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

Extended Abstract[*]

Divesh Aggarwal
Department of Computer Science, EPFL.

Daniel Dadush
Centrum Wiskunde & Informatica, Amsterdam.
dadush@cwi.nl

Oded Regev
Courant Institute of Mathematical Sciences, New York University.

Noah Stephens-Davidowitz
Courant Institute of Mathematical Sciences, New York University.

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.

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.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*

General Terms

Algorithms

[*]The full version of this paper is available at `http://arxiv.org/abs/1412.7994` [1]

Keywords

Lattices; Discrete Gaussian; Shortest Vector Problem

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 [25], coding over Gaussian channels [12], cryptanalysis [45, 10, 24], combinatorial optimization and integer programming [26, 21, 14]. 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 [15, 8, 9], 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 [2, 32, 42, 7].

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 a factor of $n^{c/\log\log n}$ for some constant $c > 0$ under reasonable complexity assumptions [28, 22, 18]. From the perspective of polynomial-time algorithms, the celebrated LLL basis reduction gives a $2^{O(n)}$ approximation algorithm for SVP [25], and Schnorr's block reduction algorithm [44], with subsequent refinements [3, 34], gives a $r^{n/r}$ approximation in $2^{O(r)}\text{poly}(n)$ 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 [21] and refined by many others [19, 17, 35], 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 poly(n) space [17].

The next landmark algorithm, developed by Ajtai, Kumar, and Sivakumar [3] (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 [4, 33, 41, 27] and heuristic [38, 48, 49, 6, 23], where the fastest provable sieving algorithm [41] for exact SVP requires $2^{2.465n+o(n)}$ time and $2^{1.233n+o(n)}$ space. It was observed by [27, 30, 47] 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 [20].

The most recent breakthrough, due to Micciancio and Voulgaris [34] (henceforth MV) and built upon the approach of Sommer, Feder, and Shalvi [46], 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 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 an algorithm for the Discrete Gaussian Sampling Problem (DGS), which will directly imply our SVP algorithm. Below, we give an informal description of this result. (See Section 3 for the details.)

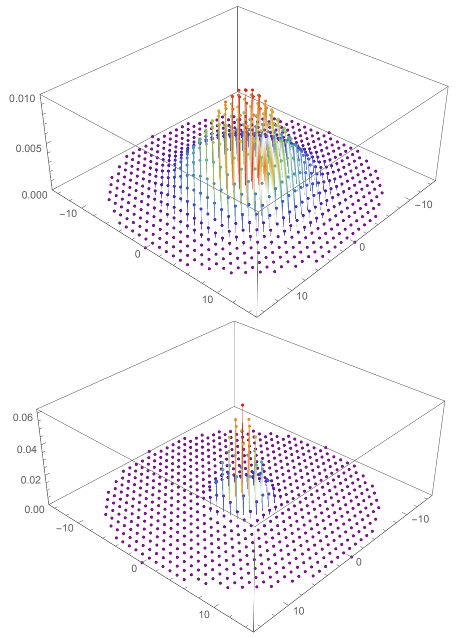

Figure 1: The discrete Gaussian distribution on $\mathbb{Z}^2$ with parameter $s = 10$ (top) and $s = 4$ (bottom)

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 (GENERAL DGS, 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 2 (SMOOTH DGS, 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.) 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 [32, 16, 42, 39, 31]. Gentry, Peikert, and Vaikuntanathan first showed how to solve DGS in polynomial time for large parameters. 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 [32, 42, 39, 31], and as a core subroutine for instantiating certain cryptographic primitives [16]. 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 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 [38, 48, 49, 23].

1.2 Proof overview

We now include a high-level description of our proofs, first that of Theorem 1 and then that of the more refined Theorem 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 [16]. 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 ([40]) 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.

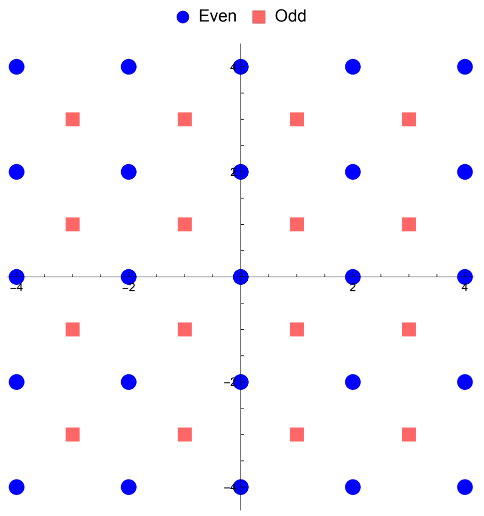

Figure 2: The lattice $\mathcal{L}^* = \{(z_1, z_2) \in \mathbb{Z}^2 : z_1 = z_2 \bmod 2\}$. Note that when we rotate this lattice by 45°, $(z_1, z_2) \mapsto (z_1 + z_2, z_1 - z_2)/\sqrt{2}$, we obtain $(\sqrt{2}\mathbb{Z})^2$. Our combiner creates samples from $D_{\mathcal{L}^*,s}$ and then outputs $(z_1 + z_2)/2$. The resulting distribution is exactly $D_{\mathbb{Z},s/\sqrt{2}}$.

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 $\mathbf{c} \bmod 2\mathcal{L}$, and then taking the average of each pair. So, the algorithm effectively samples from some distribution over the $2n$-dimensional lattice of pairs of vectors from $\mathcal{L}$ that are in the same coset mod $2\mathcal{L}$,

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

Furthermore, when we take the average of a vector pair from $\mathcal{L}^*$, $(\mathbf{X}_1, \mathbf{X}_2) \mapsto (\mathbf{X}_1 + \mathbf{X}_2)/2$, we can view this as a projection of the $2n$-dimensional *rotation* $(\mathbf{X}_1, \mathbf{X}_2) \mapsto (\mathbf{X}_1 + \mathbf{X}_2, \mathbf{X}_1 - \mathbf{X}_2)/\sqrt{2}$, scaled down by a factor of $\sqrt{2}$. Since the projected rotation of a Gaussian is again a Gaussian, it follows that the average of a vector pair sampled from $D_{\mathcal{L}^*,s}$ has *exactly* the distribution that we want, $D_{\mathcal{L},s/\sqrt{2}}$. This fact has a straightforward proof, yet we have not seen this observation before. (This fact is closely related to Riemann’s theta relations, as described in [37, Chapter 1, Section 5]. See Lemma 3 for a short algebraic proof and Figure 2 for an illustration of the rotation applied to the one-dimensional lattice $\mathcal{L} = \mathbb{Z}$.) However, note that if the combiner just greedily paired as many vectors from each coset as possible, it would *not* yield the correct distribution. In particular, the probability that a sample from $D_{\mathcal{L}^*,s}$ 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.

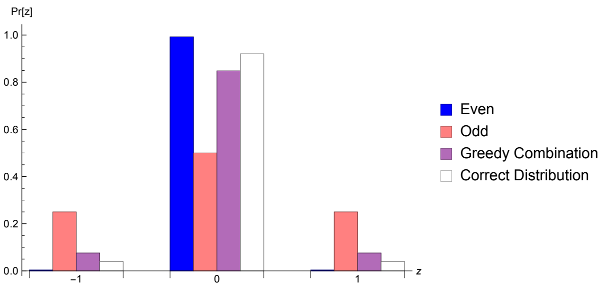

Figure 3: The distribution of averages of pairs of integers sampled from $D_{\mathbb{Z},\sqrt{2}}$ resulting from taking (1) only even pairs; (2) only odd pairs; (3) even and odd pairs with “greedy” weights proportional to $\rho_{\sqrt{2}}(2\mathbb{Z})$ and $\rho_{\sqrt{2}}(2\mathbb{Z}+1)$ respectively;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

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

[1]One can likely also obtain a $2^{O(n)}$-time algorithm for DGS above the smoothing parameter by instantiating the oracles in [31].

one!), making the loss factor of the square sampler approximately $\sum p_i^2 / \max p_i$.

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 1 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 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 [40] 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 the full version for the technical details [1]. 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 [36] and discussed in [13]) that allows one to flip a coin with probability $\sqrt{p}$ given black-box access to a coin with unknown probability p. 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 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 algorithm: 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 [43].

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

[2]While the purely algebraic proof of this fact is quite simple (see the proof of Corollary 1), we do not yet have good intuitive understanding of it. Indeed, we have found ourselves referring to it as the "magic cancellation."

[3]We note that Becker et al. [6] also use a tower of lattices in their heuristic algorithm.

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 $2^{-0.465n-o(n)}$. 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 [20]. (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 [41, 33]).

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 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." 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. With this, we obtain an $O(1)$-approximation algorithm for the smoothing parameter that runs in $2^{n/2+o(n)}$ time.

Further applications.

Based on an embedding trick of Kannan [21] and standard concentration bounds on the discrete Gaussian, we show how to use our sampler to solve 1.97-approximate CVP in time $2^{n+o(n)}$. The reductions of [29] show that this yields the same approximation factor and running time for almost all lattice problems.

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

1.3 Conclusions and open problems

We have devised a novel randomized method for solving lattice problems more efficiently based on DGS. We believe that our techniques may lead to further algorithmic progress for lattice problems, as well as provide new tools within cryptography and cryptanalysis.

Our work raises many questions and potential avenues for improvement. Firstly, we suspect that the algorithm from Theorem 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 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.s

Secondly, one may broadly ask what other uses a discrete Gaussian sampler that works below smoothing has. As such an algorithm has not existed previously, this question has received very little attention until now. Indeed, solving exact SVP and approximate CVP are in some sense the most obvious applications, but we suspect it to have many applications.

A more general 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 [21, 19, 17, 35].

Organization

In this extended abstract, we sketch the proof of the general DGS algorithm (see 3) and the SVP algorithm (see Theorem 4 and Corollary 2).

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 $\operatorname{dist}(\mathbf{t}, \mathcal{L})$ to denote the distance between $\mathbf{t}$ and the lattice, $\min_{\mathbf{y} \in \mathcal{L}}(\|\mathbf{t} - \mathbf{y}\|)$.

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 1. 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., [5]). 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 [5].

LEMMA 1 ([5, 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 ([11, 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. 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 1. *For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and $\varepsilon \in (0,1)$, if $s \geq \eta_\varepsilon(\mathcal{L})$, then for all $\mathbf{t} \in \mathbb{R}^n$,*

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

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

Definition 4. 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 $\|\mathbf{y}\| = \lambda_1(\mathcal{L})$.

2.5 Starting the sampler

In the sequel, we show a Gaussian "combiner" that takes many samples from $D_{\mathcal{L},s}$ and returns samples from $D_{\mathcal{L},s/\sqrt{2}}$. In order to turn this into a sampling algorithm, we will need to "initialize" it by obtaining Gaussian samples with some high parameter σ, using, for example, the algorithm of Gentry, Peikert, and Vaikuntanathan [16] or the modest strengthening in [7]. In particular, we would like to simply use such algorithms to solve DGS_σ for $\sigma < 2^{\mathrm{poly}(n)} \cdot \lambda_1(\mathcal{L})$, so that we can then apply our combiner $\mathrm{poly}(n)$ times to solve DGS_s for $s \approx \lambda_1(\mathcal{L})$ and therefore solve SVP. Unfortunately, this does not work. Such algorithms only allow us to sample from $D_{\mathcal{L},\sigma}$ if *all* of the Gram-Schmidt vectors of some input basis are smaller than $\approx \sigma$. We cannot hope to achieve this even for $\sigma = 2^{\mathrm{poly}} \cdot \lambda_1(\mathcal{L})$. 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. The proof is in the full version.

PROPOSITION 1. *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 $\mathcal{L}' = \mathcal{L}$.

3. SAMPLING FROM THE DISCRETE GAUSSIAN

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

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

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

$$\Pr_{(\mathbf{X}_1,\mathbf{X}_2) \sim D_{\mathcal{L},s}^2}[\mathbf{X}_1 + \mathbf{X}_2 \in 2\mathcal{L}] = \rho_s(\mathcal{L})^{-2} \sum_{\mathbf{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(\mathbf{c})^2 \; ,$$

we get for any $\mathbf{y} \in \mathcal{L}$,

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

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. $\square$

In the full version, we show how to use rejection sampling to sample from the conditional distribution in Lemma 3 to obtain the following result [1].

PROPOSITION 2. *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 $M\exp(C_1n-C_2\kappa)$ 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 $M\cdot\mathrm{poly}(n,\log\kappa)$.

By calling the algorithm from Proposition 2 repeatedly, we obtain a general discrete Gaussian combiner.

COROLLARY 1. *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 $\ell M\exp(C_1n-C_2\kappa)$ 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 2 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 1 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 $iM\exp(C_1n-C_2\kappa)$ 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 2. We can therefore apply the proposition and the induction hypothesis, and obtain that (up to statistical distance at most $(i+1)M\exp(C_1n-C_2\kappa)$), $\mathcal{X}_{i+1}$ 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 $M_\ell\geq 2^{n/2}\psi(\ell)/\psi(\ell+1)\geq 2^{n/2}$. $\square$

3.2 A general discrete Gaussian sampler

THEOREM 3. *There is an algorithm that solves $2^{-\Omega(\kappa)}$-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 1 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 1 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 1, and by Corollary 1, 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. $\square$

4. SOLVING SVP IN $2^{N+O(N)}$ TIME

In the full version, we prove the following, using the bound on the "kissing number" from [20].

PROPOSITION 3. *Let $\mathcal{L}\subset\mathbb{R}^n$ be a lattice of rank at least one. Let*

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

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

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

Using this, we obtain our SVP algorithm.

THEOREM 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 a polynomial-time algorithm to obtain d such that $\lambda_1(\mathcal{L}) < d < 2^n \cdot \lambda_1(\mathcal{L})$. 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. Note that there exists some i such that $\hat{s} \le s_i \le 1.01\hat{s}$ 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 3). We assume that the output of D is exactly $D_{\mathcal{L},s_i}^{2^{n/2}}$ when called on s_i, incurring statistical distance at most $1/2$. For such i, by Lemma 1 we have

$$\Pr_{\mathbf{X}\sim D_{\mathcal{L},s_i}}[\|\mathbf{X}\| = \lambda_1(\mathcal{L})] \ge 1.01^{-n} \cdot \Pr_{\mathbf{X}\sim D_{\mathcal{L},\hat{s}}}[\|\mathbf{X}\| = \lambda_1(\mathcal{L})]$$
$$\ge 1.4^{-n-o(n)} .$$

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 $1 - \exp(-\Omega(n))$. $\square$

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

PROOF. Combine the reduction from Theorem 4 with the algorithm from Theorem 3. $\square$

5. ACKNOWLEDGMENTS

This material is based upon work supported by the Simons Collaboration on Algorithms and Geometry and 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.

6. REFERENCES

[1] D. Aggarwal, D. Dadush, O. Regev, and N. Stephens-Davidowitz. Solving the Shortest Vector Problem in 2^n time via discrete Gaussian sampling. http://arxiv.org/abs/1412.7994.

[2] M. 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.

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

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

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

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

[7] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.

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

[9] Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, pages 1–12, 2014.

[10] E. 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.

[11] D. Dadush, O. Regev, and N. 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.

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

[13] 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).

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

[15] C. 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.

[16] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[17] G. Hanrot and D. 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.

[18] I. Haviv and O. 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.

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

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

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

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

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

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

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

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

[27] M. Liu, X. Wang, G. Xu, and X. Zheng. Shortest lattice vectors in the presence of gaps. *IACR Cryptology ePrint Archive*, 2011:139, 2011.

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

[29] D. 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.

[30] D. Micciancio. Private communication, 2014.

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

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

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

[34] D. Micciancio and P. 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.

[35] D. Micciancio and M. Walter. Fast lattice point enumeration with minimal overhead. In *SODA*, 2015.

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

[37] D. 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.

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

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

[40] C. 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.

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

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

[43] O. Regev and N. Stephens-Davidowitz. An inequality for Gaussians on lattices, 2015.

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

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

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

[47] D. Stehlé. Private communication, 2014.

[48] X. Wang, M. Liu, C. Tian, and J. 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.

[49] F. Zhang, Y. Pan, and G. Hu. A three-level sieve algorithm for the shortest vector problem. In T. Lange, K. Lauter, and P. Lisonek, editors, *Selected Areas in Cryptography – SAC 2013*, Lecture Notes in Computer Science, pages 29–47. Springer Berlin Heidelberg, 2014.