

Faster Provable Lattice Sieving with Spherical Codes

Divesh Aggarwal *

Aditya Morolia †

Noah Stephens-Davidowitz ‡

October 7, 2026

Abstract

We study the computational problem μ -SVP, in which the goal is to find a μ -approximate shortest non-zero vector in a lattice $\mathcal{L}$, for constant approximation factors $\mu > 1$. Prior to this work, there was a large gap between the fastest algorithms for this problem whose correctness had been proven and the fastest heuristic algorithms, whose correctness has not been proven (but which seem to work well in practice).

We construct a novel sieving algorithm that combines the key tool used in the fastest heuristic sieving algorithms (faster nearest neighbor subroutines) together with the perturbation argument used in sieving algorithms with proven correctness. The result is an algorithm whose correctness we prove and whose running time depends on a simple geometric quantity: the maximal size of certain spherical codes.

Under a reasonable assumption about the maximal size of spherical codes, our algorithm's running time is roughly $2^{0.3066n}$ for large constants μ , which is quite close to the best heuristic running time of roughly $(3/2)^{n/2} \approx 2^{0.2925n}$ and much faster than both the previous best known algorithm with proven correctness, which runs in time roughly $2^{0.802n}$ [Liu, Wang, Xu, and Zheng, 2011] and the very recent concurrent and independent work showing a $2^{n/2}$ -time algorithm [Hhan, 2026]. (However, the algorithm of Hhan also solves μ -SVP for $\mu = 1$, while our algorithm only works for large constant μ .) Using the best *proven* upper bound on the size of such spherical codes (which is almost certainly quite loose), we obtain a running time of roughly $2^{0.6078n}$, which is much better than the previous best work but notably slower than the concurrent work of Hhan.

*National University of Singapore. Email: divesh@comp.nus.edu.sg

†Centre for Quantum Technologies, Singapore. Email: morolia@u.nus.edu

‡Cornell University. Email: noahsd@gmail.com

Contents

1	Introduction	1
1.1	Our results	2
1.2	Our techniques	3
1.2.1	Sieving algorithms	4
1.2.2	Choosing N and the relationship with the kissing number	4
1.2.3	Perturbations	5
1.2.4	Speeding up the sieving procedure and the [BDGL16] algorithm	6
1.2.5	Our algorithm and its analysis	8
1.3	On the maximal size $A(\omega)$ of a spherical code and our geometric assumption	10
1.4	Concurrent work	11
2	Preliminaries	11
2.1	Lattices	12
2.2	A bit of geometry	13
2.3	Good spherical codes	18
3	Faster lattice sieving without heuristics	19
3.1	Sieving with good spherical codes	19
3.2	Our sieving algorithm	21
3.3	The quantum variant of our algorithm	27

1 Introduction

A lattice $\mathcal{L} \subset \mathbb{R}^n$ is the set of all integer linear combinations

$$\mathcal{L} := \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_n) := \{z_1 \mathbf{b}_1 + \dots + z_n \mathbf{b}_n : z_i \in \mathbb{Z}\}$$

of linearly independent basis vectors $\mathbf{b}_1, \dots, \mathbf{b}_n$.

The most important computational problem associated with lattices is the *Shortest Vector Problem* (SVP), in which the input is a basis $\mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{Q}^n$ for a lattice $\mathcal{L}$, and the goal is to output a non-zero lattice vector $\mathbf{v} \in \mathcal{L}_{\neq \mathbf{0}}$ with $\|\mathbf{v}\|$ as short as possible, i.e., such that

$$\|\mathbf{v}\| = \lambda_1(\mathcal{L}) := \min_{\mathbf{y} \in \mathcal{L}_{\neq \mathbf{0}}} \|\mathbf{y}\|.$$

(Throughout the paper, $\|\cdot\|$ means the ℓ_2 norm.)

The μ -approximate Shortest Vector Problem (μ -SVP) is the relaxation of this problem in which the output can be any non-zero lattice vector $\mathbf{v} \in \mathcal{L}_{\neq \mathbf{0}}$ with $\|\mathbf{v}\| \leq \mu \lambda_1(\mathcal{L})$, for some approximation factor $\mu \geq 1$.

It is difficult to overstate the importance of this problem in modern cryptography. In particular, the celebrated work of Ajtai [Ajt04] and Regev [Reg09] showed how to build cryptography under the assumption that (the decision version of) μ -SVP is hard *in the worst case* for some $\mu \leq \text{poly}(n)$. Since then, there have been innumerable works using these ideas to construct different lattice-based cryptographic schemes, including constructions of powerful cryptographic primitives like fully homomorphic encryption (FHE) [Gen09, BV11, BV14], identity-based encryption (IBE) [GPV08], attribute-based encryption (ABE) [GVW13, BGG⁺14, Tsa19], functional encryption [AFV11, GKP⁺13, ALS16], and many more. (See [Pei16] for a survey.) Perhaps most importantly, lattice-based cryptographic constructions are the preferred cryptographic schemes for *post-quantum cryptography* (that is, classical cryptography that is secure against adversaries with quantum computers). Indeed, the American National Institute of Standards and Technology has standardized a lattice-based encryption scheme [ABD⁺21, Nat24b] and a lattice-based signature [BDK⁺21, Nat24a] with the intention of replacing the widespread use of quantum-insecure cryptography, e.g., in the ubiquitous TLS protocol.

The security of all of these lattice-based constructions depends heavily on the presumed hardness of μ -SVP for different approximation factors μ . Perhaps the most important regime is when μ is a constant. In fact, the best algorithms for μ -SVP for nearly all values of μ (including those relevant to cryptography) use the best algorithms for μ' -SVP for constant μ' as a key subprocedure, via a highly non-trivial reduction [Sch87, GN08]. So, the running time of μ -SVP for nearly all values of μ is governed by the running time when μ is a constant. Therefore, in what follows, we sometimes make little distinction between exact SVP and μ -SVP for constant approximation factors μ .

The fastest known heuristic algorithm for SVP is the beautiful *heuristic* sieving algorithm of Becker, Ducas, Gama, and Laarhoven [BDGL16] (building on much prior work on heuristic sieving [NV08, MV10, BGJ15, Laa15, LdW15]). This algorithm runs in time $(3/2)^{n/2+o(n)} \approx 2^{0.2925n}$ and uses space $(4/3)^{n/2+o(n)} \approx 2^{0.2075n}$. [BDGL16] provide a compelling heuristic argument (and experimental evidence) that it does in fact solve SVP, though they are unable to provide a proof.¹ We will say much more about how this algorithm works in Section 1.2.4. For now, we note that the security analysis of lattice-based

¹Because the only known proof of correctness of this algorithm is heuristic, it is not entirely clear what approximation factor μ the algorithm actually achieves. It is plausible that the algorithm solves the hardest case of $\mu = 1$, but it seems plausible that it actually only achieves a small approximation factor greater than one.

cryptography in practice relies heavily on the assumption that the [BDGL16] sieving algorithm is essentially optimal—i.e., that no algorithm can solve SVP in time significantly faster than $2^{0.2925n}$.² For example, if, say, a $2^{0.15n}$ -time algorithm were found for this problem, some of the schemes standardized by NIST would be rendered effectively broken in practice.

One might therefore quite reasonably worry that the [BDGL16] sieving algorithm is not actually optimal. Indeed, the fact that we do not actually know how to prove that this algorithm solves SVP suggests that our understanding is lacking—both of the algorithm and of SVP itself. In fact, though we think that the [BDGL16] sieving algorithm solves SVP in time roughly $2^{0.2925n}$, the best known algorithms with provable correctness run considerably slower. Until quite recently, the best algorithm that was proven to solve $O(1)$ -SVP was the $2^{0.802n+o(n)}$ -time sieving algorithm of Liu, Wang, Xu, and Zheng [LWXZ11], building on the original sieving algorithm of [AKS01]. (We will also have much more to say about the [LWXZ11] sieving algorithm later. See also [WLW15, AUV19].) *Very* recently, a new line of works (concurrent and independent of the present work) has culminated in a $2^{n/2+o(n)}$ -time algorithm that is proven to solve *exact* SVP [GFH26, Hha26b, Kim26, Hha26a], building on work of [ADRS15, AS18b]. However, even this quite recent line of works on improved algorithms with proofs of correctness is still quite far from the best running time of heuristic algorithms of $(3/2)^{n/2+o(n)} \approx 2^{0.2925n}$. Again, in the context of the security of lattice-based cryptography in practice, the gap between running in time $2^{n/2}$ and $2^{0.2925n}$ is quite significant.

So, one of the long-standing goals in this field is to close this large gap between the best running time of an algorithm whose correctness we can prove and the best heuristic algorithm.

1.1 Our results

Our main result is to *almost* close this gap, under a natural and purely geometric assumption. Specifically, we show the following.

Theorem 1.1 (Informal; see Corollaries 3.8 and 3.9). *Under a natural assumption about the maximal size of spherical codes, there is an algorithm that solves μ -SVP for any constant $\mu > 1$ in time $2^{.3066n+\varepsilon_\mu n+o(n)}$, where ε_μ is a constant depending on μ with $\lim_{\mu \rightarrow \infty} \varepsilon_\mu = 0$.*

Unconditionally, there is an algorithm that solves μ -SVP for any constant $\mu > 1$ in time $2^{.6078n+\varepsilon_\mu n+o(n)}$.

Notice that our running time under this geometric assumption comes *quite* close to the running time of $(3/2)^{n/2+o(n)} \approx 2^{0.2925n}$ achieved by the best known *heuristic* algorithm for large constant μ .

More generally, as we will explain in detail in Section 1.2.5, we obtain an algorithm for μ -SVP whose running time depends in a relatively natural way on the maximal size of spherical codes $A(\theta)$ (with two different angular distances θ_1 and θ_2). See Theorem 3.7. The first running time mentioned in Theorem 1.1 is simply what one obtains by assuming that the largest known spherical codes in high dimensions cannot be improved upon by more than a $2^{o(n)}$ factor. Alternatively, our algorithm either runs in this time *or* the vectors generated by the algorithm will themselves form a better spherical code than the current best known.³ (We say much more about this assumption and this win-win property in Section 1.3.)

²There does exist some weak evidence to support this assumption. For example, [KL21] prove a matching lower bound for a restricted class of “sieving-like algorithms.” And, [AS18a] prove a $2^{\Omega(n)}$ lower bound for SVP under complexity-theoretic assumptions.

³We acknowledge that the reader might think that our geometric assumption is not very different from the heuristic assumptions made in works like [BDGL16]. However, as we will explain in detail below, the heuristic assumptions in

If one uses the celebrated bounds on spherical codes due to Kabatjanskiĭ and Levenšteĭn [KL78], one can prove that our algorithm solves μ -SVP in time $2^{0.6078n+\varepsilon_\mu n+o(n)}$, i.e., the second running time quoted in Theorem 1.1. This running time is already much better than the $2^{0.802n+\varepsilon_\mu n+o(n)}$ running time of the algorithm for μ -SVP from [LWXZ11, WLW15, AUV19], which was the best known algorithm for this problem until very recently. It is significantly worse than the running time of $2^{n/2+o(n)}$ achieved quite recently by independent concurrent work in [Hha26a] (using very different techniques). However, we note that the bound in [KL78] is very unlikely to be the correct answer.

Finally, we note that our algorithm extends in a few different directions.

First, like any SVP algorithm for the ℓ_2 norm, our algorithm extends generically to an algorithm for μ -SVP in any ℓ_p norm with the same running time (with a different choice of ε_μ), using the result of [ACK⁺21] (which builds heavily on [EV20]). Less obviously, one can obtain an algorithm for μ -SVP and even μ -CVP (a harder problem) in *any* norm by observing that the beautiful techniques in [RV22] convert any sieving algorithm for μ -SVP in the ℓ_2 norm into such an algorithm.⁴ As this is not the topic of the present work, we do not attempt to prove this formally.

Second, we present a quantum variant of our algorithm with a better running time.

Theorem 1.2 (Informal; see Corollaries 3.12 and 3.13). *Under a natural assumption about the maximal size of spherical codes, there is a quantum algorithm that solves μ -SVP for any constant $\mu > 1$ in time $2^{0.3n+\varepsilon_\mu n+o(n)}$, where ε_μ is a constant depending on μ with $\lim_{\mu \rightarrow \infty} \varepsilon_\mu = 0$.*

Unconditionally, there is a quantum algorithm that solves μ -SVP for any constant $\mu > 1$ in time $2^{0.5832n+\varepsilon_\mu n+o(n)}$.

This running time compares reasonably well with the best known running time of roughly $2^{0.2563n}$ for a *heuristic* quantum algorithm for SVP [BCSS23]. Prior to this work, the best known quantum algorithm that was *proven* to solve SVP was simply the $2^{n/2+o(n)}$ -time classical algorithm mentioned above [Hha26a].

1.2 Our techniques

Here, we describe the many ingredients that together make up our algorithm. We note that though there are many ingredients and they might seem rather technical, the final algorithm is remarkably simple. This is because these ideas fit together surprisingly nicely in the end. (Admittedly, before coming to our final algorithm and analysis, the authors tried many different ways of fitting these ideas together that simply did not work. Indeed, we have been trying to prove a result of this form for over six years.)

The reader who is familiar with the literature on sieving might therefore skip to Section 1.2.5.

[BDGL16] are crucially assumptions about the behavior of the algorithm itself. Specifically, like all sieving algorithms, the [BDGL16] algorithm generates a long list of lattice vectors via quite a complicated randomized process. The resulting distribution of vectors is quite hard to analyze—so hard that we do not even know how to rule out that this distribution is supported entirely on $\mathbf{0}$. The heuristic assumptions in [BDGL16] say that the resulting lattice vectors “behave like uniformly random independent vectors on a sphere.” In contrast, our purely geometric assumption is simply about spherical codes—not about the behavior of our algorithm. And, this win-win property in which our algorithm either solves μ -SVP or outputs an explicit remarkably good spherical code has no analogue in heuristic sieving.

⁴The result in [RV22] is stated using the algorithm from [AUV19].

1.2.1 Sieving algorithms

Our algorithm, like that of [BDGL16] and [LWXZ11, WLW15, AUV19] is a *sieving* algorithm. Such algorithms were introduced in the celebrated work of Ajtai, Kumar, and Sivakumar [AKS01] to give the first $2^{O(n)}$ -time algorithm for SVP.

The basic idea of a sieving algorithm is as follows. The algorithm first samples a very large number N of lattice vectors $\mathbf{y}_1^{(0)}, \dots, \mathbf{y}_N^{(0)} \in \mathcal{L}$ from some distribution of lattice vectors such that $\|\mathbf{y}_i^{(0)}\| \leq R$ for some large but not-too-large radius $R > 0$. (E.g., one might imagine simply taking random small integer linear combinations of the basis vectors, in which case R will be bounded in terms of the length of a basis vector.) If N is large enough, then there must exist many pairs of vectors $\mathbf{y}_i^{(0)}, \mathbf{y}_j^{(0)}$ with $i \neq j$ such that $\|\mathbf{y}_i^{(0)} - \mathbf{y}_j^{(0)}\| < R$. The algorithm finds such pairs and generates a new list $\mathbf{y}_1^{(1)}, \dots, \mathbf{y}_N^{(1)} \in \mathcal{L}$, where $\mathbf{y}_k^{(1)} = \mathbf{y}_i^{(0)} - \mathbf{y}_j^{(0)}$ for such a pair $\mathbf{y}_i^{(0)}, \mathbf{y}_j^{(0)}$. In particular, notice that $\mathbf{y}_i^{(1)}$ is a lattice vector with $\|\mathbf{y}_i^{(1)}\| < R$. The algorithm then repeats this “sieving” process on the $\mathbf{y}_k^{(1)}$, finding pairs of $\mathbf{y}_k^{(1)}$ that are even closer together and generating a new list $\mathbf{y}_1^{(2)}, \dots, \mathbf{y}_N^{(2)}$ of even shorter lattice vectors by taking differences of such pairs. After doing this many times, generating lists $\mathbf{y}_1^{(t)}, \dots, \mathbf{y}_N^{(t)}$ of progressively shorter lattice vectors, the algorithm simply outputs the shortest non-zero vector in its final list.⁵

Sieving algorithms differ in their choice of N ; in how they sample the original list of vectors $\mathbf{y}_i^{(0)}$; and in how they implement the *sieving procedure*, which is the process that finds the pairs $\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}$ whose differences form the next list of vectors. In particular, the space complexity of the resulting algorithm is roughly N . The running time of the resulting algorithm is typically some relatively simple function of N , where the particular function depends on the sieving procedure. In particular, prior to this work, all of the sieving algorithms with proven correctness ran in time roughly N^2 because they effectively performed a brute-force search over the pairs $(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)})$.

1.2.2 Choosing N and the relationship with the kissing number

For the reasons described above, we would like to take N to be as small as possible in our sieving algorithm. Of course, if we take N to be *too* small, then we might encounter a situation in which there simply do not exist pairs $\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}$ with $i \neq j$ whose difference is short. In this case, the sieving procedure will simply fail. In fact, we need there to exist at least N such pairs in order for the sieving procedure to succeed on the next step, since we need N vectors in the next step. (In reality, sieving algorithms typically find *slightly* fewer vectors in the subsequent steps, but this distinction is irrelevant at this high level.) So, we really want to take N to be large enough to guarantee that most vectors $\mathbf{y}_i^{(0)}$ are likely to have a nearby vector $\mathbf{y}_j^{(0)}$.

Heuristic sieving algorithms like [BDGL16] simply assume that the vectors $\mathbf{y}_i^{(t)}$ “behave like” independent uniformly random vectors on the sphere of the appropriate radius. One can in fact choose the *initial* set of vectors $\mathbf{y}_i^{(0)}$ in such a way that we can prove that this holds. However, after running many iterations of the sieving procedure, we do not know how to prove almost anything about the resulting distribution of

⁵We have made numerous simplifying assumptions here. For example, we assumed that the pairs are chosen simply to have $\|\mathbf{y}_i^{(0)} - \mathbf{y}_j^{(0)}\| < R$, whereas in a true sieving algorithm one should take $\|\mathbf{y}_i^{(0)} - \mathbf{y}_j^{(0)}\| \leq (1 - \varepsilon)R$ for some small positive $\varepsilon > 0$. We have also assumed that the number of vectors in each list is always N , whereas in typical sieving algorithms this number decreases slightly after each sieving step. And, we have left out additional details that are relevant to some sieving algorithms but not ours.

the $\mathbf{y}_i^{(t)}$. (For example, we do not even know how to rule out the possibility that this distribution is highly concentrated on a single vector.) So, we do not know how to prove that some version of this assumption is true. However, once we make this assumption, it is well known that the number of vectors N needed to guarantee that most vectors $\mathbf{y}_i^{(0)}$ will be close to another vector $\mathbf{y}_j^{(0)}$ is roughly $(4/3)^{n/2}$ [NV08]. So, heuristic sieving algorithms like [BDGL16] typically take $N \approx (4/3)^{n/2} \approx 2^{0.2075n}$, and this is typically their space complexity.

In contrast, prior work on sieving algorithms with proven correctness took N to be *much* larger. In particular, since we have essentially no understanding of the distribution of the vectors $\mathbf{y}_i^{(t)}$, we must worry that these vectors form a worst-case configuration of vectors in the ball. This concern leads naturally to the *kissing number* τ_n , which is the maximal size of a subset $T \subseteq S^{n-1}$ of the unit sphere S^{n-1} such that all distinct $\mathbf{x}, \mathbf{y} \in T$ have $\|\mathbf{x} - \mathbf{y}\| \geq 1$.⁶ Notice that if our points $\mathbf{y}_i^{(t)}$ happen to form such a set T , then our sieving procedure will certainly fail. On the other hand, it is not hard to show that if we take N a bit larger than τ_n , then there will exist N pairs of nearby points among the $\mathbf{y}_i^{(t)}$'s.

Therefore, all prior works on sieving algorithms with proven correctness take N to be larger than the best known upper bound on τ_n , so that we can provably say that $N \geq \tau_n$. Indeed, Kabatjanskiĭ and Levenšteĭn famously proved that $\tau_n \leq 2^{0.401n+o(n)}$ [KL78], and prior work relied on this bound and therefore took $N > 2^{0.401n}$. Indeed, the fastest known sieving algorithm with proven correctness before this work required $2^{0.401n+\varepsilon_\mu n}$ space and $2^{0.802n+\varepsilon_\mu n}$ time [LWXZ11, WLW15, AUV19] for precisely this reason.

One rather minor contribution of our work is to simply leave τ_n as a variable. Indeed, it seems *very* likely that the Kabatjanskiĭ and Levenšteĭn bound is quite loose. (In fact, just very recently the bound has been improved slightly, though the result has not yet been peer reviewed [Ope26].) It is even plausible that τ_n is simply $(4/3)^{n/2+o(n)}$, i.e. the same as the space complexity used by heuristic algorithms (up to lower-order terms). One can then see (up to small complications that we ignore here) that the algorithms in [LWXZ11, WLW15, AUV19] can actually take N just slightly larger than τ_n , so that their space complexity is $2^{\varepsilon_\mu n} \tau_n$ and their running time is $2^{\varepsilon_\mu n} \tau_n^2$. In particular, under the assumption that $\tau_n = (4/3)^{n/2+o(n)}$, this yields a running time of roughly $2^{0.415n}$. See Section 1.3 for more discussion.

1.2.3 Perturbations

As we have already alluded to, the major difficulty with sieving algorithms is that we have very little understanding of the behavior of the distribution of the vectors $\mathbf{y}_i^{(t)}$ generated by the algorithm. In fact, we have so little understanding of this distribution that we are unable to rule out the possibility that all of the vectors $\mathbf{y}_1^{(t)}, \dots, \mathbf{y}_N^{(t)}$ will simultaneously be $\mathbf{0}$ with high probability. This is, of course, fatal, because SVP requires us to output a short *non-zero* lattice vector.⁷

Heuristic analysis of sieving algorithms simply assumes that the vectors $\mathbf{y}_i^{(t)}$ are often non-zero. (Indeed, they assume more than this.) All known sieving algorithms with provable correctness (including our

⁶Given how we presented things here, we should work with the unit ball and not the unit sphere. But, the kissing number is defined in terms of the sphere, and in the sequel we work with the sphere. It is also known that the distinction makes little difference.

⁷For some additional intuition for why this might be a problem, notice that at the t th step, the sieving procedure generates vectors of length R_t for some decreasing sequence of radii R_t . And, notice that if $R_t < \lambda_1(\mathcal{L})$, then by definition, all of the $\mathbf{y}_i^{(t)}$ *must* be zero. For this reason, heuristic sieving algorithms naturally terminate slightly before we have $R_t < \lambda_1(\mathcal{L})$. But, one might naturally worry that this phenomenon of only seeing the zero vector occurs a bit earlier, in which case the algorithm will fail.

own), use *perturbations*, which were introduced in the original sieving algorithm of Ajtai, Kumar, and Sivakumar [AKS01]. The idea here is elegant, though a bit counterintuitive. Instead of running the sieving procedure on the lattice vectors $\mathbf{y}_i^{(t)}$ themselves, one runs the sieving procedure on *perturbed lattice vectors* $\mathbf{p}_i^{(t)} := \mathbf{y}_i^{(t)} + \mathbf{x}_i^{(t)}$ where $\mathbf{x}_i^{(t)}$ is some small noise. So, in effect, the algorithm finds short *perturbed* $\mathbf{p}_i^{(t)}$ lattice vectors, rather than finding short lattice vectors directly. One then argues formally that the sieving procedure “cannot tell the difference” between the case when $(\mathbf{y}_i^{(t)}, \mathbf{x}_i^{(t)}) = (\mathbf{y}, \mathbf{x})$ for some lattice vector $\mathbf{y}$ and some noise vector $\mathbf{x}$ and the case when $(\mathbf{y}_i^{(t)}, \mathbf{x}_i^{(t)}) = (\mathbf{y} + \mathbf{v}, \mathbf{x} - \mathbf{v})$ for some shortest non-zero lattice vector $\mathbf{v}$. In either case, the perturbed vector $\mathbf{p}_i^{(t)} = \mathbf{y}_i^{(t)} + \mathbf{x}_i^{(t)}$ is the same. If we then output the shortest non-zero vector among the actual lattice vectors $\mathbf{y}_i^{(t)}$, we can argue that even if the sieving procedure is adversarial, it simply could not produce vectors that are heavily concentrated on $\mathbf{0}$, since from its perspective “the case when $\mathbf{y}_i^{(t)} = \mathbf{0}$ looks the same as the case when $\mathbf{y}_i^{(t)} = \mathbf{v}$.”

The resulting argument is quite delicate and rather notoriously difficult. One needs to choose the noise $\mathbf{x}_i^{(0)}$ carefully.⁸ More significantly, one needs to carefully design the sieving procedure so that the noise propagates through the algorithm appropriately. At a technical level, one carefully designs the sieving procedure so that each $(\mathbf{y}_k^{(t+1)}, \mathbf{x}_k^{(t+1)})$ can be written as $(\mathbf{y}_i^{(t)} - \mathbf{y}_j^{(t)}, \mathbf{x}_i^{(t)})$, where $\mathbf{y}_j^{(t)}$ comes from a special set of “centers” and $\mathbf{y}_i^{(t)}$ is a non-center that occurs in precisely one such pairing. Then, one needs to make the above informal argument precise. See Section 3.2.

And, this argument is lossy. Indeed, the original [AKS01] sieving algorithm had a running time that was very large because they were effectively only able to use this perturbation-based argument to show that a 2^{-Cn} fraction of the vectors would be non-zero, for some large constant C . (For those familiar with the perturbation argument, what we are really talking about here is the fraction of “good vectors.”) And, for exact SVP, the fastest known algorithm using this approach still runs in time $2^{2.25n+o(n)}$ [PS09, Muk21], specifically because of the loss incurred in the analysis of the perturbation step.

However, the line of work in [LWXZ11, WLW15, AUV19] showed that one could make the perturbation argument much tighter if one is willing to solve μ -SVP for large constant μ . With this relaxation, they were effectively able to show that at least a $2^{-\varepsilon_\mu n}$ fraction of the vectors would be non-zero, for some constant ε_μ depending on μ , with $\varepsilon_\mu \rightarrow 0$ as $\mu \rightarrow \infty$.⁹ In effect, this line of work showed that the perturbation argument “is essentially for free”, in the sense that it only costs us a lower-order term in the running time if we are willing to solve μ -SVP rather than exact SVP. (Solving μ -SVP for constant μ is nearly as good as solving exact SVP for nearly all applications, so this is a very profitable tradeoff.)

1.2.4 Speeding up the sieving procedure and the [BDGL16] algorithm

Summarizing what we have said above, prior work [LWXZ11, WLW15, AUV19] has shown how to argue about perturbations with very little loss, obtaining an algorithm that provably solves μ -SVP in time $2^{\varepsilon_\mu n} \tau_n^2$ and space $2^{\varepsilon_\mu n} \tau_n$, where ε_μ is a lower-order term as μ increases. If one assumes quite reasonably that $\tau_n = (4/3)^{n/2+o(n)}$, then these algorithms already match the space of the best heuristic algorithms for SVP (though this had not been stated explicitly in prior work). But, these algorithms fall quite short

⁸In fact, one typically chooses the noise $\mathbf{x}_i^{(0)}$ first and chooses the $\mathbf{y}_i^{(0)}$ as a deterministic function of $\mathbf{x}_i^{(0)}$, as we do in Section 3.2.

⁹At a technical level, this line of work took the length $\|\mathbf{x}_i^{(0)}\|$ of the perturbations to be $C\lambda_1(\mathcal{L})$ for some constant C that increases as μ increases, whereas the work on sieving algorithms for exact SVP took this length to be much less than $\lambda_1(\mathcal{L})$. The result is that $\|\mathbf{y}_i^{(t)}\| = \|\mathbf{p}_i^{(t)} - \mathbf{x}_i^{(t)}\|$ **Adi: gpt made me change this + to -** will tend to be much larger, but we lose much less in our argument that the $\mathbf{y}_i^{(t)}$ are non-zero.

of the best known running time, which is $(3/2)^{n/2+o(n)} \approx 2^{0.2925n} \ll (4/3)^n \approx 2^{0.415n}$.

The reason for this remaining gap in running time is simply that the sieving procedures in all known algorithms whose correctness is proven perform essentially a brute-force search over the roughly N^2 pairs among the N vectors, in order to find pairs that are close together.¹⁰ In contrast, in the heuristic sieving literature, a series of beautiful works [BGJ14, Laa15, LdW15] showed how to speed up the sieving procedure in order to find sufficiently many close pairs of vectors in time much better than N^2 . This line of work culminated in [BDGL16], which showed the fastest known *heuristic* sieving procedure by showing the fastest known algorithm for finding many close pairs among randomly chosen points on a sphere.

We now describe the [BDGL16] sieving procedure in detail. (Admittedly, our description differs greatly from that of [BDGL16], and we here focus on details that are relevant to our own algorithm.)

Indeed, the [BDGL16] sieving procedure works by first sampling a spherical code $C \subset S^{n-1}$ from a cleverly chosen distribution (formally, a random product code, as in Definition 2.11). One can associate with each codeword $\mathbf{c} \in S^{n-1}$ a subset $H_{\mathbf{c}} \subseteq \mathbb{R}^n$ corresponding to all points whose angle with $\mathbf{c}$ is below some threshold ϕ . Notice that these buckets can overlap. The [BDGL16] algorithm then determines for each vector in the list the set of buckets $H_{\mathbf{c}}$ that contain the vector and performs brute-force search among triples $(i, j, \mathbf{c})$ such that $\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)} \in H_{\mathbf{c}}$ to find a close pair $\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}$.

The final running time of their algorithm therefore ends up being essentially

$$(\text{Time to generate } C) + \sum_{i=1}^N (\text{Time to compute } \{\mathbf{c} : \mathbf{y}_i^{(t)} \in H_{\mathbf{c}}\}) + |\{(i, j, \mathbf{c}) : \mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)} \in H_{\mathbf{c}}\}|.$$

[BDGL16] choose their code cleverly enough that the time to sample the code is not significant, and so that the time to find all of the $H_{\mathbf{c}}$ containing $\mathbf{y}$ is

$$2^{o(n)} \cdot |\{\mathbf{c} : \mathbf{y} \in H_{\mathbf{c}}\}|.$$

In expectation, this is simply $|C| \cdot p(\phi)$, where

$$p(\phi) := \Pr_{\mathbf{c} \sim S^{n-1}} [\text{the angle between } \mathbf{c} \text{ and } \mathbf{y} \text{ is at most } \phi] \approx \sin(\phi)^n$$

is exactly the probability that $\mathbf{y} \in H_{\mathbf{c}}$ for a single uniformly random $\mathbf{c} \in S^{n-1}$ (and arbitrary non-zero vector $\mathbf{y} \in \mathbb{R}^n$). (The marginal distribution of a codeword in C is in fact uniform over the sphere.) Ignoring lower-order terms, this yields an expected running time of

$$\mathbb{E}[\mathcal{T}_{\text{BDGL}}] \approx |C| \cdot \sum_{i=1}^N \left(p(\phi) + \mathbb{E}_{\mathbf{c}}[|\{j : \mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)} \in H_{\mathbf{c}}\}|] \right).$$

Finally, to guarantee that the sieving procedure finds enough close pairs, [BDGL16] take their code C to be just large enough that nearly every close pair is likely to land in at least one bucket together.¹¹ This amounts to taking $|C| \approx 1/p(\pi/3, \phi)$, where

$$p(\theta, \phi) := \Pr_{\mathbf{c} \sim S^{n-1}} [\text{the angle between } \mathbf{c} \text{ and } \mathbf{y} \text{ and that between } \mathbf{c} \text{ and } \mathbf{y}' \text{ is at most } \phi],$$

¹⁰More accurately, these sieving procedures adaptively choose a sublist of at most τ_n of the vectors to be *centers*, and use a brute force search to find a center that is close to every non-center in the list. So, the running time is roughly $N\tau_n \approx N^2$.

¹¹In our algorithm, we actually take many independently sampled copies of the code from [BDGL16] to guarantee that with high probability *every* close pair will land in at least one bucket together, but we ignore this distinction in the introduction.

where $\mathbf{y}$ and $\mathbf{y}'$ are arbitrary vectors with angle θ . ($\pi/3$ is exactly the angle that guarantees that $\|\mathbf{y} - \mathbf{y}'\| \leq \max\{\|\mathbf{y}\|, \|\mathbf{y}'\|\}$, which is why it arises here.) Notice that this is what one would require if the codewords were uniformly random and independent, and [BDGL16] show that this many codewords is also sufficient for their code.

Plugging this in, we see that

$$\mathbb{E}[\mathcal{T}_{\text{BDGL}}] \approx \frac{1}{p(\pi/3, \phi)} \cdot \sum_{i=1}^N \left(p(\phi) + \mathbb{E}_{\mathbf{c} \sim C} [|\{j : \mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)} \in H_{\mathbf{c}}\}|] \right), \quad (1.1)$$

Note that, though this formula is a bit complicated, each term arises relatively naturally.

Up until now, we have not used any heuristics. (This is of course intentional because we wish to compare to our own heuristic-free running time.) One can finish the analysis of the [BDGL16] sieving algorithm by applying three heuristics. First, we can simply assume heuristically that the $\mathbf{y}_i^{(t)}$ are not too concentrated on $\mathbf{0}$. Second, as we explained above, we can assume that it suffices to take $N = (4/3)^{n/2+o(n)}$ in order for the number of close pairs to be at least N in each step, so that the sieving procedure continues to produce lists of progressively shorter non-zero vectors. These two assumptions together imply that the algorithm does find a non-zero vector with length not much larger than $\lambda_1(\mathcal{L})$. Third, we can assume that $\mathbb{E}_{\mathbf{c}} [|\{j : \mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)} \in H_{\mathbf{c}}\}|] \approx Np(\phi)^2$, which is what one would expect if the $\mathbf{y}_j^{(t)}$ were sampled independently from the uniform distribution over a sphere.

After computing $p(\phi)$ and $p(\theta, \phi)$ as in Lemma 2.9 and setting $\phi := \pi/3$ (which effectively equates the two summands in parentheses below), we can plug in to Equation (1.1) to obtain

$$\mathbb{E}[T_{\text{BDGL}}] \approx \frac{N}{p(\pi/3, \phi)} \cdot (p(\phi) + Np(\phi)^2) \approx (3/2)^{n/2} N \cdot ((4/3)^{-n/2} + (4/3)^{-n} N) \approx (3/2)^{n/2}, \quad (1.2)$$

as needed.

1.2.5 Our algorithm and its analysis

Finally, we come to our algorithm, as presented in Algorithms 1 and 3. At a very high level, our algorithm is simply sieving with perturbations in the style of [LWXZ11, WLW15, AUV19] (using ideas developed in [AKS01, PS09]) but with the sieving process replaced by something like the sieving process from [BDGL16]. And, as we mentioned above, the final algorithm is remarkably simple in this context. (Though, again, there are many different variants of every step of this algorithm, and we found many approaches that did not work.)

Indeed, the algorithm starts by generating perturbations and the initial lattice vectors $\mathbf{y}_i^{(0)}$ in the same way as [LWXZ11, WLW15, AUV19], as presented in Algorithm 2. Then, as in [AKS01], the algorithm instantiates an empty list T of “centers” and iterates through the current list of perturbed lattice vectors $\mathbf{p}_i^{(t)}$ as follows. First, it checks if there is a center that is close to $\mathbf{p}_i^{(t)}$. If there is, $\mathbf{y}_i^{(t)}$ is paired with this center as we described above. In particular, we define $(\mathbf{y}_k^{(t+1)}, \mathbf{p}_k^{(t+1)}) := (\mathbf{y}_i^{(t)} - \mathbf{y}_a^{(t)}, \mathbf{p}_i^{(t)} - \mathbf{y}_a^{(t)})$ where a is the index of the nearby center. Notice that this guarantees that our sieving procedure has the property described in Section 1.2.3, which eventually allows us (in Section 3.2) to prove that the final distribution is not too concentrated on $\mathbf{0}$.

However, instead of using brute-force search to look for nearby centers to $\mathbf{p}_i^{(t)}$, we use the approach of [BDGL16] that we described in Section 1.2.4. Specifically, we sample a code $C \subset S^{n-1}$ (essentially

the same code as the one used in [BDGL16]) at the beginning of the procedure. When we consider a perturbed lattice vector $\mathbf{p}_i^{(t)}$, we first find all codewords $\mathbf{c}$ such that $\mathbf{p}_i^{(t)}$ is in the corresponding bucket $H_{\mathbf{c}}$. We then only look *inside these buckets* for the first close center that we find.

Just like with the analysis of the [BDGL16] sieving algorithm described in Section 1.2.4, the resulting sieving procedure finds all nearby centers as long as we take our code C to have large enough size, $|C| \gtrsim 1/p(\pi/3, \phi)$. This means that the actual input and output behavior of our algorithm is essentially identical to the behavior of other sieving algorithms with large perturbations, such as those in [LWXZ11, WLW15, AUV19]. So, once we have established this, we are able to prove that our algorithm will find an approximate shortest non-zero vector as long as N is a bit larger than τ_n using a proof that largely follows prior work (though it is still quite delicate; see Section 3.2).

It remains, of course, to bound the running time of the algorithm. To that end, we first notice that this particular “center-based” sieving procedure only ever considers pairs in which at least one vector is a center. And, it stops once it finds a single such pair. So, rather than considering all tuples $(i, j, \mathbf{c})$ with $\mathbf{p}_i^{(t)}, \mathbf{p}_j^{(t)} \in H_{\mathbf{c}}$, we only need to consider tuples where $j \in T$ is a center *and* $\mathbf{p}_i^{(t)}, \mathbf{p}_j^{(t)}$ are *not* close (because we only check at most one close pair, and this single check is insignificant in the analysis). So, the running time of our algorithm is actually better than the running time presented in Equation (1.1). Specifically, our running time is

$$\begin{aligned} \mathcal{T}^* &\approx \frac{1}{p(\pi/3, \phi)} \cdot \sum_{i=1}^N \left(p(\phi) + \mathbb{E}_{\mathbf{c} \sim C} [|\{j \in T : \mathbf{p}_i^{(t)}, \mathbf{p}_j^{(t)} \in H_{\mathbf{c}}, \alpha_{i,j} \geq \pi/3\}|] \right) \\ &= \frac{1}{p(\pi/3, \phi)} \cdot \sum_{i=1}^N \left(p(\phi) + \sum_{\substack{j \in T \\ \alpha_{i,j} \geq \pi/3}} \Pr_{\mathbf{c}}[\mathbf{p}_i^{(t)}, \mathbf{p}_j^{(t)} \in H_{\mathbf{c}}] \right) \\ &= \frac{1}{p(\pi/3, \phi)} \cdot \sum_{i=1}^N \left(p(\phi) + \sum_{\substack{j \in T \\ \alpha_{i,j} \geq \pi/3}} p(\alpha_{i,j}, \phi) \right), \end{aligned} \tag{1.3}$$

where we write $\alpha_{i,j}$ for the angle between $\mathbf{p}_i^{(t)}$ and $\mathbf{p}_j^{(t)}$, and $p(\theta, \phi)$ defined above.

Furthermore, the set of centers T satisfies the property that any pair $\mathbf{p}_i^{(t)}, \mathbf{p}_j^{(t)} \in T$ of distinct centers must have angular distance larger than $\pi/3$. (If not, then they either would not have been chosen as centers or they would be nearby vectors that did not land in a bucket together, which is a low probability event.) In other words, the set of centers T is a spherical code with angular distance $\pi/3$ (after projecting the vectors onto the unit sphere).¹²

This is actually extremely convenient for our analysis. In particular, a simple bucketing argument (Lemma 2.13) shows that up to lower-order terms the maximal value of the sum over T in Equation (1.3) is achieved when T happens to be a spherical code such that $\alpha_{i,j} = \alpha$ for all i, j and some fixed angle $\alpha \geq \pi/3$. In particular, in this case, $p(\alpha_{i,j}, \phi) = p(\alpha, \phi)$ evidently does not depend on i or j . More interestingly, in this case T is a spherical code that lives inside a cap of radius α . Such codes were studied in [BM07], where it is shown that one can bound the size of such a code by the maximal size $A(\omega)$ of a spherical code with angular distance ω , for some explicit choice of ω depending on α . (The reader might notice that $\tau_n = A(\pi/3)$.)

¹²To be clear, we are not claiming this as a new observation. This property is commonly used in the analysis of sieving algorithms, typically to simply show that the number of centers is bounded by the kissing number, so that the number of input vectors minus the number of output vectors is itself bounded by the kissing number.

We therefore obtain our final expression for our running time which is

$$\begin{aligned}
\mathcal{T}^* &\approx \frac{1}{p(\pi/3, \phi)} \cdot \sum_{i=1}^N \left(p(\phi) + \max_{\alpha \geq \pi/3} A(\omega) p(\alpha, \phi) \right) \\
&= \frac{N}{p(\pi/3, \phi)} \cdot \left(p(\phi) + \max_{\alpha \geq \pi/3} A(\omega) p(\alpha, \phi) \right) \\
&\approx \frac{\tau_n}{p(\pi/3, \phi)} \cdot \left(p(\phi) + \max_{\alpha \geq \pi/3} A(\omega) p(\alpha, \phi) \right), \tag{1.4}
\end{aligned}$$

where in the last step we have set $N \approx \tau_n$, which as we discussed earlier is sufficient to prove that our algorithm solves $O(1)$ -SVP.

This is in fact our final running time (up to lower-order terms such as the $2^{\varepsilon_\mu n}$ term that we have ignored here). And, the reader might notice that Equation (1.4) is quite similar to Equation (1.1). It simply has $N \approx (4/3)^{n/2}$ replaced by $N \approx \tau_n$ and the $Np(\phi)^2$ term in Equation (1.1) replaced by the more complicated maximum over all α . This maximum over all α arises naturally in our context because we cannot rule out the possibility that our vectors $\mathbf{p}_i^{(t)}$ do in fact come from a spherical code restricted to a cap of radius α .

The quantities $p(\phi)$, $p(\pi/3, \phi)$, and $p(\alpha, \phi)$ are well understood and admit simple closed-form approximations. (See Lemma 2.9.) However, obtaining tight bounds on the kissing number τ_n or more generally on the maximum size $A(\omega)$ of a spherical code with angular distance ω is a major open problem. So, it remains to discuss the values of τ_n and $A(\omega)$. Once one understands the behavior of such values, we get an algorithm whose running time simply depends on the single parameter ϕ , and we can numerically optimize our choice of ϕ .

(Of course, we are leaving out many important details here. In our actual analysis in the sequel, we must work with an angle θ that is slightly less than $\pi/3$ (so that our vectors actually do get notably shorter at every step), and we therefore replace the $p(\pi/3, \phi)$ term with $p(\theta, \phi)$ and the term τ_n by the maximal size $A(\theta)$ of a spherical code with angular distance θ slightly less than $\pi/3$. This makes essentially no difference to the running time, assuming that the function $A(\omega)$ does not increase too rapidly as ω decreases. We also largely work with the projections of vectors onto the sphere, rather than the vectors themselves.)

1.3 On the maximal size $A(\omega)$ of a spherical code and our geometric assumption

The maximal size $A(\omega)$ of a spherical code with minimum angular separation ω is extremely well studied but not well understood. In particular, notice that the kissing number τ_n is equal to $A(\pi/3)$, so that it suffices for us to simply consider the function $A(\omega)$ in what follows.

Up to lower-order-terms the best known lower bound on $A(\omega)$ for constant $\omega \in (0, \pi/2)$ is $A(\omega) \gtrsim \sin(\omega)^{-n}$, which is what one obtains by sampling roughly that many vectors uniformly at random from the sphere and then removing the small fraction of these vectors that happen to land within angular distance ω of each other.

Until just recently, the best known upper bound on $A(\omega)$ was the celebrated bound of Kabatjanskiĭ and Levenšteĭn [KL78] (see Theorem 2.6). Quite recently, OpenAI claimed a rather small improvement to this bound [Ope26], the first improvement in nearly fifty years.

So, one can simply plug in the upper bounds on $A(\omega)$ from [KL78] to obtain a proven running time

without any assumptions. This achieves a running time of $2^{0.6078n+o(n)}$. See [Corollary 3.8](#). (One can also plug in the slightly better upper bounds achieved by [\[Ope26\]](#), but this gives a tiny improvement, and these bounds are new and not yet peer reviewed.)

However, it seems quite reasonable to guess that our lower bounds on $A(\omega)$ are far closer to its true value than the upper bound. One might even guess that the current lower bounds are essentially optimal so that one might assume that we simply have $A(\omega) = 2^{o(n)} \sin(\omega)^{-n}$. We do not claim this as a conjecture, but we do note that it is a reasonable assumption.¹³ It is under this assumption that we obtain the running times quoted in [Theorems 1.1](#) and [1.2](#).

However, we note that even if one strongly disbelieves our geometric assumption, it still has meaning in the context of our algorithm. In particular, our algorithm has the stronger property that if it fails (either because the sieving procedure fails to find close pairs or because the running time is larger than expected because the search for close pairs was too slow), it *must* be because the set of centers T forms a remarkably good spherical code.¹⁴ So, if our algorithm fails to find a short non-zero lattice vector in the desired time, it will succeed in finding an explicit spherical code that is far better than the best known spherical code. (This is in contrast to heuristic sieving algorithms, which might fail because $\mathbf{y}_i^{(t)} = \mathbf{0}$ for all i .) We could therefore equally well present our algorithm as a win-win algorithm: it takes as input a basis for a lattice and (with high probability) *either* outputs an approximate shortest vector in the claimed running time *or* outputs a better spherical code. We chose not to write the sequel in this way to avoid added complexity in the presentation, but we nevertheless feel that this is a very strong guarantee offered by our algorithm.

1.4 Concurrent work

After a version of the present work was submitted but before it was made public, two related preprints became public: one by Doulgerakis and Laarhoven [\[DL26\]](#) and one by Zhou, Wang, and Wang [\[ZWW26\]](#). Both works claim a $(3/2)^{n/2+o(n)} \approx 2^{0.2925n}$ -time algorithm with proven correctness for the average-case variant of SVP in which the input lattice $\mathcal{L} \subset \mathbb{R}^n$ is sampled uniformly at random from the Haar measure. Note that such a result is incomparable to ours.

The algorithm of Doulgerakis and Laarhoven [\[DL26\]](#) is perhaps more similar to ours, in that it is also a sieving algorithm roughly in the spirit of [\[BDGL16\]](#). However, the details appear to be quite different.

2 Preliminaries

For $n \in \mathbb{N}$, we denote $[n] = \{1, \dots, n\}$. For a set S , we write $X \sim S$ for a random variable X sampled uniformly from S . We write $\|\cdot\|$ for the ℓ_2 norm. Unless otherwise stated, all lengths are in the Euclidean (ℓ_2) norm.

¹³There are two prominent examples in the literature [\[Ven13, Kla26\]](#) in which the authors speculate that a similar assumption is true for lattice sphere packings. However, we are not aware of our specific assumption being stated anywhere. In fact, there is a conjecture suggesting that $A(\omega)$ is just a bit larger than our assumption suggests for values of ω near $\pi/3$ [\[ST06\]](#), though the status of this conjecture is unclear. We could of course have alternatively plugged in the values of $A(\omega)$ conjectured in [\[ST06\]](#).

¹⁴More accurately, the centers will either be a good spherical code with distance roughly $\pi/3$ or an explicit large subset of them will form a good spherical code *inside a cap* with distance $\pi/3$, which can then be converted into a spherical code with distance ω for some ω depending on the radius α of the cap via a simple transformation as described in [\[BM07\]](#).

We will need the following variant of Grover's quantum search algorithm [Gro96] with an unknown number of solutions in Section 3.3.

Theorem 2.1 ([BBHT98], Adapted from Theorem 3). *Let L be a list with $N \in \mathbb{N}$ elements. Let $f : L \rightarrow \{0,1\}$ be a boolean function such that $t := |f^{-1}(1)|$ is unknown. There exists a quantum algorithm that finds an $x \in L$ such that $f(x) = 1$ in expected time $\mathcal{O}\left(\sqrt{N/t}\right)$, or reports that $t = 0$ with failure probability δ , in time at most $\mathcal{O}\left(\sqrt{N} \log(1/\delta)\right)$.*

2.1 Lattices

We are interested in the following computational problem.

Definition 2.2 (μ -SVP). *Let $n \in \mathbb{N}$. Given a set $B := \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\} \subset \mathbb{Q}^n$ of n linearly independent vectors, find $\mathbf{v} \in \mathcal{L}(B)$ with $0 < \|\mathbf{v}\| \leq \mu \lambda_1(\mathcal{L}(B))$.*

We note that given an arbitrary set B' of $n' \in \mathbb{N}$ vectors in $\mathbb{Q}^n$, one can find a basis for the lattice $\mathcal{L}(B')$ of rank $\leq n$ generated by B' by using, say, the LLL algorithm [LLL82]. For a lattice $\mathcal{L}$, we define

$$\lambda_1(\mathcal{L}) := \inf \{r \in \mathbb{R} : \dim(\text{span}(\mathcal{L} \cap \mathcal{B}_n(\mathbf{0}, r))) \geq 1\}$$

as the length of the shortest non-zero vector in a lattice $\mathcal{L}$. The following result from [Reg04, HPS11] shows that in order to solve the μ -SVP problem on a lattice $\mathcal{L}$, it is sufficient to find an algorithm that solves μ -SVP when given a very good approximation to $\lambda_1(\mathcal{L})$ as an input.

Lemma 2.3 (Folklore). *Let $\mu \geq 1$, and suppose that there is an algorithm $\mathcal{A}$ with the following guarantee. Given a basis B of an n -dimensional lattice and a hint λ satisfying*

$$\lambda_1(\mathcal{L}(B)) \leq \lambda \leq \left(1 + \frac{1}{n}\right) \lambda_1(\mathcal{L}(B)), \quad (2.1)$$

the algorithm $\mathcal{A}$ runs in time at most $T(n)$ and outputs a non-zero vector in $\mathcal{L}(B)$ of length at most $\mu \cdot \lambda_1(\mathcal{L}(B))$. Then there is an efficient algorithm $\mathcal{A}'$ that uses $\mathcal{A}$ as a subroutine and solves μ -SVP on n -dimensional lattices in time at most $T(n) \cdot n^2$.

We will use the following lemma to bound the initial length of the lattice vectors our algorithms start with.

Lemma 2.4 ([NV08], Lemma 3.3). *Let $(\mathbf{b}_1, \dots, \mathbf{b}_n)$ be an LLL-reduced basis of a lattice $\mathcal{L}$. If $\mathbf{s}$ is a shortest vector of $\mathcal{L}$, then there exists an index $i \in \{1, \dots, n\}$ such that $\mathbf{s}$ belongs to the lattice spanned by $(\mathbf{b}_1, \dots, \mathbf{b}_i)$ and $\|\mathbf{b}_j\| \leq 2^{3n} \lambda_1(\mathcal{L})$ for every $j = 1, \dots, i$.*

Therefore an algorithm $\mathcal{A}$ that runs in time $T(n)$ and that assumes that the input basis vectors have length at most $2^{3n} \lambda_1(\mathcal{L})$ implies an algorithm $\mathcal{A}'$ that runs in time $n \cdot T(n)$ without this assumption; $\mathcal{A}'$ merely guesses the i and uses $\mathcal{A}$ as a subroutine.

2.2 A bit of geometry

For any non-zero vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, we write $d(\mathbf{x}, \mathbf{y})$ for the angular distance between them,

$$d(\mathbf{x}, \mathbf{y}) := \arccos \left(\frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|} \right).$$

We define $\mathcal{S}^{n-1} := \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| = 1\}$ to be the unit sphere in $\mathbb{R}^n$. For $\phi \in [0, \pi]$ and $\mathbf{v} \in \mathbb{R}^n$, we define $\mathcal{C}_n(\mathbf{v}, \phi)$ as a (unit) spherical cap of angular radius ϕ centered at $\mathbf{v}$:

$$\mathcal{C}_n(\mathbf{v}, \phi) := \{\mathbf{x} \in \mathcal{S}^{n-1} : d(\mathbf{x}, \mathbf{v}) < \phi\}.$$

We write $\mathcal{C}_n(\phi)$ when the center is clear from context or unimportant.

A *spherical code* is a finite subset of the sphere, $C \subset \mathcal{S}^{n-1}$. The parameter

$$\theta^*(C) := \min_{\mathbf{x} \neq \mathbf{y} \in C} d(\mathbf{x}, \mathbf{y})$$

is called the *distance* of the code. To show correctness and bound the complexity of our lattice sieve, we frequently need upper bounds on the maximum size of *any* spherical code in n dimensions and of some distance θ .

Definition 2.5. Let $\theta \in (0, \pi]$ and $n \in \mathbb{N}$. We define $A(n, \theta)$ to be the maximum number of points in $\mathcal{S}^{n-1}$ such that the angular distance between each pair of distinct points is at least θ .

We know the following celebrated result (also see [CE03, SZ24]).

Theorem 2.6 (Kabatjanskiĭ and Levenšteĭn [KL78, Lev79]). Let $0 < \theta < \pi/2$ be a constant. Then,

$$A(n, \theta) \leq A_{\text{Lev}}(n, \theta) := 2^{R_{\text{LP}}(\theta)n + o(n)},$$

where

$$R_{\text{LP}}(\theta) := \frac{1 + \sin \theta}{2 \sin \theta} \log_2 \frac{1 + \sin \theta}{2 \sin \theta} - \frac{1 - \sin \theta}{2 \sin \theta} \log_2 \frac{1 - \sin \theta}{2 \sin \theta}.$$

When $\theta = \pi/3$ this is the well known upper bound on the (geometric) kissing number, usually denoted as τ_n .

$$\tau_n := A(n, \pi/3) \leq 2^{0.401414n + o(n)}.$$

We will also need an upper bound on the size of spherical codes restricted to a spherical cap.

Definition 2.7. Let $0 < \theta \leq \pi/2$, $\theta/2 < \phi$ and $n \in \mathbb{N}$. We define $A(n, \theta, \phi)$ to be the maximum number of points on $\mathcal{C}_n(\phi)$ such that the angular distance between each pair of distinct points is at least θ .

We will use a result by Barg and Musin [BM07, Equation 9], where they relate the rate of codes on spherical caps and strips to the rate of codes of a larger distance on the sphere. They show that for $0 < \theta < \pi/2$, $\theta/2 < \phi \leq \pi/2$, and for

$$\omega(\theta, \phi) := \arccos \left(\frac{\cos \theta - \cos^2 \phi}{\sin^2 \phi} \right), \tag{2.2}$$

we have that

$$A(n-1, \omega(\theta, \phi)) \leq A(n, \theta, \phi) \leq A(n, \omega(\theta, \phi)). \quad (2.3)$$

We will frequently use the following probability. Let $\theta \in [0, \pi]$ and $\theta/2 < \phi < \pi$ be constants. Let $\mathbf{u}, \mathbf{v} \in \mathcal{S}^{n-1}$ be two arbitrary points such that $d(\mathbf{u}, \mathbf{v}) = \theta$. We define

$$p(n, \theta, \phi) := \Pr_{\mathbf{c} \sim \mathcal{S}^{n-1}} [\mathbf{u} \in \mathcal{C}_n(\mathbf{c}, \phi) \wedge \mathbf{v} \in \mathcal{C}_n(\mathbf{c}, \phi)]. \quad (2.4)$$

We also define

$$p(n, \phi) := p(n, 0, \phi). \quad (2.5)$$

This corresponds to the probability of the event $\mathbf{v} \in \mathcal{C}_n(\mathbf{c}, \phi)$ for a fixed point $\mathbf{v} \in \mathcal{S}^{n-1}$ and a uniformly random point $\mathbf{c} \sim \mathcal{S}^{n-1}$, and hence

$$p(n, \phi) = (\sin \phi)^{n+o(n)}.$$

A reasonable assumption is that the kissing number $A(n, \pi/3)$ is asymptotically upper bounded by $(\sin(\pi/3))^{-n}$. The best known lower bounds on $A(n, \theta)$ for any constant $\theta \in (0, \pi/2)$ are in fact $2^{o(n)} \cdot \sin(\theta)^{-n}$. See [Section 1.3](#) for more discussion.

Assumption 2.8. *Let $0 < \theta < \pi/2$ be a constant. Then,*

$$A(n, \theta) \leq 2^{o(n)} \sin(\theta)^{-n}$$

Lemma 2.9. *Let $5/\sqrt{n} \leq \theta, \phi \leq \pi/2$ with $\phi \geq \theta/2 + 5/\sqrt{n}$ and $n \geq 20$. Then,*

$$\frac{e^{-3/\beta}}{2\pi n} \cdot \beta^{n/2-2} \leq p(n, \theta, \phi) \leq \frac{n}{2} \cdot \beta^{n/2-2}, \quad (2.6)$$

where

$$\beta := 1 - \frac{2 \cos^2 \phi}{1 + \cos \theta} \in (0, 1].$$

Proof. Without loss of generality let $\mathbf{u} = \mathbf{e}_1$ and $\mathbf{v} = \mathbf{e}_1 \cos \theta + \mathbf{e}_2 \sin \theta$. Let

$$T := \{(x_1, x_2) \in \mathbb{R}^2 : x_1^2 + x_2^2 \leq 1, x_1 \geq \cos \phi, x_1 \cos \theta + x_2 \sin \theta \geq \cos \phi\} \subset \mathbb{R}^2,$$

and let $\mathbf{x}' := (x_1, x_2)$. (The condition $\phi > \theta/2$ is what guarantees that T is non-empty.) We have

$$\begin{aligned} p(n, \theta, \phi) &= \Pr_{\mathbf{x} \sim \mathcal{S}^{n-1}} [\mathbf{u} \in \mathcal{C}_n(\mathbf{x}, \phi) \text{ and } \mathbf{v} \in \mathcal{C}_n(\mathbf{x}, \phi)] \\ &= \Pr_{\mathbf{x} \sim \mathcal{S}^{n-1}} [(x_1, x_2) \in T] \\ &= \frac{\sigma_{n-3}}{\sigma_{n-1}} \int_T (1 - x_1^2 - x_2^2)^{n/2-2} d\mathbf{x}' \\ &= \frac{n-2}{2\pi} \int_T (1 - x_1^2 - x_2^2)^{n/2-2} d\mathbf{x}', \end{aligned} \quad (2.7)$$

where σ_{m-1} is the surface area $\mathcal{S}^{m-1}$, and where we have applied the well known recurrence $\sigma_{n-1} = 2\pi\sigma_{n-3}/(n-2)$.

Let $\alpha := (1 - \cos \theta) \cos \phi / \sin \theta$ be the minimal value that x_2 takes in the integrand from [Equation \(2.7\)](#) when $x_1 = \cos \phi$. Notice in particular that $\beta = 1 - \cos^2 \phi - \alpha^2$. A simple check then shows that the

integrand in Equation (2.7) is maximized when $x_1 = \cos \phi$ and $x_2 = \alpha$, where its value is exactly $\beta^{n/2-2}$. It follows in particular that the integral is bounded by the area of T times $\beta^{n/2-2}$. Noting that the area of T is at most π gives the upper bound.

For the lower bound, let $\varepsilon := 1/(n-2)$. A quick check shows that T contains the square $[\cos \phi, \cos \phi + \varepsilon] \times [\alpha, \alpha + \varepsilon]$. Applying Equation (2.7), we see that

$$\begin{aligned} p(n, \theta, \phi) &\geq \frac{n-2}{2\pi} \int_{\cos \phi}^{\cos \phi + \varepsilon} \int_{\alpha}^{\alpha + \varepsilon} (1 - x_1^2 - x_2^2)^{n/2-2} dx_2 dx_1 \\ &\geq \varepsilon^2 \cdot \frac{n-2}{2\pi} \cdot (1 - (\cos \phi + \varepsilon)^2 - (\alpha + \varepsilon)^2)^{n/2-2} \\ &\geq \varepsilon^2 \cdot \frac{n-2}{2\pi} \cdot (\beta - 5\varepsilon)^{n/2-2} \\ &= \varepsilon^2 \cdot \frac{n-2}{2\pi} \cdot \beta^{n/2-2} \cdot (1 - 5\varepsilon/\beta)^{n/2-2} \\ &\geq \frac{e^{-3/\beta}}{2\pi n} \cdot \beta^{n/2-2}, \end{aligned}$$

as claimed, where the last step uses the fact that $\beta \geq 18/n$ (which follows from the lower bound on θ and ϕ). $\square$

We will write $\text{vol}_n(S)$ for the n -dimensional volume of a set $S \subset \mathbb{R}^n$. We know that for $n \in \mathbb{N}$ and $R \in \mathbb{R}$,

$$\text{vol}_n(\mathcal{B}_n(R)) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2} + 1)} \cdot R^n, \quad (2.8)$$

where Γ is the gamma function. The following lemma is a slight variant of [ABB⁺23, Lemma 2.1], whose proof uses a simple elegant trick due to Zhou [Zho19].

Lemma 2.10. *Let $n \in \mathbb{N}$. For $\mathbf{v} \in \mathbb{R}^n$ such that $\|\mathbf{v}\| \leq 2R$, consider the lens $\mathcal{B}_n(R) \cap \mathcal{B}_n(\mathbf{v}, R)$ of angle $\phi := \arccos\left(\frac{\|\mathbf{v}\|}{2R}\right)$. We have that*

$$\sqrt{\frac{2n}{\pi}} \cdot \frac{\sin^{n+1} \phi}{n+1} \leq \frac{\text{vol}_n(\mathcal{B}_n(R) \cap \mathcal{B}_n(\mathbf{v}, R))}{\text{vol}_n(\mathcal{B}_n(R))} \leq \phi \cdot \sqrt{\frac{2n+2}{\pi}} \cdot \sin^n \phi \quad (2.9)$$

Proof. Let $C_n(R, \phi) := \{\mathbf{x} \in \mathcal{B}_n(R) : \langle \mathbf{x}, \mathbf{e}_1 \rangle \geq R \cos \phi\}$. We know from [Li11] that

$$\text{vol}_n(C_n(R, \phi)) = \text{vol}_{n-1}(\mathcal{B}_{n-1}(1)) \cdot R^n \cdot \int_0^\phi \sin^n(\alpha) d\alpha.$$

We will first bound this integral. For the lower bound, we use a trick from [ABB⁺23, Equation 3]. Since $\forall \alpha \in [0, \pi/2]$, $\cos \alpha \in [0, 1]$, we have that

$$\int_0^\phi \sin^n(\alpha) d\alpha \geq \int_0^\phi \sin^n(\alpha) \cos(\alpha) d\alpha = \sin^{n+1}(\phi)/(n+1).$$

Therefore,

$$\text{vol}_n(C_n(R, \phi)) \geq \text{vol}_{n-1}(\mathcal{B}_{n-1}(1)) \cdot R^n \cdot \sin^{n+1}(\phi)/(n+1). \quad (2.10)$$

The upper bound follows from the observation that the integrand is maximized at $\alpha = \phi$:

$$\text{vol}_n(C_n(R, \phi)) \leq \text{vol}_{n-1}(\mathcal{B}_{n-1}(1)) \cdot R^n \cdot \phi \cdot \sin^n(\phi). \quad (2.11)$$

The balls $\mathcal{B}_n(R)$ and $\mathcal{B}_n(\mathbf{v}, R)$ are congruent, and therefore the volume of the lens $\text{vol}_n(\mathcal{B}_n(R) \cap \mathcal{B}_n(\mathbf{v}, R))$ is equal to the volume of the union of two disjoint spherical caps congruent to $C_n(R, \phi)$. Therefore, using Equation (2.10) and Equation (2.11), we have that

$$2 \cdot \frac{\sin^{n+1} \phi}{n+1} \cdot \left(\frac{\text{vol}_{n-1}(\mathcal{B}_{n-1}(1))}{\text{vol}_n(\mathcal{B}_n(1))} \right) \leq \frac{\text{vol}_n(\mathcal{B}_n(R) \cap \mathcal{B}_n(\mathbf{v}, R))}{\text{vol}_n(\mathcal{B}_n(R))} \leq 2\phi \cdot \sin^n \phi \cdot \left(\frac{\text{vol}_{n-1}(\mathcal{B}_{n-1}(1))}{\text{vol}_n(\mathcal{B}_n(1))} \right). \quad (2.12)$$

Using Equation (2.8) and Gautschi's inequality [Gau59], it follows that

$$\sqrt{\frac{n}{2\pi}} \leq \frac{\text{vol}_{n-1}(\mathcal{B}_{n-1}(1))}{\text{vol}_n(\mathcal{B}_n(1))} \leq \frac{1}{\sqrt{\pi}} \left(\frac{n}{2} + 1 \right)^{1/2}.$$

The result follows by combining this with Equation (2.12). $\square$

Given a spherical code $C \subset \mathcal{S}^{n-1}$ and a vector $\mathbf{v} \in \mathbb{R}^n$, we say $\mathbf{v}$ decodes to $\mathbf{c}^* \in C$ if $\mathbf{c}^* = \arg \min_{\mathbf{c} \in C} d(\mathbf{v}, \mathbf{c})$. Given a parameter $\alpha \in [0, \pi]$, we call the set

$$C' = \{\mathbf{c} \in C : d(\mathbf{v}, \mathbf{c}) < \alpha\} \subseteq C$$

the α -list decoding of $\mathbf{v}$. We use the following family of list decodable spherical codes from [BDGL16].

Definition 2.11 ($((n, m, K)$ -Random Product Codes). *Let $n, m, K, b \in \mathbb{N}$ such that $n = bm$. The distribution $\mathcal{R}_{n,m,K}$ on subsets of $\mathbb{R}^n$ of size K^m is defined as the distribution of codes C of the form*

$$C = Q \cdot (C_1 \times C_2 \times \cdots \times C_m),$$

where $Q \in \mathcal{O}_n(\mathbb{R})$ is a uniformly random rotation over $\mathbb{R}^n$ and the sub-codes $C_i \subset \sqrt{1/m} \cdot \mathcal{S}^{b-1}$ for $i \in [m]$ are sets of K uniformly random and independently sampled vectors over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$.

We will use the following lemma from [BDGL16] which says that there exists a list decoding algorithm for random product codes.

Lemma 2.12 ([BDGL16], Lemma 5.1). *There exists an algorithm that, given the description $Q, C_1, \dots, C_m$ of a random product code $C \sim \mathcal{R}_{n,m,K}$, a parameter $\phi \in (0, \pi/2)$ and a target vector $\mathbf{t} \in \mathcal{S}^{n-1}$, returns the set $S = C \cap C_n(\mathbf{t}, \phi)$ in expected time*

$$\mathbb{E}[\mathcal{T}_{LD}(n, K, m, \phi)] = \mathcal{O}(nK + mK \log K + mK^m \cdot p(n, \phi))$$

over the randomness of $C \sim \mathcal{R}_{n,m,K}$.

We will also need the following technical results.

Lemma 2.13. *Let $0 < \theta < \pi/2$ and $\theta/2 < \phi$. Let $T \subset \mathcal{S}^{n-1}$ be such that for every distinct pair $\mathbf{x}, \mathbf{y} \in T$, $d(\mathbf{x}, \mathbf{y}) \geq \theta$. Then we have that for any fixed $\mathbf{p} \in \mathbb{R}^n$,*

$$\sum_{\substack{\mathbf{x} \in T \\ d(\mathbf{p}, \mathbf{x}) \geq \theta}} p(n, d(\mathbf{p}, \mathbf{x}), \phi) \leq \mathcal{O}(n^2) \sup_{\theta \leq \alpha \leq \pi/2} A(n, \theta, \alpha) \cdot p(n, \alpha, \phi).$$

Proof. Notice that the sum over T is the same as a sum over θ -separated points in $T \setminus (T \cap \mathcal{C}(\mathbf{p}, \theta))$. Let us divide this domain into n^2 successive “rings” at an angular radius α from $\mathbf{p}$, and bound the sum restricted to each such ring with the maximum number of points possible per ring¹⁵. Let $\alpha_1 := \theta$. For $j \in \{2, \dots, n^2\}$, let α_j be such that

$$2^{-j}p(n, \theta, \phi) < p(n, \alpha_j, \phi) \leq 2^{-j+1}p(n, \theta, \phi).$$

Notice in particular that $|\{\mathbf{x} \in T : \alpha_j \leq d(\mathbf{p}, \mathbf{x}) < \alpha_{j+1}\}| \leq A(n, \theta, \alpha_{j+1})$. Omitting the dimension n in the arguments of p and A below for compactness, we have

$$\begin{aligned} \sum_{\substack{\mathbf{x} \in T \\ d(\mathbf{p}, \mathbf{x}) \geq \theta}} p(d(\mathbf{p}, \mathbf{x}), \phi) &\leq \sum_{j=1}^{n^2} \sum_{\substack{\mathbf{x} \in T \\ \alpha_j \leq d(\mathbf{p}, \mathbf{x}) < \alpha_{j+1}}} p(d(\mathbf{p}, \mathbf{x}), \phi) + 2^{-n^2}|T| \\ &\leq n^2 \max_j A(\theta, \alpha_{j+1}) \cdot p(\alpha_j, \phi) + 2^{-n^2} A(\theta) \\ &\leq 2n^2 \max_j A(\theta, \alpha_{j+1}) \cdot p(\alpha_{j+1}, \phi) + 2^{-n^2} A(\theta) \\ &\leq 2n^2 \sup_{\theta \leq \alpha \leq \pi} A(\theta, \alpha) \cdot p(\alpha, \phi) + 2^{-n^2} A(\theta) \\ &\leq 4n^2 \sup_{\theta \leq \alpha \leq \pi/2} A(\theta, \alpha) \cdot p(\alpha, \phi) + 2^{-n^2} A(\theta) \\ &\leq \mathcal{O}(n^2) \cdot \sup_{\theta \leq \alpha \leq \pi/2} A(\theta, \alpha) \cdot p(\alpha, \phi), \end{aligned}$$

as claimed, where in the third inequality we have used the fact that $p(n, \alpha_j, \phi) \leq 2p(n, \alpha_{j+1}, \phi)$, and in the fourth inequality we have used the fact that for $\alpha \geq \pi/2$, $A(\theta, \alpha)p(\alpha, \phi) \leq 2A(\theta, \pi/2)p(\alpha, \phi) \leq 2A(\theta, \pi/2)p(\pi/2, \alpha)$.¹⁶ $\square$

Lemma 2.14. *For $\gamma \in (1/2, 1)$, let*

$$\theta := \min \left(\arccos \frac{1}{2\gamma}, 2 \arcsin \frac{\gamma}{2} \right).$$

Let $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{R}^n$ be such that $\|\mathbf{p}_1\|, \|\mathbf{p}_2\| \in (\gamma R, R]$ and $d(\mathbf{p}_1, \mathbf{p}_2) \leq \theta$. Then, $\|\mathbf{p}_1 - \mathbf{p}_2\| \leq \gamma R$.

Proof. Assume $\|\mathbf{p}_1\| \leq \|\mathbf{p}_2\|$. It is sufficient to show that

$$\frac{\|\mathbf{p}_1 - \mathbf{p}_2\|^2}{\|\mathbf{p}_1\|^2} \leq \gamma^2.$$

Since $d(\mathbf{p}_1, \mathbf{p}_2) \leq \theta$, we have that

$$\|\mathbf{p}_1 - \mathbf{p}_2\|^2 \leq \|\mathbf{p}_1\|^2 + \|\mathbf{p}_2\|^2 - 2\|\mathbf{p}_1\| \|\mathbf{p}_2\| \cos \theta.$$

¹⁵We use a seemingly crude bound of a cap instead of the ring; however [BM07] show that these are related, and in fact asymptotically alike.

¹⁶The fact that $A(\theta, \alpha) \leq 2A(\theta, \pi/2)$ follows by noting that for any configuration of $N = A(\theta, \alpha)$ points in a cap with angular radius α , at least $N/2$ of these points must lie in some hemisphere (i.e., some cap with radius $\pi/2$). The fact that $p(\pi/2, \phi) \geq p(\alpha, \phi)$ follows from the monotonicity of p .

For $a \in (\gamma, 1]$, define $f(a) := 1 + a^2 - 2a \cos \theta$ and notice that for $r := \frac{\|\mathbf{p}_1\|}{\|\mathbf{p}_2\|} \in (\gamma, 1]$,

$$\frac{\|\mathbf{p}_1 - \mathbf{p}_2\|^2}{\|\mathbf{p}_1\|^2} \leq f(r).$$

Since f is a convex function, $f(r) \leq \max(f(\gamma), f(1))$. We have that

$$f(1) = 2(1 - \cos \theta) = 4 \sin^2 \frac{\theta}{2} \leq \gamma^2,$$

and that

$$f(\gamma) = 1 + \gamma^2 - 2\gamma \cos \theta \leq \gamma^2,$$

where the inequalities follow because

$$\gamma = \max \left(\frac{1}{2 \cos \theta}, 2 \sin \frac{\theta}{2} \right). \quad \square$$

2.3 Good spherical codes

Definition 2.15 ($(\theta, \phi, \delta, T)$ -Good Spherical Codes Distribution). *Let Codes be a distribution over finite subsets $C \subset \mathcal{S}^{n-1}$, together with a decoding algorithm $\text{ListDecode}_C(\mathbf{v})$ that takes as input $\mathbf{v} \in \mathcal{S}^{n-1}$ and runs in time $\mathcal{T}_C(\mathbf{v})$. We say that the distribution Codes is $(\theta, \phi, \delta, T)$ -good if the following hold for $C \sim \text{Codes}$:*

1. *The marginal distribution of every $\mathbf{c} \in C$ is uniform over $\mathcal{S}^{n-1}$.*
2. *$\forall \mathbf{v} \in \mathcal{S}^{n-1}$, $\text{ListDecode}_C(\mathbf{v}) = C \cap \mathcal{C}_n(\mathbf{v}, \phi)$.*
3. *$\forall \mathbf{v} \in \mathcal{S}^{n-1}$, $\mathbb{E}_C[\mathcal{T}_C(\mathbf{v})] \leq T$.*
4. *$\forall \mathbf{v}, \mathbf{u} \in \mathcal{S}^{n-1}$ such that $d(\mathbf{v}, \mathbf{u}) < \theta$, $\Pr_C[\exists \mathbf{c} \in C : \mathbf{c} \in \mathcal{C}(\mathbf{v}, \phi) \cap \mathcal{C}(\mathbf{u}, \phi)] \geq 1 - \delta$.*

Claim 2.16. *Let $0 < \theta < \pi/2$ and $\theta/2 < \phi < \pi/2$ be fixed constants. For all sufficiently large $n \in \mathbb{N}$, there exists a distribution $\text{Codes}_{n, \theta, \phi}$ over finite subsets $C \subset \mathcal{S}^{n-1}$ such that $|C| = 2^{o(n)} p(n, \theta, \phi)^{-1}$ and $\text{Codes}_{n, \theta, \phi}$ is $(\theta, \phi, \frac{1}{n^n}, 2^{o(n)} p(n, \theta, \phi)^{-1} p(n, \phi))$ -good spherical codes distribution.*

Proof. Let $m = \text{poly log}(n)$ be an integer. By possibly increasing n by at most an additive $m = \text{poly log}(n)$ by padding with zeros, we can assume without loss of generality that $m \mid n$. Define

$$K := \lceil (2^{n^{2/3}} p(n, \theta, \phi)^{-1})^{1/m} \rceil.$$

Each sample from the distribution Codes consists of n independent samples from $\mathcal{R}_{n, m, K}$ from [Definition 2.11](#). That is, $C = \bigcup_j C_j$ where $C_j \sim \mathcal{R}_{n, m, K}$. Clearly

$$|C| = n \cdot K^m \leq n \cdot 2^m \cdot 2^{n^{2/3}} p(n, \theta, \phi)^{-1} = 2^{o(n)} p(n, \theta, \phi)^{-1}.$$

Moreover, [Item 1](#) is implied by the uniformly random rotation Q_j applied to each of the product codes sampled according to [Definition 2.11](#).

The decoding algorithm $\text{ListDecode}_C(\mathbf{v})$ works as follows. For each $j \in [n]$, it runs the list decoding algorithm from [Lemma 2.12](#) on the j th random product code, and outputs the union of their outputs. It follows from [Lemma 2.12](#) that the output is as desired in [Item 2](#).

The expected time complexity is as in [Item 3](#):

$$n \cdot \mathbb{E}[\mathcal{T}_{LD}(n, K, m, \phi)] = 2^{o(n)} p(n, \theta, \phi)^{-1} p(n, \phi).$$

Lastly for any $\mathbf{v}, \mathbf{u} \in \mathcal{S}^{n-1}$, $d(\mathbf{u}, \mathbf{v}) \leq \theta$ and a random product code C_j , [\[BDGL16, Theorem 5.1\]](#) says that

$$\Pr_{C_j}[\forall \mathbf{c} \in C_j : \mathbf{c} \notin \mathcal{C}(\mathbf{v}, \phi) \cap \mathcal{C}(\mathbf{u}, \phi)] \leq \text{negl}(n),$$

over the randomness of the random product code. Since $\text{negl}(n) < \frac{1}{n}$ for all sufficiently large n , and the n random product codes are independent, we have that

$$\Pr[\forall j \in [n], \forall \mathbf{c} \in C_j : \mathbf{c} \notin \mathcal{C}(\mathbf{v}, \phi) \cap \mathcal{C}(\mathbf{u}, \phi)] \leq \text{negl}(n)^n \leq n^{-n},$$

and [Item 4](#) follows as the converse of this. □

3 Faster lattice sieving without heuristics

3.1 Sieving with good spherical codes

In this section we describe our sieving procedure.

Algorithm 1 Our sieving procedure

Input: Dimension n ; sieve size γ ; cap size ϕ ; list of vector pairs $S \subseteq \mathcal{L} \times \mathcal{B}_n(R)$ for some $R \in \mathbb{R}$.

Output: A list $S' \subseteq \mathcal{L} \times \mathcal{B}_n(\gamma R + \varepsilon)$ of pairs for $\varepsilon := \max_{(\mathbf{v}, \mathbf{p}) \in S} \|\mathbf{v} - \mathbf{p}\|$.

```

1:  $S' \leftarrow \emptyset$ 
2:  $W \leftarrow \emptyset$ 
3:  $\theta \leftarrow \min\left(\arccos \frac{1}{2\gamma}, 2 \arcsin \frac{\gamma}{2}\right)$ 
4: Move all pairs  $(\mathbf{v}, \mathbf{p}) \in S$  such that  $\|\mathbf{p}\| \leq \gamma R + \varepsilon$  to  $S'$ 
5:  $\text{Code} \sim \text{Codes}_{n, \theta, \phi}$ 
6: for Remaining pairs  $(\mathbf{v}, \mathbf{p}) \in S$  do
7:    $\text{words} \leftarrow \text{ListDecode}(\text{Code}, \mathbf{p}/\|\mathbf{p}\|)$ 
8:   for  $\mathbf{c} \in \text{words}$  such that the table  $H_{\mathbf{c}}$  is initialized do
9:     if  $\exists (\mathbf{v}_c, \mathbf{p}_c) \in H_{\mathbf{c}}$  such that  $d(\mathbf{p}, \mathbf{p}_c) < \theta$  then ▷ Search for a center
10:      Add  $(\mathbf{v} - \mathbf{v}_c, \mathbf{p} - \mathbf{p}_c)$  to  $S'$  and exit the loop ▷ The pair is reduced
11:    end if
12:  end for
13:  if The pair is not reduced then
14:    Add  $(\mathbf{v}, \mathbf{p})$  to  $W$ 
15:    for  $\mathbf{c} \in \text{words}$  do
16:      Add  $(\mathbf{v}, \mathbf{p})$  to  $H_{\mathbf{c}}$  ▷ If  $H_{\mathbf{c}}$  is not initialized, initialize it here.
17:    end for
18:  end if
19: end for
20: return  $S'$ 
```

Theorem 3.1 (Correctness of the sieving procedure). *For $\gamma \in (1/2, 1)$, let $0 < \theta < \pi/2$ be as in Lemma 2.14, let $\theta/2 < \phi < \pi/2$ be constant, and $n \in \mathbb{N}$. Suppose Algorithm 1 takes as an input a list $S \subseteq \mathcal{L} \times \mathcal{B}_n(R)$ of vector pairs of size $|S| \leq 2^{\mathcal{O}(n)}$, for some $R \in \mathbb{R}$. Let $\varepsilon := \max_{(v, p) \in S} \|v - p\|$. Then except with probability $\exp(-\Theta(n))$ over the choice of the good code Code, it outputs a list $S' \subset \mathcal{L} \times \mathcal{B}_n(\gamma R + \varepsilon)$, such that at the end of the algorithm*

1. Every distinct $P = (v_1, p_1), Q = (v_2, p_2) \in W$ is such that $d(p_1, p_2) \geq \theta$.
2. $|S'| \geq |S| - A(n, \theta)$.
3. Every pair (v', p') in S' was either a pair in S , or of the form $(v - v_c, p - v_c)$, where $(v_c, p_c) \in W$, and $(v, p) \in S \setminus W$. Moreover $\|v' - p'\| \leq \varepsilon$ and $\|p'\| \leq \gamma R + \varepsilon$.

Proof. The guarantee on the length of the output follows from Lemma 2.14.

Recall that Code is a sample from a $(\theta, \phi, \frac{1}{n^n}, 2^{o(n)}p(n, \theta, \phi)^{-1}p(n, \phi))$ -good spherical codes distribution as in Claim 2.16. Suppose that for fixed $P, Q \in S$, P was processed before Q and we have $P \in W$. Then by Claim 2.16 the probability that when Q is processed, the algorithm does not find P in any table H_j in Line 9 is at most $1/n^n$. Therefore Q would not be added to T , except with $1/n^n$ probability. Item 1 follows by a union bound over all $2^{\mathcal{O}(n)}$ pairs of pairs P and Q in S .

From Definition 2.5 and Item 1 it follows that $|W| \leq A(n, \theta)$ except with probability $\exp(-\Theta(n))$ over the choice of Code. Notice that every pair in S processed by Algorithm 1 either results in a pair being added to S' in Line 10 of the algorithm or a pair being added to W in Line 14. Item 2 follows from these observations.

The first part of Item 3 is clear from Line 10. Moreover, $\|p'\| = \|p - v_c\| \leq \gamma R + \varepsilon$ follows from Lemma 2.14 and the definition of ε using triangle inequality. $\square$

Theorem 3.2 (Complexity of the sieving procedure). *For $\gamma \in (1/2, 1)$, let $0 < \theta < \pi/2$ be as in Lemma 2.14, let $\theta/2 < \phi < \pi/2$ be a constant, and $n \in \mathbb{N}$. Suppose Algorithm 1 takes as an input a list $S \subseteq \mathcal{L} \times \mathcal{B}_n(R)$ of vector pairs of size $|S| \leq 2^{\mathcal{O}(n)}$, for some $R \in \mathbb{R}$. Let $\mathcal{T}_{HS}$ and $\mathcal{S}_{HS}$ respectively be the time and space complexity of Algorithm 1. Then*

1. With high probability over the randomness of Code, the running time is

$$\mathcal{T}_{HS} \leq 2^{o(n)} |S| p(n, \theta, \phi)^{-1} \left(p(n, \phi) + \sup_{\theta < \alpha < \pi/2} A(n, \theta, \alpha) \cdot p(n, \alpha, \phi) \right).$$

2. With high probability over the randomness of Code, the space is

$$\mathcal{S}_{HS} \leq |S| + 2^{o(n)} p(n, \theta, \phi)^{-1} p(n, \phi) \min(|S|, A(n, \theta)).$$

Proof. Time Complexity. Let $U = S \setminus W$, $N = |U|$, and suppose $U = \{P_j\}_{j \in [N]}$. Let $T^* \subset U$ be such that for $i \in [N]$, $P_i \in T^*$ if and only if it is θ -far from every $P_j \in T^*$ for $j < i$. T^* is the set W if we used $\phi = \pi$, $|\text{Code}| = 1$, and a trivial decoder; this corresponds to the case when there is only one bucket and we are executing the [AKS01] algorithm and does not depend on the choice of the code. From Definition 2.5 it follows that $|T^*| \leq A(n, \theta)$.

Let E be the event that [Item 4](#) of [Definition 2.15](#) holds for every pair (of pairs) in $U \times U$. From our choice of parameters for the code, and union bound over all pairs of pairs, we have that $\Pr[E^c] \leq N^2 n^{-n}$, where E^c denotes the complement of E . Moreover, conditioned on E , we have that $W = T^*$.

For each candidate pair $P \in S$, the algorithm first decodes it using the decoder from [Claim 2.16](#), then checks all the pairs so far in W that fall within the same buckets as P , then possibly spends some time adding an unreduced pair to the relevant buckets. Crucially, the loop processing P breaks as soon as P is reduced. Therefore, the time complexity (conditioned on E) is upper bounded by

$$\mathcal{T}_{HS} \leq \sum_{(\mathbf{v}, \mathbf{p}) \in S} \mathcal{T}_C(\mathbf{p}) + \sum_{(\mathbf{v}, \mathbf{p}) \in S} \sum_{\mathbf{c} \in \text{Code}} \sum_{\substack{(\mathbf{v}', \mathbf{p}') \in T^* \\ d(\mathbf{p}, \mathbf{p}') \geq \theta}} 1[d(\mathbf{c}, \mathbf{p}) \leq \phi] 1[d(\mathbf{c}, \mathbf{p}') \leq \phi]. \quad (3.1)$$

Averaging over the code, and using [Equation \(2.4\)](#) and [Claim 2.16](#), we have that

$$\mathbb{E}_{\text{Code}}[\mathcal{T}_{HS} 1_E] \leq |S| 2^{o(n)} p(n, \theta, \phi)^{-1} p(n, \phi) + |\text{Code}| \sum_{(\mathbf{v}, \mathbf{p}) \in S} \sum_{\substack{(\mathbf{v}', \mathbf{p}') \in T^* \\ d(\mathbf{p}, \mathbf{p}') \geq \theta}} p(n, d(\mathbf{p}, \mathbf{p}'), \phi).$$

Therefore from [Lemma 2.13](#) applied to the second element of the tuples it follows that

$$\mathbb{E}_{\text{Code}}[\mathcal{T}_{HS}] \leq 2^{o(n)} |S| p(n, \theta, \phi)^{-1} \left(p(n, \phi) + \sup_{\theta < \alpha < \pi/2} A(n, \theta, \alpha) \cdot p(n, \alpha, \phi) \right) + N^2 p(n, \theta, \phi)^{-1} \Pr[E^c],$$

and by our choice of parameters and [Lemma 2.9](#) the last term is negligible. This upper bound is concentrated by Markov's inequality.

Space Complexity. We have that

$$\mathcal{S}_{HS} = |S| + \sum_{\mathbf{c}_j \in \text{Code}} |H_j| = |S| + \sum_{(\mathbf{v}, \mathbf{p}) \in T^*} \sum_{\mathbf{c}_j \in \text{Code}} 1[d(\mathbf{c}, \mathbf{p}) \leq \phi].$$

Therefore following analysis similar to that of the time complexity,

$$\mathbb{E}_{\text{Code}}[\mathcal{S}_{HS}] \leq |S| + |T^*| |\text{Code}| p(n, \phi),$$

and this concentrates by Markov's inequality. $\square$

3.2 Our sieving algorithm

In this section we give our classical algorithm for the μ -SVP problem, for constant μ , that uses the sieve from the previous section $\text{poly}(n)$ times. Indeed, given what we have already done in the previous section, the remainder of the proof follows prior work closely.

We assume that a guess λ satisfying [Equation \(2.1\)](#) is known as per [Lemma 2.3](#), the input basis is LLL-reduced, and that every basis vector is of length at most $2^{3n} \lambda$ as per [Lemma 2.4](#).

We begin with an algorithm from [[Reg04](#), [HPS11](#)] that takes as an input a basis B for a lattice and a perturbation $\mathbf{x}$ (an arbitrary vector) and outputs a pair $(\mathbf{v}, \mathbf{p})$ such that $\mathbf{v} \in \mathcal{L}(B)$ and $\mathbf{p} = \mathbf{v} + \mathbf{x}$. Notice that the vector $\mathbf{p}$ is the lattice vector $\mathbf{v}$ perturbed with $\mathbf{x}$.

Algorithm 2 Sample

Input: $B = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ and perturbation $\mathbf{x}$

Output: A lattice vector $\mathbf{v}$ and its perturbed variant $\mathbf{p} = \mathbf{v} + \mathbf{x}$

- 1: Compute $\mathbf{v} = -\mathbf{x} + (\mathbf{x} \bmod \mathcal{P}(B))$
 - 2: **return** pair $(\mathbf{v}, \mathbf{p} := \mathbf{v} + \mathbf{x})$
-

Algorithm 3 Our sieving algorithm

Input: Lattice basis $B = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$; stopping radius R_0 ; sieve size $\gamma \in (1/2, 1)$; clustering parameter $\phi \in (\theta/2, \pi/2)$; list size N ; guess λ ; and perturbation length factor ε_0 .

Output: μ -approximate shortest non-zero vector of $\mathcal{L}(B)$

- 1: $S_0 = \emptyset$
 - 2: $\theta \leftarrow \min\left(\arccos \frac{1}{2\gamma}, 2 \arcsin \frac{\gamma}{2}\right)$
 - 3: $\varepsilon = \varepsilon_0 \lambda$
 - 4: **for** $i = 1 \dots N$ **do**
 - 5: Draw $\mathbf{x}$ uniformly at random from $\mathcal{B}_n(\varepsilon)$
 - 6: Add $\text{Sample}(B, \mathbf{x})$ to list S_0 ▷ Run Algorithm 2
 - 7: **end for**
 - 8: $R \leftarrow \max_i \|\mathbf{b}_i\|$
 - 9: $j \leftarrow 1$
 - 10: **while** $R > R_0$ **do**
 - 11: $S_j \leftarrow \text{Sieve}(R, S_{j-1}, \gamma, \phi)$ ▷ Run Algorithm 1
 - 12: $R \leftarrow \gamma R + \varepsilon$
 - 13: $j \leftarrow j + 1$
 - 14: **end while**
 - 15: **return** any non-zero vector in the first component of S_{j-1} , fail if none exist.
-

Every instance of the while loop in Algorithm 3 uses the set S_{j-1} to build a set of shorter vector pairs S_j . It follows from Lemma 2.14 that at the end of each while loop the set S_j contains vector pairs such that the second element (perturbed vectors) is of length at most R . We have the following lemma about the number of iterations of the while loop.

Lemma 3.3. *When Algorithm 3 is run with a basis such that $\|\mathbf{b}_i\| \leq 2^{3n}\lambda$, a constant (in n) $\varepsilon_0 > 0$, $\gamma \in (1/2, 1)$, a guess $\lambda_1 \leq \lambda \leq \lambda_1(\mathcal{L}(B)) (1 + \frac{1}{n})$, and $R_0 \geq \frac{\varepsilon_0 \lambda}{1-\gamma} (1 + \frac{1}{n})$, it terminates after $\mathcal{O}(n)$ executions of the while loop.*

We now have all the tools to analyze the time and space complexity of Algorithm 3. However, correctness of the algorithm needs more care. We have tried to make this argument as simple and modular as possible (at the expense of some looser low-order terms).

We will show in Theorem 3.7 that for a constant ε_0 and θ , if the final set of vector pairs S_{j-1} at the end of Algorithm 3 contains a non-zero vector, then that lattice vector is a constant factor approximation to a shortest non-zero vector in $\mathcal{L}(B)$. Therefore, to show correctness of our algorithm, it will be enough to show that the set S_{j-1} contains a non-zero lattice vector with high probability. To that end, we will use the perturbation trick from [AKS01]. We define *good perturbations*, and show that for any iteration j , for a *good pair* in S_j , in which the lattice vector $\mathbf{v}$ is generated from a good perturbation by Algorithm 2, the probability that this lattice vector is $\mathbf{0}$ is at most the probability that it is a shortest non-zero vector.

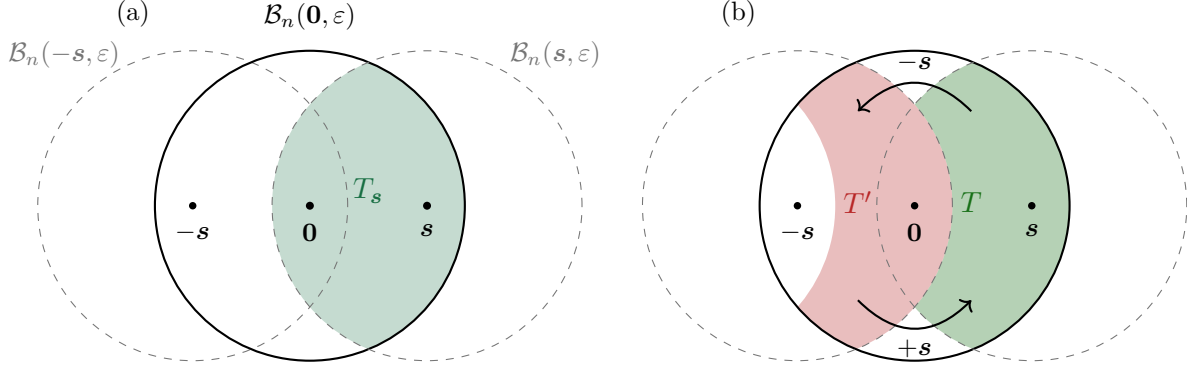


Figure 1: (a) The set $T_s = \mathcal{B}_n(\mathbf{0}, \varepsilon) \cap \mathcal{B}_n(\mathbf{s}, \varepsilon)$, together with the translated ball $\mathcal{B}_n(-\mathbf{s}, \varepsilon)$. (b) We define $T = T_s \setminus (T_s - \mathbf{s})$ and $T' = T - \mathbf{s}$. The sets T and T' are disjoint subsets of $\mathcal{B}_n(\mathbf{0}, \varepsilon)$ and are translations of one another.

Fix some shortest non-zero vector $\mathbf{s} \in \mathcal{L}(B)$, and define

$$T_s := \mathcal{B}_n(\mathbf{0}, \varepsilon) \cap \mathcal{B}_n(\mathbf{s}, \varepsilon) \quad \text{and} \quad T_{-\mathbf{s}} := \mathcal{B}_n(\mathbf{0}, \varepsilon) \cap \mathcal{B}_n(-\mathbf{s}, \varepsilon), \quad (3.2)$$

and (see Figure 1)

$$T := T_s \setminus (T_s - \mathbf{s}) \quad \text{and} \quad T' := T - \mathbf{s}. \quad (3.3)$$

Definition 3.4 (Good perturbations). *A perturbation $\mathbf{x} \in \mathcal{B}_n(\varepsilon)$ is called good when $\mathbf{x} \in T \cup T'$.*

Notice that T and T' are disjoint, and that

$$\begin{aligned} T' &= T - \mathbf{s} \\ &= T_s \setminus (T_s - \mathbf{s}) - \mathbf{s} \\ &= T_{-\mathbf{s}} \setminus (T_{-\mathbf{s}} - \mathbf{s}). \end{aligned}$$

If $\mathbf{v} \in \mathcal{L}$, we say that a pair $(\mathbf{v}, \mathbf{v} + \mathbf{x})$ is good if $\mathbf{x}$ is good. The following lemma gives a lower bound on the probability that any perturbation drawn uniformly at random from $\mathcal{B}(\varepsilon_0 \lambda)$ is good.

Lemma 3.5 (Good perturbation and good pair). *Consider $\varepsilon_0 > \frac{1}{2}$ and λ , where $\lambda_1(\mathcal{L}) \leq \lambda \leq (1 + \frac{1}{n})\lambda_1(\mathcal{L})$. For any perturbation $\mathbf{x}$ drawn uniformly at random from $\mathcal{B}_n(\varepsilon_0 \lambda)$, the probability that $\mathbf{x}$ is good satisfies*

$$\Pr_{\mathbf{x} \leftarrow \mathcal{B}_n(\varepsilon_0 \lambda)}[\mathbf{x} \in T \cup T'] \geq \frac{1}{N_G},$$

where $N_G < 2^{o(n)} \cdot (1 - 1/(4\varepsilon_0^2))^{-n/2}$.

Proof. Let $\varepsilon = \varepsilon_0 \lambda$. Notice that (see Figure 1)

$$\text{vol}_n(T) = \text{vol}_n(T_s) - \text{vol}_n(T_s \cap (T_s - \mathbf{s})) = \text{vol}_n(T_s) - \text{vol}_n(\mathcal{B}_n(\mathbf{s}, \varepsilon) \cap \mathcal{B}_n(-\mathbf{s}, \varepsilon)).$$

T_s is a lens with angular radius ϕ_1 such that $\cos \phi_1 = \frac{\lambda_1}{2\varepsilon}$, and similarly whenever $T_s \cap (T_s - \mathbf{s})$ is non-empty, it is a lens with angular radius ϕ_2 such that $\cos \phi_2 = \frac{\lambda_1}{\varepsilon}$. From Lemma 2.10 it follows that

$$\frac{\text{vol}_n(T_s \cap (T_s - \mathbf{s}))}{\text{vol}_n(T_s)} \leq 2^{-\Omega(n)}.$$

Therefore by [Lemma 2.10](#) we have that

$$\begin{aligned}\Pr_{\mathbf{x} \leftarrow \mathcal{B}_n(\varepsilon)}[\mathbf{x} \in T \cup T'] &= \frac{\text{vol}_n(T \cup T')}{\text{vol}_n(\mathcal{B}_n(\varepsilon))} \\ &\geq 2^{-o(n)} \left(1 - \frac{1}{4\varepsilon_0^2}\right)^{n/2},\end{aligned}$$

as claimed. $\square$

In the following, we use $X_i^{(0)} = (V_i^{(0)}, P_i^{(0)})$ to denote the (random variable corresponding to the) i th pair in S_0 , and use $X_i^{(j)}$ for its descendant in S_j (and similarly for V and P). Note that any pair used as a center in say the j th sieving step is discarded in that step, in which case we say that $X_i^{(j')} = \perp$ for all $j' > j$.

Lemma 3.6. *For any i and j , we have*

$$\Pr[V_i^{(j)} = \mathbf{0} \mid X_i^{(0)} \text{ is good}] \leq \Pr[V_i^{(j)} \in \{\pm \mathbf{s}\} \mid X_i^{(0)} \text{ is good}], \quad (3.4)$$

where the probability is over the randomness of the sampling process for perturbations in [Algorithm 3](#).

Proof. Recall T, T' from [Equation \(3.3\)](#). Define a function $g(\mathbf{x})$ over $\mathcal{B}_n(\varepsilon)$ as follows.

$$g(\mathbf{x}) := \begin{cases} \mathbf{x} + \mathbf{s}, & \mathbf{x} \in T' \\ \mathbf{x} - \mathbf{s}, & \mathbf{x} \in T \\ \mathbf{x}, & \text{otherwise} \end{cases} \quad (3.5)$$

Notice that g is a bijection from $\mathcal{B}_n(\varepsilon)$ to itself that maps the set T to T' and the set T' to T . Consider an algorithm Sieve2 that takes an additional input a shortest non-zero vector $\mathbf{s}$. This algorithm is almost identical to [Algorithm 3](#) except for one change. After sampling $\mathbf{x}$ in [Line 3](#) of [Algorithm 3](#), we toss a fair coin, and if the coin lands up HEADS then the algorithm replaces $\mathbf{x}$ by $g(\mathbf{x})$.

Since g is a bijection, $g(\mathbf{x})$ is identically distributed as $\mathbf{x}$. Moreover, the N perturbations remain independent after these replacements. Hence the entire execution, and therefore the output, of [Algorithm 3](#) and Sieve2 have the same distribution. It suffices to prove the required statement for Sieve2.

Consider a vector $\mathbf{x}$ sampled in [line 5](#) of [Algorithm 3](#). Without the replacement, the corresponding pair added to S_0 is $(\mathbf{v}, \mathbf{p})$, where

$$\mathbf{p} = \mathbf{x} \bmod \mathcal{P}(B), \quad \mathbf{v} = \mathbf{p} - \mathbf{x}.$$

For each $j \geq 0$, the call to sieving procedure with input S_j does the following. Each pair $(\mathbf{v}, \mathbf{p})$ is replaced by

$$(\mathbf{v} - \mathbf{v}_c, \mathbf{p} - \mathbf{v}_c)$$

for some chosen center $(\mathbf{v}_c, \mathbf{p}_c)$ and added to S_{j+1} (we can take $\mathbf{v}_c = \mathbf{0}$ whenever $\|\mathbf{p}\| \leq \gamma R + \varepsilon$), or placed in one or more buckets as a center, in which case it does not appear in S_{j+1} .

Thus, after j iterations, a pair $(\mathbf{v}, \mathbf{p})$ inserted in S_0 has either been discarded in one of these iterations, or has been replaced by $(\mathbf{v} - \mathbf{u}^*, \mathbf{p} - \mathbf{u}^*)$ for some lattice vector $\mathbf{u}^*$, which is the sum of the center vectors subtracted in the j iterations.

Now consider how this execution of the algorithm changes for Sieve2. Fix all sampled perturbations, all randomness used by [Algorithm 1](#), and all coin tosses except the coin belonging to $\mathbf{x}$. We compare the two executions corresponding to the two outcomes of this coin.

Since $g(\mathbf{x}) - \mathbf{x} \in \mathcal{L}(B)$, we have

$$g(\mathbf{x}) \bmod \mathcal{P}(B) = \mathbf{x} \bmod \mathcal{P}(B) .$$

Thus, replacing $\mathbf{x}$ by $g(\mathbf{x})$ changes its initial pair to

$$(\mathbf{v} + \mathbf{x} - g(\mathbf{x}), \mathbf{p}) ,$$

while leaving all other pairs unchanged.

Until this pair becomes a center, changing the first component does not affect any choice made by [Algorithm 1](#). The choice of a center depends only on the second component $\mathbf{p}$ and the pairs already present in the buckets. Thus, it is reduced using the same centers in both executions, or becomes a center in both.

Repeating this argument, we see that the pair survives to S_j either in both executions or in neither. If it survives, $\mathbf{u}^*$ remains the same, and the two possible pairs that we get are

$$(\mathbf{v} - \mathbf{u}^*, \mathbf{p} - \mathbf{u}^*)$$

and

$$(\mathbf{v} - \mathbf{u}^* + \mathbf{x} - g(\mathbf{x}), \mathbf{p} - \mathbf{u}^*) .$$

Equivalently, we could have postponed this pair's coin toss until it first became a center, or until we want to look at the corresponding pair in S_j .

Suppose that the pair survives until S_j and that $\mathbf{x}$ is a good vector. Then the two possible lattice components differ by $\mathbf{x} - g(\mathbf{x})$ which is $\pm \mathbf{s}$, and each occurs with probability $1/2$. Thus, whenever one is zero, the other belongs to $\pm \mathbf{s}$.

Averaging over the randomness fixed above gives the desired result. $\square$

Theorem 3.7. *Let $\mu > 1$ and $\varepsilon_0 > 1/2$ be constants such that*

$$\varepsilon_0 < \frac{\mu}{(2 + \frac{1}{n})(1 + \frac{1}{n})}, \quad \gamma := 1 - \frac{\varepsilon_0}{\mu - \varepsilon_0(1 + \frac{1}{n})} \left(1 + \frac{1}{n}\right)^2 ,$$

and $\theta = \min\left(\arccos \frac{1}{2\gamma}, 2 \arcsin \frac{\gamma}{2}\right)$. Let $\phi \in (\theta/2, \pi/2)$ and $N := 100n^2 N_G A(n, \theta)$. There exists an algorithm that solves μ -SVP with high probability in time

$$\mathcal{T}_{\mu\text{SVP}} \leq 2^{o(n)} N N_G p(n, \theta, \phi)^{-1} \left(p(n, \phi) + \sup_{\theta \leq \alpha \leq \pi/2} A(n, \theta, \alpha) \cdot p(n, \alpha, \phi) \right)$$

and space

$$\mathcal{S}_{\mu\text{SVP}} \leq 2^{o(n)} N p(n, \theta, \phi)^{-1} p(n, \phi) ,$$

where $N_G < 2^{o(n)}(1 - 1/(4\varepsilon_0^2))^{-n/2} \approx e^{n/(2\mu^2)}$ as in [Lemma 3.5](#).

Proof. The algorithm runs [Algorithm 3](#), say, $n^2 N_G$ times with these parameters and a truncated LLL-reduced basis as described in [Lemma 2.4](#) and with a guess λ as satisfying $\lambda_1(\mathcal{L}) \leq \lambda \leq (1 + 1/n)\lambda_1(\mathcal{L})$ as in [Lemma 2.3](#), and $R_0 := \varepsilon_0 \lambda / (1 - \gamma) \cdot (1 + 1/n)$. It suffices to show that one run of this algorithm has the desired complexity (up to a factor of N_G) and correctly finds an approximate shortest non-zero with probability $\Omega(1/(nN_G))$.

Correctness. By [Lemma 3.3](#), the number of iterations of the while loop in [Algorithm 3](#) is $t := \mathcal{O}(n)$. We claim that, all of the lattice vectors in S_t have length at most $\mu \cdot \lambda_1(\mathcal{L})$. Indeed, for any $(\mathbf{v}, \mathbf{p}) \in S_t$,

$$\begin{aligned} \|\mathbf{v}\| &\leq R_0 + \varepsilon_0 \lambda \\ &= \frac{\varepsilon_0 \lambda}{1 - \gamma} \cdot \left(1 + \frac{1}{n}\right) + \varepsilon_0 \lambda \\ &\leq \mu \lambda_1, \end{aligned}$$

where the first inequality follows from [Item 3](#) of [Theorem 3.1](#).

Therefore, for correctness, it suffices to show that at least one of them is non-zero with probability $\Omega(1/(nN_G))$.

By [Lemma 3.5](#) and a simple Chernoff bound, the initial list S_0 contained at least $N/(2N_G)$ good pairs with high probability. Let us look at these pairs throughout the execution of the algorithm. In each iteration, the number of these pairs lost (because they become centers) in [Algorithm 1](#) is at most $A(n, \theta)$ with high probability, by [Item 2](#) of [Theorem 3.1](#). Therefore, by the end of the algorithm there are at least $N/(2N_G) - \mathcal{O}(nA(n, \theta)) \geq \Omega(N/N_G)$ surviving good pairs with high probability. It follows that the expected number of surviving good pairs is at least $N/(nN_G)$.

On the other hand, by [Lemma 3.6](#), we have that

$$\mathbb{E}[|\{i : V_i^{(t)} = \mathbf{0} \text{ and } X_i^{(0)} \text{ is good}\}|] \leq \mathbb{E}[|\{i : V_i^{(t)} \notin \{\mathbf{0}, \perp\} \text{ and } X_i^{(0)} \text{ is good}\}|].$$

Adding $\mathbb{E}[|\{i : V_i^{(t)} \notin \{\mathbf{0}, \perp\} \text{ and } X_i^{(0)} \text{ is good}\}|]$ to both sides, we see that

$$N/(nN_G) \leq \mathbb{E}[|\{i : V_i^{(t)} \neq \perp \text{ and } X_i^{(0)} \text{ is good}\}|] \leq 2 \mathbb{E}[|\{i : V_i^{(t)} \notin \{\mathbf{0}, \perp\} \text{ and } X_i^{(0)} \text{ is good}\}|]$$

It follows by Markov's inequality that there is at least one non-zero $V_i^{(t)}$ (that also happens to have $X_i^{(0)}$ good) with probability at least $1/(2nN_G)$, and the result follows.

Time and space complexity. The execution time is bounded by $\mathcal{O}(n)$ times the sum of the time it takes to generate N pairs in [Line 4](#), plus the time it takes to sieve in [Algorithm 3](#), where the latter value is $\mathcal{O}(n) \cdot \mathcal{T}_{HS}$ by [Lemma 3.3](#) and [Theorem 3.2](#), and is the dominant cost. Therefore, we have that

$$\mathcal{T}_{\mu\text{SVP}} \leq 2^{o(n)} N_G N p(n, \theta, \phi)^{-1} \left(p(n, \phi) + \sup_{\theta < \alpha < \pi/2} A(n, \theta, \alpha) \cdot p(n, \alpha, \phi) \right),$$

as claimed.

Moreover, the space consumed is proportional to the space consumed by the first execution of [Algorithm 1](#) (since the size of the successive lists can only decrease). Therefore,

$$\mathcal{S}_{\mu\text{SVP}} \leq N \cdot \text{poly}(n) + 2^{o(n)} p(n, \theta, \phi)^{-1} p(n, \phi) N \leq 2^{o(n)} p(n, \theta, \phi)^{-1} p(n, \phi) N,$$

as claimed, where in the last step we have used the fact that $p(n, \theta, \phi) \leq p(n, \phi)$ (as is clear from their definitions). $\square$

Recall the KL upper bound on the kissing number from [Theorem 2.6](#). We now consider large approximation factors $\mu \rightarrow \infty$, in which case $\theta \rightarrow \pi/3$ by the choice of θ in [Theorem 3.7](#). By optimizing (in ϕ) [Theorem 3.7](#) for time, we have the following upper bounds on the time and space complexity.

Corollary 3.8. *There is an algorithm that solves μ -SVP in time at most $2^{0.6078n+\varepsilon_\mu n+o(n)}$ and space at most $2^{0.6078n+\varepsilon_\mu n+o(n)}$, where ε_μ is a constant depending on μ with $\lim_{\mu \rightarrow \infty} \varepsilon_\mu = 0$.*

Proof Sketch. We optimize the complexity expressions in [Theorem 3.7](#) as $\mu \rightarrow \infty$ to find that they are minimized at $\phi \approx 49.1792^\circ$ with $\alpha \approx 64.3560^\circ$. $\square$

Under [Assumption 2.8](#) we also get the following corollary.

Corollary 3.9. *Suppose that [Assumption 2.8](#) is true. Then, there is an algorithm that solves μ -SVP in time at most $2^{0.3066n+\varepsilon_\mu n+o(n)}$ and space at most $2^{0.3066n+\varepsilon_\mu n+o(n)}$, where ε_μ is a constant depending on μ with $\lim_{\mu \rightarrow \infty} \varepsilon_\mu = 0$.*

Proof Sketch. We optimize the complexity expressions in [Theorem 3.7](#) using [Lemma 2.9](#) and [Equation \(2.3\)](#) under [Assumption 2.8](#), with $\mu \rightarrow \infty$, to find that they are minimized at $\phi \approx 58.1844^\circ$ with $\alpha \approx 68.5293^\circ$. $\square$

3.3 The quantum variant of our algorithm

Finally, we sketch how to modify our classical algorithm to obtain a faster quantum algorithm. As one might expect, the key change is to replace the brute-force search in Line 9 in [Algorithm 1](#) with Grover's search algorithm. We first show the quantum analogue of [Theorem 3.1](#) and sketch how to modify [Algorithm 1](#) and the proof of [Theorem 3.1](#) to obtain this quantum version.

Theorem 3.10 (Quantum Sieving Algorithm). *For $\gamma \in (1/2, 1)$, let $0 < \theta < \pi/2$ be as in [Lemma 2.14](#), $\theta/2 < \phi < \pi/2$ be a constant, and $n \in \mathbb{N}$. There exists a quantum algorithm that takes as an input a list $S \subseteq \mathcal{L} \times \mathcal{B}_n(R)$ of vector pairs of size $|S| \leq 2^{\mathcal{O}(n)}$, for some $R \in \mathbb{R}$. Let $\theta = \min\left(\arccos \frac{1}{2\gamma}, 2 \arcsin \frac{\gamma}{2}\right)$ and $\varepsilon := \max_{(\mathbf{v}, \mathbf{p}) \in S} \|\mathbf{v} - \mathbf{p}\|$. Then except with probability $\exp(-\Theta(n))$ over the choice of the good code Code, it outputs a list $S' \subset \mathcal{L} \times \mathcal{B}_n(\gamma R + \varepsilon)$, such that at the end of the algorithm*

1. Every distinct $P = (\mathbf{v}_1, \mathbf{p}_1), Q = (\mathbf{v}_2, \mathbf{p}_2) \in W$ is such that $d(\mathbf{p}_1, \mathbf{p}_2) \geq \theta$.
2. $|S'| \geq |S| - A(n, \theta)$.
3. Every pair $(\mathbf{v}', \mathbf{p}')$ in S' was either a pair in S , or of the form $(\mathbf{v} - \mathbf{v}_c, \mathbf{p} - \mathbf{v}_c)$, where $(\mathbf{v}_c, \mathbf{p}_c) \in W$, and $(\mathbf{v}, \mathbf{p}) \in S \setminus W$. Moreover $\|\mathbf{v}' - \mathbf{p}'\| \leq \varepsilon$.
4. With high probability over the choice of codes, the time complexity is

$$\mathcal{T}_{HS}^Q \leq 2^{o(n)} |S| p(n, \theta, \phi)^{-1} \left(p(n, \phi) + \sqrt{p(n, \phi) \sup_{\theta < \alpha < \pi/2} A(n, \theta, \alpha) \cdot p(n, \alpha, \phi)} \right).$$

5. With high probability over the choice of codes, the algorithm uses $\mathcal{S}_{HS}^Q$ bits of coherently accessible classical memory, where

$$\mathcal{S}_{HS}^Q \leq |S| + 2^{o(n)} p(n, \theta, \phi)^{-1} p(n, \phi) \min(|S|, A(n, \theta)).$$

Proof. The algorithm works identically to [Algorithm 1](#), except it uses coherently accessible classical RAM (c-QRAM) to build the buckets H_j 's. Moreover, in [Line 9](#) it uses the algorithm from [Theorem 2.1](#) to search for a reducing pair (with a failure probability n^{-Cn} for a large enough C). The proof of [Items 1 to 3](#) and [5](#) remains identical to the proof of [Theorems 3.1](#) and [3.2](#). Time complexity analysis follows similarly to the proof of [Theorem 3.2](#), except at [Equation \(3.1\)](#) we instead have

$$\mathcal{T}_{HS} \leq \sum_{(\mathbf{v}, \mathbf{p}) \in S} \mathcal{T}_C(\mathbf{p}) + \sum_{(\mathbf{v}, \mathbf{p}) \in S} \sum_{\mathbf{c} \in \text{Code}} 1[d(\mathbf{c}, \mathbf{p}) \leq \phi] \left(1 + \sqrt{\sum_{\substack{(\mathbf{v}', \mathbf{p}') \in T^* \\ d(\mathbf{p}, \mathbf{p}') \geq \theta}} 1[d(\mathbf{c}, \mathbf{p}') \leq \phi]} \right).$$

Notice that for fixed $\mathbf{p}$, since every codeword is marginally uniform, we have that

$$\begin{aligned} & \mathbb{E}_{\text{Code}} \left[1[d(\mathbf{c}, \mathbf{p}) \leq \phi] \sqrt{\sum_{\substack{(\mathbf{v}', \mathbf{p}') \in T^* \\ d(\mathbf{p}, \mathbf{p}') \geq \theta}} 1[d(\mathbf{c}, \mathbf{p}') \leq \phi]} \right] \\ & \leq \sqrt{\mathbb{E}_{\text{Code}} [1[d(\mathbf{c}, \mathbf{p}) \leq \phi]]} \cdot \sqrt{\mathbb{E}_{\text{Code}} \left[\sum_{\substack{(\mathbf{v}', \mathbf{p}') \in T^* \\ d(\mathbf{p}, \mathbf{p}') \geq \theta}} 1[d(\mathbf{c}, \mathbf{p}) \leq \phi] 1[d(\mathbf{c}, \mathbf{p}') \leq \phi] \right]} \\ & = \sqrt{p(n, \phi) \sum_{\substack{(\mathbf{v}', \mathbf{p}') \in T^* \\ d(\mathbf{p}, \mathbf{p}') \geq \theta}} p(n, d(\mathbf{p}, \mathbf{p}'), \phi)}, \end{aligned}$$

where we have used the Cauchy-Schwarz inequality for random variables X and Y with $\langle X, Y \rangle := \mathbb{E}[XY]$, and the rest of the proof follows identically. $\square$

By instantiating [Algorithm 3](#) with the quantum sieving procedure from [Theorem 3.10](#) instead of the classical sieving procedure in [Algorithm 1](#), we immediately obtain the following result.

Theorem 3.11. *Let $\mu > 1$ and $\varepsilon_0 > 1/2$ be constants such that*

$$\varepsilon_0 < \frac{\mu}{\left(2 + \frac{1}{n}\right) \left(1 + \frac{1}{n}\right)}, \quad \gamma := 1 - \frac{\varepsilon_0}{\mu - \varepsilon_0 \left(1 + \frac{1}{n}\right)} \left(1 + \frac{1}{n}\right)^2,$$

$\theta = \min\left(\arccos \frac{1}{2\gamma}, 2 \arcsin \frac{\gamma}{2}\right)$. Let $\phi \in (\theta/2, \pi/2)$ and $N := 100n^2 N_G A(n, \theta)$. There exists a quantum algorithm that solves μ -SVP with high probability in time

$$\mathcal{T}_{\mu\text{SVP}} \leq 2^{o(n)} N N_G p(n, \theta, \phi)^{-1} \left(p(n, \phi) + \sqrt{p(n, \phi) \cdot \sup_{\theta \leq \alpha \leq \pi/2} A(n, \theta, \alpha) \cdot p(n, \alpha, \phi)} \right)$$

and uses

$$S_{\mu\text{SVP}} \leq 2^{o(n)} N p(n, \theta, \phi)^{-1} p(n, \phi),$$

bits of coherently accessible classical memory, where $N_G < 2^{o(n)} (1 - 1/(4\varepsilon_0^2))^{-n/2} \approx e^{n/(2\mu^2)}$ as in [Lemma 3.5](#).

From this, we obtain the following corollaries.

Corollary 3.12. *The algorithm from [Theorem 3.11](#) solves μ -SVP in time at most $2^{0.5832n+\varepsilon_\mu n+o(n)}$ and uses at most $2^{0.4868n+\varepsilon_\mu n+o(n)}$ bits of coherently accessible classical memory, where ε_μ is a constant depending on μ with $\lim_{\mu \rightarrow \infty} \varepsilon_\mu = 0$.*

Proof Sketch. We optimize the complexity expressions in [Theorem 3.11](#) as $\mu \rightarrow \infty$ to find that they are minimized at $\phi \approx 59.9438^\circ$ with $\alpha \approx 75.5099^\circ$. $\square$

Corollary 3.13. *Suppose that [Assumption 2.8](#) is true. Then, the algorithm from [Theorem 3.11](#) solves μ -SVP in time at most $2^{0.3000n+\varepsilon_\mu n+o(n)}$ and uses at most $2^{0.2593n+\varepsilon_\mu n+o(n)}$ bits of coherently accessible classical memory, where ε_μ is a constant depending on μ with $\lim_{\mu \rightarrow \infty} \varepsilon_\mu = 0$.*

Proof Sketch. We optimize the complexity expressions in [Theorem 3.11](#) using [Lemma 2.9](#) and [Equation \(2.3\)](#) under [Assumption 2.8](#), with $\mu \rightarrow \infty$, to find that they are minimized at $\phi \approx 65.5094^\circ$ with $\alpha \approx 76.5923^\circ$. $\square$

Acknowledgments

This work was supported by NRF grant NRF-NRFI09-0005, NSF Grant No. CCF-2312296, a Packard Foundation Fellowship, and a generous gift from Google.

Statement on AI Use

Our algorithm was originally developed without use of AI tools. And, without use of AI tools, we initially had a proof that the algorithm ran in time roughly $2^{0.37n}$ under [Assumption 2.8](#). We then consulted GPT Astra, and the AI tool found our current improved analysis, which yields roughly $2^{0.3066n}$.

We also used AI tools for more minor tasks, such as balancing lower-order terms, creating scripts to find the optimal value of ϕ and the worst-case value of α , and proofreading.

References

- [ABB⁺23] Divesh Aggarwal, Huck Bennett, Zvika Brakerski, Alexander Golovnev, Rajendra Kumar, Zeyong Li, Spencer Peters, Noah Stephens-Davidowitz, and Vinod Vaikuntanathan. Lattice problems beyond polynomial time. In *STOC*, 2023. 15
- [ABD⁺21] Roberto Avanzi, Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber (version 3.02) – Submission to round 3 of the NIST post-quantum project. <https://pq-crystals.org/kyber/resources.shtml>, 2021. 1
- [ACK⁺21] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, Zeyong Li, and Noah Stephens-Davidowitz. Dimension-preserving reductions between SVP and CVP in different p -norms. In *SODA*, 2021. 3

- [ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the Shortest Vector Problem in 2^n time via discrete Gaussian sampling. In *STOC*, 2015. 2
- [AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from Learning with Errors. In *ASIACRYPT*, 2011. 1
- [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. 1
- [AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001. 2, 4, 6, 8, 20, 22
- [ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *CRYPTO*, 2016. 1
- [AS18a] Divesh Aggarwal and Noah Stephens-Davidowitz. (Gap/S)ETH hardness of SVP. In *STOC*, 2018. 2
- [AS18b] Divesh Aggarwal and Noah Stephens-Davidowitz. Just take the average! An embarrassingly simple 2^n -time algorithm for SVP (and CVP). In *SOSA*, 2018. 2
- [AUV19] Divesh Aggarwal, Bogdan Ursu, and Serge Vaudenay. Faster sieving algorithm for approximate SVP with constant approximation factors. [ePrint:2019/1028](https://arxiv.org/abs/2019.1028), 2019. 2, 3, 4, 5, 6, 8, 9
- [BBHT98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998. 12
- [BCSS23] Xavier Bonnetain, André Chailloux, André Schrottenloher, and Yixin Shen. Finding many collisions via reusable quantum walks. In *Eurocrypt*, 2023. 3
- [BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, 2016. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 16, 19
- [BDK⁺21] Shi Bai, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium (version 3.1) – Submission to round 3 of the NIST post-quantum project. <https://pq-crystals.org/dilithium/resources.shtml>, 2021. 1
- [BGG⁺14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014. 1
- [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. 7
- [BGJ15] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Paper 2015/522, 2015. 1
- [BM07] Alexander Barg and Oleg R. Musin. Codes in spherical caps. *Advances in Mathematics of Communications*, 1(1):131–149, 2007. 9, 11, 13, 17

- [BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, 2011. 1
- [BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, 2014. 1
- [CE03] Henry Cohn and Noam Elkies. New upper bounds on sphere packings i. *Annals of Mathematics*, 157(2):689–714, 2003. 13
- [DL26] Emmanouil Doulgerakis and Thijs Laarhoven. Smooth sailing through spherical shells: Provable random-lattice sieving in time $2^{0.292n}$. [arXiv:2609.37482v1](#), 2026. 11
- [EV20] Friedrich Eisenbrand and Moritz Venzin. Approximate CVP in Time $2^{0.802n}$. In *ESA*, 2020. 3
- [Gau59] Walter Gautschi. Some elementary inequalities relating to the gamma and incomplete gamma function. *Journal of Mathematics and Physics*, 38(1-4):77–81, 1959. 16
- [Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009. 1
- [GFH26] Yiming Gao, Yansong Feng, and Honggang Hu. Solving the Shortest Vector Problem in $2^{0.7314n+o(n)}$ time via discrete Gaussian sampling on superlattices. [ePrint:2026/1587](#), 2026. 2
- [GKP⁺13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013. 1
- [GN08] Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within Mordell’s inequality. In *STOC*, 2008. 1
- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008. 1
- [Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, 1996. 12
- [GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013. 1
- [Hha26a] Minki Hhan. Finding a Shortest Vector and more in $2^{n/2+o(n)}$ time using q -ary coset difference tree. [arXiv:2609.02764v1](#), 2026. 2, 3
- [Hha26b] Minki Hhan. Solving the Shortest Vector Problem in $2^{0.6039n}$ time via mid-point Hessian. [arXiv:2608.02478v2](#), 2026. 2
- [HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the Shortest and Closest Lattice Vector Problems. In *Coding and Cryptology*, pages 159–190. Springer, 2011. 12, 21
- [Kim26] Jiseung Kim. One discrete Gaussian sample in $2^{n/2+o(n)}$ time. [arXiv:2608.03220v1](#), 2026. 2
- [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. 3, 5, 10, 13
- [KL21] Elena Kirshanova and Thijs Laarhoven. Lower bounds on lattice sieving and information set decoding. In *CRYPTO*, 2021. 2

- [Kla26] Boaz Klartag. Lattice packing of spheres in high dimensions using a stochastically evolving ellipsoid. *Inventiones mathematicae*, 244:1251–1279, 2026. 11
- [Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, 2015. 1, 7
- [LdW15] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *Latincrypt*, 2015. 1, 7
- [Lev79] Vladimir Iosifovich Levenshtein. On bounds for packings in n -dimensional euclidean space. *Doklady Akademii Nauk SSSR*, 245(6):1299–1303, 1979. 13
- [Li11] S. Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian Journal of Mathematics & Statistics*, 4(1):66–70, 2011. 15
- [LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982. 12
- [LWXZ11] Mingjie Liu, Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. Shortest lattice vectors in the presence of gaps. *ePrint:2011/139*, 2011. 2, 3, 4, 5, 6, 8, 9
- [Muk21] Priyanka Mukhopadhyay. Faster provable sieving algorithms for the Shortest Vector Problem and the Closest Vector Problem on lattices in ℓ_p norm. *Algorithms*, 14:362, 2021. 6
- [MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the Shortest Vector Problem. In *SODA*, 2010. 1
- [Nat24a] National Institute of Standards and Technology. Module-lattice-based digital signature standard. Technical Report Federal Information Processing Standards Publications (FIPS) NIST FIPS 204, U.S. Department of Commerce, Washington, D.C., 2024. The version released on August 13, 2024. 1
- [Nat24b] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Technical Report Federal Information Processing Standards Publications (FIPS) NIST FIPS 203, U.S. Department of Commerce, Washington, D.C., 2024. 1
- [NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. 1, 5, 12
- [Ope26] OpenAI. Ten advances in mathematics and theoretical computer science, 2026. 5, 10, 11
- [Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016. 1
- [PS09] Xavier Pujol and Damien Stehlé. Solving the Shortest Lattice Vector Problem in time $2^{2.465n}$. *ePrint:2009/605*, 2009. 6, 8
- [Reg04] Oded Regev. Lattices in Computer Science. https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/, 2004. [Online; accessed 11-Jan-2025]. 12, 21
- [Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):Art. 34, 40, 2009. A preliminary version appeared in STOC 2005. 1

- [RV22] Thomas Rothvoss and Moritz Venzin. Approximate CVP in time $2^{0.802n}$ — now in any norm! In *Integer Programming and Combinatorial Optimization*, 2022. [3](#)
- [Sch87] C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53(2):201–224, June 1987. [1](#)
- [ST06] F. H. Stillinger and S. Torquato. New conjectural lower bounds on the optimal density of sphere packings. *Experimental Mathematics*, 15:307–331, 2006. [11](#)
- [SZ24] Naser Talebizadeh Sardari and Masoud Zargar. New upper bounds for spherical codes and packings. *Mathematische Annalen*, 389(4):3653–3703, Aug 2024. [13](#)
- [Tsa19] Rotem Tsabary. Fully secure attribute-based encryption for t-CNF from LWE. In *Crypto*, 2019. [1](#)
- [Ven13] Akshay Venkatesh. A note on sphere packings in high dimension. *International Mathematics Research Notices*, 2013:1628–1642, 2013. [11](#)
- [WLW15] Wei Wei, Mingjie Liu, and Xiaoyun Wang. Finding shortest lattice vectors in the presence of gaps. In *CT-RSA*, 2015. [2](#), [3](#), [4](#), [5](#), [6](#), [8](#), [9](#)
- [Zho19] Yunkun Zhou. Personal communication, 2019. [15](#)
- [ZWW26] Yang Zhou, Zhedong Wang, and Mingsheng Wang. Exact SVP on Haar-random lattices at the heuristic sieving exponents. [ePrint:2026/2271](#), 2026. [11](#)