Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 14 (2010)

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

Daniele Micciancio Panagiotis Voulgaris

University of California at San Diego, Department of Computer Science and Engineering,
9500 Gilman Dr., Mail Code 0404, La Jolla, CA 92093, USA. Emails:
daniele@cs.ucsd.edu, pvoulgar@ucsd.edu

Abstract

We give deterministic $\tilde{O}(2^{2n})$-time $\tilde{O}(2^n)$-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, 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. A direct consequence of our results is a derandomization of the best current polynomial time approximation algorithms for SVP and CVP achieving $2^{O(n \log\log n/\log n)}$ approximation factor.

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 a 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.[1]

Beside being classic mathematical problems in the study of the *geometry of numbers* [Cas71], these problems play an important role in many computer science and communication theory applications. SVP and CVP

*A preliminary version of this work appears in the proceedings of STOC 2010 [MV10a]. This is the full version of the paper. Invitated submission to STOC 2010 special issue of SIAM J. on Computing.

[1]More technically, the smallest real r such that there exist n linearly independent lattice vectors of length bounded by r is denoted λ_n. SIVP asks to find n linearly independent lattice vectors of length at most λ_n.

ISSN 1433-8092

have been used to solve many landmark algorithmic problems in theoretical computer science, like integer programming [Len83, Kan87], factoring polynomials over the rationals [LLL82], checking the solvability by radicals [LM85], solving low density subset-sum problems [CJL$^+$92] and breaking the Merkle-Hellman cryptosystem [Odl89] (among many other cryptanalysis problems, e.g. see [JS98, NS01]). SIVP is the main problem underlying the construction of lattice based cryptographic functions with worst-case/average-case connection [Ajt04, MR07, Reg09]. SVP and CVP also have many applications in communication theory, e.g., lattice coding for the Gaussian channel and vector quantization [CS98].

The complexity of lattice problems has been investigated intensively. All three problems mentioned above have been shown to be NP-hard (possibly under randomized reductions) both to solve exactly [vEB81, Ajt98, BS99], or even approximately within small (constant or sub-polynomial in n) approximation factors [BS99, ABSS97, DKRS03, CN99, Mic01b, Kho05, HR07]. Much effort has gone into the development and analysis of algorithms both to solve these problems exactly [Kan87, Hel85, HS07, Blö00, AKS01, AKS02, BN09] and to efficiently find approximate solutions [LLL82, Sch88, Sch87, SE94, Sch06, GHGKN06, GN08].

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 [CS98]) involve lattices in relatively small dimension, and because approximation algorithms for high dimensional lattices [Sch87, SE94, GHGKN06, GN08] (for which exact solution is not feasible) typically involve the exact solution of low dimensional subproblems. Prior to our work, the best deterministic algorithm to solve any of these lattice problems exactly was still essentially the one discovered by Kannan [Kan87] in 1983, running in time $n^{O(n)}$, where n is the dimension of the lattice. Subsequent work [Hel85, HS07] led 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 had seen asymptotically significant improvements in the exponent is SVP, for which Ajtai, Kumar and Sivakumar [AKS01] gave a *randomized* algorithm running in $2^{O(n)}$ time (and space), typically referred to as the AKS sieve. Following [AKS01] much work has been devoted to better understand and improve the sieve algorithm [AKS02, BN09, AJ08, NV08, MV10b, PS09, HPS11, WLTB11]. Still the main questions raised by [AKS01] 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 [NV08, MV10b, PS09] improved analyses 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 [AKS02, BN09, AJ08, EHN11], 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 [BN09]. A possible explanation for the difficulty of extending the result of [AKS01] to the exact solution of SIVP and CVP was offered by Micciancio in [Mic08], where it is shown (among other things) that CVP, SIVP and all other lattice problems considered in [BN09], 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 [GMSS99, Mic08], 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. [Kan87, BN09] from $n^{O(n)}$ to $\tilde{O}(2^{2n})$. In the case of SVP, we achieve single exponential time as in [AKS01, NV08, MV10b, PS09, WLTB11], but without using randomization and with a better theoretical constant in the exponent. In the process, we also provide deterministic single exponential time algorithms for various other classic computational problems on lattices, like computing the kissing number, and computing the list of all Voronoi relevant vectors.

We remark that all our algorithms, just like [AKS01], use exponential space. So, the question whether exponential space is required to solve lattice problems in single exponential time remains open.

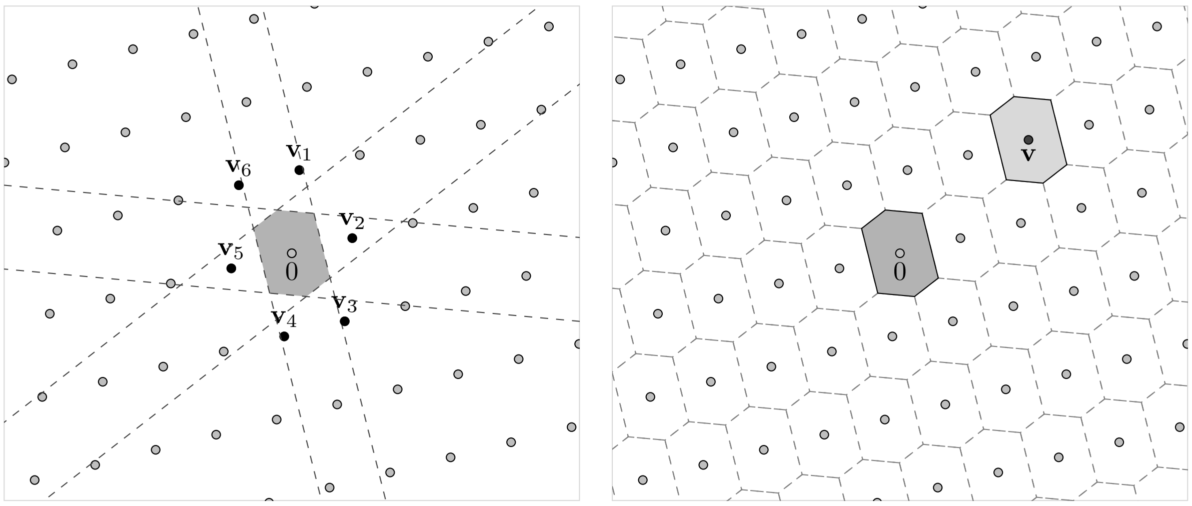

Figure 1: The Voronoi cell of a lattice is the set of all points that are closer to the origin than to any other lattice point. (Left) A lattice and its Voronoi cell (shaded area). The Voronoi cell is the intersection of the half-spaces defined by all nonzero lattice vectors, where the half-space of $\mathbf{v}$ is the set of points that are closer to the origin than to $\mathbf{v}$. Not all half-spaces/lattice vectors are relevant when taking this intersection. The figure shows the relevant vectors $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5, \mathbf{v}_6$ and the (1-dimensional) hyperplanes (dashed lines) defining the corresponding half-spaces. (Right) The Voronoi cell is a fundamental region of the lattice: copies of the Voronoi cell (centered around all lattice points) tile the whole space. The Voronoi cell of a lattice vector $\mathbf{v}$ is the set of points that are closer to $\mathbf{v}$ than to any other lattice vector.

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 (open) Voronoi cell of a lattice Λ 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 vector $\mathbf{v}$, $H_{\mathbf{v}} = \{\mathbf{x}\colon \|\mathbf{x}\| < \|\mathbf{x} - \mathbf{v}\|\}$ is the set of all points that are closer to the origin than to $\mathbf{v}$. It is not necessary to consider all $\mathbf{v} \in \Lambda \setminus \{\mathbf{0}\}$ 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 $\mathcal{V}$. (See Figure 1.) 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 or relevant vectors.[2]

The relation between the Voronoi cell and CVPP is well known, and easy to explain. (See Figure 3 and Observation 3.1.) 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 $\mathbf{t}' = \mathbf{t} - \mathbf{v}$ 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 $(\mathbf{t} - \Lambda) \cap \bar{\mathcal{V}} = (\Lambda + \mathbf{t}) \cap \bar{\mathcal{V}}$ where $\bar{\mathcal{V}}$ is the topological closure of $\mathcal{V}$. The idea of using

[2]Since the Voronoi cell is symmetric about the origin, the half-spaces come in pairs $H_{\mathbf{v}}, H_{-\mathbf{v}}$, and it is enough to store one vector $\mathbf{v}$ to represents both half-spaces from each pair. For simplicity, in the rest of the paper, we omit this simple optimization, and consider the full list of Voronoi relevant vectors that stores both $\mathbf{v}$ and $-\mathbf{v}$ explicitly.

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 [SFS09]. 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 $\mathbf{t} \notin \bar{H}_{\mathbf{v}} = \{\mathbf{x}\colon \|\mathbf{x}\| \le \|\mathbf{x} - \mathbf{v}\|\}$, then the length of $\mathbf{t}$ can be reduced by subtracting $\mathbf{v}$ from $\mathbf{t}$. So, as long as $\mathbf{t}$ is outside $\bar{\mathcal{V}} = \bigcap_{\mathbf{v}} \bar{H}_{\mathbf{v}}$, we can make 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. The work [SFS09] 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 [SFS09] is of the form $n^{O(n)}$: the running time of the iterative slicer is bounded by a volume argument, counting the number of lattice points within a sphere of radius $\|\mathbf{t}\|$, and this can be well above $2^{O(n)}$ or even $n^{O(n)}$.

In the next two paragraphs we first sketch (a simplified version of) 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 and within a similar running time. The algorithm outlined here roughly corresponds to the basic algorithm presented in Section 4 and the conference version of this paper [MV10a], with running time $\tilde{O}(2^{3.5n})$. In Section 5 we use some additional ideas to improve the running time to $\tilde{O}(2^{2n})$.

The CVPP algorithm As already noted, the goal of CVPP with target vector $\mathbf{t}$ can be restated as the problem of finding a point $\mathbf{t}' \in (\Lambda + \mathbf{t}) \cap \bar{\mathcal{V}}$, or, equivalently, a shortest vector in the coset $\Lambda + \mathbf{t}$. (See Observation 3.1). We follow an approach similar to the iterative slicer of [SFS09]. 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 coset. However, in order to bound the number of iterations, we introduce two important modifications to the greedy strategy of [SFS09]. 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, we show that it is possible to generate a sequence of shorter and shorter vectors in $\Lambda + \mathbf{t}$, with the additional property that all the vectors are inside $2\bar{\mathcal{V}}$. This allows to bound the length of the sequence by $2^{O(n)}$. For each vector of the sequence the algorithm spends $2^{O(n)}$ time, which gives a total time complexity of $2^{O(n)}$. We stress that our algorithm is essentially the same as the iterative slicer of [SFS09], but with a different selection strategy. The advantage of our selection process over the greedy method of [SFS09] is primarily theoretical: it allows to prove termination in single exponential time. Much work still needs to be done in order to better understand the practical impact of using different selection strategies within the algorithm of [SFS09].

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 [AEVZ02] 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 [Kan87, HS07]. This procedure allows to solve CVPP in a lattice Λ of rank n making only $2^{O(n)}$ calls to a CVPP oracle for a properly chosen sub-lattice Λ' of rank $n-1$, which can be found in polynomial time. 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 $\text{rank}(\Lambda_i) = i$, and iteratively computing the Voronoi cell of Λ_{i+1} using the previously computed Voronoi cell of Λ_i. Since each $\mathcal{V}(\Lambda_i)$ can be computed from $\mathcal{V}(\Lambda_{i-1})$ in time $2^{O(n)}$ (by means of $2^{O(n)}$ CVPP computations, each taking $2^{O(n)}$ time), the total running time is $2^{O(n)}$.

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. In Section 3 we present some preliminary results on the geometry of the Voronoi cell that will play an important role in the design and analysis of our algorithms. In Section 4 we present the basic variant of our algorithm with complexity

$\tilde{O}(2^{3.5n})$, while in Section 5 we show how to improve the time complexity to $\tilde{O}(2^{2n})$. Finally, Section 6 concludes with a discussion of open problems and directions for future research.

1.2 Related work

A preliminary version of this work appears in the proceedings of STOC 2010 [MV10a]. 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 [Mic01a, FM03, Reg04, CM06, AKKV12, KPV12], 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 factors [AKKV12] (and quasi NP-hard for certain sub-polynomial factors [KPV12]), and it can be approximated in polynomial time within a factor $O(\sqrt{n/\log n})$ [AR05], at least in its distance estimation variant. In this paper 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 solve the actual CVP instance.

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 [AEVZ02, SFS09]. Sometimes one wants to generate the list of vertices (i.e., zero dimensional faces), or even a complete description including all faces in dimension 1 to $n-1$. This is done in [VB96, SSV09], but it is a much more complex problem, as in general the Voronoi cell can have as many as $(n+1)! = n^{\Omega(n)}$ vertices, so they cannot be computed in single exponential time.

A related problem is that of computing the covering radius of a lattice, i.e., the radius of the smallest sphere containing the Voronoi cell. Our CVP algorithm allows to approximate the covering radius within a factor 2 in single exponential time using a (randomized polynomial time) reduction from [GMR05]. In principle, our Voronoi cell computation algorithm could also be used to compute the exact value of the covering radius by enumerating all the vertices of the Voronoi cell and selecting the longest. However, this would not lead to a single exponential time algorithm because the number of vertices of a Voronoi cell 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 [HR06] 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 [Mic04, GMR05] for small approximation factors. Counting the number of vertices of the Voronoi cell [SSV09] or the number of lattice points of a given length [Cha07] is also known to be $\#P$-hard.

In this paper we only consider lattice problems with respect to the Euclidean norm ℓ_2. By linearily, all results trivially extend to ellipsoidal norms. Recently, [DPV11] gave a simple deterministic reduction from SVP in any ℓ_p norm to a single exponential number of CVP computations in the Euclidean norm. Together with our CVP algorithm, this yields a deterministic single exponential time solution to SVP, generalizing our SVP result from ℓ_2 to any ℓ_p norm.[3] The methods of [DPV11] only provide exact solution to SVP, and the problem of generalizing our CVP result from ℓ_2 (or ellipsoidal) to other norms (even using randomized algorithms) remains open.

2 Preliminaries

In this section we give some background about lattices. For a more in-depth introduction to lattices and algorithmic problems associated with them, see [MG02, NV09]. 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., $\mathbf{B}$) 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 $a \in \mathbb{R}$, let $S + \mathbf{x} = \{\mathbf{y} + \mathbf{x} \colon \mathbf{y} \in S\}$ and $aS = \{a\mathbf{y} \colon \mathbf{y} \in S\}$. The Euclidean (or 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 $\text{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$. The linear space spanned by a set

[3] In fact, [DPV11] shows how to solve SVP with respect to any norm, not just ℓ_p, but the general reduction for arbitrary norms is randomized, and results in a probabilistic SVP algorithm.

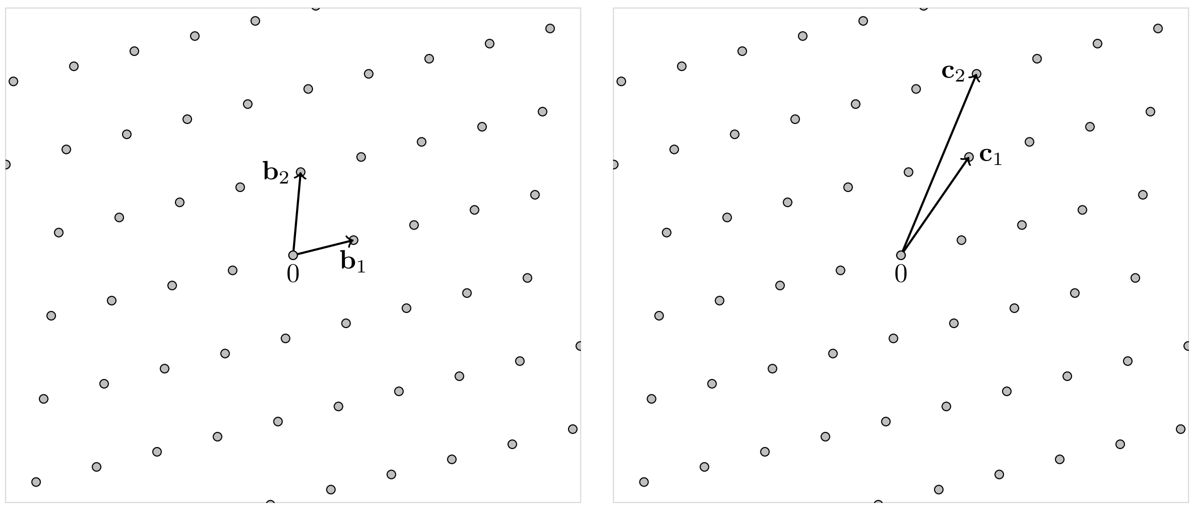

Figure 2: (Left) A two dimensional lattice generated by the basis $[\mathbf{b}_1, \mathbf{b}_2]$. The lattice is the set of all integer linear combinations of the basis vectors. (Right) A different basis $[\mathbf{c}_1, \mathbf{c}_2]$ for the same lattice. Each basis is the result of applying an integer linear transformation to the other. The bases in the figure satisfy $\mathbf{c}_1 = \mathbf{b}_1 + \mathbf{b}_2$, $\mathbf{c}_2 = \mathbf{b}_1 + 2\mathbf{b}_2$ and $\mathbf{b}_1 = 2\mathbf{c}_1 - \mathbf{c}_2$, $\mathbf{b}_2 = \mathbf{c}_2 - \mathbf{c}_1$. So, $[\mathbf{b}_1, \mathbf{b}_2]$ and $[\mathbf{c}_1, \mathbf{c}_2]$ generate the same lattice.

of vectors S is $\text{span}(S) = \{\sum_i x_i \mathbf{s}_i \;:\; x_i \in \mathbb{R}, \mathbf{s}_i \in S\}$. The affine span of a set of vectors S is defined as $\mathbf{x} + \text{span}(S - \mathbf{x})$ for any $\mathbf{x} \in S$, and does not depend on the choice of $\mathbf{x}$. Any vector $\mathbf{t}$ can be written uniquely as the sum $\mathbf{t} = \pi_S(\mathbf{t}) + \pi_S^\perp(\mathbf{t})$ of two vectors such that $\pi_S(\mathbf{t}) \in \text{span}(S)$ and $\pi_S^\perp(\mathbf{t})$ is orthogonal to $\text{span}(S)$. The vector $\pi_S^\perp(\mathbf{t})$ is called the component of $\mathbf{t}$ orthogonal to $\text{span}(S)$.

Lattices A *lattice* Λ is the set of all integer linear combinations

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

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, and the integer n is called the lattice *rank* or *dimension.* (See Figure 2 for a 2-dimensional example.) 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 $\mathcal{L}(\mathbf{S}) \subseteq \mathcal{L}(\mathbf{B})$. We use the convention that when a basis matrix $\mathbf{B}_n$ is written with a subscript, the subscript n represents the dimension of the lattice, i.e., the number of basis vectors $\mathbf{B}_n = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$. When the same matrix is used with a different subscript $k \leq n$, it represents the basis $\mathbf{B}_k = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ of the sublattice generated by the first k vectors of $\mathbf{B}_n$. The Gram-Schmidt orthogonalization of a basis $\mathbf{B}_n$ is the sequence of vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$, where $\mathbf{b}_i^* = \pi_{\mathbf{B}_{i-1}}^\perp(\mathbf{b}_i)$ is the component of $\mathbf{b}_i$ orthogonal to $\text{span}(\mathbf{B}_{i-1})$.

Two fundamental quantities associated to any lattice Λ are its minimum distance $\lambda(\Lambda) = \inf_{\mathbf{x} \in \Lambda \setminus \{\mathbf{0}\}} \|\mathbf{x}\|$ and covering radius $\mu(\Lambda) = \sup_{\mathbf{t} \in \text{span}(\Lambda)} \inf_{\mathbf{v} \in \Lambda} \text{dist}(\mathbf{t}, \mathbf{x})$. The dual $\Lambda^\dagger$ of a lattice Λ is the set of all the vectors $\mathbf{x}$ in the linear span of Λ that have integer scalar product $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_i x_i \cdot y_i \in \mathbb{Z}$ with all lattice vectors $\mathbf{y} \in \Lambda$. Banaszczyk's transference theorem [Ban93] shows that the minimum distance and covering radius of a lattice and its dual are related by the bound $1 \leq 2\lambda(\Lambda) \cdot \mu(\Lambda^\dagger) \leq n$.

The following classical lattice algorithms are used in this paper. The Nearest Plane algorithm [Bab86], 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 [LLL82], on input a lattice basis, outputs a basis for the same lattice such that $\|\mathbf{b}_{i+1}^*\|^2 \geq \|\mathbf{b}_i^*\|^2/2$ for all i. (LLL reduced bases have other properties, but this is all we need here.) Both algorithms run in polynomial time.

In Section 5 we also need block reduction algorithms [GN08, Sch87] to compute bases with stronger reduction properties than LLL. In particular, we use the *slide reduction* algorithm of [GN08], which, for any "block size" parameter $k \leq n$, requires a polynomial number of SVP computations on lattices of dimension at most k. As for LLL, the exact definition of slide reduced basis is not relevant here, and all we need to know is that the first vector of a slide reduced basis satisfies $\|\mathbf{b}_1\| \leq O(k)^{(n-k)/(k-1)} \cdot \lambda(\mathcal{L}(\mathbf{B}_n))$. This is proved in [GN08] assuming that the lattice dimension n is an integer multiple of the block size k, but it is easy to verify that the analysis in [GN08] works for any n, yielding the bound $\|\mathbf{b}_1\| \leq O(k)^{(k\lceil n/k\rceil-k)/(k-1)} \cdot \lambda(\mathcal{L}(\mathbf{B}_n))$. (Notice that this more general bound specializes to $O(k)^{(n-k)/(k-1)}$ when n/k is an integer.) We will use the slide reduction algorithm of [GN08] with block size $k = \lceil n/2\rceil$, which yields a basis $\mathbf{B}_n$ satisfying $\|\mathbf{b}_1\| \leq O(n) \cdot \lambda(\mathcal{L}(\mathbf{B}_n))$. Combining this bound with Banaszczyk's transference theorem [Ban93] we obtain $\|\mathbf{b}_1\| \cdot \mu(\mathcal{L}(\mathbf{B}_n)^\dagger) \leq O(n^2)$. Equivalently, applying the slide reduction algorithm to the dual lattice $\Lambda^\dagger$, rather than Λ, one obtains a basis $\mathbf{B}_n$ for $\Lambda = \mathcal{L}(\mathbf{B}_n)$ such that $\mu(\mathcal{L}(\mathbf{B}_n))/\|\mathbf{b}_n^*\| \leq O(n^2)$. Finally, we observe that (similarly to LLL) if a basis $\mathbf{B}_n = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ is slide reduced, then also the projection of $\mathbf{B}$ onto the orthogonal complement of $\mathbf{B}_i = [\mathbf{b}_1, \ldots, \mathbf{b}_i]$ is also slide reduced (for any i and the same block size k.) In particular, applying the reduction algorithm with $k = \lceil n/2\rceil$ to the dual lattice, one obtains a basis $\mathbf{B}_n$ such that $\mu(\mathcal{L}(\mathbf{B}_i))/\|\mathbf{b}_i^*\| \leq O(n^2)$ for all i. In the rest of the paper we use directly the last inequality as summarized in the following lemma, without any further reference to the dual lattice or Banaszczyk's theorem. For a full description of the slide reduction algorithm and the relation between the primal and dual lattice bases the reader is referred to [GN08].

Lemma 2.1 (Implicit in [GN08]) *There is a polynomial time algorithm that on input an n-dimensional lattice, and given oracle access to a procedure to solve SVP in lattices up to dimension $\lceil n/2\rceil$, outputs a basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ for the same lattice such that $\mu(\mathcal{L}([\mathbf{b}_1, \ldots, \mathbf{b}_i])/\|\mathbf{b}_i^*\| \leq O(n^2)$ for all $i = 1, \ldots, n$.*

Lattice problems In this paper we are mostly concerned with the Closest Vector Problem (CVP) defined in the introduction: given a lattice basis $\mathbf{B}$ and a target vector $\mathbf{t}$, find a lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ that minimizes the distance $\|\mathbf{t} - \mathbf{v}\|$. Our results give algorithms for several other lattice problems like the Shortest Vector Problem (SVP), Shortest Independent Vectors Problem (SIVP), Subspace Avoiding Problem (SAP), the Generalized Closest Vector Problem (GCVP), and the Successive Minima Problem (SMP) considered in the lattice algorithms literature [BN09, Mic08]. The results for all problems other than CVP can be obtained obtained in a black-box way by (dimension preserving) reduction to CVP [Mic08], and we refer the reader to [GMSS99, BN09, Mic08] for details. Also, using our new algorithm to implement the SVP oracle within the block reduction algorithms of [GN08, Sch87], yields *deterministic* polynomial time approximations for SVP, SIVP, CVP, etc. within a factor $2^{O(n \log\log n/\log n)}$. Previous polynomial time algorithms either used randomization, or achieved a weaker approximation factor $2^{O(n(\log\log n)^2/\log n)}$.

For simplicity we assume that the input lattices $\mathbf{B} \in \mathbb{Z}^{d\times n}$ and target vectors $\mathbf{t} \in \mathbb{Z}^d$ have integer entries with bit-size polynomial in the lattice dimension n, and that the number of entries d in each vector is also polynomial in n. This allows to express the complexity of lattice problems simply as a function of a single parameter, the lattice dimension n. All the results in this paper can be easily adapted to the general case by introducing an explicit bound $\log |b_{i,j}| \leq M$ on the size of the entries, and letting the time and space complexity bounds depend polynomially on M and d. We write $f = \tilde{O}(g)$ 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.

Coding conventions In order to make the description of our algorithms both precise, but also intuitive and easy to understand and analyze, we have used pseudocode written whenever possible in a high level, mathematical/functional style. We use two different notations for variable assignments, $x = \ldots$ for immutable variables that do not change their values during the execution, and $x \leftarrow \ldots$ for destructive assignment in stateful computations where the value of x is updated. Iteration is often expressed as tail recursion, in order to facilitate inductive proofs of correctness. Some of the loops are written in the form "for $x \in A$ sorted by $f(x)$", where x is a variable, A a set, and $f(x)$ an integer or real valued function. This represents a loop over all elements of the set A, in order of nondecreasing value of $f(x)$. Enumerating the elements of A in order of nondecreasing value of $f(x)$ is either straightforward, or (when A is finite) it can be easily achieved using any efficient sorting algorithm running in $O(|A| \log |A|)$ time. The body of loops where the set A is (coutably) infinite is executed indefinitely, until some exit condition is satisfied, and the function terminates by executing a "return" instruction.

3 The geometry of the Voronoi cell

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 use the closed cell

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

which is the topological closure of $\mathcal{V}$. We omit Λ, and simply write $\mathcal{V}$ or $\bar{\mathcal{V}}$ when the lattice is clear from the context. The Voronoi cell of a lattice point $\mathbf{v} \in \Lambda$ is defined similarly, and equals $\mathbf{v} + \mathcal{V}$. (See Figure 1.) For any (lattice) point $\mathbf{v}$, define the half-space

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

We also use notation $\bar{H}_{\mathbf{v}} = \{\mathbf{x} : \|\mathbf{x}\| \leq \|\mathbf{x} - \mathbf{v}\|\}$ for the closed half-space and $H^o_{\mathbf{v}} = \bar{H}_{\mathbf{v}} \setminus H_{\mathbf{v}} = \{\mathbf{x} : \|\mathbf{x}\| = \|\mathbf{x} - \mathbf{v}\|\}$ for the associated hyperplane. 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 such $H_{\mathbf{v}}$. 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 and it is denoted $V(\Lambda)$. The Voronoi cell $\mathcal{V}$ is a polytope, and the Voronoi relevant vectors are precisely the centers of the ($(n-1)$-dimensional) facets of $2\mathcal{V}$. The following observation connects the Voronoi cell, solving CVP and finding shortest vectors in lattice cosets $\Lambda + \mathbf{t}$. (See Figure 3.) These formulations of CVP will be used interchangeably throughout the paper.

Observation 3.1 *Let Λ be a lattice, $\mathcal{V}$ its Voronoi cell and $\mathbf{t}$, $\mathbf{t}'$ two vectors. The following statements are equivalent:*

1. *$\mathbf{t}'$ is a shortest vector in the coset $\Lambda + \mathbf{t}$*
2. *$\mathbf{t}'$ belongs to $(\Lambda + \mathbf{t}) \cap \bar{\mathcal{V}}$.*
3. *$\mathbf{v} = \mathbf{t} - \mathbf{t}' \in \Lambda$ is a lattice vector closest to $\mathbf{t}$.*

Proof: We prove that the three statements are equivalent by demonstrating the chain of implications $1 \to 2$, $2 \to 3$, $3 \to 1$. We start with the implication $1 \to 2$. Assume $\mathbf{t}'$ belongs to $\Lambda + \mathbf{t}$ and it is a shortest vector within this set. Notice that for all $\mathbf{w} \in \Lambda$, $\mathbf{t}' - \mathbf{w} \in \Lambda + \mathbf{t}$. Therefore $\|\mathbf{t}'\| \leq \|\mathbf{t}' - \mathbf{w}\|$ for all $\mathbf{w} \in \Lambda$, which is exactly the definition of the closed Voronoi cell. So $\mathbf{t}' \in \bar{\mathcal{V}}$.

Next, we prove $2 \to 3$. If $\mathbf{t}' \in (\Lambda + \mathbf{t}) \cap \bar{\mathcal{V}}$, then $\mathbf{v} = \mathbf{t} - \mathbf{t}'$ is a lattice vector and $\mathbf{t} = \mathbf{v} + \mathbf{t}' \in \bar{\mathcal{V}} + \mathbf{v}$. By definition of the Voronoi cell, $\mathbf{v}$ is a lattice vector closest to $\mathbf{t}$.

Finally, we show that $3 \to 1$. If $\mathbf{v}$ is a lattice vector closest to $\mathbf{t}$, then $\|\mathbf{t} - \mathbf{v}\| \leq \|\mathbf{t} - \mathbf{w}\|$ for all $\mathbf{w} \in \Lambda$. Notice that $\{\mathbf{t} - \mathbf{w} : \mathbf{w} \in \Lambda\} = \Lambda + \mathbf{t}$, so $\mathbf{t}' = \mathbf{t} - \mathbf{v}$ is a shortest vector in the coset $\Lambda + \mathbf{t}$. ■

Our algorithms for computing the Voronoi cell of a lattice are based on the following classical theorem of Voronoi.

Theorem 3.2 (Voronoi, see [CS98]) *Let Λ be a lattice and $\mathbf{v} \in \Lambda$ any lattice vector. Then $\mathbf{v}$ is Voronoi relevant if and only if $\pm\mathbf{v}$ are the only two shortest vectors in the coset $2\Lambda + \mathbf{v}$.*

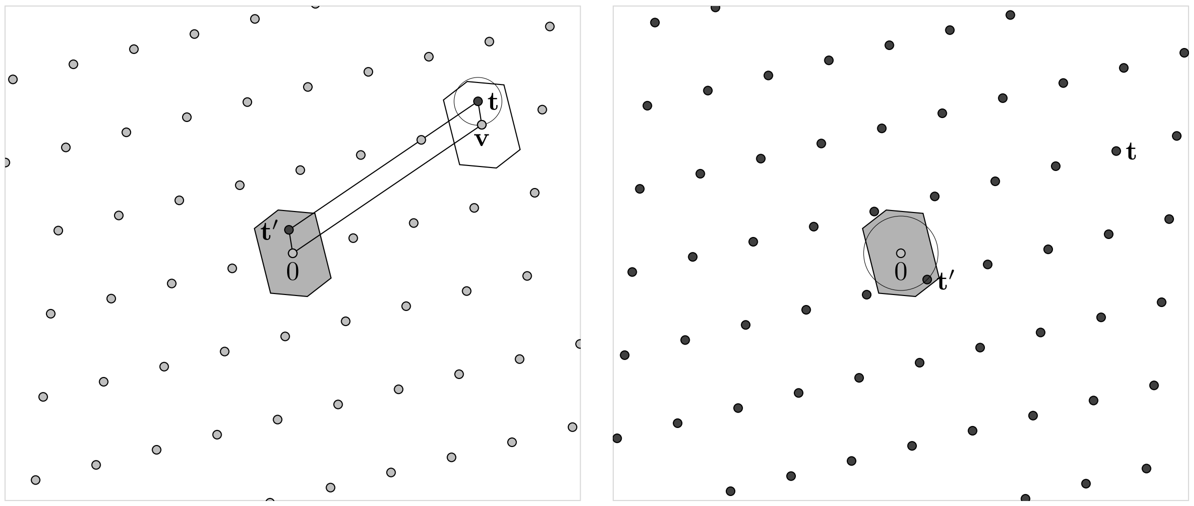

Figure 3: (Left) Finding a lattice point $\mathbf{v}$ closest to a target $\mathbf{t}$ is equivalent to finding a point $\mathbf{t}' \in \mathbf{t} + \Lambda$ that belongs to the Voronoi cell of the lattice (shaded). If $\mathbf{t}'$ is in the Voronoi cell, then $\mathbf{v} = \mathbf{t} - \mathbf{t}'$ is the lattice point closest to $\mathbf{t}$. (Right) Coset formulation of the Closest Vector Problem: given a lattice Λ and a target $\mathbf{t}$, find a shortest vector $\mathbf{t}'$ in the lattice coset $\mathbf{t} + \Lambda$. The vector $\mathbf{t}'$ is shortest in the coset $\mathbf{t} + \Lambda = \mathbf{t}' + \Lambda$ if and only if it belongs to the Voronoi cell of the lattice.

In order to analyze the faster algorithm in Section 5 we also need the following theorem of Horváth about the lattice points on the boundary of $2\bar{\mathcal{V}}$.

Theorem 3.3 ([Hor96, Thm. 4]) *For any lattice Λ, any lattice point $\mathbf{u} \in \Lambda$ on the boundary of $2\bar{\mathcal{V}}(\Lambda)$ can be written as a sum of mutually orthogonal relevant vectors.*

In the rest of this section we prove several geometric and combinatorial properties of lattices and their Voronoi cells that will be used in the design and analysis of our algorithms. We start from two simple symmetries of the Voronoi cell.

Lemma 3.4 *Let $\bar{\mathcal{V}}$ be the closed Voronoi cell of a lattice Λ and $\mathbf{t} \in \bar{\mathcal{V}}$ a point of the closed Voronoi cell that lies on the hyperplane $H^o_{\mathbf{v}} = \{\mathbf{t} \colon \|\mathbf{t}\| = \|\mathbf{t} - \mathbf{v}\|\}$ defined by $\mathbf{v} \in \Lambda$. Then $\mathbf{t} - \mathbf{v} \in \bar{\mathcal{V}}$ also belongs to the Voronoi cell.*

Proof: By Observation 3.1 (with $\mathbf{t}' = \mathbf{t}$) we get that $\mathbf{t} \in \bar{\mathcal{V}}$ is a shortest vector in the coset $\Lambda + \mathbf{t}$. Notice that the cosets $\Lambda + \mathbf{t} - \mathbf{v}$ and $\Lambda + \mathbf{t}$ are identical because $\mathbf{v} \in \Lambda$. Also, by assumption, $\|\mathbf{t} - \mathbf{v}\| = \|\mathbf{t}\|$. We conclude that $\mathbf{t} - \mathbf{v}$ too is a shortest vector in the coset $\Lambda + (\mathbf{t} - \mathbf{v})$ and by Observation 3.1 it belongs to $\bar{\mathcal{V}}$. ∎

Lemma 3.5 *Let $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{u} \in \bar{\mathcal{V}}$ be four points in the Voronoi cell of a lattice $\bar{\mathcal{V}} = \bar{\mathcal{V}}(\Lambda)$ such that $\mathbf{z}$ belongs to the segment joining $\mathbf{x}$ and $\mathbf{y}$ (i.e., $\mathbf{z} = \delta \cdot \mathbf{x} + (1 - \delta)\mathbf{y}$ for some $0 < \delta < 1$) and $\mathbf{z} - \mathbf{u} \in \Lambda$. Then, $\mathbf{u} + \mathbf{y} - \mathbf{z} \in \bar{\mathcal{V}}$.*

Proof: Let $\mathbf{v} = \mathbf{z} - \mathbf{u} \in \Lambda$. By convexity, since $\mathbf{x}, \mathbf{y} \in \bar{\mathcal{V}} \subset \bar{H}_{\mathbf{v}}$, we also have $\mathbf{z} \in \bar{\mathcal{V}} \subset \bar{H}_{\mathbf{v}}$, i.e., $\|\mathbf{z}\| \leq \|\mathbf{z} - \mathbf{v}\|$. Since $\mathbf{z} - \mathbf{v} = \mathbf{u} \in \bar{\mathcal{V}} \subset \bar{H}_{-\mathbf{v}}$, we also have $\|\mathbf{z} - \mathbf{v}\| \leq \|(\mathbf{z} - \mathbf{v}) - (-\mathbf{v})\| = \|\mathbf{z}\|$. So, $\|\mathbf{z}\| = \|\mathbf{z} - \mathbf{v}\|$. Notice that if $\mathbf{y} \in H_{\mathbf{v}}$ belonged to the interior of the halfspace, then also $\mathbf{z}$ would be in $H_{\mathbf{v}} = \bar{H}_{\mathbf{v}} \setminus H^o_{\mathbf{v}}$. Since $\mathbf{z}$ is on the hyperplane $H^o_{\mathbf{v}} = \{\mathbf{z} \colon \|\mathbf{z}\| = \|\mathbf{z} - \mathbf{v}\|\}$, it must be $\mathbf{y} \in H^o_{\mathbf{v}}$ too, and by Lemma 3.4 the point $\mathbf{y} - \mathbf{v} = \mathbf{y} + \mathbf{u} - \mathbf{z}$ belongs to $\bar{\mathcal{V}}$. ∎

The main step to reduce the norm of the vectors in our CVPP algorithm is based on the following lemma. The lemma admits a natural geometric interpretation. (See Figure 4 (Left).) The quantity $\alpha = \max_{\mathbf{v}\in V(\Lambda)} 2\langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2$ is the smallest positive real such that $\mathbf{t} \in \alpha\bar{\mathcal{V}}$. So, the vector $\mathbf{t}$ belongs to the boundary of $\alpha\bar{\mathcal{V}}$. The relevant vector $\mathbf{v} \in \Lambda$ achieving the maximum $\alpha = 2\langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2$ is the one corresponding to the facet of $\alpha\bar{\mathcal{V}}$ that $\mathbf{t}$ belongs to. For example, in Figure 4 (Left), the relevant vector that maximizes α for the point $\mathbf{t} = \mathbf{t}_0$ is $\mathbf{v}_2$, while the relevant vector that maximizes α for $\mathbf{t}_1$ is $\mathbf{v}_4$. The next lemma states that if we subtract $\mathbf{v}_2$ from $\mathbf{t}$ (or $\mathbf{v}_4$ from $\mathbf{t}_1$), we get a strictly shorter vector that still belongs to $\alpha\bar{\mathcal{V}}$.

Lemma 3.6 *For all $\mathbf{t} \notin \bar{\mathcal{V}}(\Lambda)$, if $\mathbf{v} \in V(\Lambda)$ is a relevant vector that maximizes the quantity $\alpha = 2\langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2$, then $\|\mathbf{t}-\mathbf{v}\| < \|\mathbf{t}\|$, and $\mathbf{t}-\mathbf{v} \in \alpha\bar{\mathcal{V}}(\Lambda)$.*

Proof: Let $\mathbf{v} \in V(\Lambda)$ be such that $\alpha = 2\langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2$ is maximized. Then, for all $\mathbf{u} \in V(\Lambda)$ we have $2\langle \mathbf{t},\mathbf{u}\rangle/\|\mathbf{u}\|^2 \le \alpha$, or equivalently $\|\mathbf{t}\| \le \|\mathbf{t}-\alpha\mathbf{u}\|$, i.e., $\mathbf{t} \in \bar{\mathcal{V}}(\alpha\Lambda)$. Since $\mathbf{t} \notin \bar{\mathcal{V}}(\Lambda)$, it must be $\alpha > 1$. From $\alpha = 2\langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2$ we also get that $\|\mathbf{t}\| = \|\mathbf{t}-\alpha\mathbf{v}\|$, i.e., $\mathbf{t}$ is on the hyperplane $H^o_{\alpha\mathbf{v}}$. Therefore, by Lemma 3.4 (applied to lattice $\alpha\Lambda$) $\mathbf{t}-\alpha\mathbf{v}$ belongs to $\bar{\mathcal{V}}(\alpha\Lambda)$. Given that both $\mathbf{t}$ and $\mathbf{t}-\alpha\mathbf{v}$ are in $\bar{\mathcal{V}}(\alpha\Lambda)$ and $\alpha > 1$, by convexity of the Voronoi cell we have that $\mathbf{t}-\mathbf{v}$ is also in $\bar{\mathcal{V}}(\alpha\Lambda)$. Finally, using $2\langle \mathbf{t},\mathbf{v}\rangle = \alpha\|\mathbf{v}\|^2$, we get

$$\|\mathbf{t}-\mathbf{v}\|^2 = \|\mathbf{t}\|^2 + \|\mathbf{v}\|^2 - 2\langle \mathbf{t},\mathbf{v}\rangle = \|\mathbf{t}\|^2 - (\alpha-1)\|\mathbf{v}\|^2 < \|\mathbf{t}\|^2,$$

where we have used the fact that $\alpha > 1$ and $\mathbf{v} \neq \mathbf{0}$. ■

The next lemma shows that the vectors of a lattice coset $\mathbf{t}+\Lambda_n$ inside a scaled Voronoi cell $k\bar{\mathcal{V}}(\Lambda_n)$ belong to at most k^n distinct spherical shells $\mathcal{S}_\alpha = \{\mathbf{x}\colon \|\mathbf{x}\| = \alpha\}$.

Lemma 3.7 *For any lattice Λ_n, vector $\mathbf{t}$, integer $k \ge 1$ and $U \subseteq (\mathbf{t}+\Lambda_n) \cap k\bar{\mathcal{V}}(\Lambda_n)$, we have $|\{\|\mathbf{u}\| \mid \mathbf{u} \in U\}| \le k^n$.*

Proof: By Observation 3.1 (applied to the scaled lattice $k\Lambda_n$ and its Voronoi cell $\mathcal{V}(k\Lambda_n) = k\cdot\mathcal{V}(\Lambda_n)$,) for any vector $\mathbf{v}$, all the points in $(k\Lambda_n+\mathbf{v}) \cap (k\bar{\mathcal{V}})$ have the same euclidean norm. Since $\Lambda_n+\mathbf{t}$ can be partitioned in precisely k^n cosets of the form $k\Lambda_n+\mathbf{v}$ with $\mathbf{v} \in \Lambda_n+\mathbf{t}$, and all vectors from $\bar{\mathcal{V}}(k\Lambda_n)$ in each coset belong to the same shell, the total number of shells is at most k^n. ■

In particular, when $k=1$, all points in $U = (\Lambda+\mathbf{t}) \cap \bar{\mathcal{V}}(\Lambda)$ are all on the same shell, as we already knew from Observation 3.1. The last lemma give some additional properties of the vectors in U.

Lemma 3.8 *For any lattice Λ, target vector $\mathbf{t}$ and any two vectors $\mathbf{u},\mathbf{u}'$ in the set $U = (\Lambda+\mathbf{t}) \cap \bar{\mathcal{V}}(\Lambda)$, there is a set of (mutually orthogonal) relevant vectors $\mathbf{v}_1,\ldots,\mathbf{v}_k \in V(\Lambda)$ such that $\mathbf{u}' = \mathbf{u}+\sum_{i=1}^k \mathbf{v}_i$ and $\mathbf{u}+\sum_{i\in S}\mathbf{v}_i \in U$ for all $S \subseteq \{1,\ldots,n\}$.*

Proof: The proof proceeds in two steps. First we show that for any such $\mathbf{u},\mathbf{u}'$, there is a set of $k \le n$ mutually orthogonal *relevant* vectors $\{\mathbf{v}_1,\ldots,\mathbf{v}_k\} \subseteq V(\Lambda)$ such that $\mathbf{u}' = \mathbf{u}+\sum_{i=1}^k \mathbf{v}_i$. Next, we show that whenever there is such a set of mutually orthogonal relevant vectors, $\mathbf{u}+\sum_{i\in S}\mathbf{v}_i \in U$ for all $S \subseteq \{1,\ldots,n\}$.

By Observation 3.1, $\mathbf{u},\mathbf{u}'$ are both shortest vectors of $\Lambda+\mathbf{t}$. In particular, they have the same length $\|\mathbf{u}\| = \|\mathbf{u}'\|$. Their difference $\mathbf{u}-\mathbf{u}'$ belongs to Λ because $\mathbf{u},\mathbf{u}' \in \Lambda+\mathbf{t}$ are in the same lattice coset. Also, $\mathbf{u}-\mathbf{u}' \in \bar{\mathcal{V}}-\bar{\mathcal{V}} = 2\bar{\mathcal{V}}$. Therefore, $\mathbf{u}-\mathbf{u}' \in \Lambda \cap 2\bar{\mathcal{V}}$, and we may also assume that $\mathbf{u}-\mathbf{u}' \neq \mathbf{0}$ because otherwise the statement is trivial. Since the only lattice point in the open cell $2\mathcal{V}$ is the origin, $\mathbf{u}-\mathbf{u}'$ belongs to the boundary $2\bar{\mathcal{V}} \setminus 2\mathcal{V}$ and by Theorem 3.3 there exists a subset $\{\mathbf{v}_1,\ldots,\mathbf{v}_k\} \subseteq V(\Lambda)$ of mutually orthogonal relevant vectors such that $\mathbf{u}' = \mathbf{u}+\sum_{i=1}^k \mathbf{v}_i$. Since (nonzero) mutually orthogonal vectors are linearly independent, it must be $k \le n$. This concludes the first part of the proof.

For the second part, we prove, by induction on k, that if there are k mutually orthogonal relevant vectors such that $\mathbf{u}' = \mathbf{u}+\sum_{i=1}^k \mathbf{v}_i$, then $\mathbf{u}_S = \mathbf{u}+\sum_{i\in S}\mathbf{v}_i \in U$ for all $S \subseteq \{1,\ldots,n\}$. If $S = \emptyset$, then $\mathbf{u}_S = \mathbf{u} \in U$, and the statement is true. So, assume $S \neq \emptyset$, let $i \in S$ and define $S' = S \setminus \{i\}$. Below we prove that $\mathbf{u}'' = \mathbf{u}+\mathbf{v}_i \in U$. Since $\mathbf{u}' = \mathbf{u}''+\sum_{j\neq i}\mathbf{v}_j$, it follows by induction that $\mathbf{u}+\sum_{j\in S}\mathbf{v}_j = \mathbf{u}''+\sum_{j\in S'}\mathbf{v}_j \in U$

We need to show that $\mathbf{u} + \mathbf{v}_i$ belongs to U. Using the orthogonality of the relevant vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$, we get

$$\begin{aligned}
\sum_{i=1}^{k} \|\mathbf{u} + \mathbf{v}_i\|^2 &= k \cdot \|\mathbf{u}\|^2 + 2\langle \mathbf{u}, \sum_{i=1}^{k} \mathbf{v}_i \rangle + \left\| \sum_{i=1}^{k} \mathbf{v}_i \right\|^2 \\
&= (k-1) \cdot \|\mathbf{u}\|^2 + \left\| \mathbf{u} + \sum_{i=1}^{k} \mathbf{v}_i \right\|^2 \\
&= (k-1) \cdot \|\mathbf{u}\|^2 + \|\mathbf{u}'\|^2 = k \cdot \|\mathbf{u}\|^2.
\end{aligned}$$

Notice that the vectors $\mathbf{u}+\mathbf{v}_i$ are in $\Lambda+\mathbf{t}$, and by the minimality of $\|\mathbf{u}\|$, their norm is at least $\|\mathbf{u}+\mathbf{v}_i\| \geq \|\mathbf{u}\|$. We conclude that $\|\mathbf{u} + \mathbf{v}_i\| = \|\mathbf{u}\|$ for all i. So the vectors $\mathbf{u} + \mathbf{v}_i$ are also shortest vectors of $\Lambda + \mathbf{t}$, and by Observation 3.1 they belong to U. ■

4 The basic algorithm

In this section we describe and analyze a $\tilde{O}(2^{3.5n})$-time algorithm for computing the Voronoi cell of a lattice and solving several related lattice problems. The pseudocode is given by a collection of functions in Algorithms 1, 2 and 3. The running time is improved to $\tilde{O}(2^{2n})$ in Section 5. We describe the slower algorithm first as it allows for a more modular description, it better illustrates the connections with previous work, and the faster algorithm uses it to preprocess the lattice. The algorithm presented in this section has three components:

1. A $\tilde{O}(2^{2n})$-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 list of relevant vectors.

2. A *preprocessing* function and *rank reduction* procedure such that for any lattice basis $\mathbf{B}_n$ produced by the preprocessing function, and for any $k \leq n$, the rank reduction procedure solves CVP in $\Lambda_k = \mathcal{L}(\mathbf{B}_k)$ making at most $2^{k/2}$ calls to a CVP oracle for the lower-dimensional sublattice $\Lambda_{k-1} = \mathcal{L}(\mathbf{B}_{k-1})$.

3. A reduction from the problem of computing the *Voronoi cell* of a lattice Λ_n to 2^n CVP computations, all on the same input lattice Λ_n.

These three components are described and analyzed in Subsections 4.1, 4.2, 4.3. In Subsection 4.3 we also show how these components are combined together into an algorithm for Voronoi cell computation with running time $\tilde{O}(2^{3.5n})$.

Notice that the rank reduction procedure immediately gives a recursive algorithm to solve CVP (and, by reduction, also compute the Voronoi cell) for arbitrary lattices. However, the obvious way to turn the rank reduction procedure into a recursive program results in an algorithm with $2^{O(n^2)}$ running time. This is 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 algorithm to give a more efficient transformation. The idea is to compute the Voronoi cells of all sub-lattices $\Lambda_k = \mathcal{L}(\mathbf{B}_k)$ sequentially for $k = 1, \ldots, n$, where $\mathbf{B}_n$ is the lattice basis produced by the preprocessing function. Each Voronoi cell $\mathcal{V}(\Lambda_k)$ is computed by (rank) reduction to $2^{O(k)}$ CVP computations in the lower-dimensional lattice Λ_{k-1}. In turn, these CVP computations are performed using the CVPP algorithm with the help of the previously computed Voronoi cell $\mathcal{V}(\Lambda_{k-1})$. This allows to compute all the Voronoi cells $\mathcal{V}(\Lambda_k)$ (for $k = 1, \ldots, n$) in time $\sum_{k=1}^{n} 2^{O(k)} = 2^{O(n)}$.

4.1 CVP with preprocessing

We give a $\tilde{O}(2^n \cdot |V_n|)$-time algorithm to solve the closest vector problem with preprocessing, given as input the list V_n of Voronoi relevant vectors of a lattice Λ_n. Throughout this section, we use the coset formulation

Algorithm 1 Single exponential time algorithm for CVPP

```
function CVPP($\mathbf{t}, V_n$)
    $\alpha = 2\max_{\mathbf{v}\in V_n} \langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2$
    if $\alpha > 1$ then
        $\mathbf{t}' = $ CVPP2($\mathbf{t}, \lceil \alpha/2 \rceil V_n$)
        return CVPP($\mathbf{t}', V_n$)
    else return $\mathbf{t}$

function CVPP2($\mathbf{t}, V_n$)
    $\mathbf{v} = \mathrm{argmax}_{\mathbf{v}\in V_n} 2\langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2$
    if $2\langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2 \le 1$ then return $\mathbf{t}$
    else return CVPP2($\mathbf{t}-\mathbf{v}, V_n$)
```

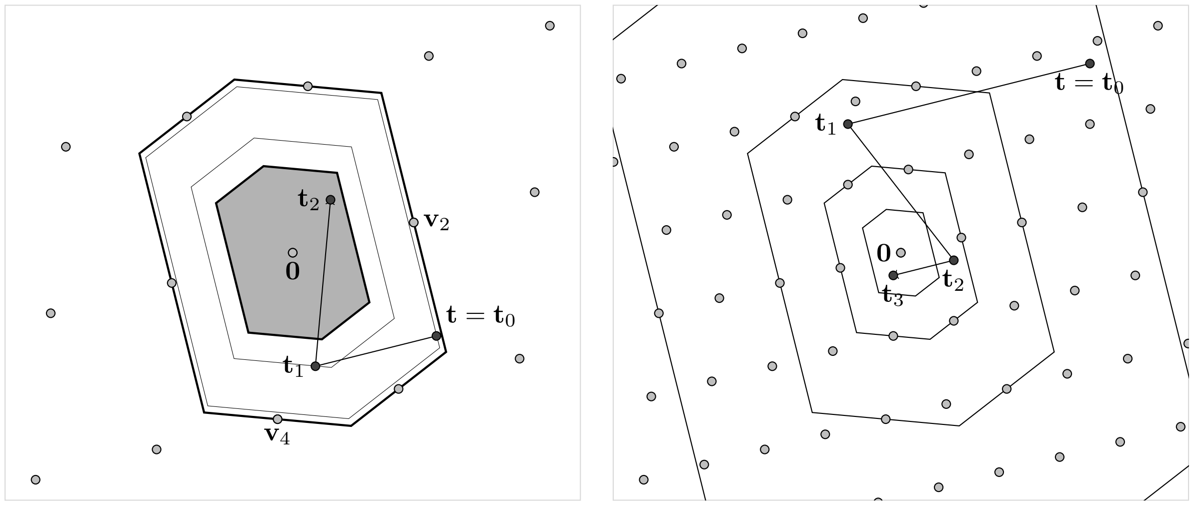

Figure 4: (Left) An execution of the CVPP2 algorithm. The gray points are the vectors of a lattice Λ. The shaded area is the Voronoi cell of the lattice. The lattice vectors $\mathbf{v}_2 = \mathbf{b}_1$ and $\mathbf{v}_4 = -\mathbf{b}_2$ are Voronoi relevant vectors. On input a vector $\mathbf{t}$ that belongs to twice the Voronoi cell of the lattice, the algorithm computes a sequence of points in the coset $\mathbf{t} + \Lambda$ (black points), repeatedly subtracting properly chosen relevant vectors: $\mathbf{t}_1 = \mathbf{t}_0 - \mathbf{v}_2$, $\mathbf{t}_2 = \mathbf{t}_1 - \mathbf{v}_4$. (Right) The CVPP problem for arbitrary targets $\mathbf{t}$ is solved by consindering integer multiples of the lattice $c\Lambda$ so that $\mathbf{t}$ belongs to twice the Voronoi cell of $c\Lambda$, and using CVPP2 repeatedly to bring $\mathbf{t}$ inside smaller and smaller integer multiples of the Voronoi cell: $\mathbf{t}_0 \in 2\mathcal{V}(4\Lambda)$, $\mathbf{t}_1 \in 2\mathcal{V}(2\Lambda)$, $\mathbf{t}_2 \in 2\mathcal{V}(\Lambda)$ and $\mathbf{t}_3 \in \mathcal{V}(\Lambda)$. The lattice point closest to the target $\mathbf{t}$ is $\mathbf{t} - \mathbf{t}_3 = 3\mathbf{b}_1 + 2\mathbf{b}_2$.

of CVP, where the goal is to find the shortest vector in a lattice coset $\mathbf{t} + \Lambda_n$, or, equivalently, a vector $\mathbf{t}' \in (\mathbf{t} + \Lambda_n) \cap \bar{\mathcal{V}}(\Lambda_n)$. (See Observation 3.1.) A lattice point $\mathbf{v} \in \Lambda_n$ closest to the target $\mathbf{t}$, if desired, can be easily computed from $\mathbf{t}'$ as $\mathbf{v} = \mathbf{t} - \mathbf{t}'$. The pseudocode is given in Algorithm 1, and consists of two functions:

- a basic function CVPP2 which solves the problem for the special case when $\mathbf{t} \in 2\bar{\mathcal{V}}(\Lambda_n)$; and
- a main function CVPP which solves the general problem with polynomially many calls to CVPP2.

Both functions are illustrated in Figure 4. We first analyze CVPP2.

Lemma 4.1 *Given the list $V_n = V(\Lambda_n)$ of relevant vectors of an n-dimensional lattice Λ_n, and a target point $\mathbf{t} \in 2\bar{\mathcal{V}}(\Lambda_n)$, the* CVPP2 *function from Algorithm 1 finds a vector in $(\mathbf{t} + \Lambda_n) \cap \bar{\mathcal{V}}(\Lambda_n)$ in time $\tilde{O}(|V_n| \cdot 2^n) \le \tilde{O}(2^{2n})$.*

Proof: Function CVPP2($\mathbf{t}, V_n$) first finds the relevant vector $\mathbf{v} \in V_n$ that maximizes the quantity $\alpha = 2\langle \mathbf{t},\mathbf{v}\rangle/\|\mathbf{v}\|^2$. If $\alpha \le 1$, then all vectors $\mathbf{w} \in V_n$ satisfy $2\langle \mathbf{t},\mathbf{w}\rangle \le \|\mathbf{w}\|^2$, or, equivalently, $\|\mathbf{t}\| \le \|\mathbf{t} - \mathbf{w}\|$. So, the input vector $\mathbf{t}$ belongs to the Voronoi cell of the lattice, and the function correctly returns $\mathbf{t}$. If $\alpha > 1$, then the function recursively invokes CVPP2$(\mathbf{t}', V_n)$ on target vector $\mathbf{t}' = \mathbf{t} - \mathbf{v} \in \mathbf{t} + \Lambda_n$. Notice that $\alpha \leq 2$ because $\mathbf{t} \in 2\bar{\mathcal{V}}(\Lambda_n)$. Moreover, by Lemma 3.6 the new target satisfies $\mathbf{t}' \in \alpha\bar{\mathcal{V}}(\Lambda_n) \subseteq 2\bar{\mathcal{V}}(\Lambda_n)$. So, $\mathbf{t}'$ is a valid input to CVPP2, and the correctness of the function immediately follows by computational induction, provided the recursion terminates.

It remains to bound the running time. Each recursive invocation of CVPP2 takes time $\tilde{O}(|V_n|)$ to scan the list V_n and select the vector $\mathbf{v}$. Moreover, from Lemma 3.6 we know that $\|\mathbf{t}'\| < \|\mathbf{t}\|$. So, the input vectors $\mathbf{t}$ passed to successive invocations of CVPP2 have strictly decreasing euclidean norm. Since they all belong to $(\mathbf{t}+\Lambda_n)\cap 2\bar{\mathcal{V}}(\Lambda_n)$, it follows from Lemma 3.7 (with $k = 2$) that the number of recursive invocations of CVPP2 is bounded by 2^n. This gives a total running time of $\tilde{O}(2^n \cdot |V_n|)$. ■

The next theorem shows how to solve CVPP for any target vector.

Theorem 4.2 *On input the list $V_n = V(\Lambda_n)$ of relevant vectors of an n-dimensional lattice Λ_n and a target point $\mathbf{t}$, the* CVPP *function from Algorithm 1 finds a vector in $(\mathbf{t}+\Lambda_n)\cap\bar{\mathcal{V}}(\Lambda_n)$ in time $\tilde{O}(|V_n|\cdot 2^n) \leq \tilde{O}(2^{2n})$.*

Proof: Let $V_n = V(\Lambda_n)$ be the relevant vectors of the input lattice. Function CVPP$(\mathbf{t}, V_n)$ starts by computing the smallest value α such that $\mathbf{t} \in \alpha\bar{\mathcal{V}}(\Lambda_n)$. If $\alpha \leq 1$, then the vector $\mathbf{t}$ belongs to the Voronoi cell $\bar{\mathcal{V}}(\Lambda_n)$, and the function correctly returns $\mathbf{t}$. If $\alpha > 1$, then the function invokes CVPP2$(\mathbf{t}, \lceil\alpha/2\rceil V_n)$. Consider the scaled sublattice $\Lambda_n' = \alpha'\Lambda_n \subseteq \Lambda_n$ for $\alpha' = \lceil\alpha/2\rceil$. Notice that $\mathbf{t} \in \alpha\bar{\mathcal{V}}(\Lambda_n) \subseteq 2\alpha'\bar{\mathcal{V}}(\Lambda_n) = 2\bar{\mathcal{V}}(\Lambda_n')$. So, $(\mathbf{t}, \alpha' V_n)$ is a valid input to function CVPP2 which (by Lemma 4.1) correctly returns a vector $\mathbf{t}' \in \mathbf{t} + \Lambda_n' \subseteq \mathbf{t} + \Lambda_n$ such that $\mathbf{t}' \in \bar{\mathcal{V}}(\Lambda_n')$. It follows by computational induction that the recursive call CVPP$(\mathbf{t}', V_n)$ returns the correct result.

Each recursive invocation of CVPP takes time $\tilde{O}(|V_n|)$ to scan the list V_n and determine α, plus the time to evaluate CVPP2, which, by Lemma 4.1, is $\tilde{O}(2^n \cdot |V_n|)$. So, the cost incurred for each recursive invocation of CVPP is $\tilde{O}(2^n \cdot |V_n|)$. In order to bound the depth of the recursion, we observe that the vector $\mathbf{t}'$ returned by CVPP2 (and passed as input to the next invocation of CVPP) satisfies $\mathbf{t}' \in \alpha'\bar{\mathcal{V}}(\Lambda_n)$. Now, if $\alpha \leq 2^k$, then $\alpha' = \lceil\alpha/2\rceil \leq 2^{k-1}$. So, after at most $\lceil\log_2 \alpha\rceil$ recursive calls, CVPP is invoked on a vector $\mathbf{t}$ such that $\alpha \leq 1$, terminating the recursion. Since α is polynomial in the euclidean length of the input vectors, the number of recursive calls is polynomial in n (in fact, linear in $\log\|\mathbf{t}\|$) and the total running time of CVPP is $\tilde{O}(2^n \cdot |V_n|)$. ■

We conclude this subsection with some remarks about the efficiency and implementation of our CVPP algorithm.

Remark 4.3 *The algorithms* CVPP *and* CVPP2 *are* space efficient*, in the sense that they use very little memory, beside that required to store the input list V_n of relevant vectors. The exponential space complexity of the main algorithm comes exclusively from the need to store the description of the Voronoi cells V_k as intermediate results of the computation, for use with* CVPP.

Remark 4.4 *We assumed that the list V_n given as input to* CVPP *and* CVPP2 *contains precisely the Voronoi relevant vectors only for simplicity. The algorithms still work even if they are given as input a larger list (such that $V(\Lambda_n) \subseteq V_n \subseteq \Lambda_n \setminus \{\mathbf{0}\}$) that may contain other nonzero lattice vectors. This can be useful to avoid the use of the* VFilter *function in Section 4.3.*

Remark 4.5 *The target vector $\mathbf{t}$ is only used to compute scalar products $\langle\mathbf{t}, \mathbf{v}\rangle$ with lattice vectors $\mathbf{v} \in V_n \subset \Lambda_n$, and these products are not affected by any component of the target $\mathbf{t}$ orthogonal to the linear span of the lattice. So, while for simplicity one may project the target onto* $\mathrm{span}(V_n) = \mathrm{span}(\Lambda_n)$ *before invoking our* CVPP/CVPP2 *functions, this is not necessary, and Algorithm 1 works as described even when the target $\mathbf{t}$ is not in* $\mathrm{span}(V_n) = \mathrm{span}(\Lambda_n)$.

4.2 Rank reduction

We describe a procedure that, given a basis $\mathbf{B}_n$ for an n-dimensional lattice Λ_n, reduces any CVP instance on lattice $\Lambda_n = \mathcal{L}(\mathbf{B}_n)$, to a bounded number of CVP instances on the lower-dimensional sublattice $\Lambda_{n-1} = \mathcal{L}(\mathbf{B}_{n-1})$. The number of CVP subproblems is bounded by the quantity $1 + 2\mu(\Lambda_n)/\|\mathbf{b}_n^*\|$ where $\mu(\Lambda_n)$ is the covering radius of Λ_n, so the efficiency of the reduction depends on the quality of the input basis. To make good use of the rank reduction procedure, we first preprocess the lattice basis, so that all the ratios $\mu(\Lambda_k)/\|\mathbf{b}_k^*\|$ are relatively small. The next lemma shows that the bound $2^{k/2}$ can be achieved simply by running the LLL basis reduction algorithm [LLL82].

Lemma 4.6 *Any LLL reduced basis $\mathbf{B}_n$ satisfies $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| < 2^{(k-2)/2}$ for all $k \leq n$, where $\Lambda_k = \mathcal{L}(\mathbf{B}_k)$, and $\mathbf{b}_k^*$ is the component of $\mathbf{b}_k$ orthogonal to* $\operatorname{span}(\Lambda_{k-1})$.

Proof: Notice that the covering radius $\mu(\Lambda_k)$ is at most $\rho_k = \frac{1}{2}\sqrt{\sum_{i=1}^{k} \|\mathbf{b}_i^*\|^2}$ because the set $\{\sum \mathbf{x}_i \mathbf{b}_i^* : -\frac{1}{2} \leq x_i < \frac{1}{2}\}$ is a fundamental region of Λ_k and it is contained in a ball of radius ρ_k. If $\mathbf{B}_n$ is LLL reduced then $2\|\mathbf{b}_{i+1}^*\|^2 \geq \|\mathbf{b}_i^*\|^2$ for all i and, by induction, $2^{k-i}\|\mathbf{b}_k^*\|^2 \geq \|\mathbf{b}_i^*\|^2$. Thus,

$$\rho_k = \frac{1}{2}\sqrt{\sum_{i=1}^{k} \|\mathbf{b}_i^*\|^2} \leq \frac{1}{2}\sqrt{\sum_{i=1}^{k} 2^{k-i}\|\mathbf{b}_k^*\|^2} = \frac{1}{2}\sqrt{2^k - 1}\|\mathbf{b}_k^*\| < 2^{(k-2)/2}\|\mathbf{b}_k^*\|.$$

From the above inequalities we conclude that if $\mathbf{B}_n$ is an LLL reduced basis for Λ_n then $\mu(\Lambda_k)/\|\mathbf{b}_k^*\| < 2^{(k-2)/2}$ for all $k \leq n$. ■

The rank reduction procedure is described and analyzed in the following lemma. As in Section 4.1, we use the coset formulation of CVP.

Lemma 4.7 *On input a basis $\mathbf{B}_n$, a target vector $\mathbf{t}$, and a function* CVP *that on input a vector $\mathbf{t}$ returns a shortest vector in $\mathbf{t} + \mathcal{L}(\mathbf{B}_{n-1})$, the function* RankReduce *from Algorithm 2 finds a shortest vector in $\mathbf{t} + \mathcal{L}(\mathbf{B}_n)$ making at most $h = 1 + \lfloor 2\mu(\Lambda_n)/\|\mathbf{b}_n^*\| \rfloor$ calls to* CVP.

Proof: Let $\Lambda_n = \mathcal{L}(\mathbf{B}_n)$ be the input lattice and $\Lambda_{n-1} = \mathcal{L}(\mathbf{B}_{n-1})$ the lower dimensional sublattice generated by the first $n-1$ basis vectors. We want to find a shortest vector in the coset $\mathbf{t} + \Lambda_n$. This is accomplished by partitioning $\mathbf{t} + \Lambda_n$ into lower dimensional layers $(\mathbf{t} + i\mathbf{b}_n) + \Lambda_{n-1}$ indexed by $i \in \mathbb{Z}$, and finding a shortest vector in each one of them using CVP. Details follow. Let τ be the distance of $\mathbf{t}$ to the linear span of the lattice $\operatorname{span}(\mathbf{B}_n)$ and α the coefficient of $\mathbf{t}$ with respect to $\mathbf{b}_n^*$, as computed by RankReduce. (For simplicity, the reader may think of $\mathbf{t}$ as being in the linear span of the lattice, in which case $\tau = 0$.) The algorithm starts by setting $\mathbf{r} \leftarrow \mathbf{t}$ to an arbitrary vector in the coset $\mathbf{t} + \Lambda_n$, and updates $\mathbf{r}$ whenever a shorter vector $\mathbf{r}' \in \mathbf{t} + \Lambda_n$ is found. The main loop of the algorithm goes over (a subset of) all possible layers $(\mathbf{t} + i\mathbf{b}_n) + \Lambda_{n-1}$, and for each layer it computes a shortest vector in the lower dimensional coset $\mathbf{r}' \in (\mathbf{t} + i\mathbf{b}_n) + \Lambda_{n-1}$ using the CVP function. But before doing so, for each i, the algorithm first computes the squared norm $\ell_i^2 = (i+\alpha)^2 \cdot \|\mathbf{b}_n^*\|^2 + \tau^2$ of the shortest vector in the affine span of the layer $(\mathbf{t} + i\mathbf{b}_n + \operatorname{span}(\Lambda_{n-1}))$. Clearly, ℓ_i^2 is a lower bound on the squared norm of any point in $(\mathbf{t} + i\mathbf{b}_n) + \Lambda_{n-1}$. Notice that ℓ_i^2 is a monotonically increasing function of $|i + \alpha|$. So, as layers are considered in order of nondecreasing values of $|i + \alpha|$, the quantity ℓ_i^2 computed by the algorithm at iteration i is also a lower bound on the squared norm of any point in all the layers $(\mathbf{t} + j\mathbf{b}_n) + \Lambda_{n-1}$ yet to be considered in subsequent iterations. Therefore, if at any time ℓ_i^2 is at least as large as the best solution $\|\mathbf{r}\|^2$ found so far, $\mathbf{r}$ is an optimal solution the algorithm may immediately terminate with output $\mathbf{r}$. This proves the correctness of the algorithm.

We still need to bound the running time as a function of $h' = \mu(\Lambda_n)/\|\mathbf{b}_n^*\|$. We will show that the number of iterations performed by the algorithm is at most $h = 1 + \lfloor 2h' \rfloor$. Let $\mathbf{r}$ be a shortest vector in $\mathbf{t} + \Lambda_n$, and let i be the index of the layer $(\mathbf{t} + i\mathbf{b}_n) + \Lambda_{n-1}$ containing $\mathbf{r}$. By definition of ℓ_i^2, we have $\|\mathbf{r}\|^2 \geq \ell_i^2$. But the squared norm of the shortest vector in $\mathbf{t} + \Lambda_n$ (or, equivalently, the squared distance of $\mathbf{t}$ to Λ_n) is at most the sum of the squared distance τ^2 of $\mathbf{t}$ to the linear span of the lattice Λ_n, and the squared covering radius

Algorithm 2 Finding relevant vectors by rank reduction

```
function RankReduce($\mathbf{t}, \mathbf{B}_n, \textsc{cvp}(\cdot)$)
    $\tau = \|\pi^{\perp}_{\mathbf{B}_n}(\mathbf{t})\|$
    $\alpha = \langle \mathbf{t}, \mathbf{b}^*_n \rangle / \langle \mathbf{b}^*_n, \mathbf{b}^*_n \rangle$
    $\mathbf{r} \leftarrow \mathbf{t}$
    for $i \in \mathbb{Z}$ sorted by $|i + \alpha|$ do
        $\ell_i^2 = ((i+\alpha) \cdot \|\mathbf{b}^*_n\|)^2 + \tau^2$
        if $\ell_i^2 < \|\mathbf{r}\|^2$ then
            $\mathbf{r}' = \textsc{cvp}(\mathbf{t} + i\mathbf{b}_n)$
            if $\|\mathbf{r}'\| < \|\mathbf{r}\|$ then $\mathbf{r} \leftarrow \mathbf{r}'$
        else return $\mathbf{r}$

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

function VRevelant($\mathbf{B}_n, V_{n-1}$)
    for $\mathbf{c} \in \{0,1\}^n$ do
        $\mathbf{t_c} = \textsc{RankReduce}(\mathbf{B}_n\mathbf{c}, \mathbf{B}_n, \text{CVPP}(\cdot, 2V_{n-1}))$
    return $\{\mathbf{t_c} \mid \mathbf{c} \in \{0,1\}^n\}$
```

of the lattice $\mu(\Lambda_n)^2$. It follows that $\mu(\Lambda_n)^2 + \tau^2 \geq \|\mathbf{r}\|^2 \geq \ell_i^2$, and $\mu(\Lambda_n)^2 \geq \ell_i^2 - \tau^2 = (i+\alpha)^2\|\mathbf{b}^*_n\|^2$. So, $|i+\alpha| \leq \mu(\Lambda_n)/\|\mathbf{b}^*_n\| = h'$. Since there are at most $h = 1 + \lfloor 2h' \rfloor$ integers within distance h' from any real α, we conclude that $\mathbf{r}$ (or some other point in $\mathbf{t} + \Lambda_n$ achieving the same norm) is found within the first h iterations of the loop. Conversely, after h iterations, the index j of the layer considered in the body of the loop must satisfy $|j + \alpha| > h'$. So, $\ell_j = (j+\alpha)^2\|\mathbf{b}^*_n\|^2 + \tau^2 > (h')^2\|\mathbf{b}^*_n\|^2 + \tau^2 = \mu(\Lambda_n)^2 + \tau^2 \geq \|\mathbf{r}\|^2$ and the loop terminates. This proves that the number of iterations (and calls to the CVP function) is at most h. ■

4.3 Computing the Voronoi cell

In this section we show how to compute the Voronoi cell of an n-dimensional lattice $\mathcal{L}(\mathbf{B}_n)$, which immediately gives solutions to several other lattice problems. As discussed in the introduction, this is accomplished by computing the Voronoi cells of all sublattices $\mathcal{L}(\mathbf{B}_k)$ sequentially for $k = 1, \ldots, n$. Following [AEVZ02], the Voronoi cell of each sublattice $\mathcal{L}(\mathbf{B}_k)$ is computed resorting to the characterization of the Voronoi relevant vectors given in Theorem 3.2. This leads to the formulation of the following problem: given a lattice Λ, find a shortest vector in $\mathbf{t} + 2\Lambda$, for each $\mathbf{t} \in \Lambda$. The next lemma gives a straightforward solution to this problem, based on the CVPP algorithm and rank reduction procedure from Subsections 4.1 and 4.2.

Lemma 4.8 *Given a lattice $\Lambda_n = \mathcal{L}(\mathbf{B}_n)$ and the list of relevant vectors $V_{n-1} = V(\Lambda_{n-1})$ of $\Lambda_{n-1} = \mathcal{L}(\mathbf{B}_{n-1})$, the function* VRevelant *of Algorithm 2 runs in time $\tilde{O}(h \cdot 2^{2n}|V_{n-1}|) = \tilde{O}(h \cdot 2^{3n})$ where $h = O(\mu(\Lambda_{n-1})/\|\mathbf{b}^*_n\|)$, and outputs a list of 2^n lattice vectors containing a shortest vector from each coset $\mathbf{t} + 2\Lambda_n$ with $\mathbf{t} \in \Lambda_n$.*

Proof: There are precisely 2^n cosets of the form $\mathbf{t} + 2\Lambda_n$ with $\mathbf{t} \in \Lambda_n$, and for any basis $\mathbf{B}_n$, a set of coset representatives is given by $\{\mathbf{B}_n\mathbf{c} \mid \mathbf{c} \in \{0,1\}^n\}$. The function VRevelant finds a shortest vector from each coset using RankReduce with the CVP function implemented using the CVPP algorithm. The correctness of VRevelant immediately follows from Lemma 4.7 and Theorem 4.2. By the Lemma 4.7 each invocation of RankReduce makes at most h calls to CVPP, and by Theorem 4.2 each call takes time $\tilde{O}(2^n \cdot |V_{n-1}|)$. So, RankReduce runs in $\tilde{O}(h \cdot 2^n|V_{n-1}|)$ time. Since VRevelant consists of 2^n RankReduce computations, the total time complexity is $\tilde{O}(h \cdot 2^{2n}|V_{n-1}|)$. ■

It immediately follows from Theorem 3.2 that if U is the set of lattice vectors returned by VRevelant on input $(\mathbf{B}_n, V_{n-1})$, then $\pm U$ contains all the Voronoi relevant vectors of $\mathcal{L}(\mathbf{B}_n)$, among other nonrelevant lattice vectors. This is enough for most of the applications considered in this paper, but sometimes one may want a list containing precisely the relevant vectors of a lattice. For completeness, we give an algorithm that filters out the nonrelevant vectors from $\pm U$, and returns a list V_n containing precisely the relevant vectors of the input lattice. The pseudocode is given as function VFilter in Algorithm 3, and it is analyzed in the following lemma. The two algorithms VRevelant and VFilter together allow to compute the list of Voronoi relevant vectors of a lattice, making 2^n calls to a CVP oracle for the same lattice. We remark that ours is just a simple variant of the algorithm already given in [AEVZ02], which directly computes a list containing only the relevant vectors. The difference is that the algorithm of [AEVZ02] computes, for each target $\mathbf{t} \in \Lambda$, *all* shortest vectors in the coset $\mathbf{t} + 2\Lambda$, and immediately discards them if more than two solutions are found. Our algorithm only finds *one* shortest vector in each $\mathbf{t} + 2\Lambda$, and then discards the nonrelevant vectors at the end. We chose to present a variant of the algorithm of [AEVZ02] because our CVPP function (which will be used to instantiate the $\textsc{cvp}(\cdot)$ oracle) returns only one vector, and also because the problem solved by our VRevelant (and the VFilter function) will be needed again in Section 5 to design faster algorithms.

Lemma 4.9 *Given a superset U of the relevant vectors of a lattice ($V(\Lambda_n) \subseteq U \subseteq \Lambda_n$), function* VFilter *from Algorithm 3 outputs $V_n = V(\Lambda_n)$ in time $\tilde{O}(|V_n| \cdot |U|) \leq \tilde{O}(2^n \cdot |U|)$.*

Proof: The function VFilter processes the nonzero input vectors $U \setminus \{\mathbf{0}\}$ in order of nondecreasing norm. We need to prove that each vector $\mathbf{u} \in U$ is included in V if and only if $\mathbf{u}$ is relevant. The proof is based on the following two observations:

- If $\mathbf{u}$ is a relevant vector, then every $\mathbf{v} \in \Lambda_n \setminus \{\mathbf{0}, \mathbf{u}\}$, satisfies $\|2\mathbf{v} - \mathbf{u}\| > \|\mathbf{u}\|$. To see this, notice that if $\mathbf{u}$ is a relevant vector, then by Theorem 3.2 $\pm\mathbf{u}$ are the only shortest vectors of $2\Lambda_n + \mathbf{u}$. Since the vector $2\mathbf{v} - \mathbf{u}$ is in $2\Lambda_n + \mathbf{u}$, it must be $\|2\mathbf{v} - \mathbf{u}\| > \|\mathbf{u}\|$ unless $(2\mathbf{v} - \mathbf{u}) = \pm\mathbf{u}$, or equivalently, $\mathbf{v} \in \{\mathbf{0}, \mathbf{u}\}$.
- If $\mathbf{u}$ is not relevant, then there exists a relevant vector $\mathbf{v}$ such that $\|\mathbf{v}\| < \|\mathbf{u}\|$ and $\|2\mathbf{v} - \mathbf{u}\| \leq \|\mathbf{u}\|$. We prove the statement by contradiction. Assume that $\mathbf{u}$ is not relevant and that for all relevant vectors $\mathbf{v}$ either $\|\mathbf{v}\| \geq \|\mathbf{u}\|$ or $\|2\mathbf{v} - \mathbf{u}\| > \|\mathbf{u}\|$. Notice that if $\|\mathbf{v}\| \geq \|\mathbf{u}\|$, then $\|2\mathbf{v} - \mathbf{u}\| \geq 2\|\mathbf{v}\| - \|\mathbf{u}\| \geq \|\mathbf{u}\|$, with equality $\|2\mathbf{v} - \mathbf{u}\| = \|\mathbf{u}\|$ if and only if $\mathbf{v} = \mathbf{u}$. But the latter is not possible because $\mathbf{v}$ is relevant and $\mathbf{u}$ is not. So, for all relevant vectors $\mathbf{v}$ we have $\|2\mathbf{v} - \mathbf{u}\| > \|\mathbf{u}\|$, or, equivalently, $\|\mathbf{u}/2\| < \|\mathbf{u}/2 - \mathbf{v}\|$. This proves that $\mathbf{u}/2$ is *strictly* closer to the origin $\mathbf{0}$ than to any relevant vector $\mathbf{v}$, i.e., $\mathbf{u}/2$ is in the *interior* of the Voronoi cell $\mathcal{V}(\Lambda_n)$. It follows from Observation 3.1 that $\mathbf{0}$ is the *unique* closest lattice vector to $\mathbf{u}/2$. This is a contradiction because $\mathbf{0}$ and $\mathbf{u}$ have the same distance from $\mathbf{u}/2$.

Now, $\mathbf{0}$ is never included in V because $\mathbf{u} \in U \setminus \{\mathbf{0}\}$. So, by the first observation, relevant vectors $\mathbf{u} \in U$ not already in V always pass the test and are included in the output set V. It follows that when a vector $\mathbf{u} \in U$ is processed, all revelant vectors of norm strictly less than $\|\mathbf{u}\|$ are already in V. So, by the second observation, if $\mathbf{u}$ is not relevant, then it fails the test for some $\mathbf{v} \in V$ and it is discarded. This proves the correctness of the algorithm.

For the running time, the algorithm first sorts the input vectors U in time $O(|U| \log |U|) = \tilde{O}(|U|)$, and then iterates over U. Each iteration takes time at most $\tilde{O}(|V|)$ where $|V| \leq |V(\Lambda_n)|$. So, the total running time is $\tilde{O}(|U| \cdot |V(\Lambda_n)|)$. ■

Our basic algorithm to compute the Voronoi cell of a lattice easily follows from Lemmas 4.6, 4.8 and 4.9. The pseudocode is given in Algorithm 3 as function VoronoiCell. The correctness and performance (when used with the other functions from Algorithm 1 and Algorithm 2) are analyzed in the next theorem.

Theorem 4.10 *The function* VoronoiCell *computes the list of Voronoi relevant vectors of a lattice $\mathcal{L}(\mathbf{B}_n)$ in deterministic time $\tilde{O}(2^{3.5n})$ using $\tilde{O}(2^n)$ space.*

Proof: The VoronoiCell function of Algorithm 3 first preprocesses the input basis, so that, by Lemma 4.6, $h_k = 1 + \lfloor \mu(\Lambda_k)/\|\mathbf{b}_k^*\| \rfloor \leq 2^{k/2}$ for all $k = 2, \ldots, n$. The rest of the algorithm works on the new basis $\mathbf{B}_n$, and iteratively computes the Voronoi cell V_k of all sublattices $\mathcal{L}(\mathbf{B}_k)$ for $k = 1, \ldots, n$. Computing $V_1 = V(\mathcal{L}(\mathbf{B}_1)) = \{\mathbf{b}_1, -\mathbf{b}_1\}$ is straightforward. For all other $k = 2, \ldots, n$, V_k is computed from V_{k-1} using VRevelant and VFilter. By Lemma 4.8, VRevelant$(\mathbf{B}_k, V_{k-1})$ computes a list U_k containing a shortest vector of $\mathbf{t} + 2\mathcal{L}(\mathbf{B}_k)$ for all $\mathbf{t} \in \mathcal{L}(\mathbf{B}_k)$. By Theorem 3.2, if $\mathbf{t} \in \mathcal{L}(\mathbf{B}_k)$ is a relevant vector, then $\pm\mathbf{t}$ are the only shortest vectors in $\mathbf{t} + 2\mathcal{L}(\mathbf{B}_k)$, and either $\mathbf{t}$ or $-\mathbf{t}$ is included in U_k. In either case, $\mathbf{t} \in \pm U_k$. This proves that U_k contains all the relevant vectors. Finally, by Lemma 4.9, $V_k = $ VFilter$(\pm U_k)$ is the list of relevant vectors of $\mathcal{L}(\mathbf{B}_k)$.

Algorithm 3 The main algorithm

```
function VoronoiCell($\mathbf{B}$)
    $\mathbf{B} \leftarrow$ Preprocess($\mathbf{B}$)
    $V_1 = \{\mathbf{b}_1, -\mathbf{b}_1\}$
    for $k = 2 \dots n$ do
        $U_k =$ VRevelant($\mathbf{B}_k, V_{k-1}$)
        $V_k =$ VFilter($\pm U_k$)
    return $V_n$

function VFilter($U$)
    $V \leftarrow \emptyset$
    for $\mathbf{u} \in U \setminus \{\mathbf{0}\}$ sorted by $\|\mathbf{u}\|$ do
        if $\forall \mathbf{v} \in V. \|2\mathbf{v} - \mathbf{u}\| > \|\mathbf{u}\|$ then
            $V \leftarrow V \cup \{\mathbf{u}\}$
    return $V$

function SVP($\mathbf{B}$)
    $V =$ VoronoiCell($\mathbf{B}$)
    $\mathbf{v} = \mathrm{argmin}_{\mathbf{v} \in V} \|\mathbf{v}\|$
    return $\mathbf{v}$

function CVP($\mathbf{B}, \mathbf{t}$)
    $V =$ VoronoiCell($\mathbf{B}$)
    $\mathbf{t}' =$ CVPP($\mathbf{t}, V$)
    return $(\mathbf{t} - \mathbf{t}')$
```

We now analyze the running time. The preprocessing function runs in polynomial time, so its cost is negligible. By Lemmas 4.8 and 4.9, the cost of each iteration is bounded by $\tilde{O}(h_k 2^{2k} \cdot |V_{k-1}|) + \tilde{O}(2^k \cdot |V_k|) = \tilde{O}(h_k 2^{2k} \cdot |V_{k-1}|) = \tilde{O}(2^{3.5k})$. Therefore the total time complexity of the algorithm is $\sum_{k=2,\dots,n}(\tilde{O}(2^{3.5k})) = \tilde{O}(2^{3.5n})$. The space complexity is at most $\tilde{O}(2^n)$ for the storage of the intermediate Voronoi cells V_k. ■

Using Theorem 4.10 and our CVPP algorithm, it is immediate to give deterministic solutions to CVP, SVP and a host of other lattice problems with similar time and space complexity as VoronoiCell. Here we only give a formal statement for SVP, which will be used in the next section to obtain even faster algorithms for all these problems.

Corollary 4.11 SVP *can be solved deterministically in time* $\tilde{O}(2^{3.5n})$ *and space* $\tilde{O}(2^n)$.

Proof: Simply run the algorithm of Theorem 4.10 to compute the Voronoi cell of the lattice, and output a shortest vector from the list of relevant vectors. ■

5 The optimized algorithm

In this section we present an improved algorithm to compute the Voronoi cell of a lattice in time $\tilde{O}(2^{2n})$, and consequently solve CVP, SVP, and other hard lattice problems within the same time bound. The high level structure of the algorithm is identical to the VoronoiCell function from Algorithm 3. The improved time complexity comes from the use of a better preprocessing function and faster VRevelant algorithm.

Recall that in Section 4 the lattice basis was preprocessed simply by applying the LLL reduction algorithm. Here we use stronger block reduction algorithms [GN08, Sch87] that produce better quality bases by making a polynomial number of calls to an SVP oracle for lower dimensional lattices of rank at most $k < n$. The quality of the output basis depends on the block size parameter k. Block reduction algorithms are usually employed with $k = O(\log n)$, so that the SVP queries can be answered in polynomial time $2^{O(k)} = n^{O(1)}$, and the overall running time of the algorithm remains polynomial in n. Since our Voronoi computation algorithm runs in exponential time anyway, here we can use much larger block size, resulting in superpolynomial preprocessing, but without affecting the total time complexity of our algorithm by much. Specifically, we preprocess the input basis using the slide reduction algorithm of [GN08] with block size $k = n/2$ (as described in Lemma 2.1) and the SVP oracle instantiated using the algorithm of Corollary 4.11. The result needed in our algorithm is summarized in the following lemma.

Lemma 5.1 *The* Preprocess *function from Algorithm 4 runs in* $\tilde{O}(2^{2n})$*-time and* $\tilde{O}(2^n)$*-space, and computes a basis for the input lattice* $\mathbf{B}_n$ *such that* $\mu(\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_i))/\|\mathbf{b}_i^*\| \leq O(n^2)$ *for all* $i = 1, \dots, n$.

Proof: The preprocessing function runs the slide reduction algorithm of Lemma 2.1 with block size $k = \lceil n/2 \rceil$, and the SVP oracle (for lattices up to dimension k) instantiated with our SVP algorithm from Section 4. By Corollary 4.11, each call to the SVP algorithm takes time $\tilde{O}(2^{3.5k}) \leq \tilde{O}(2^{2n})$ and space $\tilde{O}(2^k) \leq \tilde{O}(2^n)$. By Lemma 2.1, the running time of the preprocessing function is higher only by a polynomial factor, so it is still $\tilde{O}(2^{2n})$. Finally, by Lemma 2.1, the output basis has the property that $\mu(\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_i))/\|\mathbf{b}_i^*\| \leq O(n^2)$ for all $i = 1, \ldots, n$. ■

The improved preprocessing results in a reduction of the running time by about $2^{\sqrt{n}}$, bringing the time complexity of the Vononoi cell computation algorithm from $\tilde{O}(2^{3.5n})$ down to $\tilde{O}(2^{3n})$. In order to reduce the running time even further, we improve the VRevelant function used in the implementation of VoronoiCell. This is done in Subsection 5.1. Subsection 5.2 concludes with our main Voronoi cell computation algorithm, and applications to other lattice problems.

The pseudocode of all functions used in this section is given in Algorithms 1, 3 and 4. Notice that most of the code (Algorithms 1 and 3) is reused from Section 4, and the Voronoi cell computation method in this section only differs by replacing Algorithm 2 with Algorithm 4.

5.1 Faster computation of relevant vectors

In this subsection we give a faster variant of the VRevelant function from Section 4. Recall that on input a lattice basis $\mathbf{B}_n$ and the list of Voronoi relevant vectors of $\mathbf{B}_{n-1}$, the goal of VRevelant is to find a shortest vector in each coset $\mathbf{t} + 2\Lambda_n$, for $\mathbf{t} \in \Lambda_n$. In Section 4, this was accomplished simply by reducing this problem to 2^n CVP computations, and then solving each CVP instance independently by rank reduction. Here we apply the rank reduction technique directly to the problem solved by the VRevelant function. This gives raise to an inhomogeneous version of the problem solved by VRevelant: given a lattice Λ and a target $\mathbf{y}$, find a shortest vector in each coset $\mathbf{t} + 2\Lambda$, for $\mathbf{t} \in \mathbf{y} + \Lambda$. Function VEnum from Algorithm 4 solves a slightly more general problem: given a lattice Λ, an integer $k \geq 2$ and a target $\mathbf{y}$, find a shortest vector in each coset $\mathbf{t} + k\Lambda$, for $\mathbf{t} \in \mathbf{y} + \Lambda$. The main applications studied in this paper only need to solve this problem for $k = 2$, but the algorithm works for any k, so we describe a solution to this more general problem which may be of independent interest. (E.g., see Corollary 5.8.)

The algorithm makes use of two data structures. The first is simply an array A with k^n entries, each corresponding to a coset of $k \cdot \Lambda$ in $\mathbf{t} + \Lambda$, and storing the shortest vector from the coset found so far. How the array entries are indexed is not important, but for concreteness the reader can think of each $\mathbf{v} \in \mathbf{t} + \Lambda$ as being mapped to the index in $\{1, \ldots, k^n\}$ obtained by expressing $\mathbf{v} - \mathbf{t} = \mathbf{B}\mathbf{x} \in \Lambda$ in terms of the basis $\mathbf{B}$, and setting the index to $\sum_{i=1}^{n} k^{i-1} \cdot (x_i \bmod k) + 1$. For simplicity, in the description of the algorithms, we omit the details of the index computation, and write $A[\mathbf{v} \bmod k\Lambda]$ to denote the array entry corresponding to the coset of $\mathbf{v}$ modulo $k\Lambda$. The other data structure is a dynamic priority queue, storing vectors in $\mathbf{t} + \Lambda$, which will be used to enumerate the output vectors in order of nondecreasing norm. The priority queue is initially empty, and each entry of the array stores ∞ as a trivial upper bound on the length of the shortest vector in each coset. The first step of the algorithm is to compute (using our CVPP algorithm) an arbitrary vector $\mathbf{u} \in (\Lambda + \mathbf{t}) \cap \bar{\mathcal{V}}$, store it in the priority queue, and update the array accordingly. At each subsequent step, the algorithm dequeues the top vector $\mathbf{u}$ in the priority queue, and inserts in the queue (and in the array) all of its neighbors that satisfy a certain length condition.

Lemma 5.2 *On input the set $V_n = V(\Lambda_n)$ of relevant vectors of a lattice Λ_n, a target vector $\mathbf{t}$, and an integer $k \geq 2$, the function* VEnum *from Algorithm 4 runs in time $\tilde{O}(|V_n|k^n)$ and space $\tilde{O}(k^n)$, and outputs an array of size k^n containing a shortest vector from each coset of $k\Lambda_n$ in $\mathbf{t} + \Lambda_n$.*

Proof: Throughout this proof, when we refer to the array A as a set, we mean the set of vectors $\{A[i] \colon A[i] \neq \infty\}$. Notice that at all times during the execution of the algorithm we have $Q \subseteq A$. This is true because whenever the array is modified by setting $A[\mathbf{y} + k\Lambda] \leftarrow \mathbf{y}$, the queue is updated accordingly by replacing $A[\mathbf{y} + k\Lambda]$ with $\mathbf{y}$. The only other operation performed on the queue Q is the removal of the shortest vector $\mathbf{u}$ at the beginning of each iteration of the "while" loop, making Q a strict subset of A. This is also the only operation that affects the set $A \setminus Q$, by removing $\mathbf{u}$ from Q and thereby adding it to $A \setminus Q$.

The time and space complexity of the algorithm is now very easy to analyze. The array A has size k^n because it has an entry for each coset of $k\Lambda$ in $\mathbf{t}+\Lambda$. It follows from $Q \subseteq A$ that the size of the queue Q is also bounded by $|Q| \leq |A| \leq k^n$ at all times during the execution of the algorithm. So, the overall space complexity is $\tilde{O}(k^n)$. For the running time, we observe that since the size of the set $A \setminus Q$ increases by 1 at each iteration of the "while" loop, and $|A \setminus Q| \leq |A| \leq k^n$, the number of iterations of the "while" loop is at most k^n. (In fact, it is precisely k^n.) For each iteration of the "while" loop, the body of the nested "for" loop is repeated $|V|$ times. So, the running time of the algorithm is bounded by one execution of CVPP (which, by Theorem 4.2, takes time $\tilde{O}(2^n \cdot |V|)$,) and a total of $k^n \cdot |V|$ basic operations on the queue Q and array A. Assuming an efficient implementation of Q as a dynamic priority queue (e.g., a binary heap data structure) supporting the insertion and removal of elements in logarithmic time $O(\log|Q|) \leq O(\log k^n) = O(n \log k)$, the overall running time of the algorithm is $\tilde{O}(|V| \cdot 2^n) + \tilde{O}(|V| \cdot k^n \cdot n \log k) = \tilde{O}(|V| \cdot k^n)$.

We now prove correctness. We need to show that by the end of the execution, the set A contains a shortest vector from each coset of $k\Lambda$ in $\mathbf{t}+\Lambda$. Notice that each cell of the array $A[\mathbf{y}+k\Lambda]$ is either empty (represented by storing the value ∞), or it stores a vector $\mathbf{y}$ from the coset $\mathbf{y}+k\Lambda$ indexing the entry. Also, array entries are overwritten $A[\mathbf{y}+k\Lambda] \leftarrow \mathbf{y}$ only to be replaced by strictly shorter vectors $\|\mathbf{y}\| < \|A[\mathbf{y}+k\Lambda]\|$ from the same coset. It follows that if a shortest vector $\mathbf{y}$ from $\mathbf{y}+k\Lambda$ is ever stored in the array, then that entry will never be overwritten, and $\mathbf{y}$ will be part of the final output of the algorithm. So, in order to prove correctness it is enough to prove that a shortest vector from each coset is stored in the array at any point during the execution of the algorithm. We also recall that, by Observation 3.1, $\mathbf{y}$ is a shortest vector in $\mathbf{y}+k\Lambda$ if and only if $\mathbf{y} \in k\bar{\mathcal{V}}(\Lambda)$. So, the algorithm is correct if for any coset C (indexing a position in the array A), at some point during the execution a vector from $C \cap k\bar{\mathcal{V}}(\Lambda)$ is stored in $A[C]$.

We first prove that all vectors in the set $G = \bar{\mathcal{V}}(\Lambda) \cap (\mathbf{t}+\Lambda)$ are eventually stored in A. By Theorem 4.2 the vector $\mathbf{y}_0$ computed by the CVPP call belongs to G and it is stored in A before entering the "while" loop. Now consider any other vector $\mathbf{y} \in G$. By Lemma 3.8, there is a set of mutually orthogonal relevant vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k \in V(\Lambda)$ such that $\mathbf{y} = \mathbf{y}_0 + \sum_{i=1}^{k} \mathbf{v}_i$, and $\mathbf{y}_0 + \sum_{i \in S} \mathbf{v}_i \in G$ for all $S \subseteq \{1, \ldots, k\}$. We prove that $\mathbf{y}$ is inserted in A by induction on k. As a base case, when $k = 0$, the vector $\mathbf{y} = \mathbf{y}_0$ is inserted in A before entering the "while" loop. So, assume $k > 0$, and consider the vector $\mathbf{y}' = \mathbf{y} - \mathbf{v}_k$. Since $\mathbf{y}' = \mathbf{y}_0 + \sum_{i=1}^{k-1} \mathbf{v}_i$ and $\mathbf{y}' \in G$, by induction hypothesis $\mathbf{y}'$ is stored in A and inserted in the queue Q at some point during the execution of the algorithm. Consider the iteration of the "while" loop when $\mathbf{u} = \mathbf{y}'$ is dequeued. Then the vector $\mathbf{y} = \mathbf{u} + \mathbf{v}_k \in \mathbf{u} + V$ is examined by the "for" loop. Now, $\|\mathbf{u}\| = \|\mathbf{y}\|$ because all vectors in G have norm $\|\mathbf{y}_0\|$. Also, $\mathbf{y} \in \bar{\mathcal{V}}(\Lambda) \subset k\bar{\mathcal{V}}(\Lambda)$ is the only vector in $\mathbf{y}+ \bmod k\Lambda$ of norm $\|\mathbf{y}\|$ because it belongs to the interior of of the Voronoi cell $\mathcal{V}(k\Lambda) = k\mathcal{V}(\Lambda)$. It follows that either $A[\mathbf{y} \bmod k\Lambda] = \mathbf{y}$ or $\|A[\mathbf{y} \bmod k\Lambda\| > \|\mathbf{y}\|$ and $\mathbf{y}$ is stored in $A[\mathbf{y} \bmod k\Lambda]$. This proves that all vectors in G are stored in A.

We now prove that the array stores a shortest vector from each coset of $k\Lambda$ in $\mathbf{t}+\Lambda$. We prove that the array stores a shortest vector from each coset $C = \mathbf{y}+k\Lambda$ by induction on the quantity $\theta(C) = \min\{\|\mathbf{v}\| \colon \mathbf{v} \in C\}$. The base case is given by all cosets C such that $\theta(C) = \min\{\|\mathbf{v}\| \colon \mathbf{v} \in \mathbf{t}+\Lambda\} = \|\mathbf{y}_0\|$. These are precisely the cosets with a shortest vector in G, and we already proved that all these vectors are stored in A. So, consider a coset C with a shortest vector $\mathbf{y} \in C \cap (k\bar{\mathcal{V}}(\Lambda) \setminus \bar{\mathcal{V}}(\Lambda))$. Let $\mathbf{v} \in V$ be a relevant vector that maximizes the quantity $\alpha = 2\langle \mathbf{y}, \mathbf{v} \rangle / \|\mathbf{v}\|^2$. Notice that α is the smallest value such that $\mathbf{y} \in \alpha\bar{\mathcal{V}}(\Lambda)$, so it must be $\alpha \leq k$. By Lemma 3.6 we have $\mathbf{y} - \mathbf{v} \in \alpha\bar{\mathcal{V}}(\Lambda) \subseteq k\bar{\mathcal{V}}(\Lambda)$ and $\|\mathbf{y} - \mathbf{v}\| < \|\mathbf{y}\|$. Let $C' = C - \mathbf{v}$ be the coset of $\mathbf{y} - \mathbf{v}$. Since $\theta(C') \leq \|\mathbf{y} - \mathbf{v}\| < \|\mathbf{y}\| = \theta(C)$, we can assume by induction that the array A stores a shortest vector $\mathbf{u}$ from C'. We will prove that $\mathbf{u} + \mathbf{v}$ is also a shortest vector in C. It follows that when $\mathbf{u}$ is selected by the "while" loop for dequeuing, the "for" loop stores $\mathbf{y} = \mathbf{u} + \mathbf{v}$ in $A[C]$ unless $A[C]$ already contains a shortest vector from C. It remains to prove that $\mathbf{u} + \mathbf{v}$ is a shortest vector in C, or equivalently $\mathbf{u} + \mathbf{v} \in k\bar{\mathcal{V}}(\Lambda)$. If $\mathbf{u} = \mathbf{y} - \mathbf{v}$, then $\mathbf{u} + \mathbf{v} = \mathbf{y} \in k\bar{\mathcal{V}}(\Lambda)$. So, assume $\mathbf{u} \neq \mathbf{y} - \mathbf{v}$ are distinct elements of C'. Since $\mathbf{y} - \mathbf{v} \in \alpha\bar{\mathcal{V}}(\Lambda) \subseteq k\bar{\mathcal{V}}$ and $\mathbf{u} \in k\bar{\mathcal{V}}$, the coset C' has multiple shortest vectors, and they must all belong to the boundary of the Voronoi cell $k\bar{\mathcal{V}} \setminus k\mathcal{V}$. In particular, $\alpha = k$ and $2\langle \mathbf{y}, \mathbf{v} \rangle / \|\mathbf{v}\|^2 = k$, or, equivalently, $\|\mathbf{y}\| = \|\mathbf{y} - k\mathbf{v}\|$. By Lemma 3.4 (applied to lattice $k\Lambda$) the vector $\mathbf{y} - k\mathbf{v}$ belongs to $k\bar{\mathcal{V}}$. Applying Lemma 3.5 to the vectors $\mathbf{x} = \mathbf{y} - k\mathbf{v}, \mathbf{y}, \mathbf{z} = \mathbf{y} - \mathbf{v}$ and $\mathbf{u}$ (and lattice $k\Lambda$), we get that $\mathbf{u} + \mathbf{v} = \mathbf{u} + \mathbf{y} - \mathbf{z}$ is in $k\mathcal{V}$. ■

Lemma 5.3 *On input a lattice* $\Lambda = \mathcal{L}(\mathbf{B}_n)$*, and the relevant vectors* $V_{n-1} = V(\mathcal{L}(\mathbf{B}_{n-1}))$*, function*

Algorithm 4 Optimized Algorithms

```
function VEnum($V, \mathbf{t}, k$)
    $\Lambda = \mathcal{L}(V)$
    for $i = 1 \ldots k^n$ do $A[i] \leftarrow \infty$;
    $\mathbf{y}_0 = \text{CVPP}(V, \mathbf{t})$;
    $Q \leftarrow \{\mathbf{y}_0\}$; $A[\mathbf{y}_0 \bmod k\Lambda] \leftarrow \mathbf{y}_0$;
    while $Q \neq \emptyset$ do
        $\mathbf{u} = \text{argmin}_{\mathbf{u} \in Q} \|\mathbf{u}\|$; $Q \leftarrow Q \setminus \{\mathbf{u}\}$;
        for $\mathbf{y} \in (\mathbf{u} + V)$ do
            if $\|\mathbf{u}\| \leq \|\mathbf{y}\| < \|A[\mathbf{y} \bmod k\Lambda]\|$ then
                $Q \leftarrow Q \setminus \{A[\mathbf{y} \bmod k\Lambda]\} \cup \{\mathbf{y}\}$;
                $A[\mathbf{y} \bmod k\Lambda] \leftarrow \mathbf{y}$;
    return $A$

function Preprocess($\mathbf{B}$)
    return SlideReduce($\mathbf{B}$,$\lceil n/2 \rceil$)

function VRevelant($\mathbf{B}_n, V_{n-1}$)
    $\Lambda = \mathcal{L}(\mathbf{B}_n)$
    for $j = 1, \ldots, 2^n$ do $T[j] \leftarrow \infty$
    for $i = 0, 1, \ldots$ do
        if $(i \cdot \|\mathbf{b}_n^*\| < \max_j \|T[j]\|)$ then
            for $\mathbf{t} \in$VEnum($V_{n-1}, i\mathbf{b}_n, 2$) do
                if $\|T[\mathbf{t} \bmod 2\Lambda]\| > \|\mathbf{t}\|$ then
                    $T[\mathbf{t} \bmod 2\Lambda] \leftarrow \mathbf{t}$
        else return T
```

VRevelant *from from Algorithm 4 outputs a list containing a shortest vector from each coset* $\mathbf{t} + 2\Lambda$ *with* $\mathbf{t} \in \Lambda$. *The function runs in* $\tilde{O}(h_n \cdot |V_{n-1}| \cdot 2^n)$ *time and* $\tilde{O}(2^n)$ *space, where* $h_n = 2\mu(\Lambda)/\|\mathbf{b}_n^*\|$.

Proof: The function uses an array T indexed by $\Lambda/2\Lambda$ and with all entries initialized to ∞, where it keeps, for each $C \in \Lambda/2\Lambda$, the shortest vector from the coset C found so far. The lattice Λ is partitioned into layers of the form $i\mathbf{b}_n + \Lambda'$ (where $\Lambda' = \mathcal{L}(\mathbf{B}_{n-1})$,) which are processed one at a time. Since each coset $\mathbf{t} + 2\Lambda$ (with $\mathbf{t} \in \Lambda$) is symmetric with respect to $\mathbf{0}$, it suffices to consider only layers with $i \geq 0$. The layers are considered in order of increasing index i. For each i, the function VEnum is used to compute a shortest vector in each coset $\mathbf{t} + 2\Lambda'$ with $\mathbf{t} \in i\mathbf{b}_n + \Lambda'$, and T is updated accordingly. Notice that all points in $i\mathbf{b}_n + \Lambda'$ have norm at least $i \cdot \|\mathbf{b}_n^*\|$. So, if at any point we have found a vector of norm at most $i \cdot \|\mathbf{b}_n^*\|$ in each coset, we do not need to consider the remaining layers, and the algorithm can immediately terminate with output T. This proves the correctness of the algorithm.

Now, we analyze the running time. The initialization of T takes time $\tilde{O}(2^n)$. Each iteration of the main loop (including the execution of VEnum and scanning its output to update T) takes time $\tilde{O}(2^n|V_{n-1}|)$ and space $\tilde{O}(2^n)$. In order to bound the running time, we prove that the algorithm terminates after at most h_n iterations of the main loop. For each index $j \in \{1, \ldots, 2^n\}$, let $\mathbf{v}_j$ be a shortest vector in the corresponding coset $\mathbf{v}_j + 2\Lambda$, and let i_j be the index of the layer $i_j\mathbf{b}_n + \Lambda'$ containing $\mathbf{v}_j$. Assume without loss of generality that $i_j \geq 0$. (If $i_j < 0$, then replace $\mathbf{v}_j$ with $-\mathbf{v}_j \in \mathbf{v}_j + 2\Lambda$.) Since $\mathbf{v}_j$ is in $\text{span}(\Lambda) = \text{span}(2\Lambda)$, it must be $\|\mathbf{v}_j\| \leq \mu(2\Lambda) = 2\mu(\Lambda)$. By taking the component of $\mathbf{v}_j$ in the direction of $\mathbf{b}_n^*$, we also get that $\|\mathbf{v}_j\| \geq i_j \cdot \|\mathbf{b}_n^*\|$. Combining the two inequalities, we get that $i_j \cdot \|\mathbf{b}_n^*\| \leq \|\mathbf{v}_j\| \leq 2\mu(\Lambda)$. This proves that $\mathbf{v}_j$ (or an equally short vector from the same coset) is found within the first $i_j \leq 2\mu(\Lambda)/\|\mathbf{b}_n^*\| = h_n$ iterations of the main loop. So, by the time the algorithm enters iteration $i = h_n + 1$, we have $\max_j \|T[j]\| \leq 2\mu(\Lambda)$, and the algorithm terminates because $i \cdot \|\mathbf{b}_n^*\| > h_n \cdot \|\mathbf{b}_n^*\| = 2\mu(\Lambda)$. ■

5.2 Main Algorithm

A faster algorithm for Voronoi cell computation is obtained simply by replacing Algorithm 2 from Section 4 with Algorithm 4.

Theorem 5.4 *There is a deterministic* $\tilde{O}(2^{2n})$*-time and* $\tilde{O}(2^n)$*-space algorithm that on input an n-dimensional lattice* Λ *with basis* $\mathbf{B}$*, outputs the relevant vectors of* Λ.

Proof: The main function is the same VoronoiCell from Algorithm 3 already analyzed in Theorem 4.10, but with Preprocess and VRevelant implemented as in Algorithm 4. The analysis is almost identical to the one in the proof of Theorem 4.10, with the following differences. The preprocessing function is no longer polynomial time, but by Lemma 5.1 it still runs within the $\tilde{O}(2^{2n})$-time and $\tilde{O}(2^n)$-space bounds. Also, after preprocessing, the basis $\mathbf{B}_n$ satisfies $h_k = \mu(\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_i))/\|\mathbf{b}_i^*\| \leq O(n^2)$ for all $i = 1, \ldots, n$. So, by Lemma 5.3, each call to the new VRevelant function runs in $\tilde{O}(h_k \cdot |V_{k-1}| \cdot 2^k) = \tilde{O}(2^{2k})$ time. The rest of the running time analysis is similar to Theorem 4.10, and yields a $\tilde{O}(2^{2n})$ bound on the total running time. The proof of correctness and space complexity bound is identical. ■

From the list of Voronoi relevant vectors, 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 5.5 *There is a deterministic $\tilde{O}(2^{2n})$-time algorithm to solve SVP, and to compute the kissing number of a lattice.*

The Voronoi relevant vectors can also be used with our CVPP algorithm to solve CVP.

Corollary 5.6 *There is a deterministic $\tilde{O}(2^{2n})$-time, $\tilde{O}(2^n)$-space algorithm to solve* CVP.

Proof: The algorithm is given by function CVP from Algorithm 3. By Theorem 5.4 the computation of the Voronoi cell takes $\tilde{O}(2^{2n})$ time and $\tilde{O}(2^n)$ space. Once the Voronoi cell has been computed, the input CVP instance is solved using the CVPP algorithm, which, by Theorem 4.2, runs in time $\tilde{O}(2^{2n})$. ■

Algorithms for other lattice problems, like SIVP, SAP, GCVP, SMP (see [BN09, Mic08]) can be obtained by reduction to CVP.

Corollary 5.7 *There are deterministic $\tilde{O}(2^{2n})$-time $\tilde{O}(2^n)$-space algorithms to solve SIVP, SAP, GCVP and SMP.*

Proof: All these problems can be solved applying the reductions in [Mic08] from these problems to CVP, and using our CVP algorithm to solve the CVP instances that arise during the reduction. Since the reductions in [Mic08] run in polynomial time and preserve the dimension of the lattice, the resulting algorithm has $\tilde{O}(2^{2n})$-time and $\tilde{O}(2^n)$-space complexity. ■

We conclude the section with a simple application of Lemma 5.2 with arbitrary $k \geq 2$.

Corollary 5.8 *There is an algorithm that on input a lattice Λ_n, a target vector $\mathbf{t}$, and an integer $k \geq 2$, runs in $\tilde{O}((4k)^n)$ time and outputs all lattice points within distance $k\lambda(\Lambda_n)$ from $\mathbf{t}$.*

Proof: Let $\lambda = \lambda(\Lambda_n)$. The algorithm begins by computing the list of relevant vectors $V = V(\Lambda_n)$ in time $\tilde{O}(2^{2n})$ using Theorem 5.4. Then, it invokes the function VEnum$(V, \mathbf{t}, 2k)$ of Lemma 5.2, and for each vector $\mathbf{v} \in \mathbf{t} + \Lambda_n$ found by VEnum, it checks if $\mathbf{v}$ is shorter than $k\lambda$, and if so it outputs $(\mathbf{t} - \mathbf{v})$. Since all vectors shorter than $k\lambda$ belong to $2k\mathcal{V}(\Lambda)$, this procedure finds all lattice vectors within distance $k\lambda$ from $\mathbf{t}$. By Lemma 5.2, the running time of the enumeration algorithm is at most $\tilde{O}((4k)^n)$. So, the total running time is also $\tilde{O}((4k)^n)$. ■

We remark that the algorithm of Corollary 5.8 can be optimized in several ways. For example, since VEnum enumerates the output vectors in order of nondecreasing length, one can limit the enumeration to the vectors shorter than $k\lambda$. It is also possible to reduce the space complexity of the algorithm from $\tilde{O}((2k)^n)$ to $\tilde{O}(2^n)$ (i.e., just for the cost of computing and storing the Voronoi relevant vectors) using standard graph traversal techniques, though at the price of increasing the running time.

6 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.

To start with, our algorithm uses exponential space. It would be nice to find an algorithm running in single exponential time and polynomial space. We remark that the high space complexity of our algorithm is due primarily to the need of storing the Voronoi cell of the lattice, in order to use it in conjunction with our CVPP algorithm. So, the problem of improving the space complexity of our algorithm is closely related to the problem of "compressing" the description of the Voronoi cell. More specifically, we ask if there is a method to compress the list of Voronoi relevant vectors of a lattice into a string of polynomial length, from which it is possible to efficiently[4] recover the relevant vectors.

Another important open problem is to generalize our algorithm to arbitrary norms. 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 all of our results. The main technical problem is that the Voronoi cells in norms other than ℓ_2 are not necessarily convex. So, extending the algorithm to any other norm is likely to require some additional idea. (Convexity of the Voronoi cell is used implicitly in several parts of our proof.) An important application of extending our algorithm to any norm is that it would yield single exponential time algorithms for integer programming [Kan87].

We have shown that CVP can be solved in time $\tilde{O}(2^{2n})$. Since CVP is NP-hard, the time complexity cannot be improved beyond $2^{\Omega(n)}$ under standard complexity assumptions. It may be possible though to improve the constant in the exponent, and prove that CVP can be solved in time 2^{cn} for some $c < 2$. In fact, this could be achieved simply by a better analysis of the CVPP algorithm and faster algorithms for VEnum. 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 [NV08, MV10b, WLTB11]) just because the Voronoi cell of an n-dimensional lattice can have as many as $O(2^n)$ facets. 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 cells have a small description. Notice that even for such lattices, our current Voronoi cell and CVP algorithms may take exponential time because they require the computation of other Voronoi cells (of lower dimensional sublattices) as intermediate results.

We remark that, as it stands, and in any practical scenarios, our algorithm is unlikely to be competitive with traditional methods based on lattice point enumeration [Kan87, Sch87, SE94, HS07, AEVZ02], despite their higher worst-case asymptotic complexity $2^{\Omega(n \log n)}$. In fact, even within enumeration methods, the best asymptotic algorithm [Kan87, HS07] with running time $2^{O(n \log n)}$ does not fare well in practice against heuristics or asymptotically inferior solutions with running time $2^{O(n^2)}$. (See [SE94, HS07, AEVZ02].) Enumeration methods perform very well in practice because of the very small hidden constants in the exponent of their running time, especially when the input lattice is chosen somehow at random. Developing practical algorithms based on the ideas and techniques of this paper that are competitive with enumeration methods is left as an open problem. A possible approach to develop practical variants of our algorithm may be to use only a sublist of Voronoi relevant vectors, at the cost of producing only approximate solutions to CVP. This could allow to reduce the complexity of the algorithms below the current 2^n barrier. The challenge is to bound the quality of the approximate solutions found.

Finally, in this paper we used 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 solve the actual CVP instance. It is an interesting open problem if a similar bootstrapping can be performed using polynomial time CVPP approximation algorithms like [AR05], to yield a polynomial time solution to $\sqrt{n}$-approximate CVP.

[4]Here efficient means in at most single exponential time, while using only polynomial space

Acknowledgments

We thank Damien Stehlé for very helpful discussion and for pointing out some problems in earlier drafts of this paper. We also thank the anonymous reviewers for their comments. This work was supported in part by NSF grants CCF-0634909, CNS-0831536 and CNS-1117936. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[ABSS97] 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, April 1997. doi:10.1006/jcss.1997.1472. Preliminary version in FOCS'93.

[AEVZ02] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, August 2002. doi:10.1109/TIT.2002.800499.

[AJ08] V. Arvind and P. S. Joglekar. Some sieving algorithms for lattice problems. In *Proceedings of FSTTCS*, pages 25–36. 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1738.

[Ajt98] M. Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of STOC*, pages 10–19. ACM, May 1998. doi:10.1145/276698.276705.

[Ajt04] 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.

[AKKV12] M. Alekhnovich, S. Khot, G. Kindler, and N. Vishnoi. Hardness of approximating the closest vector problem with pre-processing. *Computational Complexity*, 2012. doi:10.1007/s00037-011-0031-3. To appear. Preliminary version in FOCS '05.

[AKS01] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of STOC*, pages 266–275. ACM, July 2001. doi:10.1145/380752.380857.

[AKS02] M. Ajtai, R. Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *Proceedings of CCC*, pages 53–57. IEEE, May 2002. doi:10.1109/CCC.2002.1004339.

[AR05] D. Aharonov and O. Regev. Lattice problems in NP intersect coNP. *Journal of the ACM*, 52(5):749–765, 2005. doi:10.1145/1089023.1089025. Preliminary version in FOCS '04.

[Bab86] L. Babai. On Lovasz' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. doi:10.1007/BF02579403.

[Ban93] W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296:625–635, 1993.

[Blö00] J. Blömer. Closest vectors, successive minima and dual HKZ-bases of lattices. In *Proceedings of ICALP*, volume 1853 of *LNCS*, pages 248–259. Springer, July 2000. doi:10.1007/3-540-45022-X_22.

[BN09] J. Blömer and S. Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theoretical Computer Science*, 410(18):1648–1665, April 2009. doi:10.1016/j.tcs.2008.12.045. Preliminary version in ICALP '07.

[BS99] 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*, pages 711–720. ACM, May 1999. doi:10.1145/301250.301441.

[Cas71] J. W. S. Cassels. *An introduction to the geometry of numbers.* Springer-Verlag, New York, 1971.

[Cha07] D. X. Charles. Counting lattice vectors. *Journal of Computer and System Sciences*, 73(6):962 – 972, 2007. doi:10.1016/j.jcss.2007.03.014.

[CJL$^+$92] 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. doi:10.1007/BF01201999. Preliminary versions in Eurocrypt '91 and FCT '91.

[CM06] 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. doi:10.1109/TIT.2006.881835.

[CN99] 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, October 1999. doi:10.1006/jcss.1999.1649. Preliminary version in CCC '98.

[CS98] J. H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups.* Springer Verlag, 3rd edition, 1998.

[DKRS03] 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. doi:10.1007/s00493-003-0019-y. Preliminary version in FOCS '98.

[DPV11] D. Dadush, C. Peikert, and S. Vempala. Enumerative algorithms for lattice problems in any norm via M-ellipsoid coverings. In *Proceedings of FOCS*, pages 580–589. 2011. doi:10.1109/FOCS.2011.31.

[EHN11] F. Eisenbrand, N. Hähnle, and M. Niemeier. Covering cubes and the closest vector problem. In *Proceedings of SoCG*, pages 417–423. ACM, New York, NY, USA, 2011. doi:10.1145/1998196.1998264.

[FM03] 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. doi:10.1016/j.jcss.2004.01.002. Preliminary version in CCC '02.

[GHGKN06] 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, August 2006. doi:10.1007/11818175_7.

[GMR05] V. Guruswami, D. Micciancio, and O. Regev. The complexity of the covering radius problem. *Computational Complexity*, 14(2):90–121, jun 2005. doi:10.1007/s00037-005-0193-y. Preliminary version in CCC '04.

[GMSS99] 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. doi:10.1016/S0020-0190(99)00083-6.

[GN08] N. Gama and P. Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *Proceedings of STOC*, pages 207–216. ACM, May 2008. doi:10.1145/1374376.1374408.

[Hel85] B. Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theoretical Computer Science*, 41(2–3):125–139, December 1985. doi:10.1016/0304-3975(85)90067-2.

[Hor96] A. G. Horváth. On the dirichlet-voronoi cell of unimodular lattices. *Geometriae Dedicata*, 63:183–191, 1996. doi:10.1007/BF00148218.

[HPS11] G. Hanrot, X. Pujol, and D. Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Proceedings of IWCC*, volume 6639 of *LNCS*, pages 159–190. Springer, 2011. doi:http://dx.doi.org/10.1007/978-3-642-20901-7_10.

[HR06] I. Haviv and O. Regev. Hardness of the covering radius problem on lattices. In *Proceedings of CCC*, pages 145–158. IEEE, July 2006. doi:10.1109/CCC.2006.23.

[HR07] I. Haviv and O. Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proceedings of STOC*, pages 469–477. ACM, June 2007. doi:10.1145/1250790.1250859.

[HS07] G. Hanrot and D. Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *Proceedings of Crypto*, volume 4622 of *LNCS*, pages 170–186. Springer, August 2007. doi:10.1007/978-3-540-74143-5_10.

[JS98] A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185, 1998. doi:10.1007/s001459900042.

[Kan87] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operation research*, 12(3):415–440, August 1987. doi:10.1287/moor.12.3.415. Preliminary version in STOC '83.

[Kho05] S. Khot. Hardness of approximating the shortest vector problem in lattices. *Journal of the ACM*, 52(5):789–808, September 2005. doi:10.1145/1089023.1089027. Preliminary version in FOCS '04.

[KPV12] S. Khot, P. Popat, and N. Vishnoi. $2^{\log^{1-\epsilon} n}$ hardness for closest vector problem with preprocessing. In *Proceedings of STOC*. 2012. To appear.

[Len83] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, November 1983.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982.

[LM85] S. Landau and G. L. Miller. Solvability by radicals is in polynomial time. *Journal of Computer and System Sciences*, 30(2):179–208, April 1985. doi:10.1016/0022-0000(85)90013-3. Preliminary version in STOC '83.

[MG02] 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, March 2002.

[Mic01a] D. Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, March 2001. doi:10.1109/18.915688.

[Mic01b] D. Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, March 2001. doi:10.1137/S0097539700373039. Preliminary version in FOCS '98.

[Mic04] 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. doi:10.1137/S0097539703433511. Preliminary version in STOC '02.

[Mic08] D. Micciancio. Efficient reductions among lattice problems. In *Proceedings of SODA*, pages 84–93. ACM/SIAM, January 2008. doi:10.1145/1347082.1347092.

[MR07] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measure. *SIAM Journal on Computing*, 37(1):267–302, 2007. doi:10.1137/S0097539705447360. Preliminary version in FOCS 2004.

[MV10a] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proceedings of STOC*, pages 351–358. 2010. doi:10.1145/1806689.1806739.

[MV10b] D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of SODA*, pages 1468–1480. ACM/SIAM, January 2010.

[NS01] 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, March 2001. doi:10.1007/3-540-44670-2_12.

[NV08] P. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology*, 2(2):181–207, July 2008. doi:10.1515/JMC.2008.009.

[NV09] P. Q. Nguyen and B. Vallée, editors. *The LLL Algorithm: Survey and Applcations*. Information Security and Cryptography. Springer, 2009. doi:10.1007/978-3-642-02295-1.

[Odl89] 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. AMS, Boulder, Colorado, 1989.

[PS09] X. Pujol and D. Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. Report 2009/605, IACR ePrint archive, December 2009.

[Reg04] O. Regev. Improved inapproximability of lattice and coding problems with preprocessing. *IEEE Transactions on Information Theory*, 50(9):2031–2037, 2004. doi:10.1109/TIT.2004.833350. Preliminary version in CCC '03.

[Reg09] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of ACM*, 56(6):34, September 2009. doi:10.1145/1568318.1568324. Preliminary version in STOC '05.

[Sch87] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2–3):201–224, August 1987. doi:10.1016/0304-3975(87)90064-8.

[Sch88] C.-P. Schnorr. A more efficient algorithm for lattice basis reduction. *Journal of Algorithms*, 9(1):47–62, March 1988. doi:10.1016/0196-6774(88)90004-1. Preliminary version in ICALP '86.

[Sch06] C. P. Schnorr. Fast LLL-type lattice reduction. *Information and Computation*, 204(1):1–25, January 2006. doi:10.1016/j.ic.2005.04.004.

[SE94] C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, August 1994. doi:10.1007/BF01581144. Preliminary version in FCT '91.

[SFS09] N. Sommer, M. Feder, and O. Shalvi. Finding the closest lattice point by iterative slicing. *SIAM J. Discrete Math.*, 23(2):715–731, April 2009. doi:10.1137/060676362.

[SSV09] 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. doi: 10.1090/S0025-5718-09-02224-8.

[VB96] 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, January 1996. doi:10.1109/18.481786.

[vEB81] P. van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical Report 81-04, Mathematische Instituut, University of Amsterdam, 1981. Available on-line at URL `http://turing.wins.uva.nl/~peter/`.

[WLTB11] X. Wang, M. Liu, C. Tian, and J. Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In *Proceedings of ASIACCS*, pages 1–9. ACM, 2011. doi:10.1145/1966913.1966915.