Solving the Shortest Vector Problem in 2^n Time via Discrete Gaussian Sampling

Divesh Aggarwal* Daniel Dadush† Oded Regev‡§
dadush@cwi.nl
Noah Stephens-Davidowitz‡¶
noahsd@cs.nyu.edu

Abstract

We give a randomized $2^{n+o(n)}$-time and space algorithm for solving the Shortest Vector Problem (SVP) on n-dimensional Euclidean lattices. This improves on the previous fastest algorithm: the deterministic $\widetilde{O}(4^n)$-time and $\widetilde{O}(2^n)$-space algorithm of Micciancio and Voulgaris (STOC 2010, SIAM J. Comp. 2013).

In fact, we give a conceptually simple algorithm that solves the (in our opinion, even more interesting) problem of discrete Gaussian sampling (DGS). More specifically, we show how to sample $2^{n/2}$ vectors from the discrete Gaussian distribution at *any parameter* in $2^{n+o(n)}$ time and space. (Prior work only solved DGS for very large parameters.) Our SVP result then follows from a natural reduction from SVP to DGS. We also show that our DGS algorithm implies a $2^{n+o(n)}$-time algorithm that approximates the Closest Vector Problem to within a factor of 1.97.

In addition, we give a more refined algorithm for DGS above the so-called *smoothing parameter* of the lattice, which can generate $2^{n/2}$ discrete Gaussian samples in just $2^{n/2+o(n)}$ time and space. Among other things, this implies a $2^{n/2+o(n)}$-time and space algorithm for 1.93-approximate decision SVP.

Keywords. Discrete Gaussian, Shortest Vector Problem, Lattice Problems.

*Department of Computer Science, EPFL.
†Centrum Wiskunde & Informatica, Amsterdam.
‡Courant Institute of Mathematical Sciences, New York University.
§Supported by the Simons Collaboration on Algorithms and Geometry and by the National Science Foundation (NSF) under Grant No. CCF-1320188. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

¶This material is based upon work supported by the National Science Foundation under Grant No. CCF-1320188. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

A lattice $\mathcal{L}$ is defined as the set of all integer combinations of some linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$. The matrix $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is called a basis of $\mathcal{L}$, and we write $\mathcal{L}(\mathbf{B})$ for the lattice generated by $\mathbf{B}$.

Perhaps the most central computational problem on lattices is the Shortest Vector Problem (SVP). Given a basis for a lattice $\mathcal{L} \subseteq \mathbb{R}^n$, SVP is to compute a non-zero vector in $\mathcal{L}$ of minimum Euclidean norm.

Starting in the '80s, the use of approximate and exact solvers for SVP (and other lattice problems) gained prominence for their applications in algorithmic number theory [LLL82], coding over Gaussian channels [dB89], cryptanalysis [Sha84, Bri85, LO85], combinatorial optimization and integer programming [Len83, Kan87, FT87]. Over the past decade and a half, the study of lattice problems greatly increased due to newly found applications in cryptography. Many powerful cryptographic primitives, such as fully homomorphic encryption [Gen09, BV11, BV14], now have their security based on the *worst-case* hardness of approximating the decision version of SVP (and other lattice problems) to within polynomial factors [Ajt04, MR07, Reg09, BLP+13].

From the computational complexity perspective, much is known about SVP in both its exact and approximate versions. On the hardness side, SVP was shown to be NP-hard to approximate within any constant factor (under randomized reductions) and hard to approximate to within $n^{c/\log\log n}$ for some constant $c > 0$ under reasonable complexity assumptions [Mic01, Kho05, HR12]. From the perspective of polynomial-time algorithms, the celebrated LLL basis reduction gives a $2^{O(n)}$ approximation algorithm for SVP [LLL82], and Schnorr's block reduction algorithm [Sch87], with subsequent refinements [AKS01, MV13], gives a $r^{n/r}$ approximation in $2^{O(r)}\mathrm{poly}(n)$ time allowing for a smooth tradeoff between time and approximation quality.

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 and sometimes also exponential space (and the same is true even for polynomial approximation factors). We mention in passing that despite running in exponential time, these algorithms have practical importance in addition to the obvious theoretical importance. For instance, they are used for assessing the practical security of lattice-based cryptographic primitives, they are used as subroutines in the best current approximation algorithms (variants of block reduction), and they are used in some applications where low-dimensional lattices naturally arise.

While the state of the art for polynomial-time approximation of lattice problems has remained relatively static over the last two decades, the situation for exact algorithms has been markedly different. Indeed, three major (and very different) classes of algorithms for SVP have been developed.

The first class, developed by Kannan [Kan87] and refined by many others [Hel85, HS07, MW15], is based on combining strong basis reduction with exhaustive enumeration inside Euclidean balls. The fastest current algorithm in this class solves SVP in $\widetilde{O}(n^{n/(2e)})$ time while using $\mathrm{poly}(n)$ space [HS07].

The next landmark algorithm, developed by Ajtai, Kumar, and Sivakumar [AKS01] (henceforth AKS), is the most similar to this work. AKS devised a method based on "randomized sieving," whereby exponentially many randomly generated lattice vectors are iteratively combined to create shorter and shorter vectors, to give the first $2^{O(n)}$-time (and space) randomized algorithm for SVP. Many extensions and improvements of their sieving technique have been proposed, both

provable [AKS02, MV10, PS09, LWXZ11] and heuristic [NV08, WLTB11, ZPH14, BGJ14, Laa14], where the fastest provable sieving algorithm [PS09] for exact SVP requires $2^{2.465n+o(n)}$ time and $2^{1.233n+o(n)}$ space. It was observed by [LWXZ11, Mic14, Ste14] that AKS can be modified to obtain a $2^{0.802n+o(n)}$-time and $2^{0.401n+o(n)}$-space algorithm for approximating SVP to within some large constant factor. Here $2^{.401n+o(n)}$ corresponds to the best known upper bound on the n-dimensional "kissing number" (the maximum number of points one can place on the unit sphere such that the pairwise distances are ≥ 1) due to Kabatjanskiĭ and Levenšteĭn [KL78].

The most recent breakthrough, due to Micciancio and Voulgaris [MV13] (henceforth MV) and built upon the approach of Sommer, Feder, and Shalvi [SFS09], is a deterministic $\widetilde{O}(4^n)$-time and $\widetilde{O}(2^n)$-space algorithm for SVP. It uses the *Voronoi cell* of the lattice—the centrally symmetric polytope corresponding to the points closer to the origin than to any other lattice point.

Main contribution. As our main result, we give a randomized $2^{n+o(n)}$-time and space algorithm for exact SVP, improving on the $\widetilde{O}(4^n)$ deterministic running time of MV. A second main result is a much faster $2^{n/2+o(n)}$-time (and space) algorithm that approximates the decision version of SVP to within a small constant factor.

Our $2^{n+o(n)}$-time algorithm actually solves a more difficult problem, namely, that of generating many discrete Gaussian samples from a lattice with arbitrary parameter, as we describe below. We feel that this is even more interesting than the improved running time for SVP, and it should have further applications. As far as we are aware, outside of security reductions having access to powerful oracles, this is the first provable algorithm to use the discrete Gaussian *directly* to solve a classical lattice problem.

Discrete Gaussian samplers. Our first main technical contribution is a general discrete Gaussian sampler, which will directly imply our SVP algorithm. Below, we give an informal description of this result. (See Section 3 for the details.)

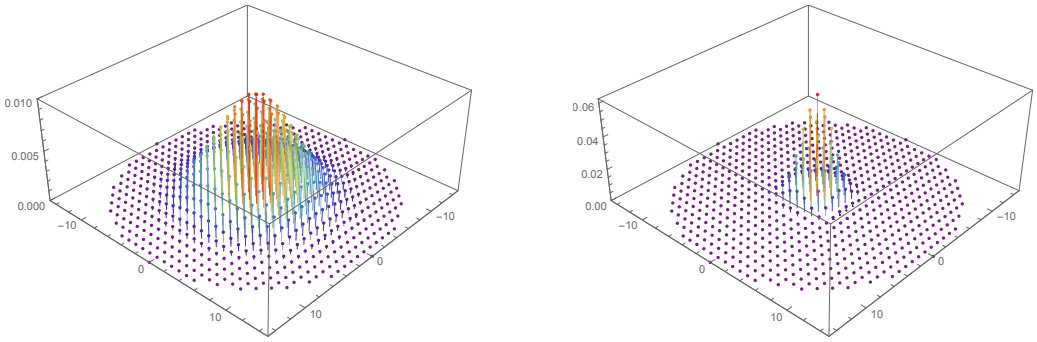

Figure 1: The discrete Gaussian distribution on $\mathbb{Z}^2$ with parameter $s = 10$ (left) and $s = 4$ (right)

Define $\rho_s(\mathbf{x}) = e^{-\pi\|\mathbf{x}\|_2^2/s^2}$ and $\rho_s(A) = \sum_{\mathbf{y}\in A}\rho_s(\mathbf{y})$ for any discrete set $A \subseteq \mathbb{R}^n$. The discrete Gaussian distribution $D_{\mathcal{L},s}$ over the lattice $\mathcal{L} \subseteq \mathbb{R}^n$ with parameter s is the distribution satisfying

$$\Pr_{\mathbf{X}\sim D_{\mathcal{L},s}}[\mathbf{X} = \mathbf{x}] = \rho_s(\mathbf{x})/\rho_s(\mathcal{L}), \quad \forall \mathbf{x} \in \mathcal{L}.$$

See Figure 1 for an illustration. The parameter s determines the "width" of the discrete Gaussian. Note that as s becomes smaller, $D_{\mathcal{L},s}$ becomes more and more concentrated on short lattice vectors. Hence it should not come as a surprise that being able to obtain sufficiently many samples from $D_{\mathcal{L},s}$ for an arbitrary s leads to a solution to SVP. We will discuss this relatively natural reduction below, but first let us describe our main technical contributions, the Gaussian samplers.

Theorem 1.1 (General discrete Gaussian Sampler, informal)**.** *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and* any parameter $s > 0$ *and outputs $2^{n/2}$ i.i.d. samples from $D_{\mathcal{L},s}$ using $2^{n+o(n)}$ time and space.*

Notice the amortized aspect of the algorithm: we obtain $2^{n/2}$ vectors in about 2^n time. We do not know how to reduce the time to $2^{(1-\varepsilon)n}$ —even if all we want is just one vector! (But see below for a faster algorithm that works for large parameters.) Improving the running time of the algorithm (while still outputting a sufficiently large number of samples) would immediately translate into an improved SVP algorithm.

As we explain below, a closer inspection of the technique used in our algorithm suggests that with some refinement it might be able to achieve a running time of $2^{n/2}$. Indeed, we actually do achieve this, but only for sufficiently large parameters s. This is our second main technical contribution.

Theorem 1.2 (Smooth discrete Gaussian sampler, informal)**.** *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a parameter s* above the smoothing parameter *of $\mathcal{L}$ and outputs $2^{n/2}$ i.i.d. samples from $D_{\mathcal{L},s}$ using $2^{n/2+o(n)}$ time and space.*

The smoothing parameter is the value of s above which $D_{\mathcal{L},s}$ "looks like" a continuous Gaussian in a certain precise mathematical sense. (See Definition 2.5.) While sampling above smoothing is apparently not enough to solve exact lattice problems, it is enough to solve major lattice problems approximately. Indeed, we show how this is sufficient to approximate the decision version of SVP to within a constant factor in time $2^{n/2+o(n)}$ (with the constant being roughly 1.93). This holds the record for the fastest provable running time of a hard lattice problem.

1.1 Comparison with prior work

The task of discrete Gaussian sampling is by no means new. It by now has a long history within cryptography [MR07, GPV08, Reg09, Pei09, MP13]. By analyzing an algorithm of Klein [Kle00], Gentry, Peikert, and Vaikuntanathan [GPV08] first showed how to solve DGS in polynomial time for large parameters. (We remark that Klein analyzed this algorithm for very small parameters and used it to solve the BDD problem. For such parameters, the algorithm does not produce samples distributed according to the discrete Gaussian distribution.) DGS has been used extensively to improve reductions from worst-case lattice problems (such as approximate decisional SVP) to the average-case Short Integer Solution (SIS) and Learning with Errors (LWE) problems [MR07, Reg09, Pei09, MP13], and as a core subroutine for instantiating certain cryptographic primitives [GPV08]. In all previous works, the DGS procedure either samples at very high parameters or requires *a priori knowledge of a relatively short lattice basis*—typically only available when a user is able to generate the lattice themselves, such as in certain trapdoor schemes—or access to *powerful oracles*, such as SIS or LWE oracles.

Furthermore, even with oracles and a short basis, none of the algorithms from prior work could be used to sample below the *smoothing parameter* of the lattice. The reason that we are able

to achieve this is because of our observation that, if we allow ourselves exponential time, we can carefully combine vectors sampled from a discrete Gaussian together to obtain vectors whose distribution is *exactly* a discrete Gaussian with a smaller parameter. (See Lemma 3.4 and the proof overview below.) All prior work only obtained a distribution that is *statistically close* to the discrete Gaussian, with error that is unbounded below the smoothing parameter.

We note that our approach is similar to that of the AKS algorithm at a high level. In particular, like AKS, we use a sieve algorithm that starts with a large collection of randomly selected vectors and proceeds to combine them together in pairs to find short lattice vectors. The major important difference between our approach and that of the AKS algorithm and its derivatives is that we maintain complete control over the distribution of the lattice points that we generate at each step. While prior work is focused (quite naturally) on controlling the *lengths* of the vectors after each step, our algorithm actually completely ignores their lengths—choosing whether to combine two vectors based only on their *coset* mod a sublattice.

Indeed, we view our $2^{n+o(n)}$-time algorithm as an efficient discrete Gaussian sampler that consequently yields an efficient solution to SVP, rather than as a sieve algorithm for SVP. It is the simplicity and elegance of the discrete Gaussian distribution that allows us to side-step many of the complications that arise with other sieve algorithms (such as the "perturbation" step). Indeed, the $2^{n+o(n)}$-time algorithm is quite simple; the most technical tool that it uses is a simple subroutine that we call the "square sampler" (described below).

One negative aspect of our approach is that it has a clear lower bound. It seems that we cannot use this approach to find any algorithm that runs in time less than $2^{n/2}$. And, the quoted running time of each algorithm ($2^{n+o(n)}$ and $2^{n/2+o(n)}$ respectively) is essentially tight in both theory and practice—for large (and relevant) parameters, our sieves yield essentially nothing when their input consists of fewer than 2^n or $2^{n/2}$ vectors respectively. This is in contrast to AKS-style algorithms, which seem to perform well heuristically [NV08, WLTB11, ZPH14, Laa14].

1.2 Proof overview

We now include a high-level description of our proofs, first that of Theorem 1.1 and then that of the more refined Theorem 1.2. We end with a brief discussion of how to use Gaussian samples to solve SVP as well as other applications.

A $2^{n+o(n)}$-time combiner for DGS. Recall that efficient algorithms are known for sampling from the discrete Gaussian at very high parameters [GPV08]. It therefore suffices to find a way to efficiently *convert* samples from the discrete Gaussian with a high parameter to samples with a parameter lowered by a constant factor. By repeating this "conversion" many times, we can obtain samples with much lower parameters.

Note that this is trivial to do for the continuous Gaussian: if we divide a vector sampled from the continuous Gaussian distribution by 2, the result is distributed as a continuous Gaussian with half the width. Of course, half of a lattice vector is not typically in the lattice, so this method fails spectacularly when applied to the discrete Gaussian. But, we can try to fix this by *conditioning* on the result staying in the lattice. I.e., we can sample many vectors from $D_{\mathcal{L},s}$, keep those that are in the "doubled lattice" $2\mathcal{L}$, and divide them by two. This method does work, but it is terribly inefficient—there are 2^n cosets of $2\mathcal{L}$, and for some typical parameters, a sample from $D_{\mathcal{L},s}$ will land in $2\mathcal{L}$ with probability as small as 2^{-n}. I.e., our "loss factor," the ratio of the number of

output vectors to the number of input vectors, can be as bad as 2^{-n} for a single step. If we wish to iterate this k times, we could need 2^{kn} input vectors for each output vector, resulting in a very slow algorithm!

We can be much more efficient, however, if we instead look for *pairs* of vectors sampled from $D_{\mathcal{L},s}$ whose *sum* is in $2\mathcal{L}$, or equivalently pairs of vectors that lie in the same coset $\mathbf{c}$ mod $2\mathcal{L}$. Taking our intuition from the continuous Gaussian, we might hope that the *average* of two such vectors will be distributed as $D_{\mathcal{L},s/\sqrt{2}}$. This suggests an *amortized* algorithm, in which we sample many vectors from $D_{\mathcal{L},s}$, place them in "buckets" according to their coset mod $2\mathcal{L}$, and then take the average of disjoint pairs of elements in the same bucket. We call such an algorithm a "combiner." The most natural combiner to consider is the "greedy combiner," which simply pairs as many vectors in each bucket as it can, leaving at most one unpaired vector per bucket. Since there are 2^n cosets, if we take, say, $\Omega(2^n)$ samples from $D_{\mathcal{L},s}$, almost all of the resulting vectors will be paired. A lemma due to Peikert ([Pei10]) shows that the resulting distribution will be statistically close to the desired distribution, $D_{\mathcal{L},s/\sqrt{2}}$, *provided that the parameter s is above the smoothing parameter*.

At this point, we can already build a roughly 2^n-time algorithm for DGS that works for such parameters. (Namely, use prior work to sample at some very high parameter and iteratively apply the combiner described above.) While this is not our main result (it is strictly weaker), we note that we have not seen this observation mentioned elsewhere.[1] But, in order to move *below smoothing* (which is necessary, e.g., for solving SVP), we need to do something else.

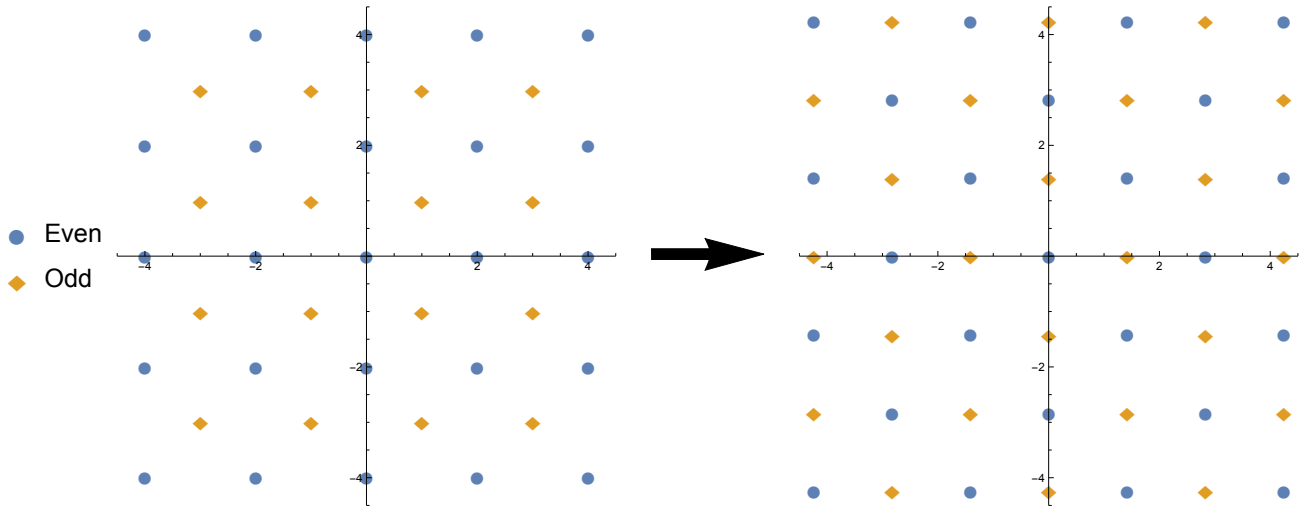

Figure 2: The rotation mapping $\overline{\mathcal{L}} = \{(z_1, z_2) \in \mathbb{Z}^2 : z_1 = z_2 \bmod 2\}$ to $(\sqrt{2}\mathbb{Z})^2$, $(z_1, z_2) \mapsto (z_1 + z_2, z_1 - z_2)/\sqrt{2}$. Our algorithm effectively creates samples from $D_{\overline{\mathcal{L}},s}$ and then outputs the first coordinate of the rotated result scaled down by a factor of $\sqrt{2}$, $(z_1 + z_2)/2$. The resulting distribution is exactly $D_{\mathbb{Z},s/\sqrt{2}}$.

[1] One can likely also obtain a $2^{O(n)}$-time algorithm for DGS above the smoothing parameter by instantiating the oracles in [MP13].

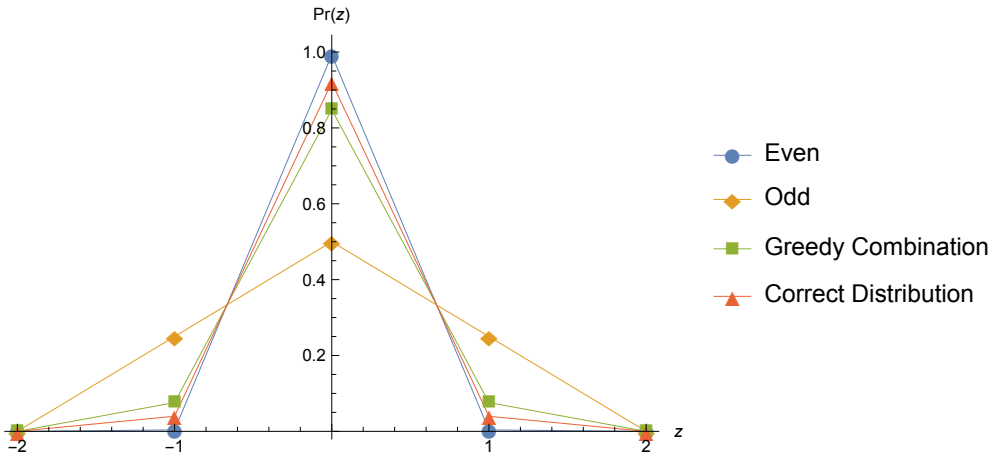

Figure 3: The distribution of averages of pairs of integers sampled from $D_{\mathbb{Z},\sqrt{2}}$ resulting from taking (1) only even pairs; (2) only odd pairs; (3) even and odd pairs with "greedy" weights proportional to $\rho_{\sqrt{2}}(2\mathbb{Z})$ and $\rho_{\sqrt{2}}(2\mathbb{Z}+1)$ respectively; and (4) even and odd pairs with "squared" weights proportional to $\rho_{\sqrt{2}}(2\mathbb{Z})^2$ and $\rho_{\sqrt{2}}(2\mathbb{Z}+1)^2$ respectively. The fourth distribution is exactly $D_{\mathbb{Z}}$.

In particular, below the smoothing parameter, combining discrete Gaussian vectors "greedily" as above will not typically give a result that is statistically close to a Gaussian distribution. However, all is not lost. Recall that our algorithm works by picking pairs of vectors sampled independently from $D_{\mathcal{L},s}$ that are in the same coset $\mathbf{c}$ mod $2\mathcal{L}$, and then taking the average of each pair. So, the algorithm effectively samples a vector $(\mathbf{X}_1, \mathbf{X}_2)$ from *some* distribution over the $2n$-dimensional lattice of pairs of vectors from $\mathcal{L}$ that are in the same coset mod $2\mathcal{L}$,

$$\overline{\mathcal{L}} := \{(\mathbf{X}_1, \mathbf{X}_2) \in \mathcal{L}^2 \ : \ \mathbf{X}_1 = \mathbf{X}_2 \bmod 2\mathcal{L}\} = \bigcup_{\mathbf{c} \in \mathcal{L}/(2\mathcal{L})} \mathbf{c} \times \mathbf{c}\,,$$

and then outputs $(\mathbf{X}_1 + \mathbf{X}_2)/2$. We claim that *assuming that that distribution is* $D_{\overline{\mathcal{L}},s}$, the output $(\mathbf{X}_1 + \mathbf{X}_2)/2$ is distributed *exactly* as $D_{\mathcal{L},s/\sqrt{2}}$. This fact, shown in Lemma 3.4, has a straightforward proof, yet we have not seen this observation before. (It is closely related to Riemann's theta relations, as described in [Mum07, Chapter 1, Section 5]; see also [RS15]) The idea is the following. It is not difficult to show that if we apply the 45° rotation given by

$$(\mathbf{X}_1, \mathbf{X}_2) \mapsto \left(\frac{\mathbf{X}_1 + \mathbf{X}_2}{\sqrt{2}}, \frac{\mathbf{X}_1 - \mathbf{X}_2}{\sqrt{2}}\right)$$

to $\overline{\mathcal{L}}$, we obtain the *product* lattice $(\sqrt{2}\mathcal{L})^2 = \{(\sqrt{2}\mathbf{X}_1, \sqrt{2}\mathbf{X}_2) \mid \mathbf{X}_1, \mathbf{X}_2 \in \mathcal{L}\}$. (Figure 2 shows the one-dimensional case.) Note that the rotation of a discrete Gaussian is again a discrete Gaussian, and the discrete Gaussian over a product lattice is a product distribution. Therefore, if $(\mathbf{X}_1, \mathbf{X}_2)$ is distributed according to $D_{\overline{\mathcal{L}},s}$, the distribution of $(\mathbf{X}_1 + \mathbf{X}_2, \mathbf{X}_1 - \mathbf{X}_2)/\sqrt{2}$ is according to $D_{(\sqrt{2}\mathcal{L})^2,s}$ which is a product distribution, and hence $(\mathbf{X}_1 + \mathbf{X}_2)/\sqrt{2}$ is distributed according to $D_{\sqrt{2}\mathcal{L},s}$, as claimed.

However, note that if the combiner just greedily paired as many vectors from each coset as possible, it would *not* yield samples from $D_{\overline{\mathcal{L}},s}$. In particular, the probability that a sample from $D_{\overline{\mathcal{L}},s}$ will land in $\mathbf{c} \times \mathbf{c}$ for some coset $\mathbf{c}$ is proportional to the "squared weight" of the coset $\rho_s(\mathbf{c})^2$. But, the greedy approach pairs vectors from $\mathbf{c}$ with probability roughly proportional to $\rho_s(\mathbf{c})$. (Figure 3 shows how the resulting distributions differ in the one-dimensional case.) For parameters above smoothing, these distributions are roughly the same, but to go below smoothing (and to avoid the statistical error resulting from the greedy approach), we need a way to sample pairs from this "squared distribution" directly.

This mismatch between the "squared distribution" that we want and the "unsquared" distribution that we get is the primary technical challenge that we must overcome to build our general discrete Gaussian combiner. To solve it, we present a generic solution for "converting any probability distribution to its square" relatively efficiently, which we call the "square sampler." Informally, the square sampler is given access to samples from some probability distribution that assigns respective (unknown) probabilities $(p_1, \ldots, p_N)$ to the elements in some (large) finite set $\{1, \ldots, N\}$. It uses this to efficiently sample a large collection of *independent* coin flips $b_{i,j}$ such that $b_{i,j} = 1$ with probability proportional to p_i. Then, using these coins, it applies rejection sampling to the input samples (accepting the jth instance of input value i if $b_{i,j} = 1$) in order to obtain the desired "squared distribution." If $\Pr[b_{i,j} = 1] = Tp_i$ for some proportionality factor T, it is not hard to see that the expected "loss factor" of this process is $T \sum p_i^2$. We therefore take T to be as large as possible by setting $T \approx 1/\max p_i$ (if we took T to be any larger, we would need a coin that lands on heads with probability greater than one!), making the loss factor of the square sampler approximately $\sum p_i^2 / \max p_i$. (See Section 3.1 and Theorem 3.3 in particular.)

In particular, when combining discrete Gaussian vectors, the loss factor is approximately the *collision probability* over the cosets $\mathbf{c}$ of $2\mathcal{L}$, $\sum \rho_s(\mathbf{c})^2 / \rho_s(\mathcal{L})^2$, divided by the maximal probability of a single coset. As a result, if one coset has a $2^{-n/2}$ fraction of the total weight and the other cosets split the remaining weight roughly evenly, then the loss factor is roughly $2^{-n/2}$ *for a single step of the combiner*. This looks terrible for us, as it could be the case that k applications of the combiner could yield a loss factor of $2^{-kn/2}$! Surprisingly, we show that the product of all loss factors for an arbitrarily long sequence of applications of the combiner is at worst $2^{-n/2}$ (ignoring loss due to other factors). I.e., the accumulated loss factor can be no worse than essentially the worst-case loss factor in a single step![2] As a result, our general combiner always returns $2^{n/2}$ vectors when its input is $2^{n+o(n)}$ vectors sampled from the discrete Gaussian. (See Corollary 3.6 for the formal analysis of repeated application of our combiner.)

A $2^{n/2+o(n)}$-time combiner for DGS above smoothing. Recall that the general combiner described above starts with many vectors and then repeatedly takes the average of pairs of vectors that lie in the same coset of $2\mathcal{L}$. We observed that this combiner necessarily needs over 2^n vectors "just to get started" because it works over the 2^n cosets of $2\mathcal{L}$. To get a faster combiner, we therefore try pairing vectors according to the cosets of some sublattice $\mathcal{L}'$ that "lies between" $\mathcal{L}$ and $2\mathcal{L}$ such that $2\mathcal{L} \subseteq \mathcal{L}' \subset \mathcal{L}$. If we simply take many samples from $D_{\mathcal{L},s}$, group them according to their cosets mod $\mathcal{L}'$, and sum them together (taking averages is a bit less natural in this context), analogy with the continuous Gaussian suggests that the resulting vectors will be distributed as roughly $D_{\mathcal{L}',\sqrt{2}s}$. Note that the parameter has increased, which is not what we wanted, but we are

[2] While the purely algebraic proof of this fact is quite simple (see the proof of Corollary 3.6), we do not yet have good intuitive understanding of it. Indeed, we have found ourselves referring to it as the "magic cancellation."

now sampling from a sparser lattice. In particular, suppose that we apply this combiner twice, so that in the second step we obtain vectors from some sublattice $\mathcal{L}''$. We then expect to obtain samples from roughly $D_{\mathcal{L}'',2s}$. So, intuitively, if we take $\mathcal{L}''$ to be a sublattice of $2\mathcal{L}$, we have "made progress," even though we have doubled the parameter. Our running time will be proportional to the index of $\mathcal{L}'$ over $\mathcal{L}$ (assuming that the index of $\mathcal{L}''$ over $\mathcal{L}'$ is the same), so we should take the index of $\mathcal{L}'$ over $\mathcal{L}$ to be as small as possible. More specifically, we can build a "tower" of progressively sparser lattices $(\mathcal{L}_0, \ldots, \mathcal{L}_\ell)$ with the index of $\mathcal{L}_i$ over $\mathcal{L}_{i-1}$ taken to be slightly larger than $2^{n/2}$.[3] If we take $\mathcal{L}_\ell$ to be the lattice from which we wish to obtain samples with parameter s and $\mathcal{L}_0$ to be a dense lattice from which we can sample efficiently with parameter $2^{-\ell/2}s$, we can hope that iteratively applying such a combiner "up the tower" will yield a sampling algorithm.

As in the description of our 2^n-time combiner, the lemma from [Pei10] shows that the above approach, when instantiated with the "greedy combiner," will yield an algorithm that can output vectors whose distribution is statistically close to the discrete Gaussian for parameters s that are above the smoothing parameter. Though this statistical distance can be made small, it is large enough to break applications such as our approximation algorithm for decision SVP.

To avoid this error, the natural hope is that the same combiner used in the 2^n-time algorithm above (the one with the "square sampler") will suffice. Unfortunately, this gives the wrong distribution. In particular, we obtain a distribution in which the cosets of $\mathcal{L}'$ over $2\mathcal{L}$ have weight that is proportional to the *square* of their weights over the discrete Gaussian. (See Lemma 5.6. Note that when $\mathcal{L}' = 2\mathcal{L}$ there is only one such coset, which is why our 2^n-time combiner does not run into this problem.) In some sense, this is the "inverse" of the problem that the square sampler solves. And, indeed, we solve it by building a "square-root sampler"—based on a clever trick (used implicitly in [MP05] and discussed in [Did12]) that allows one to flip a coin with probability $\sqrt{p}$ given black-box access to a coin with unknown probability p. (See Claim 5.2 for the trick and Theorem 5.4 for the square-root sampler.) So, our combiner works by "squaring the weights" of the cosets mod $\mathcal{L}'$ of input vectors; pairing them according to these squared weights and summing the pairs; and then "taking the square root of the weights" of the cosets mod $2\mathcal{L}$ of the resulting output vectors.

This completes the description of the proof of Theorem 1.2. The above only works above the smoothing parameter because the required size of the input to the square-root sampler depends on $1/p_{\mathsf{min}}$, where p_{min} is the probability of landing in the coset with minimal weight. We therefore only know how to use the square-root sampler to efficiently sample above the smoothing parameter, where the minimal weight is roughly equal to the maximal weight. Indeed, in this regime, both the square-root sampler and the square sampler incur "almost no loss," so that we obtain an algorithm that runs in time $2^{n/2+o(n)}$ and returns $2^{n/2}$ samples from the discrete Gaussian. But, below this, $1/p_{\mathsf{min}}$ can be arbitrarily large. (Intuitively, some sort of dependence on $1/p_{\mathsf{min}}$ is necessary for a square-root sampler because a coset whose weight is negligible could have significant weight after we "take the square root." So, we should expect that any square-root sampler would need at least enough samples to "see" such a coset.)

However, we again stress that our techniques do not incur error that depends on "how smooth the distribution is." This leaves open the possibility that our algorithm might be modified to work even *below* smoothing. The only bottleneck is that the square-root sampler requires very large input in such cases. But, we note that the way that we currently use the square-root sampler might not be optimal. More specifically, we observe the rather strange behavior of our current algo-

[3] We note that Becker et al. [BGJ14] also use a tower of lattices in their heuristic algorithm.

rithm: when the algorithm "takes the square root" of some coset weights, it typically "squares" the weights of some (different!) cosets immediately afterwards. So, while the second step is not the exact inverse of the first, it does still seem that the square-root step is a bit counterproductive. This suggests that there is room for improvement in this algorithm, and we have made some progress to that end by proving a correlation inequality that we believe should play a central role in such an improved algorithm [RS15].

Reduction from SVP to DGS. As mentioned above, if we could efficiently sample from $D_{\mathcal{L},s}$ at the *right parameter*, we can hope that a $D_{\mathcal{L},s}$ sample will hit a shortest non-zero vector of $\mathcal{L}$ with reasonable probability. One can quickly see here that there is an important trade-off in the choice of s. For s too small, the discrete Gaussian becomes completely concentrated on $\mathbf{0}$, whereas for s too large, the distribution becomes too diffuse over $\mathcal{L}$ and will rarely hit a shortest vector. By properly choosing s, we show that the discrete Gaussian yields a shortest non-zero lattice vector with probability $2^{-0.465n-o(n)}$. (In Section 4.1, we note that the optimal parameter has a nice interpretation in terms of the smoothing parameter.) Since our DGS algorithm returns $2^{n/2}$ vectors in time $2^{n+o(n)}$, we obtain a $2^{n+o(n)}$-time algorithm for SVP.

In order to obtain this bound, we use the result of Kabatjanskiĭ and Levenšteĭn that achieves the current best upper bound on the kissing number [KL78]. (The kissing number bounds from above the maximal number of shortest non-zero vectors in a lattice. Note that the reciprocal of the latter is a natural upper bound on the above probability.) At a high level, this is essentially the same problem faced by the randomized sieving algorithms, and our techniques are very similar to those developed there (in particular those in [PS09, MV10]).

Reduction from decision SVP to DGS above smoothing. In order to approximate the length of the shortest non-zero lattice vector to within a constant factor, we note (in Lemma 6.1) that it suffices to approximate the smoothing parameter of the dual lattice (for exponentially small ε) to within a constant factor. Of course, if we had a $2^{n/2}$-time discrete Gaussian sampler that worked above smoothing and always failed below smoothing, then it would be trivial to use this to approximate the smoothing parameter. However, while our $2^{n/2+o(n)}$-time sampler does in fact always work above smoothing, it is not a priori clear how it behaves when asked to provide samples below smoothing.

We handle this problem as follows. First, while we cannot guarantee that our sampler always fails below smoothing, we show (with a bit more work) that it always either fails or outputs valid discrete Gaussian samples. We call such a sampler "honest." (See Definition 5.1.) Second, we show a simple test that can distinguish between the discrete Gaussian distribution with parameter slightly above smoothing and the discrete Gaussian with parameter below smoothing. (See Lemma 6.3.) With this, we obtain an $O(1)$-approximation algorithm for the smoothing parameter that runs in $2^{n/2+o(n)}$ time.

Further applications. Another fundamental problem on lattices is the Closest Vector Problem (CVP), in which we must find a closest lattice vector to some target vector $\mathbf{t}$. CVP is known to be at least as hard as SVP, as there is a polynomial-time approximation-preserving reduction from SVP to CVP [GMSS99]. Furthermore, almost all of the major lattice problems reduce to CVP in this way [Mic08].

The fastest exact algorithm for CVP is again the $\widetilde{O}(4^n)$-time and $\widetilde{O}(2^n)$-space algorithm due to Micciancio and Voulgaris [MV13] (which in fact more directly solves CVP than SVP). For approximation factor $\gamma = 1 + r$ for $r > 0$, randomized sieving techniques have been shown capable of solving γ-approximate CVP in $2^{O(n)}(1 + 1/r)^{O(n)}$ time and space [AKS02, BN09], though little effort has been made to optimize the constant in the exponent.

Based on an embedding trick of Kannan [Kan87] and standard concentration bounds on the discrete Gaussian, we show how to use our sampler to solve 1.97-approximate CVP in time $2^{n+o(n)}$. As mentioned above, the reductions of [Mic08] show that this yields the same approximation factor and running time for almost all lattice problems.

Also, our algorithm from Theorem 1.2 gives $2^{n/2+o(n)}$-time algorithms for .422-BDD (Corollary 7.4) and $O(\sqrt{n \log n})$-approximate SIVP (Corollary 7.6).

1.3 Conclusions and open problems

Our work raises many questions and potential avenues for improvement. Firstly, we suspect that the algorithm from Theorem 1.2 can be modified to work for an arbitrary parameter s with the same running time of roughly $2^{n/2}$ (at least to sample a single vector). Such a result would subsume Theorem 1.1 and would lead to an improved algorithm for SVP, as well as other problems. We have made some modest progress towards proving this, but a solution still seems far.

Another central open problem is whether SVP can be solved in singly exponential time but only polynomial space. The best running time known for polynomial-space algorithms is the $n^{O(n)}$ obtained by enumeration-based methods [Kan87, Hel85, HS07, MW15].

Finally, this work shows that Discrete Gaussian Sampling is a lattice problem of central importance. However, DGS for parameters below smoothing is not nearly as well-understood as many other lattice problems, and many natural questions remain open. For example, is there a dimension-preserving reduction from DGS to CVP? Is DGS NP-hard?

Follow-up work. In a follow-up work by three of us [ADS15] we generalize Theorem 1.1 by presenting a $2^{n+o(n)}$-time algorithm to sample from the *shifted* discrete Gaussian $D_{\mathcal{L}-\mathbf{t},s}$ for any $\mathbf{t} \in \mathbb{R}^n$ and $s > 0$. As an application of that algorithm, we show in [ADS15] how to obtain a $2^{n+o(n)}$-time algorithm for *exact* CVP (which is a harder problem than SVP, as follows from the dimension-preserving reduction in [GMSS99]). While those results are stronger than some of the results presented in this paper, the proofs in [ADS15] are also significantly more involved.

Organization. In Section 2, we overview the necessary background material and give the basic definitions used throughout the paper. In Section 3, we give our general $2^{n+o(n)}$-time DGS sampler (Theorem 3.7). In Section 4, we prove our bound on the number of discrete Gaussian samples needed for SVP (Lemma 4.2 and Proposition 4.3) and give our reduction from SVP to DGS (Theorem 4.4). In Section 5, we give our $2^{n/2+o(n)}$-time DGS sampler for parameters above smoothing (Theorem 5.11). In Section 6, we show our reduction from GapSVP to DGS above smoothing (Theorem 6.5). Finally, in Section 7, we show our $2^{n+o(n)}$-time algorithm for 1.97-approximate CVP (Theorem 7.1) and our $2^{n/2+o(n)}$-time algorithms for .422-BDD (Corollary 7.4) and $O(\sqrt{n \log n})$-approximate SIVP (Corollary 7.6).

2 Preliminaries

Let $\mathbb{N} = \{0, 1, \dots, \}$. Except where we specify otherwise, we use C, C_1, and C_2 to denote universal positive constants, which might differ from one occurrence to the next. We use bold letters $\mathbf{x}$ for vectors and denote a vector's coordinates with indices x_i. Throughout the paper, n will always be the dimension of the ambient space $\mathbb{R}^n$.

2.1 Lattices

A rank d lattice $\mathcal{L} \subset \mathbb{R}^n$ is the set of all integer linear combinations of d linearly independent vectors $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$. $\mathbf{B}$ is called a basis of the lattice and is not unique. Formally, a lattice is represented by a basis $\mathbf{B}$ for computational purposes, though for simplicity we often do not make this explicit. If $n = d$, we say that the lattice has full rank, and we often assume this as results for full-rank lattices naturally imply results for arbitrary lattices.

Given a basis, $(\mathbf{b}_1, \dots, \mathbf{b}_d)$, we write $\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_d)$ to denote the lattice with basis $(\mathbf{b}_1, \dots, \mathbf{b}_d)$. The length of a shortest non-zero vector in the lattice is written $\lambda_1(\mathcal{L})$. For a vector $\mathbf{t} \in \mathbb{R}^n$, we write $\mathrm{dist}(\mathbf{t}, \mathcal{L})$ to denote the distance between $\mathbf{t}$ and the lattice, $\min_{\mathbf{y} \in \mathcal{L}}(\|\mathbf{t} - \mathbf{y}\|)$.

For a lattice $\mathcal{L} \subset \mathbb{R}^n$, the dual lattice, denoted $\mathcal{L}^*$, is defined as the set of all points in $\mathrm{span}(\mathcal{L})$ that have integer inner products with all lattice points,

$$\mathcal{L}^* = \{\mathbf{w} \in \mathrm{span}(\mathcal{L}) : \forall \mathbf{y} \in \mathcal{L}, \langle \mathbf{w}, \mathbf{y} \rangle \in \mathbb{Z}\} .$$

Similarly, for a lattice basis $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$, we define the dual basis $\mathbf{B}^* = (\mathbf{b}_1^*, \dots, \mathbf{b}_d^*)$ to be the unique set of vectors in $\mathrm{span}(\mathcal{L})$ satisfying $\langle \mathbf{b}_i^*, \mathbf{b}_j \rangle = \delta_{i,j}$. It is easy to show that $\mathcal{L}^*$ is itself a rank d lattice and $\mathbf{B}^*$ is a basis of $\mathcal{L}^*$.

Definition 2.1. *For a lattice $\mathcal{L}$, the ith successive minimum of $\mathcal{L}$ is*

$$\lambda_i(\mathcal{L}) = \inf\{r : \dim(\mathrm{span}(\mathcal{L} \cap B(\mathbf{0}, r))) \geq i\} .$$

In other words, the ith successive minimum of $\mathcal{L}$ is the smallest value r such that there are i linearly independent vectors in $\mathcal{L}$ of length at most r.

2.2 The discrete Gaussian distribution

For any $s > 0$, we define the function $\rho_s : \mathbb{R}^n \to \mathbb{R}$ as $\rho_s(\mathbf{t}) = \exp(-\pi \|\mathbf{t}\|^2 / s^2)$. When $s = 1$, we simply write $\rho(\mathbf{t})$. For a discrete set $A \subset \mathbb{R}^n$ we define $\rho_s(A) = \sum_{\mathbf{x} \in A} \rho_s(\mathbf{x})$.

Definition 2.2. *For a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a vector $\mathbf{t} \in \mathbb{R}^n$, let $D_{\mathcal{L}+\mathbf{t},s}$ be the probability distribution over $\mathcal{L} + \mathbf{t}$ such that the probability of drawing $\mathbf{x} \in \mathcal{L} + \mathbf{t}$ is proportional to $\rho_s(\mathbf{x})$. We call this the discrete Gaussian distribution over $\mathcal{L} + \mathbf{t}$ with parameter s.*

We make frequent use of the discrete Gaussian over the cosets of a sublattice. If $\mathcal{L}' \subseteq \mathcal{L}$ is a sublattice of $\mathcal{L}$, then the set of cosets, $\mathcal{L}/\mathcal{L}'$ is the set of translations of $\mathcal{L}'$ by lattice vectors, $\mathbf{c} = \mathcal{L}' + \mathbf{y}$ for some $\mathbf{y} \in \mathcal{L}$. It is easily seen from the Poisson summation formula that for any $\mathbf{c} \in \mathcal{L}/\mathcal{L}'$, $\rho_s(\mathcal{L}') \geq \rho_s(\mathbf{c})$, i.e., the zero coset has maximal weight (see, e.g., [Ban93]). We use this fact throughout the paper. In particular, it follows that $\rho_s(\mathcal{L})/\rho_s(\mathcal{L}') \leq |\mathcal{L}/\mathcal{L}'|$.

Banaszczyk proved the following two bounds on the discrete Gaussian [Ban93].

Lemma 2.3 ([Ban93, Lemma 1.4]). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and $s > 1$,*

$$\rho_s(\mathcal{L}) \leq s^n \rho(\mathcal{L}) \ .$$

Lemma 2.4 ([DRS14, Lemma 2.13]). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, $s > 0$, $\mathbf{u} \in \mathbb{R}^n$, and $t \geq 1/\sqrt{2\pi}$,*

$$\Pr_{\mathbf{X} \sim D_{\mathcal{L}+\mathbf{u},s}} [\|\mathbf{X}\| > ts\sqrt{n}] < \frac{\rho_s(\mathcal{L})}{\rho_s(\mathcal{L}+\mathbf{u})} \left(\sqrt{2\pi e t^2} \exp(-\pi t^2)\right)^n \ .$$

Definition 2.5. *For a lattice $\mathcal{L} \subset \mathbb{R}^n$ and $\varepsilon > 0$, we define the smoothing parameter $\eta_\varepsilon(\mathcal{L})$ as the unique value satisfying $\rho_{1/\eta_\varepsilon(\mathcal{L})}(\mathcal{L}^* \setminus \{\mathbf{0}\}) = \varepsilon$.*

We note that if $\mathcal{L}' \subseteq \mathcal{L}$, then $\eta_\varepsilon(\mathcal{L}) \leq \eta_\varepsilon(\mathcal{L}')$, and we have $\eta_\varepsilon(s\mathcal{L}) = s\eta_\varepsilon(\mathcal{L})$. The name smoothing parameter comes from the following fact.

Claim 2.6. *For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and $\varepsilon \in (0,1)$, if $s \geq \eta_\varepsilon(\mathcal{L})$, then for all $\mathbf{t} \in \mathbb{R}^n$,*

$$\frac{\rho_s(\mathcal{L}+\mathbf{t})}{\rho_s(\mathcal{L})} \geq \frac{1-\varepsilon}{1+\varepsilon} \ .$$

Finally, we will need the following basic bounds on the smoothing parameter, the first of which is essentially the same as [CDLP13, Lemma 2.4].

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

Proof. Suppose without loss of generality that $\eta_\varepsilon(\mathcal{L}) = 1$. Then,

$$\rho_{1/k}(\mathcal{L}^* \setminus \{\mathbf{0}\}) = \sum_{\mathbf{y} \in \mathcal{L}^* \setminus \{\mathbf{0}\}} \rho(\mathcal{L})^{k^2} < \Big(\sum_{\mathbf{y} \in \mathcal{L}^* \setminus \{\mathbf{0}\}} \rho(\mathcal{L}) \Big)^{k^2} = \varepsilon^{k^2} \ . \qquad \square$$

Lemma 2.8. *For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and $\varepsilon = 0.99$,*

$$\eta_\varepsilon(\mathcal{L}) \geq C \frac{\lambda_n(\mathcal{L})}{\sqrt{n}} \ .$$

Proof. Suppose $\lambda_n(\mathcal{L}) > 500\sqrt{n}\eta_\varepsilon(\mathcal{L})$. Then there exists a $\mathbf{u} \in \mathbb{R}^n$ such that $\mathrm{dist}(\mathbf{u}, \mathcal{L}) > 250\sqrt{n}\eta_\varepsilon(\mathcal{L})$. Then, using Lemma 2.4,

$$\rho_{\eta_\varepsilon(\mathcal{L})}(\mathcal{L}+\mathbf{u}) = \rho_{\eta_\varepsilon(\mathcal{L})}((\mathcal{L}+\mathbf{u}) \setminus B(\mathbf{0}, 250\sqrt{n}\eta_\varepsilon(\mathcal{L}))) \leq 200^{-n} \rho_{\eta_\varepsilon(\mathcal{L})}(\mathcal{L}).$$

Using Claim 2.6, this gives

$$\eta_\varepsilon(\mathcal{L})^n \det(\mathcal{L}^*)(1-\varepsilon) \leq 200^{-n} \eta_\varepsilon(\mathcal{L})^n \det(\mathcal{L}^*)(1+\varepsilon),$$

which is a contradiction. $\square$

2.3 The Gram-Schmidt orthogonalization

Given a basis, $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, we define its Gram-Schmidt orthogonalization $(\widetilde{\mathbf{b}}_1, \ldots, \widetilde{\mathbf{b}}_n)$ by

$$\widetilde{\mathbf{b}}_i = \pi_{\{b_1, \ldots, b_{i-1}\}^\perp}(\mathbf{b}_i) \ .$$

Here, π_A is the orthogonal projection on the subspace A and $\{b_1, \ldots, b_{i-1}\}^\perp$ denotes the subspace orthogonal to $b_1, \ldots, b_{i-1}$.

2.4 Lattice problems

The following problem plays a central role in this paper.

Definition 2.9. *For $\varepsilon \geq 0$, σ a function that maps lattices to non-negative real numbers, and $m \in \mathbb{N}$, ε-DGS$_\sigma^m$ (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 ε-close to $D_{\mathcal{L},s}^m$.*

We omit the parameter ε if $\varepsilon = 0$, the parameter σ if $\sigma = 0$, and the parameter m if $m = 1$. We stress that ε bounds the statistical distance between the *joint* distribution of the output vectors and m independent samples of $D_{\mathcal{L},s}$.

For our applications, we consider the following lattice problems.

Definition 2.10. *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 $\mathbf{y} \in \mathcal{L}$ with $\|\mathbf{y}\| = \lambda_1(\mathcal{L})$.*

Definition 2.11. *For $\gamma = \gamma(n) \geq 1$ (the approximation factor), the decision problem γ-GapSVP is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a number $d > 0$. The goal is to output yes if $\lambda_1(\mathcal{L}) < d$ and no if $\lambda_1(\mathcal{L}) \geq \gamma \cdot d$.*

Definition 2.12. *For $\gamma = \gamma(n) \geq 1$ (the approximation factor), the search problem γ-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 $\mathbf{t} \in \mathbb{R}^n$. The goal is to output a vector $\mathbf{y} \in \mathcal{L}$ with $\|\mathbf{y} - \mathbf{t}\| \leq \gamma \cdot \mathrm{dist}(\mathbf{t}, \mathcal{L})$.*

Definition 2.13. *For $\alpha = \alpha(n) < 1/2$ (the approximation factor), 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 $\mathbf{t} \in \mathbb{R}^n$ with $\mathrm{dist}(\mathbf{t}, \mathcal{L}) \leq \alpha \cdot \lambda_1(\mathcal{L})$. The goal is to output a closest lattice vector to $\mathbf{t}$.*

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.)

2.5 Some lattice algorithms

The following theorem was proven by Ajtai, Kumar, and Sivakumar [AKS01], building on work of Schnorr [Sch87]. (While we use the AKS algorithm repeatedly in the sequel for convenience, we note that we could instead use the conceptually simpler algorithm from [Sch87] to obtain asymptotically identical results.)

Theorem 2.14. *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and $r \geq 2$ and outputs an $r^{n/r}$-reduced basis of $\mathcal{L}$ in time $\exp(O(r)) \cdot \mathrm{poly}(n)$, where we say that a basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice $\mathcal{L}$ is γ-reduced for some $\gamma \geq 1$ if*

1. *$\|\mathbf{b}_1\| \leq \gamma \cdot \lambda_1(\mathcal{L})$; and*
2. *$\pi_{\{\mathbf{b}_1\}^\perp}(\mathbf{b}_2), \ldots, \pi_{\{\mathbf{b}_1\}^\perp}(\mathbf{b}_n)$ is a γ-reduced basis of $\pi_{\{\mathbf{b}_1\}^\perp}(\mathcal{L})$.*

In order to initialize our algorithm, we will need to use a Gaussian sampler such as the one given by Gentry, Peikert, and Vaikuntanathan [GPV08]. For convenience, we use the following modest strengthening of this result, which provides exact samples and gives slightly better bounds on the parameter s.

Theorem 2.15 ([BLP+13, Lemma 2.3])**.** *There is a probabilistic polynomial-time algorithm that takes as input a basis* $\mathbf{B}$ *for a lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and* $s > \|\widetilde{\mathbf{B}}\|\sqrt{C \log n}$ *and outputs a vector that is distributed exactly as* $D_{\mathcal{L},s}$*, where* $\|\widetilde{\mathbf{B}}\| := \max\|\widetilde{\mathbf{b}}_i\|$.

Ideally, we would like to use Theorem 2.14 and Theorem 2.15 to solve DGS_σ for $\sigma = \lambda_1(\mathcal{L}) \cdot r^{n/r}$ in time $\approx \exp(O(r))$. Unfortunately, this does not work. The problem is that Theorem 2.15 only allows us to sample from $D_{\mathcal{L},s}$ if *all* of the Gram-Schmidt vectors are smaller than $\approx s$. We cannot hope to achieve this even for $s \approx \lambda_1(\mathcal{L}) \cdot r^{n/r}$. Indeed, there may not even be such a basis! Instead, we show that we can sample from a sublattice for which we can find such a basis, and we show that this sublattice contains all of the "short" lattice points.

Proposition 2.16. *There is an algorithm that takes as input a lattice* $\mathcal{L} \subset \mathbb{R}^n$ *with* $n \geq 2$, $2 \leq r \leq O(n)$, $M \in \mathbb{N}$ *(the desired number of output vectors), and* $s > 0$ *and outputs a sublattice* $\mathcal{L}'$ *and* M *independent samples from* $D_{\mathcal{L}',s}$ *in time* $(2^{O(r)} + M) \cdot \mathrm{poly}(n)$*. The sublattice* $\mathcal{L}'$ *contains all vectors in* $\mathcal{L}$ *of length at most* $r^{-n/r}s$*. Furthermore, if*

$$s \geq (Cr)^{n/r} \cdot \sqrt{n \log n} \cdot \eta_{0.99}(\mathcal{L}) \ ,$$

then $\mathcal{L}' = \mathcal{L}$.

Proof. On input $\mathcal{L}$ the algorithm first runs the procedure from Theorem 2.14 on $\mathcal{L}$ with parameter $C_1 r$, receiving output $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$. Let $\widetilde{\mathbf{b}}_1, \ldots, \widetilde{\mathbf{b}}_n$ be the corresponding Gram-Schmidt vectors, and let k be maximal such that $\|\widetilde{\mathbf{b}}_i\| \leq s/\sqrt{C_2 \log n}$ for all $i \leq k$. The algorithm then runs the procedure from Theorem 2.15 M times on input $(\mathbf{b}_1, \ldots, \mathbf{b}_k)$ and s and outputs the result together with $\mathcal{L}' = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$.

The running time is clear. It follows immediately from Theorem 2.15 that the output has the correct distribution. If $k = n$, then we are done. Otherwise, let $\mathcal{K} = \pi_{\mathcal{L}'^\perp}(\mathcal{L})$. By Theorem 2.14, we have that

$$\lambda_1(\mathcal{K}) \geq (C_1 r)^{-n/(C_1 r)} \|\widetilde{\mathbf{b}}_{k+1}\| > (C_1 r)^{-n/(C_1 r)} s/\sqrt{C_2 \log n} > r^{-n/r} s \ .$$

The main result follows by noting that $\mathbf{y} \in \mathcal{L} \setminus \mathcal{L}'$ implies that $\|\mathbf{y}\| \geq \lambda_1(\mathcal{K}) > r^{-n/r}s$.

Finally, we note that $\|\widetilde{\mathbf{b}}_i\| \leq (C_1 r)^{n/(C_1 r)} \cdot \lambda_n(\mathcal{L})$ for all i. It follows that, if $s \geq (Cr)^{n/r}\sqrt{\log n} \cdot \lambda_n(\mathcal{L})$, then $k = n$, and therefore $\mathcal{L}' = \mathcal{L}$. The second statement then follows from Lemma 2.8. □

2.6 Probability distributions

Definition 2.17 (Poisson distribution)**.** *The Poisson distribution with parameter* $\lambda > 0$ *is the distribution defined by*

$$\Pr_{X \sim \mathrm{Pois}(\lambda)}[X = r] = \frac{\lambda^r}{r!} \cdot e^{-\lambda}$$

for all $m \in \mathbb{N}$.

Intuitively, the Poisson distribution is the distribution obtained by, e.g., counting the number of decay events over some fixed time period in some large, homogenous radioactive source. The parameter λ is just the expected count.

Lemma 2.18 (Poisson tail bounds [Gly87])**.** *For $\lambda > 0$ let X be a* $\mathrm{Pois}(\lambda)$ *random variable. Then,*

- *for any $0 \leq m < \lambda$,*
$$\Pr(X \leq m) \leq \frac{\exp(-\lambda)\lambda^m}{m!(1-(m/\lambda))} ,$$
- *and for any $m > \lambda - 1$,*
$$\Pr(X \geq m) \leq \frac{\exp(-\lambda)\lambda^m}{m!(1-(\lambda/(m+1)))} .$$

Corollary 2.19. *For any $\alpha > 0$, there exist $C_1, C_2 > 0$ such that the following holds for all $m \geq 1$. If X is a* $\mathrm{Pois}(\lambda)$ *random variable for some $\lambda < (1-\alpha)m$ then*

$$\Pr(X \geq m) \leq C_1 \exp(-C_2 m) ,$$

and similarly, if $\lambda > (1+\alpha)m$ then

$$\Pr(X \leq m) \leq C_1 \exp(-C_2 m) .$$

Proof. Stirling's approximation implies the inequality $m! \geq (m/e)^m$ valid for all $m \geq 1$, which together with Lemma 2.18 implies in both cases the upper bound

$$C \exp(-m(\lambda/m - \log(\lambda/m) + 1)) .$$

The function $x - \log x + 1$ is non-negative and strictly convex on $x > 0$ and obtains its minimum of 0 at $x = 1$. As a result, it is uniformly bounded away from 0 for all x satisfying $|x-1| \geq \alpha$. □

We will also need the Chernoff-Hoeffding bound [Hoe63].

Lemma 2.20 (Chernoff-Hoeffding bound)**.** *Let $X_1, \ldots, X_N$ be independent and identically distributed random variables with $0 \leq X_i \leq a$ and expectation μ. Then, for any $\delta > 0$,*

$$\Pr\left[\frac{1}{N} \cdot \sum X_i \geq (1+\delta)\mu\right] \leq \exp(-C\delta^2 N\mu/a) ,$$

and

$$\Pr\left[\frac{1}{N} \cdot \sum X_i \leq (1-\delta)\mu\right] \leq \exp(-C\delta^2 N\mu/a) .$$

Lemma 2.21 (Multinomial to independent Poisson)**.** *Let $\lambda > 0$ and $\mathbf{p} \in [0,1]^N$ with $\sum p_i = 1$. Consider the process that first samples $r \sim \mathrm{Pois}(\lambda)$ and then samples $X_1, \ldots, X_r$ independently with $\Pr[X_j = i] = p_i$. For each i, let Y_i be the number of occurrences of i in the sequence $X_1, \ldots, X_r$. Then, Y_i is distributed as* $\mathrm{Pois}(\lambda p_i)$ *independently of the other Y_j.*

Proof. Considering the joint distribution, we have

$$\begin{aligned}
\Pr[\mathbf{Y} = \mathbf{a}] &= \Pr[r = \|\mathbf{a}\|_1] \cdot \Pr[\mathbf{Y} = \mathbf{a} | r = \|\mathbf{a}\|_1] \\
&= \lambda^{\|\mathbf{a}\|_1} e^{-\lambda} \prod_i \frac{p_i^{a_i}}{a_i!} \\
&= \prod_i \left(\frac{(\lambda p_i)^{a_i}}{a_i!} \cdot e^{-\lambda p_i} \right) ,
\end{aligned}$$

as needed. □

Claim 2.22 (Poisson to Bernoulli)**.** *For $\lambda \leq 1$ and $\kappa \geq 2$, consider the procedure obtained by sampling r from $\mathrm{Pois}(\lambda)$ and then outputting 1 with probability $\min\{1,\ r/\kappa\}$ and 0 otherwise. The output of this procedure is within statistical distance $1/(\lfloor\kappa\rfloor!)$ of the Bernoulli distribution $\mathrm{B}(\lambda/\kappa)$.*

Proof. If X is distributed like $\mathrm{Pois}(\lambda)$, the statistical distance is given by

$$\begin{aligned}\mathbb{E}[X/\kappa - \min\{1,\ X/\kappa\}] &= \mathbb{E}[\max\{0,\ X/\kappa - 1\}] \\ &\leq \kappa^{-1}\,\mathbb{E}[1_{X>\kappa}\cdot X] \\ &= \kappa^{-1}\sum_{r=\lfloor\kappa\rfloor+1}^{\infty} r\lambda^r \exp(-\lambda)/r! \\ &= \kappa^{-1}\lambda\sum_{r=\lfloor\kappa\rfloor}^{\infty} \lambda^r \exp(-\lambda)/r!\ ,\end{aligned}$$

which is at most $1/(\lfloor\kappa\rfloor!)$ by Lemma 2.18 and our choice of parameters. □

3 Sampling from the discrete Gaussian

3.1 Sampling from the square

Recall that a naive bucketing procedure does not weight cosets in the way that we would like. In particular, the resulting number of vectors in the cosets is distributed with probabilities proportional to $\rho_s(\mathbf{c})$, while we would like the probabilities to be proportional to $\rho_s(\mathbf{c})^2$. Theorem 3.3 shows how to use samples from any multinomial distribution to sample from the "squared distribution" (with small error).

The "square sampler" that we present in Theorem 3.3 needs to compute an estimate of the maximal probability $\max p_i$, given samples from some probability distribution with respective probabilities $(p_1, \ldots, p_N)$. The following proposition shows that there is a relatively efficient way of estimating $\max p_i$. The proposition is included here for completeness. In our application, we will know which of the elements $1, \ldots, N$ has maximal probability, so we could instead simply estimate $\max p_i$ directly.

Proposition 3.1 (Estimating p_{max})**.** *There is an algorithm that takes as input $\kappa \geq 1$ (the confidence parameter) and a sequence of M elements from $\{1, \ldots, N\}$ and outputs a value $\tilde{p}_{\mathsf{max}}$ such that, if the input consists of $M \geq \kappa/p_{\mathsf{max}}$ independent samples from the distribution that assigns probability p_i to element i, then*

$$p_{\mathsf{max}} \leq \tilde{p}_{\mathsf{max}} \leq 4p_{\mathsf{max}}$$

except with probability at most $C_1 N \log N \exp(-C_2\kappa)$, where $p_{\mathsf{max}} = \max p_i$. The algorithm runs in time $M \cdot \mathrm{poly}(\log\kappa, \log N)$.

Proof. The algorithm is the following. Initialize $p = 1$. Sample r from $\mathrm{Pois}(\kappa/p)$ and read the next r elements in the input sequence (or fail if there are not enough elements remaining). Count how many times each $i \in \{1, \ldots, N\}$ appears in this subsequence. If there exists an i appearing at least $\kappa/3$ times, output p and stop. Otherwise, divide p by 2 and repeat.

The running time is clear. By Lemma 2.21, at each iteration the number of times i appears is distributed like $\mathrm{Pois}(\kappa p_i/p)$ independently of everything else. Consider now the iterations with

$p > 4p_{\mathsf{max}}$. Since $p_{\mathsf{max}} > 1/N$, there are at most $O(\log N)$ such iterations, and in each there are N possible values of i. Therefore, by Corollary 2.19 and a union bound, the probability that there exists an iteration with $p > 4p_{\mathsf{max}}$ and an i that appears there at least $\kappa/3$ times is at most $C_1 N \log N \cdot \exp(-C_2\kappa)$. Finally, consider the iteration in which $p_{\mathsf{max}} < p \leq 2p_{\mathsf{max}}$, and let i be the index achieving p_{max}. Then by Corollary 2.19 again, with all but probability $C_1 \exp(-C_2\kappa)$, the item i appears at least $\kappa/3$ times. To summarize, assuming none of the bad events happens, the output satisfies $p_{\mathsf{max}} < \tilde{p}_{\mathsf{max}} \leq 4p_{\mathsf{max}}$ as desired. □

Definition 3.2. *For a vector $\mathbf{p} \in [0,1]^N$ with $\sum p_i = 1$, let $\mathbf{p}^2 = (p_1^2/p_{\mathsf{col}}, \ldots, p_N^2/p_{\mathsf{col}})$ where $p_{\mathsf{col}} := \sum p_i^2$.*

Theorem 3.3 (Square sampler)**.** *There is an algorithm that takes as input $\kappa \geq 2$ (the confidence parameter) and M elements from $\{1, \ldots, N\}$ and outputs a sequence of elements from the same set such that*

1. *the running time is $M \cdot \mathrm{poly}(\log \kappa, \log N)$;*
2. *each $i \in \{1, \ldots, N\}$ appears at least twice as often in the input as in the output; and*
3. *if the input consists of $M \geq 10\kappa^2/\max p_i$ independent samples from the distribution that assigns probability p_i to element i, then the output is within statistical distance $C_1 MN \log N \exp(-C_2\kappa)$ of m independent samples with respective probabilities $\mathbf{p}^2$ where $m \geq M \cdot \sum p_i^2/(32\kappa \max p_i)$ is a random variable.*

Proof. The algorithm first runs the procedure from Proposition 3.1 on the first $M/2$ elements from its input sequence, receiving as output $\tilde{p}_{\mathsf{max}}$. The algorithm then reads the remaining elements in sequence. If it ever reads the last element of the input, it fails. For $j = 1, \ldots, M\tilde{p}_{\mathsf{max}}/4$, the algorithm samples r according to $\mathrm{Pois}(1/\tilde{p}_{\mathsf{max}})$ and takes the next r unused elements in the input. For $i = 1, \ldots, N$, let $a_{i,j}$ be the number of times element i appears in the jth such subsequence. For each i, j let $b_{i,j}$ be 1 with probability $\min\{1, a_{i,j}/\kappa\}$ and 0 otherwise. (To achieve the correct running time, we do not actually explicitly store these values when $a_{i,j} = b_{i,j} = 0$.)

Finally, the algorithm looks through the next $M/6$ elements, one element at a time (or it fails if there are not $M/6$ elements remaining). When it sees element i, it adds it to its output if $b_{i,j} = 1$ where $j \geq 1$ is the smallest index such that $b_{i,j}$ is unused (or it fails if there is no unused $b_{i,j}$).

The running time of the algorithm is clear. We first prove that the output elements have the correct distribution, ignoring failure. By Proposition 3.1, we can assume that $\max p_i \leq \tilde{p}_{\mathsf{max}} \leq 4 \max p_i$, introducing statistical distance at most $C_1 N \log N \exp(-C_2\kappa)$. Then, by Lemma 2.21, the $a_{i,j}$ are distributed independently as $\mathrm{Pois}(p_i/\tilde{p}_{\mathsf{max}})$. By Claim 2.22, each $b_{i,j}$ is within statistical distance $C_1 \exp(-C_2\kappa)$ of $\mathrm{B}(p_i/(\kappa\tilde{p}_{\mathsf{max}}))$. So, we assume that the $b_{i,j}$ are independently distributed exactly as $\mathrm{B}(p_i/(\kappa\tilde{p}_{\mathsf{max}}))$, introducing statistical distance that is at most $C_1 NM \exp(-C_2\kappa)$. Then, in the final stage of the algorithm, the probability of outputting i at each step is $p_i^2/(\kappa\tilde{p}_{\mathsf{max}})$. Hence, the individual output samples have the correct distribution. By the Chernoff-Hoeffding bound (Lemma 2.20), the size of the output will be at least $M\sum p_i^2/(8\kappa\tilde{p}_{\mathsf{max}}) \geq M\sum p_i^2/(32\kappa \max p_i)$ except with probability at most $\exp(-C\kappa)$.

We now prove that the algorithm rarely fails. The number of inputs used in the first stage is distributed as $\mathrm{Pois}(M/4)$, which by Corollary 2.19 is at most $M/2$ except with probability at most $C_1 \exp(-C_2\kappa)$. Applying the Chernoff-Hoeffding bound again, we have that the number of coins $b_{i,j}$ used for a fixed i is at most $Mp_i/4 \leq M\tilde{p}_{\mathsf{max}}/4$ except with probability at most $\exp(-C\kappa)$. So, the algorithm fails with probability at most $C_1 N \exp(-C_2\kappa)$.

Finally, we note that each i appearing in the output corresponds to two copies of i appearing in the input: one corresponding to some value $a_{i,j} > 0$ and another sampled in the final stage. □

3.2 A discrete Gaussian combiner

Ideally, we would like the average of two vectors sampled from $D_{\mathcal{L},s}$ to be distributed as $D_{\mathcal{L},s'}$ for some $s' < s$. Unfortunately, this is false for the simple reason that the average of two lattice vectors may not be in the lattice! The following lemma shows that we do obtain the desired distribution if we condition on the result being in the lattice. The number of vectors that we output will depend on the expression $\sum \rho_s(\mathbf{c})^2$ where $\mathbf{c}$ ranges over all cosets of $2\mathcal{L}$ over $\mathcal{L}$, so we analyze this expression as well. (Note that, for two lattice vectors $\mathbf{X}_1$ and $\mathbf{X}_2$, we have $(\mathbf{X}_1 + \mathbf{X}_2)/2 \in \mathcal{L}$ if and only if $\mathbf{X}_1$ and $\mathbf{X}_2$ are in the same coset over $2\mathcal{L}$. So, the cosets of $2\mathcal{L}$ arise naturally in this context.)

Lemma 3.4. *Let $\mathcal{L} \subset \mathbb{R}^n$ and $s > 0$. Then for all $\mathbf{y} \in \mathcal{L}$,*

$$\Pr_{(\mathbf{X}_1,\mathbf{X}_2)\sim D^2_{\mathcal{L},s}}[(\mathbf{X}_1 + \mathbf{X}_2)/2 = \mathbf{y} \mid \mathbf{X}_1 + \mathbf{X}_2 \in 2\mathcal{L}] = \Pr_{\mathbf{X}\sim D_{\mathcal{L},s/\sqrt{2}}}[\mathbf{X} = \mathbf{y}] \,. \tag{1}$$

Furthermore,

$$\sum_{\mathbf{c}\in\mathcal{L}/(2\mathcal{L})} \rho_s(\mathbf{c})^2 = \rho_{s/\sqrt{2}}(\mathcal{L})^2 \,.$$

Proof. Multiplying the left-hand side of (1) by $\Pr_{(\mathbf{X}_1,\mathbf{X}_2)\sim D^2_{\mathcal{L},s}}[\mathbf{X}_1 + \mathbf{X}_2 \in 2\mathcal{L}] = \rho_s(\mathcal{L})^{-2} \sum_{\mathbf{c}\in\mathcal{L}/(2\mathcal{L})} \rho_s(\mathbf{c})^2$ we get for any $\mathbf{y} \in \mathcal{L}$,

$$\begin{aligned}
\Pr_{(\mathbf{X}_1,\mathbf{X}_2)\sim D^2_{\mathcal{L},s}}[(\mathbf{X}_1 + \mathbf{X}_2)/2 = \mathbf{y}] &= \frac{1}{\rho_s(\mathcal{L})^2} \cdot \sum_{\mathbf{x}\in\mathcal{L}} \rho_s(\mathbf{x})\rho_s(2\mathbf{y} - \mathbf{x}) \\
&= \frac{\rho_{s/\sqrt{2}}(\mathbf{y})}{\rho_s(\mathcal{L})^2} \cdot \sum_{\mathbf{x}\in\mathcal{L}} \rho_{s/\sqrt{2}}(\mathbf{x} - \mathbf{y}) \\
&= \frac{\rho_{s/\sqrt{2}}(\mathbf{y})}{\rho_s(\mathcal{L})^2} \cdot \rho_{s/\sqrt{2}}(\mathcal{L}) \\
&= \rho_s(\mathcal{L})^{-2} \cdot \rho_{s/\sqrt{2}}(\mathcal{L})^2 \Pr_{\mathbf{X}\sim D_{\mathcal{L},s/\sqrt{2}}}[\mathbf{X} = \mathbf{y}] \,.
\end{aligned}$$

Hence both sides of (1) are proportional to each other. Since they are probabilities, they are actually equal. In particular, the ratio between them, $\sum_{\mathbf{c}\in\mathcal{L}/(2\mathcal{L})} \rho_s(\mathbf{c})^2/\rho_{s/\sqrt{2}}(\mathcal{L})^2$, is one. □

Proposition 3.5. *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$, $\kappa \geq 2$ (the confidence parameter), and a sequence of vectors from $\mathcal{L}$, and outputs a sequence of vectors from $\mathcal{L}$ such that, if the input consists of $M \geq 10\kappa^2 \cdot \rho_s(\mathcal{L})/\rho_s(2\mathcal{L})$ independent samples from $D_{\mathcal{L},s}$ for some $s > 0$, then the output is within statistical distance $M\exp(C_1 n - C_2\kappa)$ of m independent samples from $D_{\mathcal{L},s/\sqrt{2}}$ where m is a random variable with*

$$m \geq M \cdot \frac{1}{32\kappa} \cdot \frac{\rho_{s/\sqrt{2}}(\mathcal{L})^2}{\rho_s(\mathcal{L})\rho_s(2\mathcal{L})} \,.$$

The running time of the algorithm is at most $M \cdot \mathrm{poly}(n, \log\kappa)$.

Proof. Let $(\mathbf{X}_1, \ldots, \mathbf{X}_M)$ be the input vectors. For each i, let $\mathbf{c}_i \in \mathcal{L}/(2\mathcal{L})$ be the coset of $\mathbf{X}_i$. The combiner runs the algorithm from Theorem 3.3 with input κ and $(\mathbf{c}_1, \ldots, \mathbf{c}_M)$, receiving output $(\mathbf{c}'_1, \ldots, \mathbf{c}'_m)$. (Formally, we must encode the cosets as integers in $\{1, \ldots, 2^n\}$.) Finally, for each $\mathbf{c}'_i$, it chooses a pair of unpaired vectors $\mathbf{X}_j, \mathbf{X}_k$ with $\mathbf{c}_j = \mathbf{c}_k = \mathbf{c}'_i$ and outputs $\mathbf{Y}_i = (\mathbf{X}_j + \mathbf{X}_k)/2$.

The running time of the algorithm follows from Item 1 of Theorem 3.3. Furthermore, we note that by Item 2 of the same theorem, there will always be a pair of indices j, k for each i as above.

To prove correctness, we observe that for $\mathbf{c} \in \mathcal{L}/(2\mathcal{L})$ and $\mathbf{y} \in \mathbf{c}$,

$$\Pr[\mathbf{X}_i = \mathbf{y}] = \frac{\rho_s(\mathbf{c})}{\rho_s(\mathcal{L})} \cdot \Pr_{\mathbf{x} \sim D_{\mathbf{c},s}}[\mathbf{X} = \mathbf{y}] .$$

In particular, we have that $\Pr[\mathbf{c}_i = \mathbf{c}] = \rho_s(\mathbf{c})/\rho_s(\mathcal{L})$, and $2\mathcal{L}$ is the coset with the highest probability. Then, the cosets $(\mathbf{c}_1, \ldots, \mathbf{c}_M)$ satisfy the conditions necessary for Item 3 of Theorem 3.3 with $\max p_i = \rho_s(2\mathcal{L})/\rho_s(\mathcal{L})$.

Applying the theorem, up to statistical distance $M \exp(C_1 n - C_2 \kappa)$, we have that the output vectors are independent, and

$$m \geq M \cdot \frac{1}{32\kappa} \cdot \frac{\sum_{\mathbf{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(\mathbf{c})^2}{\rho_s(\mathcal{L})\rho_s(2\mathcal{L})} = M \cdot \frac{1}{32\kappa} \cdot \frac{\rho_{s/\sqrt{2}}(\mathcal{L})^2}{\rho_s(\mathcal{L})\rho_s(2\mathcal{L})} ,$$

where the equality follows from Lemma 3.4. Furthermore, we have $\Pr[\mathbf{c}'_i = \mathbf{c}] = \rho_s(\mathbf{c})^2 / \sum_{\mathbf{c}'} \rho_s(\mathbf{c}')^2$ for any coset $\mathbf{c} \in \mathcal{L}/(2\mathcal{L})$. Therefore, for any $\mathbf{y} \in \mathcal{L}$,

$$\begin{aligned} \Pr[\mathbf{Y}_i = \mathbf{y}] &= \frac{1}{\sum \rho_s(\mathbf{c})^2} \cdot \sum_{\mathbf{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(\mathbf{c})^2 \cdot \Pr_{(\mathbf{X}_j, \mathbf{X}_k) \sim D^2_{\mathbf{c},s}}[(\mathbf{X}_j + \mathbf{X}_k)/2 = \mathbf{y}] \\ &= \Pr_{(\mathbf{X}_1, \mathbf{X}_2) \sim D^2_{\mathcal{L},s}}[(\mathbf{X}_1 + \mathbf{X}_2)/2 = \mathbf{y} \mid \mathbf{X}_1 + \mathbf{X}_2 \in 2\mathcal{L}] . \end{aligned}$$

The result then follows from Lemma 3.4. □

By calling the algorithm from Proposition 3.5 repeatedly, we obtain a general discrete Gaussian combiner.

Corollary 3.6. *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$, $\ell \in \mathbb{N}$ (the step parameter), $\kappa \geq 2$ (the confidence parameter), and $M = (32\kappa)^{\ell+1} 2^n$ vectors in $\mathcal{L}$ such that, if the input vectors are distributed as $D_{\mathcal{L},s}$ for some $s > 0$, then the output is a sequence of $2^{n/2}$ vectors whose distribution is within statistical distance $\ell M \exp(C_1 n - C_2 \kappa)$ of independent samples from $D_{\mathcal{L},2^{-\ell/2}s}$. The algorithm runs in time $\ell M \cdot \mathrm{poly}(n, \log \kappa)$.*

Proof. Let $\mathcal{X}_0 = (\mathbf{X}_1, \ldots, \mathbf{X}_M)$ be the sequence of input vectors. For $i = 0, \ldots, \ell - 1$, the algorithm calls the procedure from Proposition 3.5 with input $\mathcal{L}$, κ, and $\mathcal{X}_i$, receiving an output sequence $\mathcal{X}_{i+1}$ of some length M_{i+1}. Finally, the algorithm outputs the first $2^{n/2}$ vectors of $\mathcal{X}_\ell$ (or fails if there are not enough vectors).

The running time is clear. Fix $\mathcal{L}$, s, and ℓ. For convenience, let $\psi(i) := \rho_{2^{-i/2}s}(\mathcal{L})$. Note that by Lemma 2.3 we have that $1 \leq \psi(i)/\psi(i+1) \leq 2^{n/2}$ for all i, a fact that we use repeatedly below. We wish to prove by induction that $\mathcal{X}_i$ is within statistical distance $iM \exp(C_1 n - C_2 \kappa)$ of $D^{M_i}_{\mathcal{L},2^{-i/2}s}$ with

$$M_i \geq (32\kappa)^{\ell-i+1} \cdot 2^{n/2} \frac{\psi(i)}{\psi(i+1)} \tag{2}$$

for all i.

Since $M_0 = M = (32\kappa)^{\ell+1}2^n$ and $\psi(0)/\psi(1) \leq 2^{n/2}$, it follows that (2) holds when $i = 0$. Suppose that $\mathcal{X}_i$ has the correct distribution and (2) holds for some i with $0 \leq i < \ell$. Notice that the right-hand side of (2) is at least $10\kappa^2\psi(i)/\psi(i+2)$ and that the latter is precisely the lower bound on M_i appearing in Proposition 3.5. We can therefore apply the proposition and the induction hypothesis, and obtain that (up to statistical distance at most $(i+1)M\exp(C_1 n - C_2\kappa)$), $\mathcal{X}_{i+1}$ has the correct distribution with

$$M_{i+1} \geq M_i \cdot \frac{1}{32\kappa} \cdot \frac{\psi(i+1)^2}{\psi(i)\psi(i+2)} \geq (32\kappa)^{\ell-i} \cdot 2^{n/2}\frac{\psi(i+1)}{\psi(i+2)} ,$$

as needed.

The result follows by noting that $M_\ell \geq 2^{n/2}\psi(\ell)/\psi(\ell+1) \geq 2^{n/2}$. □

3.3 A general discrete Gaussian sampler

Theorem 3.7. *There is an algorithm that solves* $\exp(-\Omega(\kappa))$*-DGS*$_{2^{n/2}}$ *in time* $2^{n+\mathrm{polylog}(\kappa)+o(n)}$ *for any* $\kappa \geq \Omega(n)$.

Proof. On input $\mathcal{L} \subset \mathbb{R}^n$ a lattice, $s > 0$, and $\kappa \geq \Omega(n)$, the algorithm behaves as follows. First, it runs the sampler from Proposition 2.16 on $\mathcal{L}$ with parameters r, $\hat{s} = 2^{\ell/2}s$, and $M = (32\kappa)^{\ell+2} \cdot 2^n$, with r and ℓ to be set in the analysis. It receives as output a sublattice $\mathcal{L}' \subseteq \mathcal{L}$ and vectors $\mathbf{X}_1, \ldots, \mathbf{X}_M \in \mathcal{L}'$. It then runs the combiner from Corollary 3.6 with input $\mathcal{L}'$, ℓ, κ, and $\mathbf{X}_1, \ldots, \mathbf{X}_M$ and outputs the result.

The running time of the first stage of the algorithm is $(2^{O(r)} + M) \cdot \mathrm{poly}(n)$ by Proposition 2.16, and by Corollary 3.6, the running time of the second stage is $M\ell \cdot \mathrm{poly}(n, \log\kappa)$. Setting $\ell = 4\lceil \log\kappa + \log^2 n \rceil$ and $r = n/\log n$, it follows that the running time is as claimed. Applying the proposition and corollary again, we have that the output is within statistical distance $\exp(-\Omega(\kappa))$ of $D^{2^{n/2}}_{\mathcal{L}',s}$. Furthermore, we have that $\mathcal{L}'$ contains all vectors of length at most $r^{-n/r}\hat{s} > \sqrt{\kappa}s$.

It remains to prove that $D_{\mathcal{L}',s}$ is within statistical distance $\exp(-\Omega(\kappa))$ of $D_{\mathcal{L},s}$. Notice that $D_{\mathcal{L}',s}$ is a restriction of the distribution $D_{\mathcal{L},s}$ to $\mathcal{L}'$ and hence the statistical distance between these two distributions is

$$\Pr_{\mathbf{X}\sim D_{\mathcal{L},s}}[\mathbf{X} \in \mathcal{L} \setminus \mathcal{L}'] < \Pr_{\mathbf{X}\sim D_{\mathcal{L},s}}[\|\mathbf{X}\| > \sqrt{\kappa}s] < \exp(-\Omega(\kappa)) ,$$

as needed, where we used Lemma 2.4. □

4 Solving SVP in $2^{n+o(n)}$ time

4.1 A bound on the Gaussian mass

In this section, we prove a bound on the Gaussian mass of a lattice that follows from an upper bound on the kissing number due to Kabatjanskiĭ and Levenšteĭn [KL78]. In particular, we use the following lemma from [PS09] based on [KL78]. For convenience, we define $\beta := 2^{0.401}$, and we use this notation throughout this section.

Lemma 4.1 ([PS09, Lemma 3])**.** *Let* $\mathcal{L} \subseteq \mathbb{R}^n$ *be a lattice with* $\lambda_1(\mathcal{L}) = 1$*. Then for any* $r \geq 1$*, the number of lattice vectors of length at most* r *is at most* $\beta^{n+o(n)}r^n$.

We now use Lemma 4.1 to bound $\rho_s(\mathcal{L})$.

Lemma 4.2. *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice of rank at least one. Then for any $s > \sqrt{2\pi/n} \cdot \lambda_1(\mathcal{L})$,*

$$\rho_s(\mathcal{L}) \leq 1 + \left(\frac{\beta^2 s^2 n}{2\pi e \cdot \lambda_1(\mathcal{L})^2}\right)^{n/2+1} 2^{o(n)} , \tag{3}$$

and for $s \leq \sqrt{2\pi/n} \cdot \lambda_1(\mathcal{L})$, we have

$$\rho_s(\mathcal{L}) \leq 1 + e^{-\pi\lambda_1(\mathcal{L})^2/s^2} \cdot \beta^{n+o(n)} . \tag{4}$$

We note that an easy calculation shows that the right-hand side of Eq. (3) is never smaller than the right-hand side of Eq. (4). In particular, this means that Eq. (3) actually applies for all s.

Proof of Lemma 4.2. We assume without loss of generality that $\mathcal{L}$ is normalized so that $\lambda_1(\mathcal{L}) = 1$. Let $t := 1 + 1/n$. For $r \geq 1$, define $T_r := \{\mathbf{x} \in \mathbb{R}^n : r \leq \|\mathbf{x}\| < tr\}$. By Lemma 4.1, $|\mathcal{L} \cap T_r| \leq \beta^{n+o(n)} r^n$, and, for any vector $\mathbf{y} \in \mathcal{L} \cap T_r$, $\rho_s(\mathbf{y}) \leq e^{-\pi r^2/s^2}$. Therefore,

$$\rho_s(\mathcal{L} \cap T_r) \leq e^{-\pi r^2/s^2} \cdot \beta^{n+o(n)} \cdot r^n .$$

So, we have

$$\begin{aligned}
\rho_s(\mathcal{L}) &= 1 + \sum_{i=0}^{\infty} \rho_s(\mathcal{L} \cap T_{t^i}) \\
&\leq 1 + \beta^{n+o(n)} \cdot \sum_{i=0}^{\infty} e^{-\pi t^{2i}/s^2} t^{in} \\
&\leq 1 + (1+s)\beta^{n+o(n)} \cdot \max_{r \geq 1} e^{-\pi r^2/s^2} r^n ,
\end{aligned}$$

where we have used the fact that $e^{-\pi t^{2i}/s^2} t^{in}$ decays geometrically when i is at least, say, $(1+s) \cdot \mathrm{poly}(n)$, and so the sum up to that point is the same as the infinite sum up to a constant factor. Note that for any $a, b > 0$, the maximum of $e^{-ar^2} r^b$ over the interval $r \geq 1$ is obtained at $r = \sqrt{b/(2a)}$ if this value is at least 1 or at $r = 1$ otherwise. The result follows. □

Proposition 4.3. *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice of rank at least one. Let*

$$s = \sqrt{\frac{2\pi e}{\beta^2 n}} \cdot \lambda_1(\mathcal{L}) .$$

Then,

$$\Pr_{\mathbf{X} \sim D_{\mathcal{L},s}}[\|\mathbf{X}\| = \lambda_1(\mathcal{L})] \geq e^{-\beta^2 n/(2e) - o(n)} \approx 1.38^{-n-o(n)} .$$

Proof. By Lemma 4.2, we have that $\rho_s(\mathcal{L}) = 2^{o(n)}$. Therefore,

$$\Pr_{\mathbf{X} \sim D_{\mathcal{L},s}}[\|\mathbf{X}\| = \lambda_1(\mathcal{L})] \geq e^{-\pi/s^2}/\rho_s(\mathcal{L}) \geq e^{-\pi/s^2 - o(n)} = e^{-\beta^2 n/(2e) - o(n)} ,$$

as needed. □

An easy calculation shows that the probability in Proposition 4.3 is maximized to within a factor of two when $s = 1/\eta_1(\mathcal{L}^*)$. I.e., for any shortest non-zero vector $\mathbf{y} \in \mathcal{L}$,

$$\max_s \Pr_{\mathbf{X} \sim D_{\mathcal{L},s}}[\mathbf{X} = \mathbf{y}] \leq 2 \Pr_{\mathbf{X} \sim D_{\mathcal{L},1/\eta_1(\mathcal{L}^*)}}[\mathbf{X} = \mathbf{y}] = \exp(-\pi\eta_1(\mathcal{L}^*)^2 \cdot \lambda_1(\mathcal{L})^2) .$$

4.2 A reduction from SVP to DGS

Theorem 4.4. *There is a reduction from SVP to $\frac{1}{2}$-DGS$^{2^{n/2}}$. The reduction makes $O(n)$ calls to the DGS oracle, preserves the dimension of the lattice, and runs in time $2^{n/2} \cdot \mathrm{poly}(n)$.*

Proof. Let D be an oracle solving $\frac{1}{2}$-DGS$^{2^{n/2}}$. We construct an algorithm solving SVP as follows. It first runs the procedure from Theorem 2.14 on $\mathcal{L}$ with $r = 2$. Let d be the length of the first basis vector in the output. For $i = 0, \ldots, 100n$, the algorithm calls D on $\mathcal{L}$ with parameter $s_i = 1.01^{-i} \cdot d$. Let $\mathbf{x}_i$ be a shortest non-zero vector in the output. Finally, the algorithm outputs a shortest vector among the $\mathbf{x}_i$.

The running time of the algorithm is clear. By Theorem 2.14, we have $d \leq 2^{n/2}\lambda_1(\mathcal{L})$. It follows that there exists some i such that $\hat{s} \leq s_i \leq 1.01\hat{s}$ where $\hat{s} = \sqrt{\pi e/n} \cdot 2^{0.099} \cdot \lambda_1(\mathcal{L})$ (i.e., $\hat{s}$ is the parameter from Proposition 4.3). We assume that the output of D is exactly $D^{2^{n/2}}_{\mathcal{L},s_i}$ when called on s_i, incurring statistical distance at most $1/2$. For such i, by Lemma 2.3 we have

$$\begin{aligned}\Pr_{\mathbf{X}\sim D_{\mathcal{L},s_i}}[\|\mathbf{X}\| = \lambda_1(\mathcal{L})] &\geq 1.01^{-n} \cdot \Pr_{\mathbf{X}\sim D_{\mathcal{L},\hat{s}}}[\|\mathbf{X}\| = \lambda_1(\mathcal{L})] \\ &\geq 1.4^{-n-o(n)} \qquad \text{(Proposition 4.3)} .\end{aligned}$$

The result follows by noting that $1.4 < \sqrt{2}$, so $2^{n/2}$ samples from $D_{\mathcal{L},s_i}$ will contain a shortest vector with probability at least $1 - \exp(-\Omega(n))$. □

Corollary 4.5. *There is an algorithm that solves SVP in time $2^{n+o(n)}$.*

Proof. Combine the reduction from Theorem 4.4 with the algorithm from Theorem 3.7. □

5 Sampling $2^{n/2}$ vectors above smoothing in $2^{n/2}$ time

In this section we present a $2^{n/2}$-time algorithm for DGS$_\sigma$ with σ approximately the smoothing parameter. For our applications (in particular for solving $O(1)$-GapSVP) we will need a slightly stronger guarantee from the algorithm. Namely, when asked to produce samples with too small a parameter $s \leq \sigma$, the output should still consist of discrete Gaussian samples of the desired parameter, but potentially less of them (or even none at all). We make this property formal in the following slight modification of Definition 2.9.

Definition 5.1. *For $\varepsilon \geq 0$, σ a function that maps lattices to non-negative real numbers, and $m \in \mathbb{N}$, ε-hDGS$^m_\sigma$ (the* honest *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 $s > 0$. The goal is for the output distribution to be ε-close to $D^{m'}_{\mathcal{L},s}$ for some independent random variable $m' \geq 0$. If $s > \sigma(\mathcal{L})$, then m' must equal m.*

5.1 Sampling from the square root

Claim 5.2. *There is an algorithm that, given black-box access to the Bernoulli distribution $\mathrm{B}(p)$ for unknown $0 < p \leq 1$, outputs $b \in \{0,1\}$ such that the distribution of b is exactly $\mathrm{B}(\sqrt{p})$. The expected running time of the algorithm is $O(1/p)$. Furthermore, if the algorithm's input is restricted to $\tau \in \mathbb{N}$ independent samples from the Bernoulli distribution $\mathrm{B}(p)$ for unknown $0 \leq p \leq 1$, then the distribution of b is within statistical distance $(1-p)^\tau$ of $\mathrm{B}(\sqrt{p})$, and the algorithm runs in time $O(\tau)$.*

Proof. The algorithm repeatedly samples from $\mathrm{B}(p)$ until the first time that it sees a 1. Let k be the number of times that zero appears before this first one. (E.g., if the input sequence is $(0,0,0,1,\ldots)$, $k = 3$.) Then, the algorithm tosses $2k$ unbiased coins and outputs 1 if exactly half of them are heads and 0 otherwise. (In the case when the algorithm's input length is restricted, it outputs 0 if there are no ones in its input sequence.)

Correctness is immediate from the following identity, which can be obtained by taking the Taylor expansion of $p^{-1/2}$ around $p = 1$ and then multiplying by p,

$$\sqrt{p} = \sum_{k=0}^{\infty} \binom{2k}{k} 2^{-2k} (1-p)^k p \; .$$

The expected running time is clearly proportional to $\sum (1-p)^k = 1/p$. □

Definition 5.3. *For a vector $\mathbf{p} \in [0,1]^N$ with $\sum p_i = 1$, let $\sqrt{\mathbf{p}} = (\sqrt{p_1}/p_{\mathsf{sqrt}}, \ldots, \sqrt{p_N}/p_{\mathsf{sqrt}})$ where $p_{\mathsf{sqrt}} = \sum \sqrt{p_i}$.*

Theorem 5.4 (Square-root sampler)**.** *There is an algorithm that takes as input $\kappa \geq 2$ (the confidence parameter), $t \geq 1$ (an upper bound on the ratio $\max_{i,j} p_i/p_j$), and M elements from $\{1,\ldots,N\}$ and outputs a sequence of elements from the same set such that*

1. *the running time is $M \cdot \mathrm{poly}(\log N, \kappa, t)$;*
2. *each $i \in \{1,\ldots,N\}$ appears at least as often in the input as in the output; and*
3. *if the input consists of $M \geq 4\kappa^4 t^2 / \max p_i$ independent samples from the distribution that assigns probability $p_i > 0$ to element i with $t \geq \max_{i,j} p_i/p_j$, then the output is within statistical distance $C_1 MN \log N \exp(-C_2\kappa)$ of m independent samples with respective probabilities $\sqrt{\mathbf{p}}$ where $m = M/(16\kappa^3 t^{3/2})$.*

Proof. The algorithm first runs the procedure from Proposition 3.1 on the first $M/2$ elements from its input sequence, receiving as output $\tilde{p}_{\mathsf{max}}$. If $\tilde{p}_{\mathsf{max}} > 4t/N$, it fails. Otherwise, for $j = 1,\ldots,M\tilde{p}_{\mathsf{max}}/3$, the algorithm samples r from $\mathrm{Pois}(1/\tilde{p}_{\mathsf{max}})$ and takes the next r unused elements in the input (or fails if there are not r unused elements remaining). For $i = 1,\ldots,N$, let $a_{i,j}$ be the number of times element i appears in the jth such subsequence. For each i,j let $b_{i,j}$ be 1 with probability $\min\{1,\ a_{i,j}/\kappa\}$ and 0 otherwise. Let $\tau = \lceil \kappa^2 t \rceil$. Then, for $i = 1,\ldots,N$ and $k = 0,\ldots,M\tilde{p}_{\mathsf{max}}/(3\tau) - 1$, let $b^*_{i,k}$ be the output of the procedure from Claim 5.2 on input $(b_{i,\tau k+1},\ldots,b_{i,\tau(k+1)})$.

Finally, the algorithm repeats the following at most $M/(5\tau)$ times: it samples $i \in \{1,\ldots,N\}$ uniformly at random, and adds it to its output if $b^*_{i,k} = 1$ where $k \geq 1$ is the smallest index such that $b^*_{i,k}$ is unused (or it fails if there is no unused $b^*_{i,k}$). The algorithm stops as soon as its output contains m samples. It fails if the output contains fewer than m samples when the loop ends.

Note that the number of coins $b_{i,j}$ and $b^*_{i,k}$ is less than $MN\tilde{p}_{\mathsf{max}} \leq 4Mt$. It follows that the running time is as claimed. Note also that each i appearing in the output corresponds to some $a_{i,j} > 0$, which in turn corresponds to at least one element i in the input.

We first prove that the output elements have the correct distribution, ignoring failure. By Proposition 3.1, we can assume that $\max p_i \leq \tilde{p}_{\mathsf{max}} \leq 4 \max p_i$, introducing statistical distance at most $C_1 N \log N \exp(-C_2\kappa)$. By Lemma 2.21 and Claim 2.22, we can further assume that $b_{i,j}$ are independent and distributed exactly as $\mathrm{B}(p_i/(\kappa\tilde{p}_{\mathsf{max}}))$, introducing statistical distance that is

at most $C_1 NM \exp(-C_2\kappa)$. Applying Claim 5.2, we have that $b^*_{i,k}$ is within statistical distance $(1 - p_i/(\kappa\tilde{p}_{\max}))^\tau \le \exp(-C\kappa)$ of $\mathrm{B}(\sqrt{p_i/(\kappa\tilde{p}_{\max})})$. So, the outputs have the correct distribution.

We now prove that the algorithm rarely fails. The number of inputs used in the second stage is distributed as $\mathrm{Pois}(M/3)$, which by Corollary 2.19 is at most $M/2$ except with probability at most $C_1 \exp(-C_2\kappa)$. Applying the Chernoff-Hoeffding bound (Lemma 2.20), we have that the number of coins $b^*_{i,k}$ used for a fixed i is at most $M/(3\tau N) \le M\tilde{p}_{\max}/(3\tau)$ except with probability at most $\exp(-C\kappa)$ (where we have used the fact that $M/(3\tau N) \ge \kappa$). Finally, applying the Chernoff-Hoeffding bound again, we have that after $M/(5\tau)$ steps, the size of the output will be at least $M/(8\tau) \cdot \sqrt{\min p_i/(\kappa\tilde{p}_{\max})} \ge m$ except with probability at most $\exp(-C\kappa)$. So, the algorithm fails with probability at most $C_1 N \exp(-C_2\kappa)$. □

In order to create an "honest" discrete Gaussian sampler as in Definition 5.1 that uses the square-root sampler, we will need a way to "check t," so that we only use the square-root sampler when we can be sure that Item 3 applies. We achieve this with the following simple claim.

Claim 5.5. *There is an algorithm that takes as input a number $t \ge 1$ and M independent samples from the distribution that assigns probability $p_i > 0$ to each element $i \in \{1, \dots, N\}$, runs in time $M \cdot \mathrm{polylog}(N, M, t)$, and satisfies*

1. *if $t < \max p_i/p_j$, then the algorithm outputs no with probability at least $1 - 2\exp(-CM/(tN))$; and*
2. *if $t \ge 4 \max p_i/p_j$, then the algorithm outputs yes with probability at least $1 - N\exp(-CM/(tN))$.*

Proof. The algorithm is quite simple. On input $X_1, \dots, X_M$, let $T_{\max} := \max_i |\{j : X_j = i\}|$ and $T_{\min} := \min_i |\{j : X_j = i\}|$. The algorithm outputs no if $t \cdot T_{\min} < 2T_{\max}$ and yes otherwise.

The running time of the algorithm is clear. Let $p_{\max} = \max p_i$ and $p_{\min} = \min p_i$. Suppose $t < p_{\max}/p_{\min}$. Then, by the Chernoff-Hoeffding bound (Lemma 2.20), we have $T_{\max} > p_{\max}M/\sqrt{2}$ except with probability at most $\exp(-Cp_{\max}M) \le \exp(-CM/(tN))$. Similarly, we have that $T_{\min} < \sqrt{2}p_{\max}M/t$ except with probability at most $\exp(-Cp_{\max}M/t) \le \exp(-CM/(tN))$. Item 1 follows.

Now, suppose $t \ge 4 \max p_i/p_j$. Then, for any i, by the Chernoff-Hoeffding bound we have $p_{\min}M/\sqrt{2} < |\{j : X_j = i\}| < \sqrt{2}p_{\max}M$ except with probability at most $2\exp(-CM/(tN))$. Item 2 then follows by union bound. □

5.2 A more efficient combiner that works above smoothing

The following lemma generalizes the first part of Lemma 3.4. In particular, we recover Lemma 3.4 when $\mathcal{L}' = 2\mathcal{L}$. (Note that, since we require that $2\mathcal{L} \subseteq \mathcal{L}'$, we have that the sum of two lattice vectors $\mathbf{X}_1 + \mathbf{X}_2$ is in $\mathcal{L}'$ if and only if $\mathbf{X}_1$ and $\mathbf{X}_2$ are in the same coset of $\mathcal{L}'$ over $\mathcal{L}$.)

Lemma 5.6. *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice, and let $\mathcal{L}' \subseteq \mathcal{L}$ be a sublattice with $2\mathcal{L} \subseteq \mathcal{L}'$. Then for any $\mathbf{y} \in \mathcal{L}'$ and $s > 0$, we have*

$$\Pr_{(\mathbf{X}_1,\mathbf{X}_2)\sim D^2_{\mathcal{L},s}}[\mathbf{X}_1 + \mathbf{X}_2 = \mathbf{y} \mid \mathbf{X}_1 + \mathbf{X}_2 \in \mathcal{L}'] = \frac{\rho_{\sqrt{2}s}(2\mathcal{L} + \mathbf{y})^2}{p_{\mathsf{col}}} \cdot \Pr_{\mathbf{X}\sim D_{2\mathcal{L}+\mathbf{y},\sqrt{2}s}}[\mathbf{X} = \mathbf{y}] \ ,$$

where $p_{\mathsf{col}} = \sum_{\mathbf{d} \in \mathcal{L}'/(2\mathcal{L})} \rho_{\sqrt{2}s}(\mathbf{d})^2$.

Proof. It suffices to show that the probability on the left-hand side is proportional to $\rho_{\sqrt{2}s}(2\mathcal{L} + \mathbf{y})\rho_{\sqrt{2}s}(\mathbf{y})$. Indeed,

$$\begin{aligned}\Pr_{(\mathbf{X}_1,\mathbf{X}_2)\sim D^2_{\mathcal{L},s}}[\mathbf{X}_1 + \mathbf{X}_2 = \mathbf{y}] &= \frac{1}{\rho_s(\mathcal{L})^2} \cdot \sum_{\mathbf{x}\in\mathcal{L}} \rho_s(\mathbf{x})\rho_s(\mathbf{y}-\mathbf{x}) \\ &= \frac{\rho_{\sqrt{2}s}(\mathbf{y})}{\rho_s(\mathcal{L})^2} \cdot \sum_{\mathbf{x}\in\mathcal{L}} \rho_{\sqrt{2}s}(2\mathbf{x}-\mathbf{y}) \\ &= \frac{\rho_{\sqrt{2}s}(\mathbf{y})}{\rho_s(\mathcal{L})^2} \cdot \rho_{\sqrt{2}s}(2\mathcal{L}+\mathbf{y}) \ . \end{aligned}$$

□

Proposition 5.7. *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$, a sublattice $\mathcal{L}' \subseteq \mathcal{L}$ of index $2^a \geq 2^{n/2}$ with $2\mathcal{L} \subseteq \mathcal{L}'$, $\kappa \geq 2$ (the confidence parameter), and a sequence of vectors from $\mathcal{L}$ such that, if the input consists of $M \geq C\kappa^5 2^a$ independent samples from $D_{\mathcal{L},s}$ for some $s > 0$, then*

1. *the running time of the algorithm is $M \cdot \mathrm{poly}(n,\kappa)$;*
2. *the output distribution is $M\exp(C_1 n - C_2\kappa)$-close to m independent samples from $D_{\mathcal{L}',\sqrt{2}s}$ where $m \in \{0, M/(C\kappa^4)\}$ is an independent random variable; and*
3. *if $s \geq \eta_\varepsilon(\mathcal{L}')$ and $s \geq \sqrt{2}\eta_\varepsilon(\mathcal{L})$, then $m = M/(C\kappa^4)$ where $\varepsilon := 3/4$.*

Proof. Let $(\mathbf{X}_1,\ldots,\mathbf{X}_M)$ be the input vectors, and for each i, let $\mathbf{c}_i \in \mathcal{L}/\mathcal{L}'$ be the coset of $\mathbf{X}_i$. The algorithm first applies the square sampler in a manner similar to that of the algorithm from Proposition 3.5. Namely, the algorithm runs the procedure from Theorem 3.3 with input κ and $(\mathbf{c}_1,\ldots,\mathbf{c}_M)$, receiving output $(\mathbf{c}'_1,\ldots,\mathbf{c}'_q)$. For each $i = 1,\ldots,q$, it chooses a pair of unpaired vectors $\mathbf{X}_j, \mathbf{X}_k$ with $\mathbf{c}_j = \mathbf{c}_k = \mathbf{c}'_i$ and sets $\mathbf{Y}_i = \mathbf{X}_j + \mathbf{X}_k \in \mathcal{L}'$.

Let $t := (1+\varepsilon)^2/(1-\varepsilon)^2 = C$. The algorithm now applies two tests to "check that the distribution is sufficiently smooth." First, it checks if $q \geq M/(32\kappa\sqrt{t})$. If not, it halts and outputs nothing. Next, let $\mathbf{d}_i \in \mathcal{L}'/(2\mathcal{L})$ be the coset of $\mathbf{Y}_i$. The algorithm runs the procedure from Claim 5.5 on the first $\lfloor q/2 \rfloor$ such cosets with parameter $t' := 4t$. It halts and outputs nothing if this procedure outputs no.

The algorithm now applies the square-root sampler to the remaining cosets. Namely, it runs the procedure from Theorem 5.4 with input κ, t', and $(\mathbf{d}_{\lfloor q/2 \rfloor+1},\ldots,\mathbf{d}_q)$, receiving output $(\mathbf{d}'_1,\ldots,\mathbf{d}'_L)$. If $L < M/(C\kappa^4)$, it halts and outputs nothing. Otherwise, for each $i \leq M/(C\kappa^4)$, it chooses an unused vector $\mathbf{Y}_j$ with $j > q/2$ and $\mathbf{d}_j = \mathbf{d}'_i$ and adds it to its output.

The running time of the algorithm follows from Item 1 of Theorem 3.3 and the corresponding Item 1 of Theorem 5.4. Furthermore, we note that by Item 2 of Theorem 3.3, the first step of the above algorithm will always be able to find unused j,k satisfying $\mathbf{c}_j = \mathbf{c}_k = \mathbf{c}'_i$, and by Item 2 of Theorem 5.4, the second step will always be able to find an unused j satisfying $\mathbf{d}_j = \mathbf{d}'_i$.

We now prove Item 2. Note that, since the index of $\mathcal{L}'$ over $\mathcal{L}$ is 2^a, the maximal probability of a coset must be at least 2^{-a}. It follows that $(\mathbf{c}_1,\ldots,\mathbf{c}_M)$ satisfy the conditions necessary for Item 3 of Theorem 3.3. Applying the theorem, we have that (up to statistical distance $M\exp(C_1 n - C_2\kappa)$) the output vectors $(\mathbf{Y}_1,\ldots,\mathbf{Y}_q)$ are independent and

$$q \geq M \cdot \frac{1}{32\kappa} \cdot \frac{\sum_{\mathbf{c}\in\mathcal{L}/\mathcal{L}'} \rho_s(\mathbf{c})^2}{\rho_s(\mathcal{L})\rho_s(\mathcal{L}')} \ . \tag{5}$$

Furthermore, they assign to each $\mathbf{y} \in \mathcal{L}'$ the probability

$$\begin{aligned}\Pr[\mathbf{Y}_i = \mathbf{y}] &= \frac{1}{\sum_{\mathbf{c}\in\mathcal{L}/\mathcal{L}'} \rho_s(\mathbf{c})^2} \cdot \sum_{\mathbf{c}\in\mathcal{L}/\mathcal{L}'} \rho_s(\mathbf{c})^2 \cdot \Pr_{(\mathbf{X}_1,\mathbf{X}_2)\sim D^2_{\mathbf{c},s}}[\mathbf{X}_1 + \mathbf{X}_2 = \mathbf{y}] \\ &= \Pr_{(\mathbf{X}_1,\mathbf{X}_2)\sim D^2_{\mathcal{L},s}}[\mathbf{X}_1 + \mathbf{X}_2 = \mathbf{y} \mid \mathbf{X}_1 + \mathbf{X}_2 \in \mathcal{L}'] \\ &= \frac{\rho_{\sqrt{2}s}(2\mathcal{L}+\mathbf{y})^2}{\sum_{\mathbf{d}\in\mathcal{L}'/(2\mathcal{L})} \rho_{\sqrt{2}s}(\mathbf{d})^2} \cdot \Pr_{\mathbf{X}\sim D_{2\mathcal{L}+\mathbf{y},\sqrt{2}s}}[\mathbf{X} = \mathbf{y}]\,,\end{aligned}$$

where we have used Lemma 5.6. In particular, the distribution of each $\mathbf{d}_i$ is given by

$$\Pr[\mathbf{d}_i = \mathbf{d}] = \frac{\rho_{\sqrt{2}s}(\mathbf{d})^2}{\sum_{\mathbf{d}'\in\mathcal{L}'/(2\mathcal{L})} \rho_{\sqrt{2}s}(\mathbf{d}')^2}\,,$$

for $\mathbf{d} \in \mathcal{L}'/(2\mathcal{L})$. The highest probability is obtained at $\mathbf{d} = 2\mathcal{L}$, and we denote it by $p_{\max}$.

Since the algorithm outputs nothing otherwise (in which case Item 2 trivially holds), we only need to consider the case when

$$q \geq \frac{M}{32\kappa\sqrt{t}}\,, \tag{6}$$

so we assume this below. In addition, by Item 1 of Claim 5.5, the algorithm will halt after the second "smoothness test" with probability at least $1 - 2\exp(C\kappa)$ unless

$$t' \geq \frac{\rho_{\sqrt{2}s}(2\mathcal{L})^2}{\min_{\mathbf{d}\in\mathcal{L}'/(2\mathcal{L})} \rho_{\sqrt{2}s}(\mathbf{d})^2}\,. \tag{7}$$

So, we can also assume that Eq. (7) holds. Using Eq. (6) and the fact that the index of $\mathcal{L}'$ over $2\mathcal{L}$ is $2^{n-a} \leq 2^a$, we see that $q/2 \geq 4\kappa^4 t'^2/p_{\max}$. Combining this with Eq. (7), we see that the conditions for Item 3 of Theorem 5.4 are satisfied. Let $\mathbf{W}_1, \ldots, \mathbf{W}_L$ be the vectors "chosen by the square-root sampler." Applying the theorem, up to statistical distance $M\exp(C_1 n - C_2\kappa)$, we have that the $\mathbf{W}_i$ are independently distributed, and

$$L = \frac{q}{32\kappa^3 t'^{3/2}} \geq M \cdot \frac{1}{C\kappa^4}\,,$$

as needed, where we have used Eq. (6). Furthermore, we have that for any coset $\mathbf{d} \in \mathcal{L}'/(2\mathcal{L})$,

$$\Pr[\mathbf{d}'_i = \mathbf{d}] = \frac{\rho_{\sqrt{2}s}(\mathbf{d})}{\sum_{\mathbf{d}'\in\mathcal{L}'/(2\mathcal{L})} \rho_{\sqrt{2}s}(\mathbf{d}')} = \frac{\rho_{\sqrt{2}s}(\mathbf{d})}{\rho_{\sqrt{2}s}(\mathcal{L}')}\,.$$

Therefore, for any $\mathbf{y} \in \mathcal{L}'$, we have

$$\Pr[\mathbf{W}_i = \mathbf{y}] = \frac{\rho_{\sqrt{2}s}(2\mathcal{L}+\mathbf{y})}{\rho_{\sqrt{2}s}(\mathcal{L}')} \cdot \Pr_{\mathbf{X}\sim D_{2\mathcal{L}+\mathbf{y},\sqrt{2}s}}[\mathbf{X} = \mathbf{y}] \;= \Pr_{\mathbf{X}\sim D_{\mathcal{L}',\sqrt{2}s}}[\mathbf{X} = \mathbf{y}]\,,$$

as needed.

Finally, we prove Item 3. Suppose that s satisfies $s \geq \eta_\varepsilon(\mathcal{L}')$ and $s \geq \sqrt{2}\eta_\varepsilon(\mathcal{L})$. Note that by Claim 2.6, we have that

$$\frac{\rho_s(\mathcal{L}')\rho_s(\mathcal{L})}{\sum_{\mathbf{c}\in\mathcal{L}/\mathcal{L}'}\rho_s(\mathbf{c})^2} \leq \frac{1+\varepsilon}{1-\varepsilon}\cdot\frac{\rho_s(\mathcal{L})}{\sum_{\mathbf{c}\in\mathcal{L}/\mathcal{L}'}\rho_s(\mathbf{c})} = \sqrt{t}\,.$$

Combining this with Eq. (5) shows that the algorithm will not halt after the first "smoothness test" except with probability at most $M\exp(C_1 n - C_2\kappa)$. Similarly, since $\sqrt{2}s \geq \eta_\varepsilon(2\mathcal{L})$,

$$\frac{\rho_{\sqrt{2}s}(2\mathcal{L})^2}{\min_{\mathbf{d}\in\mathcal{L}'/(2\mathcal{L})}\rho_{\sqrt{2}s}(\mathbf{d})^2} \leq t\,.$$

By applying Item 2 of Claim 5.5, we see that the algorithm also will not halt after the second "smoothness test" except with negligible probability. Therefore, Item 3 holds. □

We are going to apply Proposition 5.7 repeatedly, to a "tower" of lattices $(\mathcal{L}_0,\ldots,\mathcal{L}_\ell)$, as defined next.

Definition 5.8. *For an integer a satisfying $n/2 \leq a \leq n$, we say that $(\mathcal{L}_0,\ldots,\mathcal{L}_\ell)$ is a* tower of lattices in $\mathbb{R}^n$ of index 2^a *if for all i we have $2\mathcal{L}_{i-1} \subseteq \mathcal{L}_i \subset \mathcal{L}_{i-1}$, $\mathcal{L}_i/2 \subseteq \mathcal{L}_{i-2}$, and the index of $\mathcal{L}_i$ in $\mathcal{L}_{i-1}$ is 2^a.*

We next observe that it is easy to construct a tower with any desired final lattice $\mathcal{L}_\ell$. In fact, one can even choose $\mathcal{L}_{\ell-1}$, the second-to-last lattice in the tower.

Claim 5.9. *There is a polynomial-time algorithm that given integers $\ell \geq 1$ and $n/2 \leq a \leq n$, as well as two lattices $\mathcal{L}$ and $\mathcal{L}'$ in $\mathbb{R}^n$ satisfying $\mathcal{L} \subseteq \mathcal{L}' \subseteq \mathcal{L}/2$ with the index of $\mathcal{L}$ in $\mathcal{L}'$ being 2^a, outputs a tower of lattices $(\mathcal{L}_0,\ldots,\mathcal{L}_\ell)$ of index 2^a with $\mathcal{L}_\ell = \mathcal{L}$, $\mathcal{L}_{\ell-1} = \mathcal{L}'$, and $\mathcal{L}_0 \supseteq 2^{-\lfloor \ell a/n \rfloor}\mathcal{L}$.*

Proof. Let $\mathbf{b}_1,\ldots,\mathbf{b}_n$ be a basis of $\mathcal{L}$ chosen so that $\mathbf{b}_1/2,\ldots,\mathbf{b}_a/2,\mathbf{b}_{a+1},\ldots,\mathbf{b}_n$ is a basis of $\mathcal{L}'$. It is not difficult to see that such a basis exists. Then define the tower by "cyclically halving a coordinates," namely,

$$\begin{aligned}
\mathcal{L}_\ell &= \mathcal{L}(\mathbf{b}_1,\ldots,\mathbf{b}_n),\\
\mathcal{L}_{\ell-1} &= \mathcal{L}(\mathbf{b}_1/2,\ldots,\mathbf{b}_a/2,\mathbf{b}_{a+1},\ldots,\mathbf{b}_n),\\
\mathcal{L}_{\ell-2} &= \mathcal{L}(\mathbf{b}_1/4,\ldots,\mathbf{b}_{2a-n}/4,\mathbf{b}_{2a-n+1}/2,\ldots,\mathbf{b}_n/2),
\end{aligned}$$

etc. It is easy to check that this satisfies all the required properties. □

Corollary 5.10. *There is an algorithm that takes as input a tower of lattices $(\mathcal{L}_0,\ldots,\mathcal{L}_\ell)$ in $\mathbb{R}^n$ of index $2^a \geq 2^{n/2}$, $\kappa \geq 2$ (the confidence parameter), and $M = (C\kappa^4)^{\ell+1}\cdot 2^a$ vectors in $\mathcal{L}_0$ such that,*

1. *the algorithm runs in time $M\cdot\mathrm{poly}(n,\kappa,\ell)$;*
2. *if the input vectors are distributed as $D_{\mathcal{L}_0,s}$ for some s, then the output is $M\ell\exp(C_1 n - C_2\kappa)$-close to m independent samples from $D_{\mathcal{L}_\ell,2^{\ell/2}s}$ where $m \in \{0,2^{n/2}\}$ is an independent random variable; and*
3. *if $2^{\ell/2}s \geq 2\eta_{3/4}(\mathcal{L}_{\ell-1})$ and $2^{\ell/2}s \geq \sqrt{2}\eta_{3/4}(\mathcal{L}_\ell)$, then $m = 2^{n/2}$.*

Proof. Let $\mathcal{X}_0 = (\mathbf{X}_1, \ldots, \mathbf{X}_M)$ be the sequence of input vectors. For $i = 0, \ldots, \ell - 1$, the algorithm calls the procedure from Proposition 5.7 with input $\mathcal{L}_i$, $\mathcal{L}_{i+1}$, κ, and $\mathcal{X}_i$, receiving output $\mathcal{X}_{i+1}$. If $\mathcal{X}_{i+1}$ is empty, it halts and outputs nothing. Finally, the algorithm outputs the first $2^{n/2}$ vectors in $\mathcal{X}_\ell$.

The running time is clear. Define $M_i = M/(C\kappa^4)^i$. Since $M_i \geq C\kappa^5 2^a$ for $0 \leq i \leq \ell - 1$, we have by induction using Item 2 of Proposition 5.7 that for $i = 0, \ldots, \ell$, up to statistical distance $iM\exp(C_1 n - C_2\kappa)$, $\mathcal{X}_i$ is distributed like m_i independent random samples from $D_{\mathcal{L}_i, 2^{i/2}s}$ where $m_i \in \{0, M_i\}$ is an independent random variable. Here for convenience, if the algorithm aborts at some stage j, we define $\mathcal{X}_i$ for $i > j$ as the empty set. Since $M_\ell > 2^{n/2}$, Item 2 follows.

Finally, suppose $2^{\ell/2}s \geq \max\{2\eta_{3/4}(\mathcal{L}_{\ell-1}),\ \sqrt{2}\eta_{3/4}(\mathcal{L}_\ell)\}$. Since $\mathcal{L}_i/2 \subseteq \mathcal{L}_{i-2}$, we have that $\eta_{3/4}(\mathcal{L}_{i-2}) \leq \eta_{3/4}(\mathcal{L}_i)/2$. It follows that $2^{i/2}s \geq \max\{\sqrt{2}\eta_{3/4}(\mathcal{L}_i),\ \eta_{3/4}(\mathcal{L}_{i+1})\}$ for all $i = 0, \ldots, \ell - 1$. Item 3 then follows immediately from Item 3 of Proposition 5.7. □

5.3 Sampling above smoothing in time $2^{n/2}$

Theorem 5.11. *Let σ be the function that maps a lattice $\mathcal{L}$ to $\sqrt{2} \cdot \eta_{1/2}(\mathcal{L})$. Then, there is an algorithm that solves $\exp(-\Omega(\kappa))$-hDGS$_\sigma^{2^{n/2}}$ in time $2^{n/2+\mathrm{polylog}(\kappa)+o(n)}$ for any $\kappa \geq \Omega(n)$.*

Proof. We first present an algorithm that works for $\sigma(\mathcal{L}) = 2\eta_{3/4}(\mathcal{L})$ and then modify it to achieve the desired $\sigma(\mathcal{L}) = \sqrt{2} \cdot \eta_{1/2}(\mathcal{L})$. On input $\mathcal{L} \subset \mathbb{R}^n$ a lattice of rank n and $s > 0$, the algorithm behaves as follows. It first applies the algorithm from Claim 5.9 with parameters $a > n/2$ and $\ell \geq 1$ to be set in the analysis, the lattice $\mathcal{L}$, and an arbitrary choice of $\mathcal{L}'$ satisfying the properties there. It obtained a tower of lattices $(\mathcal{L}_0, \ldots, \mathcal{L}_\ell)$ of index 2^a such that $\mathcal{L}_\ell = \mathcal{L}$ and $\mathcal{L}_0 \supseteq 2^{-\lfloor \ell a/n \rfloor}\mathcal{L}$.

The algorithm then runs the sampler from Proposition 2.16 on $\mathcal{L}_0$ with parameters r (to be set in the analysis), $\hat{s} = 2^{-\ell/2}s$, and $M = (C\kappa^4)^{\ell+1}2^a$. It receives as output a sublattice $\mathcal{L}_0' \subseteq \mathcal{L}_0$ and vectors $\mathbf{X}_1, \ldots, \mathbf{X}_M \in \mathcal{L}_0'$. If $\mathcal{L}_0' \neq \mathcal{L}_0$, it outputs nothing and halts. Otherwise, it runs the procedure from Corollary 5.10 with input $(\mathcal{L}_0, \ldots, \mathcal{L}_\ell)$, κ, and $(\mathbf{X}_1, \ldots, \mathbf{X}_M)$ and outputs the result.

Let $a = \lceil n/2 + Cn/\log n \rceil$, $\ell = C\lceil \log^4 n \rceil$, and $r = Cn/\log n$. Applying Proposition 2.16 and Item 2 of Corollary 5.10, we have that the output will be distributed as $D^m_{\mathcal{L},s}$ for some $m \in \{0, 2^{n/2}\}$ up to statistical distance $\exp(-\Omega(\kappa))$, as needed. We wish to show that, if $s > 2\eta_{3/4}(\mathcal{L})$, then we have $m = 2^{n/2}$. Note that

$$\hat{s} > 2^{-\ell/2+1}\eta_{3/4}(\mathcal{L}) \geq 2^{\ell(a/n-1/2)}\eta_{3/4}(\mathcal{L}_0) \geq (Cr)^{n/r}\sqrt{n\log n} \cdot \eta_{3/4}(\mathcal{L}_0)\ .$$

Therefore, by Proposition 2.16, we have that $\mathcal{L}_0' = \mathcal{L}_0$, so that the algorithm will not halt after running the sampler from Proposition 2.16. Furthermore, since $s > 2\eta_{3/4}(\mathcal{L}) = 2\eta_{3/4}(\mathcal{L}_\ell)$, we have $2^{\ell/2}\hat{s} > \sqrt{2}\eta_{3/4}(\mathcal{L}_\ell)$, and since $\mathcal{L}_{\ell-1} \supset \mathcal{L}_\ell$, we obviously also have

$$2^{\ell/2}\hat{s} > 2\eta_{3/4}(\mathcal{L}_{\ell-1})\ . \tag{8}$$

Therefore, by Item 3 of Corollary 5.10, we have that $m = 2^{n/2}$ as needed.

Now, consider the running time. The tower of lattices can be built in polynomial time. The procedure from Proposition 2.16 runs in time $(2^{O(r)} + M) \cdot \mathrm{poly}(n)$, and the procedure from Corollary 5.10 runs in time $M \cdot \mathrm{poly}(n, \kappa, \ell)$. It follows that the running time is as claimed.

We now show how to modify the above algorithm to work for $\sigma(\mathcal{L}) = \sqrt{2} \cdot \eta_{1/2}(\mathcal{L})$. The bottleneck in the above proof is the condition in Eq. (8) needed for Item 3 of Corollary 5.10 to apply. The trouble is that we used the trivial inequality $\eta_{3/4}(\mathcal{L}_{\ell-1}) \leq \eta_{3/4}(\mathcal{L}_\ell)$ in order to show

that this holds, even though $\mathcal{L}_{\ell-1}$ is a superlattice of $\mathcal{L}_\ell$ of index greater than $2^{n/2}$, and so one might expect a gap of about $\sqrt{2}$ between these two quantities. Indeed, Lemma 5.12 below shows how to randomly choose such a superlattice $\mathcal{L}_{\ell-1}$ such that $\eta_{3/4}(\mathcal{L}_{\ell-1}) \leq \eta_{1/2}(\mathcal{L}_\ell)/\sqrt{2}$ holds with constant positive probability. So we now use the same procedure as above, except we apply the algorithm in Claim 5.9 with that choice of $\mathcal{L}_{\ell-1}$. Assuming $\mathcal{L}_{\ell-1}$ satisfies this constraint, the constraint (8) holds, whenever $s > \sqrt{2}\eta_{1/2}(\mathcal{L})$ and hence the algorithm would be successful. This almost completes the proof, except for one minor caveat: as described above, our algorithm successfully outputs $2^{n/2}$ vectors (in the "good" case of $s > \sqrt{2}\eta_{1/2}(\mathcal{L})$) only with some constant positive probability, whereas our goal is to be successful with probability $1 - \exp(-\kappa)$. This can easily be mended by repeating the algorithm κ times, each time choosing an independent $\mathcal{L}_{\ell-1}$. □

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 superlattice $\mathcal{L}' \supset \mathcal{L}$ 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 $\varepsilon' := 2\varepsilon^2 + 2^{n/2+1-a}(1+\varepsilon)$.*

Proof. The algorithm simply selects a superlattice $\mathcal{L}' \supset \mathcal{L}$ of index 2^a with $\mathcal{L}' \subseteq \mathcal{L}/2$ uniformly at random. It will be convenient to equivalently work in the dual and to instead pick $\mathcal{L}'^* \subset \mathcal{L}^*$ of index 2^a with $2\mathcal{L}^* \subseteq \mathcal{L}'^*$. In more detail, let $(\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ be a basis of the dual lattice $\mathcal{L}^*$. This defines a group isomorphism $h : \mathbb{F}_2^n \to \mathcal{L}^*/(2\mathcal{L}^*)$ given by $h(\mathbf{a}) = \sum_i a_i \mathbf{b}_i^* \bmod 2\mathcal{L}^*$. The algorithm picks a random subspace $V \subseteq \mathbb{F}_2^n$ of dimension $n-a$ and sets $\mathcal{L}'^* = h(V)$ to be the union of the cosets corresponding to the points in V. (It can do this efficiently by, e.g., taking a basis $\mathbf{v}_1, \ldots, \mathbf{v}_{n-a}$ of V, and taking the lattice generated by $(2\mathbf{b}_1^*, \ldots, 2\mathbf{b}_n^*, \mathbf{y}_1, \ldots, \mathbf{y}_{n-a})$ where $\mathbf{y}_i$ is any coset representative of $h(\mathbf{v}_i)$.) It then returns the primal lattice $\mathcal{L}'$.

It is clear that the algorithm runs in polynomial time and that $\mathcal{L}$ has index 2^a over $\mathcal{L}'$ with $\mathcal{L} \subset \mathcal{L}' \subseteq \mathcal{L}/2$ as needed. Note that all vectors in $\mathbb{F}_2^n \setminus \{\mathbf{0}\}$ have equal probability $(2^{n-a}-1)/(2^n-1)$ of being in the subspace V. Therefore, for any dual coset $\mathbf{c}^* \in \mathcal{L}^*/(2\mathcal{L}^*)$ with $\mathbf{c}^* \neq 2\mathcal{L}^*$, we have $\Pr[\mathbf{c}^* \in \mathcal{L}'^*] = (2^{n-a}-1)/(2^n-1)$. Then, assuming without loss of generality that $\eta_\varepsilon(\mathcal{L}) = 1$, we have

$$\begin{aligned}
\mathbb{E}\left[\rho_{\sqrt{2}}(\mathcal{L}'^*)\right] &= \sum_{\mathbf{c}^* \in \mathcal{L}^*/(2\mathcal{L}^*)} \Pr[\mathbf{c}^* \in \mathcal{L}'^*]\rho_{\sqrt{2}}(\mathbf{c}^*) \\
&= \rho_{\sqrt{2}}(2\mathcal{L}^*) + \frac{2^{n-a}-1}{2^n-1} \cdot \sum_{\mathbf{c}^* \in \mathcal{L}^*/(2\mathcal{L}^*) \setminus \{2\mathcal{L}^*\}} \rho_{\sqrt{2}}(\mathbf{c}^*) \\
&< 1 + \varepsilon^2 + 2^{-a}\rho_{\sqrt{2}}(\mathcal{L}^*) && \text{(Lemma 2.7)} \\
&\leq 1 + \varepsilon^2 + 2^{n/2-a}(1+\varepsilon) && \text{(Lemma 2.3) .}
\end{aligned}$$

By Markov's inequality, $\rho_{\sqrt{2}}(\mathcal{L}'^* \setminus \{\mathbf{0}\}) < 2\varepsilon^2 + 2^{n/2+1-a}(1+\varepsilon)$ with probability at least $1/2$, and the result follows. □

6 Solving $O(1)$-GapSVP in $2^{n/2+o(n)}$ time

In this section we present our GapSVP algorithm. The main idea is to approximate the smoothing parameter of $\mathcal{L}^*$ and then use Lemma 6.1 to relate it to $\lambda_1(\mathcal{L})$. To distinguish a parameter above

smoothing from a parameter below smoothing, we call the hDGS oracle with the given parameter. It is below smoothing if the oracle does not produce enough samples or if a statistical test on the output (Lemma 6.3) fails.

Lemma 6.1. *For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and $\varepsilon \in (0,1)$, if $\varepsilon > (e/\beta^2 + o(1))^{-n/2}$, we have*

$$\sqrt{\frac{\log(1/\varepsilon)}{\pi}} < \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\beta^2 n}{2\pi e}} \cdot \varepsilon^{-1/n} \cdot (1 + o(1)) \,, \tag{9}$$

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

$$\sqrt{\frac{\log(1/\varepsilon)}{\pi}} < \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\log(1/\varepsilon) + n\log\beta + o(n)}{\pi}} \,, \tag{10}$$

where $\beta := 2^{0.401}$.

As will be apparent from the proof and the remark after Lemma 4.2, Eq. (9) actually holds for all $\varepsilon \in (0,1)$.

Proof of Lemma 6.1. Throughout the proof we assume without loss of generality that $\lambda_1(\mathcal{L}) = 1$. For the lower bound in both cases, let $s \leq \sqrt{\log(1/\varepsilon)/\pi}$. Then, $\rho_{1/s}(\mathcal{L} \setminus \{\mathbf{0}\}) > e^{-\pi s^2} \geq \varepsilon$, as needed.

Let $\varepsilon > (e/\beta^2 + o(1))^{-n/2}$, and let s be the expression in the right-hand side of (9). Then, noting that $s < \sqrt{n/(2\pi)}$, by Eq. (3) in Lemma 4.2, we have

$$\rho_{1/s}(\mathcal{L} \setminus \{\mathbf{0}\}) \leq \left(\frac{\beta^2 n}{2\pi e s^2}\right)^{n/2+1} 2^{o(n)} < \varepsilon \,,$$

as needed.

Now, let $\varepsilon \leq (e/\beta^2 + o(1))^{-n/2}$, and let s be the expression in the right-hand side of (10). Then, noting that $s \geq \sqrt{n/(2\pi)}$, by Eq. (4) in Lemma 4.2, we have

$$\rho_{1/s}(\mathcal{L} \setminus \{\mathbf{0}\}) < e^{-\pi s^2} \cdot \beta^{n+o(n)} \leq \varepsilon \,,$$

as needed. □

Definition 6.2. *For a matrix $M \in \mathbb{R}^{n \times n}$, the spectral norm of M is defined as*

$$\|M\| := \sup_{\|\mathbf{x}\|=1} \|M\mathbf{x}\| \,.$$

For a symmetric matrix M (the only case that interests us), $\|M\|$ is equivalently the largest absolute value of an eigenvalue of M.

Lemma 6.3. *For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and $\varepsilon \in (0,1)$,*

$$\frac{\varepsilon}{2\pi n} \cdot \log(1/\varepsilon) \leq \left\| \frac{1}{\eta_\varepsilon(\mathcal{L})^2} \cdot \mathop{\mathbb{E}}_{\mathbf{x} \sim D_{\mathcal{L},\eta_\varepsilon(\mathcal{L})}} [\mathbf{X}\mathbf{X}^T] - \frac{1}{2\pi} \cdot I_n \right\| \leq \frac{\varepsilon}{\pi} \cdot \left(\log \frac{2(1+\varepsilon)}{\varepsilon} + 1 \right) \,,$$

where I_n is the $n \times n$ identity matrix.

Proof. For the upper bound, see [DRS14, Lemma 4.4].

For the lower bound, from the same source, we have that for any $s > 0$,

$$\frac{1}{s^2} \cdot \mathop{\mathbb{E}}_{\mathbf{X} \sim D_{\mathcal{L},s}} [\mathbf{X}\mathbf{X}^T] - \frac{1}{2\pi} \cdot I_n = s^2 \cdot \mathop{\mathbb{E}}_{\mathbf{Y} \sim D_{\mathcal{L}^*,1/s}} [\mathbf{Y}\mathbf{Y}^T] \,.$$

Note that for any positive semidefinite matrix $A \in \mathbb{R}^{n \times n}$, we have $\|A\| \geq \mathrm{Tr}(A)/n$. Therefore,

$$\begin{aligned} \eta_\varepsilon(\mathcal{L})^2 \cdot \Big\| \mathop{\mathbb{E}}_{\mathbf{Y} \sim D_{\mathcal{L}^*,1/\eta_\varepsilon(\mathcal{L})}} [\mathbf{Y}\mathbf{Y}^T] \Big\| &\geq \frac{\eta_\varepsilon(\mathcal{L})^2}{n} \cdot \mathrm{Tr}\Big(\mathop{\mathbb{E}}_{\mathbf{Y} \sim D_{\mathcal{L}^*,1/\eta_\varepsilon(\mathcal{L})}} [\mathbf{Y}\mathbf{Y}^T] \Big) \\ &\geq \frac{\eta_\varepsilon(\mathcal{L})^2}{n} \cdot \frac{\varepsilon \lambda_1(\mathcal{L}^*)^2}{1+\varepsilon} \\ &\geq \frac{\varepsilon}{2\pi n} \cdot \log(1/\varepsilon) \,, \end{aligned}$$

where we have used the lower bound in Lemma 6.1. □

We will also need a form of the matrix Chernoff bound. In particular, we use a less general version of [Ver12, Theorem 5.29].

Lemma 6.4 (Matrix Chernoff bound)**.** *Let $A_1, \ldots, A_N$ be independent and identically distributed random symmetric matrices in $\mathbb{R}^{n \times n}$ with $\|A_i\| \leq a$ and expectation μ. Then, for any $t \in (0, a)$,*

$$\Pr\Big[\Big\| \frac{1}{N} \sum A_i - \mu \Big\| \geq t \Big] \leq 2n \exp(-CNt^2/a^2) \,.$$

Theorem 6.5. *For any $\varepsilon \in [2^{-n/2}, 1/e]$, there is a probabilistic polynomial-time reduction from γ-GapSVP to $\frac{1}{4}$-hDGS$^m_{\sqrt{2}\eta_{1/2}}$ where $m := n^5/\varepsilon^2$ and*

$$\gamma := \sqrt{\frac{\beta^2 n + o(n)}{e \log(1/\varepsilon)}} \cdot \Big(1 + \frac{2 \log n}{\log(1/\varepsilon)}\Big) \,,$$

where $\beta := 2^{0.401}$. The reduction preserves dimension, makes a single call to the hDGS oracle, and runs in time $m \cdot \mathrm{poly}(n)$.

Proof. On input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and $d > 0$, the reduction calls the hDGS oracle with input $\mathcal{L}^*$ and parameter $s > 0$ to be set in the analysis. If the oracle outputs fewer than m vectors, the reduction immediately outputs yes (i.e., the reduction guesses that $\lambda_1(\mathcal{L}) < d$). Otherwise, it receives as output $\mathbf{X}_1, \ldots, \mathbf{X}_m \in \mathcal{L}^*$. Let $\Sigma := \frac{1}{m} \cdot \sum \mathbf{X}_i \mathbf{X}_i^T$ be the sample covariance. If

$$\Big\| \frac{1}{s^2} \cdot \Sigma - \frac{1}{2\pi} \cdot I_n \Big\| < \frac{\varepsilon}{10n} \cdot \log(1/\varepsilon) \,,$$

the reduction outputs no (i.e., the reduction guesses that $\lambda_1(\mathcal{L}) \geq \gamma \cdot d$). Otherwise, it outputs yes.

The running time is clear. Let

$$s := \sqrt{\frac{\log(1/\varepsilon)}{\pi}} \cdot \frac{1}{d} \,.$$

Suppose $\lambda_1(\mathcal{L}) < d$. Then, by the lower bound in Lemma 6.1, we have $s < \eta_\varepsilon(\mathcal{L}^*)$. By the definition of hDGS, we have that the output of the oracle is statistically close to $D^{m'}_{\mathcal{L}^*,s}$ for some

independent random variable $0 \leq m' \leq m$. So, we assume that the oracle outputs exactly this distribution, introducing statistical distance at most $1/4$. If $m' < m$, then the reduction correctly outputs yes. Conditioning on $m' = m$ and using Lemma 6.3, we have

$$\left\| \frac{1}{s^2} \cdot \mathbb{E}[\Sigma] - \frac{1}{2\pi} \cdot I_n \right\| \geq \frac{\varepsilon}{2\pi n} \cdot \log(1/\varepsilon) \ ,$$

where we have used the fact that $\varepsilon \log(1/\varepsilon)$ is monotonically increasing for $\varepsilon \leq 1/e$. So, in order to show that the reduction will output yes, it suffices to show that Σ is concentrated around its mean. By Lemma 2.3 and union bound, we can assume that $\|\mathbf{X}_i\| \leq 100\sqrt{n}s$, introducing only negligible statistical distance. Assuming that this is the case, we can apply Lemma 6.4 with $a = 100^2 n$ and $t = \varepsilon/(1000n)$, and we have that

$$\Pr\left[\left\| \frac{1}{s^2}\Sigma - \frac{1}{s^2}\mathbb{E}[\Sigma] \right\| \geq t \right] \leq 2n \exp(-Cmt^2/a^2) \leq \exp(-Cn) \ ,$$

where we have used the fact that $mt^2/a^2 \geq Cn$. It follows that the reduction correctly outputs yes with all but negligible probability.

Now, suppose $\lambda_1(\mathcal{L}) \geq \gamma \cdot d$. Then, by Eq. (9) of Lemma 6.1 with ε there taken to be $1/2$, we have $s > \sqrt{2}\eta_{1/2}(\mathcal{L}^*)$. In this regime, by the definition of hDGS, we have that the output of the oracle is within statistical distance $1/4$ of $D^m_{\mathcal{L}^*,s}$. So, we can assume that the output is exactly $D^m_{\mathcal{L}^*,s}$, introducing statistical distance at most $1/4$. Applying Lemma 6.1 again, we have

$$s > (1 + 2\log n/\log(1/\varepsilon))\eta_\varepsilon(\mathcal{L}^*) > \eta_{\varepsilon/n^2}(\mathcal{L}^*) \ ,$$

where we have used Lemma 2.7, the fact that $\varepsilon > 2^{-n/2}$ and the observation that Eq. (9) applies for all $\varepsilon \in (0,1)$. And, applying Lemma 6.3, we have that

$$\left\| \frac{1}{s^2} \cdot \mathbb{E}[\Sigma] - \frac{1}{2\pi} \cdot I_n \right\| < \frac{\varepsilon}{\pi n^2} \cdot \left(\log \frac{2(n^2+\varepsilon)}{\varepsilon} + 1 \right) < \frac{\varepsilon}{20n} \cdot \log(1/\varepsilon) \ ,$$

for sufficiently large n (where we have used the fact that the upper bound in Lemma 6.3 is monotonically increasing). Finally, applying Lemma 6.4 as above shows that the oracle correctly outputs no with all but negligible probability. □

Corollary 6.6. *There is a randomized algorithm that solves γ-GapSVP for $\gamma := 1.93 + o(1)$ in time $2^{n/2+o(n)}$.*

Proof. Combine the algorithm from Theorem 5.11 with the reduction from Theorem 6.5 with $\varepsilon = 2^{-n/4}$. □

7 Other applications

7.1 Approximating CVP in $2^{n+o(n)}$ time

Theorem 7.1. *For $\gamma = 1.97$, there is a reduction from γ-CVP to $\frac{1}{2}$-DGS$_{2^{n/2}}$. The reduction makes $O(n^2)$ calls to the DGS oracle on an $(n+1)$-dimensional lattice and runs in time $2^{n/2} \cdot \mathrm{poly}(n)$.*

Proof. On input $\mathcal{L} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) \subset \mathbb{R}^n$ and $\mathbf{t} \in \mathbb{R}^n$, the reduction behaves as follows. It first uses Babai's nearest plane algorithm [Bab86] to approximate the distance to the lattice $\mathrm{dist}(\mathbf{t}, \mathcal{L})$, receiving as output $\tilde{d}$. Fix $\delta := 1/n$. Then for $j = 1, \ldots, 10n^2$, let $s_j = \tilde{d}/(1+\delta)^j$, and let $\mathcal{L}_j$ be the $(n+1)$-dimensional lattice generated by $(\mathbf{b}_i, 0)$ for $i \in [n]$, and the additional basis vector $(-\mathbf{t}, s_j)$. The reduction calls the DGS oracle on $\mathcal{L}_j$ with parameter s_j, and let $\mathbf{x}_j$ be the shortest vector among the returned vectors whose last coordinate is s_j. Finally, the reduction outputs the first n coordinates of $\mathbf{x}_j - (-\mathbf{t}, s_j)$ where j is such that $\mathbf{x}_j$ is shortest.

The running time of the algorithm is clear. As was shown in [Bab86], we have $d \leq \tilde{d} \leq 2^{n/2} d$. Thus, there exists a j such that $\alpha d/\sqrt{n} \leq s_j \leq (1+\delta)\alpha d/\sqrt{n}$, where $\alpha := \sqrt{2\pi/\log 2}$. Let $A = \{\mathbf{y} \in \mathcal{L}_j : \langle \mathbf{y}, \mathbf{e}_{n+1} \rangle = s_j\}$ be the set of vectors from which we choose $\mathbf{x}_j$. We note that it suffices to show that a sample from $D_{\mathcal{L}_j, s_j}$ will land in A and have length at most γd with probability at least $2^{-n/2-O(1)}$. Indeed, if this is the case, then the algorithm will find a vector in A of length at most γd with constant probability, and its output will be a γ-approximate closest vector.

We first consider the probability that a vector lands in A, $\rho_{s_j}(A)/\rho_{s_j}(\mathcal{L}_j)$. For the denominator, using the fact that $\rho_s(\mathcal{L}) \geq \rho_s(\mathcal{L} + \mathbf{w})$ for any $\mathbf{w}$, we have

$$\begin{aligned}
\rho_{s_j}(\mathcal{L}_j) &= \sum_{k=-\infty}^{\infty} \rho_{s_j}(k s_j) \rho_{s_j}(\mathcal{L} + k\mathbf{t}) \\
&\leq \rho_{s_j}(\mathcal{L}) \sum_{k=-\infty}^{\infty} e^{-\pi k^2} \\
&\leq 2\rho_{s_j}(\mathcal{L}) \,.
\end{aligned}$$

Turning to the numerator,

$$\rho_{s_j}(A) \geq e^{-\pi(d^2 + s_j^2)/s_j^2} \cdot \rho_{s_j}(\mathcal{L}) \geq e^{-\pi(n/\alpha^2 + 1)} \cdot \rho_{s_j}(\mathcal{L}) \,.$$

Thus, we have

$$\Pr_{\mathbf{X} \sim D_{\mathcal{L}_j, s_j}} [\mathbf{X} \in A] \geq e^{-\pi n/\alpha^2}/100 \,. \tag{11}$$

Set $t = \gamma/((1+\delta)\alpha)$. Recall from Lemma 2.4 that

$$\Pr_{\mathbf{X} \sim D_{\mathcal{L}_j, s_j}} \left[\|\mathbf{X}\| > s_j t\sqrt{n}\right] \leq \left(\sqrt{2\pi e t^2} \exp(-\pi t^2)\right)^n \,. \tag{12}$$

Then, combining (11) and (12), plugging in the values for α, t, and γ, and assuming n is sufficiently large, gives

$$\begin{aligned}
\Pr_{\mathbf{X} \sim D_{\mathcal{L}_j, s_j}} \left[\mathbf{X} \in A,\ \|\mathbf{X}\| \leq s_j t\sqrt{n}\right] &\geq e^{-\pi n/\alpha^2}/100 - (2\pi e t^2)^{n/2} \cdot e^{-\pi t^2 n} \\
&\geq 2^{-n/2 - O(1)} \,,
\end{aligned}$$

where we have used the fact that $e^{-\pi/\alpha^2} = 1/\sqrt{2}$ and $\sqrt{2\pi e t^2} \cdot e^{-\pi t^2} < 1/\sqrt{2}$. The result follows from the fact that $s_j t\sqrt{n} \leq (1+\delta)\alpha t d = \gamma d$. □

We note that the above proof actually yields a more general statement. In particular, for any $t > 1/\sqrt{2\pi}$, there is a reduction from γ-CVP to ε-DGS^M where

$$\gamma = \sqrt{\frac{2\pi t^2}{2\pi t^2 - \log(2\pi t^2) - 1}}\,,$$

and

$$M \approx \exp(\pi t^2 n/\gamma^2) = \exp(\pi n t^2)/(2\pi e t^2)^{n/2}\,.$$

We recover Theorem 7.1 by setting $t \approx 0.654$.

Corollary 7.2. *There is a randomized algorithm that solves* 1.97*-CVP in time* $2^{n+o(n)}$.

Proof. Combine the reduction from Theorem 7.1 with the algorithm from Theorem 3.7. □

7.2 Solving $O(1)$-BDD in $2^{n/2+o(n)}$ time

Lyubashevsky and Micciancio show a polynomial-time reduction from $\frac{1}{2\gamma}$-BDD to γ-GapSVP [LM09]. By combining this with Theorem 6.5, we immediately get a solution to α-BDD for $\alpha \approx 1/4$. But, we can improve this to $\alpha \approx .422$ by using the following (slightly modified) theorem from [DRS14] that shows how to solve a variant of BDD directly using discrete Gaussian samples.

Theorem 7.3 ([DRS14, Theorem 3.1])**.** *For any* $\varepsilon \in (0, 1/200)$*, let*

$$\phi(\mathcal{L}) := \frac{\sqrt{\log(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)}\,.$$

Then, there exists a reduction from CVP$^\phi$ to $\frac{1}{2}$-DGS$^m_{\eta_\varepsilon}$ where $m = O(n\log(1/\varepsilon)/\sqrt{\varepsilon})$ and CVP$^\phi$ 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 $m \cdot \mathrm{poly}(n)$.

Corollary 7.4. *There is a randomized algorithm that solves α-BDD in time $2^{n/2+o(n)}$ for $\alpha := .422 - o(1)$.*

Proof. Let $\varepsilon := 2^{-n}$, and let $\phi(\mathcal{L}) := \sqrt{\log(1/\varepsilon)/\pi - o(1)}/(2\eta_\varepsilon(\mathcal{L}^*))$ as above. By Eq. (10) of Lemma 6.1, any algorithm that solves CVP$^\phi$ is also a solution to α-BDD with

$$\alpha := \frac{1}{2}\cdot\sqrt{\frac{\log(1/\varepsilon)}{\log(1/\varepsilon) + n\log\beta + o(n)}} > .422 - o(1)\,.$$

Applying Theorem 7.3 gives a reduction from α-BDD to $\frac{1}{2}$-DGS$^m_{\eta_\varepsilon}$ with $m = O(n\log(1/\varepsilon)/\sqrt{\varepsilon}) = 2^{n/2+o(n)}$ that runs in time $m \cdot \mathrm{poly}(n)$. Finally, we note that Lemma 6.1 implies that, $\sqrt{2}\eta_{1/2}(\mathcal{L}) < \eta_\varepsilon(\mathcal{L})$ for sufficiently large n. Therefore, Theorem 5.11 gives a solution to $\exp(-\Omega(n))$-DGS$^m_\varepsilon$ with the desired running time. □

7.3 Approximating SIVP in $2^{n/2+o(n)}$ time

We use the following lemma, which is a slight variant of [Reg09, Lemma 3.17] combined with Lemma 2.12 there.

Lemma 7.5. *There is a polynomial-time reduction from γ-SIVP to $\frac{1}{2}$-DGS$_{2\eta_\varepsilon}$ where $\gamma := O(\sqrt{n \log n})$ and $\varepsilon := 1/10$.*

Corollary 7.6. *There is a randomized algorithm that solves γ-SIVP in time $2^{n/2+o(n)}$ where $\gamma := O(\sqrt{n \log n})$.*

Proof. Combine the reduction from Lemma 7.5 with the algorithm from Theorem 5.11. □

References

[ADS15] Divesh Aggarwal, Daniel Dadush, and Noah Stephens-Davidowitz. Solving the Closest Vector Problem in 2^n time— the discrete Gaussian strikes again!, 2015. `http://arxiv.org/abs/1409.8063`.

[Ajt04] Miklós Ajtai. Generating hard instances of lattice problems. In *Complexity of computations and proofs*, volume 13 of *Quad. Mat.*, pages 1–32. Dept. Math., Seconda Univ. Napoli, Caserta, 2004. Preliminary version in STOC'96.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

[AKS02] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *CCC*, pages 41–45, 2002.

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

[Ban93] W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(4):625–635, 1993.

[BGJ14] Anja Becker, Nicolas Gama, and Antoine Joux. A sieve algorithm based on overlattices. *LMS Journal of Computation and Mathematics*, 17(A):49–70, 2014.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.

[BN09] Johannes Blömer and Stefanie Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theoret. Comput. Sci.*, 410(18):1648–1665, 2009.

[Bri85] Ernest F. Brickell. Breaking iterated knapsacks. In *Advances in cryptology (Santa Barbara, Calif., 1984)*, volume 196 of *Lecture Notes in Comput. Sci.*, pages 342–358. Springer, Berlin, 1985.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106. IEEE, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, pages 1–12, 2014.

[CDLP13] Kai-Min Chung, Daniel Dadush, Feng-Hao Liu, and Chris Peikert. On the lattice smoothing parameter problem. In *IEEE 28th Conference on Computational Complexity*, pages 230–241, 2013.

[dB89] R. de Buda. Some optimal codes have structure. *Selected Areas in Communications, IEEE Journal on*, 7(6):893–899, Aug 1989.

[Did12] Did (http://math.stackexchange.com/users/6179/did). Understanding what $\sqrt{p}$ means for an event of probability p. Mathematics Stack Exchange, 2012. `http://math.stackexchange.com/q/182821` (version: 2012-08-15).

[DRS14] Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. On the closest vector problem with a distance guarantee. In *IEEE 29th Conference on Computational Complexity*, pages 98–109, 2014. Full version available at `http://arxiv.org/abs/1409.8063`.

[FT87] András Frank and Éva Tardos. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC'09—Proceedings of the 2009 ACM International Symposium on Theory of Computing*, pages 169–178. ACM, New York, 2009.

[Gly87] Peter W. Glynn. Upper bounds on Poisson tail probabilities. *Oper. Res. Lett.*, 6(1):9–14, 1987.

[GMSS99] O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55 – 61, 1999.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[Hel85] Bettina Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theoret. Comput. Sci.*, 41(2-3):125–139 (1986), 1985.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[HR12] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. *Theory of Computing*, 8(23):513–531, 2012. Preliminary version in STOC'07.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm (extended abstract). In *Advances in cryptology—CRYPTO 2007*, volume 4622 of *Lecture Notes in Comput. Sci.*, pages 170–186. Springer, Berlin, 2007.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

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

[KL78] G. A. Kabatjanskiĭ and V. I. Levenšteĭn. Bounds for packings on the sphere and in space. *Problemy Peredači Informacii*, 14(1):3–25, 1978.

[Kle00] Philip Klein. Finding the closest lattice vector when it's unusually close. In *SODA*, pages 937–941, 2000.

[Laa14] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. *IACR Cryptology ePrint Archive*, 2014:744, 2014.

[Len83] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *Advances in Cryptology-CRYPTO 2009*, pages 577–594. Springer, 2009.

[LO85] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *J. Assoc. Comput. Mach.*, 32(1):229–246, 1985.

[LWXZ11] Mingjie Liu, Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. Shortest lattice vectors in the presence of gaps. *IACR Cryptology ePrint Archive*, 2011:139, 2011.

[Mic01] Daniele Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, March 2001. Preliminary version in FOCS 1998.

[Mic08] Daniele Micciancio. Efficient reductions among lattice problems. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 84–93. ACM, New York, 2008.

[Mic14] Daniele Micciancio. Private communication, 2014.

[MP05] Elchanan Mossel and Yuval Peres. New coins from old: Computing with unknown bias. *Combinatorica*, 25(6):707–724, 2005.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In *CRYPTO*, volume 8042 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2013.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302 (electronic), 2007.

[Mum07] David Mumford. *Tata lectures on theta. I.* Modern Birkhäuser Classics. Birkhäuser Boston, Inc., Boston, MA, 2007. With the collaboration of C. Musili, M. Nori, E. Previato and M. Stillman, Reprint of the 1983 edition.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA*, pages 1468–1480, 2010.

[MV13] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. *SIAM Journal on Computing*, 42(3):1364–1391, 2013.

[MW15] Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *SODA*, 2015.

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Math. Cryptol.*, 2(2):181–207, 2008.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, pages 333–342. ACM, 2009.

[Pei10] Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In *Advances in cryptology—CRYPTO 2010*, volume 6223 of *Lecture Notes in Comput. Sci.*, pages 80–97. Springer, Berlin, 2010.

[PS09] Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *IACR Cryptology ePrint Archive*, 2009:605, 2009.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):Art. 34, 40, 2009.

[RS15] Oded Regev and Noah Stephens-Davidowitz. An inequality for Gaussians on lattices, 2015.

[Sch87] C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(23):201 – 224, 1987.

[SFS09] Naftali Sommer, Meir Feder, and Ofir Shalvi. Finding the closest lattice point by iterative slicing. *SIAM J. Discrete Math.*, 23(2):715–731, 2009.

[Sha84] Adi Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE Trans. Inform. Theory*, 30(5):699–704, 1984.

[Ste14] Damien Stehlé. Private communication, 2014.

[Ver12] Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. In Y.C. Eldar and G. Kutyniok, editors, *Compressed Sensing: Theory and Applications*, pages 210–268. Cambridge Univ Press, 2012.

[WLTB11] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 1–9, New York, NY, USA, 2011. ACM.

[ZPH14] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the shortest vector problem. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography – SAC 2013*, Lecture Notes in Computer Science, pages 29–47. Springer Berlin Heidelberg, 2014.