Improved (Provable) Algorithms for the Shortest Vector Problem via Bounded Distance Decoding

Divesh Aggarwal
Centre for Quantum Technologies, Singapore
National University of Singapore, Singapore

Yanlin Chen
Institute of Information Science, Academia Sinica, Taipei, Taiwan

Rajendra Kumar
IIT Kanpur, India
National University of Singapore, Singapore

Yixin Shen
Université de Paris, IRIF, CNRS, F-75006, France

Abstract

The most important computational problem on lattices is the Shortest Vector Problem (SVP). In this paper, we present new algorithms that improve the state-of-the-art for provable classical/quantum algorithms for SVP. We present the following results.

1. A new algorithm for SVP that provides a smooth tradeoff between time complexity and memory requirement. For any positive integer $4 \leq q \leq \sqrt{n}$, our algorithm takes $q^{13n+o(n)}$ time and requires $poly(n) \cdot q^{16n/q^2}$ memory. This tradeoff which ranges from enumeration ($q = \sqrt{n}$) to sieving (q constant), is a consequence of a new time-memory tradeoff for Discrete Gaussian sampling above the smoothing parameter.
2. A quantum algorithm that runs in time $2^{0.9533n+o(n)}$ and requires $2^{0.5n+o(n)}$ classical memory and poly(n) qubits. This improves over the previously fastest classical (which is also the fastest quantum) algorithm due to [2] that has a time and space complexity $2^{n+o(n)}$.
3. A classical algorithm for SVP that runs in time $2^{1.741n+o(n)}$ time and $2^{0.5n+o(n)}$ space. This improves over an algorithm of [15] that has the same space complexity.

The time complexity of our classical and quantum algorithms are expressed using a quantity related to the kissing number of a lattice. A known upper bound of this quantity is $2^{0.402n}$, but in practice for most lattices, it can be much smaller and even $2^{o(n)}$. In that case, our classical algorithm runs in time $2^{1.292n}$ and our quantum algorithm runs in time $2^{0.750n}$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Lattices, Shortest Vector Problem, Discrete Gaussian Sampling, Time-Space Tradeoff, Quantum computation, Bounded distance decoding

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.4

Related Version *Full Version*: `https://arxiv.org/abs/2002.07955`

Funding *Divesh Aggarwal*: This work was partially supported by the Singapore Ministry of Education under grant MOE2019-T2-1-145, National Research Foundation under grant R-710-000-012-135.
Rajendra Kumar: Supported in part by National Research Foundation Singapore under its AI Singapore Programme [Award Number: AISG-RP-2018-005].

Acknowledgements We would like to thank Pierre-Alain Fouque, Paul Kirchner, Amaury Pouly and Noah Stephens-Davidowitz for useful comments and suggestions.

1 Introduction

A lattice $\mathcal{L} = \mathcal{L}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) := \{\sum_{i=1}^{n} z_i \boldsymbol{b}_i : z_i \in \mathbb{Z}\}$ is the set of all integer combinations of linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{R}^n$. We call n the rank of the lattice and $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ a basis of the lattice.

The most important computational problem on lattices is the Shortest Vector Problem (SVP). Given a basis for a lattice $\mathcal{L} \subseteq \mathbb{R}^n$, SVP asks us to compute a non-zero vector in $\mathcal{L}$ with the smallest Euclidean norm. Starting from the '80s, the use of approximate and exact solvers for SVP (and other lattice problems) gained prominence for their applications in algorithmic number theory [41], convex optimization [32, 34, 20], coding theory [17], and cryptanalysis tool [56, 14, 40]. The security of many cryptographic primitives is based on the worst-case hardness of (a decision variant of) approximate SVP to within polynomial factors [6, 44, 53, 52, 45, 23, 13] in the sense that any cryptanalytic attack on these cryptosystems that runs in time polynomial in the security parameter implies a polynomial time algorithm to solve approximate SVP to within polynomial factors. Such cryptosystems have attracted a lot of research interest due to their conjectured resistance to quantum attacks.

The SVP is a well studied computational problem in both its exact and approximate (decision) versions. By a randomized reduction, it is known to be NP-hard to approximate within any constant factor, and hard to approximate within a factor $n^{c/\log\log n}$ for some $c > 0$ under reasonable complexity-theoretic assumptions [42, 35, 27]. For an approximation factor $2^{\mathcal{O}(n)}$, one can solve SVP in time polynomial in n using the celebrated LLL lattice basis reduction algorithm [41]. In general, the fastest known algorithm(s) for approximating SVP within factors polynomial in n rely on (a variant of) the BKZ lattice basis reduction algorithm [54, 55, 7, 21, 25, 3], which can be seen as a generalization of the LLL algorithm and gives an $r^{n/r}$ approximation in $2^{\mathcal{O}(r)} \operatorname{poly}(n)$ time. All these algorithms internally use an algorithm for solving (near) exact SVP in lower-dimensional lattices. Therefore, finding faster algorithms to solve SVP is critical to choosing security parameters of cryptographic primitives.

As one would expect from the hardness results above, all known algorithms for solving exact SVP, including the ones we present here, require at least exponential time. In fact, the fastest known algorithms also require exponential space. There has been some recent evidence [4] showing that one cannot hope to get a $2^{o(n)}$ time algorithm for SVP if one believes in complexity theoretic conjectures such as the (Gap) Exponential Time Hypothesis. Most of the known algorithms for SVP can be broadly classified into two classes: (i) the algorithms that require memory polynomial in n but run in time $n^{\mathcal{O}(n)}$ and (ii) the algorithms that require memory $2^{\mathcal{O}(n)}$ and run in time $2^{\mathcal{O}(n)}$.

The first class, initiated by Kannan [34, 28, 26, 22, 48], combines basis reduction with exhaustive enumeration inside Euclidean balls. While enumerating vectors requires $2^{\mathcal{O}(n \log n)}$ time, it is much more space-efficient than other kinds of algorithms for exact SVP.

Another class of algorithms, and currently the fastest, is based on sieving. First developed by Ajtai, Kumar, and Sivakumar [7], they generate many lattices vectors and then divide-and-sieve to create shorter and shorter vectors iteratively. A sequence of improvements [51, 49, 46, 50, 2, 5], has led to a $2^{n+o(n)}$ time and space algorithm by sieving the lattice vectors and carefully controlling the distribution of output, thereby outputting a set of lattice vectors that contains the shortest vector with overwhelming probability.

An alternative approach using the Voronoi cell of the lattice was proposed by Micciancio and Voulgaris [47] and gives a deterministic $2^{2n+o(n)}$-time and $2^{n+o(n)}$-space algorithm for SVP (and many other lattice problems).

There are variants [49, 46, 39, 11] of the above mentioned sieving algorithms that, under some heuristic assumptions, have an asymptotically smaller (but still $2^{\Theta(n)}$) time and space complexity than their provable counterparts.

Algorithms giving a time/space tradeoff

Even though sieving algorithms are asymptotically the fastest known algorithms for SVP, the memory requirement, in high dimension, has historically been a limiting factor to run these algorithms. Some recent works [18, 8] have shown how to use new tricks to make it possible to use sieving on high-dimensional lattices in practice and benefit from their efficient running time [57].

Nevertheless, it would be ideal and has been a long standing open question to obtain an algorithm that achieves the "best of both worlds", i.e. an algorithm that runs in time $2^{\mathcal{O}(n)}$ and requires memory polynomial in n. In the absence of such an algorithm, it is desirable to have a smooth tradeoff between time and memory requirement that interpolates between the current best sieving algorithms and the current best enumeration algorithms.

To this end, Bai, Laarhoven, and Stehlé [10] proposed the tuple sieving algorithm, providing such a tradeoff based on heuristic assumptions similar in nature to prior sieving algorithms. They conjectured a running time $k^{n+o(n)}$ and space complexity $k^{n/k+o(n)}$. One can vary the parameter k to obtain a smooth time/space tradeoff. Nevertheless, it is still desirable to obtain a provable variant of this algorithm that does not rely on any heuristics. The complexity of this algorithm was later proven, under the same heuristic assumptions [29], but only for constant k, therefore leaving the subexponential memory regime open.

Kirchner and Fouque [36] attempted to do this. They claim an algorithm for solving SVP in time $q^{\Theta(n)}$ and in space $q^{\Theta(n/q)}$ for any positive integer $q > 1$. Unfortunately, their analysis falls short of supporting their claimed result, and the correctness of the algorithm is not clear. We refer the reader to the full version of the paper for more details.

In addition to the above, Chen, Chung, and Lai [15] propose a variant of the algorithm based on Discrete Gaussian sampling in [2]. Their algorithm runs in time $2^{2.05n+o(n)}$ and the memory requirement is $2^{0.5n+o(n)}$. The quantum variant of their algorithm runs in time $2^{1.2553n+o(n)}$ time and has the same space complexity. Their algorithm has the best space complexity among known provably correct algorithms that run in time $2^{O(n)}$.

A number of works have also investigated the potential quantum speedups for lattice algorithms, and SVP in particular. A similar landscape to the classical one exists, although the quantum memory model has its importance. While quantum enumeration algorithms only require qubits [9], sieving algorithms require more powerful QRAMs [39, 37].

1.1 Our results

We first present a new algorithm for SVP that provides a smooth tradeoff between the time complexity and memory requirement of SVP without any heuristic assumptions. This algorithm is obtained by giving a new algorithm for sampling lattice vectors from the Discrete Gaussian distribution that runs in time $q^{\mathcal{O}(n)}$ and requires $q^{\mathcal{O}(n/q^2)}$ space.

▶ **Theorem 1** (Time-space tradeoff for smooth discrete Gaussian, informal). *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$, a positive integer q, and a parameter s above the smoothing parameter of $\mathcal{L}$, and outputs q^{16n/q^2} samples from $D_{\mathcal{L},s}$ using $q^{13n+o(n)}$ time and poly$(q) \cdot q^{16n/q^2}$ space.*

Using the standard reduction from Bounded Distance Decoding (BDD) with preprocessing (where an algorithm solving the problem is allowed unlimited preprocessing time on the lattice before the algorithm receives the target vector) to Discrete Gaussian Sampling (DGS) from [16] and a reduction from SVP to BDD given in [15], we obtain the following.

▶ **Theorem 2** (Time-space tradeoff for SVP). *Let $n \in \mathbb{N}, q \in [4, \sqrt{n}]$ be a positive integer. Let $\mathcal{L}$ be the lattice of rank n. There is a randomized algorithm that solves SVP in time $q^{13n+o(n)}$ and in space $poly(n) \cdot q^{\frac{16n}{q^2}}$.*

If we take $k = q^2$, then the time complexity of the previous SVP algorithm becomes $k^{6.5n+o(n)}$ and the space complexity $\mathrm{poly}(n) \cdot k^{(8n/k)}$. Our tradeoff is thus the same (up to a constant in the exponents) as what was claimed by Kirchner and Fouque [36] and proven in [29] *under heuristic assumptions.*

Our second result is a quantum algorithm for SVP that improves over the current *fastest quantum algorithm* for SVP [2] (Notice that the algorithm in [2] is still the fastest classical algorithm for SVP).

▶ **Theorem 3** (Quantum Algorithm for SVP). *There is a quantum algorithm that solves SVP in $2^{0.9533n+o(n)}$ time and classical $2^{0.5n+o(n)}$ space with an additional number of qubits polynomial in n.*

Our third result is a classical algorithm for SVP that improves over the algorithm from [15] and results in the fastest classical algorithm that has a space complexity $2^{0.5n+o(n)}$.

▶ **Theorem 4** (Algorithm for SVP with $2^{0.5n+o(n)}$ space). *There is a classical algorithm that solves SVP in $2^{1.740n+o(n)}$ time and $2^{0.5n+o(n)}$ space.*

The time complexity of our second and third results are obtained using a quantity related to the kissing number of a lattice. A known upper bound of this quantity is $2^{0.402n}$, but in practice for most lattices, it can be much smaller and even $2^{o(n)}$. In that case, our classical algorithm runs in time $2^{1.292n}$ and our quantum algorithm runs in time $2^{0.750n}$. See Section 5 of the full version of the paper for more details [1].

We summarize known provable Classical and Quantum algorithms in Table 1. Note that all the classical algorithms are also quantum algorithms but they don't use any quantum power.

Table 1 Comparison of algorithms for the Shortest vector problem. [39] uses the quantum RAM model. [15] and our quantum algorithm need only polynomial qubits and $2^{0.5n+o(n)}$ classical space.

Classical Algorithms		
Time	**Space**	**Reference**
$n^{n+o(n)}$	$\mathrm{poly}(n)$	[34]
$2^{n+o(n)}$	$2^{n+o(n)}$	[2]
$2^{2.05n+o(n)}$	$2^{0.5n+o(n)}$	[15]
$2^{1.741n+o(n)}$	$2^{0.5n+o(n)}$	This paper

Quantum Algorithms		
Time	**Space**	**Reference**
$2^{1.799n+o(n)}$	$2^{1.286n+o(n)}$, QRAM	[39]
$2^{1.2553n+o(n)}$	$2^{0.5n+o(n)}$	[15]
$2^{0.9533n+o(n)}$	$2^{0.5n+o(n)}$	This paper

▶ Remark 5 (Magic constants). Most of the constants that appear in this paper were calculated by optimising the complexity with respect to a quantity related to the kissing number and then instantiating with $b = 0.402$, the best known upper-bound on this quantity. The details of these calculations are available in the full version, section 5.

Roadmap

In the following, we give a high-level overview of our proofs in Section 1.2. Section 2 contain some preliminaries on lattices. The proofs of the time-space tradeoff for Discrete Gaussian sampling above the smoothing parameter and the time-space tradeoff for SVP are given in Section 3. Our classical and quantum algorithms for solving SVP with space complexity $2^{0.5n+o(n)}$ are presented in Section 4. We also shows how the time complexity of our algorithms varies with a quantity related to the kissing number in Section 5 of the full version of the paper [1].

1.2 Proof overview

We now include a high-level description of our proofs. Before describing our proof ideas, we emphasize that it was shown in [16, 2] that given an algorithm for DGS a constant factor c above the smoothing parameter, we can solve the problem of BDD where the target vector is within distance $\alpha\lambda_1(\mathcal{L})$ of the lattice, where the constant $\alpha < 0.5$ depends on the constant c. Additionally, using [15], one can enumerate all lattice points within distance $p\delta$ to a target $\boldsymbol{t}$ by querying p^n times a BDD oracle with decoding distance δ (or $p^{n/2}$ times if we are given a quantum BDD oracle). Thus, by choosing $p = \lceil \lambda_1(\mathcal{L})/\delta \rceil$ and $\boldsymbol{t} = \mathbf{0}$, an algorithm for BDD immediately gives us an algorithm for SVP. Therefore, it suffices to give an algorithm for DGS above the smoothing parameter.

1.2.1 Time-space tradeoff for DGS above smoothing

Recall that efficient algorithms are known for sampling from a discrete Gaussian with a large enough parameter (width) [38, 24, 12]. In [2], the authors begin by sampling $N = 2^{n+o(n)}$ vectors from the Discrete Gaussian distribution with (large) parameter s and then look for pairs of vectors whose sum is in $2\mathcal{L}$, or equivalently pairs of vectors that lie in the same coset $\boldsymbol{c} \in \mathcal{L}/2\mathcal{L}$. Since there are 2^n cosets, if we take $\Omega(2^n)$ samples from $D_{\mathcal{L},s}$, almost all of the resulting vectors (except at most 2^n vectors) will be paired and are statistically close to independent samples from the distribution $D_{\mathcal{L},s/\sqrt{2}}$, provided that the parameter s is sufficiently above the smoothing parameter.

To reduce the space complexity, we modify the algorithm by generating random samples and checking if the sum of d of those samples is in $q\mathcal{L}$ for some integer q. Intuitively, if we start with two lists of vectors (L_1 and L_2) of size $q^{\mathcal{O}(n/d)}$ from $D_{\mathcal{L},s}$, where s is sufficiently above the smoothing parameter, each of these vectors is contained in any coset $q\mathcal{L}+\boldsymbol{c}$ for any $\boldsymbol{c} \in \mathcal{L}/q\mathcal{L}$ with probability roughly $1/q^n$. We therefore expect that the coset of a uniformly random d-combination of vectors from L_2 is uniformly distributed in $\mathcal{L}/q\mathcal{L}$. The proof of this statement follows from the Leftover Hash Lemma [31]. We therefore expect that for any vector $\boldsymbol{v} \in L_1$, with high probability, there is a set of d vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d$ in L_2 that sum to a vector in $q\mathcal{L}+\boldsymbol{v}$, and hence $\frac{1}{q}\left(\sum_{i=1}^d \boldsymbol{x}_i - \boldsymbol{v}\right) \in \mathcal{L}$. A lemma by Micciancio and Peikert ([43]) shows that this vector is statistically close to a sample from the distribution $D_{\mathcal{L},s\sqrt{d+1}/q}$. We can find such a combination by trying all subsets of d vectors.

We would like to repeat this and find $q^{\mathcal{O}(n/d)}$ (nearly) independent vectors in $q\mathcal{L}$. It is not immediately clear how to continue since, in order to guarantee independence, one would not want to reuse the already used vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d$ and conditioned on the choice of these vectors, the distribution of the cosets containing the remaining vectors is disturbed and is no longer nearly uniform. By using a simple combinatorial argument, we show that even after removing any $1/\operatorname{poly}(d)$ fraction of vectors from the list L_2, the d-combination of vectors in L_2 has at least cq^n different cosets. This is sufficient to output $q^{\mathcal{O}(n/d)}$ independent vectors in $q\mathcal{L}$ with overwhelming probability.

1.2.2 A new algorithm for BDD with preprocessing leads to a faster quantum algorithm for SVP

This result improves the quantum algorithm from [15]. As mentioned above, a BDD oracle from discrete Gaussian sampling can have a decoding distance $\alpha\lambda_1(\mathcal{L})$ with $\alpha < 0.5$, and, using [15], one needs to enumerate all lattice points within distance $p\alpha\lambda_1(\mathcal{L})$ to a target $\boldsymbol{t}$ by querying p^n times a BDD oracle with decoding distance $\alpha\lambda_1(\mathcal{L})$ (or $p^{n/2}$ times if we are given a quantum BDD oracle). Hence, we need to take $p = 3$ so that $p\alpha\lambda_1(\mathcal{L}) \geqslant \lambda_1(\mathcal{L})$, and the search space is at least 3^n, or $3^{n/2}$ quantum queries. Thus, towards optimizing the algorithm for SVP, one should aim to solve α-BDD for α slightly larger than 1/3 since a larger value of α will still lead to the same running time for SVP. Using known bounds, it can be shown that such an algorithm requires $2^{0.1605n+o(n)}$ independent (preprocessed) samples from $D_{\mathcal{L},\eta_\varepsilon(\mathcal{L})}$[1] for $\varepsilon = 2^{-cn}$ for some constant c.

In [2], the authors gave an algorithm that runs in time $2^{n/2+o(n)}$ and outputs $2^{n/2+o(n)}$ samples from $D_{\mathcal{L},s}$ for any $s \geq \sqrt{2}\eta_{0.5}(\mathcal{L})$, i.e. a factor $\sqrt{2}$ above the smoothing parameter). In order to obtain samples at the smoothing parameter, we construct a dense lattice $\mathcal{L}'$ of smaller smoothing parameter than $\mathcal{L}$. We then sample $2^{0.5n+o(n)}$ vectors from $D_{\mathcal{L}',s}$ and reject those that are not in $\mathcal{L}$. Using the reduction from BDD to DGS, and by repeating this algorithm, we obtain a $2^{0.661n+o(n)}$ time and $2^{0.5n+o(n)}$-space algorithm to solve 1/3-BDD with preprocessing, where each call to BDD requires $2^{0.161n+o(n)}$ time. Thus, the total time complexity of the classical algorithm is $3^n \cdot 2^{0.161n+o(n)}$, and that of the corresponding quantum algorithm is $3^{n/2} \cdot 2^{0.161n+o(n)}$.

1.2.3 Covering surface of a ball by spherical caps

As we mentioned above, one can enumerate all lattice points within a $p\delta$ distance to a target $\boldsymbol{t}$ by querying p^n times a BDD oracle with decoding distance δ. Our algorithm for BDD is obtained by preparing samples from the discrete Gaussian distribution. However, note that the decoding distance of BDD oracle built by discrete Gaussian samples as shown in [16] is successful if the target vector is within a radius $\alpha\lambda_1(\mathcal{L})$ for $\alpha < 1/2$ (there is a tradeoff between α and the number of DGS samples needed), and therefore, if we choose $\boldsymbol{t}$ to be $\mathbf{0}$, as we do in the other algorithms mentioned above, then p has to be at least 3 to ensure that the shortest vector is one of the vectors output by the enumeration algorithm. We observe here that if we choose a target $\boldsymbol{t}$ to be a random vector "close to" but not at the origin, then the shortest vector will be within a radius 2δ from the target $\boldsymbol{t}$ with some probability P, and thus we can find the shortest vector by making $2^n/P$ calls to the BDD oracle. An appropriate choice of the target $\boldsymbol{t}$ and the factor α gives an algorithm that runs in time $2^n \cdot 2^{0.74n+o(n)}$, which is faster than the algorithm (running in time $3^n 2^{0.161n+o(n)}$) mentioned above.

We note that the corresponding quantum algorithm runs in time $2^{n/2} \cdot 2^{0.74n+o(n)}$, which is significantly slower than the quantum algorithm mentioned above.

We also note that the running time of this algorithm crucially depends on a quantity related to the kissing number of a lattice. Since a tight bound on this quantity is not known, the actual running time of this algorithm might be smaller than that promised above. For a more elaborate discussion on this, see Section 5 of the full version [1].

[1] The number of samples depends on a quantity related to the kissing number of a lattice, we used the best known upper bound on this quantity due to [33].

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \dots, \}$. We use bold letters $\boldsymbol{x}$ for vectors and denote a vector's coordinates with indices x_i. We use log to represent the logarithm base 2 and ln to represent the natural logarithm. Throughout the paper, n will always be the dimension of the ambient space $\mathbb{R}^n$.

Lattices

A *lattice* $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^n$, *i.e.* the set $\mathcal{L}(\boldsymbol{b}_1, \dots, \boldsymbol{b}_n) = \{\sum_{i=1}^{m} x_i \boldsymbol{b}_i \ : \ x_i \in \mathbb{Z}\}$ of all integer combinations of m linearly independent vectors $\boldsymbol{b}_1, \dots, \boldsymbol{b}_n \in \mathbb{R}^n$. Such $\boldsymbol{b}_i$'s form a *basis* of $\mathcal{L}$. The lattice $\mathcal{L}$ is said to be *full-rank* if $n = m$. We denote by $\lambda_1(\mathcal{L})$ the first minimum of $\mathcal{L}$, defined as the length of a shortest non-zero vector of $\mathcal{L}$.

For a rank n lattice $\mathcal{L} \subset \mathbb{R}^n$, the *dual lattice*, denoted $\mathcal{L}^*$, is defined as the set of all points in $\mathsf{span}(\mathcal{L})$ that have integer inner products with all lattice points, $\mathcal{L}^* = \{\vec{w} \in \mathsf{span}(\mathcal{L}) : \forall \vec{y} \in \mathcal{L}, \langle \vec{w}, \vec{y} \rangle \in \mathbb{Z}\}$. Similarly, for a lattice basis $\mathbf{B} = (\vec{b}_1, \dots, \vec{b}_n)$, we define the dual basis $\mathbf{B}^* = (\vec{b}_1^*, \dots, \vec{b}_n^*)$ to be the unique set of vectors in $\mathsf{span}(\mathcal{L})$ satisfying $\langle \vec{b}_i^*, \vec{b}_j \rangle = 1$ if $i = j$, and 0, otherwise. It is easy to show that $\mathcal{L}^*$ is itself a rank n lattice and $\mathbf{B}^*$ is a basis of $\mathcal{L}^*$. Given a lattice $\mathbf{B} = (\vec{b}_1, \dots, \vec{b}_n)$, we denote $\| \mathbf{B} \|_2 = \max_i \|b_i\|$.

Probability distributions

Given two random variables X and Y on a set E, we denote by d_{SD} the *statistical distance* between X and Y, which is defined by

$$\mathrm{d}_{\mathrm{SD}}(X, Y) = \frac{1}{2} \sum_{z \in E} \left| \Pr_X[X = z] - \Pr_Y[Y = z] \right| = \sum_{\substack{z \in E : \\ \Pr_X[X=z] > \Pr_Y[Y=z]}} \left(\Pr_X[X = z] - \Pr_Y[Y = z] \right) .$$

We write X is ε-close to Y to denote that the statistical distance between X and Y is at most ε. Given a finite set E, we denote by U_E a uniform random variable on E, i.e., for all $x \in E$, $\Pr_{U_E}[U_E = x] = \frac{1}{|E|}$.

Discrete Gaussian Distribution

For any $s > 0$, define $\rho_s(\boldsymbol{x}) = \exp(-\pi \|\boldsymbol{x}\|^2 / s^2)$ for all $\boldsymbol{x} \in \mathbb{R}^n$. We write ρ for ρ_1. For a discrete set S, we extend ρ to sets by $\rho_s(S) = \sum_{\boldsymbol{x} \in S} \rho_s(\boldsymbol{x})$. Given a lattice $\mathcal{L}$, the *discrete Gaussian* $D_{\mathcal{L},s}$ is the distribution over $\mathcal{L}$ such that the probability of a vector $\boldsymbol{y} \in \mathcal{L}$ is proportional to $\rho_s(\boldsymbol{y})$: $\Pr_{X \sim D_{\mathcal{L},s}}[X = \boldsymbol{y}] = \frac{\rho_s(\boldsymbol{y})}{\rho_s(\mathcal{L})}$.

2.1 Lattice problems

The following problem plays a central role in this paper.

▶ **Definition 6.** *For $\delta = \delta(n) \geq 0$, σ a function that maps lattices to non-negative real numbers, and $m = m(n) \in \mathbb{N}$, δ-DGS_σ^m (the Discrete Gaussian Sampling problem) is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a parameter $s > \sigma(\mathcal{L})$. The goal is to output a sequence of m vectors whose joint distribution is δ-close to m independent samples from $D_{\mathcal{L},s}$.*

We omit the parameter δ if $\delta = 0$, and the parameter m if $m = 1$. We stress that δ bounds the statistical distance between the *joint* distribution of the output vectors and m independent samples from $D_{\mathcal{L},s}$. We consider the following lattice problems.

▶ **Definition 7.** *The search problem SVP (Shortest Vector Problem) is defined as follows: The input is a basis* $\mathbf{B}$ *for a lattice* $\mathcal{L} \subset \mathbb{R}^n$. *The goal is to output a vector* $\boldsymbol{y} \in \mathcal{L}$ *with* $\|\vec{y}\| = \lambda_1(\mathcal{L})$.

▶ **Definition 8.** *The search problem CVP (Closest Vector Problem) is defined as follows: The input is a basis* $\mathbf{B}$ *for a lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and a target vector* $\vec{t} \in \mathbb{R}^n$. *The goal is to output a vector* $\vec{y} \in \mathcal{L}$ *with* $\|\vec{y} - \vec{t}\| = \mathsf{dist}(\vec{t}, \mathcal{L})$.

▶ **Definition 9.** *For* $\alpha = \alpha(n) < 1/2$, *the search problem* α*-BDD (Bounded Distance Decoding) is defined as follows: The input is a basis* $\mathbf{B}$ *for a lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and a target vector* $\vec{t} \in \mathbb{R}^n$ *with* $\mathsf{dist}(\boldsymbol{t}, \mathcal{L}) \le \alpha \cdot \lambda_1(\mathcal{L})$. *The goal is to output a vector* $\vec{y} \in \mathcal{L}$ *with* $\|\vec{y} - \vec{t}\| = \mathsf{dist}(\vec{t}, \mathcal{L})$.

Note that while our other problems become more difficult as the approximation factor γ becomes smaller, α-BDD becomes more difficult as α gets larger.

For convenience, when we discuss the running time of algorithms solving the above problems, we ignore polynomial factors in the bit-length of the individual input basis vectors (i.e. we consider only the dependence on the ambient dimension n).

For a lattice $\mathcal{L}$ and $\varepsilon > 0$, the *smoothing parameter* $\eta_\varepsilon(\mathcal{L})$ is the smallest s such that $\rho_{1/s}(\mathcal{L}^*) = 1 + \varepsilon$. Recall that if $\mathcal{L}$ is a lattice and $\boldsymbol{v} \in \mathcal{L}$ then $\rho_s(\mathcal{L} + \boldsymbol{v}) = \rho_s(\mathcal{L})$ for all s. The *smoothing parameter* has the following well-known property.

▶ **Lemma 10** ([53, Claim 3.8]). *For any lattice* $\mathcal{L} \subset \mathbb{R}^n$, $\boldsymbol{c} \in \mathbb{R}^n$, $\varepsilon > 0$, *and* $s \ge \eta_\varepsilon(\mathcal{L})$,

$$\frac{1-\varepsilon}{1+\varepsilon} \le \frac{\rho_s(\mathcal{L} + \boldsymbol{c})}{\rho_s(\mathcal{L})} \le 1 \ .$$

▶ **Corollary 11** ([1, Corollary 10]). *Let* $\mathcal{L} \subset \mathbb{R}^n$ *be a lattice,* q *be a positive integer, and let* $s \ge \eta_\varepsilon(q\,\mathcal{L})$. *Let* C *be a random coset in* $\mathcal{L}/q\,\mathcal{L}$ *sampled such that* $\Pr[C = q\,\mathcal{L} + \boldsymbol{c}] = \frac{\rho_s(q\,\mathcal{L} + \boldsymbol{c})}{\rho_s(\mathcal{L})}$. *Also, let* U *be a coset in* $\mathcal{L}/q\,\mathcal{L}$ *sampled uniformly at random. Then* $\mathrm{d}_{SD}(C, U) \le 2\varepsilon$.

The following lemma gives a bound on the smoothing parameter.

▶ **Lemma 12** ([2, Lemma 2.7]). *For any lattice* $\mathcal{L} \subset \mathbb{R}^n, \varepsilon \in (0,1)$ *and* $k > 1$, *we have* $k\eta_\varepsilon(\mathcal{L}) > \eta_{\varepsilon^{k^2}}(\mathcal{L})$

Micciancio and Peikert [43] showed the following result about resulting distribution from the sum of many Gaussian samples.

▶ **Theorem 13** ([43, Theorem 3.3]). *Let* $\mathcal{L}$ *be an* n *dimensional lattice,* $\boldsymbol{z} \in \mathbb{Z}^m$ *a nonzero integer vector,* $s_i \ge \sqrt{2}\|\boldsymbol{z}\|_\infty \cdot \eta_\varepsilon(\mathcal{L})$, *and* $\mathcal{L} + \boldsymbol{c_i}$ *arbitrary cosets of* $\mathcal{L}$ *for* $i = 1 \cdots, m$. *Let* $\boldsymbol{y_i}$ *be independent vectors with distributions* $D_{\mathcal{L} + \boldsymbol{c_i}, s_i}$, *respectively. Then the distribution of* $\boldsymbol{y} = \sum\limits_i z_i \boldsymbol{y_i}$ *is* $m\varepsilon$ *close to* $D_{Y,s}$, *where* $Y = gcd(\boldsymbol{z})\,\mathcal{L} + \sum\limits_i z_i \boldsymbol{c_i}$, *and* $s = \sqrt{\sum (z_i s_i)^2}$.

We will need the following reduction from α-BDD to DGS that was shown in [16].

▶ **Theorem 14** ([16, Theorem 3.1], [2, Theorem 7.3]). *For any* $\varepsilon \in (0, 1/200)$, *let* $\phi(\mathcal{L}) \equiv \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)}$. *Then, there exists a randomized reduction from* CVP^ϕ *to* 0.5*-*$\mathsf{DGS}^m_{\eta_\varepsilon}$, *where* $m = \mathcal{O}(\frac{n \log(1/\varepsilon)}{\sqrt{\varepsilon}})$ *and* CVP^ϕ *is the problem of solving* CVP *for target vectors that are guaranteed to be within a distance* $\phi(\mathcal{L})$ *of the lattice. The reduction preserves the dimension, makes a single call to the* DGS *oracle, and runs in time* $m \cdot poly(n)$. *Furthermore, the reduction always reduces an instance of* CVP^ϕ *on a lattice* $\mathcal{L}$ *to an instance of* DGS *on the dual lattice* $\mathcal{L}^*$.

We need the following relation between the first minimum of lattice and the smoothing parameter of dual lattice. We will use this to compute the decoding distance of BDD oracle.

▶ **Lemma 15** ([2, Lemma 6.1]). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, $\varepsilon \in (0,1)$, we have*

$$\sqrt{\frac{\ln(1/\varepsilon)}{\pi}} < \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\beta^2 n}{2\pi e}} \cdot \varepsilon^{-1/n} \cdot (1 + o(1)), \tag{1}$$

and if $\varepsilon \leq (e/\beta^2 + o(1))^{-\frac{n}{2}}$, we have

$$\sqrt{\frac{\ln(1/\varepsilon)}{\pi}} < \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\ln(1/\varepsilon) + n\ln\beta + o(n)}{\pi}}. \tag{2}$$

where $\beta(\mathcal{L}) = \inf\{\gamma : \forall r \geq 1,\ N(\mathcal{L}, r\lambda_1(\mathcal{L})) \leq (\gamma r)^n\}$. In particular $\beta(\mathcal{L}) \in [1, 2^{0.402}]$.

The following theorem proved in [15], is required to solve SVP by exponential number of calls to α-BDD oracle.

▶ **Theorem 16** ([15, Theorem 8]). *Given a basis matrix $\mathbf{B} \subset \mathbb{R}^{n\times n}$ for lattice $\mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$, a target vector $\boldsymbol{t} \in \mathbb{R}^n$, an α-BDD oracle BDD_α with $\alpha < 0.5$, and an integer scalar $p > 0$. Let $f_p^\alpha : \mathbb{Z}_p^n \to \mathbb{R}^n$ be $f_p^\alpha(\boldsymbol{s}) = -p \cdot BDD_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{s} - \boldsymbol{t})/p) + \mathbf{B}\,\boldsymbol{s}$. If $\mathsf{dist}(\mathcal{L}, \boldsymbol{t}) \leq \alpha\lambda_1(\mathcal{L})$, then the list $m = \{f_p^\alpha(\boldsymbol{s}) \mid \boldsymbol{s} \in \mathbb{Z}_p^n\}$ contains all lattice points within distance $p\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$.*

We will need the following theorems to sample the DGS vectors with a large width.

▶ **Theorem 17** ([2],Proposition 2.17). *For any $\varepsilon \leq 0.99$, there is an algorithm that takes as input a lattice $\mathcal{L} \in \mathbb{R}^n$, $M \in \mathbb{Z}_{>0}$ (the desired number of output vectors), and $s > 2^{n\log\log n/\log n} \cdot \eta_\varepsilon(\mathcal{L})$, and outputs M independent samples from $D_{\mathcal{L},s}$ in time $M \cdot poly(n)$.*

▶ **Theorem 18** ([2, Theorem 5.11]). *For a lattice $\mathcal{L} \subset \mathbb{R}^n$, let $\sigma(\mathcal{L}) = \sqrt{2}\eta_{1/2}(\mathcal{L})$. Then there exists an algorithm that solves $\exp(-\Omega(\kappa))$-$DGS_\sigma^{2^{n/2}}$ in time $2^{n/2+\mathrm{polylog}(\kappa)+o(n)}$ with space $\mathcal{O}(2^{n/2})$ for any $\kappa \geq \Omega(n)$. Moreover, if the input does not satisfy the promise, and the input parameter $s < \sigma(\mathcal{L}) = \sqrt{2}\eta_{1/2}(L)$, then the algorithm may output M vectors for some $M \leq 2^{n/2}$ that are $\exp(-\Omega(\kappa))$-close to M independent samples from $D_{\mathcal{L},s}$.*

▶ **Lemma 19** ([2, Lemma 5.12]). *There is a probabilistic polynomial-time algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ of rank n and an integer a with $n/2 \leq a < n$ and returns a super lattice $\mathcal{L}' \supset \mathcal{L}$ of index 2^a with $\mathcal{L}' \subseteq \mathcal{L}/2$ such that for any $\varepsilon \in (0,1)$, we have $\eta_{\varepsilon'}(\mathcal{L}') \leq \eta_\varepsilon(\mathcal{L})/\sqrt{2}$ with probability at least $1/2$ where $\varepsilon' := 2\varepsilon^2 + 2^{(n/2)+1-a}(1+\varepsilon)$.*

2.2 Probability

We need the following lemma on distribution of vector inner product which directly follows from the Leftover Hash Lemma [31].

▶ **Lemma 20.** *Let $\mathbb{G}$ be a finite abelian group, and let f be a positive integer. Let $\mathcal{Y} \subseteq \{0,1\}^f$. Define the inner product $\langle\cdot,\cdot\rangle : \mathbb{G}^f \times \mathcal{Y} \to \mathbb{G}$ by $\langle x, y\rangle = \sum_i x_i y_i$ for all $x \in \mathbb{G}^f, y \in \mathcal{Y}$. Let X, Y be independent and uniformly random variables on $\mathbb{G}^f, \mathcal{Y}$, respectively. Then $\mathrm{d}_{SD}((\langle X, Y\rangle, X), (U_\mathbb{G}, X)) \leq \frac{1}{2} \cdot \sqrt{\frac{|\mathbb{G}|}{|\mathcal{Y}|}}$, where $U_\mathbb{G}$ is uniform in $\mathbb{G}$ and independent of X.*

We will also need the Chernoff-Hoeffding bound [30].

▶ **Lemma 21.** *Let $X_1, \ldots, X_M$ be the independent and identically distributed random boolean variables of expectation p. Then for $\varepsilon > 0$, $\Pr\left[\frac{1}{M}\sum_{i=1}^{M} X_i \leq p(1-\delta)\right] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{pM}$.*

For preliminaries on quantum computing, see [1, Section 2.2]

3 Algorithms with a time-memory tradeoff for lattice problems

In this section, we present a new algorithm for Discrete Gaussian sampling above the smoothing parameter.

3.1 Algorithm for Discrete Gaussian Sampling

We now present the main result of this section.

▶ **Theorem 22.** *Let $n \in \mathbb{N}, q \geq 2, d \in [1,n]$ be positive integers, and let $\varepsilon > 0$. Let C be any positive integer. Let $\mathcal{L}$ be a lattice of rank n, and let $s \geq 2\sqrt{d}\eta_\varepsilon(q\,\mathcal{L}) = 2\sqrt{d}q\eta_\varepsilon(\mathcal{L})$. There is an algorithm that, given $N = 160d^2 \cdot C \cdot q^{n/d}$ independent samples from $D_{\mathcal{L},s}$, outputs a list of vectors that is $(10d\varepsilon^{2d}N + 11Cq^{-5n/2})$-close to $Cq^{n/d}$ independent vectors from $D_{\mathcal{L},\frac{\sqrt{8d+1}}{q}s}$. The algorithm runs in time $C \cdot (10e \cdot d)^{8d} \cdot q^{8n+n/d+o(n)}$ and requires memory $\mathrm{poly}(d) \cdot q^{n/d}$ excluding the input and output memory.*

Proof. We prove the result for $C = 1$, and the general result follows by repeating the algorithm. Let $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ be the N input vectors and let $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_N\}$ be the corresponding cosets in $\mathcal{L}/q\,\mathcal{L}$. The algorithm does the following:

1. Initialize two lists $L_1 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\frac{N}{2}}\}$ and $L_2 = \{\boldsymbol{x}_{\frac{N}{2}+1}, \ldots, \boldsymbol{x}_N\}$ each with $\frac{N}{2}$ input vectors, and let $Q = 0$.
2. Let $\boldsymbol{v}$ be the first vector in L_1.
3. Find $8d$ vectors (by trying all $8d$-tuples) $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{8d}}$ from L_2 such that $\boldsymbol{c}_{i_1} + \cdots + \boldsymbol{c}_{i_{8d}} - \boldsymbol{v} \in q\,\mathcal{L}$. If no such vectors exist go to step(6).
4. Output the vector $\frac{\boldsymbol{x}_{i_1} + \cdots + \boldsymbol{x}_{i_{8d}} - \boldsymbol{v}}{q} \in \mathcal{L}$, and let $Q = Q + 1$. If $Q = q^{n/d}$, then END.
5. Remove vectors $\boldsymbol{x}_{i_1}, \cdots, \boldsymbol{x}_{i_{8d}}$ from L_2
6. Remove vector $\vec{v}$ from L_1 and repeat Steps (2) to (5).

The time complexity of the algorithm is $\frac{N}{2} \cdot \binom{N/2}{8d} \leq \frac{N}{2}\left(\frac{eN}{16d}\right)^{8d} \leq (10e \cdot d)^{8d} \cdot q^{8n+n/d+o(n)}$, and memory requirement of the algorithm is immediate. We now show correctness. Let $\varepsilon' = \varepsilon^{2d}$ so that $s \geq \sqrt{2}\eta_{\varepsilon'}(q\,\mathcal{L})$ by Lemma 12. Without loss of generality, we can assume that the vectors $\boldsymbol{x}_i$ for $i \in [N]$ are sampled by first sampling $\boldsymbol{c}_i \in \mathcal{L}/q\,\mathcal{L}$ such that $\Pr[\boldsymbol{c}_i = \boldsymbol{c}] = \Pr[D_{\mathcal{L},s} \in q\,\mathcal{L} + \boldsymbol{c}]$ and then sampling the vector $\boldsymbol{x}_i$ according to $D_{q\,\mathcal{L}+\boldsymbol{c}_i,s}$. Moreover, by Corollary 11, this distribution is $2\varepsilon' N$-close to sampling $\boldsymbol{c}_i$ for $i \in [N]$, independently and uniformly from $\mathcal{L}/q\,\mathcal{L}$, and then sampling the vectors $\boldsymbol{x}_i$ according to $D_{q\,\mathcal{L}+\boldsymbol{c}_i,s}$. We now assume that the input is sampled from this distribution.

Without loss of generality, we can assume that the algorithm initially gets only the corresponding cosets as input, and the vectors $\boldsymbol{x}_{i_j} \in q\,\mathcal{L} + \boldsymbol{c}_{i_j}$ for $j \in [8d]$, and $\boldsymbol{v} \in q\,\mathcal{L} + \boldsymbol{c}$ are sampled from $D_{q\,\mathcal{L}+\boldsymbol{c}_{i_j},s}$ and $D_{q\,\mathcal{L}+\boldsymbol{c},s}$ only before such a tuple is needed in Step 4 of the algorithm. Since any input vector is used only once in Step 4, these samples are independent of all prior steps. This implies, by Theorem 13, that the vector obtained in Step 4 of the algorithm is $\varepsilon'(8d+1)$-close to being distributed as $D_{\mathcal{L},s\frac{\sqrt{8d+1}}{q}}$.

It remains to show that our algorithm finds $q^{n/d}$ vectors (with high probability). Let $N' = \frac{N}{2}$ be an integer, X be a random variable uniform over $(\mathcal{L}/q\,\mathcal{L})^{N'}$, and let Y be a random variable independent of X and uniform over vectors in $\{0,1\}^{N'}$ with Hamming weight $8d$. The number of such vectors is

$$\binom{N'}{8d} \geq \left(\frac{N'}{8d}\right)^{8d} \geq q^{8n}\,. \tag{3}$$

Let U be a uniformly random coset of $\mathcal{L}/q\mathcal{L}$. By Lemma 20 and (3), we have

$$\mathrm{d}_{\mathrm{SD}}((\langle X,Y\rangle,X),(U,X)) \le \frac{1}{2}\cdot\sqrt{\frac{q^n}{q^{8n}}} < q^{-7n/2}\,,$$

for a large enough value of n. By Markov inequality, with probability greater than $1-(10\cdot q^{-5n/2})$ over the choice of $x \leftarrow X$, we have that the statistical distance between $\langle x,Y\rangle$ and U is less than $\frac{q^{-n}}{10}$, which implies for any $\boldsymbol{v}\in\mathcal{L}/q\mathcal{L}$,

$$q^{-n}+\frac{q^{-n}}{10} > \Pr[\langle x,Y\rangle = \boldsymbol{v} \mod q\mathcal{L}] > q^{-n}-\frac{q^{-n}}{10}. \tag{4}$$

We assume that the input vectors in list L_2 satisfy (4), introducing a statistical distance of at most $10\cdot q^{-5n/2}$. Notice that after the algorithm found i vectors for any $i<q^{n/d}$, it has removed $8id$ vectors from L_2. We will show that for each vector from L_1 (which is uniformly sampled from $\mathcal{L}/q\mathcal{L}$) with constant probability we will find $8d$-vectors in Step (3).

After $i<q^{n/d}$ output vectors have been found, there are $M=N'-8id$ vectors remaining in the list L_2. There are $\binom{M}{8d}$ different $8d$-combinations possible with vectors remaining in L_2.

$$\begin{aligned}\binom{N'}{8d}\Big/\binom{M}{8d} &= \frac{N'\cdots(N'-8d+1)}{M\cdots(M-8d+1)} < \left(\frac{N'}{N'-8d(i+1)}\right)^{8d} \leqslant \left(1+\frac{8dq^{n/d}}{N'-8dq^{n/d}}\right)^{8d}\\ &= \left(1+\frac{1}{10d-1}\right)^{8d} < \frac{5}{2} \qquad \text{since } N'=80d^2q^{n/d} \text{ for } C=1\end{aligned} \tag{5}$$

At the beginning of the algorithm, there are $\binom{N'}{8d}$ combinations, and hence by (4), each of the q^n cosets appears at least $0.9q^{-n}\binom{N'}{8d}$ times. After $i<q^{n/d}$ output vectors have been found, there are only $\binom{M}{8d}$ combinations left, and $\binom{N'}{8d}-\binom{M}{8d}$ possible combinations have been removed. We say that a coset $\boldsymbol{c}$ *disappears* if there is no set of $8d$ vectors in L_2 that add to $\boldsymbol{c}$. In order for a coset to disappear, all of the at least $0.9q^{-n}\binom{N'}{8d}$ combinations from the initial list must be removed. Hence, the number of cosets that disappear is at most $\frac{\binom{N'}{8d}-\binom{M}{8d}}{0.9q^{-n}\binom{N'}{8d}} < \frac{3/5}{0.9}q^n = \frac{2}{3}q^n$ distinct cosets by (5). Hence with probability at least $1/3$, we find $8d$ vectors $\boldsymbol{x}_{i_1},\ldots,\boldsymbol{x}_{i_{8d}}$ from L_2 such that $\boldsymbol{x}_{i_1}+\cdots+\boldsymbol{x}_{i_{8d}}-\boldsymbol{v}\in q\mathcal{L}$. By Chernoff-Hoeffding bound with probability greater than $1-e^{-d^2q^{n/d}}$, the algorithm finds at least $q^{n/d}$ vectors. In total, the statistical distance from the desired distribution is

$$(8d+1)\varepsilon'\cdot N + 2\varepsilon' q^{n/d} + 10\cdot q^{-5n/2} + e^{-d^2q^{n/d}} \le 10d\varepsilon'\cdot N + 11\cdot q^{-5n/2}. \qquad ◀$$

▶ **Corollary 23.** *Let $n\in\mathbb{N}$, $q\in[4,\sqrt{n}]$ be an integer, and let $\varepsilon = q^{-32n/q^2}$. Let $\mathcal{L}$ be a lattice of rank n, and let $s\ge\eta_\varepsilon(\mathcal{L})$. There is an algorithm that outputs a list of vectors that is $q^{-\Omega(n)}$-close to q^{16n/q^2} independent vectors from $D_{\mathcal{L},s}$. The algorithm runs in time $q^{13n+o(n)}$ and requires memory* $\mathrm{poly}(n)\cdot q^{16n/q^2}$.

Proof. Choose d so that $16d-16<q^2\leqslant 16d$, which is possible when $q\geqslant 4$, and let $\alpha = q/\sqrt{8d+1}$ – this is the ratio by which we decrease the Gaussian width in Theorem 22 – and note that $\alpha\ge 1.2$.

Let $p=\lceil 2\sqrt{d}q\rceil < q^2$ and k be the smallest integer such that $\alpha^k\cdot p\ge 2^{n\log\log n/\log n}$. Thus $k=O(n\log\log n/\log n)$. Let $g=\alpha^kps\ge 2^{n\log\log n/\log n}\cdot\eta_\varepsilon(\mathcal{L})$. By Theorem 17, in time $N_0\cdot\mathrm{poly}(n)$, we get $N_0=(160d^2)^kq^{n/d}$ samples from $D_{\mathcal{L},g}$.

We now iterate k times the algorithm from Theorem 22. Initially we have N_0 vectors. At the beginning of the i-th iteration for $i\le k-1$, we have $N_i := N_0\cdot(160d^2)^{-i}$ vectors that are Δ_i-close to being independently distributed from $D_{\mathcal{L},\alpha^{-i}g}$, where $\alpha^{-i}g\geqslant \alpha p\cdot\eta_\varepsilon(\mathcal{L})$. Hence, we can apply Theorem 22 and get $N_{i+1}=N_i/160d^2$ vectors that are Δ_{i+1}-close to being independently distributed from $D_{\mathcal{L},\alpha^{-(i+1)}g}$, where $\Delta_{i+1} \leqslant \Delta_i + 4\varepsilon^{2d} N_i + 11(160d^2)^{k-i} q^{-5n/2}$. At each iteration we had $N_i \geq 160d^2 q^{n/d}$ vectors, a necessary condition to apply Theorem 22. Therefore after k iterations, we have at least $N_k = N_0/(160d^2)^k = q^{n/d}$ samples that are Δ_k-close to being independently distributed from $D_{\mathcal{L},\alpha^{-k}g}$, where

$$\begin{aligned}
\Delta_k &\leqslant 11q^{-5n/2}\sum_{i=1}^{k}(160d^2)^{k-i} + \sum_{i=0}^{k-1} 10d\varepsilon^{2d}N_i \\
&\leq 11(160d^2)^k q^{-5n/2} + 10dq^{-4n}q^{n/d}\sum_{i=0}^{k-1}(160d^2)^{k-i} && \text{since } 16d \geqslant q^2 \\
&\leq \left(11q^{-5n/2} + 10dq^{-4n+n/d}\right)(160d^2)^{k+1} = q^{-5n/2+o(n)} && \text{since } (160d^2)^{k+1} = q^{o(n)}.
\end{aligned}$$

Any vector distributed as $D_{\mathcal{L},ps}$ is in $p\mathcal{L}$ with probability at least p^{-n}. We repeat the algorithm $2p^n = O(q^{2n})$ times to obtain $p^n \cdot 2 \cdot q^{n/d}$ vectors that are $2p^n q^{-5n/2+o(n)} = q^{-n/2+o(n)}$ close to $2p^n \cdot q^{n/d}$ independent samples from $D_{\mathcal{L},ps}$. Of these samples obtained, we only keep vectors that fall in $p\mathcal{L}$ and divide them by p. Let $M = p^n \cdot 2 \cdot q^{n/d}$. By Chernoff-Hoeffding (Lemma 21) with $P = p^{-n}$, and $\delta = \frac{1}{2}$, the probability to obtain less than $(1-\delta)PM = q^{n/d}$ samples is at most $\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{PM} \leqslant e^{-\frac{1}{10}q^{n/d}}$. Furthermore, $d \leqslant \frac{q^2+16}{16}$ and $q \mapsto \frac{\ln q}{16+q^2}$ is decreasing for $q \geqslant 4$, hence for $q \leqslant \sqrt{n}$,

$$q^{n/d} \geqslant e^{16n\frac{\ln q}{16+q^2}} \geqslant e^{16n\frac{\ln\sqrt{n}}{16+n}} \geqslant e^{16\ln\sqrt{n}-o(1)} = \Omega(n^8).$$

Hence with probability greater than $1 - e^{-\frac{1}{10}q^{n/d}} = 1 - q^{-\Omega(n^8)}$, we get $q^{n/d}$ vectors from the distribution $D_{\mathcal{L},s}$. The statistical distance from the desired distribution is $q^{-\Omega(n^8)} + q^{-n/2+o(n)} \leq q^{-n/2+o(n)}$. We repeat this for $\frac{q^{16n/q^2}}{q^{n/d}}$ times, to get q^{16n/q^2} vectors. The total statistical distance from the desired distribution is $\frac{q^{16n/q^2}}{q^{n/d}} \cdot q^{-n/2+o(n)} \leq q^{-\Omega(n)}$. The total running time is bounded by

$$q^{2n}\left(\frac{q^{16n/q^2}}{q^{n/d}}\right)\left(\text{poly}(n)\cdot N_0 + \sum_{i=0}^{k-1}(10ed)^{8d}\cdot(160d^2)^{k-i}q^{8n+n/d+o(n)}\right) \leqslant q^{13n+o(n)}.$$

The memory usage is slightly more involved: we can think of the k iterations as a pipeline with k intermediate lists and we observe that as soon as a list (at any level) has more than $160d^2q^{16n/q^2}$ elements, we can apply Theorem 22 to produce q^{16n/q^2} vectors at the next level. Hence, we can ensure that at any time, each level contains at most $160d^2q^{16n/q^2}$ vectors, so in total we only need to store at most $k \cdot 160d^2q^{16n/q^2} = \text{poly}(n)q^{16n/q^2}$ vectors, to which we add the memory usage of the algorithm of Theorem 22 which is bounded by $\text{poly}(n)\cdot q^{n/d} \leqslant \text{poly}(n)\cdot q^{16n/q^2}$. Finally, we run the filter $(p\mathcal{L})$ on the fly at the end of the k iterations to avoid storing useless samples. ◀

This tradeoff works for any $q \geq 4$, and the running time can be bounded by $c_1^{n+o(n)} \cdot q^{c_2 n}$ for some constants c_1 and c_2 that we have not tried to optimize.

3.2 Algorithms for BDD and SVP

▶ **Theorem 24.** *Let $n \in \mathbb{N}, q \in [4, \sqrt{n}]$ be a positive integer. Let $\mathcal{L}$ be a lattice of rank n, there exists an algorithm that creates a $0.1/q$-BDD oracle in time $q^{13n+o(n)}$ and space $\text{poly}(n)\cdot q^{16n/q^2}$. Every call to this oracle takes time $\text{poly}(n)q^{16n/q^2}$ time.*

Proof. Let $\varepsilon = q^{\frac{-32n}{q^2}}$ and $s = \eta_\varepsilon(\mathcal{L}^*)$. From corollary 23, there exists an algorithm that outputs q^{16n/q^2} vectors whose distribution is statistically close to time $D_{\mathcal{L}^*,s}$ in $q^{13n+o(n)}$ and space $\mathrm{poly}(n) \cdot q^{16n/q^2}$.

By Theorem 14, there is a reduction from α-BDD to $\frac{1}{2}$-DGS$^m_{\eta_\varepsilon}$ with $m = \mathcal{O}(\frac{n \log(1/\varepsilon)}{\sqrt{\varepsilon}}) = \mathcal{O}(\frac{n^2}{q^2} q^{16n/q^2})$, where the decoding coefficient is $\alpha = \frac{\sqrt{\log(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1(\mathcal{L})}$. By repeating $\mathrm{poly}(n)$ times the algorithm from Corollary 23, we get m vectors from $D_{\mathcal{L}^*,\eta_\varepsilon(\mathcal{L}^*)}$. By Lemma 15, we get

$$\alpha(\mathcal{L}) = \frac{\sqrt{\log(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1(\mathcal{L})} \geq \sqrt{\frac{\log(1/\varepsilon)}{2n(\beta^2/e)\varepsilon^{-1/n}}} \cdot (1 - o(1)) \geq \frac{1}{q}\sqrt{\frac{32 \cdot e \cdot \log q}{2\beta^2 q^{32/q^2}}} \geq (10q)^{-1}.$$

Note that here we are using the fact that the reduction in Theorem 14 always reduces an instance on a lattice $\mathcal{L}$ to an instance on the dual lattice $\mathcal{L}^*$: this is why we generate samples from $D_{\mathcal{L}^*,\eta_\varepsilon(\mathcal{L}^*)}$ in the preprocessing phase, even before any call to the oracle is made. Finally, by Theorem 14, each call to the oracle takes time $m \cdot \mathrm{poly}(n) = \mathcal{O}(q^{16n/q^2}\,\mathrm{poly}(n))$. ◀

▶ **Theorem 25.** *Let $n \in \mathbb{N}, q \in [4, \sqrt{n}]$ be a positive integer. Let $\mathcal{L}$ be a lattice of rank n. There is a randomized algorithm that solves SVP in time $q^{13n+o(n)}$ and in space* $\mathrm{poly}(n) \cdot q^{\frac{16n}{q^2}}$.

Proof. By Theorem 24, we can construct a $\frac{0.1}{q}$-BDD oracle in time $q^{13n+o(n)}$ and in space $\mathrm{poly}(n) \cdot q^{\frac{16n}{q^2}}$. Each execution of the BBD oracle now takes $\mathcal{O}(\mathrm{poly}(n) q^{16n/q^2})$ time. By Theorem 16, with $(10q)^n$ queries to $\frac{0.1}{q}$-BDD oracle, we can find the shortest vector. The total time complexity is $q^{13n+o(n)} + \mathrm{poly}(n) q^{16n/q^2} \cdot (10q)^n = q^{13n+o(n)}$. ◀

▶ Remark 26. If we take $q = \sqrt{n}$, Theorem 25 gives a SVP algorithm that takes $n^{\mathcal{O}(n)}$ time and $\mathrm{poly}(n)$ space. The constant in the exponent of time complexity is worse than the best enumeration algorithms. When q is a large enough constant, for any constant $\varepsilon > 0$, there exists a constant $C = C(\varepsilon) > 2$, such that there is a 2^{Cn} time and $2^{\varepsilon n}$ space algorithm for DGS, and SVP. In particular, the time complexity of the algorithm in this regime is worse than the best sieving algorithms.

4 New space efficient algorithms for SVP

In this section, we present relatively space-efficient classical and quantum algorithms to find a shortest nonzero lattice vector. Our quantum algorithm is the first provable algorithm for exact-SVP that takes less than $\mathcal{O}(2^n)$ time. Recall that there exists an algorithm [15] that, given a lattice $\mathcal{L}$ and a target vector $\boldsymbol{t}$, outputs all lattice vectors within distance $p\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$, by making p^n calls to an α-BDD oracle. We present a quantum algorithm for SVP that takes $2^{0.9532n+o(n)}$ time and $2^{0.5n+o(n)}$ space with $\mathrm{poly}(n)$ qubits. We also present a classical algorithm for SVP that takes $2^{1.741n+o(n)}$ time and $2^{0.5n+o(n)}$ space.

The strategy followed by [15] is to choose $p = \lceil 1/\alpha \rceil$, the target vector $\boldsymbol{t}$ to be the origin, and sequentially compute the candidate vectors for SVP. There are two ways to reduce the time complexity: one can improve the BDD oracle or reduce the number of queries. We will show how to improve both aspects.

4.1 Quantum algorithm for SVP

In order to solve SVP by the method in [15], it is sufficient to use a BDD oracle with decoding coefficient α slightly greater than $1/3$. In [15], the authors use a reduction from BDD to DGS by [16] and use the Gaussian sampler of [2] to obtain many samples with standard deviation equal to $\sqrt{2}\eta_{1/2}$. This allows them to construct a 0.391-BDD but each call to the BDD oracle uses many DGS samples. This is wasteful since we really only need a 1/3-BDD. The reason why it is so expensive is that in the analysis they need to find ε such that $\eta_\varepsilon > \sqrt{2}\eta_{1/2}$ to apply the reduction, and it requires them to take ε much smaller than would be strictly necessary to construct a 1/3-BDD oracle; this smaller ε explains the bigger decoding radius.

We obtain a BDD oracle with decoding distance 1/3 by using the same reduction but making each call cheaper. This is achieved by building a sampler that directly samples at the smoothing parameter, hence avoiding the $\sqrt{2}$ factor, allowing us to take a bigger ε. In [2], it was shown how to construct a dense lattice $\mathcal{L}'$ whose smoothing parameter $\eta(\mathcal{L}')$ is $\sqrt{2}$ times smaller than the original lattice, and that contains all lattice points of the original lattice. Suppose that we first use such a dense lattice to construct a corresponding discrete Gaussian sampler with standard deviation equal to $s = \sqrt{2}\eta(\mathcal{L}')$. We then do the rejection sampling on condition that the output is in the original lattice $\mathcal{L}$. We thus have constructed a discrete Gaussian sampler of $\mathcal{L}$ whose standard deviation is $\sqrt{2}\eta(\mathcal{L}') = \eta(\mathcal{L})$. Nevertheless, $|\,\mathcal{L}' \,/\, \mathcal{L}\,|$ will be at least $2^{0.5n}$, which implies that this procedure needs at least $2^{0.5n}$ input vectors to produce an output vector. We use this idea to obtain the following lemma.

► **Lemma 27.** *There is an probabilistic algorithm that, given a lattice $\mathcal{L} \subset \mathbb{R}^n$, $m \in \mathbb{Z}_+$ and $s \geq \eta_{1/3}(\mathcal{L})$ as input, outputs m samples from a distribution $(m \cdot 2^{-\Omega(n^2)})$-close to $D_{\mathcal{L},s}$ in expected time $m \cdot 2^{(n/2)+o(n)}$ and $(m + 2^{n/2}) \cdot 2^{o(n)}$ space.*

Proof. Let $a = \frac{n}{2} + 4$. We repeat the following until we output m vectors. We use the algorithm in Lemma 19 to obtain a lattice $\mathcal{L}' \supset \mathcal{L}$ of index 2^a. We then run the algorithm from Theorem 18 with input $(\mathcal{L}', s)$ to obtain a list of vectors from $\mathcal{L}'$. We output the vectors in this list that belong to $\mathcal{L}$.

By Theorem 18, we obtain, in time and space $2^{(n/2)+o(n)}$, $M = 2^{n/2}$ vectors that are $2^{-\Omega(n^2)}$-close to M vectors independently sampled from $D_{\mathcal{L}',s}$. Also, by Lemma 19, with probability at least 1/2, we have $s \geq \eta_{1/3}(\mathcal{L}) \geq \sqrt{2}\eta_{1/2}(\mathcal{L}')$.

From these M vectors, we will reject the vectors which are not in lattice $\mathcal{L}$. It is easy to see that the probability that a vector sampled from the distribution $D_{\mathcal{L}',s}$ is in $\mathcal{L}$ is at least $\rho_s(\mathcal{L})/\rho_s(\mathcal{L}') \geq \frac{1}{2^a}$ using Lemma 10. Thus, the probability that we obtain at least one vector from $\mathcal{L}$ (which is distributed as $D_{\mathcal{L},s}$) is at least

$$\frac{1}{2}\left(1 - (1 - 1/2^a)^{2^{n/2}}\right) \geq \frac{1}{2} \cdot \left(1 - e^{-2^{n/2}/2^{n/2+4}}\right) = \frac{1}{2}(1 - e^{-1/16}).$$

It implies that after rejection of vectors, with constant probability we will get at least one vector from $D_{\mathcal{L},s}$. Thus, the expected number of times we need to repeat the algorithm is $O(m)$ until we obtain vectors $\boldsymbol{y_1}, \ldots, \boldsymbol{y_m}$ whose distribution is statistically close to being independently distributed from $D_{\mathcal{L},s}$. The time and space complexity is clear from the algorithm. ◄

► **Theorem 28.** *For any sufficiently large integer n, any integer $m > 0$, and a lattice $\mathcal{L} \subset \mathbb{R}^n$, there exists an algorithm that creates a 1/3-BDD oracle in $2^{0.6608n+o(n)}$ time and $2^{0.5n+o(n)}$ space. Every call to this oracle takes $2^{0.1608n+o(n)}$ time and space.*

Proof. See full version [1, Theorem 30]; it is similar to Theorem 24 but using Lemma 27. ◄

From [15], we can enumerate all vectors of length $p \cdot \frac{1}{3}\lambda_1(\mathcal{L})$ by making p^n calls to 1/3-BDD oracle. Although naively searching for the minimum in the set of vectors of length less than or equal to $p \cdot \frac{1}{3}\lambda_1(\mathcal{L})$, will find the origin with high probability, one can work around this issue by shifting the zero vector. Choosing an arbitrary nonzero lattice vector as the shift, we are guaranteed to obtain a vector of length at least λ_1 for $p \geq 3$. Hence by combining the 1/3-BDD oracle from Theorem 28 and the quantum minimum finding algorithm from [19, Theorem 1], we can find the shortest vector. Note that, we can directly use the quantum speedup construction from [15]. The following theorem is a simplified construction for the quantum algorithm.

▶ **Theorem 29.** *For any $n \geq 5$, there is a quantum algorithm that solves SVP in time $2^{0.9533n+o(n)}$ and classical-space $2^{0.5n+o(n)}$ with polynomial number of qubits.*

Proof. Let $\mathbf{B}$ be a basis of the lattice, $\mathsf{BDD}_{1/3}$ be a 1/3-BDD oracle and let $f : \mathbb{Z}_3^n \to \mathcal{L}$ be $f(\boldsymbol{s}) = -3 \cdot \mathsf{BDD}_{1/3}(\mathcal{L}, (\boldsymbol{Bs})/3) + \boldsymbol{Bs}$. The algorithm works on three quantum registers and our goal is to build a superposition of states of the form $|\boldsymbol{s}\rangle|f(\boldsymbol{s})\rangle|x\rangle$ where $x = \|f(\boldsymbol{s})\|$ most the time (see the definition of U below). The algorithm goes like this, we first use Theorem 28 to construct a quantum oracle O_{BDD} on the first two registers that satisfies $O_{\mathsf{BDD}}|\boldsymbol{s}\rangle|\mathbf{0}\rangle = |\boldsymbol{s}\rangle|f(\boldsymbol{s})\rangle$ for all $\boldsymbol{s} \in \mathbb{Z}_3^n$. We then construct another quantum circuit U satisfying

$$U(|\boldsymbol{\omega}\rangle|0\rangle) = \begin{cases} |\boldsymbol{\omega}\rangle|\,\|\boldsymbol{\omega}\|\rangle & \text{if } \boldsymbol{\omega} \neq \mathbf{0} \\ |\boldsymbol{\omega}\rangle|\,\|\boldsymbol{Be_1}\| + 1\rangle & \text{if } \boldsymbol{\omega} = \mathbf{0}, \end{cases}$$

and apply it on the second and third registers. Here $\boldsymbol{e_1} \in \mathbb{Z}^n$ is a vector whose first coordinate is one and rest are zero. After that, we apply the quantum minimum finding algorithm on the first and third registers and get an index $\boldsymbol{s'}$. The output of the algorithm will be $f_3^{1/3}(\boldsymbol{s'})$.

By Theorem 28, in $2^{0.661n+o(n)}$-time and $2^{0.5n+o(n)}$ space, we can generate $2^{0.161n+o(n)}$ vectors to construct a 1/3-BDD oracle. Thus O_{BDD} can be built using $2^{0.1608n+o(n)}$ Toffoli gates and $\mathrm{poly}(n)$ qubits. To see that we only need $\mathrm{poly}(n)$ qubits, we only keep the vectors of size smaller than $\exp(n)$ in the constuction of O_{BDD}, they thus can all be stored within $\mathrm{poly}(n)$ qubits. Since the vectors are sampled from a Gaussian with width at most $\exp(n)$, the error induced by throwing away the tail of the distribution is negligible. Furthermore all functions acting on these vectors can be implemented with $\mathrm{poly}(n)$ qubits.

We can also construct U efficiently. Hence, the algorithm needs $\mathcal{O}(2^{0.1608n+o(n)})$ Toffoli gates and $\mathrm{poly}(n)$ qubits for three registers. As a result by applying the quantum minimum finding algorithm from [19, Theorem 1], the quantum algorithm takes $3^{0.5n} \cdot 2^{0.1608n+o(n)} = 2^{0.9533n+o(n)}$ time and $2^{0.5n+o(n)}$ classical space with a polynomial number of qubits.

Lastly, we show that the quantum algorithm will output a shortest non-zero vector with constant probability. Since $\|\boldsymbol{Be_1}\| + 1 > \lambda_1(\mathcal{L})$, with at least 1/2 probability one will find the index $\boldsymbol{i}$ such that $f(\boldsymbol{i})$ is a shortest nonzero vector by using the quantum minimum finding algorithm from [19, Theorem 1]. Therefore it suffices to show that there is an index $\boldsymbol{i} \in \mathbb{Z}_3^n$ such that $\|f(\boldsymbol{i})\| = \lambda_1(\mathcal{L})$. By Theorem 16, the list $\{f(\boldsymbol{s})|\boldsymbol{s} \in \mathbb{Z}_3^n\}$ contains all lattice points within radius $3 \cdot \frac{1}{3}\lambda_1(\mathcal{L}) = \lambda_1(\mathcal{L})$ from $\mathbf{0}$, including the lattice vector with length $\lambda_1(\mathcal{L})$. Hence with at least 1/2 probability, the algorithm outputs a non-zero shortest lattice vector. ◀

4.2 Solving SVP by spherical caps on the sphere

We now explain how to reduce the number of queries to the α-BDD oracle. Consider a uniformly random target vector $\boldsymbol{t}$ such that $\alpha(1 - \frac{1}{n})\lambda_1(\mathcal{L}) \leq \|\boldsymbol{t}\| < \alpha\lambda_1(\mathcal{L})$, it satisfies the condition of Theorem 16, *i.e.* $\mathrm{dist}(\mathcal{L}, \boldsymbol{t}) \leq \alpha\lambda_1(\mathcal{L})$. We enumerate all lattice vectors within distance $2\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$ and keep only the shortest nonzero one. We show that for $\alpha = 0.4097$, we will get the shortest nonzero vector of the lattice with probability at least $2^{-0.3298n+o(n)}$. By repeating this $\mathcal{O}(2^{0.3298n+o(n)})$ times, the algorithm will succeed with constant probability. We rely on the following construction of a 0.4097-BDD oracle.

▶ **Theorem 30.** *For any dimension $n \geq 4$, any integer $m > 0$, and a lattice $\mathcal{L} \subset \mathbb{R}^n$, there exists an algorithm that constructs a 0.4097-BDD oracle in $2^{0.9108n+o(n)}$ time and $2^{0.5n+o(n)}$ space. Each call to the oracle takes $2^{0.4108n+o(n)}$ time and space.*

Proof. See full version [1, Theorem 32]; it is similar to Theorem 24 but using Lemma 27. ◀

▶ **Theorem 31.** *There is a randomized algorithm that solves SVP in time* $2^{1.741n+o(n)}$ *and in space* $2^{0.5n+o(n)}$ *with constant probability.*

Proof. On input lattice $\mathcal{L}(\mathbf{B})$, use the LLL algorithm [41] to get a number d (the norm of the first vector of the basis) that satisfies $\lambda_1(\mathcal{L}) \leq d \leq 2^{n/2}\lambda_1(\mathcal{L})$. For $i = 1, \dots, n^2$, let $d_i = d/(1+\frac{1}{n})^i$, and let $\alpha = 0.4097$. There exists a j such that $\lambda_1(\mathcal{L}) \leq d_j \leq (1+\frac{1}{n})\lambda_1(\mathcal{L})$. We repeat the following procedure for all $i = 1, \dots, n^2$:

For $j = 1$ to $2^{0.3298n+o(n)}$, pick a uniformly random vector $\boldsymbol{v_{ij}}$ on the surface of the ball of radius $\alpha(1-\frac{1}{n})d_i$. By Theorem 16, we can enumerate 2^n lattice points using the function $f_{ij} : \mathbb{Z}_2^n \to \mathcal{L}$ defined by $f_{ij}(\boldsymbol{x}) = \mathbf{B}\,\boldsymbol{x} - 2 \cdot \mathsf{BDD}_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{x} - \boldsymbol{v_{ij}})/2)$. At each step we only store the shortest nonzero vector. At the end, we output the shortest among them.

The running time of the algorithm is straightforward. We make 2^n queries to a α-BDD oracle that takes $2^{0.4108n+o(n)}$ time and space by Theorem 30. We further repeat this $n^2 2^{0.3298n+o(n)}$ times. Therefore the algorithm takes $2^{1.741n+o(n)}$ time and $2^{0.5n+o(n)}$ space.

To prove the correctness of the algorithm, it suffices to show that there exists an $i \in [n^2]$ for which the algorithm finds the shortest vector with high probability. Recall that there exists an i such that $\lambda_1(\mathcal{L}) \leq d_i \leq (1+\frac{1}{n})\lambda_1(\mathcal{L})$ and let that index be k. We will show that for a uniformly random vector $\boldsymbol{v}$ of length $\alpha(1-\frac{1}{n})d_k$, if we enumerate 2^n vectors by the function $f : \mathbb{Z}_2^n \to \mathcal{L}$, $f(\boldsymbol{x}) = \mathbf{B}\,\boldsymbol{x} - 2 \cdot \mathrm{BDD}_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{x} - \boldsymbol{v})/2)$, then with probability $2^{-0.3298n-o(n)}$ there exists $\boldsymbol{x} \in \mathbb{Z}_2^n$ such that $f(\boldsymbol{x})$ is the shortest nonzero lattice vector.

We show that we can cover the sphere of radius λ_1 by $2^{0.3298n+o(n)}$ balls of radius $2\alpha\lambda_1 = 0.4097 * 2\lambda_1$ whose centers are at distance $\alpha(1-\frac{1}{n})d_k \leq 0.4097\lambda_1$ from the origin (see figure 1). We have two concentric circles of radius $\alpha(1-\frac{1}{n})d_k$ and λ_1, and let P be a uniformly random point on the surface of the ball of radius $\alpha(1-\frac{1}{n})d_k$. A ball of radius $2\alpha\lambda_1$ at center P will cover the spherical cap with angle ϕ of the ball of radius λ_1. By the law of the cosines, we can compute $\phi \approx \cos^{-1}(\frac{1-3\alpha^2}{2\alpha})$ and hence, by [4, Lemma 5.6], if we randomly choose $\boldsymbol{v}$, the corresponding spherical caps will cover the shortest vector with probability at least $\int_0^\phi \sin^{n-2}\theta d\theta \geq 2^{-0.3298n-o(n)}$. Besides, by Theorem 16, the list $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{Z}_2^n\}$ will contain all lattice points within radius $2\alpha d_k$ from $\boldsymbol{v}$. Hence, the list will contain a shortest vector with probability $2^{-0.3298n+o(n)}$. By repeating this process $2^{0.3298n+o(n)}$ times, we can find the shortest vector with constant probability. ◀

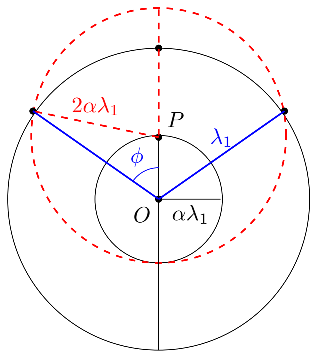

Figure 1 One can cover the sphere of radius λ_1 by balls of radius $2\alpha\lambda_1$, where $\frac{1}{3} \leqslant \alpha < \frac{1}{2}$, whose centers (here P) are at distance $\alpha\lambda_1$ from the origin O. Each such ball covers a spherical cap of half-angle ϕ.

References

1 Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved (provable) algorithms for the shortest vector problem via bounded distance decoding (full version), 2020. arXiv:2002.07955.

2 Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete gaussian sampling: Extended abstract. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 733–742, 2015. doi:10.1145/2746539.2746606.

3 Divesh Aggarwal, Jianwei Li, Phong Q. Nguyen, and Noah Stephens-Davidowitz. Slide reduction, revisited - filling the gaps in SVP approximation. *CoRR*, abs/1908.03724, 2019. arXiv:1908.03724.

4 Divesh Aggarwal and Noah Stephens-Davidowitz. (gap/s) eth hardness of svp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 228–238, 2018.

5 Divesh Aggarwal and Noah Stephens-Davidowitz. Just take the average! an embarrassingly simple 2^n-time algorithm for SVP (and CVP). In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 12:1–12:19, 2018. doi:10.4230/OASIcs.SOSA.2018.12.

6 Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 99–108, 1996. doi:10.1145/237814.237838.

7 Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 601–610, New York, NY, USA, 2001. ACM. doi:10.1145/380752.380857.

8 Martin R Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 717–746. Springer, 2019.

9 Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 405–434, Cham, 2018. Springer International Publishing.

10 Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *IACR Cryptology ePrint Archive*, 2016:713, 2016. URL: http://eprint.iacr.org/2016/713.

11 Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 10–24, 2016. doi:10.1137/1.9781611974331.ch2.

12 Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013.

13 Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 1–12, 2014. doi:10.1145/2554797.2554799.

14 Ernest F. Brickell. Breaking iterated knapsacks. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 342–358, 1984. doi:10.1007/3-540-39568-7_27.

15 Yanlin Chen, Kai-Min Chung, and Ching-Yi Lai. Space-efficient classical and quantum algorithms for the shortest vector problem. *Quantum Information & Computation*, 18(3&4):285–306, 2018. URL: http://www.rintonpress.com/xxqic18/qic-18-34/0285-0306.pdf.

16 Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. On the closest vector problem with a distance guarantee. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 98–109, 2014. doi:10.1109/CCC.2014.18.

17 Rudi de Buda. Some optimal codes have structure. *IEEE Journal on Selected Areas in Communications*, 7(6):893–899, 1989. doi:10.1109/49.29612.

18 Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 125–145. Springer, 2018. doi:10.1007/978-3-319-78381-9_5.

19 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *CoRR*, quant-ph/9607014, 1996. arXiv:quant-ph/9607014.

20 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

21 Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 207–216, 2008. doi:10.1145/1374376.1374408.

22 Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 257–278, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

23 Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009. doi:10.1145/1536414.1536440.

24 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

25 Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 447–464, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

26 Guillaume Hanrot and Damien Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In *Advances in Cryptology – CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 170–186, 2007. doi:10.1007/978-3-540-74143-5_10.

27 Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 469–477, 2007.

28 Bettina Helfrich. Algorithms to construct minkowski reduced and hermite reduced lattice bases. *Theor. Comput. Sci.*, 41(2–3):125–139, December 1985.

29 Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In Serge Fehr, editor, *Public-Key Cryptography – PKC 2017*, pages 16–40, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

30 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

31 Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24, 1989.

32 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

33 Grigorii Anatol'evich Kabatiansky and Vladimir Iosifovich Levenshtein. On bounds for packings on a sphere and in space. *Problemy Peredachi Informatsii*, 14(1):3–25, 1978.

34 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

35 Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, 2005. doi:10.1145/1089023.1089027.

36 Paul Kirchner and Pierre-Alain Fouque. Time-memory trade-off for lattice enumeration in a ball. Cryptology ePrint Archive, Report 2016/222, 2016. URL: `https://eprint.iacr.org/2016/222`.

37 Elena Kirshanova, Erik Mårtensson, Eamonn W Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 521–551. Springer, 2019.

38 Philip Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 937–941, USA, 2000. Society for Industrial and Applied Mathematics.

39 Thijs Laarhoven, Michele Mosca, and Joop Van De Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77, December 2015. `doi:10.1007/s10623-015-0067-5`.

40 J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985. `doi:10.1145/2455.2461`.

41 A.K. Lenstra, H.W. Lenstra, and Lászlo Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

42 Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, page 92, USA, 1998. IEEE Computer Society.

43 Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In *Advances in Cryptology – CRYPTO 2013 – 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 21–39, 2013. `doi:10.1007/978-3-642-40041-4_2`.

44 Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 372–381, 2004. `doi:10.1109/FOCS.2004.72`.

45 Daniele Micciancio and Oded Regev. Lattice-based cryptography, 2008.

46 Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1468–1480, 2010. `doi:10.1137/1.9781611973075.119`.

47 Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. *SIAM J. Comput.*, 42(3):1364–1391, 2013. `doi:10.1137/100811970`.

48 Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 276–294, 2015. `doi:10.1137/1.9781611973730.21`.

49 Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Mathematical Cryptology*, 2(2):181–207, 2008. `doi:10.1515/JMC.2008.009`.

50 Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *IACR Cryptology ePrint Archive*, 2009:605, 2009. URL: `http://eprint.iacr.org/2009/605`.

51 Oded Regev. Lattices in computer science, lecture 8, Fall 2004.

52 Oded Regev. Lattice-based cryptography. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 131–141, 2006. `doi:10.1007/11818175_8`.

53 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, September 2009. `doi:10.1145/1568318.1568324`.

54 Claus Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

55 Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994. doi:10.1007/BF01581144.

56 Adi Shamir. A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. *IEEE Trans. Information Theory*, 30(5):699–704, 1984. doi:10.1109/TIT.1984.1056964.

57 SVP Challenges. https://www.latticechallenge.org/svp-challenge/.