L.

A detailed analysis of quasi-HKZ-reduced bases $(\boldsymbol{b}_i)_{i \leq n}$ gives that (see [32]):

$$\max_{I \subseteq [1,n]} \frac{\|\boldsymbol{b}_1\|^{|I|}}{\sqrt{n}^{|I|} \cdot \prod_{i \in I} \|\boldsymbol{b}_i^*\|} \leq 2^{O(n)} n^{n/(2e)}.$$

The call to `Enum` dominates the overall cost of the HKZ-reduction algorithm, so that Kannan's SVP solver terminates within $n^{n/(2e)+o(n)}$ bit operations.

Kannan's CVP algorithm is `Enum`, starting with an HKZ-reduced basis $(\boldsymbol{b}_i)_{i\leq n}$. Wlog, one may assume that $\|\boldsymbol{b}_1\| = \max_i \|\boldsymbol{b}_i\|$ (see [42]), and take $A = \sqrt{n}\|\boldsymbol{b}_1\|$. In that situation, we have:

$$\max_{I\subseteq[1,n]} \frac{(\sqrt{n}\|\boldsymbol{b}_1\|)^{|I|}}{\sqrt{n}^{|I|}\prod_{i\in I}\|\boldsymbol{b}_i^*\|} = \frac{\|\boldsymbol{b}_1\|^n}{\det L} \leq n^{n/2},$$

by Minkowski's theorem. This leads to an overall $n^{n/2+o(n)}$ complexity upper bound for Kannan's CVP solver.

5.3 Practical improvements

As `Enum` is the most practical SVP/CVP solver, much effort has been spent on optimizing it. Schnorr and Euchner [75] introduced a zig-zag strategy for enumerating the interval of Equation (2): starting from the middle of the interval increases the likeliness of finding short/close vectors faster, and to decrease A quickly.

Recently, Gama *et al.* [25, App. D] described a way of computing the term $\sum_{j>i} x_j\mu_{j,i}$ from `Enum` more efficiently by storing partial sums that can be re-used later in the execution, and thus removing redundant operations. This can save a significant amount of computation, at the expense of a small increase in space requirement.

In `Enum`, the quantities $(\mu_{i,j}, \|\boldsymbol{b}_i^*\|^2, t_i, A)$ are rational numbers, so that all computations can be performed exactly. However, the bitsizes of these numbers can be as large as $O(n\log B)$ (where B is the largest bitsize of an entry in the input basis matrix), which leads to a large arithmetic cost. In practice, approximate arithmetics such as floating-point or fixed-point are used instead. Analyzing the effect of these approximate computations requires to undertake a detailed analysis of the accumulation and amplification of the errors which occur at every step, as they could lead to incorrect results. These issues have been studied in detail in [67, 16]. It is possible to derive a sufficient precision such that when starting with an arbitrary LLL-reduced basis, the output is guaranteed to be a close to optimal solution. However, a dynamic analysis of the error should be preferred, because the bounds on diverse quantities used in the worst-case analysis are very pessimistic.

The parallelization of `Enum` has been investigated in [37] and [16]. In the latter, the authors use the Gaussian heuristic to balance the sizes of the sub-tasks sent to the slave machines.

The main heuristic used for accelerating `Enum` consists in pruning the corresponding enumeration tree, by cutting off branches with low ratio between the estimated branch size and the likeliness of containing the desired solution [75, 76, 80, 25]. Pruning consists in replacing the sets $L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, A)$ by subsets $L^{(i)} \cap \mathcal{B}(\boldsymbol{t}^{(i)}, p_i \cdot A)$, for some pruning coefficients $0 < p_n \leq \ldots \leq p_1 \leq 1$. The approximate branch size and success likeliness can be derived from the Gaussian heuristic, by estimating the volumes of the following truncated hyperballs (for all $i \leq n$):

$$\left\{(y_j)_{i \leq j \leq n} \in \mathbb{R}^{n-i+1} : \forall k \geq j, \left\|(y_j)_{k \leq j \leq n} - \mathbf{t}^{(k)}\right\| \leq p_k \cdot A\right\}.$$

Gama *et al.* recently proposed in [25] a further refinement, called extreme pruning. It consists in calling an enumeration with aggressive pruning coefficients several times on randomized inputs: this allows the success probability of a single call to be lowered, and to decrease the overall cost even more.

6 Open Problems

Many important techniques and results on solving SVP and CVP have been discovered in the last few years. The recent increase of interest in this topic stems at least in large part from the rise of lattice-based cryptography. We draw here a list of open problems that we find of particular importance.

The description and analysis of the Voronoi-based solver from Section 3 is extremely recent [57]. So far, no practical implementation nor heuristic improvement that could lead to a competitive implementation is known. Is the worst-case complexity bound sharp, and if so, is it likely to be reached for "average" inputs? Are there reasonable heuristics that could be exploited to speed up this algorithm?

The solvers relying on saturation and enumeration have undergone significantly more scrutiny, but some important questions remain open. For example, it is unclear whether the perturbations of the lattice vectors in saturation-based solvers are inherently necessary or just an artifact of the proof. As these perturbations lead to increased complexity bounds, proving them unnecessary could make these solvers competitive with [57]. Also, is it a valid heuristic to remove them in practice?

The theoretical and practical efficiency of enumeration-based solvers greatly depends on the pre-processing of the input basis. It is suggested in [62] that enumeration-based SVP solvers cost at least $n^{n/8+o(n)}$ in the

worst case: Is it possible to reach this lower bound? In practice, what is the best trade-off between pre-processing and enumerating? With respect to enumeration-based CVP solvers, an intriguing question is whether the best known complexity upper bound $n^{n/2+o(n)}$ can be reached in the worst case: contrarily to SVP [33], there is a gap between the best known worst-case complexity upper and lower bounds.

Overall, we have three very different families of solvers for SVP/CVP, which seem largely unrelated (except the dimension reduction step in the algorithm based on the Voronoi cell, which is inspired from `Enum`). Furthermore, their complexity upper bounds are incomparable: the solvers relying on the Voronoi cell and on saturation have time and space complexity upper bounds that are simply exponential, whereas Kannan's SVP solver has a worse time complexity bound but requires polynomially bounded space. Are there hidden links between these algorithms? Can they be combined to achieve interesting time/space trade-offs? Enumeration-based heuristic solvers are currently more practical for handleable dimensions, but the other solvers enjoy lower asymptotic complexity bounds: is it possible to estimate the cut-off dimension after which they become the most efficient?

It is also conceivable that faster solvers exist, that remain to be discovered. For example, is it possible to achieve exponential time complexity with a polynomially bounded space requirement? Are there ways to exploit quantum computations to obtain better complexity bounds? Finally, lattice-based cryptography does not rely on the hardness of SVP but of relaxed variants of it. In particular, if there were a polynomial-time algorithm that could find non-zero lattice vectors that are no more than polynomially longer (in the dimension) than the lattice minimum, then lattice-based cryptography would become insecure. Does such an SVP solver exist, or is it possible to argue that it does not exist?

Acknowledgments We thank Panagiotis Voulgaris for very helpful discussions on the Voronoi-based SVP/CVP solver. We are also very grateful to Thijs Laarhoven who pointed out several errors concerning the AKS algorithm and its birthday paradox variant (Section 4.1 and Appendix A). We also thank the anonymous reviewer for her/his comments. The third author was partly supported by the Australian Research Council under ARC Discovery Grant DP110100628.

References

1. E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, 2002.

2. M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proc. of STOC*, pages 99–108. ACM, 1996.
3. M. Ajtai. The worst-case behavior of Schnorr's algorithm approximating the shortest nonzero vector in a lattice. In *Proc. of STOC*, pages 396–406. ACM, 2003.
4. M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proc. of STOC*, pages 284–293. ACM, 1997.
5. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proc. of STOC*, pages 601–610. ACM, 2001.
6. M. Ajtai, R. Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *Proc. of CCC*, pages 53–57, 2002.
7. L. Babai. On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.
8. W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Math. Ann*, 296:625–635, 1993.
9. J. Blömer and S. Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theor. Comput. Science*, 410(18):1648–1665, 2009.
10. W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3–4):235–265, 1997. `http://magma.maths.usyd.edu.au/magma/`.
11. J. Buchmann. Reducing Lattice Bases by Means of Approximations. In *Proc. of ANTS*, volume 877 of *LNCS*, pages 160–168. Springer, 1994.
12. D. Cadé, X. Pujol, and D. Stehlé. fplll-3.1, a floating-point LLL implementation. `http://perso.ens-lyon.fr/damien.stehle`.
13. J. W. S. Cassels. *An Introduction to the Geometry of Numbers, 2nd edition*. Springer, 1971.
14. J.H. Conway and N.J.A. Sloane. *Sphere Packings, Lattices and Groups*. Springer, 1998. Third edition.
15. D. Dadush, C. Peikert, and S. Vempala. Enumerative algorithms for the shortest and closest lattice vector problems in any norm via M-ellipsoid coverings, 2011. submitted.
16. J. Detrey, G. Hanrot, X. Pujol, and D. Stehlé. Accelerating lattice reduction with FPGAs. In *Proc. of LATINCRYPT*, volume 6212 of *LNCS*, pages 124–143. Springer, 2010.
17. F. Eisenbrand. *50 Years of Integer Programming 1958-2008, From the Early Years to the State-of-the-Art*, chapter Integer Programming and Algorithmic Geometry of Numbers. Springer, 2009.
18. F. Eisenbrand, N. Hähnle, and M. Niemeier. Covering cubes and the closest vector problem, 2011. To appear in the proceedings of SoCG.
19. P. van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical report 81-04, Mathematisch Instituut, Universiteit van Amsterdam, 1981.
20. U. Fincke and M. Pohst. A procedure for determining algebraic integers of given norm. In *Proc. of EUROCAL*, volume 162 of *LNCS*, pages 194–202, 1983.
21. U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Math. Comp.*, 44(170):463–471, 1985.
22. N. Gama, N. Howgrave-Graham, H. Koy, and P. Q. Nguyen. Rankin's constant and blockwise lattice reduction. In *Proc. of CRYPTO*, number 4117 in LNCS, pages 112–130. Springer, 2006.
23. N. Gama and P. Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *Proc. of STOC*, pages 207–216. ACM, 2008.

24. N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *Proceedings of Eurocrypt 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, 2008.
25. N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *Proc. of Eurocrypt*, volume 6110 of *LNCS*, pages 257–278. Springer, 2010.
26. N. Gama and M. Schneider. The SVP challenge homepage. `http://www.latticechallenge.org/svp-challenge/`.
27. O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *Proc. of CRYPTO*, volume 1294 of *LNCS*, pages 112–131. Springer, 1997.
28. O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Inf. Process. Lett.*, 71(2):55–61, 1999.
29. D. Goldstein and A. Mayer. On the equidistribution of Hecke points. *Forum Mathematicum*, 15:165–189, 2003.
30. M. Gruber and C. G. Lekkerkerker. *Geometry of Numbers*. North-Holland, 1987.
31. V. Guruswami, D. Micciancio, and O. Regev. The complexity of the covering radius problem. *Computational Complexity*, 14(2):90–121, 2005.
32. G. Hanrot and D. Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm (extended abstract). In *Proc. of CRYPTO*, volume 4622 of *LNCS*, pages 170–186. Springer, 2007.
33. G. Hanrot and D. Stehlé. Worst-case Hermite-Korkine-Zolotarev reduced lattice bases. *CoRR*, abs/0801.3331, 2008.
34. A. Hassibi and S. Boyd. Integer parameter estimation in linear models with applications to GPS. *IEEE Transactions on signal process.*, 46(11):2938–2952, 1998.
35. I. Haviv and O. Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proc. of STOC*, pages 469–477. ACM, 2007.
36. B. Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theor. Comput. Science*, 41:125–139, 1985.
37. J. Hermans, M. Schneider, J. Buchmann, F. Vercauteren, and B. Preneel. Parallel shortest lattice vector enumeration on graphics cards. In *Proc. of Africacrypt*, volume 6055 of *LNCS*, pages 52–68. Springer, 2010.
38. C. Hermite. *Œuvres*. Gauthiers-Villars, 1905.
39. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: a ring based public key cryptosystem. In *Proc. of ANTS*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998.
40. Á. G. Horváth. On the Dirichlet-Voronoi cells of the unimodular lattices. *Geometricæ Dedicata*, 63:183–191, 1996.
41. G. A. Kabatyansky and V. I. Levenshtein. Bounds for packings on a sphere and in space. *Probl. Peredachi Inf.*, 14(1):3–25, 1978.
42. R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proc. of STOC*, pages 99–108. ACM, 1983.
43. R. Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.
44. S. Khot. Inapproximability results for computational problems on lattices. Chapter of [64].
45. P. N. Klein. Finding the closest lattice vector when it's unusually close. In *Proc. of SODA*, pages 937–941. ACM, 2000.
46. A. Korkine and G. Zolotarev. Sur les formes quadratiques. *Math. Ann*, 6:336–389, 1873.

47. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann*, 261:515–534, 1982.
48. H. Lenstra Jr. *Algorithmic number theory, J. P. Buhler, P. Stevenhagen (eds)*, chapter Lattices, pages 127–181. MSRI Publications, Cambridge University Press, 2008.
49. R. Lindner and M. Rückert. The lattice challenge homepage. `http://www.latticechallenge.org/`.
50. Y.-K. Liu, V. Lyubashevsky, and D. Micciancio. On bounded distance decoding for general lattices. In *Proc. of RANDOM*, volume 4110 of *LNCS*, pages 450–461. Springer, 2006.
51. V. Lyubashevsky and D. Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *Proc. of CRYPTO*, volume 5677 of *LNCS*, pages 577–594. Springer, 2009.
52. J. Martinet. *Perfect Lattices in Euclidean Spaces*. Springer, 2002.
53. D. Micciancio. Efficient reductions among lattice problems. In *Proc. of SODA*, pages 84–93. SIAM, 2008.
54. D. Micciancio and S. Goldwasser. *Complexity of lattice problems: a cryptographic perspective*. Kluwer Academic Press, 2002.
55. D. Micciancio and O. Regev. Lattice-based cryptography. In *Post-Quantum Cryptography, D. J. Bernstein, J. Buchmann, E. Dahmen (Eds)*, pages 147–191. Springer, 2009.
56. D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. `http://cseweb.ucsd.edu/~pvoulgar/pub.html` – Draft of the full version of [57], dated December 8, 2010.
57. D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proc. of STOC*, pages 351–358. ACM, 2010.
58. D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proc. of SODA*. ACM, 2010.
59. H. Minkowski. *Geometrie der Zahlen*. Teubner-Verlag, 1896.
60. W. H. Mow. Maximum likelihood sequence estimation from the lattice viewpoint. *IEEE Transactions on Information Theory*, 40:1591–1600, 1994.
61. W. H. Mow. Universal lattice decoding: Principle and recent advances. *Wireless Communications and Mobile Computing, Special Issue on Coding and Its Applications in Wireless CDMA Systems*, 3(5):553–569, 2003.
62. P. Q. Nguyen. Hermite's constant and lattice algorithms. Chapter of [64].
63. P. Q. Nguyen and D. Stehlé. LLL on the average. In *Proc. of ANTS*, LNCS, pages 238–256. Springer, 2006.
64. P. Q. Nguyen and B. Vallée (editors). *The LLL Algorithm: Survey and Applications*. Information Security and Cryptography. Springer, 2009.
65. P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2), 2008.
66. A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In *Cryptology and Computational Number Theory*, volume 42 of *Proc. of Symposia in Applied Mathematics*, pages 75–88. A.M.S., 1990.
67. X. Pujol and D. Stehlé. Rigorous and efficient short lattice vectors enumeration. In *Proc. of ASIACRYPT*, volume 5350 of *LNCS*, pages 390–405. Springer, 2008.
68. X. Pujol and D. Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. Cryptology ePrint Archive, 2009. `http://eprint.iacr.org/2009/605`.

69. O. Regev. Lecture notes of *lattices in computer science*, taught at the Computer Science Tel Aviv University. `http://www.cs.tau.il/~odedr`.
70. O. Regev. On the complexity of lattice problems with polynomial approximation factors. Chapter of [64].
71. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proc. of STOC*, pages 84–93. ACM, 2005.
72. O. Regev. The learning with errors problem, 2010. Invited survey in CCC 2010, available at `http://www.cs.tau.ac.il/~odedr/`.
73. C. P. Schnorr. Progress on LLL and lattice reduction. Chapter of [64].
74. C. P. Schnorr. A hierarchy of polynomial lattice basis reduction algorithms. *Theor. Comput. Science*, 53:201–224, 1987.
75. C. P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematics of Programming*, 66:181–199, 1994.
76. C. P. Schnorr and H. H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *Proc. of Eurocrypt*, volume 921 of *LNCS*, pages 1–12. Springer, 1995.
77. V. Shoup. NTL, Number Theory C++ Library. `http://www.shoup.net/ntl/`.
78. C. L. Siegel. *Lectures on the Geometry of Numbers*. Springer, 1989.
79. N. Sommer, M. Feder, and O. Shalvi. Finding the closest lattice point by iterative slicing. *SIAM J. Discrete Math.*, 23(2):715–731, 2009.
80. D. Stehlé and M. Watkins. On the extremality of an 80-dimensional lattice. In *Proc. of ANTS*, volume 6197 of *LNCS*, pages 340–356. Springer, 2010.
81. G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.
82. P. Voulgaris. Personal communication.
83. X. Wang, M. Liu, C. Tian, and J. Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. Cryptology ePrint Archive, 2010. `http://eprint.iacr.org/2010/647`.

Appendix

The proofs for the given version of `AKS` and the improved version based on the birthday paradox have not been published in detail yet. For reference, we give a compact proof for all theorems of Section 4, largely based on [65, 58, 68].

A Analysis of the AKS algorithm

Lemma A.1 (Adapted from [58, Th. 4.1]) *Let* $c_t = -\log_2 \gamma + 0.401$. *If* $\mathcal{T}$ *is a set of points in* $\mathcal{B}_n(R)$ *such that the distance between two points is at least* γR, *then* $|\mathcal{T}|$ *is at most* $N_T(n) = 2^{c_t n + o(n)}$. *In particular, for any application of the* `Sieve` *function, the number of centers is at most* $2^{c_t n + o(n)}$.

Proof. Let $\alpha = 1 + \frac{1}{n}$. The ball $\mathcal{B}_n(\gamma R/3)$ contains at most one point. We cover $\mathcal{B}_n(R) \setminus \mathcal{B}_n(\gamma R/3)$ with coronas $T_r = \mathcal{B}_n(\alpha r) \setminus \mathcal{B}_n(r)$ for $r = \frac{\gamma R}{3}, \frac{\gamma R}{3}\alpha, \ldots, \frac{\gamma R}{3}\alpha^k$, with $k = \lceil n \log_2 \frac{3}{\gamma} \rceil = O(n)$. It suffices to prove that any corona T_r contains at most $2^{c_t n + o(n)}$ points.

Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be two distinct points in $T_r \cap \mathcal{B}_n(R)$. We have $\langle \boldsymbol{u} - \boldsymbol{v}, \boldsymbol{u} - \boldsymbol{v} \rangle \geq (\gamma R)^2$, so $\langle \boldsymbol{u}, \boldsymbol{v} \rangle \leq \frac{1}{2}\left(\|\boldsymbol{u}\|^2 + \|\boldsymbol{v}\|^2 - (\gamma R)^2\right)$. We let $\phi_{\boldsymbol{u},\boldsymbol{v}}$ denote the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$. The above implies that:

$$\begin{aligned}\cos \phi_{\boldsymbol{u},\boldsymbol{v}} = \frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{\|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|} &\leq \frac{1}{2}\left(\frac{\|\boldsymbol{u}\|}{\|\boldsymbol{v}\|} + \frac{\|\boldsymbol{v}\|}{\|\boldsymbol{u}\|} - \frac{(\gamma R)^2}{\|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|}\right) \\ &\leq 1 + \frac{1}{n} - \frac{(\gamma R)^2}{2R^2} = 1 + \frac{1}{n} - \frac{\gamma^2}{2} \underset{n \to \infty}{\longrightarrow} 1 - \frac{\gamma^2}{2}.\end{aligned}$$

For any $\varepsilon \in (0, \frac{\gamma^2}{2})$ and sufficiently large n we can apply Theorem 4.1 with $\phi_0 = \cos^{-1}\left(1 - \frac{\gamma^2}{2} + \varepsilon\right) \leq 60°$. This provides the result. □

Lemma A.2 *Let $c_b = \log_2 r_0 + 0.401$. For any lattice L, we have $|\mathcal{B}_n(r_0\lambda) \cap L| \leq N_B(n) = 2^{c_b n + o(n)}$.*

Proof. The distance between two lattice points is at least $\lambda(L)$. Let $\varepsilon > 0$ and $\gamma = \frac{1}{r_0(1+\varepsilon)}$. Provided that $n > \frac{1}{\varepsilon}$, we have $\frac{\lambda(L)}{r_0\lambda} \geq \gamma$. Applying Lemma A.1 with $R = r_0\lambda$ and γ for any $\varepsilon > 0$ provides the result. □

Lemma A.3 ([65, Th. 3.4]) *Let $c_g = -\frac{1}{2}\log_2(1 - \frac{1}{4\xi^2})$ and $\boldsymbol{s}$ be a shortest non-zero vector of L. Let $I_{\boldsymbol{s}} = \mathcal{B}_n(\boldsymbol{0}, \xi\lambda) \cap \mathcal{B}_n(-\boldsymbol{s}, \xi\lambda)$. If $\boldsymbol{x}$ is chosen uniformly in $\mathcal{B}_n(\boldsymbol{0}, \xi\lambda)$, then $\Pr(\boldsymbol{x} \in I_{\boldsymbol{s}}) \geq \frac{1}{N_G(n)}$ with $N_G(n) = 2^{c_g n + o(n)}$.*

Lemma A.4 *If* `AKS` *is run with $N \geq 1.01 N_G(n)(n^3 N_T(n) + N_B(n) + 1)$, then it returns a shortest non-zero vector with probability $\geq \frac{1}{2} - \underset{n \to \infty}{o}(1)$.*

Proof. From Lemma A.3, the initial set $\mathcal{T}$ contains at least $n^3 N_T(n) + N_B(n) + 1$ vectors such that $\boldsymbol{u}' - \boldsymbol{u} \in I_{\boldsymbol{s}} = \mathcal{B}_n(\boldsymbol{0}, \xi\lambda) \cap \mathcal{B}_n(-\boldsymbol{s}, \xi\lambda)$ with probability exponentially close to 1 as $n \to \infty$. Since no more than $\left\lceil \log_\gamma\left(\frac{\xi}{nR(1-\gamma)}\right)\right\rceil N_T(n) = O(n^2 N_T(n)) = o(n^3 N_T(n))$ vectors are used as centers and lost, this implies that the final set contains at least $N_B(n) + 1$ vectors such that $\boldsymbol{x} = \boldsymbol{u}' - \boldsymbol{u} \in I_{\boldsymbol{s}}$.

For $\boldsymbol{x} \in \mathcal{B}_n(\boldsymbol{0}, \xi\lambda)$, let $\tau(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{s}$ if $\boldsymbol{x} \in I_{\boldsymbol{s}}$, $\tau(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{s}$ if $\boldsymbol{x} - \boldsymbol{s} \in I_{\boldsymbol{s}}$ and $\tau(\boldsymbol{x}) = \boldsymbol{x}$ otherwise. Consider `AKS2`, a modified version of `AKS` where τ is applied to each x_i with probability $\frac{1}{2}$ immediately after it is chosen. If $\boldsymbol{x}$ is sampled uniformly in $\mathcal{B}_n(\boldsymbol{0}, \xi\lambda)$, then so is $\tau(\boldsymbol{x})$ (assuming

that $\xi < 1$). As a consequence, the outputs of `AKS` and `AKS2` follow the same distribution.

Since $\tau(\boldsymbol{x}) - \boldsymbol{x} \in L$, the lattice vector generated with the perturbation $\tau(\boldsymbol{x})$ is the same as the lattice vector generated with the perturbation $\boldsymbol{x}$. When the same random perturbations are chosen at the beginning, this implies that the behaviors of `AKS` and `AKS2` (test results and control flow) are exactly the same until the very last line of the algorithm. At the last line of `AKS`, at least one vector for which the perturbation is in $I_{\boldsymbol{s}}$ appears twice in $\mathcal{T}$ with probability exponentially close to 1. In `AKS2`, with probability close to $\frac{1}{2}$, the function τ is applied to exactly one of the two perturbations which have generated the lattice vector. If this occurs, the difference between the two lattice vectors corresponding to the same perturbation in `AKS2` is exactly $\boldsymbol{s}$. This implies that `AKS2` returns a correct result with probability arbitrarily close to $\frac{1}{2}$. Because of the property given above on the output distribution, this result also holds for `AKS`. □

Proof of Theorem 4.2. When the number of points is chosen according to Lemma A.4, the space complexity is $\leq 2^{c_{\text{space}}n+o(n)}$ with $c_{\text{space}} = c_g + \max(c_t, c_b)$. The time complexity for the sieve is $\leq 2^{(c_{\text{space}}+c_t)n+o(n)}$. The removal of pairs corresponding to the same lattice points may be performed in time $\leq 2^{(c_{\text{space}}+c_b)n+o(n)}$. Finally, the time complexity for the computation of pairwise differences is $\leq 2^{2c_b n+o(n)}$. So the global time complexity is $\leq 2^{c_{\text{time}}n+o(n)}$ with $c_{\text{time}} = \max(c_{\text{space}} + c_t, c_{\text{space}} + c_b, 2c_b)$, i.e., $c_{\text{time}} = c_g + 2\max(c_t, c_b)$.

The optimal choice for constants ξ and γ to minimize the time complexity is $\gamma \approx 0.496$, $\xi \approx 0.676$ which leads to $c_{\text{space}} \leq 1.984$ and $c_{\text{time}} \leq 3.397$. Note that we have only proved that the algorithm succeeds with non-negligible probability, but running it n times ensures that it succeeds with probability exponentially close to 1 without significantly increasing the complexity. □

Improvement based on the birthday paradox. The number of sampled pairs if fixed to $N_G(n)(n^5 N_T(n) + n\sqrt{N_B(n)})$. Let $K = O(n^2)$ be the number of sieving steps. For each step, $\left\lfloor \frac{n^5}{K} N_G(n) N_T(n) \right\rfloor = \Omega\left(\frac{n^3}{K} N_G(n) N_T(n)\right)$ pairs are set aside to be used as centers.

At the beginning of the algorithm, among the pairs set aside to be used as centers for the first step, there are $\Omega(n^3 N_T(n))$ good pairs with high probability. As these pairs are processed during a sieving step, the probability that the distance between the next perturbed vector and the closest center is more than γR decreases. The sum of these probabilities

is bounded from above by the expectancy of the number of centers, which is $\leq N_T(n)$. As a consequence, once all centers have been processed, the probability for any of the subsequent pairs to be lost is $O\left(\frac{1}{n^3}\right)$. By induction, it can be proved the same proportion of pairs are lost at each step of the sieve with high probability. As a consequence, no more than $1-(1-\frac{1}{n^3})^{O(n^2)} = O\left(\frac{1}{n}\right)$ pairs are lost during the whole algorithm. This means that, in the final ball $\mathcal{B}_n(\mathbf{0}, \gamma R)$, there are $\Omega(n\sqrt{N_B(n)})$ (probabilistically) independent lattice points corresponding to good pairs with high probability. The birthday paradox implies that a collision occurs with probability close to 1. As in the proof of Lemma A.4, this implies that the algorithm returns a shortest vector with probability $\geq \frac{1}{2} - o(1)$.

The analysis above leads to the following further modifications of Algorithm 6. At the end of the sieving steps, we keep only $N_G(n)\sqrt{N_B(n)}2^{o(n)}$ distinct lattice vectors, before listing all the differences between these vectors to disclose a shortest non-zero lattice vector.

In this birthday paradox variant, the space complexity is $\leq 2^{c_{\text{space}}n+o(n)}$ with $c_{\text{space}} = c_g + \max(c_t, c_b/2)$, and the time complexity is $\leq 2^{c_{\text{time}}n+o(n)}$ with $c_{\text{time}} = \max(c_{\text{space}}+c_t, c_{\text{space}}+c_g+c_b/2, 2c_g+c_b)$. The optimal choice for constants ξ and γ to minimize the time complexity is $\gamma \approx 0.609$, $\xi = 1$ which leads to $c_{\text{space}} \leq 1.325$ and $c_{\text{time}} \leq 2.648$.

B Analysis of the ListSieve algorithm

Lemma B.1 (Adapted from [58, Th. 4.1]) *Let $\xi > \frac{1}{2}$, $r_0 > 2\xi$ and $c_t = -\frac{1}{2}\log_2(1-\frac{2\xi}{r_0}) + 0.401$. At any moment during the execution of* `ListSieve`*, the list $\mathcal{T}$ contains at most $N_T(n) = 2^{c_t n+o(n)}$ vectors of norm at least $r_0\lambda$.*

Proof. We first bound the norm of any vector of $\mathcal{T}$. `NewPair` returns $(\boldsymbol{t}, \boldsymbol{t}')$ such that $\boldsymbol{t}' \in \mathcal{P}(B)$ and $\|\boldsymbol{t}'-\boldsymbol{t}\| \leq \xi\lambda$. We have assumed that $\max_i \|\boldsymbol{b}_i\| = 2^{O(n)}\lambda$. Hence $\|\boldsymbol{t}'\| \leq n \cdot \max_i \|\boldsymbol{b}_i\| \leq 2^{O(n)}\lambda$. After applying `Reduction`, the norm of $\boldsymbol{t}'$ does not increase and $\boldsymbol{t}'-\boldsymbol{t}$ is unchanged, so, for any $\boldsymbol{t}_i \in \mathcal{T}$, we have $r_0\lambda \leq \|\boldsymbol{t}_i\| \leq (2^{O(n)}+\xi)\lambda$. It now suffices to prove that any $\mathcal{T}_r = \{\boldsymbol{t}_i \in \mathcal{T} \mid r\lambda \leq \|\boldsymbol{t}_i\| \leq \left(1+\frac{1}{n}\right) r\lambda\}$ for $r \geq r_0$ contains at most $2^{c_t n+o(n)}$ points. Indeed, the list $\mathcal{T}$ is contained in a union of $O(n^2)$ sets $\mathcal{T}_r$.

Let $i < j$ such that $\boldsymbol{t}_i, \boldsymbol{t}_j \in \mathcal{T}_r$. The idea of the proof is that for large n, the angle between $\boldsymbol{t}_j'$ and $\boldsymbol{t}_i$ is not far from being above $\frac{\pi}{3}$ because $\boldsymbol{t}_i$ was already in $\mathcal{T}$ when $\boldsymbol{t}_j$ was reduced. We use the inequality $\|\boldsymbol{t}_j - \boldsymbol{t}_j'\| \leq \xi\lambda$ to obtain a lower bound for the angle $\phi_{\boldsymbol{t}_i,\boldsymbol{t}_j}$ between $\boldsymbol{t}_i$ and $\boldsymbol{t}_j$ and then apply Theorem 4.1.

Note that $\|\boldsymbol{t}'_j\| \leq \|\boldsymbol{t}_j\| + \xi\lambda \leq 3r\lambda$. Since $\boldsymbol{t}_j$ was added after $\boldsymbol{t}_i$, we have:

$$\|\boldsymbol{t}'_j - \boldsymbol{t}_i\|^2 > \left(1 - \frac{1}{n}\right)^2 \|\boldsymbol{t}'_j\|^2 \geq \left(1 - \frac{2}{n}\right) \|\boldsymbol{t}'_j\|^2$$
$$\langle \boldsymbol{t}'_j, \boldsymbol{t}_i \rangle < \frac{1}{2}\left[\|\boldsymbol{t}_i\|^2 + \frac{2}{n}\|\boldsymbol{t}'_j\|^2\right] \leq \frac{1}{2}\|\boldsymbol{t}_i\|^2 + \frac{1}{n}(3r\lambda)^2.$$

Moreover, we have $\langle \boldsymbol{t}_j - \boldsymbol{t}'_j, \boldsymbol{t}_i \rangle \leq \xi\lambda\|\boldsymbol{t}_i\|$. We can now bound $\cos(\phi_{\boldsymbol{t}_i, \boldsymbol{t}_j})$.

$$\begin{aligned}
\langle \boldsymbol{t}_j, \boldsymbol{t}_i \rangle = \langle \boldsymbol{t}'_j, \boldsymbol{t}_i \rangle + \langle \boldsymbol{t}_j - \boldsymbol{t}'_j, \boldsymbol{t}_i \rangle &\leq \frac{1}{2}\|\boldsymbol{t}_i\|^2 + \frac{1}{n}(3r\lambda)^2 + \xi\lambda\|\boldsymbol{t}_i\| \\
\cos(\phi_{\boldsymbol{t}_i, \boldsymbol{t}_j}) = \frac{\langle \boldsymbol{t}_j, \boldsymbol{t}_i \rangle}{\|\boldsymbol{t}_i\| \cdot \|\boldsymbol{t}_j\|} &\leq \frac{1}{2}\frac{\|\boldsymbol{t}_i\|}{\|\boldsymbol{t}_j\|} + \frac{1}{n} \cdot \frac{(3r\lambda)^2}{\|\boldsymbol{t}_i\| \cdot \|\boldsymbol{t}_j\|} + \frac{\xi\lambda}{\|\boldsymbol{t}_j\|} \\
&\leq \frac{1}{2}\left(1 + \frac{1}{n}\right) + \frac{9}{n} + \frac{\xi}{r} \\
&\leq \frac{1}{2} + \frac{\xi}{r_0} + O\left(\frac{1}{n}\right).
\end{aligned}$$

The bound on $|\mathcal{T}_r|$ follows directly from Theorem 4.1. □

Lemma B.2 *Let $c_t = \log_2(\xi + \sqrt{\xi^2 + 1}) + 0.401$. At any moment during the execution of* `ListSieve`*, the list $\mathcal{T}$ contains at most $N_T(n) = 2^{c_t n + o(n)}$ vectors.*

Proof. Let $r_0 > 2\xi$. Since the list $\mathcal{T}$ contains only lattice points, one can combine Lemma B.1 with Lemma A.2 to obtain the upper bound $|\mathcal{T}| \leq 2^{\max(c_t, c_b)n + o(n)}$. This upper bound is minimized when $c_t = c_b$, which occurs when $r_0 = \xi + \sqrt{\xi^2 + 1}$. □

Proof of Theorem 4.3 [Adapted from [58, Sec. 4.1]]. Let $I_{\boldsymbol{s}} = \mathcal{B}_n(\mathbf{0}, \xi\lambda) \cap \mathcal{B}_n(-\boldsymbol{s}, \xi\lambda)$, $I_{-\boldsymbol{s}} = I_{\boldsymbol{s}} + \boldsymbol{s}$ and $J = I_{\boldsymbol{s}} \cup I_{-\boldsymbol{s}}$. A pair $(\boldsymbol{u}, \boldsymbol{u}')$ is *good* if the perturbation $\boldsymbol{x} = \boldsymbol{u}' - \boldsymbol{u}$ is in J. The number of points in $\mathcal{T}$ is bounded by $N_T(n)$. All perturbations are sampled independently with uniform distribution and the probability that a perturbation $\boldsymbol{x}$ is in J is at least $2N_G(n)^{-1}$ (Lemma A.3). For $\boldsymbol{x} \in \mathcal{B}_n(0, \xi\lambda)$, we define $\tau(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{s}$ if $\boldsymbol{x} \in I_{\boldsymbol{s}}$, $\tau(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{s}$ if $\boldsymbol{x} \in I_{-\boldsymbol{s}}$ and $\tau(\boldsymbol{x}) = \boldsymbol{x}$ otherwise (τ is well defined provided that $\xi < 1$). We define the following events:

- E_i: "the lattice point produced by the reduction of the i-th good pair is added to the list" (assuming that we stop the algorithm when at least i good pairs have been sampled);

- F: "`ListSieve` returns a shortest non-zero vector";
- G: "at least $2N_T(n)$ good pairs are sampled".

When $N = 1.01N_T(n)N_G(n)$, $\Pr(G)$ tends to 1 as n increases, so if n is large enough $\Pr(G) \geq \frac{1}{2}$. The fact that $\sum_{i=1}^{2N_T(n)} \Pr(E_i|G) \leq N_T(n)$ implies that for some $i \leq 2N_T(n)$, we have $\Pr(E_i|G) \leq \frac{1}{2}$.

If $\boldsymbol{x}_i$ is sampled with uniform distribution in J, so is $\tau(\boldsymbol{x}_i)$. Consider the i-th good pair. Assume that $\boldsymbol{x}_i$ is such that after the reduction of the pair $(\boldsymbol{u}_i, \boldsymbol{u}'_i)$, the vector $\boldsymbol{u}_i$ is already in the list. With the perturbation $\tau(\boldsymbol{x}_i) = \boldsymbol{x}_i \pm \boldsymbol{s}$, the pair would be $(\boldsymbol{u}_i \pm \boldsymbol{s}, \boldsymbol{u}'_i)$ so the reduction of the pair would produce $\boldsymbol{t} \pm \boldsymbol{s}$. It might or might not already be in the list. In both cases, the algorithm succeeds. This implies that $\Pr(F|G) \geq \Pr\left(\bar{E}|G\right) \geq \frac{1}{2}$ so $\Pr(F) \geq \Pr(F|G)\Pr(G) \geq \frac{1}{4}$. □

C Analysis of the ListSieveBirthday algorithm

What follows is a short version of [68]. The bound $N_T(n)$ from Lemma B.1 holds for `ListSieveBirthday` and is an upper bound on $|\mathcal{T}|$ The first part of the proof of Theorem 4.4 consists in proving that $|\mathcal{U}|$ is large enough with non-negligible probability. It relies on Lemmas A.2 and A.3, where $N_B(n)$, $N_G(n)$ and $I_{\boldsymbol{s}}$ are defined.

Lemma C.1 *Let $N_1^{\max} = 4N_G(n)N_T(n)$. Consider* `ListSieveBirthday` *with $\xi > \frac{1}{2}$, $r_0 > 2\xi$ and $N_1 = N_1^{\max}$. For $i \leq N_1^{\max}$, we define the event $E_i : \|\boldsymbol{t}_i\| < r_0\lambda$. We let $p_i = \Pr(E_i \mid \boldsymbol{x}_i \in I_{\boldsymbol{s}})$, where the probability is taken over the randomness of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i$, and $J = \{i \leq N_1^{\max} : p_i \leq \frac{1}{2}\}$. Then $|J| \leq N_1^{\max}/2$.*

Proof. Assume (for contradiction) that $|J| > N_1^{\max}/2$. Then Lemma A.3 implies that $\sum_{i\in J}(1-p_i)\Pr(\boldsymbol{x}_i \in I_{\boldsymbol{s}}) \geq \frac{|J|}{2N_G} > N_T$. This contradicts the following (the last inequality comes from the bound $|\mathcal{T}| \leq N_T(n)$):

$$\begin{aligned}\sum_{i\in J}(1-p_i)\Pr(\boldsymbol{x}_i \in I_{\boldsymbol{s}}) &= \sum_{i\in J}\Pr(\neg E_i \cap (\boldsymbol{x}_i \in I_{\boldsymbol{s}})) \\ &\leq \sum_{i\geq 1}\Pr(\neg E_i) \leq N_T. \qquad \square\end{aligned}$$

In the second loop of `ListSieveBirthday`, we do not add any point to $\mathcal{T}$. Therefore, the points that are added to $\mathcal{U}$ are iid. The procedure to reduce points being the same in both loops, we have that for any $i \leq N_2$ such that $\boldsymbol{y}_i \in I_{\boldsymbol{s}}$, the probability that $\|\boldsymbol{u}_i\| < r_0\lambda$ is p_{N_1+1}. When N_1 is

sampled uniformly in $[0, N_1^{\max} - 1]$, we have $p_{N_1+1} \geq \frac{1}{2}$ with probability $\geq \frac{1}{2}$, by Lemma C.1.

Lemma C.2 *If n is sufficiently large, then with probability $\geq 1/4$ (taken over the randomness of N_1, the x_k's and the y_k's), there exist two distinct indices $i, j \leq N_2$ such that $\boldsymbol{u}_i = \boldsymbol{u}_j$ and $\boldsymbol{y}_i, \boldsymbol{y}_j \in I_{\boldsymbol{s}}$.*

Proof. In this proof, we assume that $p_{N_1+1} \geq \frac{1}{2}$. This occurs with probability $\geq \frac{1}{2}$ and implies that $\Pr(\|\boldsymbol{u}_i\| \leq r_0\lambda \mid \boldsymbol{y}_i \in I_{\boldsymbol{s}}) \geq \frac{1}{2}$ for all $i \leq N_2$. Let $X = |\{i \leq N_2 : (\|\boldsymbol{u}_i\| \leq r_0\lambda) \cap (\boldsymbol{y}_i \in I_{\boldsymbol{s}})\}|$. Lemma A.3 gives

$$\begin{aligned}\Pr((\|\boldsymbol{u}_i\| \leq r_0\lambda) \cap (\boldsymbol{y}_i \in I_{\boldsymbol{s}})) &= \Pr(\|\boldsymbol{u}_i\| \leq r_0\lambda \mid \boldsymbol{y}_i \in I_{\boldsymbol{s}}) \Pr(\boldsymbol{y}_i \in I_{\boldsymbol{s}}) \\ &\geq \frac{1}{2N_G}.\end{aligned}$$

Let $N = 2\lceil\sqrt{N_B}\rceil$. The variable X has a binomial distribution of parameter $p \geq \frac{1}{2N_G}$. We have $\mathbb{E}(X) = pN_2 \geq 2N$ and $\mathrm{Var}(X) = p(1-p)N_2 \leq \mathbb{E}(X)$. Therefore, by using Chebyshev's inequality, we have (since $N_B \geq 25$ holds for n large enough, we have $N \geq 10$):

$$\begin{aligned}\Pr(X \leq N) &\leq \Pr(|X - \mathbb{E}(X)| \geq \mathbb{E}(X) - N) \\ &\leq \frac{\mathrm{Var}(X)}{(\mathbb{E}(X) - N)^2} \leq \frac{\mathbb{E}(X)}{(\mathbb{E}(X) - N)^2} \leq \frac{2}{N} \leq \frac{1}{5}.\end{aligned}$$

So with probability $\geq \frac{4}{5}$, `ListSieveBirthday` samples at least N iid points in $S_0 = \mathcal{B}_n(r_0\lambda) \cap L$. The probability that a collision occurs is minimized when the distribution is uniform, i.e., the probability of each point is $1/|S_0|$. Since we have chosen $N \geq \sqrt{|S_0|}$ (by Lemma A.2), the birthday paradox implies that the probability that i and j exist will be large. More precisely it is bounded from below by

$$\frac{4}{5}\left(1 - \prod_{i<N}\left(1 - \frac{i}{|S_0|}\right)\right) \geq \frac{4}{5}\left(1 - \mathrm{e}^{-\frac{N(N-1)}{2N_B}}\right) \geq \frac{4}{5}\left(1 - \frac{1}{\mathrm{e}}\right) \geq \frac{1}{2},$$

where we used the fact that $|S_0| \leq N_B(n)$ (by Lemma A.2). □

Proof of Theorem 4.4. Introducing a modification `ListSieveBirthday2` of `ListSieveBirthday` that applies τ to every y_i with probability $\frac{1}{2}$ and using the same reasoning as in the proof of Lemma A.4 leads to the fact that the algorithm succeeds with probability $\geq \frac{1}{8}$.

The space complexity is $|\mathcal{T}| + |\mathcal{U}|$. We have $|\mathcal{T}| \leq 2^{c_t n + o(n)}$, and, by definition of N_2, we have $|\mathcal{U}| \leq 2^{(c_g + c_b/2)n + o(n)}$. Since $\|\boldsymbol{b}_i\| = 2^{O(n)}\lambda$ for

all i, the complexity of `Reduction` is $|\mathcal{T}|\mathcal{P}oly(n, |B|)$. Omitting the polynomial factor, the time complexity of the first loop is $|\mathcal{T}|N_1 \leq |\mathcal{T}|N_1^{\max} \leq 2^{(c_g+2c_t)n+o(n)}$. The time required to find a closest pair of points in $\mathcal{U}$ with the naive algorithm is $|\mathcal{U}|^2$. Finally, the time complexity of the second loop is $|\mathcal{T}| \cdot |\mathcal{U}| \leq 2^{(c_t+c_g+c_b/2)n+o(n)}$, which is negligibly smaller than the cost of one of the other components.

The time complexity bound is minimized when $2c_t = c_g + c_b$. By Lemmas A.2, B.1 and A.3, this is equivalent to $r_0 = 2\xi + 2^{0.401}\sqrt{1 - \frac{1}{4\xi^2}}$. Optimizing with respect to ξ leads to $\xi \simeq 0.9476$, $r_0 \simeq 3.0169$, $c_{\text{time}} \leq 2.465$ and $c_{\text{space}} \leq 1.233$. Calling the algorithm n times allows to ensure that it succeeds with probability exponentially close to 1. □