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

A Deterministic Single Exponential Time Algorithm for Most Lattice Problems based on Voronoi Cell Computations (Extended Abstract)

Daniele Micciancio Panagiotis Voulgaris

University of California, San Diego.
Computer Science and Engineering department.
9500 Gilman Dr, Mail code 0404, La Jolla, CA 92093
Email: {daniele,pvoulgar}@cs.ucsd.edu

November 5, 2009

Abstract

We give deterministic $2^{O(n)}$-time algorithms to solve all the most important computational problems on point lattices in NP, including the Shortest Vector Problem (SVP), Closest Vector Problem (CVP), and Shortest Independent Vectors Problem (SIVP). This improves the $n^{O(n)}$ running time of the best previously known algorithms for CVP (Kannan, Math. Operation Research 12(3):415-440, 1987) and SIVP (Micciancio, Proc. of SODA, 2008), and gives a deterministic 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. In the process, we also give algorithms for several other lattice problems, including computing the kissing number of a lattice, and computing the set of all Voronoi relevant vectors. All our algorithms are deterministic, and have $2^{O(n)}$ time and space complexity

ISSN 1433-8092

1 Introduction

An n-dimensional lattice Λ is a discrete subgroup of the Euclidean space $\mathbb{R}^n$, and is customarily represented as the set of all integer linear combinations of $k \leq n$ basis vectors $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k] \in \mathbb{R}^{n \times k}$. 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 the shortest nonzero vector in the lattice generated by $\mathbf{B}$.
- The closest vector problem (CVP): given a basis $\mathbf{B}$ and a target vector $\mathbf{t} \in \mathbb{R}^n$, find the lattice vector generated by $\mathbf{B}$ that is closest to $\mathbf{t}$.
- The shortest independent vectors problem (SIVP): given a basis $\mathbf{B}$, find n linearly independent lattice vectors in $\mathbf{B}$ that are as short as possible.

Beside being classic mathematical problems in the study of the *geometry of numbers* [14], these problems play an important role in many computer science and communication theory applications. SVP and CVP have been used to solve many landmark algorithmic problems in theoretical computer science, like integer programming [34, 30], factoring polynomials over the rationals [33], checking the solvability by radicals [32], solving low density subset-sum problems [18] and breaking the Merkle-Hellman cryptosystem [44] (among many other cryptanalysis problems [29, 42].) SIVP is the main problem underlying the construction of lattice based cryptographic hash functions with worst-case/average-case connection [4, 40]. SVP and CVP also have many applications in communication theory, e.g., lattice coding for the Gaussian channel and vector quantization [17].

The computational complexity of lattice problems has been investigated intensively. All three problems mentioned above have been shown to be NP-hard both to solve exactly [52, 3, 12], or even approximate within small (constant or sub-polynomial in n) approximation factors [12, 8, 19, 13, 36, 31, 27]. Much effort has gone into the development and analysis of algorithms both to solve these problems exactly [30, 28, 25, 10, 5, 6, 11] or to efficiently find approximate solutions [33, 47, 46, 49, 48, 21, 22].

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 [17]) involve lattices in relatively small dimension, and because approximation algorithms for high dimensional lattices [46, 49, 21, 22] (for which exact solution is not feasible) typically involve the exact solution of low dimensional subproblems. The best deterministic polynomial time algorithm to solve any of these lattice problems exactly is still essentially the one discovered by Kannan [30] in 1983, running in time $n^{O(n)}$, where n is the dimension of the lattice. Subsequent work [28, 25] lead to improvements in the constant in the exponent, mostly through a better analysis, reducing the upper bound on the running time down to $n^{0.184n}$ for SVP and $n^{0.5n}$ for CVP and SIVP. The only problem that has seen asymptotically significant improvements in the exponent is SVP, for which Ajtai, Kumar and Sivakumar [5] gave a *randomized* algorithm running in time (and space) $2^{O(n)}$, typically referred to as the AKS Sieve. Following [5] much work has been devoted to better understand and improve the Sieve algorithm. Still the main questions posed in [5] didn't see much progress. Is the use of randomization (and exponential space) necessary to lower the time complexity of SVP from $n^{O(n)}$ to $2^{O(n)}$? Can algorithms with similar running time be devised for other lattice problems, like SIVP and CVP?

In [43, 41], improved analysis and variants of the AKS sieve are studied, but still using the same approach leading to randomized algorithms. Extensions of the AKS sieve algorithm to other lattice problems like CVP and SIVP have been investigated in [6, 11], 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 [11]. A possible explanation for the the difficulty of extending the result of [5] to the exact solution of SIVP and CVP was offered by Micciancio in [38], where it is shown (among other things) that CVP, SIVP and all other lattice problems considered in [11], 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 [23, 38], 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. [30] from $n^{O(n)}$ to $2^{O(n)}$. In the case of SVP, we achieve single exponential time as in [5], but without using randomization. In the process, we also provide deterministic single exponential time algorithms for various other classic computational problems in lattices, like computing the kissing number, and computing the list of all Voronoi relevant vectors.

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

1.1 Our techniques.

At the core of all our results is a new technique for the solution of the closest vector problem with preprocessing (CVPP). We recall that CVPP is a variant of CVP where some side information about the lattice is given as a hint together with the input. The hint may depend on the lattice, but not on the target vector. Typically, in the context of polynomial time algorithms, the hint is restricted to have polynomial size, but since here we study exponential time algorithms, one can reasonably consider hints that have size $2^{O(n)}$. The hint used by our algorithm is a description of the Voronoi cell of the lattice. We recall that the Voronoi cell of a lattice is the set $\mathcal{V}$ of all points (in Euclidean space) that are closer to the origin than to any other lattice point. The Voronoi cell $\mathcal{V}$ is a convex body, symmetric about the origin, and can be described as the intersection of half-spaces $H_{\mathbf{v}}$, where for any nonzero lattice point $\mathbf{v}$, $H_{\mathbf{v}}$ is the set of all points that are closer to the origin than to $\mathbf{v}$. It is not necessary to consider all $\mathbf{v} \in V \setminus \{\mathbf{0}\}$ when taking this intersection. One can restrict the intersection to the so called *Voronoi relevant* vectors, which are the lattice vectors $\mathbf{v}$ such that $\mathbf{v}/2$ is the center of a facet of $\mathcal{V}$. Since the Voronoi cell of a lattice can be shown to have at most $2(2^n - 1)$ facets, $\mathcal{V}$ can be expressed as a finite intersection of at most $2(2^n - 1)$ half-spaces. Throughout this paper, we assume that the Voronoi cell of a lattices is always described by such a list of half-spaces.

The relation between the Voronoi cell and CVPP is well known, and easy to explain. In CVPP, we want to find the lattice point $\mathbf{v}$ closest to a given target vector $\mathbf{t}$. It is easy to see that this is equivalent to finding a lattice vector $\mathbf{v}$ such that $\mathbf{t} - \mathbf{v}$ belongs to the Voronoi cell of the lattice. In other words, CVP can be equivalently formulated as the problem of finding a (typically unique) point in the set $(\mathbf{t}+\Lambda)\cap\mathcal{V}$. The idea of using the Voronoi cell to solve CVP is not new. For example, a simple greedy algorithm for CVPP based on the knowledge of the Voronoi cell of the lattice is given in [51]. The idea behind this algorithm (called the Iterative Slicer) is to make $\mathbf{t}$ shorter and shorter by subtracting Voronoi relevant vectors from it. Notice that if $\mathbf{t} \notin H_{\mathbf{v}}$, then the length of $\mathbf{t}$ can be reduced by subtracting $\mathbf{v}$ from $\mathbf{t}$. So, as long as $\mathbf{t}$ is outside $\mathcal{V}$, we can make further progress and find a shorter vector. Unfortunately, this simple and appealing algorithm to solve CVPP using the Voronoi cell is not known to perform any better than previous algorithms. [51] only proves that the algorithm terminates after a finite number of iterations, and a close inspection of the proof in [51] reveals that the best upper bound that can be derived using the methods of [51] is of the form $n^{O(n)}$: the running time of the Iterative Slicer [51] is bound by a volume argument, counting the number of lattice points of norm at most $\|\mathbf{t}\|$, and this can be well above $2^{O(n)}$ or even $n^{O(n)}$. Some of the techniques presented in this paper can be used to slightly modify the Iterative Slicer, so to guarantee $n^{O(n)}$ time complexity, but we do not know how to go below that. In order to achieve $2^{O(n)}$ running time, we need a different algorithmic approach.

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

The CVPP algorithm. Our CVPP algorithm works as follows. First we reduce the general CVPP to a special case where the target vector is guaranteed to belong to twice the Voronoi cell $2\mathcal{V}$. This can be done very easily by a polynomial time Turing reduction. Next we solve the restricted CVPP problem using a combinatorial, graph traversal approach. We recall that the goal of CVPP can be restated as the problem of finding a point $\mathbf{t}' \in \mathcal{V} \cap (\mathbf{t} + \Lambda)$ inside the Voronoi cell. (This point is also characterized as being a shortest vector in the set $\mathbf{t} + \Lambda$.) We view $\mathbf{t} + \Lambda$ as the set of nodes of an infinite graph, where two nodes are connected if the corresponding Voronoi cells share a facet. We start from a node $\mathbf{t} \in G \cap 2\mathcal{V}$, and we want to find $\mathbf{t}' \in G \cap \mathcal{V}$. If we had an explicit description of all the nodes in $G \cap 2\mathcal{V}$, then we would be done: we could simply scan the list of nodes, and select the one with smallest norm. However, we do not have such a list. So, we proceed as follows:

- We consider the finite subgraph $G' = G \cap 4\mathcal{V}$, and observe that given a node in G' it is easy to generate the list of all its neighbors in G'
- We observe that G' contains $\mathbf{t}, \mathbf{t}'$ and a total of at most 4^n nodes
- We prove that $\mathbf{t}$ and $\mathbf{t}'$ belong to the same connected component of G'.

It follows that $\mathbf{t}'$ can be found in 4^n time using any graph traversal algorithm (e.g., depth first search), starting from $\mathbf{t}$, and selecting the shortest of all nodes in its connected component. We remark that the only tricky part of the proof is showing that $\mathbf{t}, \mathbf{t}'$ belong to the same connected component of $G' = G \cap 4\mathcal{V}$. This is not hard to prove, but it is not quite trivial. For example, we do not know how to prove that $\mathbf{t}$ and $\mathbf{t}'$ are in the same connected component of $G \cap 2\mathcal{V}$, and in fact, we do not even know if that's true. Enlarging $G \cap 2\mathcal{V}$ to a bigger graph appears necessary for our proof to go through. Our proof can be easily adapted to yield a graph with at most 3^n nodes (rather than 4^n), but we do not know how to go below that.

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

Organization. The rest of the paper is organized as follow. In the next subsection we mention some additional related work. In Section 2 we give some background about lattice. Our algorithms are described and analyzed in Section 3. Section 4 concludes with a discussion of open problems and directions for future research.

1.2 Related work

Most relevant work has already been described in the introduction. Here we mention a few more related papers. The closest vector problem with preprocessing has been investigated in several papers [35, 20, 45, 16, 7], mostly with the goal of showing that CVP is NP-hard even for fixed families of lattices, or devising polynomial time approximation algorithms (with super-polynomial time preprocessing). In summary, CVPP is NP-hard to approximate for any constant (or certain subpolynomtial) factors [7], and it can be approximated in polynomial time within a factor $\sqrt{n}$ [2], at least in its distance estimation variant. Here we use CVPP mostly as a building block to give a modular description of our CVP algorithm. We use CVPP to recursively implement the preprocessing function, and then to solve the actual CVP instance. It is an interesting open problem if a similar bootstrapping can be performed using the polynomial time CVPP approximation algorithm of [2], to yield a polynomial time solution to $\sqrt{n}$-approximate CVP.

The problem of computing the Voronoi cell of a lattice is of fundamental importance in many mathematics and communication theory applications. There are several formulations of this problem. In this paper we consider the problem of generating the list of facets ($(n-1)$-dimensional faces) of the Voronoi cell, as done also in [1, 51]. Sometime one wants to generate the list of vertices (i.e., one dimensional faces), or even a complete description including all faces in dimension 1 to $n-1$. This is done in [53, 50], but it is a much more complex problem, as in general the Voronoi cell can have as many as $n! = n^{\Omega(n)}$ vertices, so they cannot be computed in single exponential time. In [50] it is also shown that computing the number of vertices of the Voronoi cell of a lattice is $\#P$-hard.

Graph traversal techniques to solve problems associated to the Voronoi cell of a lattice have recently been used in [50], but with a different goal and on a very different (in fact dual) graph. In [50] the authors present an algorithm to enumerate all the vertices of the Voronoi cell up to isomorphism, and use it to compute the covering radius of a lattice. These vectors are obtained by traversing the adjacency graph of the *Delone* cells of the lattice. This is quite different (in fact dual) to the adjacency graph of the *Voronoi* cells used in our algorithm. Beside using graph traversal methods, our work and [50] have very little in common, both in terms of algorithmic techniques, and end goals.

2 Preliminaries

In this section we give some background about lattices, and the algorithmic problems studied in this paper. For a more in-depth discussion, see [39]. The n-dimensional Euclidean space is denoted $\mathbb{R}^n$. We use bold lower case letters (e.g., $\mathbf{x}$) to denote vectors, and bold upper case letters (e.g., $\mathbf{M}$) to denote matrices. The ith coordinate of $\mathbf{x}$ is denoted x_i. For a set $S \subseteq \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^n$ and $a \in \mathbb{R}$, we let $S + \mathbf{x} = \{\mathbf{y} + \mathbf{x} \colon \mathbf{y} \in S\}$ denote the translate of S by $\mathbf{x}$, and $aS = \{a\mathbf{y} \colon \mathbf{y} \in S\}$ denote the scaling of S by a. The Euclidean norm (also known as the ℓ_2 norm) of a vector $\mathbf{x} \in \mathbb{R}^n$ is $\|\mathbf{x}\| = (\sum_i x_i^2)^{1/2}$, and the associated distance is $\operatorname{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$. The linear space spanned by a set of vectors S is denoted $\operatorname{span}(S) = \{\sum_i x_i \mathbf{s}_i \ :\ x_i \in \mathbb{R}, \mathbf{s}_i \in S\}$. The affine span of a set of vectors S is defined as $\mathbf{x} + \operatorname{span}(S - \mathbf{x})$ for any $\mathbf{x} \in S$, and does not depend on the choice of $\mathbf{x}$.

Lattices: A k-dimensional *lattice* is the set of all integer combinations

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

of k linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k$ in $\mathbb{R}^n$. The set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k$ is called a *basis* for the lattice. A basis can be represented by the matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k] \in \mathbb{R}^{n \times k}$ having the basis vectors as columns. The lattice generated by $\mathbf{B}$ is denoted $\mathcal{L}(\mathbf{B})$. Notice that $\mathcal{L}(\mathbf{B}) = \{\mathbf{B}\mathbf{x} \colon \mathbf{x} \in \mathbb{Z}^k\}$, where $\mathbf{B}\mathbf{x}$ is the usual matrix-vector multiplication. A sub-lattice of $\mathcal{L}(\mathbf{B})$ is a lattice $\mathcal{L}(\mathbf{S})$ such that $\mathcal{L}(\mathbf{S}) \subseteq \mathcal{L}(\mathbf{B})$.

The Gram-Schmidt orthogonalization of a basis $\mathbf{B}$ is the sequence of vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$, where $\mathbf{b}_i^*$ is the component of $\mathbf{b}_i$ orthogonal to $\operatorname{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$.

The following two classical algorithms are used in this paper. The LLL basis reduction algorithm [33] runs in polynomial time, and on input a lattice basis, outputs a basis for the same lattice such that $\|\mathbf{b}_{i+1}^*\|^2 \ge \|\mathbf{b}_i^*\|^2/2$. (LLL reduced bases have other properties, but this is all we need here. The Nearest Plane algorithm [9], on input a basis $\mathbf{B}$ and a target vector $\mathbf{t}$, finds a lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v} - \mathbf{t}\| \le (1/2)\sqrt{\sum_i \|\mathbf{b}_i^*\|^2}$.

Lattice problems: In this paper we are mostly concerned with the SVP, CVP and SIVP problems defined in the introduction. Our results give algorithms of several other lattice problems like the Subspace Avoiding Problem (SAP) the Generalized Closest Vector Problem (GCVP), and the Successive Minima Problem (SMP) considered in the lattice algorithms literature [11, 38]. The results for all problems other than CVP and SVP are obtained in a back-box way by reduction to CVP [38], and we refer the reader to [11, 38] for details.

For simplicity in this paper we consider only inputs to lattice problems where all the entries in the basis matrix $\mathbf{B}$ have bit size polynomial in n, i.e., $\log(\|\mathbf{B}\|) = \operatorname{poly}(n)$. 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 $\log \|\mathbf{B}\| \le M$ on the size of the entries, and letting the time and space complexity bound depend polynomially in M.

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

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

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

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

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

3 The algorithm

In this section we describe and analyze our algorithms to solve CVP, and related lattice problems. The CVP algorithm has three components:

1. A *dimension reduction* procedure that on input an n-dimensional lattice Λ, produces a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ for Λ such that for any $k = 1, \ldots, n$, the closest vector problem in the k-dimensional sub-lattice $\Lambda_k = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$ can be reduced to the solution of at most $2^{k/2}$ (or even $2^{o(k)}$) CVP computations in $\Lambda_{k-1} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})$.

2. An exponential time algorithm to solve the *closest vector problem with preprocessing* (CVPP), where the output of the lattice preprocessing function is the Voronoi cell of the input lattice, described as the intersection of half-spaces.

3. A reduction from the problem of computing the *Voronoi cell* of a lattice Λ to performing 2^n CVP computations in Λ.

Notice that the dimension reduction procedure immediately gives a recursive algorithm to solve CVP in arbitrary lattices. However, the obvious way to turn the dimension reduction procedure into a recursive program leads to an algorithm with $2^{O(n^2)}$ running time, because each time the dimension of the input lattice is reduced by 1, the number of recursive invocations gets multiplied by $2^{O(n)}$. We use the CVPP and Voronoi cell computation algorithms to give a more efficient transformation. The idea is compute the Voronoi cells of all sub-lattices $\Lambda_k = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$ sequentially, where $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is the lattice basis produced by the dimension reduction procedure. Notice that the Voronoi cell of the sub-lattice $\Lambda_1 = \mathcal{L}(\mathbf{b}_1)$ can be trivially computed, as the list of Voronoi relevant vectors is precisely $\{\mathbf{b}_1, -\mathbf{b}_1\}$. Next, assuming the Voronoi cell of Λ_{k-1} has already been computed, the Voronoi cell of Λ_k can be computed as follows:

1. Use the Voronoi cell computation algorithm to reduce the computation of the Voronoi cell of Λ_k to 2^k CVP computations in Λ_k.

2. Use the dimension reduction procedure to perform each CVP computation in Λ_k by means of $2^{k/2}$ CVP computations in Λ_{k-1}

3. Use the knowledge of the Voronoi cell of Λ_{k-1} to solve each CVP instance in Λ_{k-1} using the CVPP algorithm.

Combining the three steps together, we see that the Voronoi cell of Λ_k can be computed from the Voronoi cell of Λ_{k-1} by means of $2^{1.5k}$ CVPP computations, each running in time $2^{O(n)}$. So, the total running time to compute the Voronoi cell of Λ_n is $\sum_{k=1}^{n} 2^{1.5k} \cdot 2^{O(n)} = 2^{O(n)}$.

The following theorem immediately follows from the previous discussion, and the detailed description of the three components given in the next subsections.

Theorem 3.1 *There is a single exponential time algorithm that on input a lattice* $\mathbf{B}$*, outputs a description of the Voronoi cell of the lattice as the intersection of at most* 2^{n+1} *half-spaces.*

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

Corollary 3.2 *There is a deterministic single exponential time algorithm to solve SVP, and to compute the kissing number of a lattice.*

Once the Voronoi cell of Λ_n has been computed, then we can solve CVP using the CVPP algorithm. Both the preprocessing and CVPP computation times are $2^{O(n)}$, so the total running time to solve an arbitrary CVP instance is $2^{O(n)} + 2^{O(n)} = 2^{O(n)}$. Algorithms for other lattice problems, like CVP, SIVP, SAP, GCVP, SMP, can be obtained by reduction to CVP[38]

Corollary 3.3 *There is a deterministic single exponential time algorithm to solve CVP, SIVP, SAP, GCVP and SMP.*

3.1 The dimension reduction procedure

The dimension reduction procedure simply applies the LLL basis reduction algorithm [33] to the input lattice. This results in a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ such that $\|\mathbf{b}^*_{i+1}\|^2 \geq \|\mathbf{b}^*_i\|^2/2$ for all i, where $\mathbf{b}^*_1, \ldots, \mathbf{b}^*_n$ are the Gram-Schmidt orthogonalized vectors. We now show how such a basis allows to reduce CVP computations in Λ_{i+1} to $2^{O(i)}$ CVP computations in Λ_i.

Let $\mathbf{t}$ be a target vector. We want to find all lattice points in Λ_{i+1} closest to $\mathbf{t}$. We can assume without loss of generality that $\mathbf{t}$ belongs to the linear span of Λ_{i+1}, otherwise, we simply project $\mathbf{t}$ orthogonally to that subspace. Partition the lattice Λ_{i+1} into layers of the form $c\mathbf{b}_{i+1} + \Lambda_i$, where $c \in \mathbb{Z}$. Notice that

- the distance of $\mathbf{t}$ to Λ_{i+1} is bounded by $\rho = \frac{1}{2}\sqrt{\sum_{j=1}^{i+1} \|\mathbf{b}^*_j\|^2}$, because a lattice point within distance ρ from $\mathbf{t}$ can be computed using the nearest plane algorithm [9].
- the distance of all lattice points in the layer $c\mathbf{b}_{i+1} + \Lambda_i$ from $\mathbf{t}$ is at least $|c - c_t| \cdot \|\mathbf{b}^*_{i+1}\|$, where $c_t = \langle \mathbf{t}, \mathbf{b}^*_{i+1}\rangle / \langle \mathbf{b}^*_{i+1}, \mathbf{b}^*_{i+1}\rangle$, because this is the distance between $\mathbf{t}$ and the entire affine space generated by the layer.
- From the property $\|\mathbf{b}^*_{i+1}\|^2 \geq \|\mathbf{b}^*_i\|^2$ of LLL reduced basis we have

$$\rho = \frac{1}{2}\sqrt{\sum_{j=1}^{i+1} \|\mathbf{b}^*_j\|^2} \leq \frac{1}{2}\sqrt{\sum_{j=1}^{i+1} 2^{i+1-j}\|\mathbf{b}^*_{i+1}\|^2} = \frac{1}{2}\sqrt{2^{i+1}-1}\|\mathbf{b}^*_{i+1}\|.$$

It follows from the above observations that the lattice points in Λ_{i+1} closest to $\mathbf{t}$ belong to layers $c\mathbf{b}_{i+1} + \Lambda_i$ such that $|c - c_t| \leq \frac{1}{2}\sqrt{2^{i+1} - 1}$. So, in order to find a lattice point closest to $\mathbf{t}$ we can enumerate all $\sqrt{2^{i+1} - 1}$ integers c such that $|c - c_t| \leq \frac{1}{2}\sqrt{2^{i+1} - 1}$, and for each of them find a point in $c\mathbf{b}_{i+1} + \Lambda_i$ closest to $\mathbf{t}$. Notice that this is equivalent to finding a point in Λ_i closest to $\mathbf{t} - c\mathbf{b}_{i+1}$, i.e., a CVP computation in Λ_i. A lattice point closest to $\mathbf{t}$ is found selecting the best solution across all layers.

In summary, the dimension reduction algorithm performs a polynomial time computation to preprocess the input lattice a produce a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ such that any CVP computation in Λ_i can be reduced to $\sqrt{2^i - 1} < 2^{n/2}$ CVP computations in Λ_{i-1}.

We remark that the number of CVP subproblems in Λ_{i-1} required to solve a CVP instance in Λ_i can be reduced to $2^{o(n)}$ using better basis reduction algorithms. Also, in case there exist multiple solutions, the algorithm we just described outputs one of them. However, it is immediate to modify the algorithm to output the list of *all* solutions. The algorithm is easily modified also to output the number of solutions, without explicitly listing them: for each layer, output the distance of the best solution, and the number of solutions at that distance. Then, the results from each layer are combined by selecting the smallest distance, and adding up the number of solutions from each layer achieving the distance.

3.2 Voronoi cell computation

For this we use a simple variant of the RelevantVectors algorithm of [1] which reduces the computation of the Voronoi cell of a lattice to 2^n CVP instances, all for the same input lattice. The RelevantVectors algorithm works by iterating over all $(c_1, \ldots, c_n) \in \{0,1\}^n \setminus \{\mathbf{0}\}$, and for each one of them, do the following:

1. Find all lattice points in Λ that are closest to $\mathbf{t} = -\sum_i c_i \mathbf{b}_i/2$.
2. If there are precisely two solutions $\mathbf{v}, -(\mathbf{v} + 2\mathbf{t})$, then include $\pm 2(\mathbf{v} + \mathbf{t})$ in the list of Voronoi relevant vectors.

For details, and a proof of correctness, the reader is referred to [1]. Notice that the RelevantVectors algorithm as just described uses an oracle that finds *all* solutions to a given CVP instance. Here we remark that this is not required. It is enough to find any of them $\mathbf{v}$, and just include $\pm 2(\mathbf{v} + \mathbf{t})$ to the list. This may result in a list that contains some redundant vectors, but it is guaranteed that all Voronoi relevant vectors will be in the list, and the list size is still bounded by 2^{n+1}, which is enough to obtain our main results. If only the relevant vectors are desired, then one can run the variant of the CVP algorithm described in the previous subsection that computes the number of solutions, without listing them explicitly. If the number of solutions is precisely 2, then we also find an arbitrary solution $\mathbf{v}$, and include $\pm 2(\mathbf{v} + \mathbf{t})$ in the output list.

3.3 CVP with preprocessing

We now get to the most technical part of the algorithm. We give a single exponential time algorithm that on input a lattice Λ, a list V (of size at most 2^{n+1}) containing all Voronoi relevant vectors of Λ, and a target point $\mathbf{t}$, computes a lattice point closest to $\mathbf{t}$ in single exponential time. We remark that here "single exponential time" means single exponential in the lattice dimension n, rather than the size of V. The dependency of the running time on the length of the list V is linear. The dependency on the bit size of the target vector is polynomial.

Without loss of generality we may assume that the target vector $\mathbf{t}$ belongs to $2\bar{\mathcal{V}}$ (as justified below), the Voronoi cell of the lattice scaled up by a factor 2. The goal of the algorithm is to find a vector $\mathbf{x} \in \bar{\mathcal{V}}$ such that $\mathbf{t} - \mathbf{x} \in \Lambda$. Then, $\mathbf{t} - \mathbf{x}$ is a lattice point closest to $\mathbf{t}$. An algorithm for arbitrary target $\mathbf{t}$ (not necessarily in $2\bar{\mathcal{V}}$) is obtained as follows. Notice that membership in $c \cdot \bar{\mathcal{V}}$ (for any given c) can be efficiently tested by checking membership in each of the (scaled) half-spaces $c \cdot H_{\mathbf{v}}$ defining the Voronoi cell. Let k be an integer such that $\mathbf{t} \in 2^k\bar{\mathcal{V}}$. Such integer can be found by iteratively trying all possible values of k. Notice that $\mathbf{x}_k = \mathbf{t}$ is in $2(2^{k-1} \cdot \bar{\mathcal{V}})$, where $2^{k-1} \cdot \bar{\mathcal{V}}$ is the Voronoi cell of $2^{k-1}\Lambda$. So, we can use the basic algorithm to find a vector $\mathbf{x}_{k-1} \in 2^{k-1}\bar{\mathcal{V}}$ such that $\mathbf{x}_{k-1} - \mathbf{x}_k \in 2^k\Lambda \subset \Lambda$. We repeat this process k times, bringing the target $\mathbf{t} = \mathbf{x}_k$ into smaller and smaller multiples $2^i\bar{\mathcal{V}}$ of the Voronoi cell of the original lattice, until we get $\mathbf{x}_0 \in \bar{\mathcal{V}}$. At this point, the lattice point closest to $\mathbf{t}$ is obtained as $\mathbf{t} - \mathbf{x}_0$.

The basic CVPP algorithm (with $\mathbf{t} \in 2\bar{\mathcal{V}}$) works as follows:

1. Consider the graph (N, E), where the set of nodes is
$$N = \{\mathbf{t} + \mathbf{v} \colon \mathbf{v} \in \Lambda, (\mathbf{t} + \mathbf{v}) \in 4\mathcal{V}\}$$
and two nodes $\mathbf{x}, \mathbf{y}$ are connected by an edge iff $\mathbf{x} - \mathbf{y} \in V$. (Recall that V is a list containing all Voronoi relevant vectors.)

2. Start from $\mathbf{t} \in N$, and explore the connected component of $\mathbf{t}$. This can be done in time polynomial in the size of N and V (e.g., by performing a depth first search), because membership in N can be efficiently tested, and given a node $\mathbf{x}$, one can efficiently generate the list of its candidate neighbors by enumerating all $\mathbf{x} + \mathbf{v}$ with $\mathbf{v} \in V$.

3. Let $\mathbf{x}$ be a shortest vector in the connected component of $\mathbf{t}$ in (N, E). The output of the algorithm is $\mathbf{x}$.

Clearly, the algorithm outputs a lattice vector, and runs in single exponential time, because the length of V is at most 2^{n+1}, and N has size at most 4^n. In order to show correctness we need to prove that the vector $\mathbf{x} = \mathbf{t} + \mathbf{v}$ such that $\mathbf{x} \in \bar{\mathcal{V}}$ belongs to the same connected component of $\mathbf{t}$. This is the only technical part of our result, and it is proved in the next lemma.

Lemma 3.4 *Fix a lattice Λ, let $\mathcal{V}$ be its Voronoi cell, and $V \subset \Lambda \setminus \{\mathbf{0}\}$ a set of lattice point that contains all the Voronoi relevant vectors. For any target vector $\mathbf{t} \in 2\bar{\mathcal{V}}$, there exists a sequence of lattice points $\mathbf{v}_1 = \mathbf{0}, \ldots, \mathbf{v}_D$ such that $\mathbf{t} - \mathbf{D} \in \bar{\mathcal{V}}$, $\mathbf{t} - \mathbf{v}_{i+1} \in 4\mathcal{V}$ and $\mathbf{v}_i - \mathbf{v}_{i+1} \in V$ for all $i = 1, \ldots, D - 1$.*

Proof. Consider the segment $\delta\mathbf{t}$ (with $\delta \in [0, 1]$) connecting the target $\mathbf{t}$ with the origin, and for each δ let $\mathbf{v}_\delta$ be the center of the Voronoi cell $\mathbf{v}_\delta + \bar{\mathcal{V}}$ containing $\delta\mathbf{t}$. Let $\mathbf{v}_1, \ldots, \mathbf{v}_D$ be the sequence of lattice points so obtained, starting from $\mathbf{v}_1 = \mathbf{0}$ and ending up with a lattice vector $\mathbf{v}_D$ such that $\mathbf{t} - \mathbf{v}_D \in \bar{\mathcal{V}}$. Each center $\mathbf{v}_i$ corresponds to a node $\mathbf{x}_i = \mathbf{v}_i + \mathbf{t} - \mathbf{v}_D \in 4\mathcal{V}$ because $\delta\mathbf{t} \in 2\mathcal{V}$, $\delta\mathbf{t} - \mathbf{v}_i \in \bar{\mathcal{V}}$ and $(\mathbf{t} - \mathbf{v}_D) \in \bar{\mathcal{V}}$. (Here we are using the convexity of $\mathcal{V}$.) Assume without loss of generality that the target $\mathbf{t}$ is in general position, in the sense that none of the points $\delta\mathbf{t}$ belongs to more than 2 Voronoi cells. (This can be easily achieved by adding an infinitesimal perturbation to $\mathbf{t}$.) It follows that the difference between any two consecutive centers $\mathbf{v}_i - \mathbf{v}_{i+1}$ is a Voronoi relevant vector, and the nodes $(\mathbf{x}_i, \mathbf{x}_{i+1})$ form an edge. This gives a path from $\mathbf{x}_0 = \mathbf{t} - \mathbf{v}_D$ to $\mathbf{x}_D = \mathbf{t}$ as claimed. ■

4 Open problems and directions for further research

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

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

We described an algorithm for the ℓ_2 norm. Many parts of the algorithm easily adapt to other norms as well, but it is not clear how to extend our results to all ℓ_p norms. The main technical problem is that the Voronoi cells in ℓ_p norms for $p \neq 2$ are not convex. So, extending the algorithm to all ℓ_p norms may require some substantially new idea. An important application of extending our algorithm to other ℓ_p norms is that it would immediately lead to single exponential time algorithms for integer programming [30].

Although we didn't make any effort to optimize the constant in the exponent $2^{O(n)}$, just following the proof it is easy to see that the constant is at most 3 or 4, and we believe that it should be possible to bring it down to 2. However, it is clear that our approach cannot possibly lead to constants in the exponent smaller than 1 (as achieved for example by randomizes heuristics for SVP [43, 41].) Still, it may be possible to extend our ideas to develop an algorithm with running time proportional to the number of Voronoi relevant vectors. This may give interesting algorithms for special lattices whose Voronoi cell has a small description. Another possible research direction is to develop practical variants of our algorithm that use only a sublist of Voronoi relevant vectors, at the cost of producing only approximate solutions to CVP.

It would be nice to extend our algorithm to yield a single exponential time solution to the covering radius problem, or equivalently, the problem of computing the diameter of the Voronoi cell of a lattice. In principle, this could be done by enumerating the vertices of the Voronoi cell, and selecting the longest, but this would not lead to a single exponential time algorithm because the number of such vertices can be as large as $n^{\Omega(n)}$. No NP-hardness proof for the covering radius problem in the ℓ_2 norm is known (but see [26] 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 [37, 24] for small approximation factors. Counting the number of vertices of the Voronoi cell [50] or the number of lattice points of a given length [15] is also known to be $\#P$-hard.

References

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

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

[3] M. Ajtai. The shortest vector problem in l_2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of STOC '98*, pages 10–19. ACM, May 1998.

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

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

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

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

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

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

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

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

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

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

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

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

[16] 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.

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

[18] 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. Prelim. versions in Eurocrypt '91 and FCT '91.

[19] 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. Prelim. version in FOCS 1998.

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

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

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

[23] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[50] 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.

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

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

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

ECCC ISSN 1433-8092

http://eccc.hpi-web.de