A Deterministic Single Exponential Time Algorithm for Most Lattice Problems based on Voronoi Cell Computations *†

Daniele Micciancio Panagiotis Voulgaris

December 8, 2010

Abstract

We give deterministic $\tilde{O}(2^{2n})$-time and $\tilde{O}(2^n)$-space algorithms to solve all the most important computational problems on point lattices in NP, including the Shortest Vector Problem (SVP), Closest Vector Problem (CVP), and Shortest Independent Vectors Problem (SIVP). This improves the $n^{O(n)}$ running time of the best previously known algorithms for CVP (Kannan, Math. Operation Research 12(3):415-440, 1987) and SIVP (Micciancio, Proc. of SODA, 2008), and gives a deterministic and asymptotically faster alternative to the $2^{O(n)}$-time (and space) randomized algorithm for SVP of (Ajtai, Kumar and Sivakumar, STOC 2001). The core of our algorithm is a new method to solve the closest vector problem with preprocessing (CVPP) that uses the Voronoi cell of the lattice (described as intersection of half-spaces) as the result of the preprocessing function.

Terms: Algorithms, Performance, Theory
Keywords: Lattice algorithms, SVP, CVP, SIVP, Voronoi Cell

1 Introduction

A d-dimensional lattice Λ is a discrete subgroup of the Euclidean space $\mathbb{R}^d$, and is customarily represented as the set of all integer linear combinations of $n \leq d$ basis vectors $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{d \times n}$. There are many famous algorithmic problems on point lattices, the most important of which are:

- The shortest vector problem (SVP): given a basis $\mathbf{B}$, find a shortest nonzero vector in the lattice generated by $\mathbf{B}$.
- The closest vector problem (CVP): given a basis $\mathbf{B}$ and a target vector $\mathbf{t} \in \mathbb{R}^d$, find the lattice vector generated by $\mathbf{B}$ that is closest to $\mathbf{t}$.
- The shortest independent vectors problem (SIVP): given a basis $\mathbf{B}$, find n linearly independent lattice vectors in the lattice generated by $\mathbf{B}$ that are as short as possible.

Beside being classic mathematical problems in the study of the *geometry of numbers* [15], these problems play an important role in many computer science and communication theory applications. SVP and CVP have been used to solve many landmark algorithmic problems in theoretical computer science, like integer programming [35, 31], factoring polynomials over the rationals [34], checking the solvability by radicals [33], solving low density subset-sum problems [19] and breaking the Merkle-Hellman cryptosystem [45] (among many other cryptanalysis problems [30, 43].) SIVP is the main problem underlying the construction of

*An extended abstract of this paper was presented in STOC 2010.

†Supported in part by NSF grants CCF-0634909 and CNS-0831536. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

lattice based cryptographic hash functions with worst-case/average-case connection [4, 41]. SVP and CVP also have many applications in communication theory, e.g., lattice coding for the Gaussian channel and vector quantization [18].

The complexity of lattice problems has been investigated intensively. All three problems mentioned above have been shown to be NP-hard both to solve exactly [54, 3, 13], or even approximate within small (constant or sub-polynomial in n) approximation factors [13, 8, 20, 14, 37, 32, 28]. Much effort has gone into the development and analysis of algorithms both to solve these problems exactly [31, 29, 26, 11, 5, 6, 12] and to efficiently find approximate solutions [34, 49, 48, 51, 50, 22, 23].

In this paper we focus on the complexity of finding exact solutions to these problems. Of course, as the problems are NP-hard, no polynomial time solution is expected to exist. Still, the complexity of solving lattice problems exactly is interesting both because many applications (e.g., in mathematics and communication theory [18]) involve lattices in relatively small dimension, and because approximation algorithms for high dimensional lattices [48, 51, 22, 23] (for which exact solution is not feasible) typically involve the exact solution of low dimensional subproblems. The best deterministic polynomial time algorithm to solve any of these lattice problems exactly is still essentially the one discovered by Kannan [31] in 1983, running in time $n^{O(n)}$, where n is the dimension of the lattice. Subsequent work [29, 26] lead to improvements in the constant in the exponent, mostly through a better analysis, reducing the upper bound on the running time down to $n^{0.184n}$ for SVP and $n^{0.5n}$ for CVP and SIVP. The only problem that has seen asymptotically significant improvements in the exponent is SVP, for which Ajtai, Kumar and Sivakumar [5] gave a *randomized* algorithm running in time (and space) $2^{O(n)}$, typically referred to as the AKS Sieve. Following [5] much work has been devoted to better understand and improve the Sieve algorithm. Still the main questions posed in [5] didn't see much progress. Is the use of randomization (and exponential space) necessary to lower the time complexity of SVP from $n^{O(n)}$ to $2^{O(n)}$? Can algorithms with similar running time be devised for other lattice problems, like SIVP and CVP?

In [44, 42, 46], improved analysis and variants of the AKS sieve are studied, but still using the same approach leading to randomized algorithms. Extensions of the AKS sieve algorithm to other lattice problems like CVP and SIVP have been investigated in [6, 12], but only led to approximation algorithms which are not guaranteed (even probabilistically) to find the best solution, except for certain very special classes of lattices [12]. A possible explanation for the difficulty of extending the result of [5] to the exact solution of SIVP and CVP was offered by Micciancio in [39], where it is shown (among other things) that CVP, SIVP and all other lattice problems considered in [12], with the exception of SVP, are equivalent in their exact version under deterministic polynomial time dimension preserving reductions. So, either all of them are solvable in single exponential time $2^{O(n)}$, or none of them admits such an algorithm.

In this paper we resolve this question in the affirmative, giving a deterministic single exponential time algorithm for CVP, and therefore by the reductions in [24, 39], also to SVP, SIVP and several other lattice problems in NP considered in the literature. This improves the time complexity of the best previously known algorithm for CVP, SIVP, etc. [31] from $n^{O(n)}$ to $2^{O(n)}$. In the case of SVP, we achieve single exponential time as in [5, 44, 42, 46], but without using randomization and achieving a better constant in the exponent. In the process, we also provide deterministic single exponential time algorithms for various other classic computational problems in lattices, like computing the kissing number, and computing the list of all Voronoi relevant vectors.

We remark that all our algorithms, just like [5], use exponential space. So, the question whether exponential space is required to solve lattice problems in single exponential time remains open.

1.1 Our techniques.

At the core of all our results is a new technique for the solution of the closest vector problem with preprocessing (CVPP). We recall that CVPP is a variant of CVP where some side information about the lattice is given as a hint together with the input. The hint may depend on the lattice, but not on the target vector. Typically, in the context of polynomial time algorithms, the hint is restricted to have polynomial size, but since here we study exponential time algorithms, one can reasonably consider hints that have size $2^{O(n)}$. The hint used by our algorithm is a description of the Voronoi cell of the lattice. We recall that the Voronoi

cell of a lattice is the set $\mathcal{V}$ of all points (in Euclidean space) that are closer to the origin than to any other lattice point. The Voronoi cell $\mathcal{V}$ is a convex body, symmetric about the origin, and can be described as the intersection of half-spaces $H_{\mathbf{v}}$, where for any nonzero lattice point $\mathbf{v}$, $H_{\mathbf{v}}$ is the set of all points that are closer to the origin than to $\mathbf{v}$. It is not necessary to consider all $\mathbf{v} \in V \setminus \{\mathbf{0}\}$ when taking this intersection. One can restrict the intersection to the so called *Voronoi relevant* vectors, which are the lattice vectors $\mathbf{v}$ such that $\mathbf{v}/2$ is the center of a facet of $\mathcal{V}$. Since the Voronoi cell of a lattice can be shown to have at most $2(2^n - 1)$ facets, $\mathcal{V}$ can be expressed as a finite intersection of at most $2(2^n - 1)$ half-spaces. Throughout this paper, we assume that the Voronoi cell of a lattice is always described by such a list of half-spaces.

The relation between the Voronoi cell and CVPP is well known, and easy to explain. In CVPP, we want to find the lattice point $\mathbf{v}$ closest to a given target vector $\mathbf{t}$. It is easy to see that this is equivalent to finding a lattice vector $\mathbf{v}$ such that $\mathbf{t} - \mathbf{v}$ belongs to the (closed) Voronoi cell of the lattice. In other words, CVP can be equivalently formulated as the problem of finding a point in the set $(\Lambda + \mathbf{t}) \cap \bar{\mathcal{V}}$, where $\bar{\mathcal{V}}$ is the topological closure of $\mathcal{V}$. The idea of using the Voronoi cell to solve CVP is not new. For example, a simple greedy algorithm for CVPP based on the knowledge of the Voronoi cell of the lattice is given in [53]. The idea behind this algorithm (called the Iterative Slicer) is to make $\mathbf{t}$ shorter and shorter by subtracting Voronoi relevant vectors from it. Notice that if $\mathbf{t} \notin H_{\mathbf{v}}$, then the length of $\mathbf{t}$ can be reduced by subtracting $\mathbf{v}$ from $\mathbf{t}$. So, as long as $\mathbf{t}$ is outside $\bar{\mathcal{V}}$, we can make further progress and find a shorter vector. Unfortunately, this simple strategy to solve CVPP using the Voronoi cell is not known to perform any better than previous algorithms. [53] only proves that the algorithm terminates after a finite number of iterations, and a close inspection of the proof reveals that the best upper bound that can be derived using the methods of [53] is of the form $n^{O(n)}$: the running time of the Iterative Slicer is bound by a volume argument, counting the number of lattice points of norm at most $\|\mathbf{t}\|$, and this can be well above $2^{O(n)}$ or even $n^{O(n)}$.

In the next two paragraphs we first sketch our new algorithm to solve CVPP using the Voronoi cell $\mathcal{V}$ in time $2^{O(n)}$, and then we show how to use the CVPP algorithm to recursively implement the preprocessing function and compute the Voronoi cell $\mathcal{V}$. Since both the preprocessing and CVPP computation take time $2^{O(n)}$, combining the two pieces gives an algorithm to solve CVP (and a host of other lattice problems, like SVP, SIVP, etc.) without preprocessing.

The CVPP algorithm We recall that the goal of CVPP with target vector $\mathbf{t}$ can be restated as the problem of finding a point $\mathbf{t}' \in \mathcal{V} \cap (\mathbf{t} + \Lambda)$ inside the Voronoi cell. (This point is also characterized as being a shortest vector in the set $\Lambda + \mathbf{t}$.) Therefore we need to find the shortest vector of the set $\Lambda + \mathbf{t}$. We follow a greedy approach similar to the Iterative Slicer of [53]. Given the list of relevant vectors, the algorithm generates a sequence of shorter and shorter vectors from $\Lambda + \mathbf{t}$, until it finds the shortest vector of the set. Unfortunately, if we use the algorithm of [53] the length of the sequence can be very large ($n^{O(n)}$). We solve this problem as follows. First we reduce the general CVPP to a special case where the target vector is guaranteed to belong to twice the Voronoi cell $2\bar{\mathcal{V}}$. This can be done very easily by a polynomial time Turing reduction. Next, using properties of the geometry of the Voronoi cell, we show that it is possible to generate a sequence of shorter and shorter vectors of $\Lambda + \mathbf{t}$, with the additional property that all the vectors are inside $2\bar{\mathcal{V}}$. This allows to bound the length of the sequence by 2^n. For each vector of the sequence the algorithm is doing $\tilde{O}(2^n)$ computations, which gives a total time complexity of $\tilde{O}(2^{2n})$. (We mention that this is an improvement over the graph traversal technique used in earlier versions of this paper with running time (at least) $\tilde{O}(2^{3n})$).

Computing the Voronoi cell We have sketched how to solve CVPP, given the Voronoi cell of the lattice. This leaves us with the problem of computing the Voronoi cell, a task typically considered even harder than CVP. To this end, we use a method of [1] to compute the Voronoi cell of a lattice Λ, making 2^n calls to a CVPP oracle for the same lattice Λ. We combine this with a standard rank reduction procedure implicit in enumeration algorithms [31, 26]. This procedure allows to solve CVPP in a lattice Λ of rank n making only $2^{0.5n}$ calls to a CVPP oracle for a certain sub-lattice Λ' with rank $n - 1$. Combining all the pieces together we obtain an algorithm that computes the Voronoi cell of a lattice Λ by building a sequence of lattices $\Lambda_1 \subset \Lambda_2 \subset \cdots \subset \Lambda_n = \Lambda$ with rank $\mathrm{rank}(\Lambda_i) = i$, and iteratively computing the Voronoi cell of Λ_{i+1}

using the previously computed Voronoi cell of Λ_i.

Organization The rest of the paper is organized as follows. In the next subsection we mention some additional related work. In Section 2 we give some background about lattices. The core algorithms are described and analyzed in Section 3, while in Section 4 we give optimized variants. Finally, Section 5 concludes with a discussion of open problems and directions for future research.

1.2 Related work

Most relevant work has already been described in the introduction. Here we mention a few more related papers. The closest vector problem with preprocessing has been investigated in several papers [36, 21, 47, 17, 7], mostly with the goal of showing that CVP is NP-hard even for fixed families of lattices, or devising polynomial time approximation algorithms (with super-polynomial time preprocessing). In summary, CVPP is NP-hard to approximate for any constant (or certain sub-polynomial) factors [7], and it can be approximated in polynomial time within a factor $\sqrt{n}$ [2], at least in its distance estimation variant. Here we use CVPP mostly as a building block to give a modular description of our CVP algorithm. We use CVPP to recursively implement the preprocessing function, and then to solve the actual CVP instance. It is an interesting open problem if a similar bootstrapping can be performed using the polynomial time CVPP approximation algorithm of [2], to yield a polynomial time solution to $\sqrt{n}$-approximate CVP.

The problem of computing the Voronoi cell of a lattice is of fundamental importance in many mathematics and communication theory applications. There are several formulations of this problem. In this paper we consider the problem of generating the list of facets ($(n-1)$-dimensional faces) of the Voronoi cell, as done also in [1, 53]. Sometime one wants to generate the list of vertices (i.e., one dimensional faces), or even a complete description including all faces in dimension 1 to $n-1$. This is done in [55, 52], but it is a much more complex problem, as in general the Voronoi cell can have as many as $n! = n^{\Omega(n)}$ vertices, so they cannot be computed in single exponential time. In [52] it is also shown that computing the number of vertices of the Voronoi cell of a lattice is $\#P$-hard.

2 Preliminaries

In this section we give some background about lattices, and the algorithmic problems studied in this paper. For a more in-depth discussion, see [40]. The d-dimensional Euclidean space is denoted $\mathbb{R}^d$. We use bold lower case letters (e.g., $\mathbf{x}$) to denote vectors, and bold upper case letters (e.g., $\mathbf{M}$) to denote matrices. The ith coordinate of $\mathbf{x}$ is denoted x_i. For a set $S \subseteq \mathbb{R}^d$, $\mathbf{x} \in \mathbb{R}^d$ and $a \in \mathbb{R}$, we let $S + \mathbf{x} = \{\mathbf{y} + \mathbf{x} \colon \mathbf{y} \in S\}$ denote the translate of S by $\mathbf{x}$, and $aS = \{a\mathbf{y} \colon \mathbf{y} \in S\}$ denote the scaling of S by a. The Euclidean norm (also known as the ℓ_2 norm) of a vector $\mathbf{x} \in \mathbb{R}^d$ is $\|\mathbf{x}\| = (\sum_i x_i^2)^{1/2}$, and the associated distance is $\mathrm{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$. The linear space spanned by a set of vectors S is denoted $\mathrm{span}(S) = \{\sum_i x_i \mathbf{s}_i : \ x_i \in \mathbb{R}, \mathbf{s}_i \in S\}$. The affine span of a set of vectors S is defined as $\mathbf{x} + \mathrm{span}(S - \mathbf{x})$ for any $\mathbf{x} \in S$, and does not depend on the choice of $\mathbf{x}$.

Lattices A d-dimensional *lattice* Λ of rank n is the set of all integer combinations

$$\Big\{ \sum_{i=1}^{n} x_i \mathbf{b}_i \colon x_i \in \mathbb{Z} \text{ for } 1 \le i \le n \Big\}$$

of n linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^d$. The set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is called a *basis* for the lattice. A basis can be represented by the matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{d \times n}$ having the basis vectors as columns. The lattice generated by $\mathbf{B}$ is denoted $\mathcal{L}(\mathbf{B})$. Notice that $\mathcal{L}(\mathbf{B}) = \{\mathbf{B}\mathbf{x} \colon \mathbf{x} \in \mathbb{Z}^n\}$, where $\mathbf{B}\mathbf{x}$ is the usual matrix-vector multiplication. A sub-lattice of $\mathcal{L}(\mathbf{B})$ is a lattice $\mathcal{L}(\mathbf{S})$ such that $\mathcal{L}(\mathbf{S}) \subseteq \mathcal{L}(\mathbf{B})$. The Gram-Schmidt orthogonalization of a basis $\mathbf{B}$ is the sequence of vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$, where $\mathbf{b}_i^*$ is the component of $\mathbf{b}_i$ orthogonal to $\mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$.

The covering radius $\mu(\Lambda)$ of a lattice Λ, is the minimum distance r such that, for any point $\mathbf{p}$ in the linear span of Λ, $\text{dist}(\mathbf{p}, \Lambda) \leq r$. The length of the shortest nonzero vector of a lattice Λ is denoted $\lambda(\Lambda)$. The dual lattice $\Lambda^\times$ of Λ is the set of all the vectors in the linear span of Λ, that have integer dot product with any element of Λ. Banaszczyk's transference theorem [10] gives a very useful relation between the primal and dual lattices, namely $\mu(\Lambda) \cdot \lambda(\Lambda^\times) \leq n$.

The following classical lattice algorithms are used in this paper. The Nearest Plane algorithm [9], on input a basis $\mathbf{B}$ and a target vector $\mathbf{t}$, finds a lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v} - \mathbf{t}\| \leq (1/2)\sqrt{\sum_i \|\mathbf{b}_i^*\|^2}$. The LLL basis reduction algorithm [34] runs in polynomial time, and on input a lattice basis, outputs a basis for the same lattice such that $\|\mathbf{b}_{i+1}^*\|^2 \geq \|\mathbf{b}_i^*\|^2/2$. (LLL reduced bases have other properties, but this is all we need here.)

Block basis reduction The paper also uses stronger basis reduction algorithms than LLL. In particular, we consider block basis reduction [23, 48] of the dual $\Lambda^\times$ of the input lattice Λ. This algorithm on input a lattice basis, outputs a basis for the same lattice such that $1/\|\mathbf{b}_n^*\| \leq k^{n/k}\lambda(\Lambda^\times)$, using polynomially many queries to an SVP oracle for k-rank lattices, with $1 < k < n$. By Banaszczyk's transference theorem [10], the inequality above gives: $\mu(\Lambda)/\|\mathbf{b}_n^*\| \leq nk^{n/k}$. For the rest of the paper we use directly the last inequality, without any further reference to the dual lattice or Banaszczyk's theorem. For a full description of these algorithms and the connection of the dual and primal lattice bases the reader is referred to [23].

Lattice problems In this paper we are mostly concerned with the CVP problem defined in the introduction. Our results give algorithms of several other lattice problems like the Shortest Vector Problem (SVP), Shortest Independent Vector Problem (SIVP), Subspace Avoiding Problem (SAP), the Generalized Closest Vector Problem (GCVP), and the Successive Minima Problem (SMP) considered in the lattice algorithms literature [12, 39]. The results for all problems other than CVP are obtained in a back-box way by reduction to CVP [39], and we refer the reader to [12, 39] for details.

For simplicity in this paper we consider only inputs to lattice problems where all the entries in the basis matrix $\mathbf{B}$ have bit size polynomial in n, i.e., $\log(\|\mathbf{B}\|) = \text{poly}(n)$. This allows to express the complexity of lattice problems simply as a function of a single parameter, the lattice rank n. All the results in this paper can be easily adapted to the general case by introducing an explicit bound $\log \|\mathbf{B}\| \leq M$ on the size of the entries, and letting the time and space complexity bound depend polynomially in M. We will use $f = \tilde{O}(g)$ when $f(n)$ is bounded by $g(n)$ up to polylogarithmic factors, i.e., $f(n) \leq \log^c g(n) \cdot g(n)$ for some constant c and all sufficiently large n.

Voronoi cells The (open) Voronoi cell of a lattice Λ is the set

$$\mathcal{V}(\Lambda) = \{\mathbf{x} \in \mathbb{R}^n : \forall \mathbf{v} \in \Lambda \setminus \{\mathbf{0}\}. \|\mathbf{x}\| < \|\mathbf{x} - \mathbf{v}\|\}$$

of all points that are closer to the origin than to any other lattice point. We also define the closed Voronoi cell $\bar{\mathcal{V}}$ as the topological closure of $\mathcal{V}$. We omit Λ, and simply write $\mathcal{V}$, when the lattice is clear from the context. The Voronoi cell of a lattice point $\mathbf{v} \in \Lambda$ is defined similarly, and equals $\mathbf{v} + \mathcal{V}$. For any (lattice) point $\mathbf{v}$, define the half-space

$$H_{\mathbf{v}} = \{\mathbf{x} \colon \|\mathbf{x}\| < \|\mathbf{x} - \mathbf{v}\|\}.$$

Clearly, $\mathcal{V}$ is the intersection of $H_{\mathbf{v}}$ for all $\mathbf{v} \in \Lambda \setminus \{\mathbf{0}\}$. However, it is not necessary to consider all $\mathbf{v}$. The minimal set of lattice vectors V such that $\mathcal{V} = \bigcap_{\mathbf{v} \in V} H_{\mathbf{v}}$ is called the set of *Voronoi relevant* vectors. The Voronoi cell $\mathcal{V}$ is a polytope, and the Voronoi relevant vectors are precisely the centers of the $((n-1)$ dimensional) facets of $2\mathcal{V}$. We give two useful observations for the Voronoi cell.

Observation 2.1 *Let $\bar{\mathcal{V}}$ the closed Voronoi cell of a lattice Λ and $\mathbf{p}$ a point on the facet of $\bar{\mathcal{V}}$ defined by the relevant vector $\mathbf{v}$. Then $\mathbf{p} - \mathbf{v} \in \bar{\mathcal{V}}$.*

Proof: Notice that $\mathbf{p}$ is on the intersection of the closed Voronoi cells $\bar{\mathcal{V}}$ and $\bar{\mathcal{V}} + \mathbf{v}$ (the closed Voronoi cell centered around $\mathbf{v}$). Therefore $\mathbf{p} \in \bar{\mathcal{V}} + \mathbf{v}$ and $\mathbf{p} - \mathbf{v} \in \bar{\mathcal{V}}$. ■

The following observation connects the Voronoi cell, solving CVP and finding shortest vectors on cosets of the form $\Lambda + \mathbf{t}$. These problems will be used interchangeably throughout the paper.

Observation 2.2 *Let Λ a lattice, $\mathcal{V}$ its Voronoi cell and $\mathbf{t}$, $\mathbf{t}_S$ vectors in the span of Λ. The following are equivalent:*

1. *$\mathbf{t}_S$ is a shortest vector of the coset $\Lambda + \mathbf{t}$*
2. *$\mathbf{t}_S$ is a vector in $\Lambda + \mathbf{t} \cap \bar{\mathcal{V}}$*
3. *The vector $\mathbf{c} = \mathbf{t} - \mathbf{t}_S$ is a closest lattice vector to $\mathbf{t}$.*

Proof: $1 \to 2$: Notice that for all $\mathbf{v} \in \Lambda$, $\mathbf{t}_S - \mathbf{v} \in \Lambda + \mathbf{t}$. Therefore from 1 we get that $\forall \mathbf{v} \in \Lambda : \|\mathbf{t}_S\| \leq \|\mathbf{t}_S - \mathbf{v}\|$ which is exactly the definition of the Voronoi cell, so $\mathbf{t}_S \in \bar{\mathcal{V}}$.
$2 \to 3$: If $\mathbf{t}_S \in \Lambda + \mathbf{t} \cap \bar{\mathcal{V}}$, then $\mathbf{c} = \mathbf{t} - \mathbf{t}_S$ is a lattice vector and $\mathbf{t} \in \bar{\mathcal{V}} + \mathbf{c}$. By definition of the Voronoi cell, this gives that $\mathbf{c}$ is a closest vector point of $\mathbf{t}$.
$3 \to 1$: If $\mathbf{c}$ is a closest vector point, then $\forall \mathbf{v} \in \Lambda : \|\mathbf{t} - \mathbf{c}\| \leq \|\mathbf{t} - \mathbf{v}\|$. Notice that the set $\cup_{\mathbf{v} \in \Lambda} \{\mathbf{t} - \mathbf{v}\}$ is exactly the coset $\Lambda + \mathbf{t}$, so $\mathbf{t}_S = \mathbf{t} - \mathbf{c}$ is a shortest vector of the coset $\Lambda + \mathbf{t}$. ■

Relevant vectors For the rest of the paper we use the set of relevant vectors to describe the Voronoi cell. In order to identify these vectors we use Voronoi's classical theorem:

Theorem 2.3 *Voronoi, see [18]: Let Λ a lattice and $\mathbf{v}$ a non-zero lattice vector. Then $\mathbf{v}$ is Voronoi relevant if and only if the pair $\pm\mathbf{v}$ are the unique shortest vectors of the coset $2\Lambda + \mathbf{v}$.*

In this paper we use a number of techniques and observations derived directly from Voronoi's work. One of these techniques is partitioning the lattice Λ to 2^n cosets modulo 2Λ.

Observation 2.4 *Let $\mathbf{B}$ a basis for an n-rank lattice Λ. Λ can be partitioned to exactly 2^n cosets of the form $C_{\mathbf{B},\mathbf{p}} = 2\Lambda + \mathbf{B} \cdot \mathbf{p}$ with $\mathbf{p} \in \{0,1\}^n$.*

Combining Voronoi's theorem with the observation above gives the following corollary.

Corollary 2.5 *Let $\mathbf{B}$ a basis for an n-rank lattice Λ and the cosets $C_{\mathbf{B},\mathbf{p}}$ as defined above. If $\mathbf{s_p}$ is a shortest vector for $C_{\mathbf{B},\mathbf{p}}$ then the set $R = \bigcup_{\mathbf{p} \in \{0,1\}^n \setminus \{\mathbf{0}\}} \{\pm \mathbf{s_p}\}$ contains all the relevant vectors.*

Proof: Let $\mathbf{v}$ a relevant vector of Λ. By Observation 2.4 there exist a $\mathbf{p} \in \{0,1\}^n$ such that $\mathbf{v} \in C_{\mathbf{B},\mathbf{p}}$. If $\mathbf{p} \neq \mathbf{0}$ by Voronoi's theorem $\pm\mathbf{r}$ are the unique shortest vectors of $C_{\mathbf{B},\mathbf{p}}$, so they are exactly the vectors $\pm\mathbf{s_p}$ included in R. If $\mathbf{p} = \mathbf{0}$, the shortest vector of $C_{\mathbf{B},\mathbf{0}}$ is $\mathbf{0}$ and $\mathbf{r}$ cannot be a relevant vector. ■

3 The basic algorithm

In this section we describe and analyze a $\tilde{O}(2^{3.5n})$ time and $\tilde{O}(2^n)$ space algorithm for CVP and related lattice problems. This is improved to a $\tilde{O}(2^{2n})$ time and $\tilde{O}(2^n)$ space algorithm in Section 4. We describe the slower algorithm first as it allows for a more modular description and illustrates the connections with previous work. The CVP algorithm presented in this section has three components:

1. An $\tilde{O}(2^{2n})$-time algorithm to solve the *closest vector problem with preprocessing* (CVPP), where the output of the preprocessing function is the Voronoi cell of the input lattice, described as the intersection of half-spaces.
2. A *rank reduction* procedure that solves CVP on a k-rank lattice $\Lambda_k = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$ with $2^{0.5k}$ calls to a CVP oracle for $\Lambda_{k-1} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})$, where $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ an LLL reduced basis for Λ_k.
3. A procedure for computing the *Voronoi cell* of a lattice Λ by 2^n calls to a CVP oracle for Λ and 2^{2n} polynomial time computations.

These three components are described and analyzed in Subsections 3.1, 3.2, 3.3. In Subsection 3.4 we show how these components can be combined together into an algorithm for CVP with running time $\tilde{O}(2^{3.5n})$.

Notice that the rank reduction procedure immediately gives a recursive algorithm to solve CVP in arbitrary lattices. However, the obvious way to turn the rank reduction procedure into a recursive program leads to an algorithm with $2^{O(n^2)}$ running time, because each time the rank of the input lattice is reduced by 1, the number of recursive invocations gets multiplied by $2^{O(n)}$. We use the CVPP and Voronoi cell computation algorithms to give a more efficient transformation. The idea is to compute the Voronoi cells of all sub-lattices $\Lambda_k = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$ sequentially, where $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is the lattice basis produced by the rank reduction procedure. Each Voronoi cell $\mathcal{V}(\Lambda_k)$ is computed by (rank) reduction to $2^{O(k)}$ CVP computations in the lattice Λ_{k-1}. In turn, these CVP computations are performed using the CVPP algorithm and the previously computed Voronoi cell $\mathcal{V}(\Lambda_{k-1})$. Once $\mathcal{V}(\Lambda_n)$ has been computed, the input CVP instance can be solved using CVPP. We show that:

Theorem 3.1 *There is a deterministic $\tilde{O}(2^{3.5n})$-time algorithm to solve CVP.*

The proof of the theorem is postponed to Subsection 3.4, after we have described and analyzed the building blocks of the algorithm. The pseudocode of the algorithm is given in Algorithms 1, 2, 3, 4.

3.1 CVP with preprocessing

We give an algorithm that on input an n-rank lattice Λ, a list V of the Voronoi relevant vectors of Λ and a target vector $\mathbf{t}$, computes a lattice point closest to $\mathbf{t}$ in time $\tilde{O}(2^{2n})$. We proceed as follows:

1. We present $\text{CVPP}_{2\bar{\mathcal{V}}}$, a $\tilde{O}(2^{2n})$ algorithm for the special case where $\mathbf{t} \in 2\bar{\mathcal{V}}$,
2. then we solve the general CVPP problem with polynomially many calls to $\text{CVPP}_{2\bar{\mathcal{V}}}$.

For the following we assume without loss of generality that $\mathbf{t}$ belongs to the linear span of Λ, otherwise, we simply project $\mathbf{t}$ orthogonally to that subspace.

First we present the $\text{CVPP}_{2\bar{\mathcal{V}}}$ algorithm. Recall that we can solve CVP for target $\mathbf{t}$, by first computing a point $\mathbf{t}_S \in \bar{\mathcal{V}} \cap (\Lambda + \mathbf{t})$, (a shortest point of the coset $\Lambda + \mathbf{t}$,) and then output $\mathbf{c} = \mathbf{t} - \mathbf{t}_S$ (see Observation 2.2). The $\text{CVPP}_{2\bar{\mathcal{V}}}$ algorithm finds such a vector $\mathbf{t}_S$ following a technique similar to the Iterative Slicer of [53]. Given a target $\mathbf{t}$, and a list with the relevant points V of a lattice Λ, the algorithm generates a sequence $\mathbf{t} = \mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_S$ of shorter and shorter points all in the coset $\Lambda + \mathbf{t}$, until it finds a shortest vector $\mathbf{t}_S$ in $\Lambda + \mathbf{t}$. Recall that the (closed) Voronoi cell is the intersection of halfspaces $H_{\mathbf{v}} = \{\mathbf{x} : \|\mathbf{x}\| \leq \|\mathbf{v} - \mathbf{x}\|\}$, where $\mathbf{v}$ ranges over the relevant vectors. If $\mathbf{t}_i$ is not in $\bar{\mathcal{V}}$, there exist at least one halfspace $H_{\mathbf{v}_i}$ such that $\mathbf{t} \notin H_{\mathbf{v}_i}$. Now if we set $\mathbf{t}_{i+1} = \mathbf{t}_i - \mathbf{v}_i$, we obtain a vector in $\Lambda + \mathbf{t}_i$ such that $\|\mathbf{t}_{i+1}\| < \|\mathbf{t}_i\|$. Applying this procedure iteratively we get a sequence of vectors $\mathbf{t}_i$ with decreasing norm and eventually a shortest vector $\mathbf{t}_S$. The main difficulty in the analysis of the algorithm is bounding the length of this sequence. Unfortunately, the best known bound for the original algorithm, given in [53], can be arbitrarily large (even when $\mathbf{t}_i \in 2\bar{\mathcal{V}}$).

The novelty of $\text{CVPP}_{2\bar{\mathcal{V}}}$ is a new selection strategy for the relevant vectors $\mathbf{v}_i$ subtracted from the target $\mathbf{t}_i$ of each step. The new selection strategy works as follows. It scales the Voronoi cell by a factor $k > 1$ such that $\mathbf{t}_i$ is exactly on the boundary of $k\bar{\mathcal{V}}$ (k is a real number). In particular $\mathbf{t}_i$ is on a facet corresponding to some scaled relevant vector $k\mathbf{v}_i$. The strategy selects this relevant vector $\mathbf{v}_i$ and sets $\mathbf{t}_{i+1} = \mathbf{t}_i - \mathbf{v}_i$. (The value of k and corresponding vector $\mathbf{v}_i$ can be computed by maximizing the quantity $2 \max_{\mathbf{v} \in V}(\langle \mathbf{t}, \mathbf{v} \rangle / \|\mathbf{v}\|^2)$.) The following lemma shows that given a target vector $\mathbf{t}$ in $2\bar{\mathcal{V}}$ the selection strategy described above guarantees termination within 2^n iterations.

Lemma 3.2 *Given a list V of the relevant vectors of lattice Λ and a target point $\mathbf{t} \in 2\bar{\mathcal{V}}$ the $\text{CVPP}_{2\bar{\mathcal{V}}}$ algorithm (Algorithm 1) finds a closest vector to $\mathbf{t}$ in time $\tilde{O}(2^{2n})$, where n is the rank of the lattice.*

Proof: First we prove that if $\mathbf{t}_i$ is in $2\bar{\mathcal{V}}$ (but not in $\bar{\mathcal{V}}$) and $\mathbf{v}_i$ is the relevant vector selected by $\text{CVPP}_{2\bar{\mathcal{V}}}$, the vector $\mathbf{t}_{i+1} = \mathbf{t}_i - \mathbf{v}_i$ has the following two properties:

1. $\mathbf{t}_{i+1}$ is in $2\bar{\mathcal{V}}$.

Algorithm 1 Single exponential time for CVPP

function CVPP($\mathbf{t}, \mathbf{B}, V$)
  $\mathbf{t} \leftarrow$ Project $\mathbf{t}$ on the linear span of $\mathbf{B}$
  Choose smallest $p \in \mathbb{Z}$ that $\mathbf{t} \in 2^p\bar{\mathcal{V}}$
  $\mathbf{t}_p \leftarrow \mathbf{t}$
  for $i = p$ **downto** 1 **do**
    $\mathbf{t}_{i-1} \leftarrow \mathbf{t}_i - \text{CVPP}_{2\bar{\mathcal{V}}}(\mathbf{t}_i, 2^{i-1}V)$
  return $\mathbf{t} - \mathbf{t}_0$

function $\text{CVPP}_{2\bar{\mathcal{V}}}(\mathbf{t}, V)$
  $i \leftarrow 0$
  $\mathbf{t}_0 \leftarrow \mathbf{t}$
  while $\mathbf{t}_i \notin \bar{\mathcal{V}}$ **do**
    Find $\mathbf{v}_i \in V$ that maximizes $\langle \mathbf{t}_i, \mathbf{v}_i \rangle / \|\mathbf{v}_i\|^2$
    $\mathbf{t}_{i+1} \leftarrow \mathbf{t}_i - \mathbf{v}_i$
    $i \leftarrow i + 1$
  return $\mathbf{t}_0 - \mathbf{t}_i$

2. $\mathbf{t}_{i+1}$ is shorter than $\mathbf{t}_i$.

Recall that $\text{CVPP}_{2\bar{\mathcal{V}}}$ computes a scaling factor $1 < k \leq 2$ (by assumption $\mathbf{t}_i$ is in $2\bar{\mathcal{V}}$) and a relevant vector $\mathbf{v}_i$, such that $\mathbf{t}_i$ is exactly on the boundary of $k\bar{\mathcal{V}}$ and in particular on the facet corresponding to the scaled relevant vector $k\mathbf{v}_i$. In order to prove the first property, notice that if $\mathbf{t}_i$ is on the facet corresponding to $k\mathbf{v}_i$ we have that the point $\mathbf{t}_i - k\mathbf{v}_i$ is also on $k\bar{\mathcal{V}}$, by Observation 2.1 applied to the Voronoi cell $k\bar{\mathcal{V}}$ of the lattice $k \cdot \Lambda$. Now notice that if both $\mathbf{t}_i$ and $\mathbf{t}_i - k\mathbf{v}_i$ are in $k\bar{\mathcal{V}}$, then by convexity of $k\bar{\mathcal{V}}$, $\mathbf{t}_{i+1} = \mathbf{t}_i - \mathbf{v}_i$ is also in $k\bar{\mathcal{V}}$ and consequently in $2\bar{\mathcal{V}}$. For the second property, given that $\mathbf{t}_i$ is on the facet of $k\bar{\mathcal{V}}$ corresponding to $k\mathbf{v}_i$, we have $\|\mathbf{t}_i\| = \|\mathbf{t}_i - k\mathbf{v}_i\|$, which gives $2\langle \mathbf{t}_i, \mathbf{v}_i \rangle = k\|\mathbf{v}_i\|^2$, and consequently: $\|\mathbf{t}_{i+1}\|^2 = \|\mathbf{t}_i\|^2 + (1-k)\|\mathbf{v}_i\|^2 < \|\mathbf{t}_i\|^2$ (recall that $k > 1$).

Now consider the sequence $\mathbf{t} = \mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_S$ generated by $\text{CVPP}_{2\bar{\mathcal{V}}}$, where $\mathbf{t}$ is an input vector in $2\bar{\mathcal{V}}$ and $\mathbf{t}_S$ is a vector in $(\Lambda + \mathbf{t}) \cap \bar{\mathcal{V}}$. Using induction and the two properties proven above we conclude that:

1. All the vectors in the sequence are in $2\bar{\mathcal{V}}$.
2. The norms of the vectors of the sequence are strictly decreasing. In particular the norms of $\|\mathbf{t}_i\|$ are distinct.

Finally notice that all the vectors in the sequence are generated by subtracting lattice vectors from $\mathbf{t}$, so they are in $\Lambda + \mathbf{t}$. Now, if all the vectors $\mathbf{t}_i$ were in the *interior* of the Voronoi cell $2\mathcal{V}$ we could immediately conclude that the length of the sequence is at most 2^n, because $2\mathcal{V}$ contains at most 2^n points from any coset $\Lambda + \mathbf{t}$. Since $2\bar{\mathcal{V}}$ may contain more than 2^n points a more careful analysis is needed.

Given a basis $\mathbf{B}$, partition $\Lambda + \mathbf{t}$ into 2^n cosets of the form:

$$C_{\mathbf{B},\mathbf{p},\mathbf{t}} = 2\Lambda + \mathbf{B} \cdot \mathbf{p} + \mathbf{t}, \text{ with: } \mathbf{p} \in \{0,1\}^n.$$

Every point in the sequence belongs to one of these cosets. We prove that the length of the sequence is at most 2^n by contradiction. Assume that the sequence has more than 2^n vectors, then at least two vectors $\mathbf{t}_i, \mathbf{t}_j$, $i \neq j$ belong to the same coset $C_{\mathbf{B},\mathbf{p},\mathbf{t}}$. By the first property the vectors are in $2\bar{\mathcal{V}}$ and by Observation 2.2 they are shortest vectors of $C_{\mathbf{B},\mathbf{p},\mathbf{t}}$, so $\|\mathbf{t}_i\| = \|\mathbf{t}_j\|$. This leads to a contradiction because the vectors of the sequence have distinct norms. Therefore the sequence length is at most 2^n.

Concluding, the $\text{CVPP}_{2\bar{\mathcal{V}}}$ algorithm finds the required vector $\mathbf{t}_S$, and solves CVPP, after at most 2^n iterations. Recall that each iteration examines all the relevant vectors in order to find an appropriate $\mathbf{v}_i$, therefore the complexity of each such iteration is $\text{poly}(n)|V| \leq \tilde{O}(2^n)$, which gives a total complexity of $\tilde{O}(2^{2n})$. ■

The next lemma shows how to solve CVP for any target vector.

Theorem 3.3 *On input a list V of the relevant vectors of a lattice Λ and a target point $\mathbf{t}$, the* CVPP *algorithm (Algorithm 1) finds a closest vector to $\mathbf{t}$ in $\log \|\mathbf{t}\| \cdot \tilde{O}(2^{2n})$ time, where n is the rank of the input lattice.*

Proof: The CVPP algorithm computes a factor k such that $\mathbf{t}$ is exactly on the boundary of $k\bar{\mathcal{V}}$. Recall that k can be computed as $2\max_{\mathbf{v} \in V}(\langle \mathbf{t}, \mathbf{v} \rangle / \|\mathbf{v}\|^2)$, so $k < 2\|\mathbf{t}\|$. Then it computes the smallest integer p such

that $2^p > k$, so that $\mathbf{t} \in 2^p\bar{\mathcal{V}}$. The algorithm sets $\mathbf{t}_p = \mathbf{t}$ and computes a sequence of vectors $\mathbf{t}_p, \mathbf{t}_{p-1}, \ldots, \mathbf{t}_0$ in $\Lambda + \mathbf{t}$ such that each $\mathbf{t}_i$ is in $2^i\bar{\mathcal{V}}$. The algorithm obtains $\mathbf{t}_i$ from $\mathbf{t}_{i+1}$ with a call to $\text{CVPP}_{2\bar{\mathcal{V}}}$. Notice that p is bounded by $\log \|\mathbf{t}\| + 2$ and each $\text{CVPP}_{2\bar{\mathcal{V}}}$ computation costs $\tilde{O}(2^{2n})$ by Lemma 3.2. Therefore the total complexity is $\log \|\mathbf{t}\| \cdot \tilde{O}(2^{2n})$. The vector $\mathbf{t}_0$ is in $\bar{\mathcal{V}}$ and consequently a shortest vector in $\Lambda + \mathbf{t}$. As a result the algorithm can simply output $\mathbf{t} - \mathbf{t}_0$ which is a closest vector to $\mathbf{t}$.

Finally, we show how to obtain $\mathbf{t}_i$ from $\mathbf{t}_{i+1}$ with a call to $\text{CVPP}_{2\bar{\mathcal{V}}}$. The algorithm scales the lattice Λ and the corresponding relevant vectors V by 2^i and runs $\text{CVPP}_{2\bar{\mathcal{V}}}$ on $2^i\Lambda$, with target vector $\mathbf{t}_{i+1}$. Notice that $\mathbf{t}_{i+1}$ is in $2 \cdot 2^i\mathcal{V}$, therefore it is a valid input for $\text{CVPP}_{2\bar{\mathcal{V}}}$ on the lattice $2^i\Lambda$. Finally, given the closest vector $\mathbf{c}_{i+1}$ the algorithm simply sets $\mathbf{t}_i = \mathbf{t}_{i+1} - \mathbf{c}_{i+1}$. The new vector $\mathbf{t}_i$ is in $\Lambda + \mathbf{t}$ and also in $2^i\mathcal{V}$ by Observation 2.2. ■

3.2 Rank reduction

The core of this component is a procedure that given a basis $\mathbf{B}_k = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ for a k-rank lattice Λ_k and an integer H such that $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq H$, solves a CVP instance on Λ_k, with $2H + 1$ calls to CVP on the $(k-1)$-rank sub-lattice $\Lambda_{k-1} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})$ (where $\mu(\Lambda_k)$ is the covering radius of Λ_k and $\mathbf{b}_k^*$ the component of $\mathbf{b}_k$ orthogonal to Λ_{k-1}). The efficiency of this transformation depends on the bound H of the ratio $\mu(\Lambda_k)/\|\mathbf{b}_k^*\|$ and therefore on the quality of the input basis. To address this issue the rank reduction consists of two steps:

1. A polynomial time preprocessing step that outputs a basis $\mathbf{B}$ for the input lattice such that, the ratio $\mu(\Lambda_k)/\|\mathbf{b}_k^*\|$ is bounded by $2^{0.5k}$, for every $k = 2, \ldots, n$. We show that this can be achieved by simply running the LLL[34] algorithm. This step is done only once at the beginning of the algorithm and all further computations use the higher quality basis.

2. The actual rank reduction procedure that given an integer H such that $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq H$ solves a CVP instance on Λ_k with $2H + 1$ calls to CVP on Λ_{k-1}.

Combining the two steps gives a reduction from a CVP in Λ_k to $2^{0.5k}$ CVPs in Λ_{k-1}. The two procedures are analyzed in the following lemmas.

Lemma 3.4 *The LLL algorithm, on input a basis for an n-rank lattice Λ, outputs a new basis $\mathbf{B}$ for the same lattice with the following property: for every sub lattice $\Lambda_k = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$, $k \leq n$, we have $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq 2^{0.5k}$.*

Proof: Notice that:

- If $\mathbf{B}$ is a basis for Λ, then $\mu(\Lambda) \leq \rho = \frac{1}{2}\sqrt{\sum_{i=1}^n \|\mathbf{b}_i^*\|^2}$, as the set $\{\sum \mathbf{x}_i\mathbf{b}_i^* : -\frac{1}{2} < x_i \leq \frac{1}{2}\}$ is a fundamental region of Λ and it is contained in a ball with radius ρ centered at $\mathbf{0}$.
- If $\mathbf{B}$ is LLL reduced then $\|\mathbf{b}_{i+1}^*\|^2 \geq 2^{-1}\|\mathbf{b}_i^*\|^2$ and consequently $\|\mathbf{b}_k^*\|^2 \geq 2^{i-k}\|\mathbf{b}_i^*\|^2$. Thus:

$$\rho = \frac{1}{2}\sqrt{\sum_{i=1}^n \|\mathbf{b}_i^*\|^2} \leq \frac{1}{2}\sqrt{\sum_{i=1}^n 2^{n-i}\|\mathbf{b}_n^*\|^2} = \frac{1}{2}\sqrt{2^n - 1}\|\mathbf{b}_n^*\| \leq 2^{0.5n}\|\mathbf{b}_n^*\|.$$

From the above inequalities we conclude that if $\mathbf{B}$ is an LLL reduced basis for Λ then $\mu(\Lambda)/\|\mathbf{b}_n^*\| \leq 2^{0.5n}$. Finally, notice that if $\mathbf{B}$ is LLL reduced then, for all $k \leq n$, the basis $\mathbf{B}_k = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ is also LLL reduced and consequently $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq 2^{0.5k}$. ■

Lemma 3.5 *On input a basis $\mathbf{B}_k = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ for a k-rank lattice Λ_k and an integer H such that $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq H$, the algorithm* RankReduceCVP *(Algorithm 2) solves a* CVP *instance on Λ_k with $2H + 1$ calls to* CVP *on the $(k-1)$-rank sub-lattice $\Lambda_{k-1} = \mathcal{L}\{\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}\}$.*

Algorithm 2 Rank reduction procedure for CVP

function RANKREDUCECVP($\mathbf{t}, \mathbf{B}_k, V_{k-1}, H$)
  $h_t \leftarrow \langle \mathbf{t}, \mathbf{b}_k^* \rangle / \langle \mathbf{b}_k^*, \mathbf{b}_k^* \rangle$
  for all h **that:** $|h - h_t| < H$ **do**
    $\mathbf{v} \leftarrow \text{CVPP}(\mathbf{t} - h\mathbf{b}_k, V_{k-1}) + h\mathbf{b}_k$
    $\mathbf{c} \leftarrow$ The closest $\mathbf{v}$ to $\mathbf{t}$ found so far
  return c

Proof: Let $\mathbf{t}$ be the target vector of CVP on Λ_k. We can assume without loss of generality that $\mathbf{t}$ belongs to the linear span of Λ_k, otherwise, we simply project $\mathbf{t}$ orthogonally to that subspace. Partition the lattice Λ_k into layers of the form $h\mathbf{b}_k + \Lambda_{k-1}$, where $h \in \mathbb{Z}$. Notice that:

- by the definition of the covering radius, the closest vector to $\mathbf{t}$ has distance at most $\mu(\Lambda_k)$.
- the distance of all lattice points in the layer $h\mathbf{b}_k + \Lambda_{k-1}$ from $\mathbf{t}$ is at least $|h - h_t| \cdot \|\mathbf{b}_k^*\|$, where $h_t = \langle \mathbf{t}, \mathbf{b}_k^* \rangle / \langle \mathbf{b}_k^*, \mathbf{b}_k^* \rangle$, because this is the distance between $\mathbf{t}$ and the entire affine space generated by the layer.

It follows from the above observations that the lattice points in Λ_k closest to $\mathbf{t}$ belong to layers $h\mathbf{b}_k + \Lambda_{k-1}$ such that $|h - h_t| \leq \mu(\Lambda_k)/\|\mathbf{b}_k^*\|$. Recall that the algorithm is given an H such that $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq H$. Therefore, in order to find a lattice point closest to $\mathbf{t}$ the algorithm iterates over all $2H + 1$ layers with h such that $|h - h_t| \leq H$, and for each of them finds a point in $h\mathbf{b}_k + \Lambda_{k-1}$ closest to $\mathbf{t}$. Notice that this is equivalent to finding a point in Λ_{k-1} closest to $\mathbf{t} - h\mathbf{b}_k$, i.e., a CVP computation in Λ_{k-1}. A lattice point closest to $\mathbf{t}$ is found selecting the best solution across all layers. ■

3.3 Computing the Voronoi cell

This component computes the relevant vectors of a lattice Λ by $2^n - 1$ calls to CVP, all for the same lattice, plus 2^{2n} polynomial time computations. In order to achieve this we use a variant of the RelevantVectors algorithm of [1]. Our variant computes the relevant vectors in two steps:

1. First it runs the FINDRELEVANT procedure that, given a lattice Λ, outputs a list V of size $2(2^n - 1)$ that contains all the relevant vectors, with the help of $2^n - 1$ CVP queries. Notice that the list might contain non relevant vectors, but all vectors in the list are lattice points.
2. Then it feeds the output list V to the REMOVENONRELEVANT procedure. REMOVENONRELEVANT discards the non-relevant points from a list of *lattice points* V that contains all the relevant vectors in time $\text{poly}(n)|V|^2$. Notice that $|V| < 2^{n+1}$, therefore this step takes $\tilde{O}(2^{2n})$ time.

The pseudocode is given in Algorithm 3. We go on to describe and analyze the two procedures.

Lemma 3.6 *Given a basis* $\mathbf{B}$ *for an* n*-rank lattice, the* FINDRELEVANT *procedure (Algorithm 3) computes a set of at most* 2^{n+1} *lattice vectors that contains all the relevant vectors of* $\Lambda = \mathcal{L}(\mathbf{B})$ *using* $2^n - 1$ CVP *queries for the same lattice.*

Proof: On input a basis $\mathbf{B}$ for a n-rank lattice Λ, the FindRelevant procedure iterates over all $\mathbf{p} \in \{0,1\}^n \setminus \{\mathbf{0}\}$, and for each of them, does the following:

1. Find a closest vector $\mathbf{c}$ of Λ to $\mathbf{t} = -\mathbf{B} \cdot \mathbf{p}/2$.
2. Include the pair $\pm\mathbf{s_p} = \pm 2(\mathbf{c} - \mathbf{t})$ to the list V.

Algorithm 3 Voronoi cell computation

function FINDRELEVANT($\mathbf{B}_k, V_{k-1}, H$)
  for p in $\{\{0,1\}^k \setminus \mathbf{0}\}$ **do**
    $\mathbf{t} \leftarrow -\mathbf{B}_k \cdot \mathbf{p}/2$
    $\mathbf{c} \leftarrow$ RankReduceCVP$(\mathbf{t}, \mathbf{B}_k, V_{k-1}, H)$
    $V_k \leftarrow V_k \cup \{\pm 2(\mathbf{c}-\mathbf{t})\}$
  return V_k

function REMOVENONRELEVANT(V_k)
  for $\mathbf{v}_i \in V_k$ **do**
    for $\mathbf{v}_j \in V_k : \mathbf{v}_j \neq \mathbf{v}_i$ **do**
      if $\|\mathbf{v}_j - \mathbf{v}_i/2\| \leq \|\mathbf{v}_i/2\|$ **then**
        $V_k \leftarrow V_k \setminus \{\mathbf{v}_i\}$
  return V_k

It is clear that the above procedure generates a list of $2(2^n-1)$ vectors using 2^n-1 calls to a CVP oracle. Now in order to show that it contains all the relevant vectors, notice that if $\mathbf{c} \in \Lambda$ is a vector closest to $\mathbf{t} = -\mathbf{Bp}/2$, then by Observation 2.2, $2(\mathbf{c}-\mathbf{t})$ is a shortest vector of the coset $C_{\mathbf{B},\mathbf{p}} = 2\Lambda + \mathbf{B}\cdot\mathbf{p}$. Therefore the list V contains a pair of shortest vectors for all the cosets $C_{\mathbf{B},\mathbf{p}}$, with $\mathbf{p} \in \{0,1\}^n \setminus \{\mathbf{0}\}$, and by Corollary 2.5 all the relevant vectors. ■

Lemma 3.7 *Given a superset of the relevant vectors V, the* REMOVENONRELEVANT *procedure (Algorithm 3) outputs a set that contains only the relevant vectors in time poly$(n)|V|^2$.*

Proof: Given a list of vectors $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$, REMOVENONRELEVANT iterates over the vectors $\mathbf{v}_i$ and if there exist $\mathbf{v}_j \neq \mathbf{v}_i \in V$ such that $\|\mathbf{v}_j - \mathbf{v}_i/2\| \leq \|\mathbf{v}_i/2\|$, removes $\mathbf{v}_i$ from the list. Finally it outputs the list V without the discarded vectors. It is clear that the algorithm runs in time poly(n)$|V|^2$. For correctness we need the following two observations:

- If $\mathbf{v}_i$ is a relevant vector, for every $\mathbf{u} \neq \mathbf{v}_i \in \Lambda$, $\|\mathbf{u} - \mathbf{v}_i/2\| > \|\mathbf{v}_i/2\|$. For the proof, notice that if $\mathbf{v}_i$ is a relevant vector, $\pm\mathbf{v}_i$ should be unique shortest vectors of $2\Lambda + \mathbf{v}_i$. The vector $2\mathbf{u} - \mathbf{v}_i$ is in $2\Lambda + \mathbf{v}_i$ therefore $\|2\mathbf{u} - \mathbf{v}_i\| > \|\mathbf{v}_i\|$ which gives the required inequality.
- If $\mathbf{v}_i$ is not relevant, there exist a relevant vector $\mathbf{r}$ such that $\|\mathbf{r} - \mathbf{v}_i/2\| \leq \|\mathbf{v}_i/2\|$. We prove the statement by contradiction. Assume that $\mathbf{v}_i$ is not relevant and that for all relevant vectors $\mathbf{r}$, $\|\mathbf{r} - \mathbf{v}_i/2\| > \|\mathbf{v}_i/2\|$. Equivalently the vector $\mathbf{v}_i/2$ is in the interior of $\mathcal{V}$ and $\mathbf{0}$ is its *unique* closest vector. This is a contradiction because $\mathbf{0}$ and $\mathbf{v}$ have the same distance from $\mathbf{v}/2$.

Now, by the first observation none of the relevant vectors is going to be removed from the list. Therefore the second observation gives that all the non-relevant vectors are going to be discarded (because all the relevant vectors remain in the list). ■

3.4 Combining the blocks

In this section we combine the three blocks to acquire a $\tilde{O}(2^{3.5n})$ for computing the Voronoi cell. First we describe COMPUTEVCELL, an algorithm that computes the Voronoi cell of $\Lambda_n = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ with the help of CVPP instances on the lower rank sub-lattice Λ_{n-1}.

Lemma 3.8 *Given a basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ for an n-rank lattice Λ_n, an integer H such that $\mu(\Lambda_n)/\|\mathbf{b}_n^*\| \leq H$ and the Voronoi cell of $\Lambda_{n-1} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{n-1})$ the* COMPUTEVCELL *procedure (Algorithm 4) deterministically computes the Voronoi cell of Λ_n in time $H \cdot \tilde{O}(2^{3n})$.*

Proof: In order to compute the Voronoi cell of Λ_n it is enough to call the procedure FINDRELEVANT followed by REMOVENONRELEVANT (see Subsection 3.3). However, notice that FINDRELEVANT requires access to a CVP oracle for Λ_n. COMPUTEVCELL instantiates a CVP oracle for Λ_n as follows. For each query to the CVP oracle it calls RANKREDUCECVP to reduce the given CVP instance on Λ_n to $2H+1$ CVP instances on Λ_{n-1} (Lemma 3.5). Then it solves every such instance with a call to the CVPP algorithm for Λ_{n-1} (notice that the Voronoi cell of Λ_{n-1} is given as input). Consequently we have the following derivations:

Algorithm 4 The basic algorithm

function BasicVoronoiCell($\mathbf{B}$)
  $\mathbf{B} \leftarrow$ Preprocess($\mathbf{B}$)
  $V_1 = \{(\mathbf{b}_1, -\mathbf{b}_1)\}$
  for $k = 2$ **to** n **do**
    $\mathbf{B}_k \leftarrow [\mathbf{b}_1, \ldots, \mathbf{b}_k]$
    $V_k \leftarrow$ ComputeVCell($\mathbf{B}_k, V_{k-1}, 2^{0.5k}$)
  return V_n

function Preprocess($\mathbf{B}$)
  return LLL($\mathbf{B}$)

function ComputeVCell($\mathbf{B}_k, V_{k-1}, H$)
  $V_k \leftarrow$ FindRelevant($\mathbf{B}_k, V_{k-1}, H$)
  $V_k \leftarrow$ RemoveNonRelevant(V_k)
  return V_k

1. FindRelevant computes a superset of the relevant vectors of Λ_n with $2^n - 1$ CVP computations in Λ_n (Lemma 3.6).
2. RankReduceCVP reduces each such CVP computation to at most $2H + 1$ (Lemma 3.5) CVP computations in Λ_{n-1} (Lemma 3.5).
3. CVPP solves each CVP computation in Λ_{n-1} with the help of the precomputed Voronoi cell in time $\tilde{O}(2^{2n})$ (Lemma 3.3).
4. RemoveNonRelevant discards vectors that are non relevant from the list produced by FindRelevant in time $\tilde{O}(2^{2n})$ (Lemma 3.7).

We conclude that computing the Voronoi cell of Λ_n is reduced to $2H + 1$ CVP computations in Λ_{n-1} each solved in $\tilde{O}(2^{2n})$ time by CVPP plus 2^{2n} polynomial time computation by RemoveNonRelevant. Therefore the total time complexity for computing V_{n-1} from V_n is $H \cdot \tilde{O}(2^{3n})$. ■

The following theorem describes the main algorithm of this section. An algorithm that computes the Voronoi cell of a given lattice with the help of two subroutines. A preprocessing subroutine (LLL) and a subroutine for computing the Voronoi cell of Λ_{k+1} using the Voronoi cell of Λ_k (ComputeVCell). The pseudocode is given as Algorithm 4.

Theorem 3.9 *There is a deterministic $\tilde{O}(2^{3.5n})$-time and $\tilde{O}(2^n)$ space algorithm that on input a lattice $\mathbf{B}$, outputs the relevant vectors of $\Lambda_n = \mathcal{L}(\mathbf{B})$.*

Proof: The BasicVoronoiCell algorithm (Algorithm 4) combines the two subroutines as follows. First it runs LLL on the input basis, and acquires a new basis $\mathbf{B}$ for the same lattice where for all $k = 2, \ldots, n$ we have $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq 2^{0.5k}$, by Lemma 3.4. The rest of the algorithm works on the new basis $\mathbf{B}$. The algorithm sets the Voronoi cell of the one dimensional lattice $\Lambda_1 = \mathcal{L}(\mathbf{b}_1)$ to $V_1 = \{\mathbf{b}_1, -\mathbf{b}_1\}$. Then for $k = 2, \ldots, n$ it uses the ComputeVCell algorithm with $H = 2^{0.5k}$, in order to compute the Voronoi cell of Λ_{k+1} using the Voronoi cell of Λ_k. Each such call has a cost of $\tilde{O}(2^{3.5k})$, therefore the total time complexity of the algorithm is $\sum_{k=2,\ldots,n}(\tilde{O}(2^{3.5k})) = \tilde{O}(2^{3.5n})$. Finally the space requirements are at most $\tilde{O}(2^n)$, as none of the called procedures requires more space. ■

Combining the BasicVoronoiCell and CVPP algorithms gives a $\tilde{O}(2^{3.5n})$ time and $\tilde{O}(2^n)$ space algorithm for CVP and a proof for the main theorem of this section (Theorem 3.1).

Proof: In order to solve CVP for target $\mathbf{t}$ on a lattice Λ, the algorithm first computes the Voronoi relevant vectors V of Λ with BasicVoronoiCell (Theorem 3.9) in $\tilde{O}(2^{3.5n})$ time and $\tilde{O}(2^n)$ space. Then it solves the CVP instance using the $\tilde{O}(2^{2n})$ CVPP algorithm and the precomputed relevant vectors of Λ (Theorem 3.3). ■

4 The optimized algorithm

In this section we present the ImprovedVoronoiCell algorithm, a deterministic $\tilde{O}(2^{2n})$ algorithm for computing the Voronoi cell and consequently solving CVP, SVP, and other hard lattice problems. The algorithm follows the general outline of the BasicVoronoiCell algorithm. First it preprocesses the input lattice to acquire a better quality basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ and sets the Voronoi cell of Λ_1 to $\{-\mathbf{b}_1, \mathbf{b}_1\}$. Then it computes the Voronoi cells of $\Lambda_2, \dots, \Lambda_n$ iteratively, using a subroutine that computes the Voronoi cell of Λ_{k+1} given the Voronoi cell of Λ_k. The improved time complexity comes from improved subroutines for both the preprocessing and voronoi cell computation. In particular the ImprovedVoronoiCell algorithm uses the following subroutines:

1. The OptPreprocess, a $\tilde{O}(2^n)$ basis reduction subroutine that on input an n-rank lattice Λ_n outputs a basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ for the same lattice, such that for all $k = 2, \dots, n$ we have $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq k^5$, (where $\Lambda_k = \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_k)$).
2. The OptComputeVCell, a subroutine that on input a basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_{k+1}\}$, an integer H such that $\mu(\Lambda_{k+1})/\|\mathbf{b}_{k+1}^*\| \leq H$ and the Voronoi cell of Λ_k outputs the Voronoi cell of Λ_{k+1} in time $H^2 \cdot \tilde{O}(2^{2k})$.

The full description and analysis of the subroutines is given in Subsections 4.1, 4.2, 4.3. In the following paragraphs we briefly revisit the initial subroutines Preprocess and ComputeVCell and show why it is possible to improve upon them.

The goal of the preprocessing step is to output a basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$ for the input lattice, such that the quantities $\mu(\Lambda_k)/\|\mathbf{b}_k^*\|$ are relatively small. The basic Preprocess subroutine simply runs LLL and achieves $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| < 2^{0.5k}$ in poly(n) time, where n is the rank of the lattice. It is natural to ask if there are algorithms that provide stronger guarantees on the output basis, ideally bounding $\mu(\Lambda_k)/\|\mathbf{b}_k^*\|$ by a polynomial of k. We expect that the time requirements of such algorithms would be much greater than LLL. However, in our case this is not an issue. The preprocessing step is run only once at the beginning of the algorithm and as long as its time complexity is lower (or equal) with the Voronoi cell computation step it cannot affect the total time complexity of the algorithm. In Subsection 4.1 we show that OptPreprocess is such an algorithm. In particular it achieves $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq k^5$ in $\tilde{O}(2^n)$ time. This is possible using block reduction techniques from [23, 48]. We remark here that replacing the LLL preprocessing with the new preprocessing algorithm directly improves the time complexity of the BasicVoronoiCell algorithm to $\tilde{O}(2^{3n})$.

Finally we show why it is possible to improve the Voronoi cell computation subroutine. Recall that ComputeVCell computes the Voronoi cell of Λ_{k+1} with the help of $H \cdot \tilde{O}(2^k)$ calls to CVPP for the lower rank lattice Λ_k, where H an integer such that $\mu(\Lambda_{k+1})/\|\mathbf{b}_{k+1}^*\| \leq H$. It turns out that the CVP instances solved by CVPP have a very special structure that allows for more efficient strategies to compute them. In particular, it is possible to group them in such a way that solving a group of 2^k CVP instances on Λ_k can be reduced to a single instance of the following problem on Λ_k:

Definition 4.1 Enum$_{2\bar{\mathcal{V}}}$ *for target* $\mathbf{t}$ *on* Λ*: Given a basis* $\mathbf{B}$ *for an* n*-rank lattice* Λ *and a target vector* $\mathbf{t}$ *on the span of* Λ*, enumerate all the vectors of the set* $\Lambda + \mathbf{t} \cap 2\mathcal{V}$*, where* $\mathcal{V}$ *is the (open) Voronoi cell of* Λ*.*

In Subsection 4.2 we give an algorithm that solves Enum$_{2\bar{\mathcal{V}}}$ given the Voronoi cell of Λ in $\tilde{O}(2^{2n})$, (while using directly CVPP results a $\tilde{O}(2^{3n})$ algorithm). In Subsection 4.3 we show how to use this algorithm for Enum$_{2\bar{\mathcal{V}}}$ to acquire a faster Voronoi cell computation subroutine. In particular the OptComputeVCell subroutine computes the Voronoi cell of Λ_{k+1} given the Voronoi cell of Λ_k in time $H^2 \cdot \tilde{O}(2^{2k})$, compared with the $H \cdot \tilde{O}(2^{3k})$ of the ComputeVCell. We remark that the increased dependence on H is not an issue, after running OptPreprocess H can be a polynomial in k.

In Subsection 4.4 we combine the improved subroutines to acquire the OptimizedVoronoiCell algorithm and give a proof for the main theorem of this section:

Theorem 4.2 *There is a deterministic* $\tilde{O}(2^{2n})$*-time and* $\tilde{O}(2^n)$ *space algorithm that on input an* n*-rank lattice* Λ *with basis* $\mathbf{B}$*, outputs the relevant vectors of* Λ*.*

Algorithm 5 Optimized Preprocessing

function OptPreprocess($\mathbf{B}$)
  for $k = n$ **downto** 2 **do**
    Run dual block reduction with
    block size $k/4$ on $\mathbf{B}_k = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$
    and replace $\mathbf{B}_k$ with the output basis.
  return B

From the description of the Voronoi cell, we immediately get a solution to many other lattice problems, e.g., the shortest vector problem (SVP) can be solved simply by picking the shortest vector in the list of lattice points describing the Voronoi cell, and the kissing number of the lattice can be computed as the number of vectors in the list achieving the same length as the shortest vector in the lattice.

Corollary 4.3 *There is a deterministic $\tilde{O}(2^{2n})$ time algorithm to solve SVP, and to compute the kissing number of a lattice.*

Once the Voronoi cell of Λ_n has been computed, then we can solve CVP using the CVPP algorithm.

Corollary 4.4 *There is a deterministic $\tilde{O}(2^{2n})$ time, $\tilde{O}(2^n)$ space algorithm to solve* CVP.

Proof: First we run the OptimizedVoronoiCell to acquire the relevant vectors V_n of the n-rank input lattice Λ in time $\tilde{O}(2^{2n})$ and space $\tilde{O}(2^n)$. Then we solve the given instance of CVP using CVPP and the precomputed relevant vectors in time $\tilde{O}(2^{2n})$. ■

Algorithms for other lattice problems, like SIVP, SAP, GCVP, SMP, can be obtained by a reduction to CVP.

Corollary 4.5 *There is a deterministic $\tilde{O}(2^{2n})$ time, $\tilde{O}(2^n)$ space algorithm to solve SIVP, SAP, GCVP and SMP.*

Proof: See [39] for reductions of the above problems to CVP. ■

4.1 Preprocessing with block reduction

In this subsection we describe and analyze the optimized preprocessing subroutine. The results are summarized in the following lemma.

Lemma 4.6 *On input a basis for an n-rank lattice Λ, the* OptPreprocess *subroutine outputs new basis $\mathbf{B}$ for the same lattice with the following property: For every sub-lattice of the form $\Lambda_k = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$, $k \leq n$, we have $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq k^5$. The subroutine is deterministic and its complexity is $\tilde{O}(2^n)$.*

Proof: First we give a deterministic $\tilde{O}(2^n)$ procedure that on input a basis for an n-rank lattice Λ, outputs a basis $\mathbf{B}$ for the same lattice, such that $\mu(\Lambda)/\|\mathbf{b}_n^*\| \leq n^5$. We achieve this by running block basis reduction algorithms from [23, 48] to the dual of the input lattice. The dual block reduction, on input a basis for an n-rank lattice Λ, outputs a basis $\mathbf{B}$ for Λ such that $\mu(\Lambda)/\|\mathbf{b}_n^*\| \leq n\beta^{n/\beta}$ (see Preliminaries section). This is achieved with the help of polynomially many calls to a SVP oracle for β-rank lattices. (The reader is referred to [23] for a thorough analysis of block reduction and and the connections between dual and primal lattice bases).

For our purposes set $\beta = n/4$ and instantiate the SVP oracle with our unoptimized $\tilde{O}(2^{3.5n})$ algorithm, in order to avoid circular dependencies. [1] We show that for this instantiation of block reduction the required property is achieved. First, observe that for $\beta = n/4$, dual block reduction gives a basis with $\mu(\Lambda)/\|\mathbf{b}_n^*\| \leq n^5$.

[1] We could instantiate the oracle with the optimized variant, but it would make the proof more complicated because we would need an induction argument.

Algorithm 6 ENUM$_{2\bar{\mathcal{V}}}$ Algorithm

function ENUM$_{2\bar{\mathcal{V}}}(\mathbf{B}, V, \mathbf{t})$
  // The following arrays are indexed
  // by $\mathbf{p} \in \{0,1\}^n$ and initialized to none.
  Visited[] $\leftarrow$ Array of 2^n vectors
  Accessible[] $\leftarrow$ Array of 2^n vectors
  $\mathbf{s}_0 \leftarrow \mathbf{t} - CVPP(\mathbf{t}, \mathbf{B}, V)$
  Visit($\mathbf{s}_0, \mathbf{B}$, Visited, Accessible)
  while no more Accessible vectors **do**
    $\mathbf{s} \leftarrow$ Shortest Accessible vector
    Visit($\mathbf{s}, \mathbf{B}$, Visited, Accessible)
  return Visited

function VISIT($\mathbf{s}, \mathbf{B}$, Visited, Accessible)
  $\mathbf{p} \leftarrow$ parity($\mathbf{s}, \mathbf{B}$)
  Visited[$\mathbf{p}$] = $\mathbf{s}$
  Accessible[$\mathbf{p}$] = none
  for $\mathbf{v} \in V$ **do**
    $\mathbf{t} \leftarrow \mathbf{s} + \mathbf{v}$
    $\mathbf{p}_t \leftarrow$ parity($\mathbf{t}, \mathbf{B}$)
    if Visited[$\mathbf{p}_t$] = none **then**
      $\mathbf{t}_{prev} \leftarrow$ Accessible[$\mathbf{p}_t$]
      if $\mathbf{t}_{prev}$ = none OR $\|\mathbf{t}_{prev}\| > \|\mathbf{t}\|$ **then**
        Accessible[$\mathbf{p}_t$] = $\mathbf{t}$
Where:
$\mathbf{p} \leftarrow$parity$(\mathbf{s}, \mathbf{B}) \Leftrightarrow \mathbf{s} \in 2\Lambda + \mathbf{B} \cdot \mathbf{p}, \mathbf{p} \in \{0,1\}^n$

This is achieved with polynomially many calls to our algorithm for rank $n/4$ which gives a total complexity of $\tilde{O}(2^{3.5n/4}) < \tilde{O}(2^n)$ and concludes the analysis.

Now we can just apply this procedure for all Λ_k for $k = n, n-1, \ldots, 2$ and get a basis with $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq k^5$ for all k. The time complexity is essentially the same $\sum_{i=2}^{i=n} \tilde{O}(2^i) = \tilde{O}(2^n)$. ■

4.2 Enumeration of $(\Lambda + \mathbf{t}) \cap 2\mathcal{V}$

In this subsection we describe and analyze a procedure that given a basis $\mathbf{B}$ of an n-rank lattice Λ, the relevant vectors V of Λ and a target $\mathbf{t}$ on the linear span of Λ, outputs a set of at most 2^n vectors that contain all the points in $\Lambda + \mathbf{t} \cap 2\mathcal{V}$.

The algorithm considers a partitioning of the coset $\Lambda + \mathbf{t}$ to 2^n cosets of the form $C_{\mathbf{B},\mathbf{p},\mathbf{t}} = 2\Lambda + \mathbf{B} \cdot \mathbf{p} + \mathbf{t}$ for $\mathbf{p} \in \{0,1\}^n$. We say that the vector $\mathbf{u}$ has parity $\mathbf{p}$ if $\mathbf{u} \in C_{\mathbf{B},\mathbf{p},\mathbf{t}}$. We also consider the (infinite) graph G with:

- Nodes $U = \{\ldots, \mathbf{u}_i, \ldots\}$ all the points of the set $\Lambda + \mathbf{t}$ [2]
- and edges between the nodes $\mathbf{u}_i, \mathbf{u}_j$ if and only if $\mathbf{u}_i - \mathbf{u}_j$ is a relevant vector.

Let U' the nodes of U that are in $2\mathcal{V}$, and G' the subgraph of G induced by U'. G' has a number of useful properties used by our algorithm. Notice that if $\mathbf{u}_i$ is a node in G' with parity $\mathbf{p}$, ($\mathbf{u}_i \in 2\Lambda + \mathbf{B} \cdot \mathbf{p} + \mathbf{t} \cap 2\mathcal{V}$) it is a *unique* shortest vector of $C_{\mathbf{B},\mathbf{p},\mathbf{t}}$. We conclude that for every parity $\mathbf{p}$ there is at most one node in the graph G'. Given that there are 2^n different parities this bounds the number of nodes in G' to at most 2^n. Finally we have the following lemma for G'.

Lemma 4.7 *For any node $\mathbf{u} \in U'$ of the sub-graph G' defined above, there exist a path $\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_s$ in G' with nodes of decreasing norms such that $\mathbf{u}_s$ is a shortest vector of the coset $\Lambda + \mathbf{t}$.*

Proof: Recall that CVPP$_{2\bar{\mathcal{V}}}$ starts with a point in $\mathbf{u} \in 2\mathcal{V} \cap \Lambda + \mathbf{t}$ and by subtracting relevant vectors it generates a sequence of points with decreasing norm $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_s \in 2\mathcal{V} \cap \Lambda + \mathbf{t}$, such that $\mathbf{u}_s$ is a shortest vector in $\Lambda + \mathbf{t}$. This sequence of points gives the required path. ■

Now notice that traversing the nodes of G' is equivalent with enumerating the set $\Lambda + \mathbf{t} \cap 2\mathcal{V}$. Also all nodes in G' are accessible from the shortest vector $\mathbf{u}_s$ of $\Lambda + \mathbf{t}$. We could traverse G' by first computing $\mathbf{u}_s$ and then using depth first search on G'. Given any node $\mathbf{u}$ and the relevant vectors V we can easily compute all the accessible nodes on G from $\mathbf{u}$, and then discard all the nodes that are not in G'. Unfortunately checking if a node is in G' costs $\tilde{O}(2^n)$ and a simple depth first search would give $\tilde{O}(2^{3n})$ algorithm (for each of 2^n

[2] Notice that the set of vertices is denoted as U instead of the usual V. The letter V is used to refer to relevant vectors.

nodes in G', for each of their 2^n neighbors we should check if they are in G'). Our algorithm avoids checking containment in G' using the fact that for each parity there exist at most one node in G' and it is the shortest node among all the nodes in G of its parity.

We give an algorithm that traverses all the nodes of G' and consequently enumerates the set $\Lambda + \mathbf{t} \cap 2\mathcal{V}$ given the relevant vectors V of an n-rank lattice Λ in time $\tilde{O}(2^{2n})$, the complete pseudocode is given in Algorithm 6. We remark that the algorithm might visit nodes outside G', if there are parities for which there is no node inside G'. In order to efficiently traverse G' the algorithm keeps a list of Visited nodes and a list of Accessible nodes, both of length 2^n, indexed by parities $\mathbf{p} \in \{0,1\}^n$. (Whenever we add a vector $\mathbf{u}$ in a list, it is added to the index corresponding to its parity. For example if we add a vector $\mathbf{u} \in C_{\mathbf{B},\mathbf{p},\mathbf{t}}$ in the list Accessible we have that Accessible$[\mathbf{p}] = \mathbf{u}$.) The first step of the algorithm is to compute the shortest vector $\mathbf{u}_s$ of $\Lambda + \mathbf{t}$ using one CVPP computation that costs $\tilde{O}(2^{2n})$, $\mathbf{u}_s = \mathbf{t} - CVPP(\mathbf{t}, V)$. The algorithm then visits $\mathbf{u}_s$. At each subsequent step the algorithm visits the shortest vector on the Accessible list. Whenever the algorithm visits a node $\mathbf{u}$ the following computations take place:

1. The vector $\mathbf{u}$ is removed from the Accessible list and added to the Visited list to the corresponding parity index.
2. Then the algorithm considers all the accessible nodes in G from $\mathbf{u}$ by adding relevant vectors. It computes the vectors $\mathbf{a}_i = \mathbf{u} + \mathbf{v}_i$ for all $\mathbf{v}_i \in V$ and for each of them does the following:
   (a) If there is a visited node with the same parity as $\mathbf{a}_i$, or an accessible node with shorter norm for the same parity, it discards $\mathbf{a}_i$.
   (b) Else it inserts $\mathbf{a}_i$ to the accessible list, discarding possible accessible nodes with the same parity (but larger norm).

The algorithm terminates when there are no more accessible nodes, and outputs the vectors of the Visited list.

Lemma 4.8 *Given a basis $\mathbf{B}$ for an n-rank lattice Λ, the relevant vectors V for Λ and a target vector $\mathbf{t}$, the $\text{ENUM}_{2\mathcal{V}}$ algorithm (Algorithm 6) computes a set of at most 2^n vectors that includes all the vectors of $\Lambda + \mathbf{t} \cap 2\mathcal{V}$, in time $\tilde{O}(2^{2n})$.*

Proof: For the time complexity of the algorithm, notice that the initial CVPP computation costs $\tilde{O}(2^{2n})$ and that for each visited node the algorithm does $\tilde{O}(2^n)$ additional computations. It is not hard to validate that the algorithm will visit at most 1 vector from each parity. Therefore there are at most 2^n visited nodes and the complexity is $\tilde{O}(2^{2n})$.

Now in order to prove correctness we show that if $\mathbf{u}$ is in G' then it is visited. Assume that a node $\mathbf{u} \in G'$ is not visited after the algorithm terminates. By Lemma 4.7 there exist a path in G' with vectors of decreasing norms $\mathbf{u}, \mathbf{u}_1, \ldots, \mathbf{u}_N = \mathbf{u}_s$. The last node of the path is the shortest vector of $\Lambda + \mathbf{t}$ and is surely visited. Therefore there exist two nodes $\mathbf{u}_k, \mathbf{u}_{k+1}$ in the sequence such that $\mathbf{u}_k$ is not visited, while $\mathbf{u}_{k+1}$ is. When $\mathbf{u}_{k+1}$ was visited, the node $\mathbf{u}_k$ was considered for inclusion in the accessible list. Notice that $\mathbf{u}_k$ is in G' and consequently the shortest node of G for its parity. As a result $\mathbf{u}_k$ would be included in the accessible list and eventually visited, unless a node $\mathbf{x}$ with the same parity was visited first. However this is impossible because $\mathbf{u}_k$ is in $2\mathcal{V}$ and therefore it is the shortest vector of its parity, so shorter than $\mathbf{x}$. Also all the nodes in the path from $\mathbf{u}_{k+1}$ to $\mathbf{u}_s$ are shorter that $\mathbf{u}_k$. Consequently the algorithm would rather visit the nodes of the path than $\mathbf{x}$. We conclude that the assumption is false and all the points in G' are visited. ■

4.3 Relevant vectors using $\text{ENUM}_{2\mathcal{V}}$

In this subsection we give an algorithm that on input a basis $\mathbf{B}_n = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ for an n-rank lattice, the relevant vectors V_{n-1} of $\Lambda_{n-1} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{n-1})$ and an integer H such that $\mu(\Lambda_n)/\|\mathbf{b}_n^*\| \leq H$, computes the relevant vectors of V_n with $4H + 1$ calls to the algorithm $\text{ENUM}_{2\mathcal{V}}$ described in the previous subsection. First we give two useful lemmas for the relevant vectors of Λ_n. The lemmas restrict the search space for relevant vectors.

Algorithm 7 Voronoi cell computation

function OPTCOMPUTEVCELL($\mathbf{B}_k, V_{k-1}, H$)
  $V_k \leftarrow V_{k-1}$
  for all h **that:** $|h| < H$ **do**
    $V_{k,h} \leftarrow \text{ENUM}_{2\bar{\mathcal{V}}}(\mathbf{B}_k, V_k, h(\mathbf{b}_k - \mathbf{b}_k^*))$
    Add $h\mathbf{b}_k^*$ to every element of $V_{k,h}$
    $V_k \leftarrow V_k \bigcup V_{k,h}$
  $V_k \leftarrow \text{RemoveNonRelevant}(V_k)$
  return V_k

Lemma 4.9 *Let $\mathbf{B}_n$, Λ_n, Λ_{n-1} as defined above and $\mathbf{b}_n^*$ the component of $\mathbf{b}_n$ orthogonal to Λ_{n-1}. If $\mathbf{u}$ is a relevant vector of Λ_n, then it belongs to some layer $\Lambda_{n-1} + h\mathbf{b}_n$ with h an integer and $|h| < 2\mu(\Lambda_n)/\|\mathbf{b}_n^*\|$.*

Proof: Partition the lattice Λ_n in layers of the form $\Lambda_{n-1} + h\mathbf{b}_n$ with $h \in \mathbb{Z}$. Notice the following:

- If $\mathbf{u}$ is a relevant vector then it is in $2\bar{\mathcal{V}}$.
- For all the points in $\bar{\mathcal{V}}$ the vector $\mathbf{0}$ is a closest vector. Therefore the norm of any point in $\bar{\mathcal{V}}$ is at most $\mu(\Lambda_n)$.
- The norm of all lattice points in the layer $\Lambda_{n-1} + h\mathbf{b}_n$ is at least $|h| \cdot \|\mathbf{b}_n^*\|$, because this is the distance between $\mathbf{0}$ and the entire affine space generated by the layer.

By the above it is not hard to validate that the norm of the relevant vectors is at most $2\mu(\Lambda)$ and consequently they belong to layers with $|h| < 2\mu(\Lambda_n)/\|\mathbf{b}_n^*\|$. ■

Lemma 4.10 *Let $\mathbf{B}_n$, Λ_n, $\mathbf{B}_{n-1}$, Λ_{n-1} as defined above, $\mathcal{V}_{n-1}$ the open Voronoi cell of Λ_{n-1} and V_{n-1} the set of the corresponding relevant vectors. If $\mathbf{u}$ is a relevant vector of Λ_n then:*

- *If $\mathbf{u} \in \Lambda_{n-1}$, $\mathbf{u}$ is relevant for Λ_{n-1},*
- *if $\mathbf{u} \notin \Lambda_{n-1}$, $\mathbf{u}$ is in the cylinder with basis $2\mathcal{V}_{n-1}$*

Proof: Let $\mathbf{u}$ a relevant vector of Λ_n, equivalently $\pm\mathbf{u}$ are the only shortest vectors of the coset $2\Lambda_n + \mathbf{u}$. If $\mathbf{u} \in \Lambda_{n-1}$ then $\pm\mathbf{u}$ are also the only shortest vectors of the coset $2\Lambda_{n-1} + \mathbf{u}$ therefore $\mathbf{u}$ is a relevant vector of Λ_{n-1}. Now assume $\mathbf{u} \notin \Lambda_{n-1}$. The vectors $\pm\mathbf{u}$ should be the only shortest vectors of $2\Lambda + \mathbf{u}$ therefore:

$$\forall \mathbf{v} \in \Lambda_n,\ \mathbf{v} \neq \mathbf{u}\ : \|\mathbf{u} - 2\mathbf{v}\| > \|\mathbf{u}\|.$$

But $\mathbf{u} \notin \Lambda_{n-1}$, therefore for all the relevant vectors $\mathbf{v}_i$ of Λ_{n-1}: $\mathbf{u} \neq \mathbf{v}_i$. Combining this fact with the equation above we have:

$$\forall \mathbf{v}_i \in V_{n-1} : \|\mathbf{u} - 2\mathbf{v}_i\| > \|\mathbf{u}\|.$$

Notice that the equation above describes exactly the points inside a cylinder with basis $2\mathcal{V}_{n-1}$. ■

Finally we are ready to state the lemma that summarizes the results of this subsection.

Lemma 4.11 *Let $\mathbf{B}_n$, Λ_n, $\mathbf{B}_{n-1}$, Λ_{n-1}, $\mathbf{b}_n^*$, $\mathcal{V}_{n-1}$, V_{n-1} as defined above. On input the Voronoi cell of Λ_{n-1} and an integer H such that $\mu(\Lambda_n)/\|\mathbf{b}_n^*\| \leq H$, the* OPTCOMPUTEVCELL *algorithm (Algorithm 7) computes the relevant vectors of Λ_n in time $H^2 \cdot \tilde{O}(2^{2n})$ and space $H \cdot \tilde{O}(2^n)$.*

Proof: In order to compute the relevant vectors the algorithm partitions Λ_n in layers of the form $\Lambda_{n-1} + h\mathbf{b}_n$. For each layer with $|h| < 2H$, the algorithm computes a set of at most 2^n vectors that contains all the relevant vectors of the layer. By Lemma 4.9 the union of these sets is a superset of the relevant vectors. The additional

Algorithm 8 Optimized main algorithm

```
function OptimizedVoronoiCell(B)
    B ← OptPreprocess(B)
    V_1 = {(b_1, −b_1)}
    for k = 2 to n do
        B_k ← [b_1, ..., b_k]
        V_k ← OptComputeVCell(B_k, V_{k−1}, k^5)
    return V_n
```

vectors are discarded with the procedure RemoveNonRelevant. We go on to show how to find appropriate sets for each layer.

For the layer with $h = 0$ the precomputed relevant vectors of Λ_{n-1} are sufficient (by Lemma 4.10). Now, for layers with $h \neq 0$ the intersection of $\Lambda_{n-1} + h\mathbf{b}_n$ with the cylinder of basis $2\mathcal{V}_{n-1}$ is an appropriate set (by Lemma 4.10). Notice that projecting vectors from $\Lambda_{n-1} + h\mathbf{b}_n$ to the linear span of Λ_{n-1} gives a one to one mapping between vectors $\mathbf{v}$ on the layer and their projections on the linear span of Λ_{n-1} $\mathbf{v}^{\|} = \mathbf{v} - h\mathbf{b}_n^*$. In order to enumerate the intersection of $\Lambda_{n-1} + h\mathbf{b}_n$ with the cylinder with basis $2\mathcal{V}_{n-1}$ we project it to Λ_{n-1}, enumerate the projections and acquire the original vectors by adding $h\mathbf{b}_n^*$. Notice that this projection is the set $\Lambda_{n-1} + h(\mathbf{b}_n - \mathbf{b}_n^*) \cap 2\mathcal{V}_{n-1}$ and enumerating these vectors is an instance of $\text{Enum}_{2\bar{\mathcal{V}}}$.

For the complexity of the algorithm notice that it enumerates at most $H \cdot \tilde{O}(2^n)$ vectors using $4H + 1$ calls to $\text{Enum}_{2\bar{\mathcal{V}}}$ that requires $\tilde{O}(2^{2n})$ time. Finally it runs RemoveNonRelevant to discard non relevant vectors. Therefore the total time complexity is $H^2 \cdot \tilde{O}(2^{2n})$ and space complexity $H \cdot \tilde{O}(2^n)$. ■

We remark that we can reduce the space complexity of the above algorithm to $\tilde{O}(2^n)$ and time complexity to $H \cdot \tilde{O}(2^{2n})$ but it does not change the total asymptotics of the final algorithm because H is going to be polynomial in n.

4.4 Combining the blocks

Finally we present OptimizedVoronoiCell, a deterministic $\tilde{O}(2^{2n})$-time and $\tilde{O}(2^n)$-space algorithm for computing the Voronoi cell of an n- rank lattice Λ and prove the main theorem of this section (Theorem 4.2).

Proof: Given a basis for an n-rank lattice the OptimizedVoronoiCell algorithm does the following:

1. First it applies the $\tilde{O}(2^n)$ OptPreprocessing algorithm to get a basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ for the same lattice such that $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| \leq k^5$ for all $k = 2, \ldots, n$ (Lemma 4.1). The rest of the algorithm is using the new basis.

2. It sets the voronoi cell of $\Lambda_1 = \mathcal{L}(\mathbf{b}_1)$ as $V_1 = \{\mathbf{b}_1, -\mathbf{b}_1\}$.

3. Then for $k = 2, \ldots, n$ it runs the OptComputeVCell algorithm to acquire V_k from V_{k-1} in time $\tilde{O}(2^{2k})$ and space $\tilde{O}(2^k)$ (Lemma 4.11, with $H \leq n^5$).

Notice that the total time complexity of the algorithm is $\tilde{O}(2^n) + \sum_{k=1}^{n} \tilde{O}(2^{2k}) = \tilde{O}(2^{2n})$ and the space complexity is $\tilde{O}(2^n)$. ■

5 Open problems, directions for further research

We have shown that CVP, SVP, SIVP and many other lattice problems can be solved in deterministic single exponential time. Many open problems remain. Here we list those that we think are most important or interesting.

Our algorithm uses exponential space. It would be nice to find an algorithm running in exponential time and polynomial space.

We described an algorithm for the ℓ_2 norm. Many parts of the algorithm easily adapt to other norms as well, but it is not clear how to extend our results to all ℓ_p norms. The main technical problem is that the Voronoi cells in ℓ_p norms for $p \neq 2$ are not necessarily convex. So, extending the algorithm to all ℓ_p norms may require some substantially new idea. An important application of extending our algorithm to other ℓ_p norms is that it would immediately lead to single exponential time algorithms for integer programming [31].

As we have already discussed we can solve CVP in time $\tilde{O}(2^{2n})$. It is an open question if we can further decrease the constant in the exponent, with better analysis of the CVPP algorithm and faster algorithms for $\text{ENUM}_{2\bar{\mathcal{V}}}$. However, it is clear that our approach cannot possibly lead to constants in the exponent smaller than 1 (as achieved for example by randomized heuristics for SVP [44, 42].) Still, it may be possible to extend our ideas to develop an algorithm with running time proportional to the number of Voronoi relevant vectors. This may give interesting algorithms for special lattices whose Voronoi cell has a small description. Another possible research direction is to develop practical variants of our algorithm that use only a sublist of Voronoi relevant vectors, at the cost of producing only approximate solutions to CVP.

It would be nice to extend our algorithm to yield a single exponential time solution to the covering radius problem, or equivalently, the problem of computing the diameter of the Voronoi cell of a lattice. In principle, this could be done by enumerating the vertices of the Voronoi cell, and selecting the longest, but this would not lead to a single exponential time algorithm because the number of such vertices can be as large as $n^{\Omega(n)}$. No NP-hardness proof for the covering radius problem in the ℓ_2 norm is known (but see [27] for NP-hardness results in ℓ_p norm for large p). Still, the problem seems quite hard: the covering radius problem is not even known to be in NP, and it is conjectured to be Π_2-hard [38, 25] for small approximation factors. Counting the number of vertices of the Voronoi cell [52] or the number of lattice points of a given length [16] is also known to be $\#P$-hard.

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, Aug. 2002.

[2] D. Aharonov and O. Regev. Lattice problems in NP intersect coNP. *Journal of the ACM*, 52(5):749–765, 2005. Preliminary version in FOCS 2004.

[3] M. Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of STOC '98*, pages 10–19. ACM, May 1998.

[4] M. Ajtai. Generating hard instances of lattice problems. *Complexity of Computations and Proofs, Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.

[5] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of STOC '01*, pages 266–275. ACM, July 2001.

[6] M. Ajtai, R. Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *Proceedings of CCC '02*, pages 53–57. IEEE, May 2002.

[7] M. Alekhnovich, S. Khot, G. Kindler, and N. Vishnoi. Hardness of approximating the closest vector problem with pre-processing. In *Proceedings of FOCS 2005*. IEEE, Oct. 2005.

[8] S. Arora, L. Babai, J. Stern, and E. Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *Journal of Computer and System Sciences*, 54(2):317–331, Apr. 1997. Preliminary version in FOCS'93.

[9] L. Babai. On Lovasz' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[10] W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(1):625–635, 1993.

[11] J. Blömer. Closest vectors, successive minima and dual HKZ-bases of lattices. In *Proceedings of ICALP '00*, volume 1853 of *LNCS*, pages 248–259. Springer, July 2000.

[12] J. Blömer and S. Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theoretical Computer Science*, 410(18):1648–1665, Apr. 2009. Preliminary version in ICALP 2007.

[13] J. Blömer and J.-P. Seifert. On the complexity of computing short linearly independent vectors and short bases in a lattice. In *Proceedings of STOC '99*, pages 711–720. ACM, May 1999.

[14] J.-Y. Cai and A. P. Nerurkar. Approximating the SVP to within a factor $(1+1/dim^{\epsilon})$ is NP-hard under randomized reductions. *Journal of Computer and System Sciences*, 59(2):221–239, Oct. 1999. Prelim. version in CCC 1998.

[15] J. W. S. Cassels. *An introduction to the geometry of numbers*. Springer-Verlag, New York, 1971.

[16] D. X. Charles. Counting lattice vectors. *Journal of Computer and System Sciences*, 73(6):962 – 972, 2007.

[17] W. Chen and J. Meng. The hardness of the closest vector problem with preprocessing over ℓ_∞ norm. *IEEE Transactions on Information Theory*, 52(10):4603–4606, 2006.

[18] J. H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*. Springer Verlag, 3rd edition, 1998.

[19] M. J. Coster, A. Joux, B. A. LaMacchia, A. M. Odlyzko, C.-P. Schnorr, and J. Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2(2):111–128, 1992. Preliminary versions in Eurocrypt '91 and FCT '91.

[20] I. Dinur, G. Kindler, R. Raz, and S. Safra. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243, 2003. Preliminary version in FOCS 1998.

[21] U. Feige and D. Micciancio. The inapproximability of lattice and coding problems with preprocessing. *Journal of Computer and System Sciences*, 69(1):45–67, 2003. Preliminary version in CCC 2002.

[22] N. Gama, N. Howgrave-Graham, H. Koy, and P. Nguyen. Rankin's constant and blockwise lattice reduction. In *Advances in Cryptology – Proceedings of CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 112–130. Springer, Aug. 2006.

[23] N. Gama and P. Q. Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of STOC '08*, pages 207–216. ACM, May 2008.

[24] O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999.

[25] V. Guruswami, D. Micciancio, and O. Regev. The complexity of the covering radius problem. *Computational Complexity*, 14(2):90–121, jun 2005. Preliminary version in CCC 2004.

[26] G. Hanrot and D. Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In *Proceedings of CRYPTO '07*, volume 4622 of *LNCS*, pages 170–186. Springer, Aug. 2007.

[27] I. Haviv and O. Regev. Hardness of the covering radius problem on lattices. In *Proceedings of CCC '06*, pages 145–158. IEEE, July 2006.

[28] I. Haviv and O. Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proceedings of STOC '07*, pages 469–477. ACM, June 2007.

[29] B. Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theoretical Computer Science*, 41(2–3):125–139, Dec. 1985.

[30] A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185, 1998.

[31] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operation research*, 12(3):415–440, Aug. 1987. Prelim. version in STOC 1983.

[32] S. Khot. Hardness of approximating the shortest vector problem in lattices. *Journal of the ACM*, 52(5):789–808, Sept. 2005. Preliminary version in FOCS 2004.

[33] S. Landau and G. L. Miller. Solvability by radicals is in polynomial time. *Journal of Computer and System Sciences*, 30(2):179–208, Apr. 1985. Preliminary version in STOC 1983.

[34] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982.

[35] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, Nov. 1983.

[36] D. Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, Mar. 2001.

[37] D. Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, Mar. 2001. Prelim. version in FOCS 1998.

[38] D. Micciancio. Almost perfect lattices, the covering radius problem, and applications to Ajtai's connection factor. *SIAM Journal on Computing*, 34(1):118–169, 2004. Preliminary version in STOC 2002.

[39] D. Micciancio. Efficient reductions among lattice problems. In *Proceedings of SODA 2008*, pages 84–93. ACM/SIAM, Jan. 2008.

[40] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, Mar. 2002.

[41] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measure. *SIAM Journal on Computing*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[42] D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of SODA 2010*. ACM/SIAM, Jan. 2010.

[43] P. Nguyen and J. Stern. The two faces of lattices in cryptology. In *Proceedings of CaLC '01*, volume 2146 of *LNCS*, pages 146–180. Springer, Mar. 2001.

[44] P. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology*, 2(2):181–207, jul 2008.

[45] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In C. Pomerance, editor, *Cryptology and computational number theory*, volume 42 of *Procedings of Symposia in Applied Mathematics*, pages 75–88, Boulder, Colorado, 1989. AMS.

[46] X. Pujol and D. Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *Cryptology ePrint Archive, Report 2009/605*, 2009.

[47] O. Regev. Improved inapproximability of lattice and coding problems with preprocessing. *IEEE Transactions on Information Theory*, 50(9):2031–2037, 2004. Preliminary version in CCC 2003.

[48] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2–3):201–224, Aug. 1987.

[49] C.-P. Schnorr. A more efficient algorithm for lattice basis reduction. *Journal of Algorithms*, 9(1):47–62, Mar. 1988.

[50] C. P. Schnorr. Fast LLL-type lattice reduction. *Information and Computation*, 204(1):1–25, Jan. 2006.

[51] C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, Aug. 1994. Preliminary version in FCT 1991.

[52] M. D. Sikirić, A. Schürmann, and F. Vallentin. Complexity and algorithms for computing Voronoi cells of lattices. *Mathematics of Computation*, 78(267):1713–1731, July 2009.

[53] N. Sommer, M. Feder, and O. Shalvi. Finding the closest lattice point by iterative slicing. *SIAM J. Discrete Math.*, 23(2):715–731, Apr. 2009.

[54] P. van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical Report 81-04, Mathematische Instituut, Universiry of Amsterdam, 1981. Available on-line at URL `http://turing.wins.uva.nl/~peter/`.

[55] E. Viterbo and E. Biglieri. Computing the Voronoi cell of a lattice: the diamond-cutting algorithm. *IEEE Trans. on Information Theory*, 42(1):161–171, Jan. 1996.