Recursive lattice reduction—
A framework for finding short lattice vectors

July 14, 2023

Abstract

We propose a new framework called recursive lattice reduction for finding short non-zero vectors in a lattice or for finding dense sublattices of a lattice. At a high level, the framework works by recursively searching for dense sublattices of dense sublattices (or their duals) with progressively lower rank. Eventually, the procedure encounters a recursive call on a lattice $\mathcal{L}$ with relatively low rank k, at which point we simply use a known algorithm to find a shortest non-zero vector in $\mathcal{L}$.

We view this new framework as complementary to basis reduction algorithms, which similarly work to reduce an n-dimensional lattice problem with some approximation factor γ to a lower-dimensional exact lattice problem in some lower dimension k, with a tradeoff between γ, n, and k. Our framework provides an alternative and arguably simpler perspective, which in particular can be described without explicitly referencing any specific basis of the lattice, the Gram-Schmidt orthogonalization, or even projection (though, of course, concrete implementations of algorithms in this framework will likely make use of such things).

We present a number of specific instantiations of our framework to illustrate its usefulness. Our main concrete result is an efficient reduction that matches the tradeoff between γ, n, and k achieved by the best-known basis reduction algorithms (in terms of the Hermite factor, up to low-order terms) across all parameter regimes. In fact, this reduction also can be used to find dense *sublattices* with any rank ℓ satisfying $\min\{\ell, n-\ell\} \leq n-k+1$, using only an oracle for SVP (or even just Hermite SVP) in k dimensions, which is itself a novel result (as far as the authors know).

We also show an extremely simple reduction that achieves the same tradeoff (up to low order terms) in quasipolynomial time, and a reduction from the problem of finding dense sublattices of a high-dimensional lattice to the problem of finding dense sublattices of lower-dimensional lattices. Finally, we present an automated approach for searching for algorithms in this framework that achieve better approximations with fewer oracle calls.

Contents

1 Introduction

A lattice $\mathcal{L} \subset \mathbb{R}^d$ is the set of all integer linear combinations of n linearly independent basis vectors $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{R}^{d \times n}$, i.e.,

$$\mathcal{L} := \{z_1\mathbf{b}_1 + z_2\mathbf{b}_2 + \cdots + z_n\mathbf{b}_n \ : \ z_i \in \mathbb{Z}\} \ .$$

We call n the *rank* of the lattice and d the *dimension* (or sometimes *ambient dimension*). We often implicitly assume that $d = n$ (which is essentially without loss of generality, by identifying $\mathrm{span}(\mathcal{L})$ with $\mathbb{R}^n$).

The most important geometric quantities associated with a lattice $\mathcal{L} \subset \mathbb{R}^d$ are the length of a shortest non-zero lattice vector, $\lambda_1(\mathcal{L}) := \min_{\mathbf{y} \in \mathcal{L}_{\neq \mathbf{0}}} \|\mathbf{y}\|$, and the determinant, $\det(\mathcal{L}) := \det(\mathbf{B}^T\mathbf{B})^{1/2}$. (Here and throughout this paper, $\|\mathbf{x}\| := (x_1^2 + \cdots + x_d^2)^{1/2}$ means the Euclidean norm.) The determinant is best viewed as a measure of the *global density* of a lattice, with lattices with smaller determinant being more dense, while $\lambda_1(\mathcal{L})$ is similarly a measure of the *local density*. These two quantities are related by *Hermite's* constant, which is given by

$$\delta_n := \sup_{\mathcal{L} \subset \mathbb{R}^d \colon \mathrm{rank}(\mathcal{L}) = n} \lambda_1(\mathcal{L})^2 / \det(\mathcal{L})^{\frac{2}{n}} \ ,$$

This is a very well studied quantity. Indeed, Minkowski's celebrated first theorem tells us that $\delta_n \leq O(n)$, and the Minkowski-Hlawka theorem tells us that $\delta_n \geq \Omega(n)$. So, we know that $\delta_n = \Theta(n)$.

We are interested in the γ-Hermite Shortest Vector Problem (γ-HSVP) for $\gamma = \gamma(n) \geq \sqrt{\delta_n}$. The input in γ-HSVP is a basis for a lattice $\mathcal{L} \subset \mathbb{R}^n$, and the goal is to find a non-zero lattice vector $\mathbf{y} \in \mathcal{L}_{\neq \mathbf{0}}$ such that $\|\mathbf{y}\| \leq \gamma \cdot \det(\mathcal{L})^{1/n}$.[1] The parameter γ is called the *approximation factor* or sometimes the Hermite factor. Notice that a solution is guaranteed to exist if and only if $\gamma \geq \sqrt{\delta_n}$.

This problem is central to lattice-based cryptography, which is in the process of widespread deployment [NIS22]. In particular, the best known attacks on lattice-based cryptography essentially work via reduction to γ-HSVP for $\gamma = \mathrm{poly}(n)$. Thus, *precise* estimates of the time complexity of γ-HSVP are necessary for assessing the security of these schemes. (See, e.g., [ACD+18].)

Algorithms for γ-HSVP have a *very* rich history.[2] In the hardest possible case when $\gamma = \sqrt{\delta_n}$, the fastest known algorithm for HSVP runs in $2^{\Theta(n)}$-time [AKS01], with a long line of work improving on the constant in the exponent [PS09, MV10b, MV10a, ADRS15, ALS21] (and another long line of work on heuristic algorithms [NV08, Laa15, BDGL16], and yet another long line of work on algorithms that do not quite achieve $\gamma = \sqrt{\delta_n}$ but do come very close, e.g., achieving $\gamma \leq \sqrt{\delta_n} \cdot \mathrm{poly}(\log n)$ [LWXZ11, WLW15, AUV19, ALS21]). The case of larger γ (say $\gamma > n^{1/2+\varepsilon}$) is more relevant to the current work, and we discuss it in depth below.

1.1 Basis reduction algorithms

The first non-trivial[3] algorithm for γ-HSVP for larger $\gamma \gg \sqrt{\delta_n}$ was the celebrated LLL algorithm by Lenstra, Lenstra, and Lovász [LLL82]. Their algorithm runs in polynomial time and achieves

[1] In Section 2.5, we discuss the relationship between HSVP and SVP.

[2] Most of the algorithms that we list here were originally presented as algorithms for *the Shortest Vector Problem* (SVP). We are describing them as algorithms for HSVP, since this is what interests us (and, indeed, what is typically relevant to cryptographers). See Section 2.5 for more discussion about the relationship between SVP and HSVP.

[3] By specifying "non-trivial" algorithms for γ-HSVP for $\gamma \gg \sqrt{\delta_n}$ here and elsewhere, we mean to exclude algorithms that work by simply solving γ'-HSVP with $\gamma' \approx \sqrt{\delta_n}$ on the input lattice. E.g., it is "trivial" to solve, say,

an approximation factor of $\gamma = 2^{O(n)}$. Indeed, forty years later, essentially all known non-trivial algorithms for γ-HSVP for large $\gamma \gg \sqrt{\delta_n}$ still use the technique introduced by LLL: basis reduction. So, in some (imprecise) sense, basis reduction is the only known framework for solving approximate lattice problems, and the community has spent the past forty years generating a beautiful body of literature devoted to perfecting this technique [Sch87, SE94, GN08b, GN08a, HPS11, MW16]. (See [Wal20] for a recent popular survey.)

At a high level, basis reduction algorithms work to progressively find a shorter basis for the input lattice $\mathcal{L} \subset \mathbb{R}^n$ by solving exact SVP instances on carefully chosen lattices with rank $k \geq 2$, where $k < n$ is known as the block size. Since the fastest known algorithms for $\sqrt{\delta_k}$-HSVP in k dimensions run in time $2^{\Theta(k)}$ (ignoring lower-order terms), the specific relationship between the block size k and approximation factor γ is quite important. Basis reduction achieves a smooth tradeoff, yielding an efficient reduction from γ_{BR}-HSVP in n dimensions to $\sqrt{\delta_k}$-HSVP in k dimensions that achieves

$$\gamma_{\mathsf{BR}} \approx \delta_k^{\frac{n-1}{2(k-1)}} \approx k^{\frac{n}{2k}} \; .$$

See, e.g., [GN08a, MW16, ALNS20, ALS21, Wal20]. (For cryptography, we are mostly concerned with the case when k is rather large, e.g., $k = Cn$ for some not-too-small constant $C > 0$.)

In more detail (which is not actually necessary for understanding the rest of this paper), basis reduction algorithms work by attempting to find a "good" basis of the given lattice (and in particular, a basis whose first vector $\mathbf{b}_1$ is quite short) by manipulating the Gram-Schmidt orthogonalization of the basis, $\widetilde{\mathbf{b}}_1 := \mathbf{b}_1, \widetilde{\mathbf{b}}_2 := \Pi_{\{\mathbf{b}_1\}^\perp}(\mathbf{b}_2), \ldots, \widetilde{\mathbf{b}}_n := \Pi_{\{\mathbf{b}_1,\ldots,\mathbf{b}_{n-1}\}^\perp}(\mathbf{b}_n)$, where $\Pi_{\{\mathbf{b}_1,\ldots,\mathbf{b}_i\}^\perp}$ represents projection onto the subspace orthogonal to $\mathbf{b}_1, \ldots, \mathbf{b}_i$. The idea behind all basis reduction algorithms is to make earlier Gram-Schmidt vectors $\widetilde{\mathbf{b}}_i$ shorter at the expense of making later Gram-Schmidt vectors $\widetilde{\mathbf{b}}_j$ for $j > i$ longer.[4] In particular, one hopes to make the first Gram-Schmidt vector $\widetilde{\mathbf{b}}_1 = \mathbf{b}_1$ short, since it is actually a lattice vector (while all other Gram-Schmidt vectors are *projections* of lattice vectors, which are typically not themselves lattice vectors). To make $\widetilde{\mathbf{b}}_i$ shorter relative to later Gram-Schmidt vectors, basis reduction algorithms rely on the fact that $\widetilde{\mathbf{b}}_i$ is contained in a k-dimensional lattice whose determinant is $\|\widetilde{\mathbf{b}}_i\| \cdots \|\widetilde{\mathbf{b}}_{i+k-1}\|$. Using an oracle for $\sqrt{\delta_k}$-HSVP in k dimensions allows us to find a non-zero vector in this lattice that is short relative to the geometric mean of $\|\widetilde{\mathbf{b}}_{i+1}\|, \ldots, \|\widetilde{\mathbf{b}}_{i+k-1}\|$. We can then substitute that vector for $\widetilde{\mathbf{b}}_i$. By doing this many different times with $i = 1, \ldots, n-k+1$, we can hope to eventually guarantee that $\|\widetilde{\mathbf{b}}_1\|, \|\widetilde{\mathbf{b}}_2\|, \ldots, \|\widetilde{\mathbf{b}}_k\|$ are not that large relative to the determinant of the lattice. And, with one more call to an HSVP oracle on the lattice generated by $\mathbf{b}_1, \ldots, \mathbf{b}_k$ (whose determinant is $\|\widetilde{\mathbf{b}}_1\|\|\widetilde{\mathbf{b}}_2\| \cdots \|\widetilde{\mathbf{b}}_k\|$), we can find a relatively short lattice vector.

So, the high-level goal of basis reduction algorithms is to find a k-dimensional sublattice $\mathcal{L}' \subset \mathcal{L}$ with low determinant (the lattice generated by $\mathbf{b}_1, \ldots, \mathbf{b}_k$ above) and then to use a HSVP oracle in k dimensions to find a short non-zero vector in $\mathcal{L}'$ (though, to be clear, basis reduction algorithms are not typically described this way). However, the details matter quite a bit in basis reduction, and they get complicated quickly. As a result, analyzing these algorithms can be rather difficult, because the effect of each oracle call on the lengths of the different Gram-Schmidt vectors is difficult to control.

2^n-HSVP by simply running, say, the $\sqrt{\delta_n}$-HSVP algorithm from [AKS01] on the input lattice. Of course, we do not mean to suggest that the [AKS01] algorithm is trivial or that it is trivial to solve $\sqrt{\delta_n}$-HSVP!

[4]The product of the lengths of the Gram-Schmidt vectors is the determinant of the lattice, and therefore cannot be changed without changing the lattice. So, if one wishes to make earlier Gram-Schmidt vectors shorter, one *must* lengthen later Gram-Schmidt vectors.

In fact, the best basis reduction algorithms are *heuristic*, meaning that we do not know how to prove their correctness [CN11]. *And*, our best way of analyzing the behavior of these algorithms is via sophisticated computer simulations [GN08b, CN11].[5]

2 An overview of recursive lattice reduction

In this work, we present a new framework for reductions from γ-HSVP in n dimensions to $\sqrt{\delta_k}$-HSVP in k dimensions, which we call *recursive lattice reduction*.[6] This new framework maintains the high-level idea that, in order to find a short vector in a high-rank lattice, one should first find a dense lower-dimensional sublattice $\mathcal{L}' \subset \mathcal{L}$ and then find a short vector in $\mathcal{L}'$. And, our framework therefore is rather similar to basis reduction.

However, our framework is in some sense more flexible than basis reduction. E.g., we do not require that $\mathcal{L}'$ has rank k. And, we do not even require that the algorithm explicitly works with a basis—let alone the Gram-Schmidt orthogonalization. (This is why we are careful to call this recursive *lattice* reduction.) This independence from the specific representation of the lattice makes our reductions very easy to describe at a high level. (Of course, any actual concrete implementation of our reductions will have to represent the lattice in some way—presumably with a basis, and perhaps even using the Gram-Schmidt orthogonalization. But, at a high level our framework is agnostic to this, and we instead view the specific choice of representation as an implementation detail.)

Our basic framework works as follows. Our reductions $\mathcal{A}(\mathcal{L}, \mathsf{aux})$ take as input a lattice $\mathcal{L}$ and some simple auxiliary information aux. If the lattice $\mathcal{L}$ has sufficiently small rank $n \le k$, then we of course use an oracle call to find a short vector. Otherwise, the reduction makes (at least) two recursive calls. The first recursive call is used to find a sublattice $\mathcal{L}' \subset \mathcal{L}$ with rank less than n and relatively small determinant. The next recursive call is simply $\mathcal{A}(\mathcal{L}', \mathsf{aux}')$.

In our instantiations, we use the auxiliary input aux to do two things: (1) to keep track of the rank of the sublattice that we are currently looking for; and (2) to control the running time of the reduction (and its subsequent recursive calls).

As we will see, this is a sufficiently modular and simple paradigm to allow for computer-aided search of the space of reductions in this framework. For example, one can write a simple computer program that takes as input a dimension n, block size k, and a bound T on the number of allowed oracle calls. The program will output a description of a reduction from γ-HSVP in n dimensions to $\sqrt{\delta_k}$-HSVP in k dimensions, together with an approximation factor γ that the reduction is guaranteed to achieve. Furthermore, the value of γ will be optimal, in some (admittedly rather weak) sense.

[5]Gama and Nguyen's elegant slide reduction algorithm [GN08a] is a bit of an exception—a basis reduction algorithm that can be analyzed quite simply. However, its performance in practice still lags behind heuristic basis reduction algorithms whose behavior is not nearly so well understood. One can modify slide reduction to improve its performance [Wal21], but at the expense of losing the simple proof of correctness.

[6]The word "reduction" has two meanings here, as it often does in the literature on basis reduction. First, it simply means a "reduction" in the traditional computer science sense: an algorithm for one problem that requires access to an oracle that solves another problem. Second, it means "reduction" in a sense that is specific to the context of lattices; a "reduction" in this sense describes a procedure that slowly *reduces* the lengths of lattice vectors (or the density of sublattices). This double meaning can be rather confusing at times—e.g., "basis reduction algorithms" should more accurately be called "basis reduction reductions." Here, we are using the term rather ambiguously in analogy with basis reduction. It is an open question whether the authors will regret our use of this ambiguous terminology.

Below, we build up the framework slowly by first developing two natural (and novel) reductions in the framework, in Section 2.1 and Section 2.2. Then in Sections 2.2 and 2.3, we provide a high-level overview of the more sophisticated reductions whose details we leave for the sequel. Section 2.3 (and the corresponding formal details in Section 5) in particular shows a reduction from the problem of finding a dense sublattice in n dimensions to the problem of finding a short vector in k dimensions, which is itself a novel result (to the authors' knowledge).

We note that throughout this paper, our purpose is to illustrate what is possible to do with this framework, rather than to optimize parameters. We have therefore made little attempt to, e.g., choose optimal constants. And, we have resisted the urge to include optimizations that one absolutely would wish to include if one were to implement these algorithms. (Perhaps most prominently, the astute reader might notice that all of our reductions as described run the LLL algorithm many more times than is truly necessary—sometimes running the LLL algorithm up to n times on the same lattice!)

2.1 A first example

To illustrate the new framework, consider the following simple (and novel) idea for a reduction from γ-HSVP on a rank n lattice to $\sqrt{\delta_k}$-HSVP on rank k lattices. Indeed, we provide a complete description and analysis of this algorithm in this overview.

At a high level, the reduction takes as input a lattice with rank n and finds a sublattice $\mathcal{L}' \subset \mathcal{L}$ with rank $n-1$ and with relatively small determinant. The reduction then calls itself recursively on $\mathcal{L}'$, therefore finding a short non-zero vector in $\mathcal{L}'$, which is of course also a short non-zero vector in $\mathcal{L}$.

To make the above precise, we first explain how to find a dense sublattice $\mathcal{L}' \subset \mathcal{L}$ with rank $n-1$ of $\mathcal{L}$. We do this using duality and recursion. In particular, it is a basic fact that the *dual* $\mathcal{L}^*$ of a lattice $\mathcal{L}$ is itself a lattice with $\det(\mathcal{L}^*) = 1/\det(\mathcal{L})$ and with the property that every (primitive) sublattice $\mathcal{L}'$ of $\mathcal{L}$ with rank $n-1$ is simply the intersection $\mathcal{L}' = \mathcal{L} \cap \mathbf{w}^\perp$ of $\mathcal{L}$ with the subspace $\mathbf{w}^\perp$ orthogonal to some non-zero dual vector $\mathbf{w} \in \mathcal{L}^*$. (See Section 3.1 for a formal definition of the dual lattice $\mathcal{L}^*$ and of primitivity, and for a discussion of many related properties.) Furthermore, $\det(\mathcal{L}') = \|\mathbf{w}\| \det(\mathcal{L})$ (assuming that $\mathbf{w}$ is primitive). Therefore, finding a dense sublattice $\mathcal{L}' \subseteq \mathcal{L}$ with rank $n-1$ is *equivalent* to finding a short non-zero dual vector $\mathbf{w}$. (This basic fact is used quite a bit already in basis reduction algorithms, e.g., in [GN08a, MW16].)

So, to find such a dense sublattice $\mathcal{L}'$, it suffices to find a short vector in the dual. We would like to do this by simply calling the algorithm recursively on $\mathcal{L}^*$. However, this would obviously lead to an infinite loop! Indeed, to find a short vector in $\mathcal{L}$, a naive implementation of this procedure would attempt to find a short vector in $\mathcal{L}^*$ by attempting to find a short vector in $\mathcal{L}$, etc! The solution is to add a *depth parameter* τ as auxiliary input to the reduction. Intuitively, when the depth parameter is larger, the reduction takes more time but achieves a better approximation factor γ (i.e., finds shorter vectors). The hope is that the recursive call on $\mathcal{L}^*$ can afford a worse approximation factor, i.e., the short vector in $\mathcal{L}^*$ does not need to be quite as short as our final output vector. By making this recursive call with a lower depth parameter, we avoid the infinite loop.

To finish specifying the reduction, we need to define two base cases. In one base case, the input lattice has rank $n = k$, in which case the reduction simply uses its $\sqrt{\delta_k}$-HSVP oracle in dimension k to output a short lattice vector in $\mathcal{L}$. In the other base case, the depth parameter τ is zero, in which case the algorithm uses an efficient procedure such as the LLL algorithm to output a not-too-long vector in the lattice.

Here is pseudocode for the reduction $\mathcal{A}(\mathcal{L}, \tau)$ described above.

```
1. Base cases:
    (a) If $\tau = 0$, run LLL on $\mathcal{L}$ and output the resulting vector.
    (b) If $\mathrm{rank}(\mathcal{L}) = k$, use an oracle for $\sqrt{\delta_k}$-HSVP to output a short non-zero lattice vector in $\mathcal{L}$.
2. Compute $\mathbf{w} \leftarrow \mathcal{A}(\mathcal{L}^*, \tau - 1)$ and output $\mathbf{y} \leftarrow \mathcal{A}(\mathcal{L} \cap \mathbf{w}^\perp, \tau)$.
```

Notice how simple this reduction is! It can be described in three short lines of pseudocode, and it makes no mention of a basis or Gram-Schmidt orthogonalization. Of course, an actual implementation of this algorithm would need some way to represent the lattice $\mathcal{L}$ and some way to compute a representation of $\mathcal{L}' = \mathcal{L} \cap \mathbf{w}^\perp$, which might be best done with bases and the Gram-Schmidt orthogonalization. But, while bases and Gram-Schmidt vectors are fundamental primitives in any basis reduction algorithm, in this new framework, we view such things as low-level implementation details. This level of abstraction allows us to specify such algorithms remarkably succinctly, as above.

Analyzing this reduction is also relatively simple, and we therefore give a complete analysis here in the overview. Specifically, let $\gamma(n, \tau)$ be the approximation factor achieved by the above reduction when the input lattice has rank n and the depth parameter is τ. By definition we have

$$\begin{aligned}
\|\mathbf{y}\|/\det(\mathcal{L})^{\frac{1}{n}} &\le \gamma(n-1, \tau) \cdot \det(\mathcal{L}')^{\frac{1}{n-1}}/\det(\mathcal{L})^{\frac{1}{n}} \\
&= \gamma(n-1, \tau)\left(\|\mathbf{w}\|/\det(\mathcal{L}^*)^{\frac{1}{n}}\right)^{\frac{1}{n-1}} \\
&\le \gamma(n-1, \tau)\gamma(n, \tau-1)^{\frac{1}{n-1}} .
\end{aligned}$$

In other words, $\gamma(n, \tau)$ satisfies the recurrence

$$\gamma(n, \tau) \le \gamma(n-1, \tau)\gamma(n, \tau-1)^{\frac{1}{n-1}} ,$$

with base cases $\gamma(k, \tau) \le \sqrt{\delta_k}$ and, say, $\gamma(n, 0) \le 2^n$ (by LLL). (The fact that the $\gamma(n, \tau-1)$ term comes with such a small exponent in the recurrence relation intuitively explains why we can afford to use a lower depth parameter in the recursive call on $\mathcal{L}^*$.) A simple induction argument then shows that, e.g., $\gamma(n, \tau) \le \delta_k^{(n-1)/(2(k-1))} \cdot 2^{n^3/2^\tau}$.

This analysis is *much* simpler than the similar analysis for basis reduction algorithms, and for $\tau \gtrsim 3\log n$, it matches the approximation factor γ_{BR} achieved by basis reduction up to (insignificant) low-order terms (which are also present in the basis reduction algorithms, and which can be controlled by taking τ to be larger).

One can similarly analyze the number of oracle calls $T(n, \tau)$ made by the reduction (which is the same as the running time up to a polynomial factor, if we count each oracle call as a unit-cost operation) via the recurrence $T(n, \tau) = T(n, \tau-1) + T(n-1, \tau)$ with base cases $T(n, 0) = 0$ and $T(k, \tau) = 1$. It is easy to check that this recurrence is satisfied by the binomial coefficients $T(n, \tau) = \binom{n-k+\tau-1}{\tau-1} \approx n^\tau$. In particular, for $\tau \approx 3\log n$, we get a running time of $n^{\Theta(\log n)}$. This is already quite interesting, since in the context of cryptography a factor of $n^{\Theta(\log n)}$ in the running time is not particularly significant. But, it's not ideal, and we would certainly prefer to find a variant of this reduction that achieves essentially the same approximation factor with only polynomial running time (counting oracle calls as a unit-cost operation).

Fortunately, this is possible, as we describe below.

2.2 An efficient reduction to a harder problem (and from a harder problem)

Intuitively, the reason that the above reduction required superpolynomial time is because it produces a rather unbalanced binary tree of recursive calls with height $n-k \approx n$ when the input lattice $\mathcal{L}$ has rank n. We would of course prefer a more balanced tree with height $O(\log n)$. Of course, the reason that the depth of the tree is so large is because the sublattice $\mathcal{L} \cap \mathbf{w}^\perp$ has rank just one less than $\mathcal{L}$. I.e., we reduce the rank of the lattice by one in each step, and therefore it takes $n-k$ steps to get down to rank k. We would of course prefer to reduce the rank more quickly, perhaps reducing the rank by a constant factor at every step, or more accurately, reducing by a constant factor the difference $n-k$ between the current rank n and the rank k in which our oracle works.

So, suppose instead of finding a single short non-zero vector $\mathbf{w} \in \mathcal{L}^*$, we found a whole *dense sublattice* $\mathcal{L}' \subset \mathcal{L}^*$ with rank $\ell^* \geq 1$, i.e., a sublattice $\mathcal{L}'$ of the dual with relatively small determinant. Just like before, the intersection $\mathcal{L} \cap (\mathcal{L}')^\perp$ of $\mathcal{L}$ with the subspace orthogonal to $\mathcal{L}'$ is a sublattice of $\mathcal{L}$, now with rank $n - \ell^*$. Furthermore, we have $\det(\mathcal{L} \cap (\mathcal{L}')^\perp) = \det(\mathcal{L}')\det(\mathcal{L})$ (provided, again, that $\mathcal{L}'$ is primitive, which we may assume without loss of generality; see Section 3.1). So, finding a dense sublattice with rank $n - \ell^*$ in $\mathcal{L}$ is exactly equivalent to finding a dense sublattice with rank ℓ^* in $\mathcal{L}^*$.

We therefore generalize the above from γ-HSVP to the γ-Densest Sublattice Problem (γ-DSP). In this problem, the input is a basis for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and *also* a rank ℓ with $1 \leq \ell \leq n-1$. The goal is to find a sublattice $\mathcal{L}'$ of $\mathcal{L}$ with rank ℓ with $\det(\mathcal{L}') \leq \gamma \det(\mathcal{L})^{\ell/n}$. Notice that when $\ell = 1$, this is exactly γ-HSVP. However, the problem is still interesting for larger ℓ. (Indeed, γ-DSP has been studied quite a bit, e.g., in [DM13, LN14, Dad19, Wal21, LW23].) For example, it is relatively easy to see that the LLL algorithm solves γ-DSP with an approximation factor of $\gamma \leq 2^{\ell(n-\ell)}$. (See Sections 2.4 and 3.2 for more discussion of DSP.)

This leads naturally to the following idea. To solve a DSP instance $(\mathcal{L}, \ell)$ where $\mathcal{L}$ has rank n, we first recursively solve the DSP instance $(\mathcal{L}^*, \ell^*)$ for some ℓ^* (and, of course, lower depth parameter), receiving as output a dense dual sublattice $\mathcal{L}' \subset \mathcal{L}$ with rank ℓ^*. We then recursively solve the DSP instance $(\mathcal{L} \cap (\mathcal{L}')^\perp, \ell)$, where the lattice $\mathcal{L} \cap (\mathcal{L}')^\perp$ has rank $n - \ell^*$. In particular, if we choose ℓ^* to be, say, $\lceil (n-k)/20 \rceil$, then the difference $n-k$ will shrink by a factor of at least $19/20$ at every step.

To make sense of this, we will need base cases like before for when the depth parameter τ drops to zero and when the rank n drops to k. In particular, when the depth parameter τ drops to zero, we simply apply LLL.

However, we also need to handle a new corner case: if $\ell > n - \ell^*$, then the above does not make sense, since we will be asking for a rank-ℓ sublattice of a lattice with rank less than ℓ! (Notice that, even if we start running this reduction with $\ell = 1$, some of the calls in our recursive tree will have much larger ℓ.) To fix this, we need some way to convert DSP instances with ℓ nearly as large as n to DSP instances with much smaller ℓ.

Of course, duality allows us to do this! That is, if ℓ is getting too large relative to n, then instead of directly finding a dense rank-ℓ sublattice of the primal lattice $\mathcal{L}$, we (recursively) find a dense sublattice $\mathcal{L}'$ of the dual $\mathcal{L}^*$ with rank $n - \ell$ and output $\mathcal{L} \cap (\mathcal{L}')^\perp$. We call this a *duality step*. In this way, we can always keep ℓ small enough. Specifically, in the below we simply apply this duality step whenever $\ell > n/2$, which is sufficient in this context. (Later, we are much more careful with when we do this.)

When n drops to k, we will call an oracle that solves γ-DSP in k dimensions with γ as small as possible. This is a major drawback of this reduction (which we will fix in the sequel) for two reasons.

First, DSP is a seemingly harder problem than HSVP, and the known algorithms that solve DSP are notably slower than the known algorithms for HSVP. Indeed, the fastest known algorithm for finding an exact densest sublattice with rank ℓ in n dimensions runs in time $\ell^{\Theta(n\ell)}$ [DM13], which in the worst case gives $n^{\Theta(n^2)}$. Second, it is not as clear how small we can take γ to be for our γ-DSP oracle. The analogue of Hermite's constant δ_n in this setting is *Rankin's constant*,

$$\delta_{n,\ell} := \sup_{\mathcal{L} \in \mathscr{L}(n)} \min_{\substack{\mathcal{L}' \subseteq \mathcal{L} \\ \mathrm{rank}(\mathcal{L}')=\ell}} \frac{\det(\mathcal{L}')^2}{\det(\mathcal{L})^{\frac{2\ell}{n}}} .$$

Rankin's constant is not nearly as well understood as Hermite's constant (except for $\delta_{n,n} = 1$ and $\delta_{n,1} = \delta_{n,n-1} = \delta_n$). Instead, $\delta_{n,\ell}$ is only known up to a constant factor in the exponent, $\delta_{n,\ell} = n^{\Theta(\ell(n-\ell)/n)}$ (though the best known upper and lower bounds on this constant factor in the exponent are not so far from each other [HS07]). However, a reasonable guess is that $\delta_{n,\ell} \approx n^{\ell(n-\ell)/(2n)}$. (See, e.g., [SW14] and the references therein.) For now, let us therefore simply suppose that we have access to an oracle that solves $k^{\ell(k-\ell)/(2k)}$-DSP oracle in k dimensions.[7]

With all of this out of the way, we can finally present our reduction $\mathcal{A}(\mathcal{L}, \ell, \tau)$, as below. Here, we leave ℓ^* unspecified, so that we actually give a family of algorithms depending on how ℓ^* is chosen (possibly depending on ℓ, n, k, and τ, though we choose it much more simply).

1. **Base cases:**
   (a) If $\ell > n/2$, output $\mathcal{L} \cap \mathcal{A}(\mathcal{L}^*, n-\ell, \tau)^\perp$, where $n := \mathrm{rank}(\mathcal{L})$.
   (b) If $n = k$, use an oracle for $k^{\ell(k-\ell)/(2k)}$-DSP to output a dense sublattice of $\mathcal{L}$ with rank ℓ.
   (c) If $\tau = 0$, run LLL on $\mathcal{L}$ and output the lattice generated by the first ℓ vectors of the resulting basis.
2. Compute $\mathcal{L}' \leftarrow \mathcal{A}(\mathcal{L}^*, \ell^*, \tau - 1)$ and output $\mathcal{L}'' \leftarrow \mathcal{A}(\mathcal{L} \cap (\mathcal{L}')^\perp, \ell, \tau)$.

Notice that the reduction is still quite simple.

The analysis of the approximation factor $\gamma(n, \ell, \tau)$ achieved by this algorithm is quite similar to the above. In particular, a similar argument to the above shows that $\gamma(n, \ell, \tau)$ satisfies the recurrence

$$\gamma(n, \ell, \tau) \le \gamma(n - \ell^*, \ell, \tau)\gamma(n, \ell^*, \tau - 1)^{\frac{\ell}{(n-\ell^*)}} , \tag{1}$$

and after plugging in the base cases a simple argument shows that, e.g.,

$$\gamma(n, \ell, \tau) \le k^{\frac{\ell(n-\ell)}{2k}} \cdot 2^{\frac{n^2 \ell(n-\ell)}{C^\tau}}$$

provided that, after applying duality, we maintain the invariant that $\ell + \ell^* \le (2 - C)n$ for some constant $1 < C < 2$. Specifically, if we take $\tau \ge \Omega(\log n)$ and $\ell = 1$, we get essentially the same approximation factor as before. (See Section 4 for a full analysis.)

[7] Because of the uncertainty in Rankin's constant, we do not know whether this is even possible—i.e., there might be k-dimensional lattices that do not even have a dense enough sublattice for this to work. So, in Section 4 we are more careful about this, but in this overview we proudly forge ahead under this simplifying assumption. We will anyway later show how to replace the DSP oracle with an HSVP oracle by changing the reduction slightly.

The benefit of this approach is, of course, that the running time is much better. In particular, notice that if we take $\ell^* \geq \beta(n-k)$ for some constant $0 < \beta < 1/2$, then the recursive calls made by the reduction form a binary tree with height bounded by

$$h := \lceil \log_{1/(1-\beta)}(n) \rceil + \tau = O(\log n) + \tau \ ,$$

so that the running time of the reduction is bounded by $2^h \leq 2^\tau \cdot \mathrm{poly}(n)$ (again, counting oracle calls as unit cost). Taking $\tau = \Theta(\log n)$ gives a reduction that achieves essentially the same approximation factor as before but now runs in polynomial time.

2.3 Getting an efficient reduction (from either HSVP or DSP) to HSVP! And, letting a computer find reductions for us.

The reduction described in the previous section does not achieve what we are truly after, since it requires an oracle for DSP, rather than HSVP (and since, in order to get the claimed approximation factor, it relies on an unproven conjecture about Rankin's constant). However, we have a lot of flexibility that we have not yet exploited. In particular, we can be more careful about our choice of ℓ^*, *and* more importantly, we can try to be clever about when we apply the duality step. Perhaps a more clever set of choices here will allow to achieve a good approximation factor with a good running time while simultaneously managing to always have $\ell = 1$ whenever we make a recursive call with rank k.

To see why this might be possible, suppose that at some step in the reduction there is a recursive call with $(n, \ell) = (2k+1, k)$. (Here, we are ignoring the depth parameter τ for simplicity.) If we choose $\ell^* = k$ here, then our reduction will make two recursive calls, with (n, ℓ) respectively equal to $(k+1, k)$ and $(2k+1, k)$. Since the latter recursive call is the same as where we started (though, of course, the depth parameter will be lower), we can safely ignore it—if we can make the reduction work from where we started, then we should be able to make it work here too. For the call with $(n, \ell) = (k+1, k)$, notice that after duality this becomes $(n, \ell) = (k+1, 1)$. We can then make a recursive call with $\ell^* = 1$ to get down to a base case of $(k, 1)$, which is what we want! (The other recursive call on the dual will produce another $(k+1, 1)$ node, which we have just shown how to handle.) Once we see that we can do this with $(n, \ell) = (2k+1, k)$, we can think about clever choices of ℓ^* that allow us to eventually get to this point from other values of (n, ℓ), etc. As it happens, this is possible if $\min\{\ell, n-\ell\} \leq n-k+1$, as we show in Section 5.

Things are a bit more subtle than we are suggesting here because one must still worry about keeping the approximation factor down. In particular, the recurrence Equation (1) satisfied by our approximation factor is not favorable when we take $\ell \approx \ell^* \approx n/2$. In general, we need to keep the exponent $\ell/(n-\ell^*)$ low. However, these issues can be overcome via careful choices of parameters. In particular, in Section 5 we show a concrete polynomial-time reduction from γ-HSVP in n dimensions to $\sqrt{\delta_k}$-HSVP in k dimensions that achieves an approximation factor of $\gamma = (1+o(1)) \cdot \delta_k^{(n-1)/(2(k-1))}$.

In fact, we achieve an approximation factor of $(1+o(1)) \cdot \delta_k^{\ell(n-\ell)/(2(k-1))} \cdot 2^{1/\mathrm{poly}(n)}$ for DSP whenever $1 \leq \min\{\ell, n-\ell\} \leq n-k+1$ (and not just the HSVP case, which corresponds to $\ell = 1$). (In particular, when $n \geq 2k$, this works for all ℓ.) And, notice that we do this with only an oracle for HSVP in k dimensions (and *not* a DSP oracle)! As far as we know, no such reduction appeared in prior work.[8]

On using automated search to find such reductions. However, for any fixed dimension n, block size k, and number of oracle calls C, we can also compute the "best possible" reduction of this form in the following sense. We can find a reduction that in C oracle calls provably achieves an approximation factor $\gamma(n, \ell, C)$ satisfying

$$\gamma(n,\ell,C) = \min\left\{\gamma(n, n-\ell, C),\ \min_{1\le \ell^*\le n-k}\ \min_{C^*\le C} \gamma(n-\ell^*, \ell, C-C^*)\gamma(n,\ell^*,C^*)^{\frac{\ell}{n-\ell^*}}\right\}, \qquad (2)$$

with base cases given by $\gamma(k, 1, C) = \sqrt{\delta_k}$ for $C \ge 1$ and $\gamma(n, \ell, C) = 2^{\ell(n-\ell)}$ if *either* $C = 0$ or $n = k$ and $\ell \notin \{1, n-1\}$ (i.e., the algorithm applies LLL if it runs out of oracle calls *or* it reaches dimension k but requires a dense sublattice with rank $\ell > 1$ rather than simply a short vector). Equation (2) formally captures the idea that the reduction may either apply a duality step (which explains the $\gamma(n, n-\ell, C)$ term) *or* choose the best possible rank ℓ^* of a dual sublattice to find and the best possible number of oracle calls C^* to allocate to finding a dense dual sublattice.

Indeed, using a simple dynamic programming algorithm, one can compute $\gamma(n, \ell, C)$ in time $\widetilde{O}(n^3C^2)$. For reasonable parameters, this running time is far less than the time required to instantiate the $\sqrt{\delta_k}$-HSVP oracle. If we restrict our algorithm to a sparse subset of possible choices for C^*, then one can also compute very good upper bounds on $\gamma(n, \ell, C)$ in time $n^3 \cdot \mathrm{poly}(\log n, \log C)$, which is quite practical even for quite large numbers. See Section 6.

From this perspective, our specific reduction is perhaps best viewed as an upper bound on the value of $\gamma(n, \ell, C)$ found by this more sophisticated automated procedure. Indeed, we discovered the specific approach used in Section 5 by first studying the reductions returned by an automated procedure!

See Section 6 for discussions of how to further generalize this.

2.4 Some more context, and the relationship with prior work

Of course, our recursive lattice reduction framework shares much in common with basis reduction. At a very high level, both paradigms work to find a short non-zero vector in a lattice $\mathcal{L}$ by first finding a dense lower-rank sublattice $\mathcal{L}' \subset \mathcal{L}$ and then finding a short non-zero vector in $\mathcal{L}'$. Indeed, one could argue that recursive lattice reduction is essentially a repackaging of basis reduction (though we do not know of a direct way to view our framework in terms of basis reduction—or vice versa, see Section 7). We instead think of it as a related but novel framework, which we hope will lead to a better understanding of lattice problems (and therefore to a better understanding of the security of lattice-based cryptography).

Here, we simply describe some of the basis reduction algorithms that use particularly similar ideas to those in this work. (Of course, a full survey of the extensive literature on basis reduction is far beyond the scope of this work. As we discuss in Section 7, we expect that additional ideas used in basis reduction should be useful in our new framework as well.)

[8]Of course, essentially every basis reduction algorithm solves some form of DSP for some fixed rank ℓ, typically $\ell = k$, but this is of course very different from solving DSP for many different choices of ℓ. There is also a *heuristic* reduction in [Wal21, LW23]). And, a reduction from DSP to (H)SVP with a worse approximation factor is implicit in [HS07].

The technique of moving between the primal and the dual, and in particular of doing so to exploit the identity $\det(\mathcal{L} \cap (\mathcal{L}')^{\perp}) = \det(\mathcal{L})\det(\mathcal{L}')$ for a (primitive) dual sublattice $\mathcal{L}' \subset \mathcal{L}^*$, is well known in the basis reduction literature. In particular, the special case when $\mathcal{L}'$ is generated by a single vector plays a prominent role in both Gama and Nguyen's celebrated slide reduction algorithm [GN08a] and Micciancio and Walter's beautiful self-dual BKZ algorithm [MW16]. Indeed, both of these algorithms repeatedly use this identity in order to use an oracle for HSVP in k dimensions to find a dense rank-$(k-1)$ sublattice in some k-dimensional lattice (where the k-dimensional lattice is itself some projection of a sublattice of the input lattice $\mathcal{L}$). (Of course, in both cases, the algorithms are described in terms of the behavior of the Gram-Schmidt vectors, and not directly in terms of (projections of) sublattices. One can also argue that all basis reduction algorithms implicitly use this identity, via the identity $\Pi_{(\mathcal{L}')^{\perp}}(\mathcal{L})^* = \mathcal{L}^* \cap (\mathcal{L}')^{\perp}$.) The more general technique when $\mathcal{L}'$ has higher rank has also been used in basis reduction algorithms, including algorithms that explicitly work to find dense sublattices of a rank-n lattice using an oracle to find dense sublattices of lower-dimensional lattices [LN14, LW23], as well as algorithms that use an oracle for finding dense sublattices of low-rank lattices in order to eventually find a short lattice vector in a rank-n lattice [Wal21, LW23].

DSP, the computational problem of finding dense sublattices of a given rank ℓ, has been the subject of a lot of work in the basis reduction literature. Li and Nguyen showed how to generalize Gama and Nguyen's slide reduction algorithm [LN14] to reduce γ-DSP with parameters n and $\ell \le k$ to $\sqrt{\delta_{k,\ell}}$-DSP with parameters k and ℓ, where $\gamma \approx \delta_{k,\ell}^{(n-\ell)/(2(k-\ell))}$. This is essentially the same high-level result that we achieve in our DSP-to-DSP reduction (described at a high level in Section 2.2 and in detail in Section 4), except that (1) they require that $\ell \le k$ and that $n \bmod k$ is divisible by ℓ; (2) they only require a DSP oracle that finds rank-ℓ sublattices of rank-k lattices, while we require one that finds rank-ℓ' sublattices for all $1 \le \ell' < k$ (and, relatedly, their approximation factor depends only on $\delta_{k,\ell}$, while ours depends on $\delta_{k,\ell'}$ for all ℓ'). Walter [Wal21] and Li and Walter [LW23] later showed variants of the algorithm with better performance (at least heuristically). The same works also gave heuristic arguments that suggested that using certain (H)SVP algorithms could be used in place of the DSP oracle, at the price of slightly increasing the approximation factor, and that this technique of finding dense sublattices using an (H)SVP algorithm in k dimensions can even be favorably used for solving HSVP in higher dimensions. This latter result is (roughly) analogous to our main reduction, which similarly uses DSP as an intermediate problem in a reduction from HSVP in n dimensions via reduction to HSVP in $k < n$ dimensions.

There are also algorithms for DSP that do not really use basis reduction. Dadush and Micciancio gave an $\ell^{O(n\ell)}$-time algorithm for finding the *exact* densest sublattice with rank ℓ [DM13]. And, Dadush gave an elegant $2^{O(n)}$ time algorithm that finds a sublattice $\mathcal{L}' \subseteq \mathcal{L}$ with the property that $\det(\mathcal{L}')^{1/\mathrm{rank}(\mathcal{L}')}$ is within a polylogarithmic factor of the minimum possible [Dad19]. (Notice that this algorithm does not allow us to choose the rank ℓ of the resulting sublattice. E.g, for some input lattices $\mathcal{L}$, the algorithm might simply return $\mathcal{L}$ itself. In particular, it is not clear how to make use of this algorithm as an oracle in either our framework or in basis reduction.)

Finally, we note that recursive approaches for basis reduction algorithms (specifically, LLL-like algorithms with block size $k = 2$) have been considered in the literature, for example, [NS16, KEF21, RH23]. One might reasonably expect that our "recursive lattice reduction" framework is quite similar to these "lattice basis reduction algorithms that use recursion," but in fact the use of recursion in these algorithms is fundamentally different from that in our framework. The algorithms from prior work are still largely iterative, and in particular still aim to iteratively improve the quality of a basis at each step. The use of recursion in these works is mainly concerned with minimizing the precision necessary in the Gram-Schmidt computations, and the resulting pattern of oracle calls is quite similar to the pattern in purely iterative basis reduction algorithms. Furthermore, all of these works focus on block size $k = 2$ (while our framework is not particularly interesting for block size $k = 2$).

2.5 On SVP vs. HSVP

Our framework solves HSVP, i.e., the problem of finding a short non-zero vector in a lattice $\mathcal{L}$ *relative to the (normalized) determinant* $\det(\mathcal{L})^{1/n}$. One can also consider the γ-*approximate Shortest Vector Problem* (γ-SVP), in which the goal is to find a non-zero lattice vector $\mathbf{y} \in \mathcal{L}_{\neq \mathbf{0}}$ such that $\|\mathbf{y}\| \le \gamma\lambda_1(\mathcal{L})$, where $\lambda_1(\mathcal{L}) := \min_{\mathbf{x}\in\mathcal{L}_{\neq \mathbf{0}}} \|\mathbf{x}\|$. I.e., γ-SVP asks for a non-zero lattice vector whose length is within a factor γ of the shortest in the lattice.

SVP is arguably the more natural problem, and is more discussed in the literature—at least the literature that is less directly concerned with cryptography. However, in cryptography, HSVP is typically the more important problem, and one typically judges the performance of an algorithm based on its Hermite factor, i.e., based on how well it solves HSVP. This is because in cryptography one typically encounters random lattices, which have $\lambda_1(\mathcal{L}) \approx \sqrt{n/(2\pi e)} \cdot \det(\mathcal{L})^{1/n} \approx \sqrt{\delta_n}\det(\mathcal{L})^{1/n}$. (However, this is not always the case. E.g., some cryptography uses lattices with planted short vectors, or planted dense sublattices.) Indeed, in heuristic analysis of lattice algorithms, one often assumes not only that the input lattice $\mathcal{L}$ satisfies $\lambda_1(\mathcal{L}) \approx \sqrt{n/(2\pi e)} \cdot \det(\mathcal{L})^{1/n}$, but *also* that this holds for all lattices $\mathcal{L}'$ encountered by the algorithm's subroutines.

There seems to be some consensus among experts that γ-HSVP for nearly minimal $\gamma \approx \sqrt{\delta_n}$ is likely to be more-or-less as hard to solve as γ'-SVP for γ' not much larger than one, in the sense that the fastest algorithms for γ'-SVP should be as fast as the fastest algorithms for γ-HSVP for such small values of γ, γ'.[9] But, the situation for larger approximation factors is less clear. Notice that there is a trivial reduction from $\gamma\sqrt{\delta_n}$-HSVP to γ-SVP. Furthermore, there is a (non-trivial) reduction due to Lovász from γ^2-SVP to γ-HSVP. So, our reductions can be composed with Lovász's to achieve reductions that solve γ^2-SVP (granted, at the expense of a factor of n in the number of oracle calls, since Lovász's reduction requires n calls to a γ-HSVP oracle). However, many basis reduction algorithms that solve γ_{BR}-HSVP can also be used to solve $\gamma_{\mathsf{BR}}^2/\sqrt{\delta_k}$-SVP in essentially the same running time, provided that the k-dimensional $\sqrt{\delta_k}$-HSVP oracle is replaced by a k-dimensional exact SVP oracle (which is anyway what is used in practice).

So, while we essentially match the Hermite factor γ_{BR} of basis reduction, we do not know how to use our framework to match the approximation factor $\gamma_{\mathsf{BR}}^2/\sqrt{\delta_k}$ that can be achieved for SVP. (We do not even know how to achieve an approximation factor of γ_{BR}^2 for SVP without increasing the number of oracle calls by a factor of $O(n)$.) A similar situation holds for the self-dual BKZ algorithm [MW16] (i.e., self-dual BKZ solves γ_{BR}-HSVP but is not known to solve $\gamma_{\mathsf{BR}}^2/\sqrt{\delta_k}$-SVP).

The same story more-or-less applies for DSP as well. In particular, we describe DSP as the problem of finding a dense sublattice that is dense *relative to the determinant of the input lattice*. But, one can also consider the version of DSP in which the goal is to find a dense sublattice of rank ℓ whose determinant is small *relative to the minimum possible*. If one wishes to distinguish

[9] However, currently the fastest known algorithm with proven correctness for $\sqrt{\delta_n} \cdot \mathrm{poly}(\log n)$-HSVP runs in time $2^{n/2+o(n)}$ [ALS21], while the fastest known algorithm with proven correctness for $\mathrm{poly}(\log n)$-SVP (or even $O(1)$-SVP) runs in time $2^{0.802n}$ [LWXZ11, WLW15, AUV19]. This seems to be more about our proof techniques than the inherent difficulty of the two problems, however.

between the two versions, one might use "RankinDSP" to refer to the version in which the density is relative to the determinant.

3 Preliminaries

Given a lattice $\mathcal{L} \subseteq \mathbb{R}^d$, its *dual* is $\mathcal{L}^* := \{\mathbf{w} \in \mathrm{span}(\mathcal{L}) : \forall \mathbf{y} \in \mathcal{L}, \langle \mathbf{w}, \mathbf{y} \rangle \in \mathbb{Z}\}$. Although this is somewhat nonstandard, it will be useful to generalize this definition to arbitrary sets $S \subseteq \mathbb{R}^d$ in the following natural way:

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

We say that a sublattice $\mathcal{L}'$ of a lattice $\mathcal{L}$ is *primitive* if $\mathcal{L}' = \mathcal{L} \cap \mathrm{span}(\mathcal{L}')$. And, we call a vector $\mathbf{y} \in \mathcal{L}$ primitive if the sublattice generated by the vector is primitive.

3.1 Duality and sublattices

Central to our results is a notion of duality for sublattices. Specifically, if $\mathcal{L}'$ is a sublattice of $\mathcal{L}$, then $\mathcal{L}^* \cap (\mathcal{L}')^\perp$ is a sublattice of the dual lattice $\mathcal{L}^*$. Moreover, the *duality map* $(\mathcal{L}, \mathcal{L}') \mapsto (\mathcal{L}^*, \mathcal{L}^* \cap (\mathcal{L}')^\perp)$ is an involution on the set of (lattice, primitive sublattice) pairs, that preserves a natural notion of the relative density of $\mathcal{L}'$ in $\mathcal{L}$. In this section, we state and prove these properties of the duality map.

Lemma 3.1. *Suppose that $\mathcal{L}'$ of rank ℓ is a sublattice of $\mathcal{L}$ of rank n. Then $\Pi_{(\mathcal{L}')^\perp}(\mathcal{L})$ is a lattice with rank $n - \ell$, and*

$$\Pi_{(\mathcal{L}')^\perp}(\mathcal{L})^* = \mathcal{L}^* \cap (\mathcal{L}')^\perp.$$

Proof. We may assume without loss of generality that $\mathcal{L}$ is full-rank ($d = n$). If $\mathcal{L}$ is not full-rank, simply rotate $\mathcal{L}$ so that it is orthogonal to the last $d - n$ coordinates and drop these coordinates; the statement to prove is invariant under rotation.

To simplify the notation, we will write $\Pi = \Pi_{(\mathcal{L}')^\perp}$. First we will prove that $\Pi(\mathcal{L})^* = \mathcal{L}^* \cap (\mathcal{L}')^\perp$. Indeed, both sets are subsets of $(\mathcal{L}')^\perp$. But for any $\mathbf{w} \in (\mathcal{L}')^\perp$ and $\mathbf{y} \in \mathbb{R}^d$, $\langle \mathbf{w}, \Pi(\mathbf{y}) \rangle = \langle \Pi(\mathbf{w}), \mathbf{y} \rangle = \langle \mathbf{w}, \mathbf{y} \rangle$. It follows that for all $\mathbf{w} \in (\mathcal{L}')^\perp$, $\langle \mathbf{w}, \Pi(\mathbf{y}) \rangle$ is an integer simultaneously for all $\mathbf{y} \in \mathcal{L}$ if and only if $\mathbf{w} \in \mathcal{L}^*$. Thus

$$\Pi(\mathcal{L})^* = \mathrm{span}(\Pi(\mathcal{L})) \cap \left(\mathcal{L}^* \cap (\mathcal{L}')^\perp\right) .$$

But $\mathrm{span}(\Pi(\mathcal{L})) = (\mathcal{L}')^\perp$. Hence $\Pi(\mathcal{L})^* = \mathcal{L}^* \cap (\mathcal{L}')^\perp$, as claimed.

It remains to show that $\Pi(\mathcal{L})$ is a lattice with rank $n - \ell$. We will first show that $\Pi(\mathcal{L})^*$ has these properties! Indeed, $\Pi(\mathcal{L})^* = \mathcal{L}^* \cap (\mathcal{L}')^\perp$ is a lattice, since it is a subgroup of a lattice. And $\mathcal{L}^* \cap (\mathcal{L}')^\perp$ has rank at most $n - \ell$, because it is contained in a $(n - \ell)$-dimensional subspace; namely, the intersection of the n-dimensional subspace $\mathrm{span}(\mathcal{L}^*)$ with the complement of its ℓ-dimensional subset subspace $\mathrm{span}(\mathcal{L}')$. To see that it has rank at least $n - \ell$, notice that for every linearly independent set of lattice vectors $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathcal{L}$ there exists a dual vector that has non-zero inner product with $\mathbf{y}_1$ and inner product zero with $\mathbf{y}_2, \ldots, \mathbf{y}_n$. Thus, fixing any basis for $(\mathcal{L}')$ and extending it to a set of n linearly independent vectors in $\mathcal{L}$, we can find $n - \ell$ dual vectors in $\mathcal{L}^* \cap (\mathcal{L}')^\perp$ that are all linearly independent.

It is clear that $\mathrm{span}(\Pi(\mathcal{L}))$ has dimension exactly $n - \ell$. It follows that $\mathrm{span}(\Pi(\mathcal{L}^*)) = \mathrm{span}(\Pi(\mathcal{L}))$. It is clear from this combined with the definition of dual set that $\Pi(\mathcal{L}) \subseteq (\Pi(\mathcal{L})^*)^*$.

But $(\Pi(\mathcal{L})^*)^*$ is a lattice, because $\Pi(\mathcal{L})^*$ is. Thus $\Pi(\mathcal{L})$ is a subgroup of a lattice, and therefore is also a lattice. □

Lemma 3.2. *Suppose that $\mathcal{L}'$ of rank ℓ is a primitive sublattice of $\mathcal{L}$ of rank n. Then, for any basis $(\mathbf{b}_1, \ldots, \mathbf{b}_\ell)$ of $\mathcal{L}'$, there exists $\mathbf{b}_{\ell+1}, \ldots, \mathbf{b}_n$ such that $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is a basis of $\mathcal{L}$.*

Proof. By Lemma 3.1, $\Pi_{(\mathcal{L}')^\perp}(\mathcal{L})$ is a lattice with rank $n-\ell$. Therefore, it has a basis $(\mathbf{b}'_1, \ldots, \mathbf{b}'_{n-\ell})$. For $i \in [n-\ell]$, let $\mathbf{b}_{\ell+i} \in \mathcal{L}$ be such that $\Pi_{(\mathcal{L}')^\perp}(\mathbf{b}_{\ell+i}) = \mathbf{b}'_i$. Suppose for contradiction that $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is not a basis for $\mathcal{L}$, that is, that there is some vector $\mathbf{y} \in \mathcal{L}$ such that $\mathbf{y} = a_1\mathbf{b}_1 + \cdots + a_n\mathbf{b}_n$ and not all a_i are integers. Since $\Pi_{(\mathcal{L}')^\perp}(\mathbf{y}) = a_{\ell+1}\mathbf{b}'_1 + \cdots + a_n\mathbf{b}'_{n-\ell} \in \Pi_{(\mathcal{L}')^\perp}(\mathcal{L})$, it follows that $a_{\ell+1}, \ldots, a_n$ are all integers. Thus, one of $a_1, \ldots, a_\ell$ is not an integer. But then $\mathbf{v} := a_1\mathbf{b}_1 + \cdots + a_\ell\mathbf{b}_\ell$ is a point in $\mathcal{L} \cap \operatorname{span}(\mathcal{L}')$ which is not an integer linear combination of $\mathbf{b}_1, \ldots, \mathbf{b}_\ell$, contradicting that $\mathcal{L}'$ is a primitive sublattice. □

Lemma 3.3. *Suppose that $\mathcal{L}'$ of rank ℓ is a primitive sublattice of $\mathcal{L}$ of rank n. Then $\mathcal{L}'' := \mathcal{L}^* \cap (\mathcal{L}')^\perp$ is a rank $(n-\ell)$ primitive sublattice of $\mathcal{L}^*$ satisfying $\det(\mathcal{L}'') = \det(\mathcal{L}^*) \cdot \det(\mathcal{L}')$.*

Proof. By Lemma 3.1, $\mathcal{L}'' = \Pi_{(\mathcal{L}')^\perp}(\mathcal{L})^*$, where $\Pi_{(\mathcal{L}')^\perp}(\mathcal{L})$ is a lattice of rank $n-\ell$. Hence $\mathcal{L}''$ is also a lattice of rank $n-\ell$. And it is a primitive sublattice of $\mathcal{L}^*$ by definition, since it is the intersection of $\mathcal{L}^*$ with a subspace. It remains to verify the determinant identity. Let $\mathbf{b}_1, \ldots, \mathbf{b}_n$ be a basis for $\mathcal{L}$ such that $\mathbf{b}_1, \ldots, \mathbf{b}_\ell$ is a basis for $\mathcal{L}'$, as guaranteed by Lemma 3.2. Then $\det(\mathcal{L}) = \det((\mathbf{b}_1, \ldots, \mathbf{b}_\ell)) \cdot \det((\Pi_{(\mathcal{L}')^\perp}(\mathbf{b}_{\ell+1}), \ldots, \Pi_{(\mathcal{L}')^\perp}(\mathbf{b}_n)))$. The latter projected basis is evidently a basis for $\Pi_{(\mathcal{L}')^\perp}(\mathcal{L})$, so this equation can be written as $\det(\mathcal{L}) = \det(\mathcal{L}') \cdot \det(\Pi_{(\mathcal{L}')^\perp}(\mathcal{L}))$. Substituting in $\det(\mathcal{L}) = 1/\det(\mathcal{L}^*)$ and $\det(\mathcal{L}''^*) = 1/\det(\mathcal{L}'')$, we get $1/\det(\mathcal{L}^*) = \det(\mathcal{L}')/\det(\mathcal{L}'')$, which upon rearranging yields the claimed identity. □

We remark that a basis for $\mathcal{L}''$ can be efficiently computed given bases for $\mathcal{L}$ and $\mathcal{L}'$.

Given a rank-ℓ sublattice $\mathcal{L}'$ of a lattice ℓ of rank n, define $\gamma(\mathcal{L}, \mathcal{L}') = \det(\mathcal{L}')/\det(\mathcal{L})^{\ell/n}$.

Lemma 3.4. *For all $n \geq \ell \geq 1$, the duality map $(\mathcal{L}, \mathcal{L}') \mapsto (\mathcal{L}^*, \mathcal{L}^* \cap (\mathcal{L}')^\perp)$ is an involution on the set of (lattice, primitive-sublattice) pairs, that preserves γ.*

Proof. Fix a lattice $\mathcal{L}$, which we may again assume is full-rank without loss of generality. Let $\mathcal{L}'' := \mathcal{L}^* \cap (\mathcal{L}')^\perp$. Lemma 3.3 shows that $\mathcal{L}''$ is primitive. It also implies that the approximation factor is preserved:

$$\begin{aligned}\gamma = \gamma(\mathcal{L}'', \mathcal{L}^*) &:= \det(\mathcal{L}'')/\det(\mathcal{L}^*)^{\frac{n-\ell}{n}} \\ &= \det(\mathcal{L}^*) \cdot \det(\mathcal{L}')/\det(\mathcal{L}^*)^{\frac{n-\ell}{n}} \\ &= \det(\mathcal{L})^{\frac{n-\ell}{n}} \cdot \det(\mathcal{L}')/\det(\mathcal{L}) \\ &= \det(\mathcal{L}')/\det(\mathcal{L})^{\frac{\ell}{n}} \\ &= \gamma(\mathcal{L}', \mathcal{L}).\end{aligned}$$

It remains to show that duality is an involution, that is, $\mathcal{L}' = \mathcal{L} \cap (\mathcal{L}^* \cap (\mathcal{L}')^\perp)^\perp$. But since $\mathcal{L}^* \cap (\mathcal{L}')^\perp$ is rank $n-\ell$, $\operatorname{span}(\mathcal{L}^* \cap (\mathcal{L}')^\perp) = \operatorname{span}((\mathcal{L}')^\perp)$. Thus $(\mathcal{L}^* \cap (\mathcal{L}')^\perp)^\perp = \operatorname{span}(\mathcal{L}')$, so the desired claim is simply $\mathcal{L}' = \mathcal{L} \cap \operatorname{span}(\mathcal{L}')$, which holds by the definition of primitive sublattice. □

3.2 The Densest Sublattice Problem (DSP)

Definition 3.5. *For $n \geq \ell \geq 1$ and $\gamma \geq 1$, the (n, ℓ, γ)-approximate densest sublattice problem ((n, ℓ, γ)-DSP) is defined as follows. The input is (a basis for) a full-rank lattice $\mathcal{L} \subset \mathbb{R}^n$. The output is (a basis for) a sublattice $\mathcal{L}' \subseteq \mathcal{L}$ of rank ℓ satisfying $\gamma(\mathcal{L}', \mathcal{L}) \leq \gamma$.*

We refer to $\gamma(\mathcal{L}', \mathcal{L})$ as the *approximation factor* achieved by $\mathcal{L}'$. It is convenient to write (n, ℓ, γ)-DSP$(\mathcal{L})$ for the set of solutions of (n, ℓ, γ)-DSP on input $\mathcal{L}$. For our results, we need two important properties of (n, ℓ, γ)-DSP. First, from Lemma 3.4, it is immediate that (n, ℓ, γ)-DSP enjoys the following self-duality property.

Corollary 3.6. *Let $\mathcal{L}'$ be a primitive sublattice of $\mathcal{L}$. Then $\mathcal{L}' \in (n, \ell, \gamma)$-DSP$(\mathcal{L})$ if and only if $\mathcal{L}^* \cap \mathcal{L}'^{\perp} \in (n, n - \ell, \gamma)$-DSP$(\mathcal{L}^*)$.*

Second, (n, ℓ, γ)-DSP behaves nicely under composition, in the following sense.

Lemma 3.7. *Suppose that $\mathcal{L}' \in (n, m, \gamma_1)$-DSP$(\mathcal{L})$, and $\mathcal{L}'' \in (m, \ell, \gamma_2)$-DSP$(\mathcal{L}')$. Then*

$$\mathcal{L}'' \in \left(n, \ell, \gamma_2 \cdot \gamma_1^{\frac{\ell}{m}}\right)\text{-DSP}(\mathcal{L}) .$$

Proof. We may directly calculate

$$\begin{aligned}\gamma(\mathcal{L}', \mathcal{L}'') \cdot \gamma(\mathcal{L}, \mathcal{L}')^{\frac{\ell}{m}} &= (\det(\mathcal{L}'')/\det(\mathcal{L}')^{\frac{\ell}{m}}) \cdot (\det(\mathcal{L}')/\det(\mathcal{L})^{m/n})^{\frac{\ell}{m}} \\ &= \det(\mathcal{L}'')/\det(\mathcal{L})^{\frac{\ell}{n}} =: \gamma(\mathcal{L}, \mathcal{L}'') .\end{aligned}$$

□

4 Reducing approximate DSP to DSP in lower dimensions

We show a reduction from approximate DSP in dimension n to DSP in dimension $k < n$. This reduction is interesting in its own right, but we also note that the reduction and analysis serve as a useful warmup for Section 5. On input a (basis for a) lattice $\mathcal{L}$ with rank n, an integer ℓ with $n > \ell \geq 1$, and an integer $\tau \geq 0$, the reduction $\mathcal{A}(\mathcal{L}, \ell, \tau)$ behaves as follows.

1. **Duality step:** If $\ell > n/2$, output $\mathcal{L} \cap \mathcal{A}(\mathcal{L}^*, n - \ell, \tau)^{\perp}$.
2. **Base cases:**
   (a) **DSP:** If $n = k$, use an oracle for $\gamma = \gamma(k, \ell)$-DSP to output a dense sublattice of $\mathcal{L}$ with rank ℓ.
   (b) **LLL:** If $\tau = 0$, run LLL on $\mathcal{L}$ and output the lattice generated by the first ℓ vectors of the resulting basis.
3. **Recursive step:** Otherwise, output $\mathcal{A}(\mathcal{L} \cap \mathcal{A}(\mathcal{L}, \ell^*, \tau - 1)^{\perp}, \ell, \tau)$, where $\ell^* = \lceil (n - k)/20 \rceil$.

Theorem 4.1. *Let $\mathcal{L}$ be a lattice with rank $n \geq k \geq 10$, $\tau \geq 0$, and $1 \leq \ell < n$. Then on input $(\mathcal{L}, \ell, \tau)$, the above reduction $\mathcal{A}$ runs in time $\mathrm{poly}(n) \cdot 2^{\tau}$, and returns $\mathcal{L}' \subset \mathcal{L}$ with rank ℓ with*

$$\gamma' := \det \mathcal{L}' / (\det \mathcal{L})^{\ell/n} \leq \alpha_k^{\ell(n-\ell)} \cdot 2^{\frac{\ell(n-\ell)n^2}{2^{\tau/2}}} ,$$

where

$$\alpha_k := \max_{1 \le \ell' < k} \gamma(k,\ell)^{\frac{1}{\ell'(k-\ell')}} \; .$$

We remark that if the DSP oracle is exact, that is, $\gamma(k,\ell) = \sqrt{\delta_{k,\ell}}$ for all $1 \le \ell < k$, we have $\alpha_k = k^{1/k}$ and therefore $\gamma' = k^{\Theta(\ell(n-\ell)/k)}$ (where the hidden constant in the exponent is unknown but likely about 1/2).

4.1 Proof of Theorem 4.1

In the recursive step, we call the inner recursive call a "right child" and the outer recursive call a "left child," and we write the parameters associated with the right child as (n_R, ℓ_R, τ_R) and similarly for the left child (n_L, ℓ_L, τ_L).

We first observe that the reduction maintains the invariants that $n \ge k$ and and that $1 \le \ell < n$. It is trivial to see that the reduction maintains the invariant that $1 \le \ell < n$ by simply noting that in the recursive step the right child has $\ell_R := \ell^* = \lceil (n-k)/20 \rceil$ and $n_R = n \ge k$, which certainly satisfies this. The left child has $\ell_L = \ell \le n/2$ (since if $\ell > n/2$, we would have applied a duality step and not a recursive step) and $n_L = n - \ell^* = n - \lceil (n-k)/20 \rceil > n/2 \ge \ell_L$ and also $n_L = n - \lceil (n-k)/20 \rceil \ge k$. Duality does not affect these invariants, so all steps maintain the invariants.

We next prove that the reduction runs in the claimed time. Notice that the right child has $\tau_R = \tau - 1$, and the left child has $n_L - k \le 19(n-k)/20$. The duality step does not affect $n-k$ or τ. It immediately follows that the tree of recursive steps has height $O(\log n) + \tau$. The total number of nodes in this tree is therefore bounded by $\mathrm{poly}(n) \cdot 2^\tau$. Furthermore, the number of duality nodes is at most the number of recursive steps, so the total number of calls to $\mathcal{A}$ is bounded by $\mathrm{poly}(n) \cdot 2^\tau$ and the running time of the reduction is similarly bounded by $\mathrm{poly}(n) \cdot 2^\tau$.

Finally, we bound the approximation factor. To that end, let $\gamma'(n,\ell,\tau)$ be the approximation factor achieved by the above reduction when called on a lattice with rank n with parameters (ℓ,τ). We prove by induction on n and τ that

$$\gamma'(n,\ell,\tau) \le f(n,\ell,\tau) := \alpha_k^{\ell(n-\ell)} \cdot 2^{\frac{\ell(n-\ell)n^2}{2^{\tau/2}}} \; ,$$

as needed. Indeed, it is trivial to check that the base cases satisfy this, since the LLL algorithm satisfies, e.g., $\gamma'(n,\ell,0) \le 2^{\ell(n-\ell)} \le f(n,\ell,0)$ [PT09], and by definition the output of the DSP oracle satisfies $\gamma'(k,\ell,\tau) \le \gamma(k,\ell) \le \alpha_k^{\ell(k-\ell)} \le f(k,\ell,\tau)$. Furthermore, the duality step does not change either γ' or f, so we may ignore duality steps.

It remains to handle recursive steps. By Lemma 3.7, in a recursive step, γ satisfies the recurrence

$$\gamma'(n,\ell,\tau) \le \gamma'(n-\ell^*,\ell,\tau) \cdot \gamma'(n,\ell^*,\tau-1)^{\frac{\ell}{n-\ell^*}} \; .$$

By induction, we have

$$\begin{aligned} \gamma'(n,\ell,\tau) &\le f(n-\ell^*,\ell,\tau) f(n,\ell^*,\tau-1)^{\frac{\ell}{n-\ell^*}} \\ &= \alpha_k^{\ell(n-\ell-\ell^*)+\ell^*\ell} \cdot 2^{\ell(n-\ell^*-\ell)(n-\ell^*)^2/2^{\tau/2} + \ell^*\ell n^2/2^{\tau/2-1/2}} \\ &= \alpha_k^{\ell(n-\ell)} \cdot 2^{\frac{\ell}{2^{\tau/2}}(\sqrt{2}\ell^* n^2 + (n-\ell^*-\ell)(n-\ell^*)^2)} \; . \end{aligned}$$

Therefore,

$$\begin{aligned}\beta &:= 2^{\tau/2} \cdot \log(\gamma'(n,\ell,\tau)/f(n,\ell,\tau)) \\ &\le \sqrt{2}\ell\ell^* n^2 + \ell(n-\ell^*-\ell)(n-\ell^*)^2 - \ell(n-\ell)n^2 \\ &= \ell\ell^*(\ell^*(3n-\ell^*) + \ell(2n-\ell^*) - (3-\sqrt{2})n^2)\end{aligned}$$

Finally, recalling that $\ell \le n/2$ and $\ell^* = \lceil (n-k)/20 \rceil \le n/10$, we see that

$$\beta \le \ell\ell^* \cdot (3n^2/10 + n^2 - (3-\sqrt{2})n^2) \le 0 \; ,$$

i.e., $\gamma'(n,\ell,\tau) \le f(n,\ell,\tau)$, as needed. □

5 Reducing HSVP (and even DSP!) to HSVP in lower dimension

We now show how to modify the reduction from the previous section into a reduction from approximate DSP in dimension n to HSVP in dimension $k < n$. (In other words, we show how to replace the DSP oracle with an HSVP oracle.) The high-level idea is simply to modify the above reduction in such a way that, whenever we make a recursive call with $n = k$, we will always have $\ell = 1$ or $\ell = n-1$. (Notice that $\ell = n-1$ is just as good, since after applying duality, we have $\ell' = n - \ell = 1$. Of course, we must do this while maintaining the invariant that ℓ^* is typically a constant fraction of $n-k$, so that the depth of the tree will still be logarithmic in $n-k$.) To do so, it suffices to design our recursive calls to maintain the invariant that

$$\Delta := n - k + 1 - \min\{\ell, n-\ell\} \ge 0 \; . \tag{3}$$

Notice that this is quite a natural condition to impose, given that we wish to have $\min\{\ell, n-\ell\} = 1$ when $n = k$.

Let's refer to the inner call made by a recursive call as its *left child*, with parameters n_L, ℓ_L, τ_L, and $\Delta_L := n_L - k - \min\{\ell_L, n_L - \ell_L\} + 1$; similarly, we'll refer to the outer call as the *right child* and denote its parameters by n_R, ℓ_R, and so on. (This terminology of "right and left children" of course comes from thinking of the recursion tree that our reduction follows.) Since right children have rank $n_R = n$ and $\ell_R = \ell^*$, maintaining Equation (3) for right children simply amounts to choosing $\ell^* \le (n-k)+1$. We actually take $\ell^* \approx (n-k)/20$ to be much smaller than this, so this is fine.

On the other hand, left children have $n_L := n - \ell^*$ and $\ell_L = \ell$. In particular, notice that if $\ell \ge n/2$, then $\Delta_L := n_L - k + 1 - (n_L - \ell) = \Delta$, so that $\Delta_L \ge \Delta$, and we clearly maintain (3) for left children. On the other hand, if $\ell < n/2$, then (ignoring the corner case when $\ell < n/2$ but $\ell_L \ge n_L/2$, which never actually occurs in our reduction), we have

$$\Delta_L = n_L - k + 1 - \ell = \Delta - \ell^* \; ,$$

so that Δ_L is decreasing in this case, which could be bad.

To deal with this issue, we simply "make sure that whenever we make a recursive call with $\ell \le n/2$, Δ must be larger than ℓ^*." To do that, we simply apply duality whenever $\ell \le n/2$ and Δ is small. This amounts to applying duality whenever ℓ is nearly as large as $n-k$. (Notice that this is actually quite a natural way to keep $\min\{\ell, n-\ell\}$ small.)

In this way, we are able to guarantee that whenever we have $n = k$, we must have either $\ell = 1$ or $n - \ell = 1$. (The above description ignores the fact that we must of course maintain the invariant that $\ell < n$. In order to account for that, we also apply duality if, say, $\ell \gtrsim n - (n-k)/10$.)

But, we must still of course worry about the approximation factor achieved by this approach. In particular, recall that the recurrence relation for our approximation factor in the previous section was

$$\gamma(n, \ell, \tau) \le \gamma(n - \ell^*, \ell, \tau)\gamma(n, \ell^*, \tau - 1)^{\ell/(n-\ell^*)} .$$

Unsurprisingly, we obtain essentially the same recurrence here. But, in the previous section, we kept the exponent $\ell/(n - \ell^*) < C$ for some constant $C < 1$, and intuitively, this meant that the contribution of the right child to the approximation factor was somehow significantly smaller than the contribution of the left child. (And, of course, this is made formal in our analysis of the recurrence.) This is what justified taking depth parameter $\tau_R = \tau - 1$ for the right child.

However, with our new approach, we can no longer guarantee that this exponent $\ell/(n - \ell^*)$ is always small. In particular, though we would like to apply duality whenever, say, $\ell > n/2$ in order to keep ℓ small and therefore keep this exponent bounded, we cannot apply duality when $\ell \lesssim k$, since this could bring us back to the case $\min\{\ell, n - \ell\} \gtrsim n - k$ that we were trying to avoid above. When $n < 2k$, we could have both $\ell > n/2$ and $\ell \lesssim k$, in which case it is not clear what to do.

In this rather frustrating corner case, we simply do not lower τ for the right child (or the left child, for that matter), i.e., we take $\tau_R = \tau$ instead of $\tau_R = \tau - 1$. This might seem like it would substantially increase the running time of our algorithm, or even lead to an infinite loop! Intuitively, this is not a problem because the case when $\ell > n/2$ but $\ell \lesssim k$ is relatively rare, and indeed, we formally show that our running time is essentially unaffected by this (by showing that the grandchildren of any such node must either have lower τ or substantially smaller $n - k$).

5.1 The algorithm

On input a (basis for a) lattice $\mathcal{L}$ of rank n, an integer ℓ with $n > \ell \ge 1$, and an integer $\tau \ge 0$, the reduction $\mathcal{A}(\mathcal{L}, \ell, \tau)$ behaves as follows.

1. **Duality step:** If $\max\{1, (n-k)/5\} < \ell < n/2$ or $\ell \ge n - \max\{1, (n-k)/10\}$, output $\mathcal{L} \cap \mathcal{A}(\mathcal{L}^*, n - \ell, \tau)^\perp$.
2. **Base cases:**
   (a) **SVP:** If $n = k$ and $\ell = 1$, use an oracle for $\gamma = \gamma(k)$-HSVP to output (the lattice generated by) a short vector in $\mathcal{L}$.
   (b) **LLL:** If $\tau = 0$, run LLL on $\mathcal{L}$ and output the lattice generated by the first ℓ vectors of the resulting basis.
3. **Recursive step:** Otherwise, set $\ell^* = \lceil (n-k)/20 \rceil$ and return $\mathcal{A}(\mathcal{L} \cap \mathcal{A}(\mathcal{L}^*, \ell^*, \tau - b)^\perp, \ell, \tau)$, where $b = 1$ if $\ell < n/2$ and $b = 0$ otherwise.

Theorem 5.1. *Let $\mathcal{L}$ be a lattice with rank $n \ge k \ge 10$, and $\tau \ge 0$, and suppose that $1 \le \ell \le n - k + 1$. Then on input $(\mathcal{L}, \ell, \tau)$, $\mathcal{A}$ runs in time* $\mathrm{poly}(n) \cdot 4^\tau$*, and returns $\mathcal{L}' \subset \mathcal{L}$ with rank ℓ satisfying*

$$\gamma' := \det \mathcal{L}' / (\det \mathcal{L})^{\frac{\ell}{n}} \le \gamma(k)^{\frac{\ell(n-\ell)}{k-1}} \cdot 2^{\frac{\ell(n-\ell)n^2}{2^\tau}} .$$

5.2 Proof of Theorem 5.1

First, we must check that all calls to $\mathcal{A}$ have $n \geq k$ and $n > \ell \geq 1$. The initial call satisfies these conditions by assumption, so we need only check the children of a call that does satisfy these conditions. We can ignore the duality step, since interchanging ℓ with $n - \ell$ leaves the two conditions unchanged. The right child of a recursive case clearly satisfies the conditions, since $1 \leq \lceil (n-k)/20 \rceil < n$ (recalling that $\ell_R = \lceil (n-k)/20 \rceil$ and $n_R = n$). So, it only remains to consider the left child of a recursive case, which has $n_L = n - \lceil (n-k)/20 \rceil$. Clearly $n_L \geq k$. Moreover, the duality step ensures that $\ell_L = \ell < n - \max\{1, (n-k)/10\} \leq n_L$, as needed. (I.e., if ℓ were larger than this, we would have applied duality. So, we never reach a recursive call with $\ell \geq n - \max\{1, (n-k)/10\}$.)

Next, we must verify that any recursive call with $n = k$ must have $\ell = 1$ or $\ell = n - 1$. This follows as an immediate corollary of the following lemma, which formalizes the discussion about Δ above.

Lemma 5.2. *All recursive calls satisfy the* invariant $\min\{\ell, n-\ell\} \leq n - k + 1$.

Proof. As in Equation (3) above, we define

$$\Delta(n, \ell) := n - k - \min\{\ell, n - \ell\} + 1 \; ,$$

and we note that it suffices to prove that Δ is non-negative for all recursive calls. The initial call to $\mathcal{A}$ satisfies the invariant by assumption, so we need only check that the children of a call with non-negative $\Delta(n, \ell)$ will have $\Delta(n_L, \ell_L) \geq 0$ and $\Delta(n_R, \ell_R) \geq 0$. We can again ignore the duality step, because interchanging ℓ and $n - \ell$ clearly does not affect Δ. The right children of recursion steps trivially satisfy $\Delta(n_R, \ell_R) \geq 0$, by our choice of $\ell_R = \ell^* = \lceil (n-k)/20 \rceil \leq n - k + 1 = n_R - k + 1$. Furthermore, if $\ell \geq n/2$, then the left child of a recursion step has $n_L = n - \ell^*$ and $\ell_L = \ell$ and therefore must satisfy

$$\Delta(n_L, \ell_L) = n_L - k - (n_L - \ell) + 1 = n - \ell^* - k - (n - \ell^* - \ell) + 1 = \Delta + \ell^*$$

So, if $\ell \geq n/2$ for a recursion step and the recursion step has non-negative Δ, then so will its children.

It remains to show that if a recursion step has $\ell < n/2$, then its left child still satisfies the invariant. To see this, notice that if $\ell \leq n/2$, Δ is actually quite large. Indeed, because of the duality step, we only reach a recursion step when with $\ell \leq n/2$ if we in fact have the stronger condition that $\ell \leq \max\{(n-k)/5, 1\}$. This is enough to infer that the left child has non-negative Δ, since $n_L := n - \lceil (n-k)/20 \rceil \leq n - \lceil (n_L - k)/20 \rceil$ and

$$\ell_L := \ell \leq \max\{(n-k)/5, 1\} \leq (n_L + (n_L - k)/20 - k)/5 + 1 \leq n_L - k + 1 \; ,$$

as needed. □

Next we would like to argue that the total number of recursive calls is appropriately bounded.

Lemma 5.3. *The total number of recursive calls made by the algorithm is bounded by* $4^\tau \cdot \mathrm{poly}(n)$.

Proof. Notice that it suffices to argue that the total number of recursion steps in any path in the tree is bounded by $2\tau + O(\log n)$. (Indeed, since every duality step is followed by a recursion step, it suffices to prove that the number of recursion steps and base cases is bounded by $4^\tau \cdot \mathrm{poly}(n)$. Of course, the size of a binary tree is at most $2 \cdot 2^h$, where h is the height of the tree, so the total number of base cases and recursion steps is bounded by $2 \cdot 2^h$, where h is the length of the longest path of recursion steps.)

Define the *potential* Φ for a recursive call with parameter n, ℓ, and τ as

$$\Phi = \tau + 20\log(n-k+1) \ .$$

Notice that the potential of a child is always at most the potential of its parent. Thus it suffices to argue that every recursion step's grandchildren have potential $\Phi' \le \Phi - 1$. Indeed, if this holds, then a simple induction argument shows the desired bound on the number of recursion steps in any path.

It is easy to check that the left children of recursion steps have potential $\Phi_L \le \Phi - 1$. Furthermore, when $\ell < n/2$, the right children of recursion steps have $\Phi_R \le \Phi - 1$ (since we lower τ by one). So, when $\ell < n/2$, both children already have smaller potential (and, since the potential never increases, the same is true of grandchildren). However, right children of a recursion step with $\ell \ge n/2$ have $\Phi_R = \Phi$. So, to finish the proof, we must show that all children of a right child of such a recursion step have potential $\Phi' \le \Phi_R - 1 = \Phi - 1$. But, to see this, it suffices to notice that the right child of a recursion step with $\ell \le n/2$ is either a base case (in which case there are simply no grandchildren to worry about) *or* is itself a recursive step with has $\ell_R = \ell^* = \lceil (n-k)/20 \rceil < n/2 = n_R/2$. But, we have already seen that nodes with $\ell < n/2$ have children with strictly smaller potential. The result follows. □

Finally, we bound the approximation factor. Let $\gamma'(n, \ell, \tau)$ be the (worst-case) approximation factor achieved by the reduction $\mathcal{A}(\mathcal{L}, \ell, \tau)$ when its input lattice has rank n. It suffices to show that

$$\gamma'(n, \ell, \tau) \le f(n, \ell, \tau) := \gamma(k)^{\frac{\ell(n-\ell)}{k-1}} \cdot 2^{\frac{\ell(n-\ell)n^2}{2^\tau}} \ .$$

The proof is by induction on n and τ. The base cases are easy to check using the HSVP and LLL guarantees $\gamma'(k, 1, \tau) \le \gamma(k)$, and $\gamma'(n, \ell, 0) \le 2^{\ell(n-\ell)}$. So, we assume that the result holds for all $(n', \tau') < (n, \tau)$ (under the lexicographic order). Notice that we can ignore duality steps, as both γ and f are unchanged if we replace ℓ with $n - \ell$.

By Lemma 3.7, in a recursive step, γ satisfies the recurrence

$$\gamma'(n, \ell, \tau) \le \gamma'(n-\ell^*, \ell, \tau) \cdot \gamma'(n, \ell^*, \tau - b)^{\frac{\ell}{n-\ell^*}} \ .$$

Using the recurrence and the induction hypothesis we get

$$\gamma'(n, \ell, \tau) \le \gamma(k)^{\frac{\ell(n-\ell)}{k-1}} \cdot 2^{\frac{\ell(n-\ell^*-\ell)(n-\ell^*)^2}{2^\tau} + \frac{\ell\ell^* n^2}{2^{\tau-b}}} \ .$$

Therefore, we have

$$\begin{aligned} \alpha &:= 2^\tau \cdot \log(\gamma'(n, \ell, \tau)/f(n, \ell, \tau)) \\ &\le \ell(n-\ell^*-\ell)(n-\ell^*)^2 + 2^b \cdot \ell\ell^* n^2 - \ell(n-\ell)n^2 \\ &= \ell\ell^* \cdot \big(2\ell n + 3\ell^* n - (3-2^b)n^2 - (\ell^*)^2 - \ell\ell^*\big) \ . \end{aligned}$$

When $b = 0$, we therefore have

$$\alpha = \ell\ell^*(2n - \ell^*)(\ell + \ell^* - n) \; ,$$

which is negative since we chose $\ell^* = \lceil (n-k)/20 \rceil$ and by the duality step we have $\ell < n - \max\{1, (n-k)/10\} \le n - \ell^*$. On the other hand, when $b = 1$, we have $\ell \le n/2$ by definition, but notice that by the duality step this actually implies that $\ell \le \max\{1, (n-k)/5\} \le n/5$ and $\ell^* = \lceil (n-k)/20 \rceil \le n/20 + 1$, so that

$$\alpha < \ell\ell^* n(2\ell + 3\ell^* - n) < \ell\ell^* n(2n/5 + 3n/20 + 3 - n) = \ell\ell^* n^2(-9n/20 + 3) \; ,$$

which is clearly negative for $n \ge 10$.

Therefore, $\alpha \le 0$ in both cases, i.e., $\gamma'(n, \ell, \tau) \le f(n, \ell, \tau)$, as needed. This completes the proof of Theorem 5.1.

6 Computer-aided search for reductions, and numerical comparisons

One feature of our DSP to DSP and DSP to HSVP reductions (and reductions in our framework more generally), is that the approximation factors that they achieve satisfies a simple recurrence. This means that concrete provable bounds on the approximation factor achieved by each recursive call can be computed recursively, using exactly the same pattern of recursive calls as the reduction itself. In this section, we illustrate how these concrete bounds can be used to (1) find optimal parameters for the recursion; (2) compare variants of our reductions with each other; and (3) compare our reductions to basis reduction. For simplicity, we will restrict our attention to the case $\ell = 1$, which corresponds to reducing from HSVP.

We stress that our purpose is to illustrate what is possible here, and certainly *not* to provide a definitive analysis on the performance of our reductions (or the performance of basis reduction). In particular, we do not wish to get caught up by thorny issues about (1) the precise value of Hermite's constant in different dimensions; (2) the precise time required to solve $\sqrt{\delta_k}$-HSVP in k dimensions; (3) the precise behavior of basis reduction, including the precise approximation factor achieved by LLL; etc. So, when we encounter such issues, we endeavor to make choices (detailed in Section 6.2) that yield bounds that are reasonable, simple, and *provably correct* (up to lower-order terms in the running time of (H)SVP algorithms, which we do not attempt to model), but certainly not optimal.[10] We leave it to future work to consider such issues more carefully (mindful of the ongoing effort to understand such issues in the context of basis reduction), and here simply show what is possible when these issues are largely ignored.

With that disclaimer out of the way, we note that our framework is quite amenable to such computations. First, not only can we obtain concrete bounds for a particular reduction with particular parameter choices such as the one in Theorem 5.1, one can use dynamic programming to compute the optimal approximation factor achievable with a given number of oracle calls, along

[10]While this thicket of thorny issues is not ideal, we note that the situation is far worse in the context of basis reduction. And, in some sense, these thorns are not the fault of the recursive lattice reduction framework. Indeed, our first thorny issue—the imprecision in Hermite's constant—of course comes with the territory (and is really rather minor). The remaining thorny issues fundamentally boil down to difficulties in nailing down the behavior of algorithms that are outside of our framework, specifically LLL and sieving.

with a description of the reduction achieving it. And, (in stark contrast to, e.g., simulations for basis reduction) the resulting bound on the approximation factor is *proven* correct (assuming that one uses proven bounds on Hermite's constant and proven bounds on the approximation factor achieved by LLL for the base cases—one can also of course plug in conjectured bounds).

Concretely, consider the following variant $\mathcal{A}(\mathcal{L}, \ell, C)$ of our DSP to SVP reduction in Section 5. The reduction is underspecified; namely, the choices of whether to apply duality, and which values ℓ^* and C' to use in the recursive step, are left unspecified.

1. **Duality step:** Choose either to set $\mathcal{L}' := \mathcal{L}$ or set $\mathcal{L}' = \mathcal{L} \cap \mathcal{A}(\mathcal{L}^*, n-\ell, C)^{\perp}$ (where $n := \mathrm{rank}(\mathcal{L})$).

2. **Base cases:**

   (a) **LLL:** If $C = 0$, run LLL on $\mathcal{L}'$ and output the lattice generated by the first ℓ vectors of the resulting basis.

   (b) **SVP:** If $n = k$ and $\ell = 1$, use an oracle for $\gamma = \gamma(k)$-HSVP to output (the lattice generated by) a short vector in $\mathcal{L}'$.

   (c) **Trivial:** If $n = \ell$, output $\mathcal{L}'$.

3. **Recursive step:** Otherwise, choose integers $1 \le \ell^* \le n - \ell$, $0 \le C' \le C$, and output $\mathcal{A}(\mathcal{L}' \cap \mathcal{A}((\mathcal{L}')^*, \ell^*, C')^{\perp}, \ell, C - C')$.

Notice that C bounds the number of HSVP oracle calls the reduction can make, since if $C = 0$ the reduction must run LLL, and in the recursive step, the reduction must divide its budget C between the two recursive calls.

Now we can define a quantity $\gamma'(n, \ell, C)$ that is the best provable bound on the approximation factor the reduction can obtain, over all of its choices. In detail, the base cases are $\gamma'(n, \ell, 0) := (4/3)^{\ell(n-\ell)/4}$, $\gamma'(k, 1, C) := \gamma(k)$, $\gamma'(\ell, \ell, C) := 1$, and for all other values,

$$\gamma'(n, \ell, C) := \min_{\widehat{\ell} \in \{\ell, n-\ell\}} \ \min_{1 \le \ell^* \le n - \widehat{\ell}} \ \min_{0 \le C' \le C} \gamma'(n - \ell^*, \widehat{\ell}, C - C') \cdot \gamma'(n, \ell^*, C')^{\frac{\widehat{\ell}}{n-\ell^*}} \ .$$

The value $\gamma'(n, \ell, C)$, along with a description of the reduction achieving it (that is, the choices of ℓ^* and C' made by the reduction at each step) can then be straightforwardly computed by dynamic programming in space $\widetilde{O}(n^2 C)$ and time $\widetilde{O}(n^3 C^2)$.

We will also want to consider a variant of the above reduction that can make HSVP oracle calls, not just in a single fixed dimension k, but in many different dimensions k. To reflect the fact that oracle calls in larger dimensions are more computationally expensive, instead of simply tracking the total number of oracle calls made by each recursive call, we will track an estimate of the running time. Concretely, suppose that γ-HSVP can be solved in time $T(k)$ in dimension k. We modify the reduction so that (1) the SVP base case no longer requires $n = k$, and (2) the reduction can *choose* whether to apply it or not. We then modify the definition of γ', by letting the base case $\gamma'(n', 1, C) = \gamma(n')$ apply in *arbitrary* dimension n', *provided that* $C \ge T(n')$. With these modifications, C now bounds the running time of the reduction (neglecting the time taken by operations other than HSVP calls).

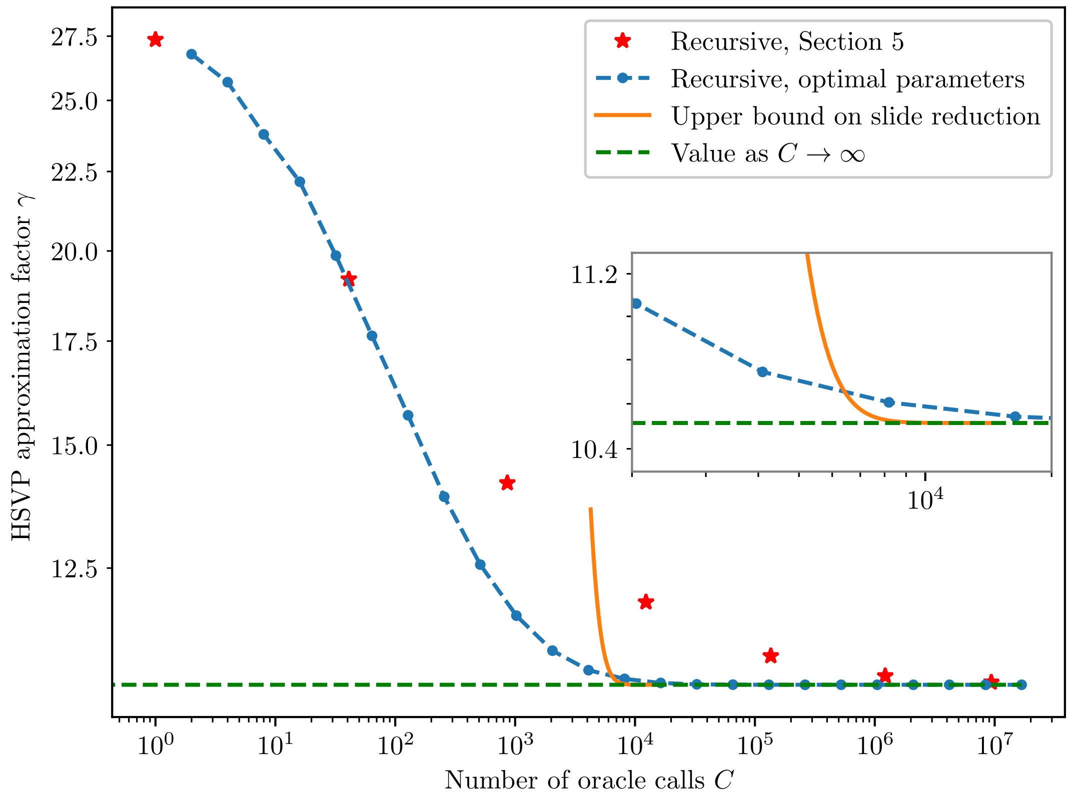

Figure 1: A comparison of tradeoffs between the approximation factor γ and the running time (measured by the number of HSVP oracle calls C) when reducing γ-HSVP in dimension $n = 50$ to HSVP in dimension $k = 10$. The blue dotted curve is our recursive DSP to SVP reduction with nearly optimal parameters chosen by computer search. The red stars show the tradeoff achieved by the recursive reduction of Theorem 5.1. The orange curve is an upper bound on the approximation factor obtained by slide reduction from [Wal21, Corollary 1], which we include to provide (rough) context. The blue, orange, and red curves all converge to the green dotted line as the running time grows large.

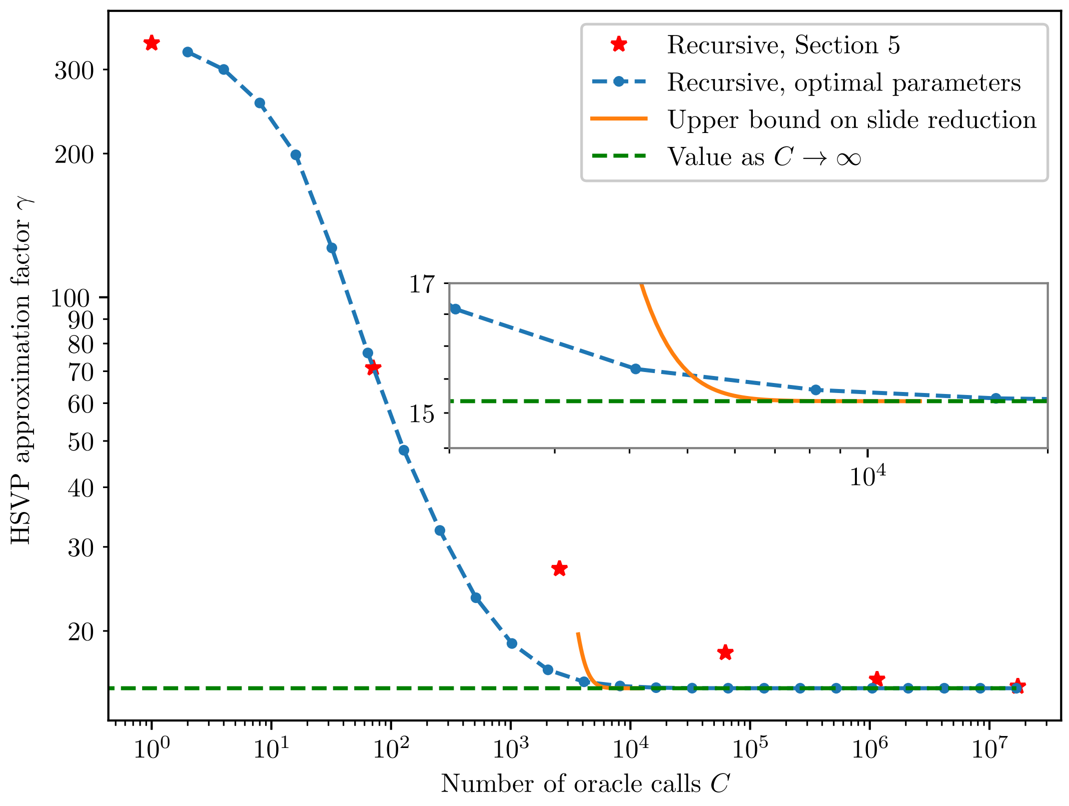

Figure 2: A similar comparison to Figure 1, but with $n = 100$ and $k = 30$.

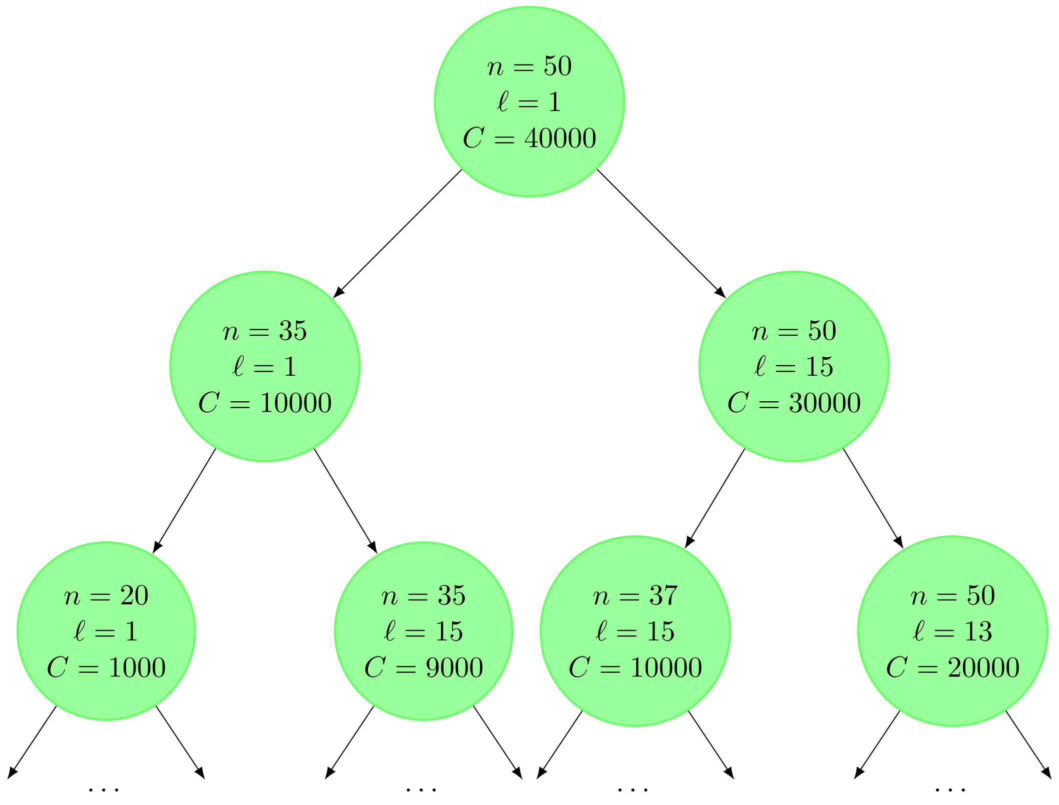
$n = 50$ $\ell = 1$ $C = 40000$
$n = 35$ $\ell = 1$ $C = 10000$
$n = 50$ $\ell = 15$ $C = 30000$
$n = 20$ $\ell = 1$ $C = 1000$
$n = 35$ $\ell = 15$ $C = 9000$
$n = 37$ $\ell = 15$ $C = 10000$
$n = 50$ $\ell = 13$ $C = 20000$
...
...
...
...

Figure 3: The first few recursive calls of our recursive DSP to HSVP reduction, with optimal parameters (up to some rather aggressive rounding) discovered by computer search. Each recursive call is labeled with the input dimension n, output dimension ℓ, and the budget C (of running time measured in HSVP oracle calls) allocated to the call. The initial parameters $n = 50, \ell = 1, C = 40000$ correspond to the regime of Figure 1 where the approximation factor achieved by the recursive reduction (blue curve) has just barely converged.

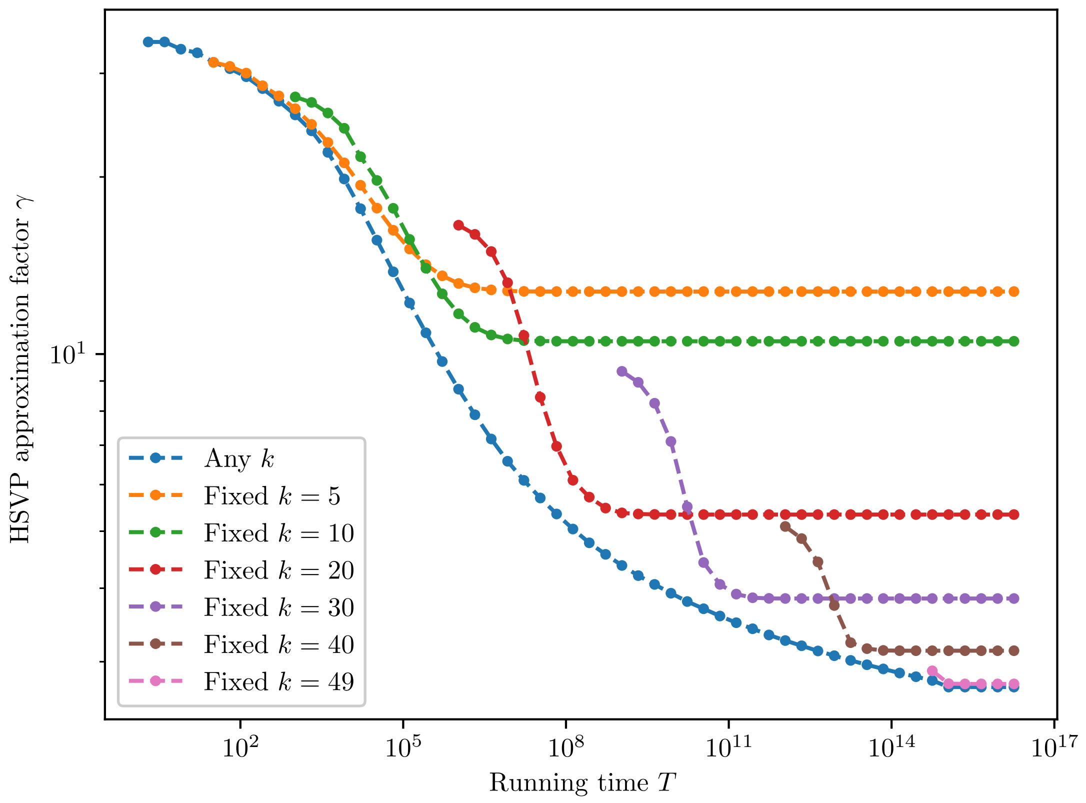

Figure 4: A comparison of tradeoffs between the approximation factor γ and the running time T for solving γ-HSVP in dimension $n = 50$, in the simplified model where we assume that an HSVP oracle call in dimension k takes time 2^k, and we neglect the time taken by all other operations. All curves correspond to our recursive HSVP to HSVP reduction with nearly optimal parameters chosen by computer search. The blue curve is allowed to make HSVP oracle calls in arbitrary dimensions $k < n$, whereas the others are only allowed to make HSVP oracle calls in a certain fixed dimension k.

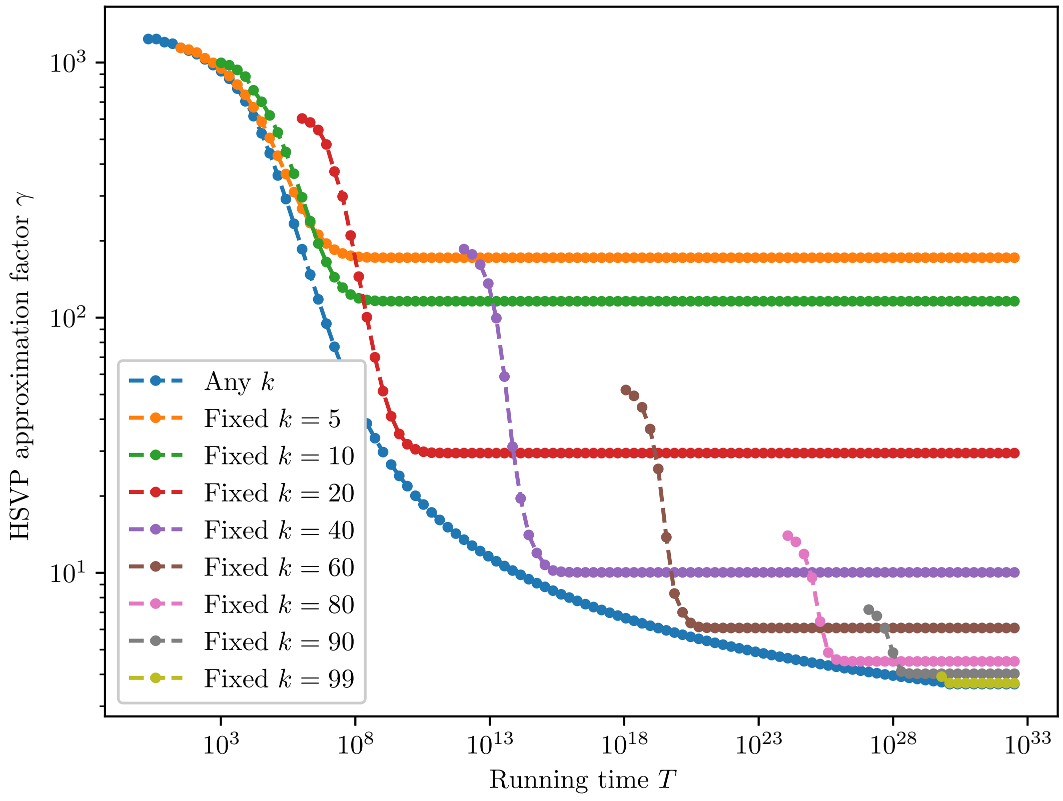

Figure 5: A similar comparison to Figure 4, but with $n = 100$ and $k = 30$.

6.1 Results

In Figures 1 and 2, we show a comparison between the explicit reduction described in Section 5 and the optimized reduction found by the automated procedure described above, with parameters $(n, k) = (50, 10)$ and $(n, k) = (100, 30)$. (The approximation factors achieved and number of oracle calls needed by the explicit reduction from Section 5 were computed by solving the associated recurrences exactly.) These plots show that the computer-generated reduction rather massively improves upon the explicit reduction described in Section 5.

Figure 3 shows some of the actual recursive tree in one of the reductions generated by our automated procedure. Notice that the parameters chosen here are significantly more subtle than the simple ones that we chose in Section 5. In particular, ℓ^* (the value of ℓ taken by the right node) is clearly not just a fixed function of n, and the allocation of oracle calls is similarly subtle–often larger for the right node than the left.

We also include in Figures 1 and 2 Walter's [Wal20, Corollary 1] closed-form (proven) upper bound on the approximation factor achieved by (a variant of) Gama and Nguyen's slide reduction algorithm as a function of the number of oracle calls [GN08a]. We stress that this is only meant for context, to show that our (computer-generated) reduction is roughly comparable to prior work. In particular, while the plots might suggest that our reduction outperforms slide reduction when the number of oracle calls is rather small, we do not claim that this is actually true. (Slide reduction is not well studied in the setting where the number of oracle calls is significantly less than the number needed to converge.)

In Figures 4 and 5, we show comparisons between the performance of different computer-generated reductions in our framework. In particular, we compare the time-approximation-factor tradeoff achieved by our computer-optimized reductions with different fixed oracle sizes k (for $n = 50$ in Figure 4 and $n = 100$ in Figure 5), as well as the tradeoff achieved by allowing the algorithm to adaptively choose the dimension k in which the HSVP oracle is run, as discussed above. As we discuss more in Section 6.2, we used quite a crude cost model of 2^k computational operations for $\sqrt{\delta_k}$-HSVP oracle calls in dimension k (and no cost for all other operations). (We found that changing this cost model did not change the results qualitatively.)

Again, the resulting plots show that substantial improvement can be made by allowing for oracle sizes in different dimensions.

6.2 Technical details

LLL oracle The LLL algorithm, viewed as a γ_{LLL}-DSP oracle, provably achieves approximation factor at most [PT09]

$$\gamma_{\mathsf{LLL}} \le (4/3)^{\frac{\ell(n-\ell)}{4}} \ .$$

We use this bound for the approximation factor achieved by the LLL oracle.[11]

SVP oracle. We assume an exact HSVP oracle, achieving approximation factor $\gamma = \sqrt{\delta_k}$ in dimension k. We use known exact values of δ_k in dimensions $k \le 8$. For $k > 8$, we use the best known asymptotic upper bound on δ_k, due to Blichtfeld [Bli29]:

$$\delta_k \le \frac{2}{\pi}\Gamma(2 + k/2)^{\frac{2}{k}} \ ,$$

[11]In practice, LLL is known to perform *much* better than this [NS06]. So, one might instead plug in heuristic bounds on the approximation factor achieved by LLL.

where $\Gamma(x)$ is the gamma function.[12]

As a rough and simple guess for the running time T required to solve (H)SVP in dimension k, we use $T = 2^k$.[13]

Approximate minimization. Though computer-aided search for optimal recursive reductions in our framework is quite efficient, finding the exact minimum still requires non-trivial amounts of time and memory. The results presented here are therefore only approximate minima (upper bounds on the true minimum), generated by only searching for reductions where the budget C of HSVP oracle calls allocated to a given recursive call cannot be an arbitrary integer, but instead must be chosen from a smaller "coarse" set of values. (Our approach to the running time budget T is similar.) Experimentally, in small dimensions, the upper bounds obtained this way were not much worse than the optima obtained by exact search, so we felt they sufficed for these proof-of-concept illustrations.

In more detail, the coarse sets of values we use are of size $O(\log C)$ (or $O(\log T)$), and they consist of all integers $\le C$ $(\le T)$ that can be written in base b with all non-leading digits 0. (For example, choosing $b = 10$ gives $0, 1, \ldots, 9, 10, 20, \ldots, 90, 100, \ldots$ and so on up to C.) We used $b = 10$ for Figure 3 (in order to display nice round numbers) and $b = 8$ for Figures 1, 2, 4 and 5 (in order to display evenly spaced points on our log plots). The reduction is allowed to split C into C' and C'' with $C' + C'' = C$, which means that when $C = x \cdot b^a$, if the leading digit x is 1 the reduction can choose only $C' = y \cdot b^{a-1}$ for any $y \in \{0, \ldots, b-1\}$, and when $x > 1$, the reduction can choose only $C' = z \cdot b^a$ for some $0 \le z \le x$.

7 Directions for future work

We expect there to be much future work on this new paradigm. Indeed, we largely view this work as introducing new ideas that we hope and expect others will expand upon. We detail some potential future directions below.

New algorithms in this framework. Perhaps the most natural direction for future work is to investigate more (hopefully better) instantiations of the framework that we introduce. For example, while we only considered binary recursion trees, one can consider trees with larger degree. One can also consider algorithms that incorporate projection more directly (in close analogy with basis reduction algorithms). E.g., one could consider algorithms that find a dense sublattice of $\mathcal{L}$ with rank ℓ by first finding a dense sublattice $\mathcal{L}'$ with rank $\ell' < \ell$, finding a dense sublattice $\mathcal{L}'' \subset \Pi_{(\mathcal{L}')^\perp}(\mathcal{L})$ with rank $\ell - \ell'$, and then "lifting" the result to a sublattice $\widehat{\mathcal{L}} \subset \mathcal{L}$ with rank ℓ such that $\mathcal{L}' \subset \widehat{\mathcal{L}}$ and $\Pi_{(\mathcal{L}')}(\widehat{\mathcal{L}}) = \mathcal{L}''$. It would be interesting both to study such approaches theoretically and to incorporate them into automated searches for better-performing reductions in this framework.

[12]There is a lower bound of $\Gamma(k/2+1)^{2/k}/\pi$, so Blichfeldt's bound is tight up to a factor of $2 + o(1)$. One might reasonably guess that the right answer is closer to $\Gamma(k/2+1)^{\frac{2}{k}}/\pi$.

[13]This is approximately the running time of state-of-the-art (provable) exact SVP algorithms [ADRS15], which runs in time $2^{k+o(k)}$. But, we are not being too precise—in particular, dropping the unspecified $2^{o(k)}$ factor. One might instead plug in estimates for, e.g., heuristic sieving algorithms—or, in low dimensions, even for heuristic enumeration algorithms. Here, we only wish to give a qualitative picture of how our algorithm behaves when we allow oracle calls in different dimensions. (This qualitative picture is unchanged when we change how we model the running time of the oracle.)

Recursive lattice reduction and basis reduction. It would also be quite interesting to provide a better understanding of the relationship between our new framework of recursive lattice reduction and the pre-existing framework of basis reduction. As we discussed in Section 2.4, the two frameworks feel quite similar, but we know of no formal relationship between them. In particular, a major open question is whether common basis reduction algorithms can be captured in our framework. In particular, (as we also discuss in Section 2.4), all of our instantiations of recursive lattice reduction require the use of the LLL algorithm as a subprocedure. Ideally, we would like to show that the LLL algorithm itself can be captured in our framework (or a natural generalization of it), to remove this dependency. In particular, we would at least like to match the approximation factor achieved by LLL without using LLL as a subroutine.[14]

Failing that, one could also consider explicitly using other basis reduction algorithms as subprocedures in our framework. E.g., one could imagine including the possibility of running basis reduction algorithms instead of some of the recursive calls—in particular in automated searches for the best reductions, as in Section 6. Perhaps such an algorithm can achieve better performance? Or perhaps one can achieve better performance in basis reduction by using recursive lattice reduction as a subprocedure there? Perhaps there are more general combinations of the two techniques that are interesting as well.

There are also various tricks for speeding up basis reduction algorithms that we currently do not know how to apply in our framework. For example, basis reduction algorithms are often applied *progressively*, in that one reduces the basis with progressively larger block size, obtaining shorter and shorter vectors [AWHT16]. This effectively lowers the number of high-dimensional oracle calls at the expense of using many lower-dimensional oracle calls. While the technique we discuss in Section 6 of using oracle calls with different dimensions is similar to this at a high level, it is not clear that they are truly analogous to progressive basis reduction algorithms. As another example, Walter [Wal21] and Li and Walter [LW23] showed that it can be favorable to use the HKZ-reduced bases that one can easily generate with an SVP oracle to get a coarse solution to DSP. Perhaps similar tricks are possible here?

More generally, modern basis reduction algorithms are quite sophisticated and highly optimized—from high-level optimizations like the progressive reduction described above to optimizations that take into account the fact that the length of an output vector is not fixed but rather is a random variable [YD18] to low-level (but important and very clever!) optimizations related to precision issues [NS09, Ste10, RH23]—and we hope that similarly sophisticated optimizations can and will be applied in our framework. (The fact that many of the techniques for controlling precision issues use recursion [NS16, KEF21, RH23] seems promising—at least superficially.)

And, for practical performance, one should of course introduce heuristics into the study of this framework. We hope that in our setting one can reap the tremendous benefits that heuristics have provided in our understanding of basis reduction without quite needing to completely sacrifice the simple presentation and analysis that our framework provides. In particular, one can hope that it would be sufficient to, e.g., apply heuristics about Hermite's constant, the approximation

[14]We do note that there is a certain weak sense in which our framework captures LLL. If one instantiates any of our algorithms with block size $k = 2$ and replaces calls to LLL with some hypothetical algorithm achieving approximation factor 2^{n^C} for any constant C, then it is not hard to see that one obtains an algorithm more-or-less matching the performance of LLL. I.e., one can use our framework to generically convert any algorithm with approximation factor 2^{n^C} to one with approximation factor $2^{O(n)}$.

factor achieved by LLL, and the other "thorny issues" from Section 6 without, e.g,, making explicit heuristic assumptions about the behavior of reductions in our framework.

Recursive reduction of algebraic lattices. Additionally, one might ask how this new framework applies to algebraically structured lattices, such as embeddings of ideals in number fields or module lattices over the ring of integers. These lattices play a crucial role in lattice-based cryptography, and their reduction remains an active area of research.

It is known that the LLL algorithm [LPSW19] and the slide-reduction algorithm [MS20] can be readily adapted to this context. Furthermore, [KEF20] have explored recursion techniques leveraging the algebraic structure of this class of lattices to enhance the efficiency of the LLL algorithm. Investigating the potential interaction between our recursive framework and the inherent recursive structure of module lattices could yield improvements to such results.

Lattice reduction with few oracle calls. Finally, we note that our framework places new emphasis on the performance of reductions that make very few oracle calls. In particular, because of the recursive nature of our reductions, the final approximation factors that we obtain would be improved quite a bit if we could greatly improve the approximation factor achieved by reductions using, e.g., just one oracle call. More generally, our framework makes very concrete the intuitive notion that an improvement to the fastest running time needed to solve (H)SVP in some dimension n with some approximation factor γ would yield similar improvements for larger n and larger γ, and this motivates further study of lattice reduction in *all* parameter regimes.

References

[ACD$^+$18] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes!, 2018. `https://estimate-all-the-lwe-ntru-schemes.github.io/docs/`. 1

[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. 1, 28

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the Shortest Lattice Vector Problem. In *STOC*, 2001. 1, 2

[ALNS20] Divesh Aggarwal, Jianwei Li, Phong Q. Nguyen, and Noah Stephens-Davidowitz. Slide reduction, revisited—Filling the gaps in SVP approximation. In *CRYPTO*, 2020. 2

[ALS21] Divesh Aggarwal, Zeyong Li, and Noah Stephens-Davidowitz. A $2^{n/2}$-time algorithm for $\sqrt{n}$-SVP and $\sqrt{n}$-Hermite SVP, and an improved time-approximation tradeoff for (H)SVP. In *Eurocrypt*, 2021. 1, 2, 11

[AUV19] Divesh Aggarwal, Bogdan Ursu, and Serge Vaudenay. Faster sieving algorithm for approximate SVP with constant approximation factors. `https://eprint.iacr.org/2019/1028`, 2019. 1, 11

[AWHT16] Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In *Eurocrypt*, 2016. 29

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New Directions in Nearest Neighbor Searching with Applications to Lattice Sieving. In *SODA*, 2016. 1

[Bli29] H. F. Blichfeldt. The minimum value of quadratic forms, and the closest packing of spheres. *Mathematische Annalen*, 101(1):605–608, 1929. 27

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Asiacrypt*, 2011. 3

[Dad19] Daniel Dadush. On approximating the covering radius and finding dense lattice subspaces. In *STOC*, 2019. 6, 10

[DM13] Daniel Dadush and Daniele Micciancio. Algorithms for the Densest Sub-lattice Problem. In *SODA*, 2013. 6, 7, 10

[GN08a] Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *STOC*, 2008. 2, 3, 4, 10, 27

[GN08b] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *Eurocrypt*, 2008. 2, 3

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *CRYPTO*, 2011. 2

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *CRYPTO*, 2007. 7, 9

[KEF20] Paul Kirchner, Thomas Espitau, and Pierre-Alain Fouque. Fast reduction of algebraic lattices over cyclotomic fields. In *CRYPTO*, 2020. 30

[KEF21] Paul Kirchner, Thomas Espitau, and Pierre-Alain Fouque. Towards faster polynomial-time lattice reduction. In *CRYPTO*, 2021. 10, 29

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, 2015. 1

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982. 1

[LN14] Jianwei Li and Phong Q. Nguyen. Approximating the densest sublattice from Rankin's inequality. *LMS J. of Computation and Mathematics*, 17(A):92–111, 2014. 6, 10

[LPSW19] Changmin Lee, Alice Pellet-Mary, Damien Stehlé, and Alexandre Wallet. An LLL algorithm for module lattices. In *ASIACRYPT*, 2019. 30

[LW23] Jianwei Li and Michael Walter. Improving convergence and practicality of slide-type reductions. *Information and Computation*, 291(C), 2023. 6, 9, 10, 29

[LWXZ11] Mingjie Liu, Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. Shortest lattice vectors in the presence of gaps. http://eprint.iacr.org/2011/139, 2011. 1, 11

[MS20] Tamalika Mukherjee and Noah Stephens-Davidowitz. Lattice reduction for modules, or how to reduce modulesvp to modulesvp. In *CRYPTO*, 2020. 30

[MV10a] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC*, 2010. 1

[MV10b] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the Shortest Vector Problem. In *SODA*, 2010. 1

[MW16] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *Eurocrypt*, 2016. 2, 4, 10, 11

[NIS22] NIST. Selected algorithms 2022 - Post-Quantum Cryptography, 2022. https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. 1

[NS06] Phong Q. Nguyen and Damien Stehlé. LLL on the average. In *ANTS*, 2006. 27

[NS09] Phong Q. Nguyen and Damien Stehlé. An LLL Algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009. 29

[NS16] Arnold Neumaier and Damien Stehlé. Faster LLL-type reduction of lattice bases. In *ISSAC*, 2016. 10, 29

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the Shortest Vector Problem are practical. *J. Mathematical Cryptology*, 2(2):181–207, 2008. 1

[PS09] Xavier Pujol and Damien Stehlé. Solving the Shortest Lattice Vector Problem in time $2^{2.465n}$. http://eprint.iacr.org/2009/605, 2009. 1

[PT09] Gabor Pataki and Mustafa Tural. Unifying LLL inequalities. https://gaborpataki.web.unc.edu/wp-content/uploads/sites/14119/2018/07/uniflll.pdf, 2009. 15, 27

[RH23] Keegan Ryan and Nadia Heninger. Fast practical lattice reduction through iterated compression. In *CRYPTO*, 2023. 10, 29

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53(23):201–224, 1987. 2

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathmatical Programming*, 66:181–199, 1994. 2

[Ste10] Damien Stehlé. Floating-point LLL: Theoretical and practical aspects. In *The LLL Algorithm: Survey and Applications*, pages 179–213. Springer, 2010. 29

[SW14] Uri Shapira and Barak Weiss. A volume estimate for the set of stable lattices. *Comptes Rendus Mathématique. Académie des Sciences. Paris*, 352(11):875–879, 2014. 7

[Wal20] Michael Walter. Lattice blog reduction — Part I: BKZ — Calvin Café: The Simons Institute Blog, 2020. https://blog.simons.berkeley.edu/2020/04/lattice-blog-reduction-part-i-bkz/. 2, 27

[Wal21] Michael Walter. The convergence of slide-type reductions. In *PKC*, 2021. 3, 6, 9, 10, 22, 29

[WLW15] Wei Wei, Mingjie Liu, and Xiaoyun Wang. Finding shortest lattice vectors in the presence of gaps. In *CT-RSA*, 2015. 1, 11

[YD18] Yang Yu and Léo Ducas. Second order statistical behavior of LLL and BKZ. In *SAC*, 2018. 29