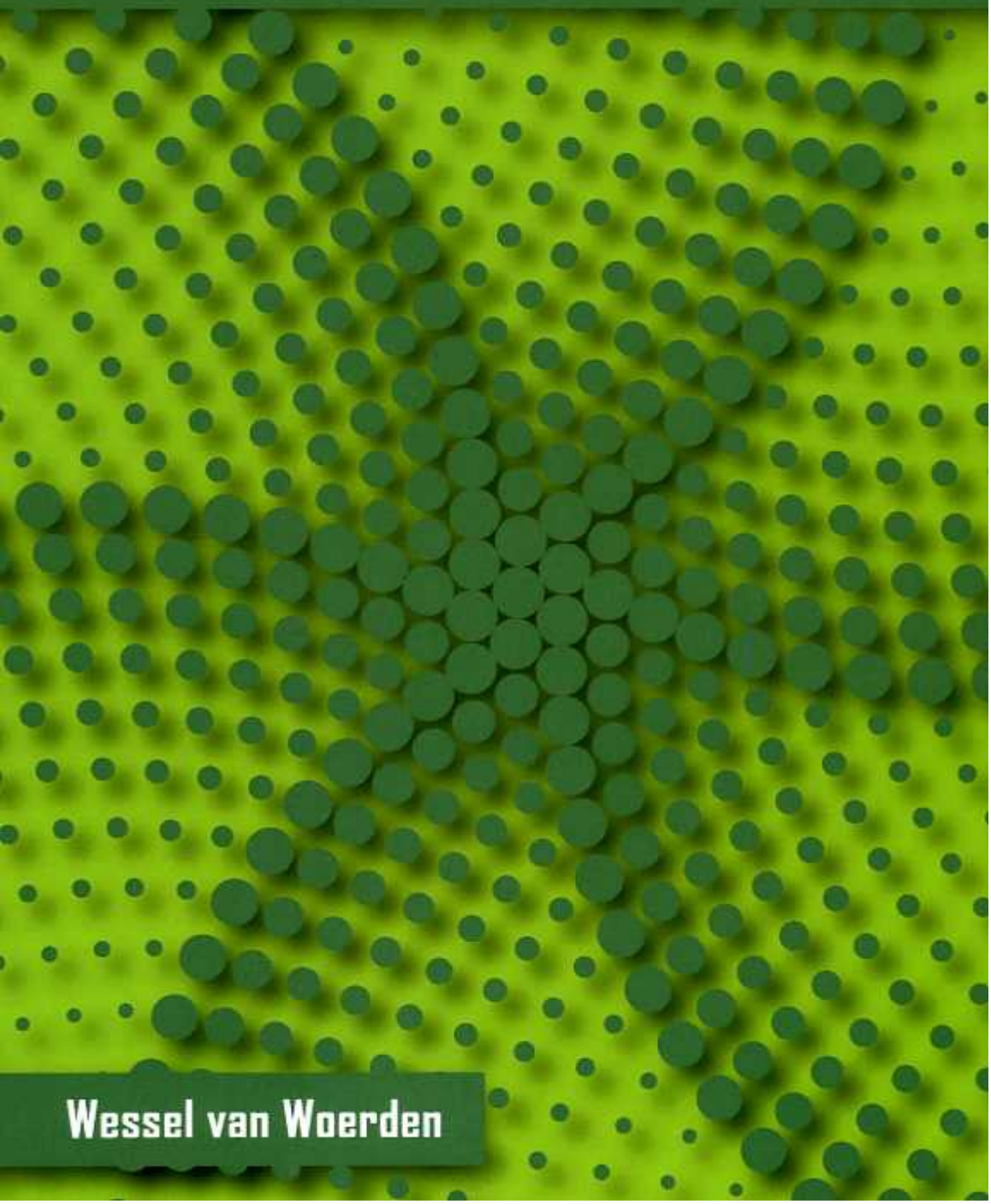
Lattice Cryptography

from Cryptanalysis to New Foundations



Wessel van Woerden

Lattice Cryptography,
from Cryptanalysis to New Foundations

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr.ir. H. Bijl,
volgens besluit van het college voor promoties
te verdedigen op donderdag 23 februari 2022
klokke 13.45 uur

door

Wessel Pieter Jacobus van Woerden
geboren te Rijnwoude, Nederland,
in 1995

Promotores:

Prof.dr. L. Ducas	(CWI Amsterdam & Universiteit Leiden)
Prof.dr. R.J.F. Cramer	(CWI Amsterdam & Universiteit Leiden)

Promotiecommissie:

Prof.dr. H.J. Hupkes	
Dr. T.C. Streng	
Prof.dr. C. Bachoc	(Université de Bordeaux)
Dr. N. Stephens-Davidowitz	(Cornell University)
Prof.dr. P.Q. Nguyen	(Inria Paris & École Normale Supérieure, Paris)

The research was carried out in the Cryptology Group at CWI Amsterdam.
The PhD position was funded by the ERC Advanced Grant 740972
(ALGSTRONGCRYPTO).

CWI
Centrum Wiskunde & Informatica

Universiteit Leiden

Wessel van Woerden

Lattice Cryptography

from Cryptanalysis to New Foundations

To my family and friends.

Contents

Part I

Introduction and Preliminaries

CHAPTER 1

Introduction

The physical world is inherently noisy. An integer vector $\mathbf{x} \in \mathbb{Z}^d$ transmitted over a channel may only be received as some distorted vector $\mathbf{x} + \mathbf{e} \in \mathbb{R}^d$. If the Euclidean norm of the error $\mathbf{e}$ is strictly less than $\frac{1}{2}$, we know that simply rounding the coordinates will recover the original vector. If the error is larger we cannot be sure that the original vector is recovered. In two dimensions one can visually interpret this error correction as the integer *sphere packing* as shown on the left in Figure 1.1. The radius of the spheres corresponds to the largest error from which we can correctly recover the original message. The spheres

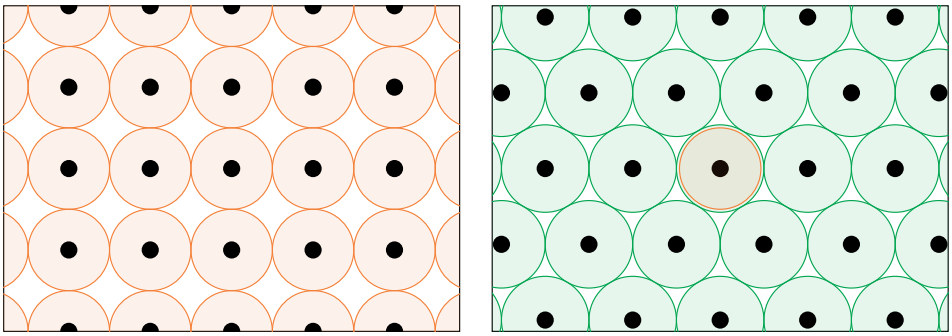


Figure 1.1: The integer and hexagonal sphere packings, with on average one sphere per unit area.

are not allowed to overlap, as that would imply ambiguity on what the original message was.

More generally, any sphere packing gives a way to recover from noise, by encoding each distinct message as the center of one of the spheres. The radius of the spheres then determines the robustness from noise. The number of spheres per fixed volume determines the (relative) number of distinct messages that can be encoded by a single signal, and thereby determines the capacity of the connection. A *denser* packing improves the robustness, while maintaining the same capacity. For example, the densest packing in dimension 2, namely the hexagonal packing in Figure 1.1, allows to recover from errors that are 7% larger than for the integer packing. The gain in dimension 2 may seem small, but in larger dimensions d this radius can increase from $\frac{1}{2}$ to $\Theta(\sqrt{d})$.

The densest sphere packing in dimension 2 follows a repetitive pattern, just as the integer packing. Both are examples of *lattice packings*, i.e., packings where the centers of the spheres form a (discrete) additive group, or a *lattice*. Such packings can be efficiently described by a generating set of vectors. This regularity is not unique to dimension 2, the densest known packings in dimensions up to 9, and in dimension 24, are attained by lattice packings. Also for higher dimensions lattice packings can be very dense. Their efficient description, and often high density is what makes lattice packings useful for error correcting purposes.

Having a dense lattice packing however is not enough, one also needs to be able to efficiently *decode* the error and recover the original message, such as the rounding algorithm for the integer lattice. While in general this is a very hard problem when the dimension grows to hundreds or even thousands, there do exist dense lattice packings with enough structure to decode efficiently up to a large radius. These well decodable dense lattice packings can thus be used for reliable and efficient communication over noisy channels.

Next to being noisy, the physical world is also inherently unsafe. Letters may be read, conversations can be overheard and wired or wireless channels may be tapped by an eavesdropper. To protect such communication we require *cryptography*. Cryptography allows to hide, or *encrypt* a message, such that no eavesdropper can learn what is communicated, in such a way that the desired receiver can still recover,

or *decrypt* the original message. Modern cryptography can roughly be divided into two categories. If the communicating parties know a common secret key, then there are efficient ways to encrypt and decrypt the message. This is known as *symmetric* cryptography. In case the parties have no common key, we require *asymmetric*, or *public-key* cryptography. In public-key encryption there are two keys, one secret key, known only to the receiver, and one public key, known to all and in particular the sender. A message can be encrypted using the public key, and decrypted using the secret key.

Again, lattice packings and noise can play a central role in building such public-key cryptography. In this case the noise is added deliberately to a message, and the security relies on the general hardness of removing such noise. The idea is to have a good and a bad description of the same lattice, the good one allows to efficiently decode, while the bad one does not. Naturally the good description forms the secret key, while the bad description forms the public key. Someone with only the bad description can 'hide' a message encoded as a lattice point by adding a small error to it, after which the receiver uses the good description to decode the error and recover the original message. To make this safe the good description should be hidden from everyone but the receiver, which for example can be attained by a randomized procedure that generates a lattice packing along with such a good description.

The main advantage of lattice-based cryptography is that, in contrast to currently used public-key cryptography (variants of RSA and Diffie-Hellman), it is believed to be hard to break even for quantum computers. Lattice-based cryptography is currently the main candidate to replace such quantum vulnerable cryptography. It is therefore of fundamental importance to study the (concrete) hardness of these constructions and underlying problems, and to search for potential weaknesses. Such research is better known as *cryptanalysis*, and it is the main topic of Parts II and III of this thesis.

The state-of-the-art cryptanalysis, to which we also contribute in this thesis, shows that the hardness is directly related to the geometry and efficient decoding capability of the lattice packing. The influence of the decoding capability is intuitive: the more noise we can add to the message the better it is hidden, and the harder it is to recover by an eavesdropper. The precise influence of the geometry of the lattice

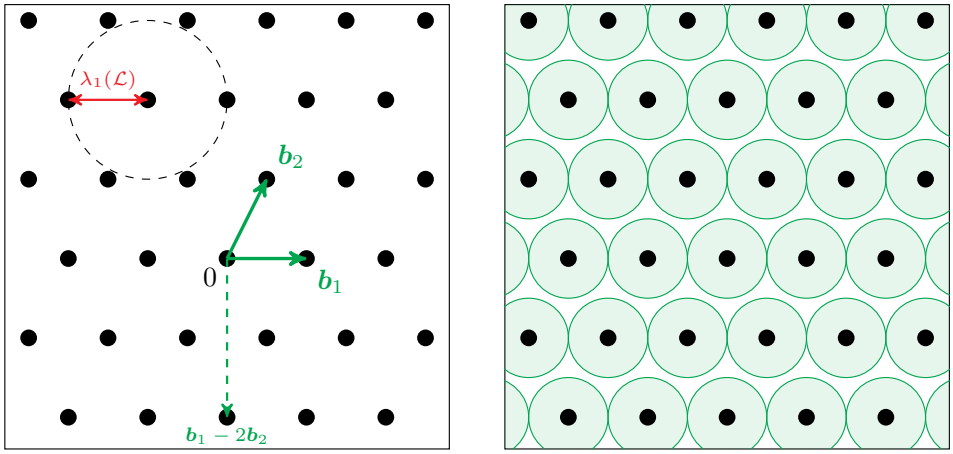


Figure 1.2: The lattice $\mathcal{L}$ with basis $[\mathbf{b}_1, \mathbf{b}_2] = [(1, 0), (\frac{1}{2}, 1)]$ and first minimum $\lambda_1(\mathcal{L}) = 1$. By adding balls of radius $\lambda_1(\mathcal{L})/2$ around each lattice point we obtain a sphere packing.

packing on the security can be much more subtle, and also needs to be taken into account.

Unfortunately, when we look at the current landscape of lattice-based cryptography, through the lens of cryptanalysis, we recognise a fundamental weakness. All currently used lattice packings have a rather poor density or decoding capability, and this is inherent to the randomized procedures that are used to generate a lattice packing along with a good description. In fact, these packings and their decoding algorithms geometrically resemble those of the integer lattice. The error added to hide a message is thus a factor $\Theta(\sqrt{d})$ smaller than optimal, which leads to weaknesses. To compensate for this, one has to increase the dimension, which in turn hurts the efficiency of currently used lattice-based cryptography.

We identify a missed opportunity: denser and efficiently decodable lattice packings are used for error correcting purposes, but not in cryptography. The main obstacle is that the remarkable structure of such lattice packings is assumed to be known to everyone, and thus at first may seem impossible to hide. In Part IV of this thesis we introduce a new foundation, based on lattice isometries, that does allow to hide the remarkable structure, and therefore does enable the use of such dense and efficiently decodable lattice packings in cryptography.

Lattices

In this thesis we define a lattice as a discrete additive subgroup $\mathcal{L} \subset \mathbb{R}^d$ of $\mathbb{R}^d$. An example of a lattice is the subgroup $\mathbb{Z}^d \subset \mathbb{R}^d$ of integer vectors. More generally, given $\mathbb{R}$-linearly independent vectors $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{n \times d}$ the subgroup $\mathcal{L}(\mathbf{B}) := \{\sum_i x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$ consisting of all integer combinations is a lattice. In fact, every lattice can be described by such a basis $\mathbf{B}$, and the number n of basis vectors defines the rank of a lattice. Throughout this introduction we assume that all lattices have full rank $n = d$.

The discrete nature of a lattice implies that there is some minimum distance $\lambda_1(\mathcal{L}) > 0$ between any two distinct lattice points. Due to the additive structure of the lattice we can always assume that one of the two points is the origin $\mathbf{0} \in \mathcal{L}$, and thus $\lambda_1(\mathcal{L}) := \min_{\mathbf{0} \neq \mathbf{v} \in \mathcal{L}} \|\mathbf{v}\|$ can equivalently be defined as the minimum length of any nonzero lattice vector. For example, we have $\lambda_1(\mathbb{Z}^n) = 1$, which is attained by any unit vector $(0, \ldots, 0, \pm 1, 0, \ldots, 0) \in \mathbb{Z}^n$. Geometrically, we can place open balls around each lattice point of radius $\lambda_1(\mathcal{L})/2$ without overlapping them. This turns any lattice into a sphere packing as illustrated in Figure 1.2.

Cryptography and the Quantum Threat

A public-key cryptosystem enables an individual, such as Alice, to securely send a message to another individual, such as Bob, across an untrusted channel so that only Bob can read the message. Any eavesdropper, such as Eve, should not be able to learn Alice's message.

The message Alice wishes to send is encrypted with a public key **pk** that Bob previously announced publicly. She then sends Bob the encrypted message $\mathbf{Enc}_{\mathbf{pk}}(\text{message})$ via the unsafe channel, and Bob uses his secret key **sk** to decrypt it and recover the message (see Figure 1.3).

For security it must be computationally hard to derive the secret key **sk** from the public key. While decryption only works if the secret key and public key are somehow related. In particular the secret key should allow Bob to compute something that no one else can do efficiently.

Historically, the most famous examples of a public-key cryptosystems are Diffie-Hellman and RSA. Almost all currently used public-key cryptography is based on (variants) of these systems. For RSA, the secret key consists of two large primes $\mathbf{sk} = (p, q)$, and the public key is given by $\mathbf{pk} = N = p \cdot q$. To obtain the secret key from the public key one needs to factor N, which is assumed to be computationally hard in the bit-length $\log(N)$. Consider the function $f(x) := x^e \bmod N$ for some exponent $1 < e < \phi(N)$ such that $\gcd(e, \phi(N)) = 1$. Knowledge of the secret key p, q allows one to compute $\phi(N)$ and thus $d := e^{-1} \bmod \phi(N)$. One can then efficiently invert $f(x)$ because $f(x)^d \equiv x \bmod N$. Without the secret key this is seemingly hard[1]. The secret key thus creates a *trapdoor* function f: one can encrypt a message $m \in (\mathbb{Z}/N\mathbb{Z})^*$ using the public key $\mathbf{pk} = N$ as $c = f(m) = m^e \bmod N$, and the secret key $\mathbf{sk} = (p, q)$ allows to invert f and thus recover the original message.

While classically both the Diffie-Hellman and the RSA cryptosystems are yet unbroken, quantumly they are not. Shor's algorithm [Sho94] solves integer factorization and the discrete logarithm problem (underlying Diffie-Hellman) in polynomial time on a quantum computer. Given the recent advent of larger and more stable quantum-computers, we need to replace these conventional problems by new ones that are resistant to quantum attacks, but still allow to create trapdoor functions.

Hard Lattice Problems

To build a trapdoor function we require hard problems, and for lattices there are some natural candidates. The most well-known and simplest to state problem is the Shortest Vector Problem (SVP). As the name suggests it asks you to compute a shortest nonzero lattice vector $\mathbf{v} \in \mathcal{L}$ of length $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$. The input of this problem is any basis of the lattice. If such a basis already consists a shortest lattice vector then the problem is trivial, thus in general you can expect to receive a basis consisting of long vectors.

[1] Technically, knowing p, q is not a necessary condition for breaking RSA, but it is a sufficient one. There do exist (inefficient) variants of RSA where the two are equivalent [Rab79].

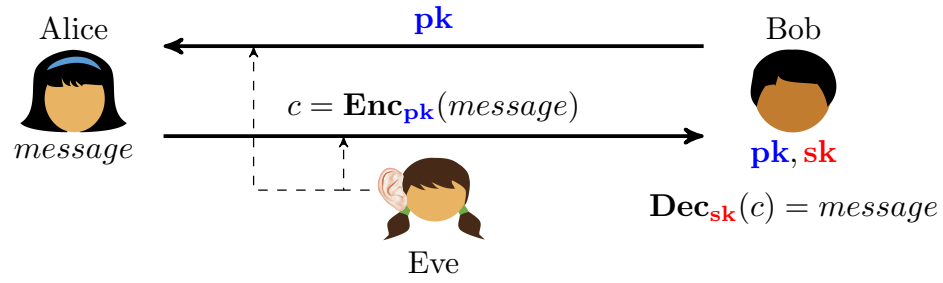


Figure 1.3: A public-key cryptosystem allows Alice to send a message to Bob, such that an eavesdropper Eve cannot learn the message.

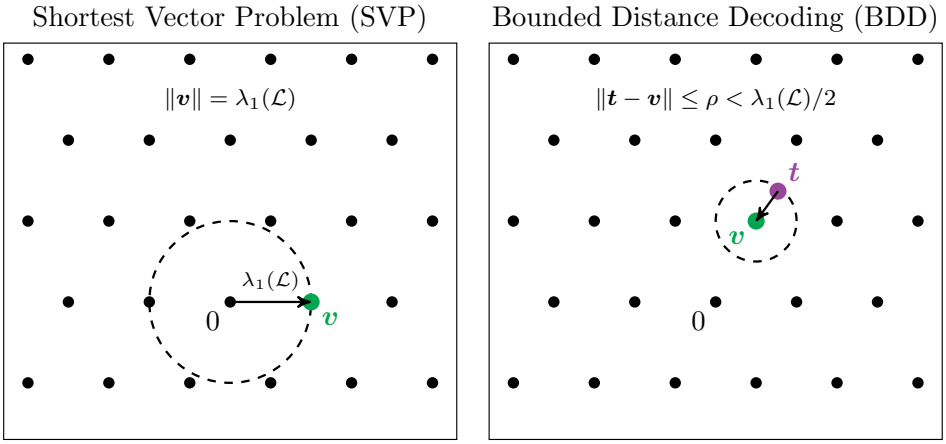


Figure 1.4: Two important lattice problems: the Shortest Vector Problem and Bounded Distance Decoding.

The inhomogeneous variant of SVP is the Closest Vector Problem (CVP). Recall that the lattice $\mathcal{L} \subset \mathbb{R}^n$ lives in a real space $\mathbb{R}^n$. Given any target $\mathbf{t} \in \mathbb{R}^n$ the Closest Vector Problem asks you to compute a lattice vector $\mathbf{v} \in \mathcal{L}$ that minimizes $\|\mathbf{t} - \mathbf{v}\|$. Note that if the distance of $\mathbf{t}$ to a closest lattice vector is strictly less than $\lambda_1(\mathcal{L})/2$ then the solution is unique. Moreover, to simplify the problem one could get the additional promise that the given target lies at distance at most some $\rho < \lambda_1(\mathcal{L})/2$ from the lattice — this is known as Bounded Distance Decoding (BDD). Intuitively, if ρ is very small, i.e., the target lies very close to some lattice vector, then it is easier to recover that lattice

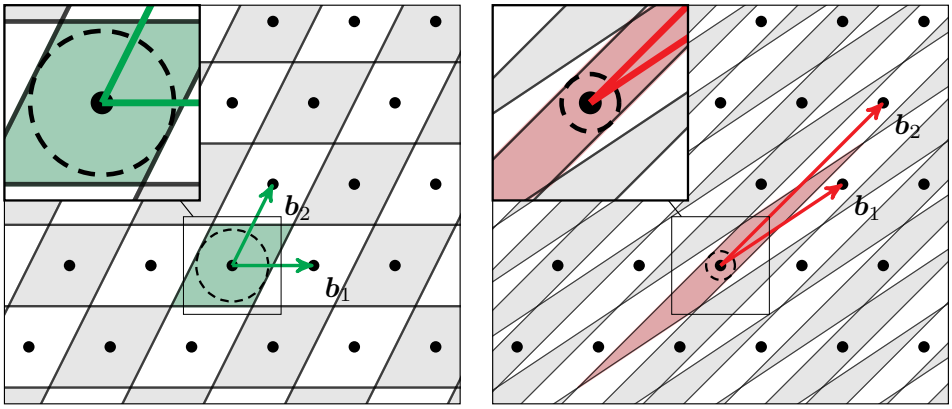


Figure 1.5: Radius for which the basis rounding algorithm solves BDD, for a good basis (left) versus a bad basis (right).

vector.

In two or even three dimensions the above problems are easy, certainly when looking at the examples in Figure 1.4. However the hardness of these problems, that is needed for cryptography, comes from increasing the dimension n to hundreds or even thousands.

Lattice-based Cryptography

Every lattice $\mathcal{L}(\mathbf{B})$ of dimension at least 2 can be described by an infinite number of bases $\mathbf{B}\cdot\mathbf{U}$ for unimodular $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$, i.e., integer $\mathbf{U} \in \mathbb{Z}^{n\times n}$ with $\det(\mathbf{U}) = \pm 1$. We call a basis *good* if it consists of short and somewhat orthogonal vectors, and *bad* if the vectors are long. The hardness of the Shortest Vector Problem indicates that given a bad basis it is hard to compute a good basis. We will use the good and bad bases to create a trapdoor function.

Any basis gives a decoding algorithm. For a target $\mathbf{t} = \mathbf{By} \in \mathbb{R}^n$ we can simply round the coordinate vector $\mathbf{y}$ to obtain a lattice vector $\mathbf{v} = \mathbf{B}\cdot\lfloor\mathbf{y}\rceil \in \mathcal{L}(\mathbf{B})$. Then the error $\mathbf{e} := \mathbf{t} - \mathbf{v}$ lies in the fundamental parallelepiped $\mathcal{P}(\mathbf{B}) := \mathbf{B}\cdot[-\frac{1}{2},\frac{1}{2})^n$ of $\mathbf{B}$. For any $\mathbf{e} \in \mathcal{P}(\mathbf{B})$ we have $\|\mathbf{e}\| \leq \|\mathbf{B}\|\cdot\frac{1}{2}\sqrt{n}$, and thus the shorter the basis, the closer the lattice vector $\mathbf{v}$ lies to $\mathbf{t}$.

Furthermore, let ρ be the largest radius of a ball that is inscribed

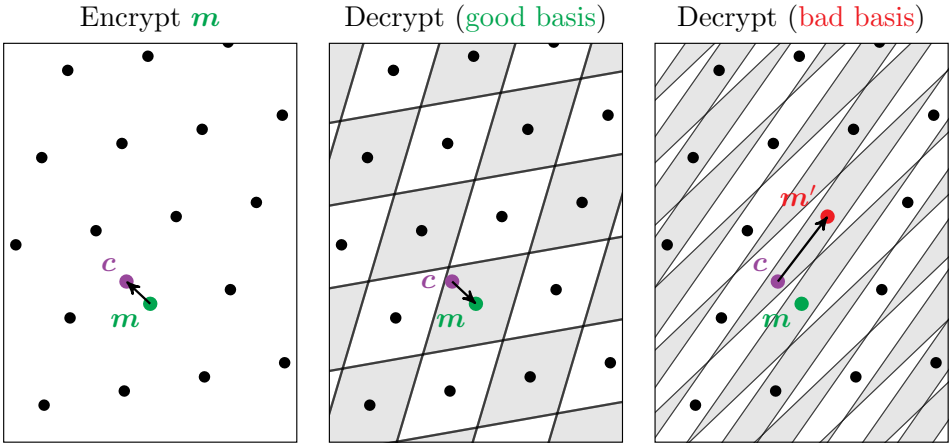
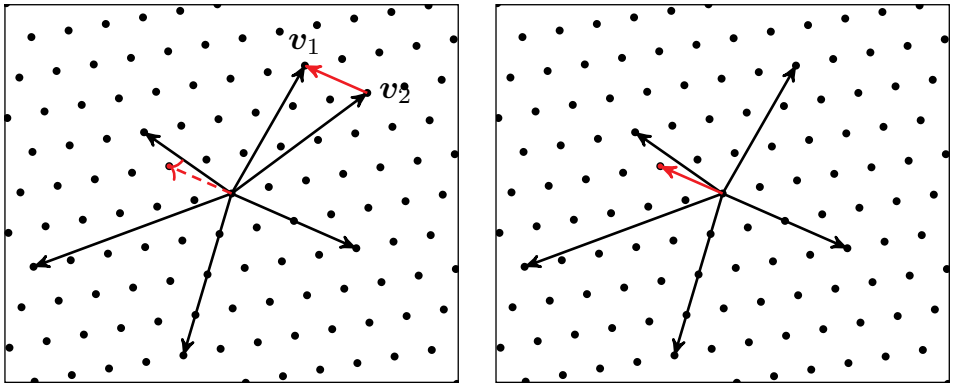


Figure 1.6: Encryption of a message $\mathbf{m} \in \mathcal{L}$ by adding a small error $\mathbf{e}$. Rounding with the good (secret) basis recovers $\mathbf{m}$, while rounding with the bad (public) basis recovers the wrong message $\mathbf{m}' \neq \mathbf{m}$.

in $\mathcal{P}(\mathbf{B})$. Then any BDD instance $\mathbf{t} \in \mathbb{R}^n$ at distance at most ρ from the lattice is decoded correctly to its unique closest vector $\mathbf{c} \in \mathcal{L}$ by the rounding algorithm. For a good basis the decoding radius ρ is generally large, while for a bad basis it is small, see Figure 1.5.

So a (secret) short basis allows us to efficiently recover from small errors, while given a (public) long basis this is a hard problem. The (randomized) trapdoor function now follows naturally. To encrypt a message $\mathbf{m} \in \mathcal{L}$ (described by the bad basis) we simply add a small uniform error $\mathbf{e}$ of length $\|\mathbf{e}\| = \rho$ to the message: $\mathbf{c} := \mathbf{m} + \mathbf{e}$. The rounding procedure together with the secret good basis allows to remove the error and recover $\mathbf{m}$. The same rounding procedure with the bad basis computes a wrong message $\mathbf{m}' \neq \mathbf{m}$ with overwhelming probability for large n, see Figure 1.6. Without the help of a good basis inverting the trapdoor function is a BDD instance, which in general is hard.

The idea behind the good-bad basis encryption scheme is easy to understand, but we did not specify so far how to obtain such a pair of bases. We call this part the key generation and this is precisely where different security assumptions come in. Note that this must be a randomized procedure, as otherwise someone else could just rerun it to obtain the same short basis. In fact, the lattice generated must be



Find close vectors $v_1, v_2 \in L$. Replace $v_2 \leftarrow v_1 - v_2$.

Figure 1.7: Lattice Sieving

random in some sense, as any short basis of the same lattice can be used to decrypt. Almost all of lattice-based cryptography is currently based on a few versatile assumptions that allow one to generate some random lattice along with a (partial) good basis: the Learning With Errors (LWE) assumption, the Short Integer Solution (SIS) assumption and the NTRU assumption. These assumptions are variants of the BDD and SVP problems, for some specific distributions of lattices.

Lattice Sieving and Heuristics

Before considering the security of these schemes directly, we first consider the basic task of computing a shortest vector of a lattice. This will form a building block for later attacks. Lattice sieving algorithms are currently the best way to compute a shortest vector of a lattice in terms of time complexity. They run in single exponential time $2^{O(n)}$ and space $2^{O(n)}$. The central idea of sieving algorithms is to start with a large list of (long) lattice vectors, and to find many sums and differences of these vectors that are shorter. These shorter combinations are inserted back into the list, possibly replacing longer vectors, and this process is repeated until the list contains many short vectors, among which (hopefully) a shortest one of the lattice. One such step is illustrated in Figure 1.7.

The best provable sieving algorithm runs in time $2^{2.456n+o(n)}$ and space $2^{1.233n+o(n)}$ [HPS11b]. Clearly, these algorithms already become infeasible in very low dimensions. However, in practice we can do much better. To understand and analyse these practical sieving algorithms we require the use of heuristics. Such an analysis should be seen as an average-case analysis that might be false in the worst-case. Fortunately, in the cryptanalytic setting these sieving algorithms are executed on lattices that are sufficiently randomized to follow these average-case heuristics. With such a heuristic average-case analysis the best sieving algorithm can be shown to run in time $2^{0.292n+o(n)}$ and space $2^{0.208n+o(n)}$ [BDGL16].

Just as there is a big gap between the complexity of provable and heuristic algorithms, there is also a large knowledge gap between an asymptotic complexity such as $2^{0.292n+o(n)}$ and a concrete estimate for the cost of solving SVP in say dimension 500. For cryptography it is extremely important to understand these concrete costs, as it will directly influence the concrete security.

Cryptanalysis and Basis Reduction

To understand the security of lattice-based schemes we have to know how hard it is to solve the specific lattice problem instances that occur. We have briefly discussed that for a usual average-case lattice the SVP instance in rank n requires about $2^{0.292n+o(n)}$ time to solve. However the lattices that arise from e.g. the LWE, SIS or NTRU assumption are geometrically non-standard. For example they contain either an unusually short vector, a dense sublattice, or the efficient decoding radius ρ is relatively small, see Figure 1.8.

Due to this divergent (secret) structure these instances can be significantly easier than the usual average-case. Often, to break these schemes we only need to find a somewhat short and orthogonal basis. Basis reduction algorithms use an average-case SVP oracle in some lower dimension $\beta \ll n$ to compute such a basis. The larger the parameter β, the better *reduced* the basis gets, but at a cost of solving SVP in dimension β. For large enough β the (secret) structure unravels and the scheme is broken. We observe, that for almost all lattice based schemes that work with n-dimensional lattices, basis reduction

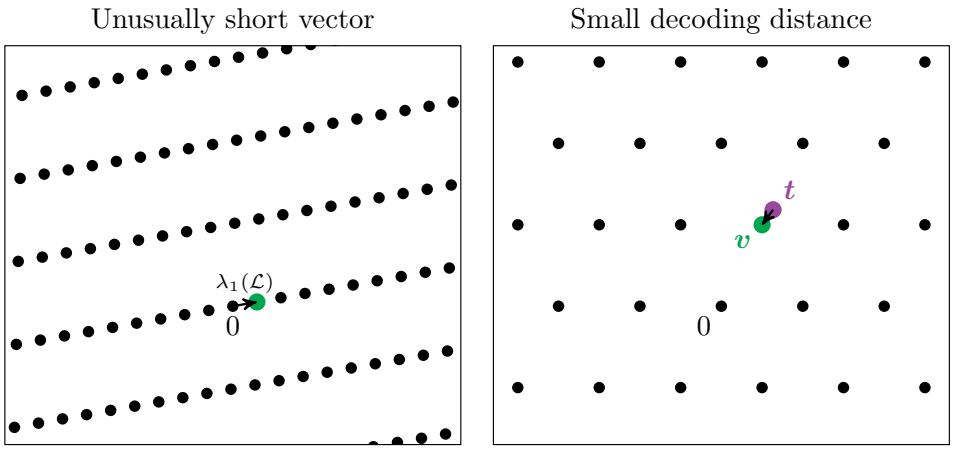


Figure 1.8: On the left a lattice with an unusually short vector (or alternatively a dense rank 1 sublattice), on the right a BDD instance with small decoding radius ρ.

with $\beta \leq n/2 + o(n)$ is sufficient to break them. As a result the lattice dimensions used have to be at least twice as large as what you would expect from the $2^{0.292n+o(n)}$ complexity to solve SVP, leading to schemes with suboptimal performance.

It is not always immediately clear what SVP dimension β is needed to break a scheme. A large task for cryptanalysts is to understand better how the basis reduction algorithm precisely discover the hidden structure. From this understanding we can then derive both asymptotic, but more importantly, concrete estimates for β. In particular, special care should be taken when parameters are taken beyond their usual range, as this might enable new attacks, or the usual attacks might perform better than predicted due to unforeseen weaknesses in the lattice geometry.

The Lattice Isomorphism Problem

Current lattice-based cryptography has several weaknesses that allow to break a scheme based on an n-dimensional lattice by only solving SVP in dimension $\beta \leq n/2 + o(n)$. This stems either from an unbalanced geometry of the lattice (or its dual), from a poor decoding dis-

tance, or from giving away a somewhat good basis. With the currently used techniques of using a trapdoor basis, the decoding distance and unbalanced geometry of the lattice are inherently related. The only way to improve the decoding distance is by making the geometry of the lattice even more unbalanced, and vice versa. The optimal trade-off still falls victim to a basis reduction attack with $\beta = n/2 + o(n)$.

In contrast there do exist wonderful lattices that have efficient decoding algorithms that do not rely on any good basis, but only on the remarkable structure of the lattice in question. Such lattices can offer a much better balance between their decoding distance and their geometry, which might allow to move beyond this $\beta \leq n/2 + o(n)$ barrier.

Now suppose we have such an efficiently decodable lattice with a balanced geometry. Given that (in general) there is no randomness in the generation of this lattice we must assume the lattice and its decoding algorithm is public knowledge. Giving away a bad basis publicly is therefore meaningless, everyone already knows how to decode the lattice. We need some way to hide the remarkable structure of the lattice, without destroying it (as we need it for decoding purposes). The only way to not influence the geometry of a lattice at all is by applying an isometry. This brings us to the lattice isomorphism problem.

Two lattices $\mathcal{L}, \mathcal{L}' \subset \mathbb{R}^n$ are called *isomorphic* or *isometric* if there exists an orthonormal transformation $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$ such that $\mathcal{L}' = \mathbf{O} \cdot \mathcal{L} = \{\mathbf{O} \cdot \mathbf{v} : \mathbf{v} \in \mathcal{L}\}$. See Figure 1.9 for an example of two isomorphic lattices. The Lattice Isomorphism Problem (LIP) asks to compute such a transformation $\mathbf{O}$ given the two lattices $\mathcal{L}, \mathcal{L}'$. The best asymptotic and practical algorithms that solve LIP all require to first compute a shortest vector, even when applied to very simple lattices such as $\mathbb{Z}^n$. LIP therefore gives us a solid foundation for lattice-based cryptography.

We sketch in Figure 1.10 how one would build an encryption scheme build on LIP. First let $\mathcal{L}$ be a lattice in which we can decode efficiently up to some radius $\rho < \lambda_1(\mathcal{L})/2$. We generate some random orthonormal matrix $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$ and give away a bad basis of the rotated lattice $\mathbf{O} \cdot \mathcal{L}$. The *secret key* is the transformation $\mathbf{O}$, while the *public key* is (some description of) the lattice $\mathbf{O} \cdot \mathcal{L}$. Now to encrypt anyone can take a message $\mathbf{m} \in \mathbf{O} \cdot \mathcal{L}$, and add a small uniformly random error $\|\mathbf{e}\| = \rho$ to the message: $\mathbf{c} := \mathbf{m} + \mathbf{e}$. Without knowing the underlying

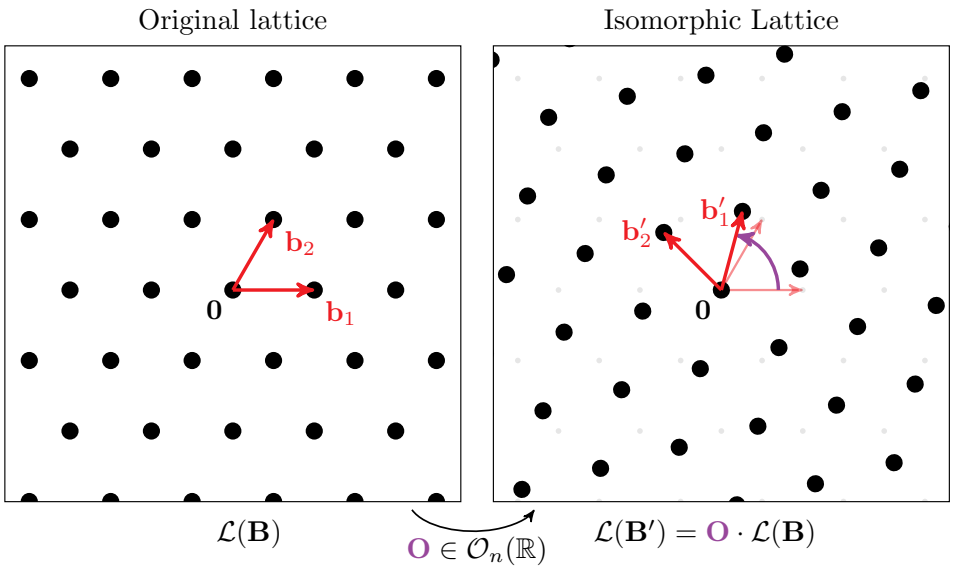


Figure 1.9: The Lattice Isomorphism Problem.

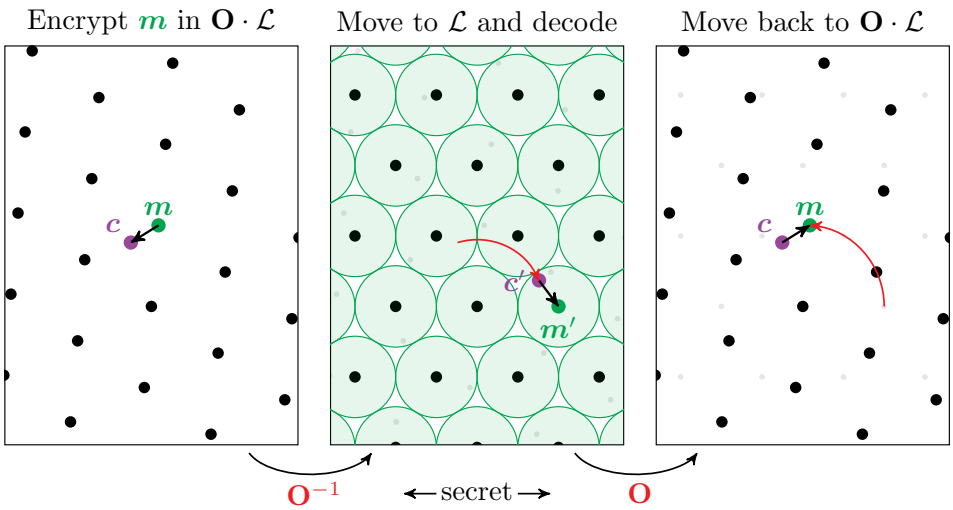


Figure 1.10: Sketch of an encryption scheme based on the lattice isomorphism problem.

structure of $\mathbf{O} \cdot \mathcal{L}$ it is hard to recover the message. However applying the secret transformation we obtain a target $\mathbf{c}' := \mathbf{O}^{-1} \cdot \mathbf{c}$ at distance ρ from the decodable lattice $\mathcal{L}$. We decode here to obtain a lattice vector $\mathbf{m}' \in \mathcal{L}$ and a small error $\|\mathbf{e}'\| \leq \rho$ such that $\mathbf{c}' = \mathbf{m}' + \mathbf{e}'$, which we then map back to the public lattice $\mathbf{O} \cdot \mathcal{L}$. By the uniqueness of the decoding we get that $\mathbf{m} = \mathbf{O} \cdot \mathbf{m}'$, and thus the original message is recovered.

Care has to be taken when formalizing the sketched encryption scheme. Firstly the particular basis chosen to represent the public lattice $\mathbf{O} \cdot \mathcal{L}$ should not leak any information about the original lattice $\mathcal{L}$. This can be solved by defining an appropriate average-case basis distribution, along with a worst-case to average-case reduction. In short, if one can recover $\mathbf{O}$ using the particular average-case description of $\mathbf{O} \cdot \mathcal{L}$, then one can recover $\mathbf{O}$ using any description of $\mathbf{O} \cdot \mathcal{L}$. Secondly the orthonormal transformation $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$, and the lattice $\mathbf{O} \cdot \mathcal{L}$ is defined over the reals and would in practice have to be approximated, say with floating point numbers, leading to loads of technical difficulties. This problem can be sidestepped by working directly with the Gram matrix $\mathbf{B}^\top \mathbf{B}$ instead of a basis $\mathbf{B}$. Given that $(\mathbf{OB})^\top (\mathbf{OB}) = \mathbf{B}^\top \mathbf{B}$ the orthonormal transformation moves out of the picture.

Outline and Contributions

After this introductory chapter, this thesis proceeds with the preliminaries in Chapter 2. Here the basic theory needed in the later parts is introduced. The remainder of the thesis is split into three parts.

Part II - Short and Close Lattice Vectors

Part II covers the Shortest and Closest Vector Problem. Chapter 3 treats the state-of-the-art on lattice sieving. In Chapter 4 we improve on this by implementing the current best lattice sieving algorithms on both CPUs and Graphical Processing Units (GPUs). With these implementations we improve the TU Darmstadt SVP records by 25 dimensions. This corresponds to a gain of about two orders of magnitude over previous records both in terms of wall-clock time and

energy efficiency. These contributions were published at Eurocrypt 2021 [DSW21].

In Chapter 5 we properly model and analyse the behaviour of the randomized iterative slicer algorithm. This is a CVP algorithm that takes as preprocessed input a large list of short lattice vectors. We present the first tight results on the time-memory trade-off for this algorithm. In the full work, published at PKC 2020 [DLW20b], we also show how to significantly reduce the memory usage of this algorithm when using nearest neighbour techniques.

Part III - Basis Reduction

Part III is all about basis reduction. Chapter 6 gives an overview of the state-of-the-art basis reduction algorithms, their heuristic behaviour, and the latest improvements.

Chapter 7 contributes to the state-of-the-art by giving a heuristically tight and concrete estimate for which parameters the BKZ algorithm solves the NTRU problem, backed by many experiments. This work, published at Asiacrypt 2021 [DW21], gives the first concrete and the best asymptotic estimates for the security of NTRU instances with large moduli, and, contrary to prior works, explains *how* the particular structure of the NTRU lattice allows to recover the secret key in this regime.

Chapter 8 introduces the notion of basis reduction to the world of binary codes (with the Hamming metric). The notion of orthogonality in the Euclidean space is ported to a corresponding notion of *orthopodality* for the Hamming metric. An adaptation of Gram-Schmidt Orthogonalisation and the LLL basis reduction algorithm from lattices to binary codes follows directly. In particular the proofs follow analogues to the lattice case. These contributions were published in IEEE Transactions of Information Theory in 2022 [DDW22].

Part IV - The Lattice Isomorphism Problem, Remarkable Lattices and Cryptography

Part IV covers the Lattice Isomorphism Problem (LIP). Chapter 9 starts with an introduction of LIP, explains how the problem is more naturally stated using quadratic forms, and discusses the best asymp-

totic and practical algorithms for solving it. In Section 9.6 a canonical function for lattice isomorphism is introduced together with an algorithm to compute it. This contribution was published at ANTS XIV in 2020 [DHVW20]. Subsequently, in Section 9.7 further practical improvements to this algorithm are discussed, which found application in ongoing work on perfect form enumeration, with the eventual goal of proving what is the densest lattice packing in dimension 9.

The thesis finishes with its main novel contribution in Chapter 10, where LIP is introduced as a new foundation for lattice-based cryptography. This is motivated by the cryptanalysis in previous parts. In particular, and contrary to currently used methods, this new foundation allows the use of remarkable lattices with great geometric and algorithmic properties. The chapter starts by establishing an average-case version of LIP, along with a worst-case to average-case reduction. Then, on this solid base, an identification, key encapsulation and signature scheme is constructed. We continue by discussing cryptanalysis and instantiations. These contributions were published at Eurocrypt 2022 [DW22].

Section 10.8 contains a short summary of the signature scheme Hawk, published at Asiacrypt 2022 [DPPW22]. This scheme, based on the LIP foundation applied to the simple lattice $\mathbb{Z}^n$, improves significantly on Falcon. The lattice $\mathbb{Z}^n$ is geometrically similar to the NTRU lattice used by Falcon. However, the simplicity of $\mathbb{Z}^n$ allows to remove the main disadvantage of Falcon: the use of floating point numbers in signing. Hawk-512 is $4\times$ to $15\times$ faster in signing than Falcon-512 for the AVX2 and reference implementation respectively, while maintaining similar security levels and signature and key sizes.

CHAPTER 2

Preliminaries

2.1 Notation

The notation $x := y$ means that x is defined to be equal to y. The set of all integer, rational and real numbers is denoted by $\mathbb{Z}, \mathbb{Q}$ and $\mathbb{R}$ respectively. For a real number $x \in \mathbb{R}$ we write $\lfloor x \rceil$ for the unique integer such that $x - \lfloor x \rceil \in [-\frac{1}{2}, \frac{1}{2})$. All vectors $\mathbf{x}, \mathbf{y}$ and matrices $\mathbf{A}, \mathbf{B}$ are denoted by bold lower and upper case letters respectively. All vectors are column-vectors and we write $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ for a matrix where the i-th column vector is $\mathbf{b}_i$. For matrices $\mathbf{A}, \mathbf{B}$, we denote the horizontally stacked matrix $\begin{pmatrix} \mathbf{A} \\ \mathbf{B} \end{pmatrix}$, by $[\mathbf{A}; \mathbf{B}]$.

We denote the standard Euclidean inner product of two vectors $\mathbf{v} = (v_1, \ldots, v_d), \mathbf{w} = (w_1, \ldots, w_d) \in \mathbb{R}^d$ by $\langle \mathbf{v}, \mathbf{w} \rangle := \sum_{i=1}^{d} v_i w_i$, and the norm of a vector $\mathbf{v}$ by $\|\mathbf{v}\| := \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$. For a non-empty discrete set $S \subset \mathbb{R}^d$ and a target $\mathbf{t} \in \mathbb{R}^d$ we denote the distance of $\mathbf{t}$ to the set by $\mathrm{dist}(S, \mathbf{t}) := \min_{\mathbf{x} \in S} \|\mathbf{t} - \mathbf{x}\|$. We write $\mathcal{S}^d \subset \mathbb{R}^{d+1}$ and $\mathcal{B}^d \subset \mathbb{R}^d$ for the Euclidean d-sphere and d-ball of radius 1 respectively. For a and b integers with $a \leq b$, we denote by $[\![a, b]\!]$ the set of integers $\{a, a+1, \ldots, b\}$. For a finite set $\mathcal{E}$, we will denote by $|\mathcal{E}|$ its cardinality. We denote the n-dimensional volume of a measurable n-dimensional

body $S \subset \mathbb{R}^d$ by $\mathrm{vol}_n(S)$. If the supposed dimension is clear from the context we just denote $\mathrm{vol}(S)$. For a function $f : A \to B$, and a subset $S \subset A$ we denote $f(S) := \sum_{x \in S} f(x)$.

2.2 Lattices and their properties

2.2.1 Lattice

Definition 1 (Lattice). *A lattice $\mathcal{L} \subset \mathbb{R}^d$ is a discrete subgroup of the vector space $\mathbb{R}^d$; i.e., $\mathcal{L} \subset \mathbb{R}^d$ is non-empty and*

- *is an additive group: for all $\mathbf{v}, \mathbf{w} \in \mathcal{L}$ we have $\mathbf{v} \pm \mathbf{w} \in \mathcal{L}$; and*
- *is discrete: the exists some positive radius $r > 0$ such that all pairs $\mathbf{v}, \mathbf{w} \in \mathcal{L}$ of distinct lattice vectors we have $\|\mathbf{v} - \mathbf{w}\| \geq r$.*

A simple example of a lattice is the orthogonal lattice $\mathbb{Z}^d$, or any subgroup $\mathcal{L} \subset \mathbb{Z}^d$ of it.

Definition 2 (Lattice Basis). *Let $\mathbf{b}_0, \ldots, \mathbf{b}_{n-1} \in \mathbb{R}^d$ be $\mathbb{R}$-linearly independent vectors, we call the matrix $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ a (lattice) basis[1], and we define the lattice $\mathcal{L}(\mathbf{B})$ generated by the $\mathbb{Z}$-span of $\mathbf{B}$ as*

$$\mathcal{L}(\mathbf{B}) := \mathbf{B} \cdot \mathbb{Z}^n = \left\{ \sum_{i=0}^{n-1} x_i \mathbf{b}_i : x_i \in \mathbb{Z}^n \right\}.$$

Lemma 3. *For every $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$, the $\mathbb{Z}$-span $\mathcal{L}(\mathbf{B})$ of $\mathbf{B}$ is a lattice. Conversely, every lattice has a basis.*

Definition 4 (Rank). *The rank $\mathrm{rk}(\mathcal{L})$ of a lattice $\mathcal{L}$ is defined as the $\mathbb{R}$-rank of its $\mathbb{R}$-span $\mathrm{rk}(\mathcal{L}) := \mathrm{rk}(\mathrm{span}\, \mathcal{L})$.*

The rank of a lattice coincides exactly with the number of vectors in any basis of it. Unless stated otherwise any lattice mentioned in the remainder of this thesis is assumed to be of rank at least 1. We call a lattice $\mathcal{L} \subset \mathbb{R}^d$ full rank if its rank equals the dimension d of the vector space $\mathbb{R}^d$. Given a basis $\mathbf{B}$ of a lattice of rank n, all the other bases are of the form $\mathbf{B} \cdot \mathbf{U}$ for a *unimodular* $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$, i.e.,

[1] We start counting at 0 for notational purposes later.

an integer matrix $\mathbf{U} \in \mathbb{Z}^{n\times n}$ with $\det(\mathbf{U}) = \pm 1$. For rank $n \geq 2$ each lattice has an infinite number of distinct bases.

Definition 5 (Primitive). *For a rank n lattice $\mathcal{L} \subset \mathbb{R}^d$, we call a set of vectors $\mathbf{v}_0, \ldots, \mathbf{v}_k \in \mathcal{L}$ primitive if they can be extended to a full basis $[\mathbf{v}_0, \ldots, \mathbf{v}_k, \mathbf{b}_{k+1}, \ldots, \mathbf{b}_{n-1}]$ of $\mathcal{L}$.*

A single vector $\mathbf{v} \in \mathcal{L}$ is primitive if it cannot be scaled down to another lattice vector, i.e. if there exists no $\lambda > 1$ such that $\mathbf{v}/\lambda \in \mathcal{L}$.

Another way to capture the geometry of a lattice is by a Gram matrix.

Definition 6 (Gram). *For a lattice basis $\mathbf{B}$ of rank n, we define its Gram matrix $\mathbf{G} \in \mathbb{R}^{n\times n}$ as the (positive definite) matrix consisting of all pairwise inner products*

$$\mathbf{G} := \mathbf{B}^\top \mathbf{B} = (\langle \mathbf{b}_i, \mathbf{b}_j \rangle)_{0 \leq i,j < n}.$$

We denote $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{G}} := \mathbf{x}^\top \mathbf{G} \mathbf{y}$, and $\|\mathbf{x}\|_{\mathbf{G}} := \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle_{\mathbf{G}}}$.

For any basis $\mathbf{B}$, its gram matrix $\mathbf{G} = \mathbf{B}^\top \mathbf{B}$, and any two vectors $\mathbf{v} = \mathbf{B}\mathbf{x}, \mathbf{w} = \mathbf{B}\mathbf{y} \in \operatorname{span}(\mathcal{L})$, we have

$$\langle \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{B}\mathbf{x}, \mathbf{B}\mathbf{y} \rangle = \mathbf{x}^\top (\mathbf{B}^\top \mathbf{B}) \mathbf{y} = \mathbf{x}^\top \mathbf{G} \mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{G}}.$$

The Gram matrix stores all geometric information, but not the particular embedding of the lattice into $\mathbb{R}^d$, e.g., for any orthonormal transformation $\mathbf{O} \in \mathcal{O}_d(\mathbb{R})$, the bases $\mathbf{B}$ and $\mathbf{OB}$ have the same Gram matrix.

2.2.2 Properties

We discuss some important properties of a lattice. Because a lattice $\mathcal{L}$ is a normal subgroup of $\operatorname{span}(\mathcal{L})$, we can consider their quotient group.

Definition 7 (Lattice Torus). *For a lattice $\mathcal{L}$ we define its lattice torus $\mathbb{T}(\mathcal{L})$ as*

$$\mathbb{T}(\mathcal{L}) := (\operatorname{span} \mathcal{L})/\mathcal{L}.$$

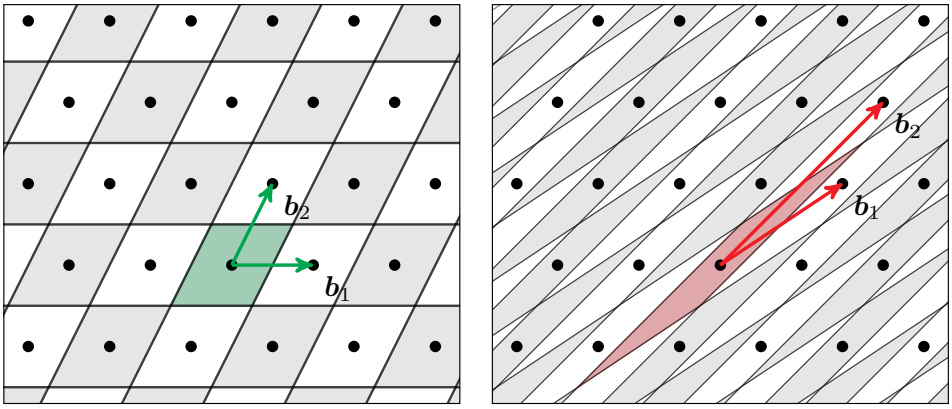


Figure 2.1: The fundamental domain $\mathcal{P}(\mathbf{B})$ for two different bases. They both have the same volume $\mathrm{vol}(\mathcal{L})$ and tile the space.

One can view $\mathbb{T}(\mathcal{L})$ as the span $\mathrm{span}(\mathcal{L})$ up to the translation by lattice vectors. We call a set $\mathcal{P} \subset \mathrm{span}(\mathcal{L})$, that represent each element of $\mathbb{T}(\mathcal{L})$ precisely once, a *fundamental domain* of the lattice $\mathcal{L}$. Geometrically, translated copies $\mathbf{v} + \mathcal{P}$ for $\mathbf{v} \in \mathcal{L}$ give a tiling of $\mathrm{span}(\mathcal{L})$.

For any basis $\mathbf{B}$ of a rank n lattice $\mathcal{L}$, the map $\mathbf{x} \mapsto \mathbf{Bx}$ gives an isomorphism from $\mathbb{R}^n/\mathbb{Z}^n$ to $\mathbb{T}(\mathcal{L})$, so $\mathbb{T}(\mathcal{L})$ can be viewed as a n-torus. Since $[-\frac{1}{2}, \frac{1}{2})^n$ is a fundamental domain of $\mathbb{Z}^n \subset \mathbb{R}^n$, the parallelepiped $\mathbf{B} \cdot [-\frac{1}{2}, \frac{1}{2})^n$ is a fundamental domain of $\mathcal{L}$.

Definition 8. *For a basis $\mathbf{B}$ of a rank n lattice $\mathcal{L} \subset \mathbb{R}^d$ we define the fundamental parallelepiped $\mathcal{P}(\mathbf{B})$ as*

$$\mathcal{P}(\mathbf{B}) = \mathbf{B} \cdot [-\tfrac{1}{2}, \tfrac{1}{2})^n = \left\{ \sum_{i=0}^{n-1} x_i \mathbf{b}_i : x_i \in [-\tfrac{1}{2}, \tfrac{1}{2}) \right\},$$

which is a fundamental domain of $\mathcal{L}$, i.e., a set of representatives of the quotient group $\mathbb{T}(\mathcal{L})$.

The volume $\mathrm{vol}(\mathcal{P}(\mathbf{B}) = \mathrm{vol}(\mathbb{T}(\mathcal{L}))$ of such a fundamental domain does not depend on the basis $\mathbf{B}$ of $\mathcal{L}$, and is called the (co)volume of a lattice.

Definition 9 ((Co)volume). *The (co)volume* $\mathrm{vol}(\mathcal{L})$ *of a lattice* $\mathcal{L}$ *is defined as*

$$\mathrm{vol}(\mathcal{L}) := \mathrm{vol}((\mathrm{span}\,\mathcal{L})/\mathcal{L}) = \det(\mathbf{B}^\top\mathbf{B})^{1/2},$$

which is independent of the basis $\mathbf{B}$ *of* $\mathcal{L}$.

The discreteness of a lattice implies that all distinct lattice vectors have at least some minimum distance $\lambda_1(\mathcal{L})$ between them. Equivalently, by the additivity, we only have to look at the minimum distance between the origin and any nonzero lattice vector.

Definition 10 (First Minimum). *The first minimum* $\lambda_1(\mathcal{L}) \in \mathbb{R}_{>0}$ *of a lattice* $\mathcal{L}$ *is defined as the minimal norm of a nonzero lattice vector*

$$\lambda_1(\mathcal{L}) := \min_{\mathbf{v}\in\mathcal{L}\setminus\{0\}} \|\mathbf{v}\|\,.$$

We introduced the volume of a lattice and its first minimum. Minkowski proved, already in 1889, that these two properties are related in the following way.

Theorem 11 (Minkowski's Theorem). *For a lattice* $\mathcal{L}$ *of rank* n *we have the following bound on the first minimum*

$$\lambda_1(\mathcal{L}) \le 2\cdot\frac{\mathrm{vol}(\mathcal{L})^{1/n}}{(\mathrm{vol}\,\mathcal{B}^n)^{1/n}} \le \sqrt{n}\cdot\mathrm{vol}(\mathcal{L})^{1/n}.$$

Proof. Let us assume without loss of generality that the lattice is full rank, i.e., $\mathrm{span}(\mathcal{L}) = \mathbb{R}^n$. We will give a rather visual proof for the first inequality, following Figure 2.2. For this consider the Voronoi domain $\mathcal{V}(\mathcal{L})$, which contains all vectors in the span of the lattice for which $\mathbf{0}$ is a closest lattice vector, namely

$$\mathcal{V}(\mathcal{L}) := \{\mathbf{v}\in\mathbb{R}^n : \|\mathbf{v}\| = \mathrm{dist}(\mathcal{L},\mathbf{v})\}.$$

See the grey box in Figure 2.2 as an example. Because every vector $\mathbf{t}\in\mathbb{R}^n$ has at least one closest lattice vector, we obtain a covering $\mathcal{V}(\mathcal{L})+\mathcal{L}$ of $\mathbb{R}^n$. Furthermore, the intersection $(\mathbf{v}_1+\mathcal{V}(\mathcal{L}))\cap(\mathbf{v}_2+\mathcal{V}(\mathcal{L}))$ for distinct lattice vectors $\mathbf{v}_1\neq\mathbf{v}_2$, is by definition contained in the rank $n-1$ hyperplane of vectors that are equidistant to both $\mathbf{v}_1$ and $\mathbf{v}_2$. In particular, their overlap has trivial n-dimensional volume. So

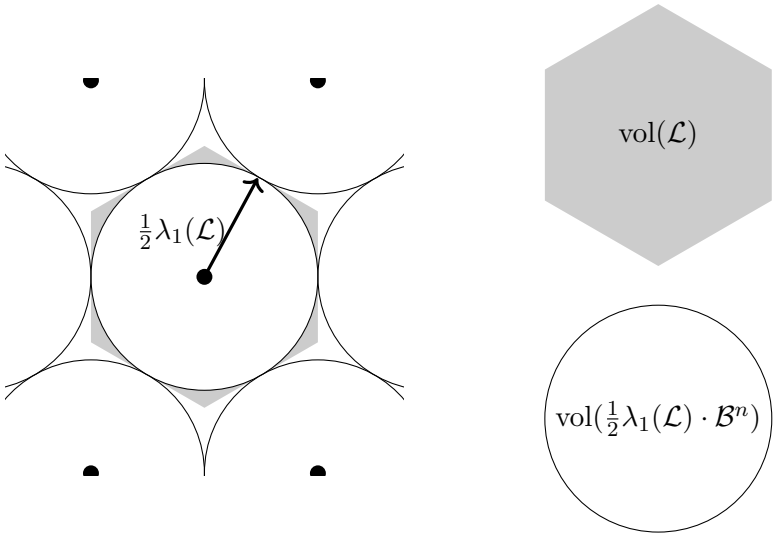


Figure 2.2: Visual proof of Minkowski's Theorem.

the translated Voronoi domains $\mathcal{L} + \mathcal{V}(\mathcal{L})$ form an essentially disjoint tiling of $\mathbb{R}^n$. We can conclude that $\mathrm{vol}(\mathcal{V}(\mathcal{L})) = \mathrm{vol}(\mathcal{L})$.

Since the ball $\frac{1}{2}\lambda_1(\mathcal{L}) \cdot \mathcal{B}^n$ of radius $\frac{1}{2}\lambda_1(\mathcal{L})$ is by definition included in $\mathcal{V}(\mathcal{L})$, we have

$$\tfrac{1}{2}\lambda_1(\mathcal{L}) \cdot \mathrm{vol}(\mathcal{B}^n)^{1/n} = \mathrm{vol}(\tfrac{1}{2}\lambda_1(\mathcal{L}) \cdot \mathcal{B}^n)^{1/n} \leq \mathrm{vol}(\mathcal{V}(\mathcal{L})) = \mathrm{vol}(\mathcal{L}),$$

and the first part of the theorem follows. The second inequality follows by lower bounding the volume $\mathrm{vol}\,\mathcal{B}^n$. □

By the Minkowski's Theorem we can also ask for each rank n what the largest possible ratio is for $\lambda_1(\mathcal{L})/\mathrm{vol}(\mathcal{L})^{1/n}$. The lattice attaining the maximal ratio corresponds to the best n dimensional lattice packing.

Definition 12 (Hermite's constant). *The Hermite constant γ_n is defined as the maximal ratio*

$$\gamma_n := \max_{\mathcal{L}:\mathrm{rk}(\mathcal{L})=n} \frac{\lambda_1(\mathcal{L})^2}{\mathrm{vol}(\mathcal{L})^{2/n}},$$

over all rank n lattices.

We have $\gamma_2 = \frac{4}{3}$ and the maximum is attained by the hexagonal lattice in Figure 2.2. More generally, we can define the n successive minima $\lambda_1(\mathcal{L}), \ldots, \lambda_n(\mathcal{L})$ of a rank n lattice.

Definition 13 (Successive Minima). *For $1 \leq i \leq n$ the i-th minimum $\lambda_i(\mathcal{L})$ of an n-dimensional lattice $\mathcal{L}$ is defined as*

$$\lambda_i(\mathcal{L}) := \min\{\lambda > 0 : \dim(\mathcal{B}_\lambda^n \cap \mathcal{L}) \geq i\}.$$

2.2.3 Dual lattice

Any lattice $\mathcal{L}$ has a corresponding dual lattice.

Definition 14 (Dual Lattice). *For a lattice $\mathcal{L}$ we define the dual lattice, denoted by $\mathcal{L}^*$, as*

$$\mathcal{L}^* := \{\mathbf{w} \in \operatorname{span}(\mathcal{L}) : \langle \mathbf{v}, \mathbf{w} \rangle \in \mathbb{Z} \text{ for all } \mathbf{v} \in \mathcal{L}\}.$$

Given any basis $\mathbf{B}$ of $\mathcal{L}$, the dual lattice $\mathcal{L}^*$ has a unique *dual basis* $\mathbf{D}$ of $\mathbf{B}$ such that $\mathbf{D}^\top \mathbf{B} = \mathbf{I}_n$. Explicitly we have $\mathbf{D} = (\mathbf{B}^\top)^{-1}$ for a full rank lattice, and $\mathbf{D} = \mathbf{B}(\mathbf{B}^\top \mathbf{B})^{-1}$ for the general case. The dual of a dual lattice is again the primal lattice $(\mathcal{L}^*)^* = \mathcal{L}$. The volumes of a lattice and its dual are also related.

Lemma 15 (Dual Volume). $\operatorname{vol}(\mathcal{L}^*) = 1/\operatorname{vol}(\mathcal{L})$

Next to their volume, a lattice and its dual have more subtle geometric connections, known as transference relations. We consider one of them by Banaszczyk [Ban93].

Theorem 16 (Transference bound [Ban93]). *For a lattice $\mathcal{L}$ of rank n and its dual $\mathcal{L}^*$ we have*

$$1 \leq \lambda_1(\mathcal{L}) \cdot \lambda_n(\mathcal{L}^*) \leq n.$$

Note that from applying Minkowski's Theorem 11 both to the primal and the dual lattice, and by using Lemma 15, we can already obtain $\lambda_1(\mathcal{L}) \cdot \lambda_1(\mathcal{L}^*) \leq n$. Theorem 16 is a strengthening of this result.

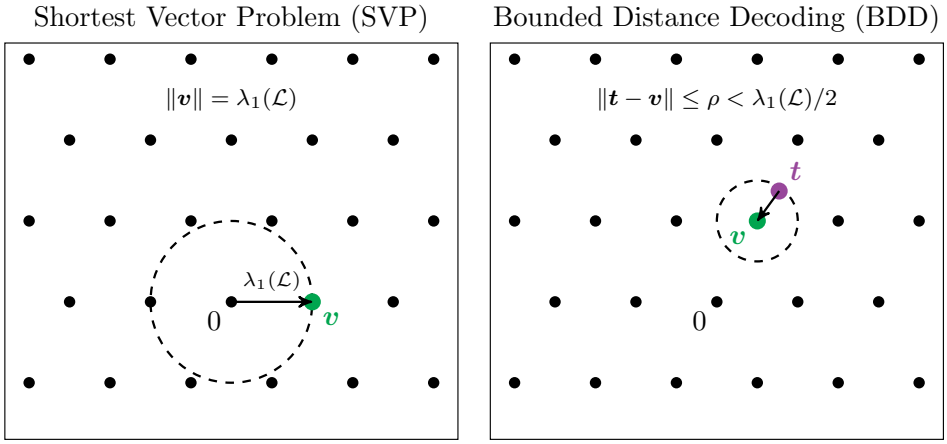


Figure 2.3: The Shortest Vector Problem and Bounded Distance Decoding.

2.3 Lattice problems

2.3.1 Short and close vectors

Cryptography relies on hard problems. For lattices the most famous of them all is the Shortest Vector Problem (SVP).

Definition 17 (Shortest Vector Problem (SVP)). *Given as input a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, compute a nonzero vector* $\mathbf{v} \in \mathcal{L}$ *of length* $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$.

The Closest Vector Problem could be interpreted as the inhomogeneous version of SVP.

Definition 18 (Closest Vector Problem (CVP)). *Given as input a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, and a target vector* $\mathbf{t} \in \mathrm{span}(\mathcal{L})$*, compute a lattice vector* $\mathbf{c} \in \mathcal{L}$ *closest to* $\mathbf{t}$*, i.e.,* $\|\mathbf{t} - \mathbf{c}\| = \mathrm{dist}(\mathcal{L}, \mathbf{t})$.

We denote oracles solving SVP and CVP by $\mathbf{v} = \mathbf{SVP}(\mathcal{L})$, and $\mathbf{c} = \mathbf{CVP}(\mathcal{L}, \mathbf{t})$.

2.3.2 Approximate variants

Instead of the exact versions of the lattice problems, it is often enough to solve an approximate version to break lattice-based schemes.

Definition 19 (Approximate Shortest Vector Problem (γ-SVP)). *Given as input a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, and an approximation factor* $\gamma \geq 1$*, compute a nonzero vector* $\mathbf{v} \in \mathcal{L}$ *of length* $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$*.*

Definition 20 (Approximate Closest Vector Problem (γ-CVP)). *Given as input a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, a target vector* $\mathbf{t} \in \mathbb{R}^d$ *and an approximation factor* $\gamma \geq 1$*, compute a vector* $\mathbf{v} \in \mathcal{L}$ *such that* $\|\mathbf{t} - \mathbf{v}\| \leq \gamma \cdot \mathrm{dist}(\mathcal{L}, \mathbf{t})$*.*

The above problems can be hard to verify as $\lambda_1(\mathcal{L})$ and $\mathrm{dist}(\mathcal{L}, \mathbf{t})$ might be unknown. A common way to state the above problems such that they become efficiently verifiable is as follows.

Definition 21 (Hermite SVP (γ-HermiteSVP)). *Given as input a basis* $\mathbf{B}$ *of a rank* n *lattice* $\mathcal{L}$*, an approximation factor* $\gamma > 0$*, compute a nonzero vector* $\mathbf{v} \in \mathcal{L}$ *of length* $\|\mathbf{v}\| \leq \gamma \cdot \det(\mathcal{L})^{1/n}$*.*

For small values of γ it is possible that no solution exists. By Minkowski's Theorem 11 there does exist a solution to γ-HermiteSVP when $\gamma \geq \sqrt{n}$.

Definition 22 (Hermite CVP (γ-HermiteCVP)). *Given a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, a target vector* $\mathbf{t} \in \mathbb{R}^d$ *and an approximation factor* $\gamma > 0$*, compute a vector* $\mathbf{v} \in \mathcal{L}$ *such that* $\|\mathbf{t} - \mathbf{v}\| \leq \gamma \cdot \det(\mathcal{L})^{1/n}$*.*

2.3.3 Unique variants

The approximate variants above can have none or multiple solutions for certain inputs. Here we discuss some variants that are promised to always have precisely one unique solution (up to sign).

Definition 23 (Unique SVP (γ-uniqSVP)). *Given as input a factor* $\gamma > 1$ *and a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$ *satisfying* $\lambda_2(\mathcal{L}) \geq \gamma \cdot \lambda_1(\mathcal{L})$*, compute the (unique up to sign) nonzero vector* $\mathbf{v} \in \mathcal{L}$ *of length* $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$*.*

The Bounded Distance Decoding (BDD) problem is a variant of CVP. To make the solution unique the target is promised to lie at a distance less than $\frac{1}{2}\lambda_1(\mathcal{L})$ from the lattice. Two distinct solutions $\mathbf{v}, \mathbf{w} \in \mathcal{L}$ would give a nonzero lattice vector of length $\|\mathbf{v} - \mathbf{w}\| \leq \|\mathbf{v} - \mathbf{t}\| + \|\mathbf{t} - \mathbf{w}\| < \lambda_1(\mathcal{L})$, which gives a contradiction.

Definition 24 (Bounded Distance Decoding (δ-BDD)). *Given as input a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, a distance promise* $\delta \in [0, \frac{1}{2})$*, and a target vector* $\mathbf{t} \in \mathrm{span}(\mathcal{L})$ *satisfying* $\mathrm{dist}(\mathcal{L}, \mathbf{t}) \leq \delta \cdot \lambda_1(\mathcal{L})$*, compute the (unique) closest lattice vector* $\mathbf{c} \in \mathcal{L}$ *to* $\mathbf{t}$*, i.e.,* $\|\mathbf{t} - \mathbf{c}\| = \mathrm{dist}(\mathcal{L}, \mathbf{t})$.

These two problems are very much related. For any $\gamma > 1$, there is a polynomial time reduction from $(\frac{1}{2\gamma})$-BDD to γ-uniqSVP, and for any polynomially bounded γ there is a polynomial time reduction from γ-uniqSVP to $(1/\gamma)$-BDD [LM09]. Essentially, (up to a small approximation factor 2), these problems are equivalent. Note that γ-uniqSVP becomes easier for large γ, while δ-BDD becomes easier for small δ.

2.4 Projecting

Projections are a fundamental tool in lattice theory and lattice algorithms. They naturally reduce high-dimensional (lattice) problems to lower-dimensional ones.

Definition 25 (Projection). *Let* $V \subset \mathbb{R}^d$ *be a linear subspace. Any element* $\mathbf{x} \in \mathbb{R}^d$ *can uniquely be written as* $\mathbf{x} = \mathbf{x}_V + \mathbf{x}_{V^\perp}$ *where* $\mathbf{x}_V \in V$ *and* $\mathbf{x}_{V^\perp} \in V^\perp$*. We define the (orthogonal) projection* $\pi_V : \mathbb{R}^d \to \mathbb{R}^d$ *onto* V *as the linear map*

$$\pi_V : \mathbf{x} \mapsto \mathbf{x}_V \text{ for all } \mathbf{v} \in \mathbb{R}^d,$$

and the orthogonal projection $\pi_{V^\perp} : \mathbb{R}^d \to \mathbb{R}^d$ *away from* V *as the linear map*

$$\pi_{V^\perp} : \mathbf{x} \mapsto \mathbf{x}_{V^\perp} \text{ for all } \mathbf{v} \in \mathbb{R}^d.$$

Note that it is only meaningful to use a projection once, i.e., for any $\mathbf{v} \in \mathbb{R}^d$ we have that $\pi^2(\mathbf{v}) = \pi(\mathbf{v})$. Projections are useful because they not only decrease the dimensionality, but also the length of elements.

Lemma 26. *For any orthogonal projection $\pi : \mathbb{R}^d \to \mathbb{R}^d$, and any $\mathbf{v} \in \mathbb{R}^d$ we have*

$$\|\pi(\mathbf{v})\| \leq \|\mathbf{v}\|.$$

2.4.1 Lattice projections

When projecting lattices we do want the discrete lattice structure to be preserved, which is not in general the case. This only happens if the projection is properly aligned with the lattice. One such a properly aligned example is a projection away from a lattice vector.

Lemma 27. *Let $\mathcal{L} \subset \mathbb{R}^d$ be a lattice. For any vector $\mathbf{v} \in \mathcal{L}$, let $\pi_{\mathbf{v}^\perp}$ be the projection away from $\mathbf{v}$, then $\pi_{\mathbf{v}^\perp}(\mathcal{L})$ is again a lattice (possibly of rank 0).*

Proof. Suppose $\mathcal{L}$ is of rank $n \geq 1$, and assume without loss of generality that $\mathbf{v}$ is scaled such that it is a primitive lattice vector. We can extend $\mathbf{v}$ to a basis $[\mathbf{v}, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}]$ of $\mathcal{L}$. Every element $\mathbf{w} \in \mathcal{L}$ can thus be uniquely written as

$$\mathbf{w} = x_0 \cdot \mathbf{v} + \sum_{i=1}^{n-1} x_i \mathbf{b}_i,$$

with $x_i \in \mathbb{Z}$. After projecting we have

$$\pi_{\mathbf{v}^\perp}(\mathbf{w}) = \sum_{i=1}^{n-1} x_i \pi_{\mathbf{v}^\perp}(\mathbf{b}_i).$$

The vectors $\pi_{\mathbf{v}^\perp}(\mathbf{b}_1), \ldots, \pi_{\mathbf{v}^\perp}(\mathbf{b}_{n-1})$ are $\mathbb{R}$-linearly independent, because $\mathbf{v}, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}$ are, and thus they form a basis $\mathbf{B}'$. We can conclude that $\pi_{\mathbf{v}^\perp}(\mathcal{L}) = \{\sum_{i=1}^{n-1} x_i \pi_{\mathbf{v}^\perp}(\mathbf{b}_i) : x_i \in \mathbb{Z}\} = \mathcal{L}(\mathbf{B}')$ is a lattice. □

Lemma 27 shows that we can project away from a lattice vector, and keep the discrete lattice structure. In the proof we use the fact that $\mathcal{L}$ has a basis starting with (a primitive multiple of) $\mathbf{v}$. In other words, we only have to consider the case where we project away from the first basis vector. Similarly, for the more general case, we only have to consider the case where we project away from the first i basis vectors. A useful object to consider in this setting is the Gram-Schmidt Orthogonalisation (GSO) of a basis.

Definition 28 (Gram-Schmidt Orthogonalisation (GSO)). *Let* $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ *be a basis. The Gram-Schmidt Orthogonalisation of* $\mathbf{B}$*, denoted by* $\tilde{\mathbf{B}} = [\tilde{\mathbf{b}}_0, \ldots, \tilde{\mathbf{b}}_{n-1}]$*, is defined as*

$$\begin{aligned}\tilde{\mathbf{b}}_0 &= \mathbf{b}_0\\ \tilde{\mathbf{b}}_i &= \mathbf{b}_i - \pi_{\mathrm{span}(\mathbf{b}_0,\ldots,\mathbf{b}_{i-1})}(\mathbf{b}_i) = \pi_i(\mathbf{b}_i),\end{aligned}$$

where $\pi_i := \pi_{\mathrm{span}(\mathbf{b}_0,\ldots,\mathbf{b}_{i-1})^\perp}$.

The GSO $\tilde{\mathbf{B}}$ of a basis $\mathbf{B}$ is in general not a basis of the same lattice, but it is a basis of the $\mathbb{R}$-span $\mathrm{span}(\mathcal{L}(\mathbf{B}))$. The GSO has a connection with the $\mathbf{QR}$ factorization of the basis $\mathbf{B}$, where $\mathbf{Q}$ is orthonormal and $\mathbf{R}$ is upper triangular, namely the diagonal of $\mathbf{R}$ then consists of $\|\tilde{\mathbf{b}}_0\|, \ldots, \|\tilde{\mathbf{b}}_{n-1}\|$. The same is true for the upper triangular matrix $\mathbf{C}$ in the Cholesky decomposition $\mathbf{G} = \mathbf{C}^\top\mathbf{C}$ of the Gram matrix $\mathbf{G}$ of $\mathbf{B}$. In later chapters the GSO norms $(\|\tilde{\mathbf{b}}_0\|, \ldots, \|\tilde{\mathbf{b}}_{n-1}\|)$ will play an important role, and we call them the *profile* of a basis.

Lemma 29. *Let* $\mathcal{L}$ *be a lattice with basis* $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$*, then* $\pi_i(\mathcal{L})$ *is a lattice for* $0 \leq i < n$*, with basis* $[\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_{n-1})]$ *and GSO* $[\tilde{\mathbf{b}}_i, \ldots, \tilde{\mathbf{b}}_{n-1}]$.

Proof. We prove this by induction on $i = 0, \ldots, n-1$. For $i = 0$, π_0 is the identity, and the result is trivially true. For $0 < i = j + 1 \leq n - 1$, we assume by the induction hypothesis that $\mathcal{L}' = \pi_j(\mathcal{L})$ is a lattice, with basis $[\pi_j(\mathbf{b}_j), \ldots, \pi_j(\mathbf{b}_{n-1})]$, and GSO $[\tilde{\mathbf{b}}_j, \ldots, \tilde{\mathbf{b}}_{n-1}]$. The first basis vector of $\mathcal{L}'$ is $\pi_j(\mathbf{b}_j) = \tilde{\mathbf{b}}_j$. Note that $\pi_{j+1} = \pi_{\tilde{\mathbf{b}}_j^\perp} \circ \pi_j$. In particular

$$\pi_{j+1}(\mathcal{L}) = \pi_{\tilde{\mathbf{b}}_j^\perp}(\pi_j(\mathcal{L})) = \pi_{\tilde{\mathbf{b}}_j^\perp}(\mathcal{L}'),$$

which is a lattice by Lemma 27. Furthermore, from the proof of Lemma 27 it follows that $\pi_{\tilde{\mathbf{b}}_j^\perp}(\mathcal{L}')$ has basis

$$[\pi_{\tilde{\mathbf{b}}_j^\perp}(\pi_j(\mathbf{b}_{j+1})), \ldots, \pi_{\tilde{\mathbf{b}}_j^\perp}(\pi_j(\mathbf{b}_{n-1}))] = [\pi_{j+1}(\mathbf{b}_i), \ldots, \pi_{j+1}(\mathbf{b}_{n-1})].$$

The GSO is clear from its definition. We can conclude by induction. □

For any rank n lattice $\mathcal{L}$, for which the basis $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ is clear from the context, we denote the rank $n - l$ projected lattice

$\pi_l(\mathcal{L}(\mathbf{B}))$ by $\mathcal{L}_{[l:n)}$. We refer to $[l:n)$ as the *context* in which we work, and we call $[0:n)$ the *full context*. Similarly, we denote the rank $0 < r \leq n$ sublattice $\mathcal{L}([\mathbf{b}_0, \ldots, \mathbf{b}_{r-1}]) \subset \mathcal{L}(\mathbf{B})$, by $\mathcal{L}_{[0:r)}$, and the corresponding context by $[0:r)$. Projections and sublattices can be combined, we denote for $0 \leq l < r \leq n$ the basis $[\pi_l(\mathbf{b}_l), \ldots, \pi_l(\mathbf{b}_{r-1})]$ by $\mathbf{B}_{[l:r)}$. We call the rank $r-l$ lattice generated by this basis a *projected sublattice* $\mathcal{L}_{[l:r)} := \mathcal{L}(\mathbf{B}_{[l:r)})$ of $\mathcal{L}$, and refer to the context as $[l:r)$. Furthermore, if the GSO of the full basis $\mathbf{B}$ is $\tilde{\mathbf{B}} := [\tilde{\mathbf{b}}_0, \ldots, \tilde{\mathbf{b}}_{n-1}]$, then the GSO of $\mathbf{B}_{[l:r)}$ is given by $\tilde{\mathbf{B}}_{[l:r)} := [\tilde{\mathbf{b}}_i, \ldots, \tilde{\mathbf{b}}_{j-1}]$.

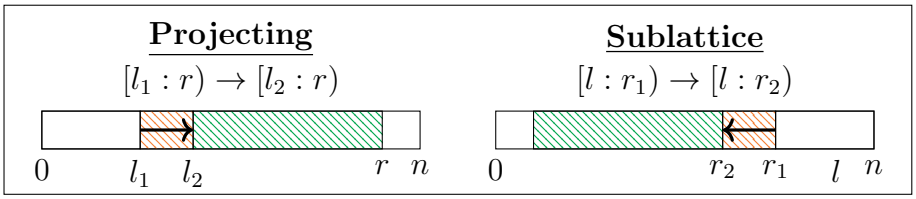


Figure 2.4: Moving to a smaller context by projecting (left), or by going to a sublattice (right).

Projections can be viewed as a way to move from a context $[l_1 : r)$, to a smaller context $[l_2 : r)$ for $l_1 < l_2 \leq r$, while taking a sublattice can be viewed as a way to move from a context $[l, r_1)$, to a smaller context $[l : r_2)$ for $l \leq r_2 < r_1$. See Figure 2.4 for an illustration.

2.4.2 Lifting

So far we have considered the technique of moving from a large context to a smaller context by projecting or going to a sublattice. Now suppose we have done such a move, but we want to revert it, i.e., we want to move back to a larger context. Moving a lattice vector from $\mathcal{L}'$ to a superlattice $\mathcal{L} \supset \mathcal{L}'$ is trivial, since for any vector $\mathbf{v} \in \mathcal{L}'$ we have by definition that $\mathbf{v} \in \mathcal{L}$.

For projections this is not so simple. To move from a context $[l_2 : r)$, back to $[l_1 : r)$ for $l_1 < l_2 \leq r$, we have to undo the projection $\pi_{(\tilde{\mathbf{b}}_{l_1}, \ldots, \tilde{\mathbf{b}}_{l_2-1})^\perp}$ away from $\text{span}(\mathcal{L}_{[l_1:l_2)})^\perp$. Concretely, given a vector $\mathbf{w}' \in \mathcal{L}_{[l_2:r)}$ we have to find a preimage $\mathbf{w} \in \mathcal{L}_{[l_1:r)}$, such that $\pi_{(\tilde{\mathbf{b}}_{l_1}, \ldots, \tilde{\mathbf{b}}_{l_2-1})^\perp}(\mathbf{w}) = \mathbf{w}'$. We call this process *lifting*, and $\mathbf{w}$ a *lift* of $\mathbf{w}'$ from the context $[l_2 : r)$ to $[l_1 : r)$.

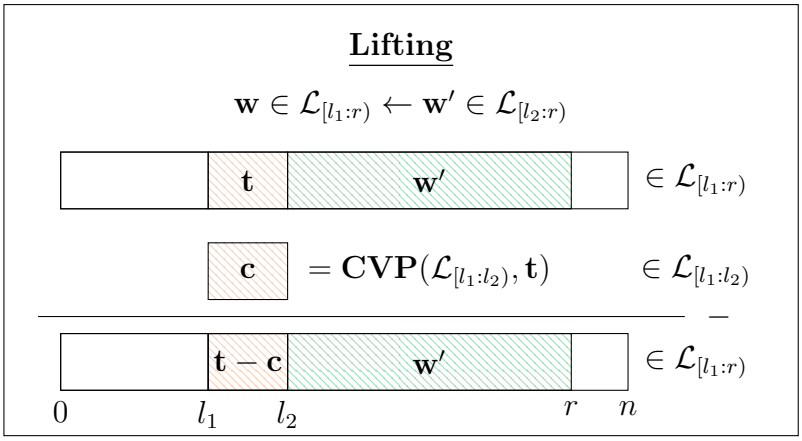


Figure 2.5: Undoing the projection from the context $[l_2 : r)$ to $[l_1 : r)$ for $l_1 < l_2 \leq r$ gives some target $\mathbf{t}$ in the context $[l_2 : r)$. To obtain a small lift we need to find a close vector $\mathbf{c} \in \mathcal{L}_{[l_1:l_2)}$ to $\mathbf{t}$ and subtract it.

Computing some lift is not hard, since $\mathbf{B}_{[l_2:r)}$ is a basis of $\mathcal{L}_{[l_2:r)}$ we can write

$$\mathbf{w}' = \sum_{i=l_2}^{r-1} x_i \cdot \pi_{l_2}(\mathbf{b}_i),$$

with $x_i \in \mathbb{Z}$. A lift of $\mathbf{w}'$ is then given by $\mathbf{w} = \sum_{i=l_2}^{r-1} x_i \cdot \pi_{l_1}(\mathbf{b}_i) \in \mathcal{L}_{[l_1:r)}$. However, such a lift is not unique, and the default lift above can be very long. Preferably, we want some control over the length of the lift. Computing the shortest lift reduces to CVP in the lattice $\mathcal{L}_{[l_1:l_2)}$, see Figure 2.5.

Lemma 30. *Let $\mathcal{L} \subset \mathbb{R}^d$ be a lattice, with basis $[\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$. For $l_1 < l_2 \leq r$, consider the vector $\mathbf{w}' \in \mathcal{L}_{[l_2:r)}$. Given any lift $\mathbf{w} = \mathbf{t} + \mathbf{w}' \in \mathcal{L}_{[l_1:r)}$ of $\mathbf{w}'$, satisfying $\pi_{l_2}(\mathbf{w}) = \mathbf{w}'$, and $\mathbf{t} \in \operatorname{span} \mathcal{L}_{[l_1:l_2)}$, a shortest lift of $\mathbf{w}'$ is given by*

$$(\mathbf{t} - \mathbf{c}) + \mathbf{w}',$$

where $\mathbf{c} = \mathbf{CVP}(\mathcal{L}_{[l_1:l_2)}, \mathbf{t})$ is a closest vector to $\mathbf{t}$.

Proof. Given the lift $\mathbf{w}$ of $\mathbf{w}'$, the full set of lifts is given by $\mathbf{w} - \mathbf{c}$ for $\mathbf{c} \in \mathcal{L}_{[l_1:l_2)} \subset \mathcal{L}_{[l_1:r)}$, since we require that $\pi_{l_2}(\mathbf{c}) = 0$. For any such

$\mathbf{c} \in \mathcal{L}_{[l_1:l_2)}$, the squared norm of the lift $\mathbf{w} - \mathbf{c}$ is given by

$$\|\mathbf{w} - \mathbf{c}\|^2 = \|\mathbf{t} - \mathbf{c} + \mathbf{w}'\|^2 = \|\mathbf{t} - \mathbf{c}\|^2 + \|\mathbf{w}'\|^2.$$

To minimize the above we have to minimize $\|\mathbf{t}-\mathbf{c}\|$ for $\mathbf{c} \in \mathcal{L}_{[l_1:l_2)}$. This is exactly a CVP instance in the lattice $\mathcal{L}_{[l_1:l_2)}$ with target $\mathbf{t}$. A closest vector $\mathbf{c} \in \mathcal{L}_{[l_1:l_2)}$ to $\mathbf{t}$ will thus give a shortest lift $(\mathbf{t} - \mathbf{c}) + \mathbf{w}'$. □

In general solving CVP instances is hard, and thus we cannot always obtain an optimal lift. In dimension 1, however, CVP is as simple as rounding.

Corollary 31. *Let $\mathcal{L} \subset \mathbb{R}^d$ be a lattice, with basis $[\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$, and let $\mathbf{w}' = \sum_{i=l+1}^{r-1} x_i \pi_{l+1}(\mathbf{b}_i) \in \mathcal{L}_{[l+1:r)}$, with $x_i \in \mathbb{Z}$, be a projected lattice vector. A shortest lift $\mathbf{w}$ of $\mathbf{w}'$ from $[l+1:r)$ to $[l:r)$ is given by*

$$\mathbf{w} = \left(\sum_{i=l+1}^{r-1} x_i \pi_l(\mathbf{b}_i)\right) - \left\lfloor \sum_{i=l+1}^{r-1} x_i \cdot \frac{\langle \tilde{\mathbf{b}}_l, \mathbf{b}_i \rangle}{\langle \tilde{\mathbf{b}}_l, \tilde{\mathbf{b}}_l \rangle} \right\rceil \cdot \tilde{\mathbf{b}}_l.$$

In particular the squared norm of $\mathbf{w}'$ to $\mathbf{w}$ increases by at most $\frac{1}{4}\|\tilde{\mathbf{b}}_l\|^2$.

Proof. One lift of $\mathbf{w}'$ is $\mathbf{w} = \sum_{i=l+1}^{r-1} x_i \pi_l(\mathbf{b}_i) \in \mathcal{L}_{[l:r)}$. We obtain the orthogonal sum $\mathbf{w} = \mathbf{t} + \mathbf{w}' \in \mathcal{L}_{[l:r)}$ for $\mathbf{t} = \left(\sum_{i=l+1}^{r-1} x_i \cdot \frac{\langle \tilde{\mathbf{b}}_l, \mathbf{b}_i \rangle}{\langle \tilde{\mathbf{b}}_l, \tilde{\mathbf{b}}_l \rangle}\right) \cdot \tilde{\mathbf{b}}_l$, where we also use that $\langle \tilde{\mathbf{b}}_l, \pi_l(\mathbf{b}_i) \rangle = \langle \tilde{\mathbf{b}}_l, \mathbf{b}_i \rangle$. By Lemma 30 a shortest lift $(\mathbf{t} - \mathbf{c}) + \mathbf{w}'$ now follows from computing a closest vector $\mathbf{c} \in \mathcal{L}_{[l:l+1)} = \tilde{\mathbf{b}}_l \cdot \mathbb{Z}$ to $\mathbf{t}$. For $\mathbf{t} = \lambda \cdot \tilde{\mathbf{b}}_l$ with $\lambda \in \mathbb{R}$, a closest vector is simply given by $\mathbf{c} = \lfloor \lambda \rceil \cdot \tilde{\mathbf{b}}_l$. □

2.4.3 Babai's Nearest Plane Algorithm

Computing the optimal shortest lift reduces to a CVP instance. So optimally lifting a lattice vector from a context of $[l:r)$ to $[0:r)$, is hard for large l. However, by Corollary 31, lifting from $[l:r)$ to $[l-1:r)$ is as simple as rounding, and the squared norm increases by at most $\frac{1}{4}\|\tilde{\mathbf{b}}_{l-1}\|^2$. If we repeat this l times, we arrive at $[0:r)$, with an increase of at most $\frac{1}{4}\sum_{i=0}^{l-1}\|\tilde{\mathbf{b}}_i\|^2$.

Let us take this idea and build a general approximate CVP algorithm from it. First we extract an important property from Corollary 31.

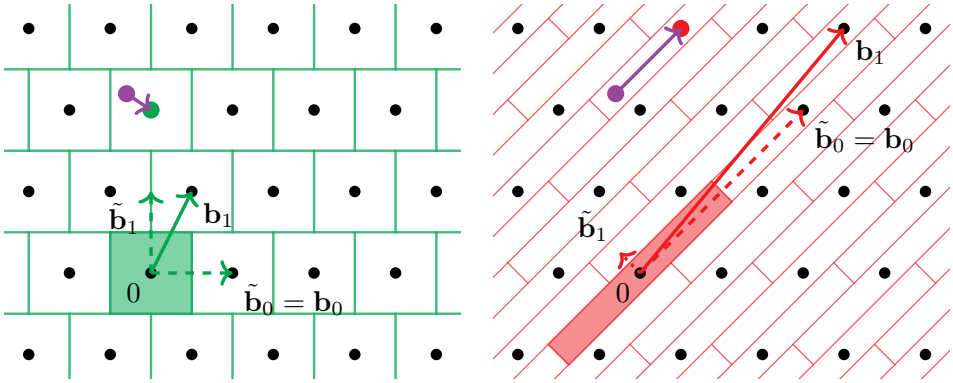


Figure 2.6: Babai's fundamental domain and Babai's nearest plane algorithm for two different bases of the same lattice.

Definition 32 (Size-reduction). *We call a vector* $\mathbf{t} \in \mathbb{R}^d$ *size-reduced w.r.t. a vector* $\mathbf{v} \in \mathbb{R}^d$ *if* $\langle \mathbf{t}, \mathbf{v} \rangle \in [-\frac{1}{2}, \frac{1}{2}) \cdot \|\mathbf{v}\|^2$. *We call a vector* $\mathbf{t} \in \mathbb{R}^d$ *size-reduced w.r.t. a basis* $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ *if* $\mathbf{t}$ *is size-reduced w.r.t.* $\tilde{\mathbf{b}}_i$ *for all* $0 \leq i < n$.

Rephrased, any target $\mathbf{t} \in \operatorname{span} \mathcal{L}(\mathbf{B})$ that is size-reduced w.r.t. the basis $\mathbf{B}$ lies in Babai's fundamental domain.

Definition 33 (Babai's fundamental domain). *For a basis* $\mathbf{B}$ *with GSO* $\tilde{\mathbf{B}} = [\tilde{\mathbf{b}}_0, \ldots, \tilde{\mathbf{b}}_{n-1}]$, *we define Babai's fundamental domain as*

$$\mathcal{P}(\tilde{\mathbf{B}}) := \tilde{\mathbf{B}} \cdot [-\tfrac{1}{2}, \tfrac{1}{2})^n = \left\{ \sum_{i=0}^{n-1} \lambda_i \tilde{\mathbf{b}}_i : \lambda_i \in [-\tfrac{1}{2}, \tfrac{1}{2}) \text{ for all } 0 \leq i < n \right\}.$$

Babai's nearest plane algorithm (see Algorithm 1) computes for any target $\mathbf{t} \in \mathcal{P}(\tilde{\mathbf{B}})$ a coset representative $\mathbf{e} \in \mathbf{t} + \mathcal{L}(\mathbf{B})$ that is size-reduced w.r.t. the basis, i.e. such that $\mathbf{e} \in \mathcal{P}(\tilde{\mathbf{B}})$. In other words it computes a close vector $\mathbf{c} = \mathbf{t} - \mathbf{e} \in \mathcal{L}(\mathbf{B})$ to $\mathbf{t}$. It does so by size-reducing one step at a time, from $i = n - 1$ to $i = 0$.

Lemma 34. *For any lattice* $\mathcal{L}$ *with basis* $\mathbf{B}$, *Babai's fundamental domain* $\mathcal{P}(\tilde{\mathbf{B}})$ *is a fundamental domain of the lattice. Let* $\mathbf{t} \in \operatorname{span}(\mathcal{L})$ *be any target and* $(\mathbf{c}, \mathbf{e}) \leftarrow \mathbf{Babai}(\mathbf{B}, \mathbf{t})$ *the output of Algorithm 1. Then* $\mathbf{e} := \mathbf{t} - \mathbf{c} \in \mathcal{P}(\tilde{\mathbf{B}})$, *i.e. Babai's nearest plane algorithms computes the unique close vector with error in* $\mathcal{P}(\tilde{\mathbf{B}})$.

Proof. After iteration i in Algorithm 1, we have by construction that $\langle \mathbf{e}, \tilde{\mathbf{b}}_i \rangle \in [-\frac{1}{2}, \frac{1}{2}) \cdot \|\tilde{\mathbf{b}}_i\|^2$. In later iterations $j < i$ we only add integer multiples of $\mathbf{b}_j$ to $\mathbf{e}$, and because $\mathbf{b}_j \perp \tilde{\mathbf{b}}_i$ the above inner product stays unchanged. So after all iterations of Algorithm 1 we have $\mathbf{e} \in \mathcal{P}(\tilde{\mathbf{B}})$. Because at the start $\mathbf{e} = \mathbf{t}$, and we only added lattice vectors to $\mathbf{e}$ we have that $\mathbf{t} - \mathbf{e} = \mathbf{c} \in \mathcal{L}$. So Algorithm 1 is correct.

From this correctness it follows that $\mathcal{P}(\tilde{\mathbf{B}})$ represents each coset $\mathbf{t} + \mathcal{L}$ for $\mathbf{t} \in \text{span}(\mathcal{L})$ at least once. For uniqueness let $\mathbf{e}, \mathbf{e}' \in \mathcal{P}(\tilde{\mathbf{B}})$ with $\mathbf{e} + \mathcal{L} = \mathbf{e}' + \mathcal{L}$. Then their difference $\mathbf{v} = \mathbf{e} - \mathbf{e}' \in \mathcal{L}$ is a lattice vector. We can write

$$\mathbf{v} = \sum_{i=0}^{n-1} x_i \mathbf{b}_i,$$

with $x_i \in \mathbb{Z}$. Suppose that $\mathbf{v} \neq 0$, and let $j = \arg\max_i \{x_i \neq 0\}$. Since $\mathbf{e}, \mathbf{e}' \in \mathcal{P}(\tilde{\mathbf{B}})$, and $|x_j| \geq 1$, we get the following contradiction

$$\begin{aligned} \|\tilde{\mathbf{b}}_j\|^2 &> |\langle \mathbf{e}, \tilde{\mathbf{b}}_j \rangle - \langle \mathbf{e}', \tilde{\mathbf{b}}_j \rangle| = |\langle \mathbf{v}, \tilde{\mathbf{b}}_j \rangle| = |\langle \sum_{i=0}^{j} x_i \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle| \\ &= |x_j \cdot \langle \mathbf{b}_j, \tilde{\mathbf{b}}_j \rangle| \geq \|\tilde{\mathbf{b}}_j\|^2. \end{aligned}$$

So $\mathbf{v} = 0$, and $\mathbf{e} = \mathbf{e}'$. □

Algorithm 1: Babai's nearest plane algorithm $\mathbf{Babai}(\mathbf{B}, \mathbf{t})$

Input : A basis $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ of a lattice $\mathcal{L}$ with Gram-Schmidt orthogonalization $\tilde{\mathbf{B}} = [\tilde{\mathbf{b}}_0, \ldots, \tilde{\mathbf{b}}_{n-1}]$, and a target $\mathbf{t} \in \text{span}(\mathcal{L})$.

Output: $(\mathbf{c}, \mathbf{e})$ such that $\mathbf{c} + \mathbf{e} = \mathbf{t}$, with $\mathbf{c} \in \mathcal{L}$ and $\mathbf{e} \in \mathcal{P}(\tilde{\mathbf{B}})$.

$\mathbf{e} := \mathbf{t}$, $\mathbf{c} := \mathbf{0}$
for $i = n - 1$ *down to* 0 **do**
  $k := \left\lceil \frac{\langle \mathbf{e}, \tilde{\mathbf{b}}_i \rangle}{\|\tilde{\mathbf{b}}_i\|^2} \right\rfloor$
  $\mathbf{e} := \mathbf{e} - k\mathbf{b}_i$
  $\mathbf{c} := \mathbf{c} + k\mathbf{b}_i$
end
return $(\mathbf{c}, \mathbf{e})$

The shape of Babai's fundamental domain, and thus the usefulness of Babai's nearest plane algorithm, depends on the norms of the GSO vectors.

Corollary 35. *For any lattice $\mathcal{L}$ with basis $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$, and target $\mathbf{t} \in \mathrm{span}(\mathcal{L})$, let $(\mathbf{c}, \mathbf{e}) \leftarrow \mathbf{Babai}(\mathcal{L}, \mathbf{t})$. Then*

$$\|\mathbf{t} - \mathbf{c}\|^2 \leq \frac{1}{4}\sum_{i=0}^{n-1}\|\tilde{\mathbf{b}}_i\|^2 \leq \frac{1}{4}\sum_{i=0}^{n-1}\|\mathbf{b}_i\|^2.$$

Furthermore, if $\mathrm{dist}(\mathcal{L}, \mathbf{t}) < \frac{1}{2}\min_i\|\tilde{\mathbf{b}}_i\|$, then $\mathbf{c}$ is the unique closest lattice vector to $\mathbf{t}$.

For a uniformly random target $\mathbf{t} \in \mathbb{T}(\mathcal{L})$ Babai's nearest plane algorithm recovers a uniform error in $\mathcal{P}(\tilde{\mathbf{B}})$. The expected squared distance is thus $\frac{1}{12}\sum_{i=0}^{n-1}\|\tilde{\mathbf{b}}_i\|^2$, a factor 3 better than the worst-case.

Since $\mathcal{P}(\tilde{\mathbf{B}})$ is a fundamental domain of $\mathcal{L}$, it has volume $\mathrm{vol}(\mathcal{L})$. As a result we obtain the following invariant.

Corollary 36. *For a rank n lattice with basis $\mathbf{B}$ and GSO $\tilde{\mathbf{B}}$, we have*

$$\mathrm{vol}(\mathcal{P}(\tilde{\mathbf{B}})) = \prod_{i=0}^{n-1}\|\tilde{\mathbf{b}}_i\| = \mathrm{vol}(\mathcal{L}).$$

More generally we have that $\mathrm{vol}(\mathcal{L}_{[l:r)}) = \prod_{i=l}^{r-1}\|\tilde{\mathbf{b}}_i\|$. Due to the invariant $\prod_{i=0}^{n-1}\|\tilde{\mathbf{b}}_i\| = \mathrm{vol}(\mathcal{L})$, the relevant quantities $\sum_{i=0}^{n-1}\|\tilde{\mathbf{b}}_i\|^2$ and $\min_i\|\tilde{\mathbf{b}}_i\|$, for the performance of Babai's nearest plane algorithm, are optimal when $\|\tilde{\mathbf{b}}_0\| = \ldots = \|\tilde{\mathbf{b}}_{n-1}\| = \mathrm{vol}(\mathcal{L})^{1/n}$, i.e. when the GSO norms are perfectly balanced. In Chapter 6 we will see that balancing these GSO norms is the main objective of *basis reduction* algorithms.

It is more than natural to size-reduce a basis $\mathbf{B}$ w.r.t. itself.

Definition 37 (Size-reduced basis). *A basis $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ is called size-reduced if $\mathbf{b}_j$ is size-reduced with respect to the preceding basis vectors $[\mathbf{b}_0, \ldots, \mathbf{b}_{j-1}]$ for all $1 \leq j < n$. In particular, we have*

$$|\langle\tilde{\mathbf{b}}_i, \mathbf{b}_j\rangle| \leq \frac{1}{2}\|\tilde{\mathbf{b}}_i\|^2 \text{ for all } 0 \leq i < j < n.$$

To size-reduce a basis we can simply apply Babai's nearest plane algorithm several times (in the order $\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}$). Size-reduction relates the basis norms $(\|\mathbf{b}_i\|)_i$ to the GSO norms $(\|\mathbf{b}_i\|)_i$.

Lemma 38 (Relations). *For a size-reduced basis* $\mathbf{B}$ *we have*

$$\|\tilde{\mathbf{b}}_j\|^2 \leq \|\mathbf{b}_j\|^2 \leq \|\tilde{\mathbf{b}}_j\|^2 + \frac{1}{4}\sum_{i<j}\|\tilde{\mathbf{b}}_i\|^2.$$

2.4.4 Dual basis

Recall that for a lattice $\mathcal{L}$ and its dual $\mathcal{L}^*$ we have the volumetric relation $\mathrm{vol}(\mathcal{L}^*) = 1/\mathrm{vol}(\mathcal{L})$. More generally, there also exists a relation between the GSO norms of a basis and the GSO norms of the corresponding dual basis, after some changes to the ordering. Let us consider a basis $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ with dual basis $\mathbf{D} = [\mathbf{d}_0, \ldots, \mathbf{d}_{n-1}]$. Note that $\mathbf{d}_{n-1}$ is by definition orthogonal to $\mathbf{b}_0, \ldots, \mathbf{b}_{n-2}$, and thus $\mathbf{d}_{n-1}$ lies in the span of $\tilde{\mathbf{b}}_{n-1}$. Furthermore we know that $\langle \mathbf{d}_{n-1}, \mathbf{b}_{n-1}\rangle = 1$, which gives us the following relation

$$\|\mathbf{d}_{n-1}\| \cdot \|\tilde{\mathbf{b}}_{n-1}\| = \langle \mathbf{d}_{n-1}, \tilde{\mathbf{b}}_{n-1}\rangle = \langle \mathbf{d}_{n-1}, \mathbf{b}_{n-1}\rangle = 1.$$

Increasing the last GSO norm $\|\tilde{\mathbf{b}}_{n-1}\|$ thus corresponds to decreasing the length of the last dual basis vector $\mathbf{d}_{n-1}$. If we apply the GSO process in the reversed order, then we obtain relations like this for every GSO norm.

Lemma 39. *Let* $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ *be a basis with dual basis* $\mathbf{D} = [\mathbf{d}_0, \ldots, \mathbf{d}_{n-1}]$. *Let* $\mathbf{D}_{-1} = [\mathbf{d}_{n-1}, \ldots, \mathbf{d}_0]$ *be the reversed dual basis with GSO* $[\tilde{\mathbf{d}}_{n-1}, \ldots, \tilde{\mathbf{d}}_0]$, *then* $[\tilde{\mathbf{d}}_i]$ *is a (reversed) dual basis of* $[\tilde{\mathbf{b}}_i]$, *i.e.,*

$$\mathrm{span}(\tilde{\mathbf{d}}_i) = \mathrm{span}(\tilde{\mathbf{b}}_i), \text{ and } \|\tilde{\mathbf{d}}_i\| \cdot \|\tilde{\mathbf{b}}_i\| = 1.$$

In the rest of this thesis, we will refer to $(\|\tilde{\mathbf{d}}_{n-1}\|, \ldots, \|\tilde{\mathbf{d}}_0\|) = (\|\tilde{\mathbf{b}}_{n-1}\|^{-1}, \ldots, \|\tilde{\mathbf{b}}_0\|^{-1})$ as the *profile of the dual basis*, i.e., we will implicitly refer to the GSO norms of the *reversed* dual basis.

2.5 Volumes & distributions

2.5.1 Volumes

We will discuss some higher dimensional geometric objects and their volumes, which play an important role in the analysis of lattice algorithms.

Definition 40 (Sphere and Ball). *For $n \geq 1$ we define the $(n-1)$-sphere $\mathcal{S}_r^{n-1} \subset \mathbb{R}^n$ and the n-ball $\mathcal{B}_r^n \subset \mathbb{R}^n$ of radius $r > 0$ by*

$$\mathcal{S}_r^{n-1} := \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| = r\}, \text{ and } \mathcal{B}_r^n := \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| \leq r\},$$

and we denote $\mathcal{S}^{n-1}$ and $\mathcal{B}^n$ for the sphere and ball of radius 1.

Their volumes are given by

$$\begin{aligned} \mathrm{vol}_n(\mathcal{B}_r^n) &= r^n \cdot V_n, \\ \mathrm{vol}_{n-1}(\mathcal{S}_r^{n-1}) &= r^{n-1} \cdot V_n \cdot n, \end{aligned}$$

where $V_n := \mathrm{vol}_n(\mathcal{B}^n) = \pi^{n/2}/\Gamma(n/2+1)$, with Γ the gamma function.

Definition 41 (Half-space). *For any $\mathbf{v} \in \mathbb{R}^n$ and $a \in \mathbb{R}$ we define the half-space $\mathcal{H}_{\mathbf{v},a}$ by*

$$\mathcal{H}_{\mathbf{v},a} := \{\mathbf{x} \in \mathbb{R}^n : \langle \mathbf{v}, \mathbf{x} \rangle \geq a\}.$$

The cap in direction $\mathbf{v}$ of a sphere consists of all spherical vectors that are somewhat close to $\mathbf{v}$, or equivalently that have a small angle with $\mathbf{v}$.

Definition 42 (Cap). *For any $\mathbf{v} \in \mathcal{S}^{n-1}$ and $a \in [0,1]$ we define the (spherical) cap $\mathcal{C}_{\mathbf{v},a}^{n-1}$ by the intersection*

$$\mathcal{C}_{\mathbf{v},a}^{n-1} := \mathcal{S}^{n-1} \cap \mathcal{H}_{\mathbf{v},a},$$

and we denote its relative $(n-1)$-dimensional volume, which is independent of $\mathbf{v}$, by $\mathcal{C}^{n-1}(a) := \mathrm{vol}_{n-1}(\mathcal{C}_{\mathbf{v},a}^{n-1}) / \mathrm{vol}_{n-1}(\mathcal{S}^{n-1})$.

The intersection of two caps is called a wedge.

Definition 43 (Wedge). *For any $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{n-1}$, and $a \in \mathbb{R}^n$, we define the (spherical) wedge $\mathcal{W}^{n-1}_{\mathbf{v},\mathbf{w},a}$ by the intersection*

$$\mathcal{W}^{n-1}_{\mathbf{v},\mathbf{w},a} := \mathcal{C}^{n-1}_{\mathbf{v},a} \cap \mathcal{C}^{n-1}_{\mathbf{w},a},$$

and for $c \in [-1, 1]$ and any $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{n-1}$ such that $\langle \mathbf{v}, \mathbf{w} \rangle = c$, we denote its relative $(n-1)$-dimensional volume by $\mathcal{W}^{n-1}(a, c) := \mathrm{vol}_{n-1}(\mathcal{W}^{n-1}_{\mathbf{v},\mathbf{w},a}) / \mathrm{vol}_{n-1}(\mathcal{S}^{n-1})$.

In Chapter 3 we will see that these volumes play an important role in the complexity analysis of lattice sieving algorithms. To compute the exact volume of caps and wedges we rely on the (regularized) incomplete beta function.

Definition 44. *For constants $a, b \in \mathbb{R}^n$ we define the incomplete beta function $B(x; a, b)$ by*

$$B(x; a, b) = \int_0^x t^{a-1}(1-t)^{b-1} dt,$$

and the regularized incomplete beta function by

$$I_x(a, b) := \frac{B(x; a, b)}{B(a, b)}.$$

The beta distribution $Beta(a, b)$ with $a, b > 0$, is the distribution with support $[0, 1]$ and CDF $x \mapsto I_x(a, b)$.

The regularized incomplete beta function allows us to give an explicit formula for the relative cap volume.

Lemma 45. *For any $a \in [0, 1]$, the relative volume $\mathcal{C}^{n-1}(a)$ of a spherical cap equals*

$$\mathcal{C}^{n-1}(a) = \frac{1}{2} I_{a^2}\left(\frac{1}{2}, \frac{n-1}{2}\right).$$

From this we can also compute wedge volumes by explicit integration. Computing the exact volumes will be useful when deriving concrete complexity estimates, but for the asymptotic complexity results we have simpler formulas.

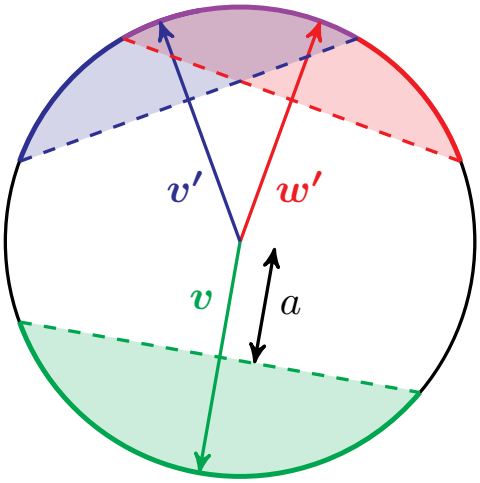


Figure 2.7: Illustration of a spherical cap $\mathcal{C}_{\mathbf{v},a}$ (in green) and a spherical wedge $\mathcal{C}_{\mathbf{v},\mathbf{w},a'} = \mathcal{C}_{\mathbf{v},a'} \cap \mathcal{C}_{\mathbf{w},a'}$ (in purple). The spherical variant consists of just the border, while the ball variant covers the filled in areas.

Lemma 46 (Volumes [MV10; BDGL16]). *For constants $a \in [0,1]$, $c \in [2a^2 - 1, 1]$, and growing n, we have*

$$\mathcal{C}^{n-1}(a) = (1-a^2)^{n/2} \cdot n^{O(1)}, \text{ and}$$
$$\mathcal{W}^{n-1}(a,c) = \left(1 - \frac{2a^2}{1+c}\right)^{n/2} \cdot n^{O(1)}.$$

We see that for fixed constants a, c the volumes scale single exponentially as $2^{Cn+o(n)}$ for some constant $C = C(a,c)$.

Caps and wedges can similarly be defined for balls, but as for large dimensions n most of the volume is located close to the edge, they are asymptotically similar (in particular Lemma 46 also applies to the ball variant).

2.5.2 Distributions

Now that we have introduced caps we can have a better understanding of the properties of uniform samples from a sphere or ball. To better understand the spherical and ball distributions we show how to

sample from them using nothing more than a (continuous) Gaussian distribution.

Definition 47. *We define a (continuous) Gaussian distribution χ_{μ,σ^2} with mean $\mu \in \mathbb{R}$, and standard deviation $\sigma > 0$ on $\mathbb{R}$ by the probability density function*

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Unless otherwise stated $\mu = 0$, and we simply denote $\chi_{\sigma^2} := \chi_{0,\sigma^2}$.

As implied we have mean $\mathbb{E}[\chi_{\mu,\sigma^2}] = \mu$ and variation $\mathbb{V}[\chi_{\mu,\sigma^2}] = \sigma^2$. We can now directly construct a uniform sample over the sphere $\mathcal{S}^{n-1}$ by sampling n Gaussian coefficients and renormalizing the vector.

Lemma 48. *Let $X_1, \dots, X_n \sim \chi_{\sigma^2}$ for any $\sigma > 0$ be independent Gaussian variables, then*

$$\frac{(X_1, \dots, X_n)}{\sqrt{X_1^2 + \dots + X_n^2}} \sim \mathcal{U}(\mathcal{S}^{n-1})$$

Note that for $\sigma = 1$ and large n the denominator $\sqrt{X_1^2 + \dots + X_n^2}$ is highly concentrated around $\sqrt{n}$, which implies that each individual spherical coordinate is distributed close to Gaussian with standard deviation $1/\sqrt{n}$. Using the cap volume formulas we can give an exact description of the coefficient distribution.

Lemma 49. *Let $(u_1, \dots, u_n) \sim \mathcal{U}(\mathcal{S}^{n-1})$, then $u_i^2 \sim Beta(\frac{1}{2}, \frac{n-1}{2})$ on $[0, 1]$ which has CDF $x \mapsto 2\mathcal{C}^{n-1}(\sqrt{x}) = I_x(\frac{1}{2}, \frac{n-1}{2})$.*

The uniform distribution over a ball is similar to that of a sphere, but in addition to the direction we now also have to sample a length.

Lemma 50 (Ball-sphere relation). *Let U be uniform over $[0, 1]$, and S uniform over $\mathcal{S}^{n-1}$, then*

$$U^{1/n} \cdot S \sim \mathcal{U}(\mathcal{B}^n).$$

This also directly implies that the expectation of $\|\mathbf{x}\|^2$ for $\mathbf{x} \sim \mathcal{U}(\mathcal{B}^n)$ is given by $\mathbb{E}[U^{1/n}] = \frac{n}{n+1}$, i.e., for growing n a sample from the unit ball $\mathcal{B}^n$ comes arbitrarily close to the boundary in expectation. Indeed up to a small difference in dimension the coefficients behave exactly the same as a spherical sample due to the following Lemma.

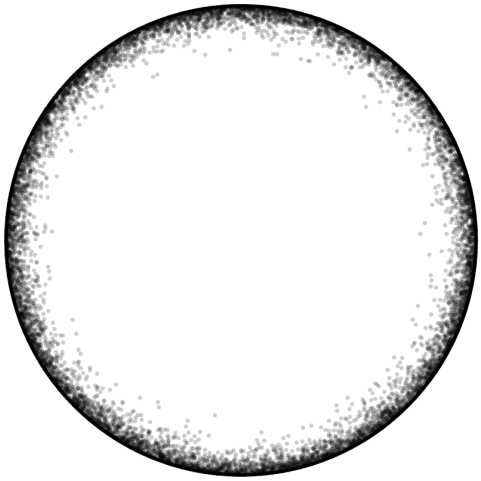


Figure 2.8: Visualisation of the norm of 10000 vectors sampled uniformly over $\mathcal{B}^{30}$, represented by vectors in $\mathcal{B}^2$ with the same norm and a uniform direction. Almost all volume of a high-dimensional ball lies near its border.

Lemma 51. *If $(u_1, \ldots, u_{n+2}) \sim \mathcal{U}(\mathcal{S}^{n+1})$ is spherically distributed, then $(u_1, \ldots, u_n) \sim \mathcal{U}(\mathcal{B}^n)$ is uniform over the ball.*

Corollary 52. *Let $(u_1, \ldots, u_n) \sim \mathcal{U}(\mathcal{B}^n)$, then $u_i^2 \sim Beta(\frac{1}{2}, \frac{n+1}{2})$ with CDF $x \mapsto I_x(\frac{1}{2}, \frac{n+1}{2})$.*

We write the chi-square distribution with k degrees of freedom as $\chi^2_{k,\sigma^2} := \sum_{i=1}^k X_i^2$, where $X_1, \ldots, X_k$ are independently distributed as χ_{σ^2}. The chi-square distribution has expectation $k\sigma^2$, and the following log-expectation.

Lemma 53. *Let X be distributed as χ^2_{k,σ^2}, then*

$$\mathbb{E}\left[\log\left(X\right)\right] = \log(2\sigma^2) + \psi(k/2),$$

where $\psi(x) := \Gamma'(x)/\Gamma(x)$ is the digamma function.

We will also define the discrete Gaussian distribution over $\mathbb{Z}$.

Definition 54. *We define the Gaussian function on $\mathbb{Z}$ with parameter $s > 0$ as $\rho_s(x) = \exp(-\pi x^2/s^2)$. Then the discrete (centered)*

Gaussian distribution $\mathcal{D}_{\mathbb{Z},\sigma^2}$ on $\mathbb{Z}$ with parameter s is defined by

$$\Pr_{X\sim\mathcal{D}_{\mathbb{Z},\sigma^2}}[X=x]=\frac{\rho_s(x)}{\rho_s(\mathbb{Z})}=\frac{\rho_s(x)}{\sum_{y\in\mathbb{Z}}\rho_s(y)}.$$

For large enough s, say $s \geq 3$, the variance of $\mathcal{D}_{\mathbb{Z},\sigma^2}$ is very close to $s^2/(2\pi)$. The discrete Gaussian distribution can easily be generalized to any rank n lattice, but we will do this in Chapter 10 in the quadratic form setting.

2.6 Heuristics

For cryptanalysis it does not matter if your algorithm cannot solve a particular worst-case instance, as long as it can solve the instance that results in breaking the cryptosystem. Problem instances from most cryptosystems, and in particular those in lattice-based cryptography, rely heavily on randomness to hide secret information.

Often these systems are not perfectly random, but have some hidden secret structure. However, before finding this secret structure (and possibly breaking the scheme), these instances 'behave' like random instances. For cryptanalysis, we are thus interested in the behaviour of our algorithms on those random average-case instances.

Analysing how algorithms behave on average-case instances is often hard. In particular, such an analysis may involve many complex distributions with many dependencies between them. As a result there is a big gap between provable and practical algorithms for lattice problems. Heuristics allow to bridge this gap, and as such, they play an important role in this thesis. For example the best provable SVP algorithm runs in time $2^{n+o(n)}$, while the best practical algorithm runs heuristically in time $2^{0.292n+o(n)}$. We can heuristically assume that some distributions are simpler (and less dependent) then they actually are, and do our analysis with those. Heuristics are not necessarily true in a mathematical sense, it is often easy to give counterexamples. However, if they give estimates that match the practical behaviour of algorithms, then they are still very useful.

One should not blindly follow heuristics. Heuristics should describe practical behaviour, and thus it is important to validate them experimentally. Experiments of feasible size can validate heuristics,

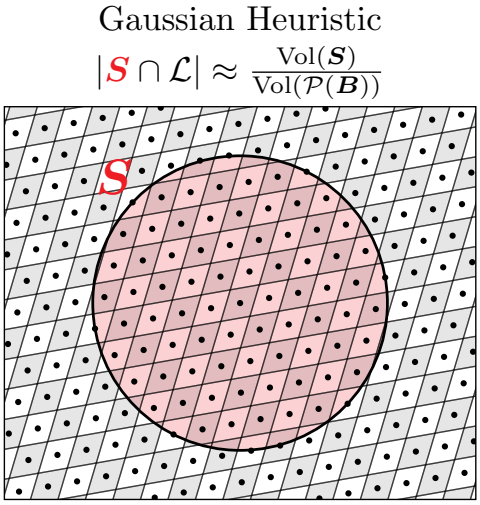


Figure 2.9: Gaussian Heuristic

and those heuristics can then be used to analyse how the algorithm would behave on instances of cryptographic size.

2.6.1 Gaussian Heuristic

The most commonly used, and heavily verified heuristic for lattice algorithms is the so-called Gaussian Heuristic. Heuristics are often inspired by provable average-case statements, and for the Gaussian Heuristic one could give the following origin.

Lemma 55. *For a rank n lattice $\mathcal{L}$, a measurable set $S \subset \mathrm{span}(\mathcal{L})$ and a uniform target $\mathbf{t} \in \mathbb{T}(\mathcal{L}) = \mathrm{span}(\mathcal{L})/\mathcal{L}$ we have*

$$\mathbb{E}_{\mathbf{t} \sim \mathcal{U}(\mathbb{T}(\mathcal{L}))} |(\mathbf{t} + \mathcal{L}) \cap S| = \frac{\mathrm{vol}_n(S)}{\mathrm{vol}(\mathcal{L})}.$$

Lemma 55 essentially says that for a volume S, the number of lattice vectors in $\mathcal{L} \cap S$ is $\mathrm{vol}(S)/\mathrm{vol}(\mathcal{L})$ in expectation if we translate S around. Here $1/\mathrm{vol}(\mathcal{L})$ should be interpreted as the density of the lattice inside its span, e.g. we expect about 1 lattice vector per volume of size $\mathrm{vol}(\mathcal{L})$. The Gaussian Heuristic builds from this interpretation.

Heuristic 56 (Gaussian Heuristic). *For a rank n lattice $\mathcal{L}$ and a measurable set $S \subset \operatorname{span}(\mathcal{L})$ the Gaussian Heuristic states that*

$$|\mathcal{L} \cap S| \approx \frac{\operatorname{vol}_n(S)}{\operatorname{vol}(\mathcal{L})}.$$

Furthermore, the lattice vectors in $\mathcal{L} \cap S$ are uniformly distributed over S.

In some sense the Gaussian Heuristic completely forgets about the lattice structure, and treats the lattice $\mathcal{L}$ as some kind of uniform cloud of vectors with density $1/\operatorname{vol}(\mathcal{L})$. In a reasonable set $S \subset \operatorname{span}(\mathcal{L})$, we then naturally expect about $\operatorname{vol}(S)/\operatorname{vol}(\mathcal{L})$ lattice vectors. The set S is often chosen as a (linear transformations of a) hypercube or ball. E.g., if we apply the Gaussian Heuristic to a ball we obtain an estimate for the first minimum of a lattice.

Heuristic claim 57. *Let $\mathcal{L}$ be a rank n lattice with volume $\operatorname{vol}(\mathcal{L})$. The expectation $\operatorname{gh}(\mathcal{L})$ of the first minimum $\lambda_1(\mathcal{L})$ under the Gaussian Heuristic is given by*

$$\operatorname{gh}(\mathcal{L}) := \frac{\operatorname{vol}(\mathcal{L})^{1/n}}{\operatorname{vol}(\mathcal{B}_1^n)^{1/n}} \approx \sqrt{n/(2\pi e)} \cdot \operatorname{vol}(\mathcal{L})^{1/n}.$$

We also denote $\operatorname{gh}(n) := \operatorname{vol}(\mathcal{B}_1^n)^{-1/n} \approx \sqrt{n/(2\pi e)}$ for the expected first minimum of a rank n lattice with volume 1.

Justification. Assume without loss of generality that the lattice is of full rank $n = d$. We apply the Gaussian Heuristic to the ball $S = \mathcal{B}_\lambda^n \subset \mathbb{R}^n$ of radius $\lambda > 0$. According to the Gaussian Heuristic the number $|\mathcal{L} \cap \mathcal{B}_\lambda^n|$ of lattice vectors in the ball is about $(\lambda/\operatorname{gh}(\mathcal{L}))^n$. E.g. for $\lambda < \operatorname{gh}(\mathcal{L})$ we expect (for large n), no lattice vectors in the ball (except $\mathbf{0}$), while for $\lambda > \operatorname{gh}(\mathcal{L})$ we expect many lattice vectors in the ball. We can conclude that $\lambda_1(\mathcal{L}) \approx \operatorname{gh}(\mathcal{L})$. △

The above estimate for $\lambda_1(\mathcal{L})$ works particularly well in dimensions $n \geq 50$. The expected minimum $\operatorname{gh}(\mathcal{L})$ is precisely a factor 2 smaller than the upper bound on $\lambda_1(\mathcal{L})$ by Minkowski's Theorem 11. So Minkowski's Theorem is rather tight, but a factor 2 can still be

significant if we want to consider the concrete performance of algorithms.

Under the Gaussian Heuristic one could say that average-case exact SVP is equal to γ-HermiteSVP with $\gamma \approx \sqrt{n/(2\pi e)}$. Under the same reasoning, a random target $\mathbf{t} \in (\operatorname{span} \mathcal{L})/\mathcal{L}$ lies at distance about $\operatorname{gh}(\mathcal{L})$ from the lattice. In particular, a random target lies at distance $\delta \cdot \operatorname{gh}(\mathcal{L})$ from the lattice with exponentially small probability $\approx \delta^n$ for a constant $0 < \delta < 1$.

Sometimes the Gaussian Heuristic is a stronger heuristic than what is actually necessary. Therefore there exists many (often weaker) variants of it. For example, sometimes it is enough to only assume that the direction $\mathbf{x}/\|\mathbf{x}\|$ of some lattice vector is uniform (over the sphere).

To avoid confusion we restrict the use of "Theorem", "Lemma", and "Corollary" to formal claims, and refer to "Heuristic claims" for claims that are based on heuristics.

2.6.2 Random lattices

We finish with a short remark on what we mean by average-case instances. For lattice problems the randomness can be in several places, but in particular we discuss here the randomness of the input lattice(s). Formally the set of full rank n lattices can be identified with the quotient group $\mathcal{GL}_n(\mathbb{R})/\mathcal{GL}_n(\mathbb{Z})$. By rescaling the set of all lattices $\mathcal{L}$ with unit (co)volume $\operatorname{vol}(\mathcal{L}) = 1$, can be identified with the group $\mathcal{SL}_n(\mathbb{R})/\mathcal{SL}_n(\mathbb{Z})$. Siegel [Sie45] showed that the Haar measure on $\mathcal{SL}_n(\mathbb{R})$ induces a natural finite measure μ_n on $\mathcal{SL}_n(\mathbb{R})/\mathcal{SL}_n(\mathbb{Z})$. We can thus speak of a uniform distribution over the set of full rank n lattice of unit (co)volume.

This gives a proper definition of what a 'random' lattice is, but such lattices do not naturally appear in cryptography. For example, we often restrict to integer, q-ary or other classes of lattices. Still it can be useful to see how certain lattice properties behave over this distribution. E.g., for a lattice taken uniform w.r.t. μ_n, the first minimum can be shown to have expectation $\operatorname{gh}(\mathcal{L})$ and also to be heavily concentrated around $\operatorname{gh}(\mathcal{L})$ for increasing n. Similarly there exist expectation and concentration results for statements like Lemma 55, see [AEN19] for a survey. These essentially say: almost all lattices follow the Gaussian Heuristic.

Goldstein-Mayer introduced a way to sample integer lattices that in the limit (after rescaling) is statistically close to uniform w.r.t. μ_n. Such lattices are often used to test the average-case behaviour of algorithms. For example the TU Darmstadt SVP challenges give a Goldstein-Mayer random lattice and require you to compute a short vector of length at most $1.05 \cdot \mathrm{gh}(\mathcal{L})$.

Throughout this thesis we will be a bit more lenient with the term random lattice. We might even turn things around and informally call a lattice random if it follows the expected average-case behaviour. E.g. if it follows the Gaussian Heuristic in the general sense, or if at least $\lambda_1(\mathcal{L}) \approx \mathrm{gh}(\mathcal{L})$. To motivate this further: from a cryptanalytic perspective, the average-case is often the hardest for lattice problems. If a lattice does not behave like the average-case, then it leaks some structure or bias that an attacker could exploit. Assuming randomness therefore does not make the problem easier, but does make it easier to analyse. Furthermore, in cryptanalysis these heuristics are often applied to sufficiently randomised (projected sub)lattices, which makes their analysis tight. A common rule of thumb is thus: lattices behave like the average-case, precisely until one recovers (part of) their (secret) structure. Because the heuristics remain valid up to that moment, they can be used to determine when this will happen.

2.7 Cryptography

2.7.1 Security reductions

In the rest of this section we will present multiple cryptographic constructions, such as for encryption and signatures. These constructions come with security notions, often of the form: no probabilistic polynomial-time attacker $\mathcal{A}$ can do X (decrypt, forge a signature). Of course, there is little hope of directly proving such statements. What we can do however, is show that if such an attacker exists, then that attacker can also solve some problem Y in polynomial-time. We call this a security reduction. The security of the (possibly complex) scheme then relies on the hardness of this problem.

One could argue that we just moved the problem somewhere else. However such a reduction gives three main advantages:

1. It shows that there are no particular weaknesses in the complex details of the scheme.

2. Cryptanalysis can focus on a (hopefully) simple to state problem.

3. It allows to reuse and standardize security assumptions.

In the early days of cryptography, before security reductions were a thing, schemes could often be broken by details that were overlooked, such as statistical leakage. A security reduction often finds these faulty details, and brings them to the foreground, because they naturally represent an obstacle in the proof.

Security reductions also allow the reuse of a security assumption: many different schemes can reduce to the same problem. For example, within lattice-based cryptography almost all schemes reduce to just a few problems (but with varying parameters). As a result many cryptanalysts have focussed on those few problems, and thereby increased (or decreased) our confidence in them. Such a toolkit of standard and versatile security assumptions prevents an explosion of ad-hoc assumptions with potentially subtle weaknesses. However, care should still be taken when picking the parameters of such problems. Wandering off the beaten path too much could lead to unexpected attacks.

2.7.2 Zero-Knowledge Proof of Knowledge

As our first cryptographic concept we consider a Zero-Knowledge Proof of Knowledge (ZKPoK). The setting is that we know a solution w to some NP problem x, and we want to convince someone else that we know a solution *without revealing it*. Given an NP relation $\mathcal{R}$, we assume to have a public problem x and two parties, the Prover that supposedly knows a witness w such that $(x, w) \in \mathcal{R}$, and a Verifier that must be convinced of this. For example, there could be a public graph $x := G$, of which the Prover knows an automorphism $w := \sigma$. The Prover then wants to convince some verified that it knows such an automorphism of G, without revealing any such automorphism.

We assume the Prover and Verifier interact with each-other following a sigma protocol, consisting of three phases: commitment, challenge and a response. As visualized in Figure 2.10, the Prover, with input the problem x and a (potential) witness w, starts by creating

Prover(x, w)		Verifier(x)
Generate commitment u		
	$\xrightarrow{u}$	
		Generate challenge $c \leftarrow \mathcal{C}$
	$\xleftarrow{c}$	
Generate response z		
	$\xrightarrow{z}$	Output accept or reject.

Figure 2.10: A Sigma-protocol.

some commitment u. The Verifier, on input of the problem x and the commitment u, responds with a uniform challenge c from some finite challenge set $\mathcal{C}$. Finally, the Prover, responds to the challenge with some response z, which the Verifier accepts or rejects. For a successful Zero-Knowledge Proof of Knowledge scheme we expect the protocol to have the following three properties. The first two relate to the Proof of Knowledge part, while the last one defines the Zero-Knowledge part.

Completeness. If a Prover honestly knows a solution, then the Verifier must be convinced. Formally, if a Prover knows a witness w for the problem x, then after the protocol the Verifier must accept with probability 1.

Special Soundness. If the Prover does not know a witness then it should not be able to convince the Verifier. By guessing the challenge a priori correctly the Prover can often cheat by constructing a commitment and response that makes the Verifier accept for this particular challenge. The initial guess has probability $1/|\mathcal{C}|$ to be correct, and thus the Prover can cheat with probability $1/|\mathcal{C}|$, e.g., a half when $|\mathcal{C}| = 2$. If the Prover can cheat with at most negligible advantage over $1/|\mathcal{C}|$, then by repeating the protocol the overall cheat probability quickly becomes negligible.

To formalize this we must introduce an efficient knowledge extractor $\mathcal{E}$ that given two accepting transcripts (u, c, z) and (u, c', z') with the same commitment but distinct challenges $c \neq c'$, recovers a witness w for the problem x such that $(x, w) \in \mathcal{R}$. I.e., if a Prover is

able to convince the Verifier on at least two distinct challenges with non-negligible probability, then it also knows or can produce a witness with non-negligible probability.

Honest-Verifier Zero-Knowledge (HVZK). The Verifier, if acting honestly, does not learn anything else from the interaction, except that the Prover knows a witness. With acting honestly we mean that the challenges are indeed uniform random over the challenge set $\mathcal{C}$. Given the existence of transformations that remove the need for the Verifier to generate the challenge, we can safely assume this.

To show that the verifier cannot learn anything from the interaction we must show that we can efficiently create a fake interaction, following the same distribution as the real one, without any knowledge of a witness. Formally, we need an efficient simulator that given x, generates accepting transcripts (u, c, z), with a distribution indistinghuisable (negligible statistical distance) from accepting transcripts in the original Sigma-protocol

2.7.3 Encryption and Key Encapsulation Mechanism

While cryptography is a rich field containing many security primitives, many non-experts might think cryptography is only about the encryption of data (or cryptocurrencies...). And indeed encryption, the act of securely transferring data between parties, is a central topic in cryptography. The area can roughly be split into two: symmetric and asymmetric cryptography. In symmetric cryptography one assumes that both parties already conversed a shared secret key (or two related keys), which is then used both for encryption and for decryption. In asymmetric encryption, also known as public key encryption, parties do no share any secret information a priori. Usually one party generates both a secret and a public key, other parties encrypt using the public key, and the generating party decrypts using the secret key.

Often these two methods of encryption are used in unison, first public key encryption is used to securely obtain a (small) shared secret key, also known as a Key Encapsulation Mechanism (KEM), after which more efficient symmetric encryption is used to transfer (large) amounts of data using the shared secret key (see [KL20, Theorem

12.12]). Here we will focus ourself on the public key encryption, or more specifically the KEM.

Definition 58 (Key-Encapsulation Mechanism (KEM) [KL20]). *Let $\lambda \in \mathbb{Z}_{>0}$ be the security parameter. A Key-Encapsulation Mechanism (KEM) $\mathcal{K}$ consists of three probabilistic polynomial-time (in λ) algorithms $\mathcal{K} := (\mathbf{Gen}, \mathbf{Encaps}, \mathbf{Decaps})$ such that:*

1. *Key-generation:* **Gen** *takes a input the security parameter λ, and outputs a secret and public key (sk, pk).*

2. *Encapsulation:* **Encaps** *takes as input a public key pk, and outputs a ciphertext c and a key $\mathbf{k} \in \{0,1\}^\ell$, where $\ell = \Omega(\lambda)$ is the key length. We write $(c, \mathbf{k}) \leftarrow \mathbf{Encaps}(pk)$.*

3. *Decapsulation:* **Decaps** *takes as input a private key sk and a ciphertext c, and outputs a key $\mathbf{k}$ or a failure symbol $\perp$. We write $\mathbf{k} \leftarrow \mathbf{Decaps}(sk, c)$.*

*We call a KEM **correct** if with all but negligible probability over the randomness of* **Gen** *and* **Encaps**, *if* $\mathbf{Encaps}(pk)$ *outputs $(c, \mathbf{k})$, then* $\mathbf{Decaps}(sk, c)$ *also outputs* $\mathbf{k}$.

We now introduce a security notion versus Chosen-Plaintext Attacks (CPA). These definitions often take the form of a game that interacts with the adversary.

Definition 59 (CPA-Security [KL20]). *Let $\mathcal{K} = (\mathbf{Gen}, \mathbf{Encaps}, \mathbf{Decaps})$ be a KEM and $\mathcal{A}$ any adversary. The KEM CPA indistinguishability experiment $\mathbf{KEM}^{\mathbf{cpa}}_{\mathcal{A},\mathcal{K}}(\lambda)$ is as follows:*

- $\mathbf{Gen}(\lambda)$ *is run to obtain a public key $pk = P$. Then* $\mathbf{Encaps}(pk)$ *is run to generate $(c, \mathbf{k})$ with $\mathbf{k} \in \{0,1\}^\ell$ of key length $\ell = \Omega(\lambda)$.*

- *A uniform bit $b \in \{0,1\}$ is chosen. If $b = 0$, set $\hat{\mathbf{k}} := \mathbf{k}$, if $b = 1$, choose a uniform $\hat{\mathbf{k}} \in \{0,1\}^\ell$.*

- *Given $(pk, c = (\mathbf{c}, Z), \hat{\mathbf{k}})$ the adversary $\mathcal{A}$ guesses b. If correct the experiment returns 1, otherwise it returns 0.*

The KEM $\mathcal{K}$ is ***CPA-secure*** *if for all probabilistic polynomial-time adversaries $\mathcal{A}$ we have*

$$\Pr[\mathbf{KEM}^{\mathbf{cpa}}_{\mathcal{A},\mathcal{K}}(\lambda) = 1] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

One can trivially guess correctly with probability at least $\frac{1}{2}$, and we call the difference away from $\frac{1}{2}$ the *advantage*. So the goal is to achieve a non-negligible advantage in distinguishing the real key from a uniformly random one given the ciphertext and public key.

Recall that usually we want to reduce the CPA-security to some (hopefully) hard problem. Given the distinguishing nature of the CPA experiment, these problems are often of a decisional or distinguishing type, even though for an actual key or message recovery attack one might need to solve a search variant.

CPA-security can be seen as passive security, i.e., it shows that the scheme is secure from a passive attacker that can only eavesdrop on the public channels, but cannot interfere in any way. A stronger security notion is that of chosen-ciphertext attacks (CCA), where the adversary can actively interact with the party holding the secret key, and is even allowed to ask for the decapsulation of ciphertexts (except for the one given in the experiment). Since we can transform a CPA-secure KEM into a CCA-secure KEM by a Fujisaki-Okomoto type of transform [FO99] we restrict our focus to CPA-security.

2.7.4 Signatures

The use of signatures on physical documents or artworks dates back as far as 3000 BC, and they have since been used to associate an identity to a physical object. In the digital world signatures are just as important to verify that e.g., a digital document or software installer was created by some known trusted party, and not by some ill-intentioned adversary. Adding some random scribble to a document is not going to work in the digital world, as one can perfectly copy a signature from one file to another. The signature should be tied to the message, i.e., changing the message should invalidate the signature. We need cryptography to solve this problem for us.

We want the trusted party, called the *sender*, to be able to sign a message, after which anyone, acting as a *verifier*, should be able to

verify the signature. Again this brings us to the public key cryptography setting, the sender uses a secret key to sign, while the public key can be used to verify the signature. Note that in some sense the identity of the sender is represented by its public key, which we assume to be known by all parties. The latter might be seen as a strong requirement, and it is, as we will still need a reliable way to distribute the public key. What a signature scheme does however is to reduce the authentication of many messages to that of authenticating the public key *once*. The remaining problem of sharing public keys is in practice solved by establishing by a Public-Key Infrastructure with a single trusted root. The public key belonging to the trusted root can for example be pre-baked into operating systems, which can then be used to bootstrap the distribution of other public keys. For more information about this see [KL20, Chapter 13.6].

Definition 60 (Signature Scheme [KL20]). *Let $\lambda \in \mathbb{Z}_{>0}$ be the security parameter. A signature scheme $\mathcal{S}$ consists of three probabilistic polynomial-time (in λ) algorithms $\mathcal{S} := (\mathbf{Gen}, \mathbf{Sign}, \mathbf{Verify})$ such that:*

1. *Key-generation:* **Gen** *takes a input the security parameter λ, and outputs a secret and public key (sk, pk).*

2. *Signing:* **Sign** *takes as input a private key sk and a message m from some message space $\mathcal{M}$, and outputs signature σ. We write $\sigma \leftarrow \mathbf{Sign}(sk, m)$.*

3. *Verification:* **Verify** *takes as input a public key pk, a message m, and a signature σ. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write $b \leftarrow \mathbf{Verify}(pk, m, \sigma)$.*

We call a signature scheme ***correct*** *if with all but negligible probability over the randomness of* **Gen**, *if* $\mathbf{Sign}(sk, m)$ *outputs σ, then* $\mathbf{Verify}(pk, m, \sigma)$ *is valid for every legal message $m \in \mathcal{M}$.*

Given any message and a valid signature we should be convinced that the message was signed by the sender. A message together with a valid signature is called a *forgery* if the message was not signed before by the sender.

Definition 61 (EUF-CMA secure [KL20]). *Let* $\mathcal{S} := (\mathbf{Gen}, \mathbf{Sign}, \mathbf{Verify})$ *be a signature scheme and* $\mathcal{A}$ *any adversary. The signature forging experiment* $\mathbf{Sig\text{-}forge}_{\mathcal{A},\mathcal{K}}(\lambda)$ *is as follows:*

- $\mathbf{Gen}(\lambda)$ *is run to obtain keys* (pk, sk).
- *The adversary* $\mathcal{A}$ *is given the public key* pk *and access to an oracle* $\mathbf{Sign}(sk, \cdot)$. *The adversary outputs* (m, σ) *where* m *was not queried before to the oracle.*
- *The output of the experiment is given by* $\mathbf{Verify}(pk, m, \sigma)$, *and we say it succeeds if it is* 1.

The signature scheme $\mathcal{S}$ *is* ***existentially unforgeable under an adaptive chosen-message attack*** *(EUF-CMA) or just* ***secure****, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *we have*

$$\Pr[\mathbf{Sig\text{-}forge}_{\mathcal{A},\mathcal{K}}(\lambda) = 1] \leq \mathrm{negl}(\lambda).$$

2.7.5 Randomness extractors

A randomness extractor allows, using a publicly known random seed, to convert a non-uniform randomness source X with high min-entropy $H_\infty(X) := -\log_2(\max_x \Pr[X = x])$ to a near-uniform random variable [HILL99; Bar+11].[2]

Definition 62 (Extractor). *An efficient function* $\mathcal{E} : \mathcal{X} \times \{0,1\}^z \to \{0,1\}^v$ *is an* (m, ϵ)*-extractor, if, for all random variable* X *distributed over* $\mathcal{X}$ *and* $H_\infty(X) \geq m$, *it holds that*

$$\mathrm{dist}\left((Z, \mathcal{E}(X, Z)), (Z, V)\right) \leq \epsilon$$

where the seed $Z \leftarrow \mathcal{U}(\{0,1\}^z)$ *and* $V \leftarrow \mathcal{U}(\{0,1\}^v)$ *are drawn uniformly at random, and independently of* X.

In Chapter 10, we will rely on the existence of an (m, ϵ)-extractor with parameters $m = \Theta(v)$ and $\epsilon = 2^{-\Theta(m)}$.

[2] For our application in Chapter 10, we do not need to relax the source to only have average min-entropy, and therefore work with the simpler worst-case version.

2.7.6 Hash functions and the Random Oracle Model

Hash functions are an important tool both in cryptography and beyond. They take a (possibly) long input and output a short fixed length 'fingerprint'.

Definition 63 (Hash Function [KL20]). *Let $\lambda \in \mathbb{Z}_{>0}$ be the security parameter. A hash function $\mathcal{H}$ is a pair of probabilistic polynomial-time algorithms $(\mathbf{Gen}, H)$ such that:*

- *Key-generation:* **Gen** *takes as input the security parameter 1^λ, and outputs a key s.*
- *Hash: H is a deterministic algorithm that takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in \{0,1\}^{\ell(\lambda)}$ of fixed length.*

An important application for hash functions is inside hash-maps, where in general a value x is stored at the memory adress $H(x)$ (or nearby). By using the right hash function, preferably acting somewhat random, and using the right parameters this allows for a map with constant time insertion and look-up.

A simple application in cryptography is to combine them with a signature scheme to obtain a so-called hash-and-sign signature scheme. Instead of directly signing a long message m, which might be inefficient, we sign the short hashed message $H(m)$. Note however that any other message $m' \neq m$ with $H(m) = H(m')$ would give a forgery in the hash-and-sign scheme. To make sure the resulting scheme is still secure we require the hash function to be collision resistant.

Definition 64 (Collision Resistance [KL20]). *Let $\mathcal{H} := (\mathbf{Gen}, H)$ be a hash function and $\mathcal{A}$ any adversary. The collision-finding experiment* $\mathbf{Hash\text{-}coll}_{\mathcal{A},\mathcal{K}}(\lambda)$ *is as follows:*

- $\mathbf{Gen}(1^\lambda)$ *is run to obtain a key s.*
- *The adversary $\mathcal{A}$ is given the key s, and outputs x, x'.*
- *The output of the experiment is defined to be 1 if and only if $x \neq x'$ and $H^s(x) = H^s(x')$. We call this a collision.*

*The hash function $\mathcal{H}$ is **collision resistant**, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ we have*

$$\Pr[\mathbf{Hash\text{-}coll}_{\mathcal{A},\mathcal{K}}(\lambda) = 1] \leq \mathrm{negl}(\lambda).$$

Note in particular that a collision resistant hash function is hard to invert, e.g., it is pre-image resistant. Hash functions that are constructed with the goal of having such cryptographic properties are often called cryptographic hash functions. In practice however most of the efficient cryptographic hash functions in use have no proof of collision resistance (even under some hardness assumptions), but have simply resisted the state-of-the-art cryptanalysis. Additionally proving security of for example a signature scheme often requires that the hash function 'behaves like random', i.e., in the ideal case the hash function is a completely random function from the space of functions from the domain to $\{0,1\}^{\ell(\lambda)}$. Although this is impossible to achieve (efficiently) in practice, we often heuristically assume this is the case. Proofs under this heuristic are known as proofs in the Random Oracle Model (ROM).

In this model there is a public function $H : \mathrm{Domain} \to \{0,1\}^{\ell(\lambda)}$, that on any new input outputs a uniformly random value from $\{0,1\}^{\ell(\lambda)}$. Furthermore, the output should stay the same on already requested values. In the security proof one can 'reprogram' the function H on certain inputs, as long as the output is still (statistically indistinguishably from) uniform, with the argument that from the adversary point of view the distribution of H is unchanged. This is useful for proofs that require simulations.

Part II

Short and Close Lattice Vectors

CHAPTER 3

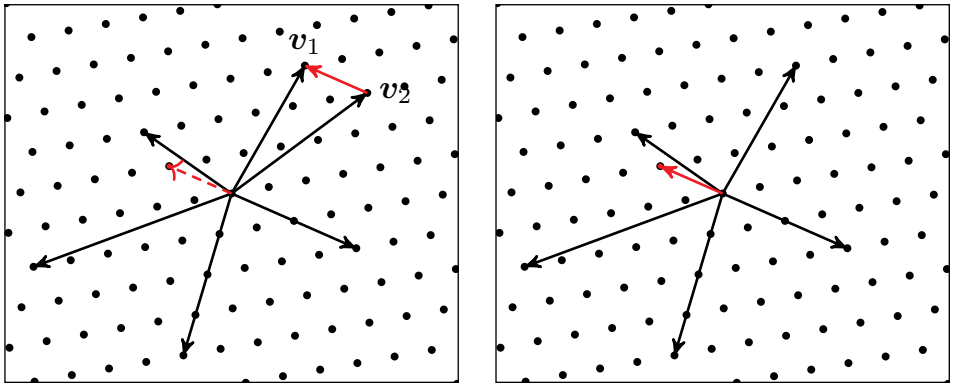
Theory of Lattice Sieving

This chapter gives an introduction to the state-of-the-art of lattice sieving.

3.1 Introduction

The hardness of the Shortest Vector Problem (SVP) is fundamental to the security of lattice-based cryptography; an efficient algorithm for SVP would break almost all lattice-based schemes. More precisely, the best asymptotic and concrete attacks on many schemes boil down to solving SVP in some (possibly lower dimensional) lattice, and thus parameters for lattice-based schemes are directly influenced by the complexity of solving SVP.

Lattice sieving algorithms solve SVP in single exponential time, making them asymptotically superior to enumeration techniques running in super-exponential time, at the cost of also using single exponential space. The central idea of sieving algorithms is to start with a large list of (long) lattice vectors, and to find many sums and differences of these vectors that are shorter. These shorter combinations are inserted back into the list, possibly replacing longer vectors, and this



Find close vectors $\boldsymbol{v}_1, \boldsymbol{v}_2 \in L$. Replace $\boldsymbol{v}_2 \leftarrow \boldsymbol{v}_1 - \boldsymbol{v}_2$.

Figure 3.1: Lattice Sieving

process is repeated until the list contains many short vectors, among which (hopefully) a shortest one of the lattice. See Figure 3.1 for one such iteration.

The first lattice sieving algorithm [AKS01] was proposed in 2001 by Ajtai, Kumar, and Sivakumar (AKS), giving the first provable algorithm that solved SVP in single exponential time $2^{O(n)}$ for a rank n lattice. By choosing the AKS parameters carefully, Nguyen et al. later showed that the AKS sieve can provably solve SVP in time $2^{5.9n+o(n)}$ and space $2^{2.95n+o(n)}$. In a further line of work [PS09; MV10; HPS11a] algorithmic and analytic improvements brought the constants down to time $2^{2.456n+o(n)}$ and space $2^{1.233n+o(n)}$. Provably solving (worst-case) SVP with lattice sieving leads to many technical problems, such as showing that we can actually find enough short combinations and in particular that they are new, i.e., they are not present in our list yet; unfortunately, side-stepping these technicalities leads to the mentioned high time and memory complexities.

In contrast, for cryptanalysis we are interested in the average-case behaviour of lattice sieving algorithms. Instead of provable results these algorithms are based on heuristics, which are often verified experimentally, or which can only be proven (partially) for average-case instances. The first and simplest of these practical sieving algorithms by Nguyen and Vidick uses a list of $N = (4/3)^{n/2+o(n)} = 2^{0.2075n+o(n)}$ vectors and runs in time $N^{2+o(1)} = 2^{0.415n+o(d)}$ by repeatedly checking

all pairs $\mathbf{v} \pm \mathbf{w}$ [NV08]. The list size of $(4/3)^{n/2+o(n)}$ is the minimal number of vectors that is needed in order to keep finding enough shorter pairs.

In a line of works [Laa15; BGJ15; BL16; BDGL16] the time complexity was gradually improved to $2^{0.292n+o(n)}$ by nearest neighbour searching techniques to find close pairs more efficiently [IM98]. Instead of checking all pairs they first apply some bucketing strategy in which close vectors are more likely to fall into the same bucket. By only considering the somewhat-close pairs inside each bucket, the total number of checked pairs can be decreased.

To lower the space cost below $N = 2^{0.2075+o(n)}$ one has to look beyond sums and differences of pairs, and consider triples $\mathbf{u} \pm \mathbf{v} \pm \mathbf{w}$ or more generally k-tuples [BLS16; HK17; HKL18]. The current best triple sieve [HKL18] has a space complexity of $2^{0.1887n+o(n)}$, but comes with a higher time complexity of $2^{0.3588n+o(n)}$.

3.1.1 Organisation

In Section 3.2 we introduce the basics of lattice sieving algorithms and their heuristic analysis. Section 3.3 introduces some Nearest Neighbour Search techniques to obtain asymptotically faster sieving algorithms. In Section 3.4 we introduce more advanced practical sieving techniques that give significant polynomial and even subexponential speed-ups. In Section 3.5 we give an overview of the General Sieve Kernel (G6K), the current state-of-the-art framework and implementation for (lattice reduction via) lattice sieving.

3.2 Heuristic lattice sieving

In this section we introduce a very basic lattice sieving algorithm inspired by the first practical sieving algorithm [NV08], and explain heuristically why it works. Lattice sieving algorithms start with a large list $L_0 \subset \mathcal{L}$ consisting of N long lattice vectors. Given any basis one can always produce long lattice vectors by discrete Gaussian sampling, or simply by taking small random combinations of the basis vectors. Note that due to the additive structure of the lattice, for any two lattice vectors $\mathbf{x}, \mathbf{y} \in L_0$, the difference $\mathbf{x} - \mathbf{y}$ is still a lattice vector. And, in particular the difference might be a *shorter* lattice

vector. The idea behind lattice sieving is to compute the difference $\mathbf{x} - \mathbf{y}$ for all lattice vectors $\mathbf{x}, \mathbf{y} \in L_0$ and store those that are short enough in a new list L_1. We call such pairs $(\mathbf{x}, \mathbf{y})$ a *reduction*.

Definition 65 (Reduction). *Let $R > 0$ be some length bound (clear from the context), and let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ be (lattice) vectors, we call $(\mathbf{x}, \mathbf{y})$ a reduction or a reducing pair if*

$$\|\mathbf{x} - \mathbf{y}\| \leq R.$$

With the appropriate length bound the new list L_1 contains shorter vectors than the original list L_0, and this process is repeated with more lists $L_2, L_3, \ldots$ and decreasing bounds $R_2 \geq R_3 \geq \ldots$ until some saturation condition is met (which will be discussed later). This basic sieving algorithm is summarized in Algorithm 2. Given the algorithm as stated it might as well happen that such reductions do not exist, or at least not enough of them, such that after some iterations we have $L_i = \{\mathbf{0}\}$. Clearly if we add more vectors to the initial list L_0, then more pairs are considered, and thus we expect to find more reductions.

Algorithm 2: Lattice sieving algorithm.

Input : A basis $\mathbf{B}$ of a lattice $\mathcal{L}$, list size N, a saturation radius R and a progress factor $2/3 < \gamma < 1$.
Output: A list L of short vectors saturating the ball of radius R.

Sample a list $L_0 \subset \mathcal{L}$ of size N.
$i \leftarrow 0$.
while L_i *does not saturate the ball of radius* R **do**
  $R_{i+1} \leftarrow \gamma \cdot \max_{\mathbf{v} \in L_i} \|\mathbf{v}\|$
  $L_{i+1} = \emptyset$.
  for *every pair* $\mathbf{v}, \mathbf{w} \in L_i$ **do**
    if $\mathbf{v} - \mathbf{w} \notin L_{i+1}$ ***and*** $\|\mathbf{v} - \mathbf{w}\| \leq R_{i+1}$ **then**
      $L_{i+1} \leftarrow L_{i+1} \cup \{\mathbf{v} - \mathbf{w}\}$
    end
  end
  $i \leftarrow i + 1$
end
return L_i

So how large does the initial list $N = |L_0|$ need to be to find enough reductions?

Reduction probability

Let's first dive deeper into sufficient conditions for lattice vectors to give a reduction. Pairs $(\mathbf{x}, \mathbf{y})$ give a reduction if their difference is small enough, i.e., if they are somewhat close to each-other. Alternatively this means that the vectors $\mathbf{x}$ and $\mathbf{y}$ have a small angle to each-other. This can be made rigorous in terms of the inner product.

Lemma 66. *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ be (lattice) vectors, then $(\mathbf{x}, \mathbf{y})$ is a reduction for some length bound $R > 0$ if and only if*

$$\langle \mathbf{x}, \mathbf{y} \rangle \geq \frac{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - R^2}{2}.$$

in particular if $\|\mathbf{x}\| = \|\mathbf{y}\| = R$ then $(\mathbf{x}, \mathbf{y})$ is a reduction if and only if

$$\langle \mathbf{x}/\|\mathbf{x}\|, \mathbf{y}/\|\mathbf{y}\| \rangle \geq \frac{1}{2}.$$

A normalized inner product of at least $\frac{1}{2}$ corresponds to the vectors having an angle less than $\pi/3$ between each-other. Independent of the length of the vectors $\mathbf{x}, \mathbf{y}$, if $R \geq \max\{\|\mathbf{x}\|, \|\mathbf{y}\|\}$ then the condition that the angle is less than $\pi/3$ is sufficient. Note that in some sense this is the worst-case scenario, the angle condition is only necessary when the vectors have the same length.

In order to say something about the probability that a pair $(\mathbf{x}, \mathbf{y}) \in L_i \times L_i$ gives a reduction we have to assume that these vectors follow some distribution. This is where the heuristics come in. Recall that the Gaussian Heuristic indicates that given some ball $R \cdot \mathcal{B}^n$ of radius R, there are approximately $|R \cdot \mathcal{B}^n \cap \mathcal{L}| = (R/\operatorname{gh}(\mathcal{L}))^n$ lattice vectors in it, and those lattice vectors are uniformly distributed over the ball. In addition, for most use cases, any dependencies between the lattice vectors is ignored. Vectors that are uniformly distributed over a ball are in particular uniformly distributed over the sphere after normalization. Given the sufficient condition on the angle we can thus work with the following weaker heuristic.

Heuristic 67. *For nonzero list vectors $\mathbf{v} \in L_i \setminus \{\mathbf{0}\}$, the directions $\mathbf{v}/\|\mathbf{v}\|$ are independently uniformly distributed over the sphere $\mathcal{S}^{n-1}$.*

This heuristic matches what we observe in practice, and one could argue that this is in fact the worst-case distribution when ones goal is to find pairs that have a small angle[1], i.e., if the distribution is less uniform and biased into a certain direction then one would expect to find more pairs with a small angle.

As a technical motivation, ignoring dependency issues, note that if we sample the initial list from a discrete Gaussian with large enough parameter, then their directions are indeed close to uniform. Now if vectors $\mathbf{x}, \mathbf{y}$ have uniform directions, then their difference $\mathbf{x} - \mathbf{y}$ also has a uniform direction. Furthermore heuristically there is no direct relation between the direction of $\mathbf{x} - \mathbf{y}$ and its shortness, so the new list also contains vectors with uniform directions.

Given this heuristic we can compute a lower bound on the probability that any two vectors give a reduction. Note that for this only one of the two vectors has to follow the heuristic.

Lemma 68. *For any $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, let $\mathbf{y} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ be such that $\mathbf{y}/\|\mathbf{y}\|$ is uniform over $\mathcal{S}^{n-1}$, and let $R \geq \max\{\|\mathbf{x}\|, \|\mathbf{y}\|\}$ be a length bound, then the probability that $(\mathbf{x}, \mathbf{y})$ is a reduction is at least*

$$\Pr[\|\mathbf{x} - \mathbf{y}\| \leq R] \geq (3/4)^{n/2+o(n)},$$

with equality if $R = \|\mathbf{x}\| = \|\mathbf{y}\|$.

Proof. Let $\tilde{\mathbf{x}} = \mathbf{x}/\|\mathbf{x}\|, \tilde{\mathbf{y}} = \mathbf{y}/\|\mathbf{y}\|$, the condition $\langle \tilde{\mathbf{x}}, \tilde{\mathbf{y}} \rangle \geq \frac{1}{2}$ implies that $(\mathbf{x}, \mathbf{y})$ is a reduction with the length bound $R \geq \max\{\|\mathbf{x}\|, \|\mathbf{y}\|\}$. The condition is necessary only when $R = \|\mathbf{x}\| = \|\mathbf{y}\|$. For any $\tilde{\mathbf{x}} \in \mathcal{S}^{n-1}$ the probability

$$\Pr_{\tilde{\mathbf{y}} \sim \mathcal{S}^{n-1}}\left[\langle \tilde{\mathbf{x}}, \tilde{\mathbf{y}} \rangle \geq \frac{1}{2}\right] = \frac{\mathrm{vol}\left(\mathcal{C}^{n-1}_{\tilde{\mathbf{x}}, \frac{1}{2}}\right)}{\mathrm{vol}(\mathcal{S}^{n-1})} = \mathcal{C}^{n-1}(1/2) = (3/4)^{n/2} \cdot n^{O(1)},$$

is given by the relative volume $\mathcal{C}^{n-1}(1/2)$ of the spherical cap, whose asymptotic formula is given by Lemma 46. □

[1] Technically, the existence of a very good non-lattice kissing number configurations would imply a counter-example to this statement, but it is not clear that such a configuration exists, and in particular such a distribution does not occur naturally in practice.

Following Lemma 68 we have to consider about $(4/3)^{n/2+o(n)}$ pairs to find at least a single reduction. Given a list L_i of length N and a length bound of $R = \max_{\mathbf{v}\in L_i} \|\mathbf{v}\|$ we can consider about $\approx \frac{1}{2}N^2$ pairs, and the total number of reductions will heuristically be at least $N^2 \cdot (3/4)^{n/2+o(n)}$. These reductions form the new list L_{i+1} which again has to be of size close to N. Solving the list balancing equation

$$N = N^2 \cdot (3/4)^{n/2+o(n)},$$

we thus need a list of size $N = (4/3)^{n/2+o(n)}$ to make sure that the lists do not decrease in size (ignoring duplicates).

In Algorithm 2 the length bound is set slightly smaller at $R = \gamma \cdot \max_{\mathbf{v}\in L_i} \|\mathbf{v}\|$ for some progress factor $\gamma < 1$. Depending on the specific factor one would have to increase the list size slightly; but by setting e.g., $\gamma = 1 - \frac{1}{n}$ an asymptotic list size of $N = (4/3)^{n/2+o(n)}$ is again sufficient, while still obtaining an exponential decrease in the length bounds R_i.

Duplicates

In the above analysis we are assuming that each of the found reductions $\mathbf{x} - \mathbf{y}$ is distinct. This cannot stay true as otherwise we would eventually find nonzero vectors shorter than $\lambda_1(\mathcal{L})$ under Heuristic 67. Suppose the lattice is normalized to $\mathrm{gh}(\mathcal{L}) = 1$, i.e., such that the Gaussian Heuristic indicates that $\lambda_1(\mathcal{L}) \approx 1$. Asymptotically, for large n almost all of the reductions in the new list have length close to the length bound R (say between $0.99R$ and R), and given Heuristic 67 it is natural to assume that these reductions give vectors close to uniform among the lattice vectors in that thin layer. By the Gaussian Heuristic there exist about $R^{n+o(n)}$ such lattice vectors. Given $(4/3)^{n/2+o(n)}$ uniform samples from this set we only expect a significant $\Theta(1)$ fraction of duplicates when $R \leq \sqrt{4/3} + o(1)$. A natural saturation condition is thus to stop whenever L_i contains a significant fraction of the lattice vectors of length at most $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$, and before this happens the relative number of duplicates is expected to be insignificant.

Definition 69 (Saturation). *For a rank n lattice $\mathcal{L}$, we say that a list $L \subset \mathcal{L}$ of lattice vectors is saturating (a ball) with radius R if*

$$|\mathcal{L} \cdot R \cdot \mathcal{B}^n| \geq \tfrac{1}{2} \cdot (R/\,\mathrm{gh}(\mathcal{L}))^n$$

The saturation condition is usually applied with radius $R = \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$. The constant $\frac{1}{2}$ is somewhat arbitrary and can be tweaked in practice. A lower constant makes it easier to reach saturation but results in less short vectors in the final list. A larger constant (of at most 1) makes saturation harder to reach and increases the occurrence of duplicates, but results in more short vectors in the final list.

Heuristic runtime analysis

Again we consider a lattice $\mathcal{L}$ that is normalized such that $\mathrm{gh}(\mathcal{L}) = 1$. Suppose we set $\gamma = 1 - \frac{1}{n}$, then a list size of $N = (4/3)^{n/2+o(n)}$ is enough, and there are $N^2 = (4/3)^{n+o(n)}$ reductions to check every iteration, each at the cost of $O(n)$ arithmetic operations to compute an inner product. Checking for duplicates can be done in $O(\log N)$ or amortized $O(1)$ vector operations per vector by keeping the list sorted or by a hash-set.

By first LLL reducing the basis and then sampling vectors by taking small random combinations (or by Gaussian sampling) we can construct an initial list L_0 of vectors such that $R_1 = \max_{\mathbf{v} \in L_0} \|\mathbf{v}\| = 2^{O(n)}$. Given that $\gamma^k = (1 - \frac{1}{n})^k$ decreases like $(1/e)^{k/n}$ we obtain $R_i \approx \sqrt{4/3}$ after at most a polynomial number of $k = O(n^2)$ iterations, and following the previous discussion the saturation condition will (have) trigger(ed).

In total the complexity is thus dominated by the $\mathrm{poly}(n) \cdot N^2 = (4/3)^{n+o(n)}$ time cost of checking all pairs and the $\mathrm{poly}(n) \cdot N = (4/3)^{n/2+o(n)}$ memory cost of storing the lists.

A single list

Algorithm 2 uses multiple lists $L_1, L_2, \ldots$ to clarify the steps in the heuristic analysis. In practice it is common to use a single list, where new reduced vectors $\mathbf{x} - \mathbf{y}$ either replace the longest of $\mathbf{x}$ or $\mathbf{y}$, or replace just the longest element in the list. The list and length bound are then continuously updated and the vectors are getting shorter and shorter until saturation is achieved. This has the added benefit of explicitly reusing all the short enough vectors found so far (although this was implicitly done in Algorithm 2 by forcing $\mathbf{0}$ to be part of each list).

One could pick random pairs to reduce, which might result in a loss of performance due to rechecking the same pair. The approach of the Gauss Sieve [MV10] is to split the list into two parts: the queue part Q and the list part L. At the start of the sieve all vectors are in the queue part, and the list part is empty. If the queue part becomes too small before saturation is achieved we sample more vectors into it.

A queue vector $\mathbf{v} \in Q$ is only inserted in the list part if it does not form a reduction with any of the list vectors (with $R = \max\{\|\mathbf{v}\|, \|\mathbf{w}\|\}$ for $\mathbf{w} \in L$). This is achieved by explicitly checking for reductions between $\mathbf{v}$ and all list vectors, where any reduction $(\mathbf{v}, \mathbf{w})$ replaces the longest of the two vectors, and is inserted back into the queue part (if it was not in the database yet). By following this strategy any pair of vectors is only checked for a reduction once (except if a vector disappears and later reappears in the database), preventing most of the duplicate checks, and giving a significant speed-up in lowish dimensions.

Another improvement is to not store both the vector $\mathbf{x}$ and its negation $-\mathbf{x}$, which saves a factor 2 in memory. Additionally we can check if either $\pm(\mathbf{x}-\mathbf{y})$ or $\pm(\mathbf{x}+\mathbf{y})$ is short by looking at the absolute inner product $|\langle \mathbf{x}, \mathbf{y} \rangle|$, which saves a factor 4 in the number of inner products we have to compute.

Triple sieve

In Section 3.3 we will explain how to decrease the time complexity below $N^2 = (4/3)^{n+o(n)}$, by making reductions easier to find. However all such methods do no improve the memory usage below $N = (4/3)^{n/2+o(n)}$, as we still require the same number of reductions.

To improve the memory cost we have to go beyond pairs and consider triples $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ such that $\mathbf{x} \pm \mathbf{y} \pm \mathbf{z}$ is short or even k-tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ [BLS16; HK17; HKL18]. The probability that a triple uniformly sampled from the sphere gives a reduction is about $(16/27)^{n/2+o(n)}$, which is much smaller than for a pair. But at the same time there are about N^3 triples to consider, leading to the list balancing equation

$$N = (16/27)^{n/2+o(n)} \cdot N^3,$$

with solution $N = (27/16)^{n/4+o(n)} = 2^{0.1887n+o(n)}$ compared to a list size of $(4/3)^{n/2+o(n)} = 2^{0.2075n+o(n)}$ for the sieve based on pairs. This comes at the cost of a higher time, as we now need to check $N^3 = 2^{0.5662n+o(n)}$ triples. Considering tuples of $k \geq 4$ vectors further decreases the memory cost, but with a large impact on the runtime.

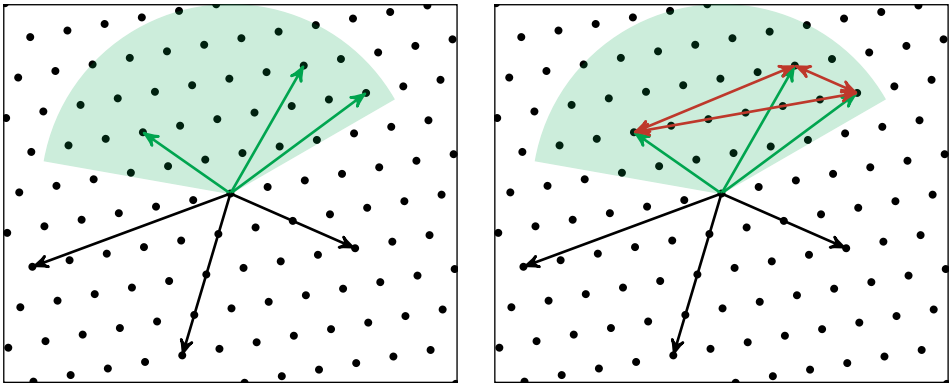
Provable sieves

In the literature on lattice sieving there is a large gap between the heuristic (and real-world average-case) performance of lattice sieving algorithms and the provable sieving algorithms. Heuristically we need a list size of about $(4/3)^{n/2+o(n)}$ to keep finding enough reduction, and this seems realistic in practice. However in the worst-case we could construct such a list that has no reduction at all. For a naive provable result on finding enough reductions one would have to at least square the list size to $(4/3)^{n+o(n)} = 2^{0.415n+o(n)}$. The exact number that is needed is directly related to the non-lattice variant of the kissing number, which bound can be improved to $2^{0.401n+o(n)}$.

Still the problem remains that many of the found reductions might collide, and this is where most of the technical difficulties in provable sieving algorithms stem from. One technical solution is to add extra random errors to the lattice point in order to somehow 'hide the lattice structure' from the sieving algorithm, which allows one to prove results about the distribution of the list and the probability of obtaining duplicates, but this comes at the cost of significantly increasing the asymptotic complexity. When allowing for a constant approximation factor, e.g., that the algorithm returns provably only a vector of size $\mu \cdot \lambda_1(\mathcal{L})$ for some constant $\mu > 1$, this increase in complexity can be prevented, leading to a $2^{0.802n+o(n)}$ time and $2^{0.401n+o(n)}$ space algorithm [AUV19; EV22; RV22], surprisingly not only in the standard Euclidean norm, but for any p-norm.

3.3 Bucketing and sieving variants

Recall from Section 3.2 that for a (pair-)sieve we need a list of $N = (4/3)^{n/2+o(n)}$ lattice vectors, and we need to find all the reductions among the N^2 pairs. Naturally the naive algorithm that checks all pairs takes a time complexity on the order of N^2.



Create a bucket $B \subset L$. Search reducing pairs in B.

Figure 3.2: Lattice sieving with bucketing.

This can be improved by bucketing methods. The idea is to first group vectors together that point somewhat in the same direction. Then within each group each pair is more likely to give a reduction; and this is where we improve the time complexity. To find all reductions we might have to repeat the bucketing process (after randomizing) for multiple iterations. See Algorithm 3.

Suppose that in each iteration we have m buckets $B_1, \ldots, B_m$. Each of the N vectors is added to on average M buckets, depending on some bucket condition, at a cost of T_b per vector. For convenience we will for now assume that $M \approx 1$, such that assuming that the buckets are well balanced, they are of size $|B_i| \approx N/m$.

We assume that any pair of vectors that lies in the same bucket has (at least) some conditional probability p to give a reduction. To find N reductions we thus have to check in total about N/p bucketed pairs, and when $p \gg N^{-1}$ this is less than in the naive algorithm.

This is only helpful if the cost of the bucketing process is less or equal to $\tilde{O}(N/p)$. Per iteration we have m buckets and we find on the order of $p \cdot (N/m)^2$ reductions per bucket. To find N reductions we thus have to repeat $r = \frac{m}{N \cdot p}$ many iterations. At each iteration we have to find an appropriate bucket for each of the N vectors at a cost of T_b per vector, the total cost of bucketing is thus given by

$$r \cdot N \cdot T_b = \frac{n}{N \cdot p} \cdot N \cdot T_b = \frac{m}{p} \cdot T_b.$$

As an alternative explanation of the bucketing cost note that each vector has after bucketing an expected number of $|B_i| \cdot p = \frac{N}{m} \cdot p$ reductions, to achieve a total of N reductions we thus have to bucket $N/(\frac{N}{m} \cdot p) = m/p$ vectors with a total cost of $\frac{m}{p} \cdot T_b$.

The total time complexity of bucketing and checking pairs for reductions is thus given by

$$T = \tilde{O}\left(\frac{m \cdot T_b}{p} + \frac{N}{p}\right).$$

Investing more time in bucketing often allows for more and higher quality buckets, which results in a larger reduction probability p and thus in a lower reduction cost. Investing less time in bucketing in general leads to a higher reduction cost. For a fixed bucketing algorithm the optimal parameters are (often) such that the above terms are balanced, which means that $m \cdot T_b \approx N$.

Algorithm 3: Bucketed lattice sieving algorithm.

Input : A basis $\mathbf{B}$ of a lattice $\mathcal{L}$, list size N, and a saturation radius R.
Output: A list L of short vectors saturating the ball of radius R.

```
Sample a list L ⊂ ℒ of size N.
while L does not saturate the ball of radius R do
    B_1, ..., B_m ← Bucket(L, m).
    for every bucket B_i do
        for every pair v, w ∈ B_i do
            if v − w ∉ L and ‖v − w‖ < max_{u∈L} ‖u‖ then
                Replace longest vector in L by v − w.
            end
        end
    end
end
return L
```

We now discuss two different bucketing algorithms, and their optimal time complexity. The first is relatively simple and very practical, while the second is somewhat more complicated but reaches the best known asymptotic time complexity.

BJG1 sieve

We start with the simple 1 layer variant of [BGJ15], better known as the BGJ1 sieve. Recall that the idea of bucketing is to subdivide our large list into groups of vectors pointing somewhat in the same direction. We can take this quite literal and pick $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathcal{S}^{n-1}$ bucket directions uniformly at random, and put a vector $\mathbf{v}$ in bucket i if it points mostly in the direction of $\mathbf{b}_i$, i.e., if $\langle \mathbf{b}_i/\|\mathbf{b}_i\|, \mathbf{v}/\|\mathbf{v}\| \rangle \geq \alpha$ for some parameter $\alpha > 0$. For a vector $\mathbf{v}$ we can compute the above inner product for each of the m buckets, and thus we obtain a bucketing cost of $T_b = O(m \cdot n)$ per vector.

Geometrically each bucket presents a spherical cap $\mathcal{C}^{n-1}_{\mathbf{b}_i,\alpha}$ in some random direction. The probability that a (normalized) lattice vector lies in such a cap is given by its relative volume $\mathcal{C}^{n-1}(\alpha) = (1-\alpha^2)^{n/2+o(n)}$, and thus we need $m = (1-\alpha^2)^{-n/2+o(n)}$ such buckets to make sure each vector is on average part of $M \approx 1$ bucket. I.e., geometrically we need about $(1-\alpha^2)^{-n/2+o(n)}$ such spherical caps to cover each point on the sphere (in expectation).

Now what about the improved probability p, that a pair inside a bucket gives a reduction. Recall that before bucketing this probability is about $(3/4)^{n/2+o(n)}$. For a pair inside a spherical cap with parameter α the probability is, as expected, better.

Lemma 70. *Let $\mathbf{b} \in \mathcal{S}^{n-1}$ be a direction, and let $\mathbf{v}, \mathbf{w} \sim \mathcal{S}^{n-1}$. Then for any parameter $\sqrt{1/2} > \alpha > 0$ we have*

$$\Pr\left[\langle \mathbf{v}, \mathbf{w} \rangle \geq \frac{1}{2} \middle| \langle \mathbf{v}, \mathbf{b} \rangle \geq \alpha \text{ and } \langle \mathbf{w}, \mathbf{b} \rangle \geq \alpha\right] = \left(\frac{\frac{3}{4} - \alpha^2}{(1-\alpha^2)^2}\right)^{n/2+o(n)}.$$

Proof. Let A be the event that $\langle \mathbf{v}, \mathbf{w} \rangle \geq \frac{1}{2}$, and let B be the event that $\langle \mathbf{v}, \mathbf{b} \rangle \geq \alpha$ *and* $\langle \mathbf{w}, \mathbf{b} \rangle \geq \alpha$. Note that $\Pr[A] = \mathcal{C}^{n-1}(1/2)$, and $\Pr[B] = \mathcal{C}^{n-1}(\alpha)^2$. For $\gamma \in [\frac{1}{2}, 1]$ the probability density function for the event $\langle \mathbf{v}, \mathbf{w} \rangle = \gamma$ is asymptotically proportional to $\mathcal{C}^{n-1}(\gamma) = (1-\gamma^2)^{n/2}$. Furthermore the probability $\Pr[B|\langle \mathbf{v}, \mathbf{w} \rangle = \gamma]$ can geometrically be interpreted as the wedge $\mathcal{W}^{n-1}_{\mathbf{v},\mathbf{w},\alpha} := \mathcal{C}^{n-1}_{\mathbf{v},\alpha} \cap \mathcal{C}^{n-1}_{\mathbf{w},\alpha}$ with relative volume $\mathcal{W}^{n-1}(\alpha, \gamma)$. We have

$$\mathcal{W}^{n-1}(\alpha, \gamma) \cdot \mathcal{C}^{n-1}(\gamma) = ((1 + \gamma - 2\alpha^2)(1-\gamma))^{n/2+o(n)},$$

and for $0 < \alpha < \sqrt{1/2}$ the constant $(1 + \gamma - 2\alpha^2)(1-\gamma)$ is strictly maximized for $\gamma = \frac{1}{2}$. Asymptotically, we thus have $\Pr[B|A] =$

$\Pr[B|\langle \mathbf{v}, \mathbf{w}\rangle \geq \frac{1}{2}] = 2^{o(n)} \cdot \Pr[B|\langle \mathbf{v}, \mathbf{w}\rangle = \frac{1}{2}] = (1 - \frac{4}{3}\alpha^2)^{n/2+o(n)}$. Now by Bayes' theorem we have

$$\Pr[A|B] = \frac{\Pr[B|A] \cdot \Pr[A]}{\Pr[B]} = \frac{(1 - \frac{4}{3}\alpha^2)^{n/2+o(n)} \cdot \mathcal{C}^{n-1}(1/2)}{\mathcal{C}^{n-1}(\alpha)^2} = \left(\frac{\frac{3}{4} - \alpha^2}{(1-\alpha^2)^2}\right)^{n/2+o(n)}.$$

□

To summarize we have $m = (1-\alpha^2)^{-n/2+o(n)}, T_b = \tilde{O}(m)$, and $p = (\frac{3}{4} - \alpha^2/((1-\alpha^2)^2))^{n/2+o(n)}$. Now let us pick α such that the total time complexity is minimized. From $N = m \cdot T_b = \tilde{O}(m^2)$ we obtain $(1-\alpha^2)^{-n/2+o(n)} = m \approx N^{1/2} = (4/3)^{n/4+o(n)}$ and thus $\alpha = \sqrt{1 - \sqrt{4/3}} + o(1) \approx 0.366$. This gives $p = (\sqrt{4/3} - \frac{1}{3})^{n/2+o(n)} = 2^{-0.142n+o(n)}$, and a total time complexity of $N/p = 2^{0.349n+o(n)}$.

When applying the same bucketing strategy recursively the time complexity can be further reduced to $2^{0.311n+o(n)}$ [BGJ15]. However, the recursion comes with large overheads, and therefore it is unclear how practical this version is.

BDGL sieve

The asymptotically optimal bucketing method from [BDGL16] is similar to BGJ1 as in that it is based on spherical caps. The difference is that in contrast to BGJ1 the bucket centers are not arbitrary but structured, allowing to find the correct bucket without having to compute the inner product with each individual bucket center. This allows us to reduce the bucketing cost below $T_b = \tilde{O}(m)$ per vector.

Following [BDGL16], such a bucketing strategy looks as follows. First we split the dimension n into k smaller blocks of similar dimensions $n_1, \ldots, n_k$ that sum up to n. In order to randomize this splitting over different iterations one first applies a random orthonormal transformation $\mathbf{Q}$ to each input vector. Then the set C of bucket centers is constructed as a direct product of random local bucket centers, i.e., $C = \pm C_1 \times \pm C_2 \cdots \times \pm C_k$ with $C_b \subset \mathbb{R}^{n_b}$. A (normalised) list vector $\mathbf{v}$ is then similarly split into parts $\mathbf{v}_1, \ldots, \mathbf{v}_k$, and we have to find the bucket centers $(\mathbf{c}_1, \ldots, \mathbf{c}_k) \in C$ such that $\sum_i \langle \mathbf{c}_i, \mathbf{v}_i\rangle \geq \alpha$. Note that to

Bucketing	T_b	$\lvert B_i\rvert$	c_{time}
None	0	$\tilde{O}(N)$	0.415
BGJ1	$\tilde{O}(m)$	$\tilde{O}(N^{1/2})$	0.349
BDGL ($k=1$)	$\tilde{O}(m)$	$\tilde{O}(N^{1/2})$	0.349
BDGL ($k=2$)	$\tilde{O}(m^{1/2})$	$\tilde{O}(N^{1/3})$	0.329
BDGL ($k=3$)	$\tilde{O}(m^{1/3})$	$\tilde{O}(N^{1/4})$	0.320
BDGL ($k=\Theta(\log(n))$)	$\tilde{O}(m^{1/k})$	$\tilde{O}(N^{1/(k+1)})$	0.292

Table 3.1: Bucketing cost T_b per vector, optimal bucket size $\lvert B_i\rvert$ and the resulting time complexity $2^{c_{\text{time}}n+o(n)}$ for the discussed bucketing algorithms with a space complexity of $N = 2^{0.2075n+o(n)}$.

find the closest global bucket center to $\mathbf{v}$, we only have to pick the closest local bucket centers $\mathbf{c}_i$ to $\mathbf{v}_i$, implicitly considering $m = 2^k \prod_b |C_b|$ global bucket centers at the cost of only $T_b = \sum_b |C_b| \approx O(m^{1/k})$ local inner products. By sorting the local inner products we can also efficiently find all bucket centers within a certain angle. For a fixed number of buckets m we can expect some performance loss compared to BGJ1 as the bucket centers are not perfectly random, but this does not influence the asymptotics. I.e., the analysis of [BDGL16, Theorem 5.1] shows this leads to at most a subexponential loss if $k = O(\log(n))$, and the experimental analysis of [Duc22] shows that it is also reasonable small in practice.

To optimize the parameters we again balance the cost of bucketing and reducing. Note that for $k = 1$ we essentially obtain BGJ1 with buckets of size $\tilde{O}(N^{1/2})$ and a time complexity of $2^{0.349n+o(n)}$. For $k = 2$ or $k = 3$ the optimal buckets become smaller of size $\tilde{O}(N^{1/3})$ and $\tilde{O}(N^{1/4})$ respectively and of higher quality, leading to a time complexity of $2^{0.3294n+o(n)}$ and $2^{0.3198n+o(n)}$ respectively. By letting k slowly grow, e.g., $k = \Theta(\log(n))$ there will only be a subexponential $2^{o(n)}$ number of vectors in each bucket, leading to the best known time complexity of $2^{0.292n+o(n)}$. Note however that a lot of subexponential factors might be hidden inside this $o(n)$, and thus for practical dimensions a rather small value of $k = 2, 3, 4$ might give best results. See Table 3.1 for an overview of the time complexity of the discussed bucketing algorithms.

3.4 Advanced lattice sieving

In this section we discuss some more advanced techniques used in lattice sieving. The ideas and implementation tricks in this section only give subexponential, polynomial or even only constant speed-ups, and do not improve the asymptotic time complexity, however they do give significant practical speed-ups. These ideas have largely contributed to pushing the cross-over between enumeration and sieving techniques as low as dimension 70.

3.4.1 XOR Popcount SimHash

The main operation in lattice sieving algorithms is the computation of many pairwise inner products $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ given a list $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{R}^n$ of vectors (or alternatively pairwise between two lists), with the goal of finding those pairs $(\mathbf{v}_i, \mathbf{v}_j)$ that are relatively close. The cost of such an inner product computation is $2n$ floating point operations, or n so-called Fused multiply-add (FMA) instructions on say 16 or 32-bit floats. Such floating point operations are complex and require many logical gates to function.

The SimHash filter, better known as the 'XOR popcount trick' [Cha02; Fit+14; Duc18; Alb+19; AGPS20], is a cheaper computation, that already throws out most of the candidates that are far away from each-other, so that the full inner product only has to be computed on the more likely candidates. The idea is that given two normalized close pairs $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{n-1}$ (with e.g., $\langle \mathbf{v}, \mathbf{w} \rangle \geq \frac{1}{2}$), and a random direction $\mathbf{y} \sim \mathcal{U}(\mathcal{S}^{n-1})$, the inner products $\langle \mathbf{v}, \mathbf{y} \rangle$ and $\langle \mathbf{w}, \mathbf{y} \rangle$ are correlated, in particular their signs are more likely to coincide. For a list of random (but fixed) directions $\mathbf{y}_1, \ldots, \mathbf{y}_h \in \mathcal{S}^{n-1}$ we can thus define the SimHash $\mathrm{SH} : \mathcal{S}^{n-1} \to \{0,1\}^h$ by

$$\mathrm{SH} : \mathbf{v} \mapsto \left(\begin{cases} 0, & \text{if } \langle \mathbf{v}, \mathbf{y}_i \rangle \leq 0, \\ 1, & \text{if } \langle \mathbf{v}, \mathbf{y}_i \rangle > 0. \end{cases} \right)_i .$$

If $\mathbf{v}, \mathbf{w}$ are uniformly random, then $\mathrm{SH}(\mathbf{v})$ and $\mathrm{SH}(\mathbf{w})$ have on expectation $h/2$ bits in common. Now if $\mathbf{v}$ and $\mathbf{w}$ or $\mathbf{v}$ and $-\mathbf{w}$ are close, then $\mathrm{SH}(\mathbf{v})$ and $\mathrm{SH}(\mathbf{w})$ are expected to have significantly more or significantly less than $h/2$ bits in common respectively. Assuming that the SimHashes are pre-computed (which takes negligible time when

amortized over many pairs), we can compute the number of bits (not) in common by doing an h-bit XOR, followed by a population count instruction (which counts the number of ones), which can be much cheaper than the $2n$ floating point operations. The number of directions h and the filter threshold can be balanced for filter strength and computation cost, for theory on this matter look at [AGPS20]. In practice the SimHash with $h = 256$ has been successfully used up to dimension 128, saving about a factor 10 on computation time [Fit+14; Alb+19].

3.4.2 Progressive sieving

Due to the exponential time complexity sieving quickly becomes more costly in higher dimensions, especially when the starting list also consists of very long vectors. Relatively, the cost of sieving in slightly lower dimensions becomes almost negligible. The idea of Progressive Sieving is to first sieve in a lower dimensional projected sublattice, and to slowly increment the dimension and list size, while also lifting the list along the way. As a result, the list vectors are already quite short when the highest dimension is reached, and thus we have to run less high dimensional sieving iterations, at the cost of (more) low dimensional sieving iterations.

Recall that for a lattice $\mathcal{L}$ with basis $\mathbf{B} = (\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ we denote the projected sublattice $\mathcal{L}(\pi_l(\mathbf{b}_l), \ldots, \pi_l(\mathbf{b}_{r-1}))$ by $\mathcal{L}_{[l:r)}$. There are two main directions to implement this idea (although a combination is also possible) [Duc18; LM18; Alb+19]. Either we form a chain of sublattices

$$\mathcal{L}_{[0:r)} \subsetneq \mathcal{L}_{[0:r+1)} \subsetneq \ldots \subsetneq \mathcal{L}_{[0:d)} = \mathcal{L},$$

for some $1 \leq r < d$ or a chain of projections

$$\mathcal{L}_{[l:d)} \xleftarrow{\pi_l} \mathcal{L}_{[l-1:d)} \xleftarrow{\pi_{l-1}} \cdots \xleftarrow{\pi_1} \mathcal{L}_{[0:d)} = \mathcal{L},$$

for some $1 \leq l < d$. Both ways give significant speed-ups compared to directly sieving in the full dimension, but they both come with their own pros and cons.

For the sublattice approach, after extending, we do not have to lift the short vectors we have in any way, they are by definition part of the super-lattice. With a normal descending basis profile we have

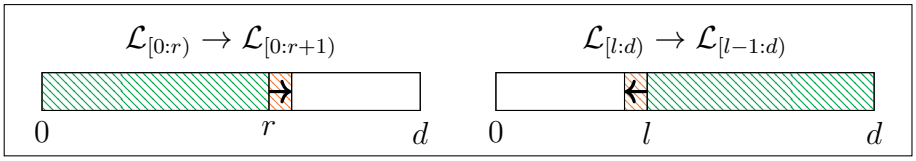


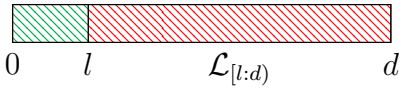
Figure 3.3: Progressive step via a chain of sublattices (on the left) versus a chain of projections (on the right).

$\mathrm{gh}(\mathcal{L}_{[0:r)}) > \mathrm{gh}(\mathcal{L}_{[0:r+1)})$, so after extending the vectors are somewhat longer relative to the local Gaussian Heuristic, but only by a bit. The main disadvantage of the sublattice approach is that after extending, all the short vectors still lie in the same lower rank sublattice, and one has to be careful with sampling new vectors, and in the further sieving process, to also obtain short vectors in $\mathcal{L}_{[0:r+1)} \setminus \mathcal{L}_{[0:r)}$. Without taking explicit care one might implicitly get stuck in a lower rank sublattice, which makes the sublattice approach tricky to work with.

For the projection approach, after extending, the vectors have to be lifted to undo the projection. Undoing e.g., a single projection π_i efficiently, while limiting the increase in size, can be done using Babai's nearest plane algorithm. The main disadvantage is that the squared length of a vector can still increase by $\frac{1}{4}\|\tilde{\mathbf{b}}_i\|^2$, which can be significant. At the same time the Gaussian Heuristic predicts that the first minimum of the extended lattice is larger and thus the relative increment in length is somewhat damped by this. The current state-of-the-art sieving implementations use this method, because it is reliable and combines well with the Dimensions for Free technique.

3.4.3 Dimensions for free

When a lattice sieving algorithm finishes it does not only recover the shortest vector, but a significant proportion of all lattice vectors up to some radius R. For a pair-sieve with a list size of $N = (4/3)^{n/2+o(n)}$ we naturally have this for $R \approx \sqrt{4/3} + o(1)$. The idea of the 'dimensions for free' technique [Duc18], is to sieve in a projected sublattice $\mathcal{L}_{[l:d)}$ for some $l > 0$, and to lift all the short vectors to the full context. If l is not too large, then with high probability one of them lifts to a shortest vector of the full lattice.



To simplify the analysis we simply assume that the sieve in the projected sublattice $\mathcal{L}_{[l:d)}$ computes a list L of all vectors of length up to $\sqrt{4/3}\cdot\mathrm{gh}(\mathcal{L}_{[l:d)})$. The projection $\pi_l(\mathbf{v})$ of a shortest vector $\mathbf{v} \in \mathcal{L}_{[0:d)}$ is thus part of the list $\pi_l(\mathbf{v}) \in L$ if $\|\pi_l(\mathbf{v})\| \leq \sqrt{4/3}\cdot\mathrm{gh}(\mathcal{L}_{[l:d)})$. For not too large l, Babai's algorithm then successfully recovers $\mathbf{v}$ back from $\pi_l(\mathbf{v})$.

The maximum value for l for which the above is true depends on the basis profile $(\|\tilde{\mathbf{b}}_0\|, \ldots, \|\tilde{\mathbf{b}}_{d-1|}\|)$. For a well-reduced basis (more on this in Part III) we obtain about $l = O(\log(d)/d)$ 'dimensions for free', and thus we only have to sieve in a lattice of dimension $n = d - O(\log(d)/d)$, leading to a subexponential speed-up.

Note that with the progressive sieve based on a chain of projections we do not have to pick the parameter l a priori. We can increase the chain step by step (decreasing l), while at every step we lift the list to the full context. At every step we are increasing the probability that the projection $\pi_l(\mathbf{v})$ is part of the list, and we recover it at the appropriate l without having to specify it beforehand.

Looking from another perspective, every short vector in a projected sublattice $\mathcal{L}_{[l:d)}$ has some probability to lift to a (close to) shortest vector of the full lattice. Therefore to increase the the dimensions for free we could lift any short enough vector that we see on the fly: for example any new vector inserted in the sieving list, or even vectors that are short, but not short enough to be inserted.

We can expect about $l = 20$ (for $d = 100$) to $l = 40$ (for $d = 450$) dimensions for free, and even more when enabling on the fly lifting, making this an extremely important technique that we explicitly have to take into account for concrete hardness estimates.

3.5 The General Sieve Kernel

The General Sieve Kernel (G6K) [Alb+19] is a lattice reduction framework based on sieving algorithms that is designed to be 'stateful' instead of treating sieving as a black-box SVP oracle. In short it allows to easily move between different contexts $\mathcal{L}_{[l:r)}$, while maintaining a

database L of short vectors. This encompasses advanced techniques like progressive sieving and dimensions for free. It includes an open-source implementation of the framework that broke several new TU Darmstadt SVP Challenges [SG10] up to dimension 155. This implementation is multi-threaded and low-level optimized, and includes many of the implementation tricks from the advanced lattice sieving literature and some more. In this section we recall the state, instructions, global strategies and some implementation details of G6K.

3.5.1 State

Naturally, the state includes a lattice basis $\mathbf{B} \in \mathbb{Z}^{d \times d}$ and its corresponding Gram-Schmidt basis $\tilde{\mathbf{B}}$. The current state keeps track of a *sieving context* $\mathcal{L}_{[l:r)}$ and a *lifting context* $\mathcal{L}_{[\kappa:r)}$ for $0 \leq \kappa \leq l \leq r \leq d$. The sieving dimension will often be denoted by $n := r - l$. There is a database L containing N lattice vectors from the sieving context $\mathcal{L}_{[l:r)}$. To conclude G6K also keeps track of good insertion candidates $\mathbf{i}_\kappa, \ldots, \mathbf{i}_l$ for the corresponding positions in the current lattice basis.

3.5.2 Instructions

We begin with several instructions that change the sieving context, and we explain how the database is updated such that it does not get invalidated.

- Extend Left: The extend left operation moves the sieving context from $\mathcal{L}_{[l:r)}$ to $\mathcal{L}_{[l-k:r)}$ for some $0 < k \leq l$. The database vectors are lifted from $\mathcal{L}_{[l:r)}$ to $\mathcal{L}_{[l-k:r)}$ using Babai's nearest plane algorithm in the context $\mathcal{L}_{[l-k:l)}$.

- Shrink Left: The shrink left operation moves the sieving context from $\mathcal{L}_{[l:r)}$ to $\mathcal{L}_{[l+k:r)}$ for some $0 < k \leq r - l$. Each database vector $\mathbf{v} \in L \subset \mathcal{L}_{[l:r)}$ is projected to $\pi_{l+k}(\mathbf{v}) \in \mathcal{L}_{[l+k:r)}$, and duplicates are removed.

- Extend Right: The extend right operation moves the sieving context from $\mathcal{L}_{[l:r)}$ to the super-lattice $\mathcal{L}_{[l:r+k)}$ for some $0 < k \leq d - r$. The database can remain unchanged.

- INSERTION: The insertion operation inserts one of the insertion candidates $\mathbf{i}_k$ (often a short vector) back into the basis, at some position $\kappa \leq k \leq l$. I.e., assuming the vector $\mathbf{i}_k$ is primitive we choose some local (basis) transformation on the context $\mathcal{L}_{[k:r)}$ such that $\mathbf{i}_k$ becomes the new basis vector at position k, and the Gram-Schmidt basis $\tilde{\mathbf{B}}$ is updated accordingly. If $k = l$ the database vectors still live in the context $\mathcal{L}_{[l:r)}$ after the basis change, and so we do not have to do anything. Typically however, we will have $k < l$, which makes it less trivial. By carefully choosing the local transformation and by moving to a slightly smaller sieving context $\mathcal{L}_{[l+1:r)}$ we can however still recycle most of the database after an insertion. For prevent confusion we assume that after an insertion operation the sieving context is always moved to $\mathcal{L}_{[l+1:r)}$.

These instruction were focused on moving between different contexts and maintaining the database along the way. We now consider some instructions that act on the database.

- SIEVE: The sieve operation applies a lattice sieving algorithm to the database vectors in order to decrease their size. The end result is an updated database that is saturated in the sieving context $\mathcal{L}_{[l:r)}$ according to some saturation condition. During the sieving process any short enough vector is lifted to the lifting context $\mathcal{L}_{[\kappa:r)}$, and, if short enough, replaces one of the insertion candidates $\mathbf{i}_\kappa, \ldots, \mathbf{i}_l$.

- GROW: The grow operation increases the size of the database by sampling new vectors. This might be needed to have enough vectors to run a sieving algorithm, for example after increasing the dimension of the sieving context. The sampling procedure is not fixed, but preferably generates shortish vectors (by using the known some-what reduced basis, or short vectors in the current database).

- SHRINK: The shrink operation decreases the size of the database by throwing away the longest vectors.

3.5.3 Global strategies

The implementation of G6K consists of a high level `Python` layer and a low-level `C++` layer. The earlier mentioned instructions can be called and parametrized from the `Python` layer, while the core implementation consists of highly optimized `C++` code. This allows one to quickly experiment with different global strategies.

For example, the leftwards progressive sieving technique explained in Section 3.4.2, can be described as follows: start in a small context of say $\mathcal{L}_{[d-40:d)}$ and alternate the Extend Left, Grow and Sieve instructions. The authors of G6K call such an alternation of extending and sieving a *pump up*, up to some final context $\mathcal{L}_{[l:d)}$, and recall that it is much more efficient than running a single Grow and Sieve instruction in the final context $\mathcal{L}_{[l:d)}$. A full *pump* consists of a pump up followed by a *pump down*: repeat the Insertion instruction to improve the basis while making the context smaller again, and optionally combine this with the Sieve instruction to find better insertion candidates. To solve SVP-instances among other things G6K combines such pumps in a *workout*, which is a sequence of longer and longer pumps, until a short enough vector is found in the full context by lifting. Each pump improves the quality of the basis, which as a result lowers the expected length increase from lifting, making consequent pumps faster and simultaneously improving the probability to find a short vector in the full context.

3.5.4 Sieve implementations

The current open-source implementation of G6K contains multiple sieving algorithms that implement the Sieve instruction. There are single-threaded implementations of the Nguyen–Vidick (`nv`) [NV08] and Gauss sieve (`gauss`) [MV10], mostly for low dimensions and testing purposes. Furthermore G6K includes a fully multi-threaded and low-level optimized version of the Becker–Gama–Joux (BGJ) sieve with a single bucketing layer (`bgj1`) [BGJ15]. The filtering techniques from `bgj1` were also extended and used in a triple sieve implementation (`hk3`) [BLS16; HK17]. This implementation considers both pairs and triples and its behaviour automatically adjusts based on the database size, allowing for a continuous time-memory trade-off between the (pair) sieve `bgj1` and a full triple sieve with minimal

memory. Note that the asymptotically best sieve algorithm, which we will refer to as BDGL [BDGL16], has been implemented before [MLB17], but not inside of G6K. In Chapter 4 we discuss an optimized implementation of the BDGL sieve inside G6K, which has since been merged into the open-source implementation of G6K.

3.5.5 Data representation

Given that lattice sieving uses an exponential number of vectors, it is of practical importance how much data is stored per vector in the database. G6K stores for each lattice vector $\mathbf{v} = \mathbf{Bx} \in \mathbb{R}^n$ the (16-bit integer) coordinates $\mathbf{x} \in \mathbb{Z}^n$ as well as the (32-bit floating-point) Gram-Schmidt representation $\mathbf{y} = (\langle \mathbf{v}, \tilde{\mathbf{b}}_i \rangle / \|\tilde{\mathbf{b}}_i\|)_i \in \mathbb{R}^n$ normalized by the Gaussian Heuristic of the current sieving context. The latter representation is used to quickly compute inner products between any two lattice vectors in the database, and to change between contexts. On top of that other preprocessed information is stored for each vector, like the corresponding lift target $\mathbf{t}$ in $\mathrm{span}(\mathcal{L}_{[\kappa:l)})$, the squared length $\|\mathbf{v}\|^2$, a 256-bit SimHash (see Section 3.4.1) and a 64-bit hash as identifier. In order to sort the database on length, without having to move the entries around, there is also a lightweight database that only stores for each vector the length, a SimHash and the corresponding database index. A hash table keeps track of all hash identifiers, which are derived from the $\mathbf{x}$-coordinates, in order to quickly check for duplicates. All of this adds up to a total of $\approx 2^{10}$ bytes per vector in a sieving dimension of $n = 128$.

CHAPTER 4

Advanced Lattice Sieving on GPUs, with Tensor Cores

This chapter is an extended version of the joint work 'Advanced lattice sieving on GPUs, with tensor cores', with Léo Ducas and Marc Stevens, published at Eurocrypt 2021.

4.1 Introduction

Lattice sieving algorithms [AKS01; NV08; MV10; HPS11a] are asymptotically superior to enumeration techniques [FP85; Kan83; SE94; GNR10], but this has only recently been shown in practice. Progress on sieving, both on its theoretical [Laa15; BGJ15; BDGL16; HKL18] and practical performances [Fit+14; Duc18; LM18; Alb+19], brought the cross-over point with enumeration as low as dimension 80. Practical performance is often measured by solving TU Darmstadt SVP Challenges, given a random challenge lattice, one has to find a lattice vector of length $1.05 \cdot \mathrm{gh}(\mathcal{L})$, i.e. a factor 1.05 above the expected length of the shortest vector. The work of M. R. Albrecht et al. at Eurocrypt 2019 [Alb+19], named the General Sieve Kernel (G6K), set new TU

Darmstadt SVP-records [SG10] on a single machine up to dimension 155, while before the highest record was at 152 using enumeration techniques on a large cluster with multiple orders of magnitude more computational resources. They achieved this with an asymptotically non-optimal sieving algorithm based on [BGJ15; HK17], which with sufficient memory runs in time $2^{0.349n+o(n)}$.

While it is now clear that, in terms of time, sieving beats enumeration not only asymptotically, but also in practice, there are still two big open questions in the development of practical lattice sieving algorithms.

Firstly, can the asymptotically optimal sieve [BDGL16], running in time $2^{0.292n+o(n)}$, be made practical and improve on the current record-holding sieve in feasible dimensions? While concrete parameters for lattice-based cryptography schemes are based on the asymptotic complexity of this sieve, there has to date been no practically (or even asymptotically) efficient implementation of it. The asymptotic analysis of the algorithm hides some potentially large polynomial and subexponential factors, which are poorly understood from a concrete perspective. Additionally, a naive implementation leads to subexponential time, or even exponential space overhead factors, that make the algorithm uncompetitive [MLB17] in feasible dimensions. Another interesting question is how the properties of this sieve interact with parallelization efforts that are necessary for large scale attacks.

This brings us to the second open question: can these advanced sieving algorithms scale efficiently beyond a shared-memory architecture, e.g., to a large cluster of computers? The large number of lattice vectors and their interactions makes lattice sieving algorithms non-trivial to parallelize. Before scaling up to a cluster of computers, a natural intermediate step is to port cryptanalytic algorithms to Graphical Processing Units (GPUs); not only are GPUs far more efficient for certain parallel tasks, but their bandwidth and computation capacity ratio are already more representative of the difficulties to expect when scaling up beyond a single computational server. This step can therefore already teach us a great deal about how a cryptanalytic algorithm should scale in practice.

4.1.1 Related work

We give here a short survey of past efforts on the parallelization of lattice sieving algorithms.

In 2011, Milde and Schneider [MS11] gave the first parallel implementation of the Gauss Sieve. The global list is distributed over several independent Gauss Sieve instances that are connected in a cycle. A vector is passed along and reduced with each local list and vice versa, before being inserted. As a result the vectors in the distributed list are nearly pairwise reduced; nearly because the lists could have changed in the meantime. This method scales almost linearly up to about 5 threads, but the efficiency quickly degrades above that due to the many non pairwise reduced vectors in the distributed list.

This problem was tackled by Ishiguro et al. [IKMT14] in 2014: by storing a full copy of the list in shared-memory on each node, they make sure that all vectors in the distributed list are pairwise reduced. The resulting implementation scales well to hundreds of threads and, by also abusing the ideal structure, was used to solve a TU Darmstadt Ideal SVP challenge in dimension 128. However, because each node has to store the full list, the maximum list size, and thus the largest achievable dimension, is limited by the smallest node.

Also in 2014, Mariano et al. [MDB14] implemented a parallel version of the List Sieve algorithm, with super-linear scalability at least up to 32 threads. The claimed super-linear speed-up is possible due to relaxations to the List Sieve properties made by the parallel variant. Around the same time Mariano et al. [MTB14] proposed a fine-grained parallel shared-memory implementation of the Gauss Sieve that achieves almost linear speed-up at least up to 64 cores, and achieves similar run times as Ishiguro et al.

Later that year Bos et al. [BNP17] presented another distributed implementation of the Gauss Sieve similar to the works of Milde et al., and Ishiguro et al., solving the scalability problems of the former in a different way, while maintaining the use of fully distributed lists.

In 2017, both Ishiguro et al.' and Bos et al.' parallel Gauss Sieve implementations were adapted to a (multi-)GPU setting by Yang et al. [YKYC17], solving a TU Darmstadt Ideal SVP challenge in dimension 130 on 8 GPUs in 824 hours.

The first parallel implementations for more advanced and asymp-

totically faster bucketed sieves such as the Hash Sieve [Laa15] and BDGL Sieve [BDGL16] were introduced by Mariano et al. in 2015 [MBL15] and 2017 [MLB17] respectively. Both implementations use a queued model similar to the Gauss Sieve, which in combination with the bucketing techniques leads, both asymptotically and in practice, to a much higher memory usage, making them infeasible for dimensions above 100.

In 2019, a large combined effort of Albrecht, Ducas, Herold, Kirshanova, Postlethwaite, and Stevens [Alb+19] introduced the open-source General Sieve Kernel (G6K), which combined many of the state-of-the art advanced techniques such as Progressive Sieving and Dimensions for Free into a single implementation. G6K assumes a single node shared-memory model, and contains several sieving variants, from simple List and Gauss Sieves to parallel variants of [BGJ15] and [HK17]. The authors also set a new TU Darmstadt SVP challenge record at dimension 155 in about 14 days using 72 CPU cores. For more information about G6K see Section 3.5.

4.1.2 Contributions

The main contribution of this work is to answer the two open questions (partially). We show that lattice sieving, including the more advanced algorithmic improvements, can effectively be accelerated by GPUs. In particular, we show that the NVIDIA Tensor cores, only supporting specific low-precision computations, can be used efficiently for lattice sieving. We exhibit how the most computationally intensive parts of complex sieving algorithms can be executed in low-precision even in large dimensions.

We show and demonstrate by an implementation that the use of Tensor cores results in large efficiency gains for cryptanalytic attacks, both in hardware and energy costs. We present several new computational records, reaching dimension 180 for the TU Darmstadt SVP challenge record with only a single high-end machine with 4 GPUs and 1.5TB RAM in 51.6 days. Not only did we break SVP-records significant faster, but also with $< 4\%$ of the energy cost compared to a CPU only attack. For instance, we solved dimension 176 using less time and with less than 2 times the overall energy cost compared to the previous record of dimension 155. Furthermore by re-computing

data at appropriate points in our algorithms we reduced the memory usage per vector by 60% compared to the base G6K implementation with minimal computational overhead.

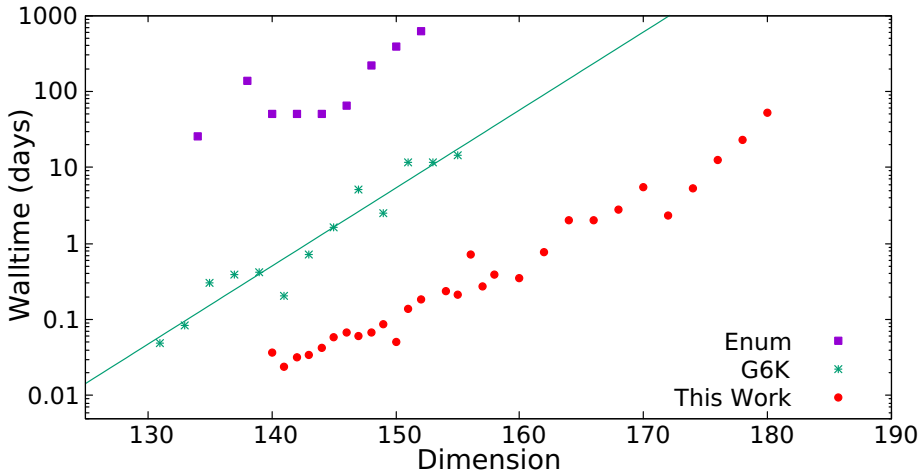


Figure 4.1: Solved TU Darmstadt lattice challenges measured in wall-clock time[1], as reported on the challenge website [SG10]. Details on the used hardware are also reported there. In short, the enumeration records used large clusters of computers, while the sieving records used a single (large) server.

Secondly, we introduce the first practical implementation of the asymptotically best BDGL Sieve from [BDGL16] inside the G6K framework, both for CPU-only (multi-threaded and `AVX2`-optimized) and with GPU acceleration. In contrast to [MLB17] our implementation does not use significant extra memory, and can therefore be used in much higher dimensions. We use this to shed some light on the practicality of this algorithm. In particular we show that our CPU-only BDGL Sieve variant already improves over the previous record-holding sieve in sieving dimensions as low as 90, with a 5× speed-up around dimension 120. Furthermore, we show that this cross-over point lies much higher for our GPU accelerated sieve due to memory-bottleneck constraints.

[1]Comparing wall-clock times between distinct hardware is in general not a good practice, but the different nature of CPUs and GPUs makes it hard to fairly

One key feature of G6K is to also consider lifts of pairs even if such a pair is not necessarily reducible, so as to check whether such lifts are short; the more such pairs are lifted, the more dimensions for free one can hope for [Duc18; Alb+19]. Yet, Babai lifting of a vector has quadratic running time which makes it too expensive to apply to each pair. We introduce a filter based on dual vectors that detects whether pairs are worth lifting. With adequate pre-computation on each vector, filtering a pair for lifting can be made linear-time, fully parallelizable, and very suitable to implement on GPUs.

Open source code

The CPU implementation of the BDGL Sieve has been integrated in G6K, with further improvements, and we aim for long term maintenance.[2]. The GPU implementations has also been made public, but with lower expectation of quality, documentation and maintenance.[3]

4.1.3 Prerequisites

The reader of this chapter is assumed to be familiar with the contents of Chapter 3, in particular lattice sieving, bucketing techniques, Dimensions for Free and G6K.

4.2 GPU architecture and sieve design

In this section we explain the architecture of a GPU, the global design of our new sieve implementations and how we reduce memory usage and movement.

4.2.1 GPU device architecture

We give a short summary of the NVIDIA Turing GPU architecture on which our implementations and experiments are based. Since this

compare them on other metrics. As further motivation the estimated energy usage of our server was only a factor 2 higher than the one used for the G6K records.

[2] https://github.com/fplll/g6k/pull/61

[3] https://github.com/WvanWoerden/G6K-GPU-Tensor

work was done a new generation named Ampere was launched, doubling many of the performance metrics mentioned here.

CUDA cores and memory

An NVIDIA GPU can have up to thousands of so-called *CUDA cores* organized into several execution units called Streaming Multiprocessors (*SM*). These SM use their many CUDA cores (e.g. 64) to service many more resident *threads* (e.g. 1024), in order to hide latencies of computation and memory operations. Threads are bundled per 32 in a *warp*, that follow the single-instruction multiple-data paradigm.

The execution of a GPU program, also called a *kernel*, consists out of multiple *blocks*, each consisting of some warps. Each individual block is executed on any available single SM. The GPU RAM, also called *global memory*, can be accessed by all cores, but is relatively slow. Global memory operations always pass through a GPU-wide L2 cache. In addition, each SM benefits from an individual L1 cache and offers a fast addressable *shared memory* that can only be used by threads in that block. The fastest type of memory are the *registers*, which are local to each thread, and which act as the input and output of computational operations.

To implement GPU kernel functions for an NVIDIA GPU one can use CUDA [NBGS08; NVF20] which is an extension of the `C`/`C++` and `FORTRAN` programming languages. A kernel is executed by a specified number of threads grouped into blocks, all with the same code and input parameters. During execution each thread learns that it is thread t inside block b and one needs to use this information to distribute the work. For example when loading data from global memory we can let thread t read the t-th integer at an offset computed from b, because the requested memory inside each block is contiguous such a memory request can be executed very efficiently; such memory request are known as *coalescing* reads or writes and they are extremely important to obtain an efficient kernel.

Tensor cores

Driven by the machine learning domain there have been tremendous efforts in the past few years to speed up low-precision matrix multiplications. This lead to the so-called Tensor cores, that are now standard

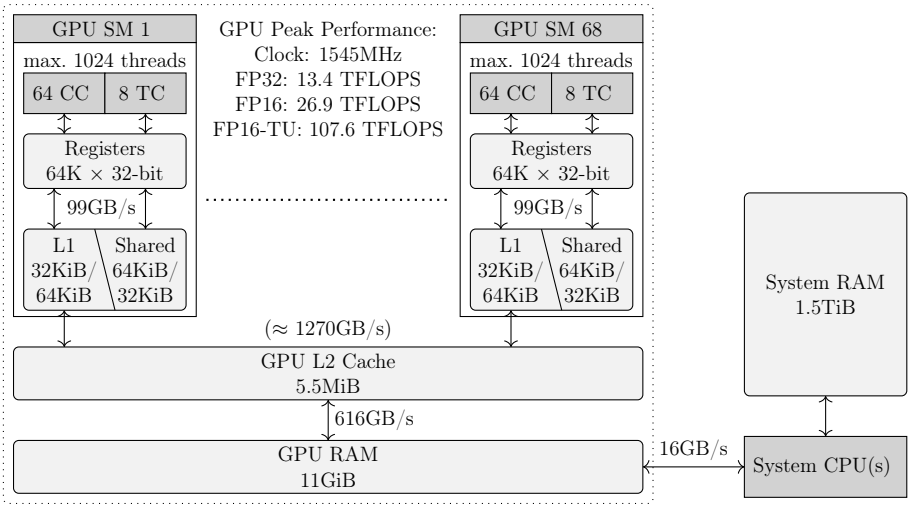


Figure 4.2: Device architecture of the NVIDIA RTX 2080 Ti and the system used in this work.

in high-end NVIDIA GPUs. Tensor cores are optimized for 4×4 matrix multiplication and also allow a trade-off between performance and precision. In particular we are interested in the 16-bit floating point format `fp16` with a 5-bit exponent and a 10-bit mantissa, for which the tensor cores obtain an $8\times$ speed-up over regular 32-bit operations on CUDA cores.

RTX2080 Ti details. In this work we used the NVIDIA RTX2080 Ti with 68 Streaming Multiprocessors, whose architecture is depicted in Figure 4.2. There are 64 cores in each SM, that can service up to 1024 resident threads. There is a large register file of 65536×32-bit registers, where up to 256 registers can be assigned to a thread. Each SM has a dedicated 96KiB memory that is split into a private L1 cache and shared memory (either 32-64 KiB or 64-32 KiB split). The Turing architecture is the 2nd NVIDIA GPU generation to feature special Tensor units, eight per SM.

While a high-end CPU with many cores can reach a performance in the order of a few tera floating point operations per second (TFLOPS), the RTX2080 Ti can achieve 13 TFLOPS for 32-bit floating point operations on its regular CUDA cores. The specialized Tensor cores

enable the RTX2080 Ti to theoretically reach in the order of 107 `fp16`-TFLOPS, improving by two orders of magnitude on a single high-end CPU.

Efficiency

For cryptanalytic purposes it is not only important how many operations are needed to solve a problem instance, but also how cost effective these operations can be executed in hardware. The massively-parallel design of GPUs with many relatively simple cores results in large efficiency gains per FLOP compared to CPU designs with a few rather complex cores; both in initial hardware cost as in power efficiency. Showing that these simple cores can be used effectively also shortens the gap to design low-cost and highly efficient customized hardware for lattice sieving.

As anecdotal evidence we compare the acquisition cost, energy usage and theoretical peak performance of the CPU and GPU in the new server we used for our experiments: the Intel Xeon Gold 6248 launched in 2019 and the NVIDIA RTX2080 Ti launched in 2018 respectively. The CPU had a price of about €2500 and has a TDP of 150 Watt, while the GPU was priced at about €1000 and has a TDP of 260 Watt. For 32-bit floating point operations the theoretical peak performance is given by 3.2 TFLOPS[4] and 13.45 TFLOPS for the CPU and GPU respectively, making the GPU a factor 2.4 better per Watt and 10.5 better per Euro spend on acquisition. For general 16-bit floating point operations these number double for the GPU, while the CPU obtains no extra speed-up (one actually has to convert the data back to 32-bit). When considering the specialized Tensor cores with 16-bit precision the GPU has a theoretical peak performance of 107.6 TFLOPS, improving by a factor 19.4 per Watt and a factor 84 per Euro spend on acquisition compared to the CPU.

4.2.2 Sieve design

The great efficiency of the GPU is only of use if the state-of-the-art algorithms are compatible with the massively-parallel architecture and

[4] With 64 FLOP per core per cycle using two `AVX`-512 FMA units and a maximal clock frequency of 2500MHz when using `AVX`-512 on all 20 cores.

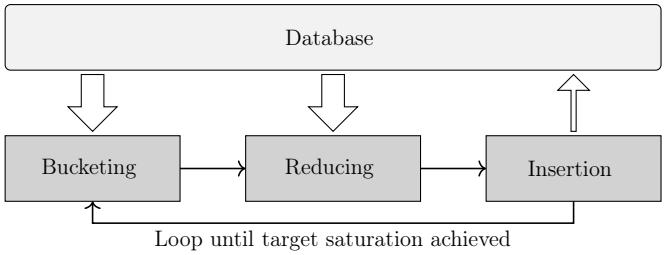


Figure 4.3: High level diagram of the implemented Sieving process.

the specific low-precision operations of the Tensor cores. To show this we extended the lattice sieving implementation of G6K. We will focus our main discussion on the sieving part, as the other G6K instructions are asymptotically irrelevant and relatively straightforward to accelerate on a GPU (which we also did).

All of our CPU multi-threaded and GPU-powered sieve implementations follow a similar design (cf. Fig. 4.3) consisting out of three sequential phases: bucketing, reducing and result insertion. We call the execution of this triplet an *iteration* and these iterations are repeated until the desired saturation is achieved. Note that our sieves are not 'queued' sieves such as the Gauss-Sieve of [MV10] and the previous record setting `bgj1` and `hk3` sieves; this relaxation aligns with the batched nature of GPU processing and allows to implement an asymptotically optimal BDGL-like sieve [BDGL16], without major memory overhead.

Bucketing

During the bucketing phase, the database is subdivided in several buckets $B_1, \ldots, B_m \subset L$, each containing relatively close vectors. How the subdivision is done is precisely where sieving variants differ, and how different (asymptotic) time complexities are achieved. We do not necessarily bucket our full database, as some vectors might be too large to be interesting for the reduction phase in the first few iterations. For each bucket we collect the database indices of the included vectors. For the sieves we consider, these buckets can geometrically be interpreted as spherical caps or cones with for each bucket B_k an explicit or implicit *bucket center* $\mathbf{c}_k \in \mathbb{R}^n$ indicating its direction. For

each included vector $\mathbf{v} \in B_k$, we also store the inner product $\langle \mathbf{c}_k, \mathbf{v} \rangle$ with the bucket center, which is obtained freely from the bucketing process. Note that a vector may be included in several buckets, something which we precisely control by the *multi-bucket* parameter, whose value we will denote by M. The optimal amount of buckets m and the expected number of vectors in a bucket differs for each of our bucketing implementations. In Section 4.3, we further exhibit our different bucketing implementations and compare their performance and quality.

Reducing

During the reduction phase, we try to find all close pairs of lattice vectors inside each bucket, i.e., at distance at most some *length bound* ℓ. Using negation, we orient the vectors inside a bucket into the direction of the bucket center based on the earlier computed inner product $\langle \mathbf{c}_k, \mathbf{v}_i \rangle$. In case the bucketing center $\mathbf{c}_k$ is itself a lattice vector (as can be the case for BGJ-like sieves, but not for BDGL), it is also interesting to check if $\mathbf{c}_k - \mathbf{v}_i - \mathbf{v}_j$ is a short lattice vector, leading to a triple reduction [HK17].

For each bucket B_k, we compute all pairwise inner products $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ for $\mathbf{v}_i, \mathbf{v}_j \in B_k$. Together with the already computed lengths $\|\mathbf{v}_i\|, \|\mathbf{v}_j\|$, $\|\mathbf{c}_k\|$ and inner products $\langle \mathbf{c}_k, \mathbf{v}_i \rangle, \langle \mathbf{c}_k, \mathbf{b}_j \rangle$ we can then efficiently decide if $\mathbf{v}_i - \mathbf{v}_j$ or $\mathbf{c}_k - \mathbf{v}_i - \mathbf{v}_j$ is short. Note that we compute the length of both the pair and the triple essentially from a single inner product computation. We return the indices of pairs and triplets that result in a vector of length at most the length bound ℓ, together with the length of the new vector. In Section 4.4 we further discuss the reduction phase and implementation details of our reduction kernel on the GPU using low-precision Tensor cores.

The number of inner products we have to compute per bucket grows quadraticly in the bucket size $|B_k|$, while the number of buckets only decreases linearly in the bucket size. Therefore, one would in principle want many buckets that are rather small and of high quality, improving the probability that a checked pair actually gives a reduction. For a fixed bucketing algorithm more buckets generally increase the cost of the bucketing phase, while decreasing the cost of the reduction phase due to smaller bucket sizes. We try to balance the cost

of these phases to obtain optimal performance.

Next to finding short vectors in the sieving context we also want to find pairs that lift to short vectors in the larger lifting context. Unfortunately it is too costly to just lift all pairs as this has a cost of at least $\Theta((l-\kappa)^2)$ per pair. In Section 4.5 we introduce a filter based on dual vectors that can be computed efficiently for each pair given a bit of pre-computed data per vector. The few pairs that survive this filter are more likely to lift to a short vector and we only lift those pairs.

Result insertion

After the sieving part we have a list of tuples with indices and the corresponding length of the new vector they represent. The hash identifier of the new vector can efficiently be recomputed by linearity of the hash function and we check for duplicates in our current database. For all non-duplicate vectors we then compute their $\mathbf{x}$-representation. After all new entries are created they are inserted back in the database, replacing entries of greater length.

4.2.3 Data storage and movement

Recall from Section 3.5 that G6K stores quite some data per vector such as the coefficients $\mathbf{x}$ in terms of the basis, a Gram-Schmidt representation $\mathbf{y}$, the lift target $\mathbf{t}$, a SimHash, and more. Theoretically we could remove all data except the $\mathbf{x}$-representation and compute all other information on-the-fly. However, as most of this other information has a cost of $\Theta(n^2)$ to compute from the $\mathbf{x}$-representation this would mean a significant computational overhead, for example increasing the cost of an inner product from $\Theta(n)$ to $\Theta(n^2)$. Also given the limited amount of performance a CPU has compared to a GPU we certainly want to minimize the amount of such overhead for the CPU. By recomputing at some well chosen points on the GPU, our accelerated sieves minimize this overhead, while only storing the $\mathbf{x}$-representation, length and a hash identifier per vector, leading to an approximately 60% reduction in storage compared to the base G6K implementation. As a result we can sieve in significantly larger dimensions with the same amount of system RAM.

While GPUs have an enormous amount of computational power, the memory bandwidth between the database in system RAM and the GPU's RAM is severely limited. These are so imbalanced that one can only reach theoretical peak performance with Tensor cores if every byte that is transferred to the GPU is used in at least 2^{13} computations. A direct result is that reducing in small buckets is (up to some threshold) bandwidth limited. Growing the bucket size in this regime would not increase the wall-clock time of the reduction phase, while one does consider more pairs. So larger buckets are preferred, in our hardware for a single active GPU the threshold seems to be around a bucket size of 2^{14}, matching the 2^{13} computations per byte ratio. Because in our hardware each pair of GPUs share their connection to the CPU, halving the bandwidth for each, the threshold grows to around 2^{15} when using all GPUs simultaneously.

The incidental benefit of large buckets is that the conversion from the $\mathbf{x}$-representation to the $\mathbf{y}$-representation, which can be done directly on the GPU, is negligible compared to computing the many pairwise inner products. To further limit the movement of data we only return indices instead of a full vector; if we find a short pair $\mathbf{v}_i - \mathbf{v}_j$ on the GPU we only return i, j and $\|\mathbf{v}_i - \mathbf{v}_j\|^2$. The new $\mathbf{x}$-representation and hash identifier can efficiently (in $O(n)$) be computed on the CPU directly from the database.

4.3 Bucketing

The difference between different lattice sieve algorithms mainly lies in their bucketing method. These methods differ in their time complexity and their performance in catching close pairs. In this section we exhibit a Tensor-GPU accelerated bucketing implementation `triple_gpu` similar to `bgj1` and `hk3` inspired by [BGJ15; HK17], and two optimized implementations of the asymptotically best known bucketing algorithm [BDGL16], one for CPU making use of `AVX2` (`bdgl`) and one for GPU (`bdgl_gpu`). After this we show the practical performance difference between these bucketing methods.

4.3.1 BGJ-like bucketing (`triple_gpu`)

Recall from Section 3.3 that the bucketing method used in `bgj1` and `hk3` is based on spherical caps, directed by explicit bucket centers that are also lattice points. Inspired by this we start the bucketing phase by first choosing some bucket centers $\mathbf{b}_1, \dots, \mathbf{b}_m$ from the database; preferably the directions of these vectors are somewhat uniformly distributed over the sphere. Then we associate each vector $\mathbf{v} \in L$ in our database to the bucket $B_{k_\mathbf{v}}$ where

$$k_\mathbf{v} = \arg\max_{1 \le k' \le m} \left| \left\langle \frac{\mathbf{b}_{k'}}{\|\mathbf{b}_{k'}\|}, \mathbf{v} \right\rangle \right|.$$

In this we differ from the original versions of `bgj1` and `hk3` [BGJ15; HK17; Alb+19] in that they use a fixed filtering threshold on the angle $|\langle \mathbf{b}_k/\|\mathbf{b}_k\|, \mathbf{v}/\|\mathbf{v}\| \rangle|$. We further relax this condition somewhat by the multi bucket parameter M, to associate a vector to the best M buckets. As a result our buckets do not exactly match spherical caps, but they should still resemble them; in particular such a change does only affect the asymptotic analysis up to subexponential factors. We chose for this alternation as this fixes the amount of buckets associated to each vector, which reduced some communication overhead in our highly parallel GPU implementations.

In each iteration the new bucket centers are chosen, normalized and stored once on each GPU. Then we stream our whole database $\mathbf{v}_1, \dots, \mathbf{v}_N$ through the GPUs and try to return for each vector the indices of the M closest normalized bucket vectors and their corresponding inner products $\langle \mathbf{v}_i, \mathbf{b}_k \rangle$. For efficiency reasons the bucket centers are distributed over 16 threads and each thread stores only the best encountered bucket for each vector. Then we return the buckets from the best $M \le 16$ threads, which are not necessarily the best M buckets overall. The main computational part of computing the pairwise inner products is similar to the Tensor-GPU implementation for reducing, and we refer to Section 4.4.1 for further implementation details.

The cost of bucketing is $O(N \cdot n)$ per bucket. Assuming that the buckets are of similar size $|B_k| \approx M \cdot N/m$ the cost to reduce is $O(\frac{M \cdot N}{m} \cdot n)$ per bucket. To balance these costs for an optimal runtime one should choose $m \sim M \cdot \sqrt{N}$ buckets per iteration. For

the regular 2-sieve strategy with an asymptotic memory usage of $N = (4/3)^{n/2+o(n)} = 2^{0.208n+o(n)}$ this leads to a total complexity of $2^{0.349n+o(n)}$ using as little as $2^{0.037n+o(n)}$ iterations. Note that in low dimensions we might prefer a lower number of buckets to achieve the minimum required bucket size to reach peak efficiency during the reduction phase.

4.3.2 BDGL-like bucketing (`bdgl` and `bdgl_gpu`)

Recall from Section 3.3 that the asymptotically optimal bucketing method from [BDGL16] improves over the `bgj1` and `hk3` sieves by using structured bucket centers. One splits the dimension n into k smaller blocks of similar dimensions $n_1, \ldots, n_k$ that sum up to n, and (after some orthonormal transformation) the set C of bucket centers is a direct product $C = C_1 \times \pm C_2 \cdots \times \pm C_k$ of local bucket centers. For a vector $\mathbf{v}$ we only have to pick the closest local bucket centers to find the closest global bucket center, implicitly considering $m = 2^{k-1} \prod_b |C_b|$ bucket centers at the cost of only $\sum_b |C_b| \approx O(m^{1/k})$ inner products.

To optimize the parameters we again balance the cost of bucketing and reducing. Note that for $k = 1$ we essentially obtain `bgj1` with buckets of size $\tilde{O}(N^{1/2})$ and a time complexity of $2^{0.349n+o(n)}$. For $k = 2$ or $k = 3$ the optimal buckets become smaller of size $\tilde{O}(N^{1/3})$ and $\tilde{O}(N^{1/4})$ respectively and of higher quality, leading to a time complexity of $2^{0.3294n+o(n)}$ and $2^{0.3198n+o(n)}$ respectively. By letting k slowly grow, e.g., $k = \Theta(\log(n))$ there will only be a subexponential $2^{o(n)}$ number of vectors in each bucket, leading to the best known time complexity of $2^{0.292n+o(n)}$.

We will take several liberties with the above strategy to address practical efficiency consideration and fine-tune the algorithm. For example, for a pure CPU implementation we may prefer to make the average bucket size somewhat larger than the $\approx N^{1/(k+1)}$ vectors that the theory prescribes; this will improve cache re-use when searching for reducible pairs inside buckets. In our GPU implementation, we make this average bucket size even larger, to prevent memory bottlenecks in the reduction phase.

Furthermore, we optimize the construction of the local bucket centers $\mathbf{c} \in C_i$ to allow for a fast computation of the local inner products $\langle \mathbf{c}, \mathbf{v} \rangle$. While [BDGL16] choose the local bucket centers C_i uniformly

at random, we apply some extra structure to compute each inner product with a vector $\mathbf{v}$ in time $O(\log(n_i))$ instead of $O(n_i)$. The main idea is to use the (Fast) Hadamard Transform $\mathcal{H}$ on say $32 \leq n_i$ coefficients of $\mathbf{v}$. Note that this computes the inner product between $\mathbf{v}$ and 32 orthogonal ternary vectors, which implicitly form the bucket centers, using only $32\log_2(32)$ additions or subtractions. To obtain more than 32 different buckets we permute and negate coefficients of $\mathbf{v}$ in a pseudo-random way before applying $\mathcal{H}$ again. This strategy can be heavily optimized both for CPU using the vectorized `AVX2` instruction set (`bdgl`) and for GPU by using special warp-wide instructions (`bdgl_gpu`). In particular this allows a CPU core to compute an inner product every 1.3 to 1.6 cycles for $17 \leq n_i \leq 128$. We first explain implementation details specifically targeted for the `AVX2` CPU instruction set, and then we discuss how to similarly implement it for the GPU.

Bucketing on `AVX2` CPU cores

For the sake of this discussion, let us fix the dimension of the local blocks to $\frac{n}{k} = 48$. We represent the input vector $\mathbf{v} \in \mathbb{R}^{48}$ using 16-bits fixed-point precision; so that we can fit $\mathbf{v}$ within 3 `AVX2` registers. We then construct on-the-fly a pseudo-random sequence of $m'' = m'/32$ permutations π_i acting over the 48 coordinates of $\mathbf{v}$. For brevity these permutations also include pseudo-random negations. The permutation are constructed by sequential updates as $\pi_i = \tau_i \circ \pi_{i-1}$.

The pseudo-randomness is obtained simply by iterating *a single round* of `AES-NI` using the seed both for fixed key material and initial value; each round produces 128 pseudo-random bits used for a permutation update (the choice for AES is due to hardware-acceleration). The permutation update τ_i is constructed from this pseudo-random bits by combining `AVX2` bytewise `shuffle` instructions within each `AVX2` registers (chosen pseudo-randomly among a few predefined shuffles), and controlled swaps between pairs of `AVX2` registers (directly constructed from the pseudo-randomness using bitwise operations).

Finally, we also rely on the Fast Hadamard transform $\mathcal{H} : \mathbb{R}^{48} \to \mathbb{R}^{32}$ over the first 32 coordinates of a permuted vector $\mathbf{w} = \pi(\mathbf{v}) \in \mathbb{R}^{48}$. We note that each output of $\mathcal{H}(\pi(\mathbf{v}))$ for a random permutation implicitly corresponds to an inner product $\langle \mathbf{v}, \mathbf{c} \rangle$ for some ternary vector $\mathbf{c} \in \{-1, 0, 1\}^{48}$ of hamming weight 32. Because the output of the

Hadamard transform $\mathcal{H}$ is also in the form of `AVX2` registers, extracting (the index of) $\arg\max_{\mathbf{c}\in S}\langle \mathbf{c}, \mathbf{v}\rangle$ can also be done in a vectorized and on-the-fly manner.

For large m', we can bucket a vector $\mathbf{v}$ in less than $1.6 \cdot m'$ many CPU cycles. Another advantage is that this whole procedure fits within register memory: there is no need to access the vectors $\mathbf{c} \in S$ from memory. This therefore leaves all the memory bandwidth and cache storage to concurrent processes working on other tasks.

The CPU implementation of `bdgl` has been integrated in G6K.[5] As it may be of independent interest, the `AVX2` bucketer is also provided as a stand-alone program.[6]

Bucketing on GPU cores

We also created a GPU accelerated version `bdgl_gpu` based on the ideas from the CPU implementation. We describe some implementation details of the local bucketing step.

Recall that 32 threads are grouped in a warp, following the single-instruction multiple-data paradigm. One could interpret instructions executed by a warp as `AVX2` instructions on very wide (1024 bit) registers. We let each warp process multiple vectors at the same time and each vector $\mathbf{v} = (v_1, \ldots, v_{n/k})$ is distributed over the 32 threads by storing v_i in thread $i \pmod{32} \in \{1, \ldots, 32\}$. Processing multiple vectors at the same time allows to amortize the cost of generating the appropriate randomness and to hide data latencies.

Depending on the dimension we use a Hadamard transform over the first 32 or 64 coefficients, extracting 32 or 64 implicit inner products at a time respectively. Similar to `AVX2` we have the `__shfl_sync` instruction to move data between threads in a warp. For example the `__shfl_xor_sync` instruction exchanges data between all threads i and `XOR` (i, c) for some value c. We used this to implement the Fast Hadamard Transform with a logarithmic number of such exchanges in a straightforward way.

The pseudo-randomness is obtained from the cuRAND library. The total permutation consists out of three parts. First we try to fully permute the coefficients inside the Hadamard region by doing

[5] https://github.com/fplll/g6k/pull/61

[6] https://github.com/lducas/AVX2-BDGL-bucketer

random swaps using the `__shfl_xor_sync` instruction and a coin-flip over several rounds. Note that exchanging threads also need to do a shared coin-flip, but only once per round for all vectors that are processed at the same time. The second step consists out of random sign flips of coefficients inside the Hadamard region. Lastly coefficients from outside the Hadamard region are randomly swapped with coefficients inside the region, this happens locally on each thread.

To prevent many branches and a high overhead each thread only stores the best encountered bucket for each vector it processes. So thread $i \in [1, \ldots, 32]$ stores the best of the buckets $i, i+32, i+64, \ldots$ for each vector. Of these 32 results we then take the best M buckets. This is a performance trade-off and as a result we do not necessarily obtain the best M buckets overall. We will see in Section 4.3.3 that for small values of M this does not influence the bucketing quality that much.

4.3.3 Quality comparison

In this section we compare the practical bucketing quality of the BGJ- and BDGL-like bucketing methods we implemented. More specifically, we consider `triple_gpu`, `1-bdgl_gpu` and `2-bdgl_gpu` where the latter two are instances of `bdgl_gpu` with $k = 1$ and $k = 2$ blocks respectively. Their quality is compared to the idealized theoretical performance of `bgj1` with uniformly distributed bucket centers.[7] For `triple_gpu`, we follow the Gaussian Heuristic and sample bucket centers whose directions are uniformly distributed. As a result the quality difference between `triple_gpu` and the idealized version highlights the quality loss resulting from our implementation decisions. Recall that compared to `bgj1` the main difference is that for every vector we return the M closest bucket centers instead of using a fixed threshold for each bucket. Also these are not exactly the M closest bucket centers, as we first distribute the buckets over 16 threads and only store a single close bucket per thread. For our `bdgl_gpu` implementation the buckets are distributed over 32 threads and we add to this that the bucket centers are not random but somewhat structured by the direct

[7] Volumes of caps and wedges for predicting the idealized behavior where extracted from [AGPS20], and more specifically https://github.com/jschanck/eprint-2019-1161/blob/main/probabilities.py.

product and Hadamard construction. In particular the product construction has an inherent subexponential quality loss, which is further analysed in [Duc22].

To compare the geometric quality of bucketing implementations, we measure how uniform vectors are distributed over the buckets and how many close pairs end up in at least one common bucket. The first measure is important as the reduction cost does not depend on the square of the average bucket size $\left(\frac{1}{m}\sum_{k=1}^{m}|B_k|\right)^2$, which is fixed, but on the average of the squared bucket size $\frac{1}{m}\sum_{k=1}^{m}|B_k|^2$, which is only minimal if the vectors are equally distributed over the buckets. For all our experiments we observed at most an overhead of 0.2% compared to perfectly equal bucket sizes and thus we will further ignore this part of the quality assessment. To measure the second part efficiently we sample 2^{20} close unit pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}^n \times \mathcal{S}^n$ uniformly at random such that $\langle \mathbf{x}, \mathbf{y} \rangle = \pm\frac{1}{2}$. Then we count the number of pairs that have at least 1 bucket in common, possibly over multiple iterations. We run these experiments with parameters that are representative for practical runs. In particular we consider (sieving) dimensions up to $n = 144$ and a database size of $N = 3.2 \cdot 2^{0.2075n}$ to compute the number of buckets given the desired average bucket size and the multi-bucket parameter M. Note that we specifically consider the geometric quality of these bucketing implementations for equivalent parameters and not the cost of the bucketing itself.

To compare the bucketing quality between the different methods and the idealized case we first consider the experimental results in graphs **a**. and **b**. of Figure 4.4. Note that the bucketing methods `triple_gpu` and `1-bdgl_gpu` obtain extremely similar results overall, showing that the structured Hadamard construction is competitive with fully random bucket centers. We see a slight degradation of 5% to 20% for `triple_gpu` with respect to the idealized case as a result of not using a fixed threshold. We do however see this gap decreasing when M grows to 4 or 8, indicating that these two methods of assigning the buckets become more similar for a larger multi-bucket parameter. At $M = 16$ we see a sudden degradation for `triple_gpu` which exactly coincides with the fact that the buckets are distributed over 16 threads and we only store the closest bucket per thread. The quality loss of `2-bdgl_gpu` seems to be between 15% and 36% in the relevant dimensions, which is quite significant but reasonable given a

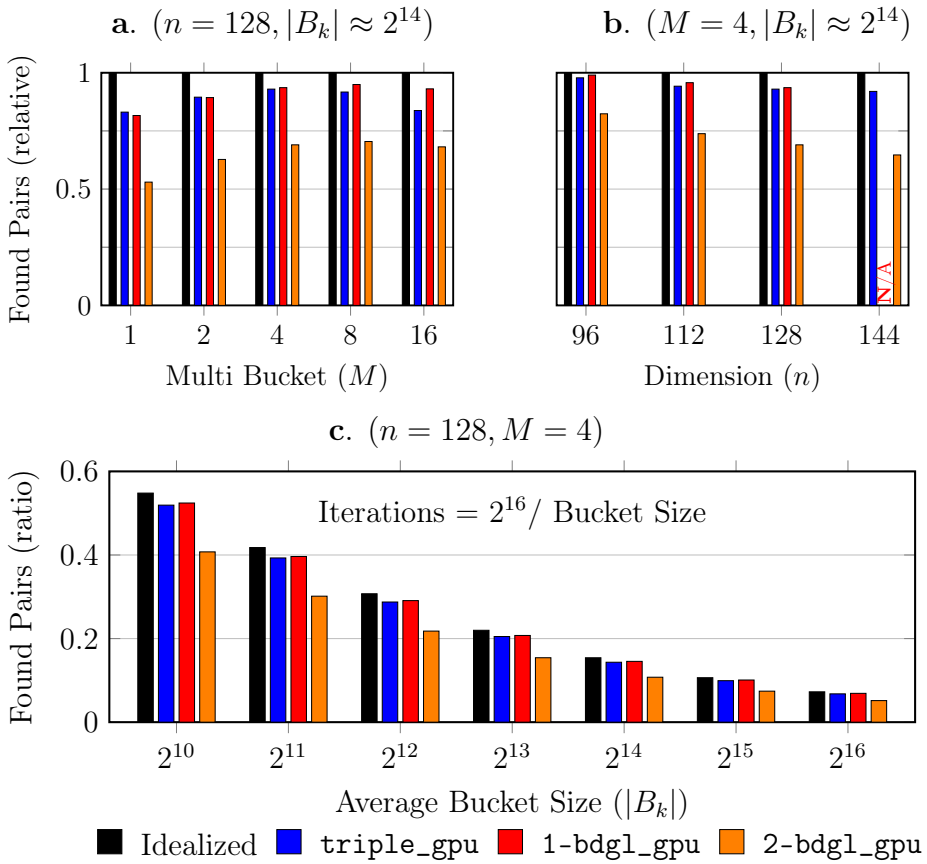


Figure 4.4: Bucketing Quality Comparison. We sampled 2^{20} pairs $\mathbf{v}, \mathbf{w}$ of unit vectors such that $|\langle \mathbf{v}, \mathbf{w} \rangle| = 0.5$ and we measured how many fell into at least 1 common bucket. The number of buckets is computed based on the desired average bucket size $|B_k|$, the multi-bucket parameter M, and a representative database size of $N = 3.2 \cdot 2^{0.2075n}$. The found pairs in **a.** and **b.** are normalized w.r.t. idealized theoretical performance of `bgj1` (perfectly random spherical caps). For **c.** the number of applied iterations is varied such that the total reduction cost is fixed.

loss potentially as large as subexponential [BDGL16, Theorem 5.1].

Now we focus our attention on graph **c.** of Figure 4.4 to consider the influence of the average bucket size on the quality. We observe that increasing the average bucket size reduces the bucketing quality;

many small buckets have a better quality than a few large ones. This is unsurprising as the asymptotically optimal BDGL sieve aims for high quality buckets of small size. Although our `k-bdgl_gpu` bucketing method has no problem with efficiently generating many small buckets, the reduction phase cannot efficiently process small buckets due to memory bottlenecks. This is the main trade-off of (our implementation of) GPU acceleration, requiring a bucket size of 2^{15} versus e.g., 2^{10} leads to a potential loss factor of 7 to 8 as shown by this graph. For `triple_gpu` this gives no major problems as for the relevant dimensions $n \geq 130$ the optimal bucket sizes are large enough. However `2-bdgl_gpu` should become faster than `bgj1` exactly by considering many smaller buckets of size $N^{1/3}$ instead of $N^{1/2}$, and a minimum bucket size of 2^{15} shifts the practical cross-over point above dimension 130, and potentially much higher.

4.4 Reducing

Together with bucketing, the most computationally intensive part of sieving algorithms is that of finding reducing pairs or triples inside a bucket. We consider a bucket of s vectors $\mathbf{v}_1, \ldots, \mathbf{v}_s \in \mathbb{R}^n$ with bucket center $\mathbf{c}$. Only the $\mathbf{x}$-representations are sent to the GPU and there they are converted to the 16-bit Gram-Schmidt representations $\mathbf{y}_1, \ldots, \mathbf{y}_s$ and $\mathbf{y_c}$ that are necessary to quickly compute inner products. Together with the pre-computed squared lengths $\|\mathbf{y}_1\|^2$, $\ldots$, $\|\mathbf{y}_s\|^2$ and inner products $\langle \mathbf{y_c}, \mathbf{y}_1 \rangle, \ldots, \langle \mathbf{y_c}, \mathbf{y}_s \rangle$, the goal is to find all pairs $\mathbf{y}_i - \mathbf{y}_j$ or triples $\mathbf{y_c} - \mathbf{y}_i - \mathbf{y}_j$ of length at most some bound ℓ. A simple derivation shows that this is the case if and only if

for **pairs:**

$$\langle \mathbf{y}_i, \mathbf{y}_j \rangle \geq \frac{\|\mathbf{y}_i\|^2 + \|\mathbf{y}_j\|^2 - \ell^2}{2}, \text{ or}$$

for **triples:**

$$\langle \mathbf{y}_i, \mathbf{y}_j \rangle \leq -\frac{\|\mathbf{y_c}\|^2 + \|\mathbf{y}_i\|^2 + \|\mathbf{y}_j\|^2 - \ell^2 - 2\langle \mathbf{y_c}, \mathbf{y}_i \rangle - 2\langle \mathbf{y_c}, \mathbf{y}_j \rangle}{2}.$$

And thus we need to compute all pairwise inner products $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$. If we consider the matrix $\mathbf{Y} := [\mathbf{y}_1, \ldots, \mathbf{y}_s] \in \mathbb{R}^{n \times s}$ then computing all

pairwise inner products is essentially the same as computing one half of the matrix product $\mathbf{Y}^t\mathbf{Y}$.

Many decades have been spend optimizing (parallel) matrix multiplication for CPUs, and this has also been a prime optimization target for GPUs. As a result we now have heavily parallelized and low-level optimized BLAS (Basic Linear Algebra Subprograms) libraries for matrix multiplication (among other things). For NVIDIA GPUs close to optimal performance can often be obtained using the proprietary cuBLAS library [NVI], or the open-source, but slightly less optimal CUTLASS library [Ker+22]. Nevertheless the BLAS functionality is not perfectly adapted to our goal. Computing and storing the matrix $\mathbf{Y}^t\mathbf{Y}$ would require multiple gigabytes of space. Streaming the result $\mathbf{Y}^t\mathbf{Y}$ to global memory takes more time than the computation itself. Indeed computing $\mathbf{Y}^t\mathbf{Y}$ using cuBLAS does not exceed 47 TFLOPS for $n \leq 160$, and this will be even lower when also filtering the results.

For high performance, in our implementation we combined the matrix multiplication with result filtering. We made sure to only return the few indices of pairs that give an actual reduction to global memory; filtering the results locally while the computed inner products are still in registers. Nevertheless the data-movement design, e.g., how we efficiently stream the vectors $\mathbf{y}_i$ into the registers of the SMs, is heavily inspired by CUTLASS and cuBLAS. To maximize memory read throughput, we had to go around the dedicated CUDA tensor API and reverse engineer the internal representation to obtain double the read throughput.

4.4.1 Reduction kernel

In this section we discuss some implementation details of our Tensor-GPU kernel to find reductions. For bucketing methods as `triple_gpu` the implementation is similar. For performance results see Figure 4.6. We consider a bucket of s vectors $\mathbf{Y} := [\mathbf{y}_1, \ldots, \mathbf{y}_s] \in \mathbb{R}^{n \times s}$ and we need to filter pairs based on their inner product. Note that we essentially want to compute one half of the matrix $\mathbf{Y}^t\mathbf{Y}$.

Fragments

When working with Tensor cores a fundamental building block is a fragment: a 16×16 `fp16` matrix that is distributed over a warp, each thread storing 8 of the 256 values. CUDA allows a matrix multiply and accumulate operation $\mathbf{C} = \mathbf{C} + \mathbf{AB}$ with fragments $\mathbf{A}, \mathbf{B}, \mathbf{C}$ accelerated by the Tensor cores.

Note that the accumulation fragments that contain the resulting inner products are distributed over the threads in a black-box manner which is not specified by NVIDIA; each thread knows some inner product values but not to which pairs they belong. The official solution is to first store the results back to shared memory using a special CUDA instruction, but this severely degrades performance. We reverse engineered the distribution so we can immediately process the inner products while they are still stored in registers, something which might break in future hardware or CUDA versions. Additionally we also used this to improve the loading of fragments from global memory. Before starting the expensive reduction kernel in which every SM loads fragments of $\mathbf{Y}$ from global memory we first reorder the storage of $\mathbf{Y}$ such that all 8 `fp16` coefficients that are stored on a single thread are stored in a consecutive 128 bits range, allowing to load it with a single coalescing instruction directly into the correct local registers, instead of 4 different non-coalescing 32-bit load instructions without this reordering.

Data movement

Optimizing a GPU kernel is all about data movement. Given the memory hierarchy (see Figure 4.2) with different capacities, bandwidths and latencies the main challenge is to get the data into the registers in time to run the appropriate computations. By reusing data locally as much as possible we can improve the ratio of computations versus memory that is moved.

Each block computes a row of $\mathbf{Y}^t\mathbf{Y}$ of 128 vectors wide, using 8 warps, i.e., 256 threads in total. As a result each coefficient that is loaded from $\mathbf{Y}$ is used for 128 multiply and add operations, enough to prevent a significant memory bottleneck between global memory and each SM. Inside each row we process a matrix block of 128×256 at a time, where each warp takes care of $4 \times 4 = 16$ fragments in a 64×64

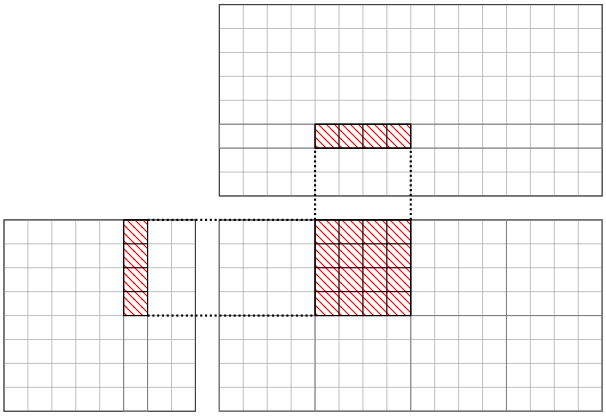


Figure 4.5: Example of the computation on a 64×64 sub-block done by a single warp. Each cell is a 16×16 matrix fragment. The full 128×256 block is processed by 8 warps in parallel.

matrix sub-block (see Figure 4.5). After processing a matrix block we shift to the next one, until we exceed the diagonal; we only want to compute half of the symmetric result matrix.

When loading a row of fragments of $\mathbf{Y}$ from global memory all 8 warps in a block work together to only fetch each element once, the values are temporarily stored in what we call *cache registers*. From here these elements are stored in shared memory after which all warps are synced such that we can be sure the shared memory contains the correct elements. Then each warp separately loads the 4 vertical and 4 horizontal fragments that it needs to compute its 64×64 matrix sub-block from shared memory to what we call the *compute registers*.

Loading data from shared and global memory involves significant latencies from tens to hundreds of cycles. During this time we need to make sure our kernel is still computing with data that is already in registers. For example while data is being loaded from global memory we can do computations with the data in our shared memory (with a much lower latency). A well know technique to further hide latencies is double buffering in which we double the amount of registers and shared memory we mentioned before. While one half of the compute registers is used for computation the other half is obtaining new data from the shared memory; continuously alternating their roles to hide the shared memory latency. Note that all these techniques are limited by

the amount of registers and shared memory we have, so they require a careful balancing. The double buffering technique is also used between the CPU and GPU to simultaneously transfer new bucket vectors and old results during kernel executions.

Processing the results

While computing the inner products is the computational intensive part of the reduction kernel filtering out the results also involves some overhead. Especially since this has to happen using regular CUDA cores, which are relatively slow compared to Tensor cores. Also filtering results involves branches, something which can heavily degrade performance when threads in a warp diverge in which branch they take.

For the filtering of pairs we have to compare the value of the computed inner product with $(\|\mathbf{y}_i\|^2 + \|\mathbf{y}_j\|^2 - \ell^2)/2$. To avoid having to compute this value completely we pre-compute for each vector the value $\frac{1}{2}\|\mathbf{y}_i\|^2 - \frac{1}{4}\ell^2$ such that for each pair the comparison value can be computed by a single addition. Similarly when checking for triples we pre-compute the value $\frac{1}{4}\ell^2 - \frac{1}{4}\|\mathbf{b}\|^2 - \|\mathbf{y}_i\|^2 + \langle \mathbf{b}, \mathbf{y}_i \rangle$ for each vector. Note that after this pre-computation we do not even have to pass the length bound ℓ to the kernel as this is already incorporated in the pre-computed values.

No SimHash filter

We quickly discuss why for our Tensor-GPU implementation we did not choose to make use of the so-called SimHash filter (see [Cha02; Fit+14; Duc18]) based on fast binary operations that has been used successfully to speed-up sieve implementations for the CPU. We do note that the Tensor cores have support for the necessary binary computations. Our reasons are as follows. Firstly the speed-up from using a SimHash filter is less with respect to the 16-bit GPU computations than with respect to the 32-bit CPU computations. Secondly post-processing the passing pairs and triplets on the GPU (recomputing their lengths) has to happen in higher precision on relatively slow CUDA cores, something which can quickly become a bottleneck with a low fidelity filter such as the SimHash. And lastly to compensate for

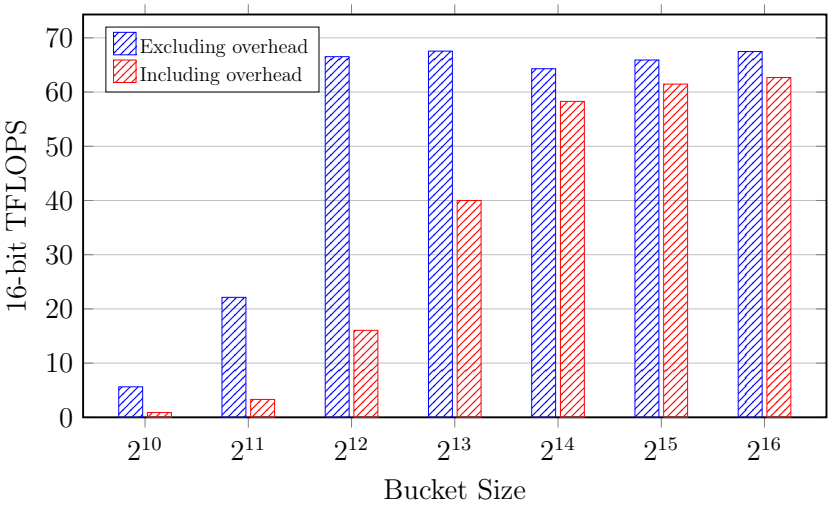


Figure 4.6: Efficiency of the reduction GPU kernel for different bucket sizes on a RTX 2080 Ti, only counting the $2n$ FLOPS per inner product. The overhead includes obtaining the vectors from the database, sending them to the GPU, conversions, recomputing length at higher precision, and retrieving the results from the GPU in a pipelined manner.

false negatives of the SimHash the database size needs to be a larger, which hinders our focus on minimizing RAM usage.

Efficiency

To measure the efficiency of our Tensor-accelerated GPU kernel we did two experiments: the first experiment runs only the kernel with all (converted) data already present in global memory on the GPU, while the second experiment emulates the practical efficiency by including all overhead. This overhead consists of obtaining the vectors from the database, sending them to the GPU, converting them to the appropriate representation, running the reduction kernel, recomputing the length of the resulting close pairs, and retrieving the results from the GPU. Each experiment processed a total of 2^{28} vectors of dimension 160 in a pipelined manner on a single NVIDIA RTX 2080 Ti GPU and with a representative number of 10 CPU threads. We only counted the $2n$ 16-bit floating point operations per inner product

and not any of the operations necessary to transfer data or to filter and process the results. The theoretical limit for this GPU when only using Tensor cores and continuously running at boost clock speeds is 107 TFLOPS, something which is unrealistic in practice.

The results of these experiments are displayed in Figure 4.6. We see that the kernel itself reaches around 65 TFLOPS starting at a bucket size of at least 2^{12}. When including the overhead we see that the performance is significantly limited below a bucket size of 2^{13} which can fully be explained by CPU-GPU memory-bottlenecks. For bucket sizes of at least 2^{14} we see that the overhead becomes reasonably small. We observed that this threshold moves to 2^{15} when using multiple GPUs, because in our hardware the CPU-GPU bandwidth is shared per pair of GPUs.

4.4.2 Tensor cores and precision

The main drawback of the high performance of the tensor cores is that the operations are at low precision. Because the runtime of sieving algorithms is dominated by computing pairwise inner products to find reductions or for bucketing (in case of `triple_gpu`) we focus our attention on this part. Other operations like converting between representations are computationally insignificant and can easily be executed by regular CUDA cores at higher precision.

As the GPU is used as a filter to find (extremely) likely candidates for reduction, we can tolerate some relative error, say up to 2^{-7} in the computed inner product, at the loss of more false positives or missed candidates. Furthermore it is acceptable for our purposes if say 1% of the close vectors are missed because of even larger errors. We consider the case that the vectors are represented using the `fp16` format. In addition we also assume that the inner product is accumulated in 16-bit precision[8].

By first normalizing the vectors we can assume for analysis that they are unit vectors (a normalization close to unit length is actually done in our implementation). Computing the inner product of two vectors using Tensor cores leads to errors in two places, first there

[8]Although the Tensor cores have an option to use 32-bit accumulate, which severely increases the precision, this is capped at half speed in some of the consumer GPUs.

is some *representation error* when representing the vectors in 16-bit precision, and secondly some *computation error* accumulates during the inner product computation.

Representation error

First we show that the representation error between a unit vector $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$ of unit length and its closest 16-bit representative $\hat{\mathbf{y}}$ is small. With a 5 bit exponent and 10 bit mantissa we have $|y_i - \hat{y}_i| \leq \max\{|y_i| \cdot 2^{-11}, 2^{-25}\}$. For the full unit vector this gives an error bound of

$$\|\mathbf{y} - \hat{\mathbf{y}}\| \leq \max\{2^{-11}, 2^{-25} \cdot \sqrt{n}\}. \tag{4.1}$$

Note that for the scope of lattice sieving the dimension n is at most a few hundred, and thus we can safely assume a fixed error bound of 2^{-11} independent of the dimension. By the Cauchy-Schwarz inequality the representation error contributes at most $\max\{2^{-10}, 2^{-24} \cdot \sqrt{n}\}$ to the eventual pairwise inner product between unit vectors and is thus small enough compared to the tolerated error of 2^{-7}.

Computation error

Next we have to consider the computation of the inner product. For each combined multiplication and addition there can be some small error at most μ; we have $\mu \leq 2^{-12}$ in the $[-1, 1]$ range. The main problem is that these errors can accumulate resulting in a worst case error of the form $\mu \cdot n$, which could already be problematic for say $n \geq 32$. For Tensor cores n can be replaced by $n/4$ because the Tensor cores act on 4×4 blocks and the internal computation is at a higher precision [Bla+20], but this still gives a worst-case bound that is too large for our regime.

These worst-case bounds assume that the error at each operation is both maximal and of the same sign, something that is not observed in practice. A common heuristic for an average-case analysis is the assumption that the error is equally likely to be positive as negative, which improves the accumulated error from $\mu \cdot n$ to $O(\mu\sqrt{n})$ with high probability. See [HM19] for probabilistic bounds under such error

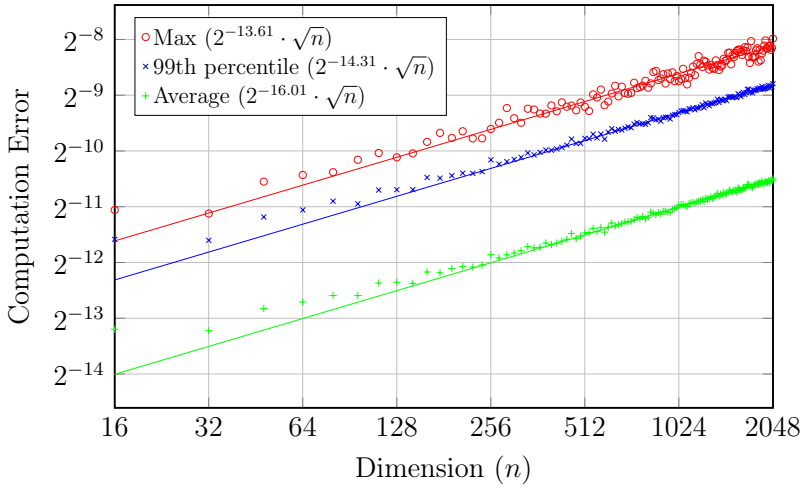


Figure 4.7: Computation error $|S - \hat{S}|$ observed in dimension n over 16384 sampled pairs of unit vectors $\mathbf{y}, \mathbf{y}'$ that satisfy $S := \langle \mathbf{y}, \mathbf{y}' \rangle \approx 0.5$.

models. We see in Figure 4.7 that this heuristic randomized analysis closely matches the experimentally observed growth $\Theta(\sqrt{n})$ of the computation error.

We conclude that when accepting some inaccuracy in a small ratio of inner products the 16-bit Tensor cores contribute effectively in dimensions as large as 2^{11}.

4.5 A dual hash for BDD

Let us recall the principle of the 'dimensions for free' trick [Duc18]; by lifting many short vectors from the sieving context $\mathcal{L}_{[l:r)}$ of dimension $n := r - l$ we can recover a short(est) vector in some larger context $\mathcal{L}_{[l-k:r)}$ for $k > 0$. The sieving implementation G6K [Alb+19] puts extra emphasis on this by lifting many vectors it encounters while reducing a bucket, even when these reduced vectors are not short enough to be added to the database. Recall that for lifting a vector we have to solve an (approx)-CVP instance w.r.t. the *lifting context* $\mathcal{L}_{[l-k:l)}$. By applying the appropriate isometry we identify $\mathcal{L}_{[l-k:l)}$ in this section with a full rank lattice in $\mathbb{R}^k$ (this coincides with the $\mathbf{y}$-

representation that G6K already stores). Lifting using Babai's nearest plane algorithm has a significant cost of $\Theta(n \cdot k + k^2)$ per vector, and thus it would be inefficient to lift all reduction pairs encountered. Therefore the G6K implementation first filters on their length in the *sieving context*, which is already being computed, and only lifts them when they are short enough. The $O(n \cdot k)$ part of the cost, to compute the corresponding target $\mathbf{t}_i - \mathbf{t}_j \in \mathbb{R}^k$, can be amortized to $O(k)$ over all pairs by pre-computing $\mathbf{t}_1, \ldots, \mathbf{t}_s$, leaving the cost of $\Theta(k^2)$ for the Babai nearest plane algorithm w.r.t. $\mathbf{B}_{[l-k:l)}$. In theory by pruning and early aborting the lifting process the cost of $\Theta(k^2)$ could be reduced somewhat more.

To minimize the overhead from the lifting process we want it to be less costly than the reduction part that has a cost of about $O(n)$ per pair. Thus the filter should have an acceptance rate of at most $O(n/k^2)$. A squared length filter of 1.8 on the sieving length, leads to an exponentially small acceptance rate in the sieving dimension $n = r - l$. However assuming BGJ1 bucketing and input vectors of squared length $\frac{4}{3}$ the acceptance rate would be about $0.9754^{n+o(n)}$; giving a concrete acceptance rate of order 10^{-2} even in sieving dimensions as large as $n = 128$.

When using GPUs we have an additional problem that we cannot run the lifting computation inline in the reduction kernel, as this would diminish the overall performance by using more registers and shared memory for storing the targets, and because the remaining threads in a warp would also have to wait for this. Therefore one would need to return all the pairs of indices that pass the filter and store them temporarily in global memory. With an acceptance rate in the order of 10^{-2} this would imply tens of millions of results for the usual bucket sizes, taking more than ten times the storage of the input vectors. This would become a bottleneck in practice, even ignoring the reduction kernel overhead of saving those results in the first place.

As an alternative we introduce a stronger filter with an emphasis on the extra length added by the lifting. Most short vectors will lift to rather large vectors, as by the Gaussian Heuristic we can expect an extra length of $\mathrm{gh}(\mathcal{L}_{[l-k:l)}) \gg \mathrm{gh}(\mathcal{L}_{[l-k:r)})$. For the few lifts that we are actually interested in we expect an extra length of only $\delta \cdot \mathrm{gh}(\mathcal{L}_{[l-k:l)})$, for some $0 < \delta < 1$ (say $\delta \in [0.1, 0.5]$ in practice). This means that we need to catch those pairs $\mathbf{t}_i - \mathbf{t}_j$ that lie exceptionally

close to the lattice $\mathcal{L}_{[l-k:l)}$, also known as δ-BDD instances, and their share decreases exponentially as δ^k, assuming the targets are uniform over $\mathrm{span}(\mathcal{L}_{\mathrm{bdd}})/\mathcal{L}_{\mathrm{bdd}}$, where $\mathcal{L}_{\mathrm{bdd}} := \mathcal{L}_{[l-k:l)}$. E.g., for reasonable parameters $\delta = 0.5$ and $k = 24$ this amounts to only $6 \cdot 10^{-8}$ of all pairs.

So we want to filter δ-BDD instances over the k-dimensional lattice $\mathcal{L}_{\mathrm{bdd}}$. More abstractly we need a filter that quickly checks if pairs are (exceptionally) close over the *torus* $\mathrm{span}(\mathcal{L}_{\mathrm{bdd}})/\mathcal{L}_{\mathrm{bdd}} \cong \mathbb{R}^k/\mathcal{L}_{\mathrm{bdd}}$. Constructing such a filter directly for this rather complex torus and our practical parameters seems to require at least quadratic time like Babai's nearest plane algorithm. Instead we introduce a *dual hash* to move the problem to the much simpler but possibly higher dimensional torus $\mathbb{R}^h/\mathbb{Z}^h \cong \left[-\frac{1}{2}, \frac{1}{2}\right)^h$. More specifically, we will use inner products with short dual vectors to build a BDD distinguisher in the spirit of the so-called dual attack on LWE given in [MR09] (the general idea can be traced back at least to [AR05]). This is however done in a different regime, where the shortest dual vectors are very easy to find (given the small dimension k of the considered lattice); we will also carefully select a subset of those dual vectors to optimize the fidelity of our filter. Recall that the dual of a lattice $\mathcal{L}_{\mathrm{bdd}}$ is defined as $\mathcal{L}^*_{\mathrm{bdd}} := \{\mathbf{w} \in \mathrm{span}(\mathcal{L}_{\mathrm{bdd}}) : \langle \mathbf{w}, \mathbf{v} \rangle \in \mathbb{Z} \text{ for all } \mathbf{v} \in \mathcal{L}_{\mathrm{bdd}}\}$.

Definition 71 (Dual hash). *For a full rank lattice $\mathcal{L} \subset \mathbb{R}^k$, dual hash length $h \geq k$ and a full (row-rank) matrix $\mathbf{D} \in \mathbb{R}^{h \times k}$ with rows in the dual $\mathcal{L}^*$, we define the dual hash*

$$\begin{aligned} \mathcal{H}_{\mathbf{D}} : \mathbb{R}^k/\mathcal{L} &\to \mathbb{R}^h/\mathbb{Z}^h \cong \left[-\frac{1}{2}, \frac{1}{2}\right)^h, \\ \mathbf{t} &\mapsto \mathbf{D}\mathbf{t}. \end{aligned}$$

The dual hash relates distances in $\mathbb{R}^k/\mathcal{L}$ to those in $\mathbb{R}^h/\mathbb{Z}^h$. Note that the distance $\mathrm{dist}(\mathbf{t}', \mathbb{Z}^h)$ is well defined for any $\mathbf{t}' \in \mathbb{R}^h/\mathbb{Z}^h$.

Lemma 72. *Let $\mathcal{L} \subset \mathbb{R}^k$ be a full rank lattice with some dual hash $\mathcal{H}_{\mathbf{D}}$ of length h. Then for any $\mathbf{t} \in \mathbb{R}^k$ we have*

$$\mathrm{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h) \leq \sigma_1(\mathbf{D}) \cdot \mathrm{dist}(\mathbf{t}, \mathcal{L}),$$

where $\sigma_1(\mathbf{D})$ denotes the largest singular value of $\mathbf{D} \in \mathbb{R}^{h \times k}$.

Proof. Let $\mathbf{x} \in \mathcal{L}$ such that $\|\mathbf{x} - \mathbf{t}\| = \text{dist}(\mathbf{t}, \mathcal{L})$. By definition we have $\mathbf{D}\mathbf{x} \in \mathbb{Z}^h$ and thus $\mathcal{H}_\mathbf{D}(\mathbf{t}-\mathbf{x}) \equiv \mathcal{H}_\mathbf{D}(\mathbf{t}) \bmod \mathbb{Z}^h$. We conclude by noting that $\text{dist}(\mathcal{H}_\mathbf{D}(\mathbf{t}-\mathbf{x}), \mathbb{Z}^h) \leq \|\mathbf{D}(\mathbf{t}-\mathbf{x})\| \leq \sigma_1(\mathbf{D})\|\mathbf{t}-\mathbf{x}\|$. □

So if a target $\mathbf{t}$ lies very close to the lattice then $\mathcal{H}_\mathbf{D}(\mathbf{t})$ lies very close to $\mathbb{Z}^h$. We can use this to define a filter that passes through BDD instances.

Definition 73 (Filter). *Let $\mathcal{L} \subset \mathbb{R}^k$ be a full rank lattice with some dual hash $\mathcal{H}_\mathbf{D}$. For a hash bound H we define the filter function*

$$\mathcal{F}_{\mathbf{D},H} : \mathbf{t} \mapsto \begin{cases} 1, & \text{if } \text{dist}(\mathcal{H}_\mathbf{D}(\mathbf{t}), \mathbb{Z}^h) \leq H, \\ 0, & \text{else.} \end{cases}$$

Note that computing the filter has a cost of $O(h \cdot k)$ for computing $\mathbf{D}\mathbf{t}$ for $\mathbf{D} \in \mathbb{R}^{h \times k}$ followed by a cost of $O(h)$ for computing $\text{dist}(\mathbf{D}\mathbf{t}, \mathbb{Z}^h)$ using simple coordinate-wise rounding. Given that $h \geq k$, computing the filter is certainly not cheaper than ordinary lifting, which is the opposite of our goal. However this changes when applying the filter to all pairs $\mathbf{t}_i - \mathbf{t}_j$ with $1 \leq i < j \leq h$. We can pre-compute $\mathbf{D}\mathbf{t}_1, \ldots, \mathbf{D}\mathbf{t}_s$ once, which gives a negligible overhead for large buckets, and then compute $\mathbf{D}(\mathbf{t}_i - \mathbf{t}_j)$ by linearity, lowering the total cost to $O(h)$ per pair.

4.5.1 An average-case analysis of the dual hash

Lemma 72 allows us to choose a dual hash bound H such that our filter $\mathcal{F}_{D,H}$ returns 1 for all δ-BDD instances. Such a worst-case bound is far from practical behaviour, thereby giving a much looser filter than needed. In this section we introduce an average-case analysis, both for which bound to pick and for the positive rate of our filter. We assume that the targets are uniformly distributed, e.g., uniform over $\mathbb{R}^k/\mathcal{L}_\text{bdd}$, more specifically unless otherwise stated we assume without loss of generality (given that the dual hash filter only depends on the coset and not the particular representative) that the targets are uniform over the Voronoi cell $\mathcal{V}(\mathcal{L}_\text{bdd}) \subset \mathbb{R}^k$.

The dual hash bound

We discuss what bound H to pick for the dual hash filter $\mathcal{F}_{\mathbf{D},H}$, such that we detect those targets $\mathbf{t}$ that lie at some close distance

$\mathrm{dist}(\mathcal{L}_{\mathrm{bdd}}, \mathbf{t}) \leq R := \delta \cdot \lambda_1(\mathcal{L}_{\mathrm{bdd}})$ of the lattice. We consider the unique decoding regime, i.e., when $\delta < \frac{1}{2}$. In this regime we can assume without loss of generality that the normalized targets $\mathbf{t}/\|\mathbf{t}\|$ are uniformly distributed over the sphere. Suppose that $\mathbf{D}^\top\mathbf{D}$ has eigenvalues $\sigma_1^2, \ldots, \sigma_k^2$ with corresponding normalized (orthogonal) eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$. By a change of basis we can equivalently assume that $\mathbf{t} = \sum_{i=1}^k t_i \mathbf{v}_i$ with $(t_1, \ldots, t_k)/\|\mathbf{t}\|$ uniformly distributed over the sphere. Computing the expectation we see

$$\begin{aligned}\mathbb{E}[\mathrm{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h)^2] \leq \mathbb{E}[\|\mathbf{D}\mathbf{t}\|^2] &= \mathbb{E}\left[\sum_{i=1}^k t_i^2 \cdot \sigma_i^2\right] \\ &= \sum_{i=1}^k \sigma_i^2 \cdot \mathbb{E}[t_i^2] = \|\mathbf{t}\|^2 \cdot \frac{1}{k}\sum_{i=1}^k \sigma_i^2\end{aligned}$$

So instead of the worst case bounds from Lemma 72, the average-case expected distance $\mathrm{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h)$ is bounded by $\sqrt{\frac{1}{k}\sum_{i=1}^k \sigma_i^2} \cdot \|\mathbf{t}\|$. Note that if the latter bound is relatively small then we can also expect that all coordinates of $\mathbf{D}\mathbf{t}$ lie in $[-\frac{1}{2}, \frac{1}{2}]$, and thus the first inequality is tight. If we explicitly assume that $\mathrm{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h) = \|\mathbf{D}\mathbf{t}\|$ in this regime we can also compute the variation which equals

$$\mathrm{Var}\left[\|\mathbf{D}\mathbf{t}\|^2\right] = \frac{\|\mathbf{t}\|^4}{(k/2+1)}\left(\frac{1}{k}\cdot\sum_{i=1}^k \sigma_i^4 - \left(\frac{1}{k}\sum_{i=1}^k \sigma_i^2\right)^2\right).$$

We see that when all eigenvalues $\sigma_1^2, \ldots, \sigma_k^2$ are equal the variance is 0, and more generally the more balanced they are the better. For the rest of the section we assume that the dual hash bound for targets at distance R is set to $R \cdot \sqrt{\frac{1}{k}\sum_{i=1}^k \sigma_i^2}$, accepting that we only detect say half of such targets. Note that the bound is based on targets at distance exactly R, the targets that lie closer are even more probable to pass the filter. By adding some multiple of the variance to this bound one could obtain more quantative values for the false negative rate using Chebyshev's inequality. For a random lattice most targets are still uniquely decodable even for δ somewhat larger than $\frac{1}{2}$, and thus we can still use the above bound as a good estimate.

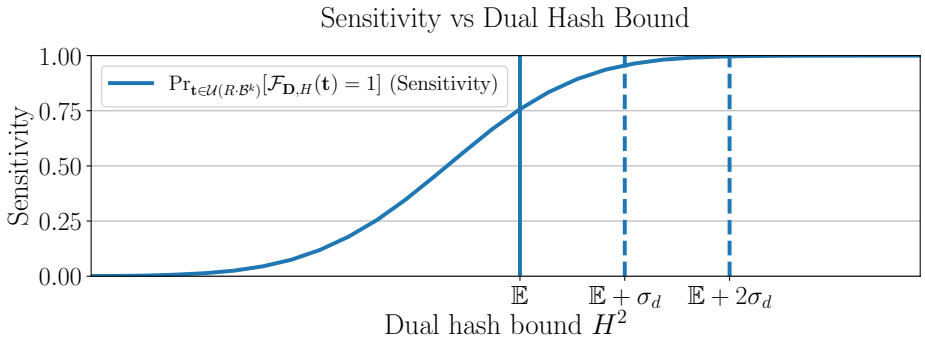


Figure 4.8: Experimental results of the true positive rate of the dual hash filter $\mathcal{F}_{\mathbf{D},H}$ in $\mathcal{L}_{\text{bdd}}$ of dimension $k = 24$ with $h = 64$ dual vectors. The 2^{17} targets were sampled uniformly in a ball of radius $R := 0.5 \cdot \text{gh}(\mathcal{L}_{\text{bdd}})$. With a bound of $H^2 = \mathbb{E} := R \cdot \frac{1}{k}\text{Tr}(\mathbf{D}^\top\mathbf{D})$ the true positive rate is 72.8%, and at 1 and 2 standard deviations higher it is respectively 94.8% and 99.5%.

Positive rate

We try to understand the positive rate p_{pos} of the filter, e.g., the ratio of targets $\mathbf{t} \sim \mathbb{R}^k/\mathcal{L}_{\text{bdd}}$ for which $\mathcal{F}_{\mathbf{D},H}(\mathbf{t}) = 1$. It seems hard to determine the exact positive rate, as multiple factors play are role. Still we can get some good estimate by splitting the analysis in two cases: the *preserved* and the *unpreserved* regime. Consider a target $\mathbf{t} \in \mathbb{R}^k$ and let $\mathbf{x}$ be a closest vector in $\mathcal{L}_{\text{bdd}}$ to $\mathbf{t}$. We will say that we are in the preserved regime whenever $\mathbf{D}(\mathbf{t} - \mathbf{x}) \in [-\frac{1}{2}, \frac{1}{2}]^h$ (i.e., $\mathbf{Dx}$ remains a closest vector of $\mathbf{Dt}$ among $\mathbb{Z}^h$), in which case it holds that $\|\mathbf{D}(\mathbf{t} - \mathbf{x})\|_2 = \text{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h)$. The unpreserved regime considers those targets for which there is no equality, i.e., $\|\mathbf{D}(\mathbf{t} - \mathbf{x})\|_2 > \text{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h)$.

Preserved Regime. We assumed without loss of generality that the targets $\mathbf{t}$ are uniform over the Voronoi cell $\mathcal{V}(\mathcal{L}_{\text{bdd}})$, such that their closest vector is $\mathbf{0}$. In the preserved regime we have the equality $\text{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h) = \|\mathbf{Dt}\|_2$, and thus $\mathcal{F}_{\mathbf{D},H}(\mathbf{t}) = 1$ if and only if $\|\mathbf{Dt}\|_2 \leq$

H. The positive rate p_{pres} is then given by

$$p_{\text{pres}} = \Pr_{\mathbf{t}\sim\mathcal{U}(\mathcal{V}(\mathcal{L}_{\text{bdd}}))}\left[\|\mathbf{Dt}\|_2 \leq H\right].$$

We proceed in the same way as earlier by picking applying a basis transformation using the normalized eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ of $\mathbf{D}^\top\mathbf{D}$. Let $\mathbf{V} := [\mathbf{v}_1; \ldots; \mathbf{v}_k]$, then we have

$$p_{\text{pres}} = \Pr_{\mathbf{t}\sim V\cdot\mathcal{U}(\mathcal{V}(\mathcal{L}_{\text{bdd}}))}\left[\sum_{i=1}^{k} t_i^2\cdot\sigma_i^2 \leq H^2\right] = \frac{\text{vol}(V\cdot\mathcal{V}(\mathcal{L}_{\text{bdd}})\cap\mathcal{E}_{H,\sigma_1^2,\ldots,\sigma_k^2})}{\text{vol}(V\cdot\mathcal{V}(\mathcal{L}_{\text{bdd}}))},$$

where $\mathcal{E}_{H,\sigma_1^2,\ldots,\sigma_k^2}$ is the ellipsoid defined by $\{\mathbf{t}\in\mathbb{R}^k : \sum_{i=1}^k t_i^2\cdot\sigma_i^2 \leq H^2\}$. We can simply bound the volume in the numerator by that of the ellipsoid, and note that the volume of the denominator equals the determinant $\det(\mathcal{L}_{\text{bdd}})$. So we conclude that

$$p_{\text{pres}} \leq \frac{\text{vol}(\mathcal{E}_{H,\sigma_1^2,\ldots,\sigma_k^2})}{\text{vol}(\mathcal{V}(\mathcal{L}_{\text{bdd}}))} = \frac{\text{vol}(\mathcal{B}^k)\cdot H^k}{\det(\mathcal{L}_{\text{bdd}})\cdot\prod_{i=1}^k\sigma_i}.$$

Note that the inequality is only provably tight when $H/\sigma_k \leq \frac{1}{2}\lambda_1(\mathcal{L}_{\text{bdd}})$, where σ_k^2 is the smallest eigenvalue, but again for random lattices we can expect the estimate to also be reasonably close also for larger bounds.

Unpreserved Regime. In the unpreserved regime $\text{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h)$ is not really a useful metric, as there will seemingly be no clear relation with $\|\mathbf{D}(\mathbf{t}-\mathbf{x})\|_2$. Inspired by practical observations, we analyse these positives from the heuristic assumption in this regime that every $\mathbf{Dt}$ is uniformly distributed over $[-\frac{1}{2},\frac{1}{2})^h$ modulo $\mathbb{Z}^h$. Then we can ask the question how probable it is that $\|\mathbf{Dt}\|_2 = \text{dist}(\mathbf{Dt},\mathbb{Z}^h) \leq H$; i.e., that the target passes the filter even though it is not close to the lattice. This is equivalent to the volume of the intersection of an h-dimensional ball with radius H and the hypercube $[-\frac{1}{2},\frac{1}{2}]^h$. We can bound this by just the volume of the ball,

$$\begin{aligned} p_{\text{unpr}} &:= \Pr_{\mathbf{v}\in\mathcal{U}[-\frac{1}{2},\frac{1}{2})^h}\left[\mathbf{v}\in H\cdot\mathcal{B}^h\right] \\ &= \Pr_{\mathbf{v}\in\mathcal{U}(H\cdot\mathcal{B}^h)}\left[\mathbf{v}\in[-\tfrac{1}{2},\tfrac{1}{2})^h\right]\cdot\text{vol}(H\cdot\mathcal{B}^h) \leq H^h\cdot\text{vol}\,\mathcal{B}^h. \end{aligned}$$

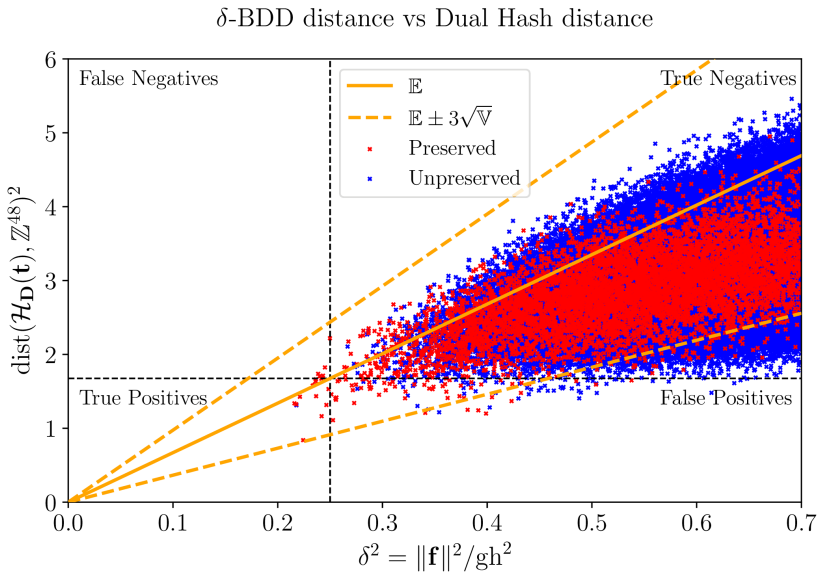


Figure 4.9: Dual hash filter correlation on the context $\mathcal{L}_{[14:30)}$ for a reduced 160-dimensional lattice using 48 dual vectors. The BDD-bound was computed with a representative squared length bound of 1.44 and the 2^{20} targets are uniformly sampled over the Voronoi cell around 0.

For a small bound H the probability $\Pr_{\mathbf{v}\in\mathcal{U}(H\cdot\mathcal{B}^h)}\left[\mathbf{v}\in[-\frac{1}{2},\frac{1}{2})^h\right]$ is close to 1 and thus the bound is rather tight. For somewhat larger values of H we can estimate $\Pr_{\mathbf{v}\in\mathcal{U}(H\cdot\mathcal{B}^h)}\left[\mathbf{v}\in[-\frac{1}{2},\frac{1}{2})^h\right]$ by heuristically assuming independence between the coordinates. Recall that if $\mathbf{v}\sim\mathcal{U}(\mathcal{B}^h)$, then each coefficient $\mathbf{v}_i^2$ follows a $\mathrm{Beta}(\frac{1}{2},\frac{h+1}{2})$ distribution with CDF $I_x(\frac{1}{2},\frac{h+1}{2})$, leading to the heuristic estimate

$$p_{\text{unpr}} \approx I_{\frac{1}{4H^2}}(\tfrac{1}{2},\tfrac{h+1}{2})^h \cdot H^h \cdot \operatorname{vol}\mathcal{B}^h.$$

Alternatively one can use Monte-Carlo sampling to estimate p_{unpr}. The number of samples (under a fixed error variance) is inversely proportional to the event probability, and thus directly estimating p_{unpr} for low values of H is too costly. Luckily for small values of H the probability $\Pr_{\mathbf{v}\in\mathcal{U}(H\cdot\mathcal{B}^h)}\left[\mathbf{v}\in[-\frac{1}{2},\frac{1}{2})^h\right]$ is large and thus we can estimate that probability instead, leading to an efficient Monte-Carlo estimate both for small and large values of H. To determine in which regime we

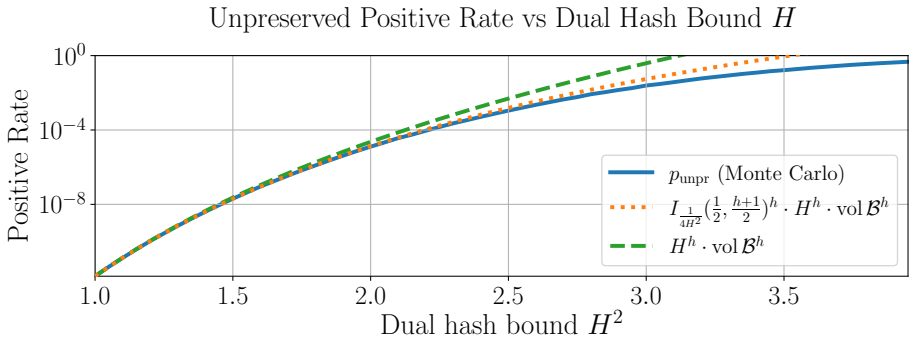
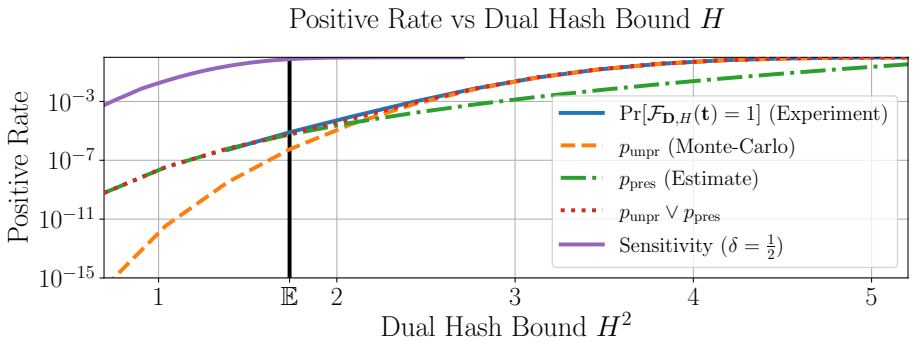


Figure 4.10: The unpreserved positive rate p_{unpr} for $h = 48$.



are one could use the heuristic estimate for $\Pr_{\mathbf{v} \in \mathcal{U}(H \cdot \mathcal{B}^h)}\left[\mathbf{v} \in [-\frac{1}{2}, \frac{1}{2})^h\right]$ presented earlier. Note that p_{unpr} only depends on the filter threshold H and the number of dual vectors h and not on the specific matrix $\mathbf{D}$. We could thus precompute p_{unpr} for relevant parameters instead of computing it on the fly.

Choosing a dual hash

We will shortly discuss how to pick the dual hash matrix $\mathbf{D} \in \mathbb{R}^{h \times k}$. The goal is to obtain a filter with a good correlation, i.e., a good trade-off between the positive-rate and the sensitivity. We fix the target distance to some $0 < R \leq \frac{1}{2}\lambda_1(\mathcal{L}_{\text{bdd}})$, and as the computational cost mostly depends on the number of dual vectors h we will try to optimize $\mathbf{D}$ for a fixed number of dual vectors h.

To simplify the overall optimization we set the dual hash bound to $H := R \cdot \sqrt{\frac{1}{k}\sum_{i=1}^{k}\sigma_i^2}$, such that the false negative rate is expected to be reasonable for uniform targets at distance at most R. We can now focus on minimizing the positive rate.

In the preserved regime the (bound on the) positive rate is proportional to $H^k/\prod_{i=1}^{k}\sigma_i$. Note that $\mathrm{Tr}(\mathbf{D}^\top\mathbf{D}) = \sum_{i=1}^{k}\sigma_i^2$ and $\det(\mathbf{D}^\top\mathbf{D}) = \prod_{i=1}^{k}\sigma_i^2$, the positive rate in the preserved regime is thus propertional to

$$p_{\text{pres}} \sim \frac{(\frac{1}{k}\,\mathrm{Tr}(\mathbf{D}^\top\mathbf{D}))^{k/2}}{\sqrt{\det(\mathbf{D}^\top\mathbf{D})}},$$

which is a well known conditioning metric for matrices. To optimize this one can think of picking h dual vectors that are of about the same length and somewhat orthogonal. For the unpreserved regime the positive rate is mostly proportional to H^k, which means we want $\mathrm{Tr}(\mathbf{D}^\top\mathbf{D})^{k/2}$ to be small; this can be achieved by working with short dual vectors.

To summarize we want to find a set of short dual vectors to form the dual hash such that $\mathbf{D}$ is well conditioned. One initial method is to just pick the h shortest dual vectors (modulo sign). This satisfies the needs of the unpreserved regime, but the conditioning of the resulting matrix is often not that great. Given a list of short dual vectors we can greedily try to improve the conditioning of $\mathbf{D}$ by replacing some of the (row) vectors from the list.

From experiments we can conclude that this greedy method to improve the filter works really well. For example with the parameters as in Figure 4.9, picking the 48 shortest dual vectors leads to a positive rate of $1.3 \cdot 10^{-4}$ for a false negative rate of 1%; using the greedy construction improves the positive rate down to $1.4 \cdot 10^{-5}$ for the same false negative rate. The additional overhead of the greedy method is negligible in somewhat large dimensions and easily won back from allowing a lower number of dual vectors h.

4.5.2 Application and results

We shortly discuss how one would use the introduced dual hash techniques in practice for solving SVP challenges more efficiently. Recall that for such challenges we are given a lattice $\mathcal{L}$ of dimension d and we

try to find a short vector of norm at most $1.05 \cdot \mathrm{gh}\,\mathcal{L}$. We do this by sieving in some smaller context $\mathcal{L}_{[l:d)}$ (for progressively decreasing l), and by lifting vectors from this context to the full lattice $\mathcal{L}_{[0:d)} = \mathcal{L}$, hoping to encounter a short enough vector. The goal of the dual hash is to efficiently filter pairs of vectors which difference lies close to the lattice $\mathcal{L}_{[0:l)}$, thereby increasing the number of dimensions for free l.

Choosing the parameters

To use the dual hash in practice as a filter we need to decide on what context to use it and what the threshold should be. Applying the dual hash to the full lift context $\mathcal{L}_{[0:l)}$ might fail to return short vectors for positions $l' > 0$, which are also needed to improve the quality of the basis. Therefore we apply the dual hash to the smaller *filter context* $\mathcal{L}_{\mathrm{bdd}} := \mathcal{L}_{[f:l)}$. If a vector is short in the context $\mathcal{L}_{[l':r)}$ for some $l' < f$ then we can also expect it to be short in the filter context, and therefore to be catched by our filter.

We also need to decide on a distance threshold. Let $\mathbf{v}$ be a lattice vector in $\mathcal{L}_{[l:r)}$ of length R. We can assume that $R \geq \ell$, where ℓ is the sieving length bound, as otherwise the vector would already be inserted (and always lifted) in the sieving database. Suppose that $\mathbf{v}$ lifts to a short vector with length at most $\ell_{l'}$ in $\mathcal{L}_{[l':r)}$ for $\kappa \leq l' \leq f$. This corresponds to a target $\mathbf{t}$ at distance at most

$$\mathrm{dist}(\mathbf{t}, \mathcal{L}_{[l':l)})^2 \leq \ell_{l'}^2 - \ell^2.$$

from the lattice $\mathcal{L}_{[l':l)}$. Although we cannot know what the length of $\mathbf{t}$ would be in the filter context we can expect this to be close to $\sqrt{\frac{l-f}{l-l'}}\|\mathbf{t}\|$ by the Gaussian Heuristic. Therefore setting the filter length bound to

$$F_{l'} := \sqrt{\frac{l-f}{l-l'}\left(\ell_{l'}^2 - \ell^2\right)}$$

allows a significant part of the short lifts in the context $\mathcal{L}_{[l':r)}$ through the filter. Note that most of the pairs we lift are much larger on the sieving part, and thus have to be even shorter in the filter context; definitely passing the above filter length bound. We conclude by setting the filter to aim for a length of at most $F := \max_{\kappa \leq l' \leq f}\{F_{l'}\}$.

Given the filter length bound we could immediately apply Lemma 72 to obtain a threshold for the dual hash that guarantees that our filter has no false negatives. However as usual there is a trade-off between the number of false negatives and the positive rate of the filter. For our purposes we set the bound at the expectation plus 3 standard deviations in the preserved regime to prevent most false negatives. For a more precise bound under a fixed false negative ratio one could fall back to Monte-Carlo sampling methods as we do not know of a closed form formula for the distribution. Figure 4.9 shows the effectiveness of the dual hash filter based on realistic parameters as encountered during a 130-dimensional pump on a 160-dimensional lattice. The pre-processing of the basis consisted of a workout with pumps up to dimension 128.

4.5.3 Implementation details

Given a list of pre-computed $\mathbf{Dt}_1, \ldots, \mathbf{Dt}_m$ we want to use the GPU to efficiently compute $\mathrm{dist}(\mathbf{D}(\mathbf{t}_i - \mathbf{t}_j), \mathbb{Z}^h)$ for all $i < j$. As usual the actual implementation requires some trade-offs to significantly improve performance. Given that the dimension of the dual hash seems to have more impact than the precision of the values we choose for an 8-bit integer representation for the dual hash coordinates in $[-1/2, 1/2)$ by dividing it in 256 equally sized intervals. The added benefit of this representation is that the $\bmod \mathbb{Z}$ operations are implicitly handled by integer overflow. Both CUDA and Tensor cores have special instructions and very good performance for 8-bit arithmetic, even when using 32 bits to accumulate inner products.

Given a list of pre-computed $\mathbf{Dt}_1, \ldots, \mathbf{Dt}_m$ we want to use the GPU to efficiently compute $\mathrm{dist}(\mathbf{D}(\mathbf{t}_i - \mathbf{t}_j), \mathbb{Z}^h)^2$ for all $i < j$, where the coordinates use an 8-bit integer representation. We focus on the core computation as for the memory movement one can apply a similar strategy as for the reduction kernel.

Dual hash for CUDA cores

Although CUDA cores are flexible we have to pack 4 of the 8-bit values together in a 32-bit register $a = a_3|a_2|a_1|a_0$ and apply special operations to obtain optimal 8-bit arithmetic performance. First we need

to take the coordinate-wise difference and then take the modulo to get back in the interval $[-1/2, 1/2)$. By representing these values using signed integers the modulo is equivalent to integer overflow. CUDA has an operation to take the coordinate-wise difference, however this actually decodes into multiple instructions. Therefore as a trade-off we just take the 32-bit integer different between a and b, ignoring off by one errors in each coordinate due to a possible carry bit. For computing the length we can use the relatively new operation `__dp4a(a,b,c)` that computes the inner product between the packed a and b and accumulates this in a 32-bit integer c. So in only two cycles we can compute the squared distance over 4 coordinates modulo $\mathbb{Z}^4$.

Dual hash for Tensor cores

Although Tensor cores can be up to 4 times faster than the CUDA cores for similar precision they can basically do only a single thing well: pairwise inner products, i.e., a matrix product. So to use the Tensor cores effectively we need to convert this computation to that of a matrix product. We quickly discuss how one could potentially make such an adjustment.

We could use the fact that $\operatorname{dist}(x, \mathbb{Z})^2 \approx (1 - \cos(x \cdot 2\pi))/16$ for $x \in [-\frac{1}{4}, \frac{1}{4}] + \mathbb{Z}$. For the remaining range the value of $(1 - \cos(x \cdot 2\pi)/16$ is somewhat smaller, but in the BDD-regime we are working these values still seem large enough to reject bad vectors. So using this similarity we want to compute

$$d_{ij} := \sum_{l=1}^{h} \frac{1 - \cos((\mathbf{D}(\mathbf{t}_i - \mathbf{t}_j))_l \cdot 2\pi)}{16}.$$

from some pre-computed values depending on $\mathbf{Dt}_i$ and $\mathbf{Dt}_j$ respectively. Rewriting using trigonometric identities we get:

$$d_{ij} = \frac{m}{16} - \frac{1}{16}\left(\sum_{l=1}^{m} \cos(2\pi \mathbf{Dt}_i)\cos(2\pi \mathbf{Dt}_j) + \sin(2\pi \mathbf{Dt}_i)\sin(2\pi \mathbf{Dt}_i)\right).$$

If we pre-compute that values $(\cos((\mathbf{Dt}_i)_l \cdot 2\pi))_l$ and $(\sin((\mathbf{Dt}_i)_l \cdot 2\pi))_l$ we can compute the above sum by computing two h-dimensional (pairwise) inner products which Tensor cores can do efficiently. Again we can use 8-bit representations and computing the inner product (with

32-bit accumulate) is up to 4 times faster on Tensor cores compared to CUDA cores. However here we need to compute two h-dimensional inner products instead of one, but this is compensated by not having to compute the coordinate-wise difference first. So one could expect up to 4 times the performance compared to CUDA cores. In the end we did not implement this adaptation as the pairwise dual hash computations were not a bottleneck, and thus improving it would only result in a minor overall speed-up.

4.6 New records

4.6.1 Comparison

We compare several of our sieve implementations experimentally. Although our BDGL-like implementations `bdgl` and `bdgl_gpu` will eventually be faster than the BGJ-like implementations `triple` by G6K[9] and `triple_gpu` by us, the cross-over point could be outside of practical dimensions. For the comparison we run a pump up to dimension 120 and 140 for CPU and GPU respectively in a lattice of dimension 160 that has been pre-processed by a workout up to dimension 118 and 138. In Figure 4.11 we display the wall-clock time taken for each Sieve operation during the pump up. All our GPU implementations use a multi-bucket parameter of 4, which should give a balanced comparison based on Figure 4.4. Any on-the-fly lifting or dual hash techniques are disabled. For the remaining parameters we refer to the next Section 4.6.2.

CPU sieves

We observe in Figure 4.11 that our BDGL implementations `2-bdgl` and `3-bdgl` are extremely practical. They both improve upon the record-holding `triple` sieve from dimension ± 85 and onwards. The speed-up of `2-bdgl` and `3-bdgl` over `triple` increases to a factor of $5.3\times$ and $3.1\times$ respectively in dimension 120[10]. We also see that while

[9] The `triple` sieve in G6K is in the meantime renamed to `hk3` after [HK17].

[10] In the original work we reported a speed-up of $2.7\times$ for `3-bdgl`. By optimising the cache locality of the reduction phase this was improved to $3.1\times$. These optimisations were much more effective for the larger buckets of `2-bdgl`, resulting

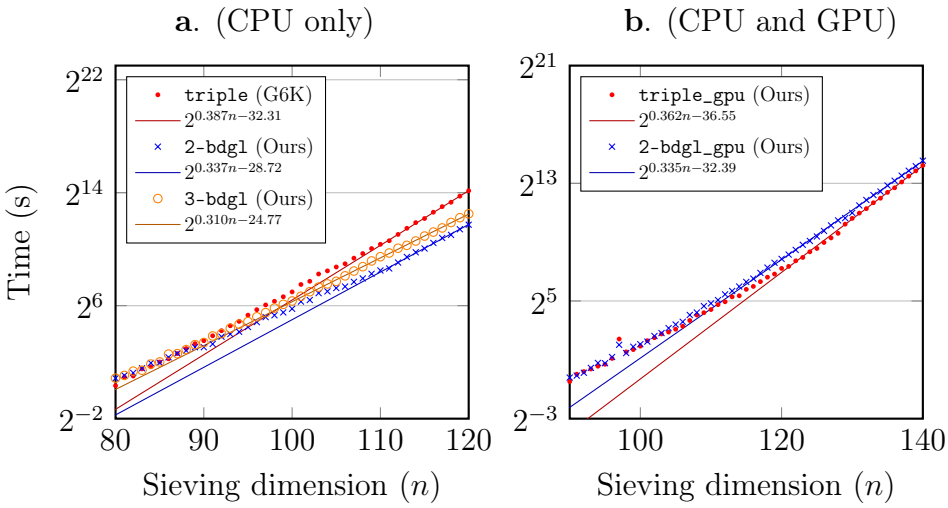


Figure 4.11: Comparison of different sieve implementations from [Alb+19] and this work. We ran a single pump up in a well-reduced 160-dimensional lattice to a sieving dimension of 120 and 140 for CPU only and GPU accelerated respectively. The timings give the amount of time spend in each sieving dimension. All experiments used a saturation of 37.5% with a database size of $2.77 \cdot 2^{0.2075n}$. All 40 CPU cores and 4 GPU's were used. The fitting is obtained by a linear least-squares regression on the last 10 dimensions in log-space.

`3-bdgl` is asymptotically faster than `2-bdgl`, this is not concretely the case below dimension 120. Given the extrapolations we expect the cross-over around dimension 146.

Given the large speed-up of the `2-bdgl` sieve over the record-holding `triple` sieve we also tried to resolve the 155-dimensional Darmstadt 1.05-approxSVP record challenge. We solved the challenge (with seed 1) with `2-bdgl` in 51.7 hours on 40 cpu cores (machine specifications as in Table 4.1), or about 2069 cpu hours, compared to a reported 1056 cpu *days* for the `triple` sieve. This is more than a factor 12× speed-up (ignoring cpu core differences). Part of the speed-up (about a factor 4×) can be explained by a few more dimensions for free (34 vs 28). It is unclear if we got lucky or if this increase can be attributed to the algorithm.

in the final 5.3× speed-up.

GPU sieves

Firstly, we observe that the GPU accelerated sieves are significantly faster than the CPU-only sieves. In dimension 120 `triple_gpu` is a tremendous factor 121× faster than `triple`. Secondly, we observe a different behaviour of the asymptotically superior sieve `2-bdgl_gpu` versus `triple_gpu`. While for the CPU-only version the cross-over between the two was already in dimensions around 85, for the GPU version the cross-over lies in a dimension greater than 140. Following the extrapolations, we only expect them to cross in dimension ±154. The main reason for this is that the BDGL sieve obtains its speed-up from having many small buckets, but for the GPU implementation there is a large minimum bucket size to prevent memory bottlenecks in the reduction phase. The large minimum bucket size shifts the cross-over point by more than 70 dimensions. In this light, it did not appear pertinent to implement `3-bdgl_gpu`, which, while being asymptotically faster, would cross-over even later.

4.6.2 SVP parameter tuning

There are many parameters in our implementation that can be tuned for optimal performance with respect to memory and time complexity. We will focus on `triple_gpu` as we have shown it to be the fastest implementation in practical sieving dimensions $n \leq 150$. As low level parameters, such as minimum bucket sizes for GPUs, are discussed earlier, here we discuss the higher level parameters to solve 1.05-approxSVP for a lattice of dimension d.

Given the large amount of computational power available with the 4 GPUs, we can potentially solve lattice 1.05-approxSVP up to dimension 180 in reasonable time on a single machine. The main limiting factor at that point is the available memory, in our case 1.5 TiB RAM. We have spent significant efforts aiming to reduce the memory footprint of our G6K-GPU implementation, such as maintaining only basis coordinates, length and a hash of each vector in our database. Many parameters can be safely tweaked in certain regions without significantly affecting time complexity, hence we focus more on suitable values that limit memory usage.

To increase dimensions-for-free, and thus decrease memory usage, we enabled DownSieve for all workouts for a stronger preprocessing.

$\mathsf{TD4F}(n) = \lfloor n/\log(n) \rfloor$	$\mathsf{MaxSieveDim}(n) = n - \mathsf{TD4F}(n) + 4$
$\mathsf{SaturationRatio} = .375$	$\mathsf{DBSizeLimit}(n) = \mathsf{DBSize}(n - \mathsf{T4DF}(n))$
$\mathsf{SaturationRadius} = 4/3$	$\mathsf{DBSize}(d) = 2.77 \times (4/3)^{(d/2)}$
$\mathsf{DualHashMinDim} = 106$	$\mathsf{DualHashDim} = 24, \mathsf{DualHashVecs} = 32$
$\mathsf{PreferLeftInsert} = 1.2$	$\mathsf{DownSieve} = \mathsf{True}$
$\mathsf{MultiBucket} = 2$	$\mathsf{Sieve} =$ `triple_gpu`

Figure 4.12: Main parameters used for the record runs.

We found that with DownSieve on, a larger PreferLeftInsert is more benificiary. I.e., prefer to insert even a slightly improved b'_i into the basis over a more significantly improved b'_{i+1}.

Another main parameter affecting memory use is the constant factor in database size, normally chosen as 3.2 in G6K [Alb+19]. We opted to reduce this to 2.77, resulting in $\mathsf{DBSize}(d) = 2.77 \times (4/3)^{(d/2)}$ for sieve dimension d, and compensate by also reducing SaturationRatio from .5 to .375.

Additionally, we introduced a database size limit by setting an experimentally-verified target $\mathsf{TD4F}(n) = \lfloor n/\log(n) \rfloor$ for the number of dimensions-for-free, and limiting the database size to $\mathsf{DBSizeLimit}(n) = \mathsf{DBSize}(n - \mathsf{T4DF}(n))$. This means that the database size limit does not affect sieving up to the target dimensions-for-free. However, for unlucky cases, we allow G6K workouts of up to 4 dimensions larger without further increasing the database size. Because `triple_gpu` also considers triples we can be certain that saturation will still be reached.

As discussed before, we use DualHash lifting: starting from a sieving dimension of 106 in the filter context $[l - 24, l]$ using 32 dual vectors. To reduce memory overhead from storing buckets and results (before insertion), we set $\mathsf{MultiBucket} = 2$. Our main parameters are summarized in Figure 4.12.

4.6.3 New SVP records

With the parameters tuned as discussed above, we have solved several Darmstadt Lattice 1.05-approxSVP Challenges for lattices with dimension in the range of 158 till 180 (all with seed=0). Details about

(T)D4F = target/actual dimensions for free
MSD = maximum sieving dimension
FLOP = # bucketing + reduction core floating point operations

Dimensions				Norm	Cost		
dim	TD4F	D4F	MSD	Norm/GH	FLOP	Walltime	Mem GiB
158	31	29	129	1.04329	$2^{62.1}$	9h 16m	89
160	31	33	127	1.02302	$2^{61.8}$	8h 24m	88
162	31	31	131	1.04220	$2^{63.2}$	18h 32m	156
164	32	28	136	1.04368	$2^{64,8}$	2d 01h	179
166	32	30	136	1.03969	$2^{64.8}$	2d 01h	234
168	32	31	137	1.04946	$2^{65.3}$	2d 18h	318
170	33	31	139	1.04594	$2^{66.3}$	5d 11h	364
172	33	35	137	1.04582	$2^{65.0}$	2d 09h	364
174	33	35	139	1.04913	$2^{66.3}$	5d 06h	518
176	34	33	143	1.04412	$2^{67.5}$	12d 11h	806
178	34	32	146	1.02725	$2^{68.6}$	22d 18h	1060
180	34	30	150	1.04003	$2^{69.9}$	51d 14h	1443

Machine specification:
2× Intel Xeon Gold 6248 (20C/40T @ 2.5-3.9GHz)
4× Gigabyte RTX 2080 TI (4352C @ 1.5-1.8GHz)
1.5 TiB RAM (2666 MHz)
Average load: 40 CPU threads @ 93%
4 GPUs @ 79%/1530MHz/242Watt

Table 4.1: Darmstadt Lattice 1.05-approxSVP Challenge results

the effort and results for each challenge are presented in Table 4.1.

With a new top record of the 1.05-approxSVP challenges with dimension 180, we improve significantly upon the last record of dimension 155 by [Alb+19]. Note that this last record was achieved on a single large machine with 72 CPU cores in 14 days and 16 hours, where we were able to find an even shorter vector of length $0.9842 \cdot \mathrm{gh}$ in about 5 hours (68× faster). Also we can improve this record from 155 by no less than 21 dimensions by solving lattice 1.05-approxSVP for dimension 176 on our 4-GPU machine in less wall-clock time: 12 days and 11 hours. As proof we present our short vector for Darm-

(68, 33, -261, 11, 101, 354, -48, -398, 196, -84, 217, 319, -137, -157, -29, 304, -14, 312, 28, -240, -347, -6, -153, -35, -214, 67, -565, 91, 365, 382, -168, 152, 30, 42, -12, -14, -230, 54, 304, 51, 398, 380, 76, -111, 437, 374, -554, -171, -90, -92, 564, 32, 217, 60, -107, 475, -290, -326, -224, -218, 27, -271, 12, 200, 463, -365, 119, -431, 92, 450, 58, 183, 342, 82, -144, 77, -95, -62, -245, 171, 169, -106, -330, 236, 194, 41, -84, -297, 567, 58, 553, 279, 260, 140, -141, -30, -183, -448, -112, 45, 135, -260, -261, 1, -105, 507, 105, -414, -161, -9, -337, -287, 431, 92, -91, 350, -376, -75, 11, -249, 119, -172, -351, 410, 97, -320, -270, 223, -287, 97, 235, 242, 279, -222, 384, -95, 501, 317, 167, -130, -103, 441, 424, 25, 187, -128, -9, -90, 328, -107, -132, -81, 2, 94, -326, -109, 465, 49, -30, 345, 125, -114, 909, 180, -5, -112, 190, 182, -65, -291, -83, 445, -68, -318, -18, -732, -241, 246, -34, 299)

Figure 4.13: A solution to the Darmstadt Lattice 1.05-approxSVP Challenge in dimension 180 with seed 0.

		Wattage		Total usage	
dim	time	CPU+GPU only	system	CPU+GPU only	system
155	352 h	560 W	720 W	197 kWh	254 kWh
176	229 h	1268 W	1428 W	379 kWh	427 kWh

Table 4.2: Power use comparison for records of dimension 155 (G6K) and 176 (ours).

stadt Lattice 1.05-approxSVP Challenge dimension 180 with seed 0 in Figure 4.13.

4.6.4 Remarks

Power use

To compare power efficiency of our new record computation for dimension 176 with the previous record computation for dimension 155, we estimated the power use as shown in Table 4.2 as follows. Their dimension 155 computation ran for 352 hours on 4 CPUs (Intel Xeon E7-8860V4) that have a TDP of 140 Watt each. Our dimension 176 computation ran for 299 hours on 2 CPUs (Intel Xeon Gold 6248) with a TDP of 150 Watt each, and 4 GPUs that typically used 242 Watt as measured through the `nvidia-smi` tool. For both systems we approximate other system power usage covering motherboard, RAM and disk as about 160W.

Note in Table 4.2 that while solving the challenge for dimension

176 is about two orders of magnitude harder compared to dimension 155, we spent less than a factor 2 more in electricity.

Memory use

From the measured memory usage in Table 4.1, we estimate that our implementation requires about 416 Bytes per vector including the amortized overheads, for dimensions higher than 137. Hence, sieving up to dimension 146 could still fit within our 1.5 TiB of available RAM, which allowed us to solve the lattice challenge of dimension 180.

CHAPTER 5

The Closest Vector Problem with Preprocessing

This chapter is based on the joint work 'The randomized slicer for CVPP: sharper, faster, smaller, batchier', with Léo Ducas and Thijs Laarhoven, published at PKC 2020.

5.1 Introduction

The Closest Vector Problem (CVP) can be viewed as an inhomogeneous version of the Shortest Vector Problem (SVP). It asks to compute a closest lattice vector to a given target vector in the real span of the lattice. This is also sometimes referred to as *decoding* the target. These two problems are closely related, and just as for SVP, solving CVP efficiently would directly break most lattice-based cryptographic schemes; e.g., decoding an encrypted message, or forging a signature often essentially boils down to recovering a close(st) lattice point. In many cases it is sufficient to solve an approximate version γ-CVP, where one has to compute a lattice vector that lies at most a factor γ further away than a closest one.

CVP is no easier than SVP, there exists an efficient reduction from approx-SVP to approx-CVP that preserves both the lattice dimension and approximation factor [GMSS99]. In fact most lattice problems have such a dimension-preserving reduction to CVP [Mic08; BN09; MV13]. Provable worst-case reductions in the other direction are less straightforward and tight, and often increase both the dimension and approximation factors.

Still from an average-case perspective, most algorithms for SVP can be adapted to work for CVP with about the same (heuristic) complexity. I.e., the current fastest approaches for solving CVP are just as SVP based on lattice sieving [AKS01; BDGL16; Alb+19; DSW21] and lattice enumeration [Kan83; FP85; GNR10; AN17; ANS18], where the former offers a better asymptotic scaling of the time complexity in terms of the lattice dimension, at the cost of an exponentially large memory consumption.

In contrast to SVP however, CVP receives as input, next to a lattice (basis), a target vector. Even though solving a single instance is as hard as SVP, one could wonder if we can amortize the cost by decoding many targets over the same lattice.

The closest vector problem with preprocessing (CVPP). The closest vector problem with preprocessing (CVPP) is a variant of CVP, where the solver is allowed to perform some preprocessing on the lattice at no additional cost, before being given the target vector. Closely related to this is batch-CVP, where many CVP instances on the same lattice are to be solved; if an efficient global preprocessing procedure can be performed using only the lattice as input, and that would help reduce the costs of single CVP instances, then this preprocessing cost can be amortized over many problem instances to obtain a faster algorithm for batch-CVP. This problem of batch-CVP most notably appears in the context of lattice enumeration for solving SVP or CVP, as a fast batch-CVP algorithm would potentially imply faster SVP and CVP algorithms based on a hybrid of enumeration and such a CVPP oracle [GNR10; DLW20a].

Voronoi cells and the iterative slicer. One method for solving CVPP is the iterative slicer by Sommer–Feder–Shalvi [SFS09]. Preprocessing consists of computing a large list of lattice vectors, and a query is

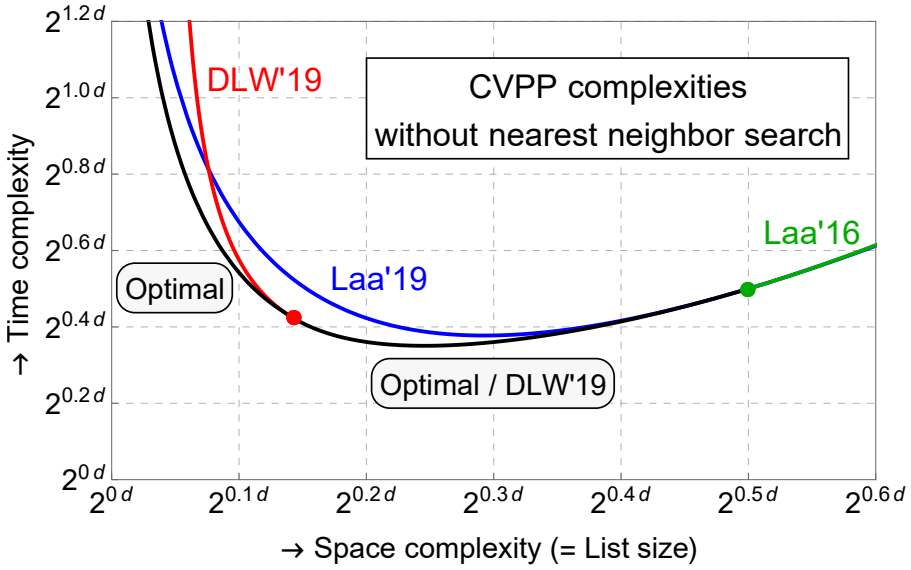


Figure 5.1: Query complexities for solving CVPP without nearest neighbour techniques. The blue curve refers to [Laa19], the red curve to [DLW19], the green curve to [Laa16], and the black curve is the result of our refined analysis. The red point indicates the point where red and black curves merge into one.

processed by "reducing" the target vector $\mathbf{t}$ with this list, i.e. repeatedly translating the target by some lattice vector until the shortest representative $\mathbf{t}'$ in the coset of the target vector is found. The closest lattice vector to $\mathbf{t}$ is then given by $\mathbf{t} - \mathbf{t}'$, which lies at distance $\|\mathbf{t}'\|$ from $\mathbf{t}$. For this method to provably succeed, the preprocessed list needs to contain all $O(2^d)$ so-called Voronoi relevant vectors of the lattice, which together define the boundaries of the Voronoi cell of the lattice. This leads to a $4^{d+o(d)}$ algorithm by bounding the number of reduction steps by $2^{d+o(d)}$ [MV13], which was later improved to an expected time of $2^{d+o(d)}$ by randomizing the algorithm such that the number of expected steps is polynomially bounded [DB15].

Approximate Voronoi cells and the randomized slicer. The large number of Voronoi relevant vectors of a lattice, needed for the iterative slicer to be provably successful, makes the straightforward application of this method impractical and does not result in an improvement over

the best (heuristic) CVP complexities without preprocessing. Therefore we fall back on heuristics to analyse the iterative slicer, as they often better represent the practical complexities of an algorithm than the proven worst-case bounds. Laarhoven [Laa16] proposed to use a smaller preprocessed list of size $2^{d/2+o(d)}$ containing all lattice vectors up to some radius, while heuristically retaining a constant success probability of finding the closest vector with the iterative slicer. Doulgerakis–Laarhoven–De Weger [DLW19] formalized this method in terms of approximate Voronoi cells, and proposed an improvement based on rerandomizations. Rather than hoping to find the solution in one run of the iterative slicer, which would require a preprocessed list of size at least $2^{d/2+o(d)}$, the algorithm uses a smaller list and runs the same reduction procedure many times. Each time starting with a randomly sampled members from the coset of the target vector. The success probability of this randomized slicing procedure, which depends on the size of the list, determines how often it has to be restarted, and thus plays an important role in the eventual time complexity of the algorithm. Doulgerakis–Laarhoven–De Weger [DLW19] obtained a heuristic lower bound on the success probability of this randomized slicer, which Laarhoven [Laa19] later improved upon in the low-memory regime. In particular, these two bounds cross each-other, which suggests that there should be a better (sharp) global bound. Thus the question remains open, what is the actual asymptotic success probability of the randomized slicing procedure, and therefore what is the actual asymptotic time complexity of the current state-of-the-art heuristic method for solving CVPP.

5.1.1 Contributions

Success probability asymptotics via random walks

The main contribution of this work is to answer the central open problem resulting from the approximate Voronoi cells line of work – giving sharp (heuristic) asymptotics on the success probability of the randomized slicer when using a list of the $\alpha^{d+o(d)}$ shortest lattice vectors. To find these sharp bounds, in Section 5.3 we show how to model the flow of the algorithm as a random walk on the coset of the lattice corresponding to the target vector, and we heuristically characterise transition probabilities between different states in this infinite graph.

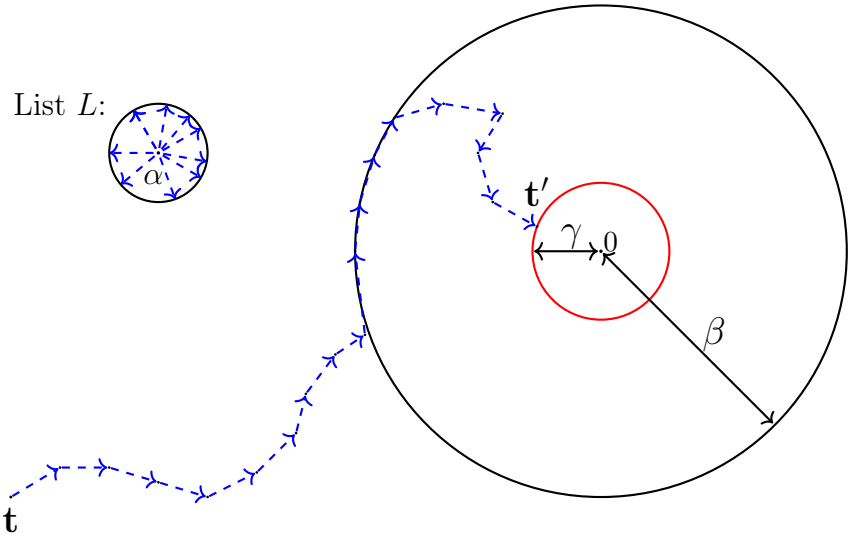


Figure 5.2: The iterative slicer as a random walk over the coset $\mathbf{t} + \mathcal{L}$ using the list of lattice vectors $L = \mathcal{L} \cap \mathcal{B}(\vec{0}, \alpha)$.

We show that the transition probabilities only depend on the norm of coset representatives and therefore the graph can be simplified to an interval $(0, \infty)$ representing the norm. The aforementioned problem of finding the success probability of the slicer then translates to: what is the probability in this graph of starting from a given initial norm and ending at any target norm of at most γ? From [DLW19] we know that we almost always reach a state of norm at most some $\beta = f(\alpha) \geq \gamma$ – reaching this state occurs with probability at least $1/\operatorname{poly}(d)$. However, reaching a state $\beta' < \beta$ occurs only with exponentially small probability $2^{-\Theta(d)}$. Now, whereas the analysis of [DLW19] can be interpreted as lower-bounding the success probability by attempting to reach the target norm in a single step after reaching radius β, we are interested in the arbitrary-step transition probabilities from β to at most γ, so as to obtain sharp bounds.

As every path in our graph from β to γ has an exponentially small probability in d, the total success probability is dominated by that of the highest probable path for large d; which after an appropriate log-transform boils down to a shortest path in a graph. Therefore obtaining the success probability of the randomized slicer is reduced

to determining a shortest path in this infinite graph. We show in Section 5.4 how we can approximately compute this shortest path numerically, using a suitably dense discretization of the search space or using convex optimization. In Section 5.5 we go a step further by proving an exact analytic expression of the shortest path, which results in sharp asymptotics on the success probability of the randomized slicer for the general case of approx-CVP.

Heuristic claim 74 (Success probability of the randomized slicer). *Given a list L of the $\alpha^{d+o(d)}$ shortest lattice vectors as input, the success probability p_{α^2,γ^2} of one iteration of the randomized slicer for γ-CVPP equals:*

$$p_{\alpha^2,\gamma^2} = \prod_{i=1}^{n} \left(\alpha^2 - \frac{(\alpha^2 + x_{i-1} - x_i)^2}{2x_{i-1}} \right)^{d/2+o(d)}$$

with n defined by equation (5.1) on page 154, and x_i as in Definition 81 depending only on α and γ.

Running the randomized slicer for $O(p_{\alpha^2,\gamma^2}^{-1})$ iterations, we expect to solve γ-CVPP with constant probability. Together with a (naive) linear search over the preprocessed list, this directly leads to explicit time and space complexities for a plain version of the randomized slicer for solving CVPP, described in Figure 5.1. When using a large list of size at least $2^{0.1436d+o(d)}$ from the preprocessing phase of CVPP, we derive that one step is optimal, thus obtaining the same asymptotic complexity as [DLW19]. When using less than $2^{0.1436d+o(d)}$ memory we gradually see an increase in the optimal number of steps in the shortest path, resulting in ever-increasing improvements in the resulting asymptotic complexities for CVPP as compared to [DLW19].

Extensions in full work

In the full version[1] of this work we exhibit the versatility of the random walk model. For example we show how to adapt the graph slightly to analyse the success probability of the iterative slicer for the BDD-variant of CVP, where the target lies unusually close to the lattice. In addition, by using a similar random walk model, we derive a tight

[1] Full version: https://eprint.iacr.org/2020/120.pdf

probability analysis of graph-based Nearest Neighbour Search techniques, and the resulting time-memory trade-off when applied to sieving algorithms, improving on previous upper bounds.

Next to the versatility we also further improve the time-memory trade-off by using the asymptotic best bucketing technique [BDGL16] as explained in Section 3.3. While in previous works [Laa16; Laa19] these bucketing techniques came at the cost of large memory overheads, we show how this can be achieved efficiently with only a small memory overhead for CVPP, and no overhead for batch-CVPP.

5.1.2 Working heuristics

The first heuristic which we use is the commonly used Gaussian heuristic, which predicts the number of lattice vectors and their density within certain regions based on the lattice volume. Its use for analysing sieve-type algorithms is well established [NV08; BGJ15; BDGL16; Laa16] and seems consistent with various experiments conducted in the past.

The second heuristic assumption we use is also central in previous work on the randomized iterative slicer [DLW19; Laa19], and consists of assuming that the input target can be randomized, yielding essentially independent experiments each time we randomize the input over the coset of the target vector. Practical experiments from [DLW19] seem to support this assumption.

5.2 The randomized iterative slicer

The iterative slicer (Algorithm 4) is a simple but effective algorithm that aims to solve the closest vector problem or variants thereof. It starts with a preprocessing stage that only depends on the input lattice, after which it can solve multiple problem instances over this fixed lattice, amortizing the cost of preprocessing.

5.2.1 The algorithm

The preprocessing consists of finding and storing a list $L \subset \mathcal{L}$ of lattice vectors. Then given a target point $\mathbf{t} \in \mathbb{R}^d$ the iterative slicer tries to reduce the target $\mathbf{t}$ by the list L to some smaller representative

Algorithm 4: The iterative slicer of [SFS09]

Input: A target vector $\mathbf{t} \in \mathbb{R}^d$, a list $L \subset \mathcal{L}$.
Output: A close vector $\mathbf{v} \in \mathcal{L}$ to $\mathbf{t}$.

Function `IterativeSlicer(`L, $\mathbf{t}$`):`
  $\mathbf{t}_0 \leftarrow \mathbf{t}$;
  for $i \leftarrow 0, 1, 2, \ldots$ **do**
    $\mathbf{t}_{i+1} \leftarrow \min_{\mathbf{v} \in L \cup \{\mathbf{0}\}} \{\mathbf{t}_i - \mathbf{v}\}$;
    if $\mathbf{t}_{i+1} = \mathbf{t}_i$ **then return** $\mathbf{t}_0 - \mathbf{t}_i$;
  end

$\mathbf{t}' \in \mathbf{t} + \mathcal{L}$ in the same coset of the lattice. This is repeated until the reduction fails or until the algorithm succeeds, i.e., when $\|\mathbf{t}'\| \leq \gamma \cdot \mathrm{dist}(\mathcal{L}, \mathbf{t})$. We then obtain the lattice point $\mathbf{t} - \mathbf{t}'$ that lies at distance at most $\gamma \cdot \mathrm{dist}(\mathcal{L}, \mathbf{t})$ to $\mathbf{t}$. Observe that $\mathbf{t}'$ is a shortest vector in $\mathbf{t} + \mathcal{L}$ if and only if $\mathbf{v} = \mathbf{t} - \mathbf{t}' \in \mathcal{L}$ is a closest lattice vector to $\mathbf{t}$.

To provably guarantee that the closest vector is found we need the preprocessed list L to contain all the Voronoi-relevant vectors; the vectors that define the Voronoi cell of the lattice. However most lattices have $O(2^d)$ relevant vectors, which is too much to be practically viable. Under the Gaussian heuristic, Laarhoven [Laa16] showed that $2^{d/2+o(d)}$ short vectors commonly suffice for the iterative slicer to succeed with high probability, but this number of vectors is still too large for any practical algorithm. The randomized slicer (Algorithm 5) of Doulgerakis–Laarhoven–De Weger [DLW19] attempts to overcome this large list requirement by using a smaller preprocessed list together with rerandomizations to obtain a reasonable probability of finding a close vector – the success probability of one run of the iterative slicer might be small, but repeating the algorithm many times using randomized inputs from $\mathbf{t} + \mathcal{L}$, the algorithm then succeeds with high probability, without requiring a larger preprocessed list.

Because we can only use a list of limited size, one can ask the question which lattice vectors to include in this list L. Later in the analysis it will become clear that short vectors are more useful to reduce a random target, so it is natural to let L consist of all short vectors up to some radius. Let $\alpha > \lambda_1(\mathcal{L})$ be this radius and denote

Algorithm 5: The randomized iterative slicer of [DLW19]

Input: A target vector $\mathbf{t} \in \mathbb{R}^d$, a list $L \subset \mathcal{L}$, a target distance $\gamma \in \mathbb{R}$.

Output: A close vector $\mathbf{v} \in \mathcal{L}$, s.t. $\|\mathbf{t} - \mathbf{v}\| \leq \gamma$.

Function `RandomizedSlicer`(L, $\mathbf{t}$, γ)**:**
  repeat
    $\mathbf{t}' \leftarrow$ `Sample`($\mathbf{t} + \mathcal{L}$);
    $\mathbf{v} \leftarrow$ `IterativeSlicer`(L, $\mathbf{t}'$);
  until $\|\mathbf{t}' - \mathbf{v}\| \leq \gamma$;
  return $\mathbf{v} + (\mathbf{t} - \mathbf{t}')$;

its square by $a := \alpha^2$. The preprocessed list then becomes

$$L_a := \{\mathbf{x} \in \mathcal{L} : \|\mathbf{x}\|^2 \leq a\}.$$

Throughout this work let us assume that the lattice $\mathcal{L}$ is normalized such that $\lambda_1(\mathcal{L}) \approx \mathrm{gh}(\mathcal{L}) = 1$. Then under the Gaussian heuristic the list consists of $|L_a| = \alpha^{d+o(d)}$ lattice points, which determines (ignoring nearest neighbour data structures) the space complexity of the algorithm and also determines the time complexity of each iteration.

5.2.2 Average-case CVPP

Recall from Section 2.3 that the approximate γ-CVP problem for $\gamma \geq 1$ asks you, given any target $\mathbf{t}$, to find a lattice point at distance at most $\gamma \cdot \mathrm{dist}(\mathcal{L}, \mathbf{t})$ from the lattice. Throughout this chapter we will write $c := \gamma^2$ for the squared approximation factor, where $c = 1$ corresponds to exact CVPP. We will only consider the average-case version of this problem, where the lattice follows the Gaussian Heuristic and the target (coset) is uniformly random $\mathbf{t} + \mathcal{L} \sim \mathcal{U}(\mathbb{R}^d/\mathcal{L})$. Under the Gaussian Heuristic this implies that $\mathrm{dist}(\mathcal{L}, \mathbf{t}) \to \mathrm{gh}(\mathcal{L}) = 1$ as $d \to \infty$, and to simplify the analysis we will simply assume that $\mathrm{dist}(\mathcal{L}, \mathbf{t}) = 1$.

The preprocessing variant γ-CVPP additionally allows to do any kind of preprocessing given only a description of the lattice $\mathcal{L}$ (and not the target $\mathbf{t}$). The size of the final preprocessing advice is counted in the eventual space complexity of the CVPP algorithm.

5.2.3 Success probability

The iterative slicer is not guaranteed to succeed as the list does not contain all relevant vectors. However, suppose that the iterative slicer has a success probability of $p_{a,c}$ given a random target. It is clear that having a larger preprocessed list increases the success probability, but in general it is hard to concretely analyse the success probability for a certain list. Heuristically however, we can do such an analysis.

Under the Gaussian heuristic we can derive bounds on $p_{a,c}$, as was first done by [DLW19]. They obtained the following two regimes for the success probability as $d \to \infty$:

- For $a \geq 2c - 2\sqrt{c^2 - c}$ we have $p_{a,c} \to 1$.
- For $a < 2c - 2\sqrt{c^2 - c}$ we have $p_{a,c} = \exp(-C \cdot d + o(d))$ for a constant $C = C(a, c) > 0$ depending on a, c.

The second case above illustrates that for a small list size the algorithm needs to be repeated a large number of times with fresh targets to guarantee a high success probability. This gives us the randomized slicer algorithm. To obtain a fresh target the idea is to sample randomly a not too large element from the coset $\mathbf{t} + \mathcal{L}$, and assume that the reduction of this new target is independent from the initial one. Experiments from [DLW19] suggest that this is a valid assumption to make, and given a success probability $p_{a,c} \ll 1$ it is enough to repeat the algorithm $\Theta(1/p_{a,c})$ times to find the closest lattice point. However this success probability in the case $a < 2c - 2\sqrt{c^2 - c}$ is not yet fully understood. Two heuristic lower bounds [DLW19; Laa19] are known and are shown in Figure 5.3. None of these lower bounds fully dominates the other, which implies that neither of the bounds is sharp. In the remainder of this work we consider this case where we have a small success probability.

5.3 The random walk model

To interpret the iterative slicer algorithm as a random walk we first look at the probability that a target $\mathbf{t}$ is reduced by a random lattice point from the preprocessed list L_a. By the Gaussian heuristic this lattice point is distributed uniformly over the ball of radius $\sqrt{a}$. To

reduce the squared norm $\|\mathbf{t}\|^2$ from x to at most $y \in [(\sqrt{x}-\sqrt{a})^2, x]$ by some $\mathbf{v}$ with $\|\mathbf{v}\|^2 = a$, the inner product between $\mathbf{t}$ and $\mathbf{v}$ must satisfy:

$$\langle \mathbf{t}, \mathbf{v} \rangle \leq -(a+x-y)/2.$$

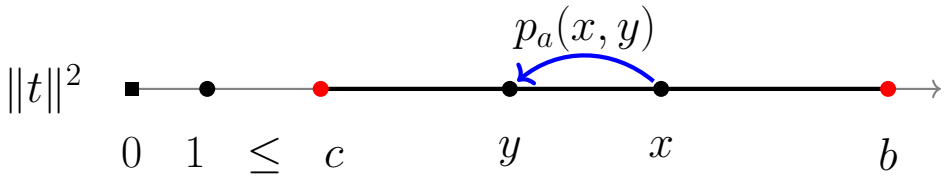
Using the asymtotic formulas in Lemma 46 for the volume of a spherical cap we then deduce the following probability:

$$\Pr_{\mathbf{v} \in \sqrt{a} \cdot \mathcal{B}^d} \left(\|\mathbf{t}+\mathbf{v}\|^2 \leq y \;\middle|\; \|\mathbf{t}\|^2 = x \right) = \left(1 - \frac{(a+x-y)^2}{4ax}\right)^{d/2+o(d)}.$$

Clearly any reduction of $\mathbf{t}$ from a squared norm of x to a squared norm strictly less than $(\sqrt{x}-\sqrt{a})^2$ is unreachable by a vector in $\sqrt{a} \cdot \mathcal{B}^d$. The probability that the target norm is successfully reduced to a value less than $\|t\|^2$ decreases in a and thus we prefer to have short vectors in our list. As the list L_a does not contain just one, but $a^{d/2}$ lattice vectors we obtain the following heuristic reduction probability for a single iteration of the iterative slicer:

$$\begin{aligned} &\Pr\left(\exists \mathbf{v} \in L_a : \|\mathbf{t}+\mathbf{v}\|^2 \leq y \;\middle|\; \|\mathbf{t}\|^2 = x \right)^{2/d} \\ &\quad \to \min\left\{1, a \cdot \left(1 - \frac{(a+x-y)^2}{4ax}\right)\right\} \end{aligned}$$

as $d \to \infty$. The reduction probability takes the form $\exp(-Cd+o(d))$ for some constant $C \geq 0$ that only depends on a, x and y. As we are interested in the limit behaviour as $d \to \infty$ we focus our attention to this base $\exp(-C)$, which we call the base-probability of this reduction and denote it by $p_a(x,y)$. Although these transition probabilities represent a reduction to any square norm $\leq y$, they should asymptotically be interpreted as a reduction to $\approx y$, as for any fixed $\varepsilon > 0$ we have that $p_a(x, y-\epsilon)^d / p_a(x,y)^d = 2^{-\Theta(d)} \to 0$ as $d \to \infty$. If $\|\mathbf{t}\|^2 = x$ is large enough we can almost certainly find a lattice point in L_a that reduces this norm successfully. In fact a simple computation shows that this is the case for any $x > b := a^2/(4a-4)$ as $d \to \infty$. So in our analysis we can assume that our target is already reduced to square norm b, and the interesting part is how probable the remaining reduction from b to c is.



Definition 75 (Transition base-probability). *Let $a, b \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a - 4)$. The transition base-probability $p_a(x, y)$ to reduce $\|\mathbf{t}\|^2$ from x to y is given by*

$$\begin{aligned} p_a : S_a &\to (0, 1], \\ (x, y) &\mapsto \left(a - \frac{(a + x - y)^2}{4x}\right)^{1/2}, \end{aligned}$$

with $S_a = \{(x, y) \in [1, b]^2 : b \geq x \geq y \text{ and } (\sqrt{x} - \sqrt{a})^2 < y\}$ the allowed transitions.

Proof. We have to show that indeed $p_a(x, y) \in (0, 1]$ for $(x, y) \in S_a$, and the given constraints on a and b. For the lower bound note that expanding $(\sqrt{x} - \sqrt{a})^2 < y$, and rewriting gives $a + x - y < 2\sqrt{a}\sqrt{x}$. Because $y \geq x$, and $a > 1$, both sides are positive and thus squaring gives $(a + x - y)^2 < 4ax$, and in particular

$$a - \frac{(a + x - y)^2}{4x} = \frac{4ax - (a + x - y)^2}{4x} > 0.$$

For the upper bound we use that $x \leq b = a^2/(4a - 4)$, and $x - y \geq 0$ to show that

$$4ax - 4x \leq a^2 \leq (a + x - y)^2,$$

from which it follows that

$$\frac{4ax - (a + x - y)^2}{4x} \leq \frac{4x}{4x} = 1.$$

□

Using the above reduction probabilities we model the iterative slicer as a random walk over an infinite graph where the nodes are given by the interval $[1, b]$ such that each node $x_i \in [1, b]$ is associated with the squared norm $\|\mathbf{t}_i\|^2$ of the partly reduced target. Note that

each possible random walk $b = x_0 \to x_1 \to \cdots \to x_n = c$ has a certain success probability. Assuming the different steps are independent this success probability is just the product of the individual reduction probabilities. For an n-step path we could split our list L_a in n parts, one for each step, to obtain this independence without changing the asymptotic size of these lists. Again this success probability is of the form $\exp(-Cd + o(d))$ for some constant $C \geq 0$ that only depends on $x_0, \ldots, x_n$ and a.

Definition 76 (Path). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. Let $n \geq 1$ be an integer. We denote an n-step path on the interval $[1, b]$ by $x_0 \to x_1 \to \cdots \to x_n$ for $x_i \in [1, b]$. We define the set of all n-step paths with positive probability from b to c by:*

$$S_a[b \xrightarrow{n} c] := \{(b = x_0, x_1, \ldots, x_n = c) \in \mathbb{R}^{n+1} : \forall i \ (x_{i-1}, x_i) \in S_a\}.$$

The transition base-probability of such a path is given by

$$P_a[b \xrightarrow{n} c] : S_a[b \xrightarrow{n} c] \to (0, 1],$$
$$\mathbf{x} \mapsto \prod_{i=1}^{n} p_a(x_{i-1}, x_i).$$

The success probability of reaching c from b is determined by the total probability of all successful paths. Note that all these paths have some probability of the form $\exp(-Cd+o(d))$ and thus the probability for the path with the smallest $C \geq 0$ will dominate all other paths for large d. As a result, almost all successful walks will go via the highest probable path, i.e., the one with the highest base-probability. After applying a log-transform this becomes equivalent to finding the shortest path in a weighted graph.

Definition 77 (Transition graph). *Let $a, b \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4)$. Let $V = [1, b]$ and $E = [1, b]^2$ be an infinite graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ given by:*

$$w(x, y) = \begin{cases} -\log p_a(x, y), & \text{if } (x, y) \in S_a; \\ \infty, & \text{otherwise.} \end{cases}$$

One can associate n-step paths in this graph from b to $c \in [1, b)$ with the space $S_a[b \xrightarrow{n} c]$. The length of a path $\mathbf{x} \in S_a[b \xrightarrow{n} c]$ is denoted by $\ell_a[b \xrightarrow{n} c](\mathbf{x})$ and the shortest path length by

$$\ell_{a,opt}[b \to c] = \inf_{n \in \mathbb{Z}_{\geq 1}} \inf_{\mathbf{x} \in S_a[b \xrightarrow{n} c]} \ell_a[b \xrightarrow{n} c](\mathbf{x}).$$

Obtaining the success probability in this model therefore becomes equivalent to obtaining the length of the shortest path $\ell_{a,\mathrm{opt}}[b \to c]$ as we have $P_a[b \xrightarrow{n} c](\mathbf{x}) = \exp(-\ell_a[b \xrightarrow{n} c](\mathbf{x}))$.

5.4 Numerical approximations

We reduced the problem of obtaining the success probability of the iterative slicer to the search of a shortest path in a specially constructed weighted infinite graph. We might not always be able to find an exact solution in the input variables to the length of the shortest path. However for fixed parameters we can always try to numerically approximate the success probability, by approximating the shortest path in our infinite graph. We present two fairly standard methods for doing so. The first method first discretizes the infinite graph and then determines the shortest path using standard algorithms such as Dijkstra's algorithm [Dij59]. The second method uses the fact that the weight function $w_a : S_a \to \mathbb{R}_{\geq 0}$ is convex. A fact that we will later use to derive an explicit solution.

5.4.1 Discretization

A natural way to approximate the shortest path in an infinite graph is to first discretize to a finite subgraph with k vertices. Then one can determine the shortest path in this subgraph using standard methods to obtain a short path in the infinite graph. The details of this approach are shown in Algorithm 6.

Using any optimized Dijkstra implementation the time and space complexity of Algorithm 6 is $O(|E_d| + |V_d| \log |V_d|) = O(k^2)$. In general this method gives a lower bound on the success probability for any fixed a and c. Because the weight function $w_a : S_a \to \mathbb{R}_{\geq 0}$ is continuous Algorithm 6 converges to the optimal path length as $k \to \infty$.

Algorithm 6: A discretized shortest path algorithm

Input: Parameters $1 < a < 2, b = a^2/(4a-4)$ and $c \in [1, b)$ describing the graph, and a discretization value k.

Output: A shortest path on the discretized graph from b to c.

Function `DiscretizedDijkstra(`a, b, c, k`):`

- Compute $V_d = \{c + \frac{i \cdot (b-c)}{k} : i = 0, \ldots, k\}$;
- Compute $E_d = \{(x, y) \in V_d^2 \cap S_a\}$ and the weights $w_a(x, y)$;
- Compute shortest path on $G_d = (V_d, E_d)$ from b to c using [Dij59].

The C++ implementation of this method used for the experiments is made available on Github[2].

For this method to converge to the shortest path in the full graph we only need a continuous weight function. Furthermore the number of steps does not have to be specified a priori. The high memory usage of $O(k^2)$ could limit the fineness of our discretization. To circumvent this we can generate the edges (and their weight) on the fly when needed, which reduces the memory consumption to $O(k)$.

5.4.2 Convex optimization

Where the first method only needed $w_a : S_a \to \mathbb{R}_{\geq 0}$ to be continuous, the second method makes use of the convexity of this function.

Lemma 78 (Convexity of S_a and w_a). *Let $1 < a < 2$, the set of allowed transitions S_a is convex and the weight function w_a is strictly convex on S_a.*

Proof. The convexity of $S_a = \{(x, y) \in [1, b]^2 : b \geq x \geq y \text{ and } (\sqrt{x} - \sqrt{a})^2 < y\}$ follows immediately from the convexity of the function $f(x) = (\sqrt{x} - \sqrt{a})^2$ on $[0, \infty)$. Remember that for $(x, y) \in S_a$

$$w_a(x, y) = -\log p_a(x, y) = -\frac{1}{2} \log \left(a - \frac{(a + x - y)^2}{4x} \right),$$

[2]Implementation: https://github.com/WvanWoerden/randomized-slicer

and thus we have

$$\frac{d^2}{dx^2}w_a(x,y) = \frac{8xp_a(x,y)^2 + (4a - 2(a+x-y))^2 - 16p_a(x,y)^4}{32x^2p_a(x,y)^4},$$
$$\frac{d}{dy}\frac{d}{dx}w_a(x,y) = \frac{-8xp_a(x,y)^2 + (4a - 2(a+x-y)) \cdot 2(a+x-y)}{32x^2p_a(x,y)^4},$$
$$\frac{d^2}{dy^2}w_a(x,y) = \frac{8xp_a(x,y)^2 + 4(a+x-y)^2}{32x^2p_a(x,y)^4}.$$

As $p_a(x,y) > 0$ and $a + x - y \geq a > 0$ for $(x,y) \in S_a$ we have $\frac{d^2}{dy^2}w_a(x,y) > 0$. We consider the Hessian H of w_a. Computing the determinant gives:

$$\det(H) = \frac{2(a+x-y)^4 \cdot (4ax - (a+x-y)^2)}{1024x^6p_a(x,y)^8}$$

and we can conclude that $\det(H) > 0$ from the fact that $4ax - (a + x - y)^2 > 0$ and $(a + x - y)^4 > 0$ for $(x,y) \in S_a$. So H is positive definite, which makes w_a strictly convex on S_a. □

Corollary 79 (Convexity of $S_a[b \xrightarrow{n} c]$ and $\ell_a[b \xrightarrow{n} c]$). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. For any (fixed) integer $n \geq 1$, the space of n-step paths $S_a[b \xrightarrow{n} c]$ is convex and the length function $\ell_a[b \xrightarrow{n} c]$ is strictly convex on $S_a[b \xrightarrow{n} c]$.*

Proof. The convexity of $S_a[b \xrightarrow{n} c]$ follows immediately from that of S_a. Note that $\ell_a[b \xrightarrow{n} c](\mathbf{x}) = \sum_{i=1}^{n} w_a(x_{i-1}, x_i)$ and thus it is convex as a sum of convex functions. Furthermore, for each variable at least one of these functions is strictly convex and thus the sum is strictly convex. □

So for any fixed $n \geq 1$ we can use convex optimization to numerically determine the optimal path of n steps. In fact, because of the strict convexity, we know that this optimal path of n steps (if it exists) is unique. However the question remains what the optimal number of steps is, i.e., for which n we should run the convex optimization algorithm. We might miss the optimal path if we do not guess the optimal number of steps correctly. Luckily because $w_a(b,b) = 0$ by definition, we can increase n without being afraid to skip some optimal path.

Lemma 80 (Longer paths are not worse). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. If $\ell_a[b \xrightarrow{n} c]$ and $\ell_a[b \xrightarrow{n+k} c]$ for $n, k \geq 0$ both attain a minimum, then*

$$\min_{\mathbf{x} \in S_a[b \xrightarrow{n} c]} \ell_a[b \xrightarrow{n} c](\mathbf{x}) \geq \min_{\mathbf{x} \in S_a[b \xrightarrow{n+k} c]} \ell_a[b \xrightarrow{n+k} c](\mathbf{x}).$$

Proof. Suppose $\ell_a[b \xrightarrow{n} c]$ attains its minimum at $\mathbf{y} = (b = y_0, y_1, \ldots, y_n = c) \in S_a[b \xrightarrow{n} c]$. Using that $w_a(b, b) = 0$ we get that:

$$\begin{aligned}\min_{\mathbf{x} \in S_a[b \xrightarrow{n+k} c]} \ell_a[b \xrightarrow{n+k} c](\mathbf{x}) &\leq \ell_a[b \xrightarrow{n+k} c](b, \ldots, b = y_0, \ldots, y_n = c) \\ &= k \cdot w_a(b, b) + \ell_a[b \xrightarrow{n} c](\mathbf{y}) \\ &= \ell_a[b \xrightarrow{n} c](\mathbf{y}).\end{aligned}$$

This completes the proof. □

So increasing n can only improve the optimal result. When running a numerical convex optimization algorithm one could start with a somewhat small n and increase it (e.g., double it) until the result does not improve any more.

5.4.3 Numerical results

We ran both numerical algorithms and got similar results. Running the convex optimization algorithm gave better results for small $a = 1+\varepsilon$ as the fineness of the discretization is not enough to represent the almost shortest paths in this regime. This is easily explained as $b \approx \frac{1}{4\varepsilon}$ and thus for fixed c the distance between b and c, i.e., the interval to be covered by the discretization quickly grows as $\varepsilon \to 0$.

The new lower bound that we obtained numerically for exact CVPP ($c = 1$) is shown in Figure 5.3. For $\alpha \leq 1.1047$ we observe that the new lower bound is strictly better than the two previous lower bounds. For $\alpha > 1.1047$ the new lower bound is identical to the lower bound from [DLW19]. Taking a closer look at the short paths we obtained numerically we see that $\alpha \approx 1.1047$ is exactly the moment where this path switches from a single step to at least 2 steps. This makes sense as in our model the lower bound from [DLW19] can be interpreted as

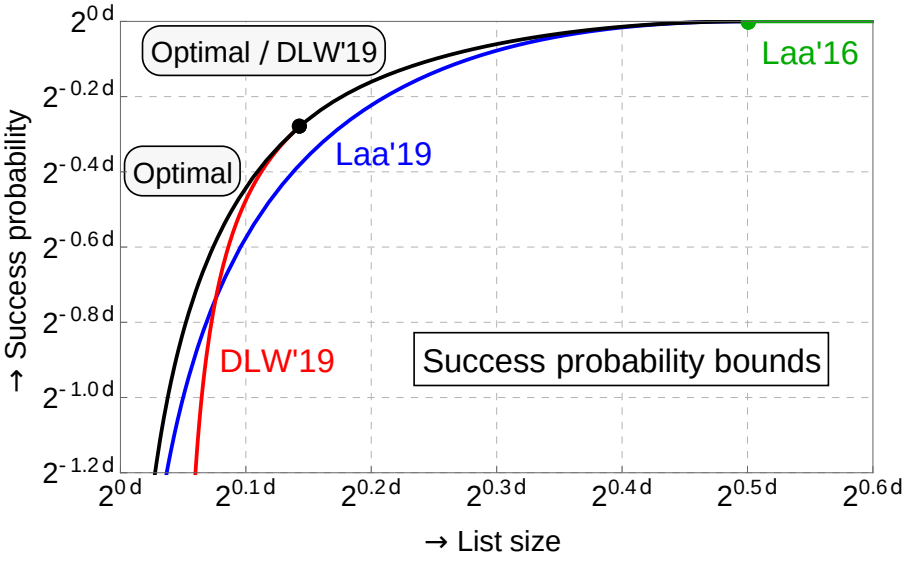


Figure 5.3: Lower bounds on success probability of the iterative slicer for CVPP ($c = 1$) computed with a discretization parameter of $k = 5000$.

a 'single step' analysis. This also explains the asymptote for this lower bound as for $\alpha \leq 1.0340$ it is not possible to walk from b to $c = 1$ in a single step.

When inspecting these short paths $b = x_0 \to x_1 \to \cdots \to x_n = c$ further we observed an almost perfect fit with a quadratic formula $x_i = u \cdot i^2 + v \cdot i + b$ for some constants u, v. In the next section we show how we use this to obtain an exact analytic solution for the shortest path.

5.5 An exact solution

In order to determine an exact solution of the shortest path, and thus an exact solution of the success probability of the iterative slicer we use some observations from the numerical results. Due to Corollary 79 we know that for any fixed $n \geq 1$ our minimization problem is strictly convex. As a result there can be at most one local minimum which, if it exists, is immediately also the unique global minimum.

What remains is to explicitly construct such a local minimum, after which we only have to optimize over n. We recall from Section 5.4.3 that the optimal path $x_0 \to \cdots \to x_n$ seems to take the shape $x_i = u \cdot i^2 + v \cdot i + b$ with $x_n = c$. So for our construction we assume this shape, which reduces the problem to determining the constants u, v. Furthermore, as we are trying to construct a local minimum, we assume that all partial derivatives in the non-constant variables are equal to 0. This gives enough restrictions to obtain a verbose, but explicit solution.

Definition 81 (Explicit construction). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$, and let $n \geq 1$ be an integer. We define*

$$x_i := u_a[b \xrightarrow{n} c] \cdot i^2 + v_a[b \xrightarrow{n} c] \cdot i + b,$$

with $u_a[b \xrightarrow{1} c] := 0$, $v_a[b \xrightarrow{1} c] := c - b$ and for $n \geq 2$:

$$u_a[b \xrightarrow{n} c] := \frac{(b+c-a)n - \sqrt{(an^2-(b+c))^2 + 4bc(n^2-1)}}{n^3-n},$$
$$v_a[b \xrightarrow{n} c] := \frac{(a-2b)n^2 + (b-c) + \sqrt{(an^2-(b+c))^2 + 4bc(n^2-1)}n}{n^3-n}.$$

Lemma 82. *Let a, b, c, n and x_i be as in Definition 81. By construction we have $x_n = c$ and*

$$\frac{\partial}{\partial x_i} \sum_{j=1}^{n} -\log p_a(x_{j-1}, x_j) = 0$$

for all $i \in \{1, \ldots, n-1\}$.

Proof. Note that the partial derivative constraints can be reduced to the single constraint $\frac{\partial}{\partial x_i}(-\log p_a(x_{i-1}, x_i) - \log p_a(x_i, x_{i+1})) = 0$ for a symbolic i. Together with the constraint $x_n = c$ one can solve for u, v in $x_i = u \cdot i^2 + v \cdot i + b$. For a symbolic verification see the Sage script in Appendix A of the full version[3]. □

What remains is to show that the explicit construction indeed gives a valid path, i.e., one that is in the domain $S_a[b \xrightarrow{n} c]$. An example of

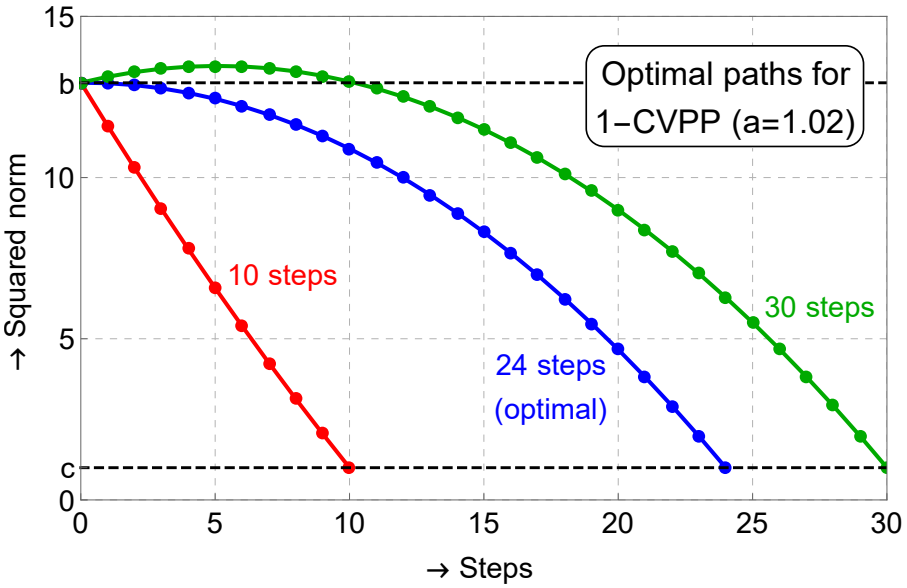


Figure 5.4: Some examples of the constructed paths in Definition 81 for $a = 1.02, c = 1$.

how these constructed paths look are given in Figure 5.4. We observe that if n becomes too large these constructed paths are invalid as they walk outside the interval $[c, b]$. This is an artefact of our simplification that $w_a(x, y) = -\log p_a(x, y)$ which does not hold for $(x, y) \notin S_a$. We can still ask the question for which n this construction is actually valid.

Lemma 83 (Valid constructions). *Let a, b, c be as in Definition 81. Let $n \geq 1$ be an integer such that $\frac{b-c}{a} \leq n < \frac{1}{2} + \frac{\sqrt{(4b-a)^2 - 8(2b-a)c}}{2a}$ and*

$$x_i = u_a[b \xrightarrow{n} c] \cdot i^2 + v_a[b \xrightarrow{n} c] \cdot i + b.$$

Then $\mathbf{x} = (x_0, \ldots, x_n) \in S_a[b \xrightarrow{n} c]$ and $\mathbf{x}$ is the unique minimum of $\ell_a[b \xrightarrow{n} c]$.

Proof. We have to check that $\mathbf{x}$ satisfies the two conditions

$$x_{i-1} \geq x_i \qquad \text{and} \qquad (\sqrt{x_{i-1}} - \sqrt{a})^2 < x_i,$$

[3]Full version: https://eprint.iacr.org/2020/120.pdf

for all $i \in \{1, \dots, n\}$. Note that for $n = 0$ we must have $b = c$ and the statement becomes trivial. For $n = 1$ we have $\mathbf{x} = (b, c)$ and the conditions follows from $0 \le b - c \le na \le a$. So we can assume that $n \ge 2$. First we rewrite $u_a[b \overset{n}{\to} c]$ to:

$$u_a[b \overset{n}{\to} c] = \frac{(b+c-a)n - \sqrt{((b+c-a)n)^2 + (a^2n^2 - (b-c)^2)(n^2-1)}}{n^3 - n},$$

which makes it clear that $u_a[b \overset{n}{\to} c] \le 0$ when $an \ge b - c$. As a result the differences

$$x_{i-1} - x_i = (1 - 2i) \cdot u_a[b \overset{n}{\to} c] - v_a[b \overset{n}{\to} c],$$

are increasing in $i \in \{1, \dots, n\}$. Therefore for the first condition it is enough to check that

$$x_0 - x_1 = \frac{(b-c) + (2b-a)n - \sqrt{(an^2 - (b+c))^2 + 4bc(n^2-1)}}{n^2 + n} \ge 0.$$

In fact a solution with $x_0 = x_1 = b$ is not so interesting, so solving for $x_0 - x_1 > 0$ gives for $n \ge 2$ the sufficient condition

$$n < \frac{1}{2} + \frac{\sqrt{(4b-a)^2 - 8(2b-a)c}}{2\,a}.$$

For the second condition we first show the stronger property that $x_{i-1} - x_i \le a$, and again by the increasing differences it is enough to show that $x_{n-1} - x_n \le a$; rewriting gives the following sufficient statement for $n \ge 2$:

$$-an + b - c \le 0.$$

Now we prove that $\sqrt{x_{i-1}} - \sqrt{x_i} < \sqrt{a}$. If $x_{i-1} = x_i$ the condition holds trivially, else $x_{i-1} > x_i$ and we get

$$(\sqrt{x_{i-1}} - \sqrt{x_i})^2 < (\sqrt{x_{i-1}} - \sqrt{x_i})(\sqrt{x_{i-1}} + \sqrt{x_i}) = x_{i-1} - x_i \le a.$$

Note that if $\sqrt{x_{i-1}} - \sqrt{a} \ge 0$, then we can rewrite $\sqrt{x_{i-1}} - \sqrt{x_i} < \sqrt{a}$ and square to obtain the condition $(\sqrt{x_{i-1}} - \sqrt{a})^2 < x_i$. Alternatively, if $\sqrt{x_{i-1}} - \sqrt{a} < 0$, then due to $a < 2$ we obtain $(\sqrt{x_{i-1}} - \sqrt{a})^2 < (1 - \sqrt{2})^2 < 1 \le x_i$.

We conclude that $\mathbf{x} \in S_a[b \overset{n}{\to} c]$, and because we have the equality $\ell_a[b \overset{n}{\to} c](\mathbf{x}) = \sum_{i=1}^{n} -\log p_a(x_{i-1}, x_i)$ on $S_a[b \overset{n}{\to} c]$, the claim that this is a global minimum follows from Definition 81 and Lemma 79. □

So by Lemma 81 there exists some $s \in \mathbb{N}$ such that for all $(b - c)/a \leq n \leq s$ we have an explicit construction for the optimal n-step path. By Lemma 80 we know that of these paths the one with $n = s$ steps must be the shortest. However for $n > s$ our construction did not work and thus we do not know if any shorter path exists. Inspired by Lemma 80 and numerical results we obtain the following alternative exact solution for $n > s$.

Theorem 84 (Optimal arbitrary-step paths). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a - 4) > c \geq 1$. Let n satisfy*

$$n = \left\lceil -\frac{1}{2} + \frac{1}{2a}\sqrt{(4b - a)^2 - 8(2b - a)c} \right\rceil . \tag{5.1}$$

For $k \geq n$ the unique global minimum of $\ell_a[b \xrightarrow{k} c]$ is given by

$$\mathbf{x} = (b, \ldots, b, b = y_0, \ldots, y_n = c) \in S_a[b \xrightarrow{k} c]$$

with $y_i = u_a[b \xrightarrow{n} c] \cdot i^2 + v_a[b \xrightarrow{n} c] \cdot i + b$ and the length is equal to $\ell_a[b \xrightarrow{n} c](\boldsymbol{y})$.

Proof. By Corollary 79 it is enough to show that $\mathbf{x}$ is a local minimum, therefore we check the partial derivatives. For $i > k - n$ we have $\frac{\partial}{\partial x_i}\ell_a[b \xrightarrow{k} c](\mathbf{x}) = \frac{\partial}{\partial x_i}\ell_a[b \xrightarrow{n} c](\mathbf{y}) = 0$ by construction. For $i < k - n$ we have $x_{i-1} = x_i = x_{i+1} = b$, which results in $\frac{\partial}{\partial x_i}\ell_a[b \xrightarrow{k} c](\mathbf{x}) = -\frac{a-1}{2b} < 0$. For the most interesting case $i = k - n$ we need that $n \geq -\frac{1}{2} + \frac{\sqrt{(4b-a)^2-8(2b-a)c}}{2\,a}$. Because as a result we get $y_0 - y_1 \leq \frac{a^2}{2b-a}$, which together with $y_0 - y_1 \leq b - c \leq b - 1$ is precisely enough to show that $\frac{\partial}{\partial x_{k-n}}\ell_a[b \xrightarrow{k} c](\mathbf{x}) \leq 0$.

To conclude let $\boldsymbol{z} \neq \mathbf{x} \in S_a[b \xrightarrow{n} c]$, then by Corollary 79 and using that $z_i - x_i = z_i - b \leq 0$ for all $0 \leq i \leq k - n$ we have:

$$\begin{aligned}
\ell_a[b \xrightarrow{k} c](\boldsymbol{z}) &> \ell_a[b \xrightarrow{k} c](\mathbf{x}) + \langle \boldsymbol{y} - \mathbf{x}, \nabla \ell_a[b \xrightarrow{k} c](\mathbf{x}) \rangle \\
&= \ell_a[b \xrightarrow{k} c](\mathbf{x}) + \sum_{i \leq k-n} (z_i - x_i) \cdot \frac{\partial}{\partial x_i}\ell_a[b \xrightarrow{k} c](\mathbf{x}) \\
&\geq \ell_a[b \xrightarrow{k} c](\mathbf{x}).
\end{aligned}$$

and thus $\mathbf{x}$ is the unique global minimum of $\ell_a[b \xrightarrow{k} c]$. □

Corollary 85 (Optimal minimum-step paths). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. The optimal path in the transition graph in Definition 77 from b to c consists of n steps, with n defined by equation (5.1). The optimal path is of the form $b = x_0 \to x_1 \to \cdots \to x_n = c$ with $x_i = u_a[b \xrightarrow{n} c] \cdot i^2 + v_a[b \xrightarrow{n} c] \cdot i + b$.*

Heuristic claim 86. *Given the optimal path $b = x_0 \to \cdots \to x_n = c$ from Corollary 85, the success probability of the iterative slice algorithm with list size $a^{n/2+o(n)}$ for $\sqrt{c}$-CVPP is given by*

$$\exp\left(-\sum_{i=1}^{n} w_a(x_{i-1}, x_i)d + o(d)\right).$$

As we have an exact formula for the optimal number of steps, and the lower bound from [DLW19] uses a 'single-step' analysis we know exactly in which regime Corollary 85 improves on theirs. Namely for those $a > 1$ and $c \geq 1$ such that for n defined by equation (5.1) we have $n > 1$. For exact CVPP we obtain improvements for $a < 1.22033$, i.e., when using less than $2^{0.1436d+o(d)}$ memory, as can be seen in Figure 5.1. This improvement can also be visualized through Figure 5.5, which plots the optimal number of steps against the size of the preprocessed list. Whenever the optimal strategy involves taking more than one step, we improve upon [DLW19]. For the crossover points where the number of optimal steps changes we have a more succinct formula for the shortest path and the success probability.

Lemma 87 (Success probability for integral n). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. If n defined similar to equation (5.1), but without rounding up, is integral, then the optimal path from b to c has probability*

$$\left(\left(\frac{a}{2-a}\right)^n \cdot \left(1 - \frac{2n(a-1)}{2-a}\right)\right)^{d/2+o(d)}.$$

Proof. For such n we obtain the expression $x_i = b - (i+1) \cdot i \cdot \frac{a^2-a}{2-a}$. The result follows from simplifying the remaining expression. □

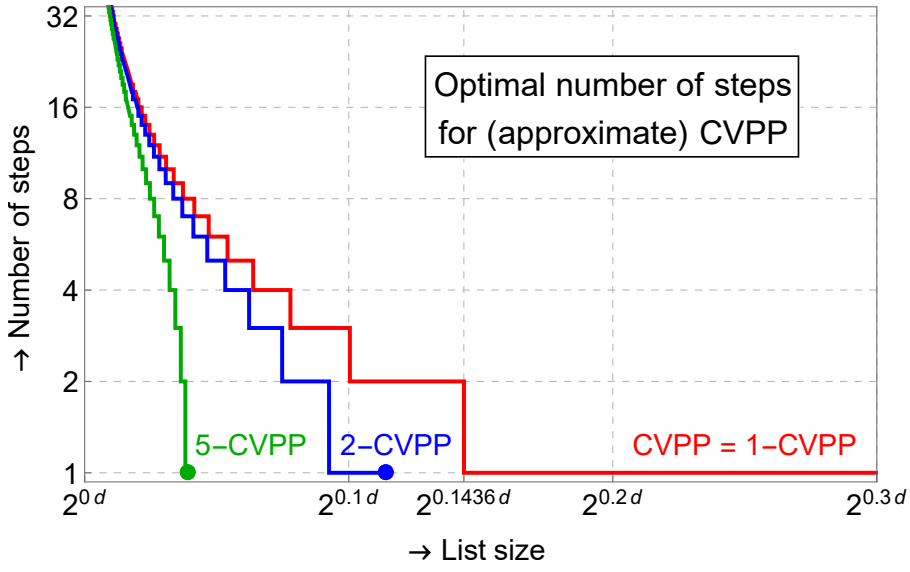


Figure 5.5: Optimal number of steps n against the list size $|L| = \alpha^{d+o(d)} = a^{d/2+o(d)}$. We improve upon [DLW19] whenever $n > 1$. For large list sizes the optimal number of steps of cost $\exp(-Cd + o(d))$ drops to 0, as then the success probability of the iterative slicer equals $2^{-o(d)}$.

Part III

Basis Reduction

CHAPTER 6

Background on Basis Reduction

This chapter gives an introduction to different types of basis reduction algorithms, and the state-of-the-art of modelling their behaviour.

6.1 Introduction

Every $\mathbb{R}$-linear subspace $V \subset \mathbb{R}^n$ has an orthogonal basis, and many efficient algorithms in linear algebra rely on such a basis. Similarly, an orthogonal basis $\mathbf{B}$ of a lattice would make many problems efficiently solvable, e.g., solving CVP for $\mathbf{t} = \mathbf{Bx}$ would be as simple as rounding the coefficients $\mathbf{c} = \mathbf{B}\lfloor \mathbf{x} \rceil$. Any rank n lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ has (for $n > 1$) an infinite number of other bases $\mathbf{B} \cdot \mathbf{U}$ for $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$, but for most lattices none of those are orthogonal. Still most lattices have close to orthogonal bases, which still allows us to do some algorithmic tasks efficiently. We call such a basis *well-reduced* or simply *good*. Basis reduction is the act of turning a (bad) basis into a good basis. A common way to measure the orthogonality of a basis is by

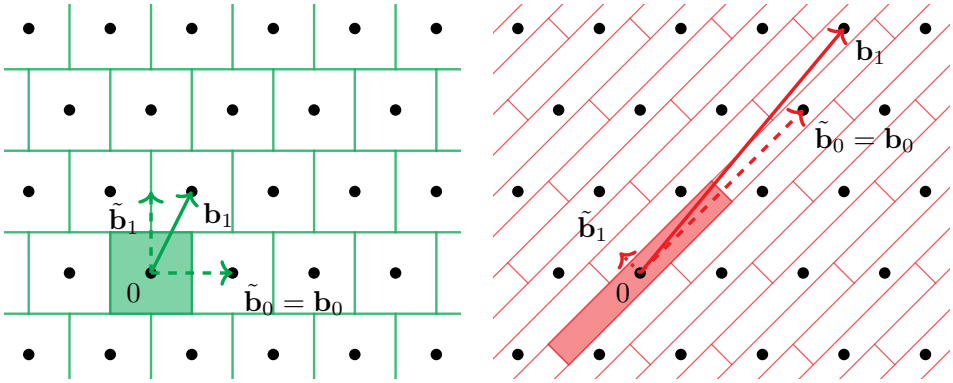


Figure 6.1: Babai's fundamental domain for a good basis (left) versus a bad (right) basis.

its orthogonality defect $\delta(\mathbf{B}) := \frac{\prod_i \|\mathbf{b}_i\|}{\prod_i \|\tilde{\mathbf{b}}_i\|}$, where $\tilde{\mathbf{b}}_i$ is the i-th Gram-Schmidt vector. We have $\delta(\mathbf{B}) \geq 1$ with equality if and only if the basis is orthogonal. Since we can rewrite $\delta(\mathbf{B}) = \frac{\prod_i \|\mathbf{b}_i\|}{\mathrm{vol}(\mathcal{L}(\mathbf{B}))}$, we see that to minimize the orthogonality defect a good basis must consist of short vectors. For a lattice basis orthogonality and shortness are thus directly connected.

To break most lattice based schemes we do not have to compute a basis with the lowest orthogonality defect, or consisting of the shortest vectors; a somewhat orthogonal basis is often sufficient. Once a certain orthogonality is reached we can use the good basis to decrypt a message, recover a secret key, or forge a signature. In part II of this thesis we have shown how to solve exact versions of the shortest and closest vector problem, in moderate dimensions. Running these algorithms on the high dimensional lattices common in cryptography is far beyond feasible, and would also find much shorter vectors than needed to break the scheme. In this chapter we show how to bootstrap these exact algorithms in moderate dimensions, to reduce a large dimensional basis. In particular this allows to compute approximate shortest vectors from lower dimensional exact SVP oracles.

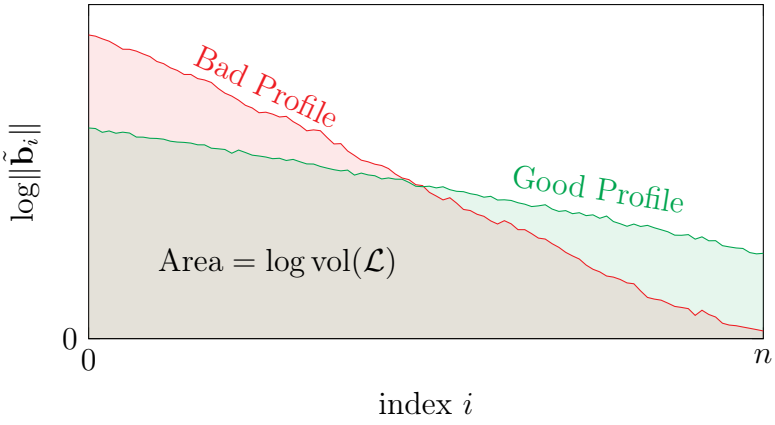


Figure 6.2: A good versus a bad basis log-profile.

6.1.1 Basis profile

It turns out that algorithmically, the profile of a basis, i.e., its GSO norms $(\|\tilde{\mathbf{b}}_i\|)_i$ are much more interesting than the norms of the basis vectors themself. They are directly related, i.e., by Lemma 38 a size-reduced basis consists of short vectors if and only if the GSO norms are short. Still, most cryptanalytic algorithms actually rely on the profile, and not the basis norms.

One such example is Babai's nearest plane algorithm. Recall from Section 2.4.3 that given a basis $\mathbf{B}$ of $\mathcal{L}$, Babai's nearest plane algorithm decodes any target $\mathbf{t} \in \mathrm{span}(\mathcal{L})$ to a close vector $\mathbf{c} \in \mathcal{L}$ such that $\mathbf{t} - \mathbf{c} \in \mathcal{P}(\tilde{\mathbf{B}})$. As a result Babai's algorithm perfectly decodes targets up to distance $\frac{1}{2}\min_i\|\tilde{\mathbf{b}}_i\|$ from the lattice, and has worst-case and average-case squared decoding distance proportional to $\sum_i\|\tilde{\mathbf{b}}_i\|^2$.

Due to the invariant $\prod_i\|\tilde{\mathbf{b}}_i\| = \mathrm{vol}(\mathcal{L})$, both $\min_i\|\tilde{\mathbf{b}}_i\|$ and $\sum_i\|\tilde{\mathbf{b}}_i\|^2$ are optimal (maximized and minimized respectively) when $\|\tilde{\mathbf{b}}_0\| = \ldots = \|\tilde{\mathbf{b}}_{n-1}\| = \mathrm{vol}(\mathcal{L})^{1/n}$, i.e. when the GSO profile is perfectly balanced. Such a basis with a perfectly balanced profile might not exist, e.g. by the Gaussian Heuristic we already have $\|\tilde{\mathbf{b}}_0\| \geq \lambda_1(\mathcal{L}) \approx \sqrt{n/(2\pi e)}\,\mathrm{vol}(\mathcal{L})^{1/n} \gg \mathrm{vol}(\mathcal{L})^{1/n}$. A typical reduced basis profile starts high and decreases quickly (see Figure 6.2). We could (informally) speak of the *slope* of a profile.

For a not so orthogonal bad basis the slope is steep, and the first GSO norm $\|\tilde{\mathbf{b}}_0\|$ is very large, leading to a poor decoding error propor-

tional to $\sum_i \|\tilde{\mathbf{b}}_i\|^2 \geq \|\tilde{\mathbf{b}}_0\|^2$. Similarly, the last GSO norm $\|\tilde{\mathbf{b}}_{n-1}\|$ is very small, leading to a poor bounded distance decoding performance. In contrast, a close to orthogonal good basis has a gradual declining and much more balanced slope, and therefore achieves much better decoding performances.

In the following Sections we consider three ways of obtaining a good basis. The exponential time HKZ algorithm, which achieves a very good slope by minimizing $\|\tilde{\mathbf{b}}_0\|$, then $\|\tilde{\mathbf{b}}_1\|$, and so on. The polynomial-time LLL algorithm, which achieves a mildly good slope, by improving the slope locally. And lastly the BKZ algorithm that combines these ideas and offers a tunable time-quality trade-off between the two.

6.2 HKZ reduction

One approach to balance the typically decreasing basis profile ($\|\tilde{\mathbf{b}}_0\|$, $\ldots, \|\tilde{\mathbf{b}}_{n-1}\|$), is by minimizing the first norm $\|\tilde{\mathbf{b}}_0\|$, then the second $\|\tilde{\mathbf{b}}_1\|$, and so on. The first GSO vector $\tilde{\mathbf{b}}_0$ is just the first basis vector $\mathbf{b}_0$, so by definition it is minimized when $\|\mathbf{b}_0\| = \lambda_1(\mathcal{L})$. For a fixed $\mathbf{b}_0$, the second GSO vector $\tilde{\mathbf{b}}_1$ is exactly the first basis vector in $\mathcal{L}_{[1:n)}$, and is thus minimized when $\|\tilde{\mathbf{b}}_1\| = \lambda_1(\mathcal{L}_{[1:n)})$. We can continue this greedy process until the end. The resulting basis is said to be Hermite-Korkine-Zolotarev reduced.

Definition 88 (Hermite-Korkine-Zolotarev reduced). *A basis* $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ *is called Hermite-Korkine-Zolotarev (HKZ) reduced if it is size-reduced and*

$$\|\tilde{\mathbf{b}}_\kappa\| = \lambda_1(\mathcal{L}_{[\kappa:n)}) \quad \textit{for all } \kappa = 0, \ldots, n-1.$$

That such a basis exists follows from the fact that any shortest vector of $\mathcal{L}_{[\kappa:n)}$ is primitive in $\mathcal{L}_{[\kappa:n)}$.

An alternative recursive definition would be as follows: a size-reduced basis $[\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ is HKZ reduced if $\|\mathbf{b}_0\| = \lambda_1(\mathcal{L})$, and $\mathbf{B}_{[1:n)}$ is HKZ reduced (or equals $\{\mathbf{0}\}$). From this Algorithm 7 to obtain an HKZ reduced basis is straightforward. Compute a shortest vector, project away from it, and repeat on the projected lattice.

HKZ reduction is very strong, and could be interpreted as (close to) optimal for many purposes. In particular, an HKZ reduction oracle

Algorithm 7: The HKZ algorithm.

Data: A rank n lattice $\mathcal{L}$ (represented by any basis).

for $\kappa = 0, \ldots, n-1$ **do**
  $\pi_\kappa := \pi_{(\mathbf{b}_0,\ldots,\mathbf{b}_{\kappa-1})^\perp}$;
  $\mathbf{w} \leftarrow$ a shortest vector in $\pi_\kappa(\mathcal{L})$;
  Lift $\mathbf{w}$ to a lattice vector $\mathbf{b}_\kappa \in \mathcal{L}$ such that $\pi_\kappa(\mathbf{b}_\kappa) = \mathbf{w}$;
  Size-reduce $\mathbf{b}_\kappa$ w.r.t. $[\mathbf{b}_0, \ldots, \mathbf{b}_{\kappa-1}]$;
end
return $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$;

would break almost all lattice-based cryptography. However, this good reduction quality comes at a cost. Since one has to solve an exact SVP instance in dimension n, computing an HKZ reduced basis quickly becomes unfeasible.

6.3 The LLL algorithm

The LLL algorithm by Lenstra, Lenstra and Lovász [LLL82] is a basis reduction algorithm that runs in polynomial time and which reaches approximation factors exponential in the dimension. It was the first polynomial time basis reduction algorithm, and it has found countless applications. Furthermore, it is a fundamental building block in more complex basis reduction algorithms.

The general idea of the LLL algorithm is to improve the basis slope, by making sure that consecutive GSO norms $\|\tilde{\mathbf{b}}_\kappa\|, \|\tilde{\mathbf{b}}_{\kappa+1}\|$ do not decrease too quickly. This is achieved by locally reducing all the rank two projected sublattices $\mathcal{L}_{[\kappa:\kappa+2)} = \mathcal{L}([\pi_\kappa(\mathbf{b}_\kappa), \pi_\kappa(\mathbf{b}_{\kappa+1})])$ for $\kappa = 0, \ldots, n-2$.

6.3.1 Lagrange reduction

So let us first focus on the rank two case $\mathcal{L} = \mathcal{L}([\mathbf{b}_0, \mathbf{b}_1])$. We want two basis vectors to be as short and as orthogonal as possible. Following the HKZ reduction algorithm it is natural to let the first basis vector be a shortest vector of the lattice. In this way we obtain a good GSO profile, and by size-reduction a close to orthogonal basis. The resulting

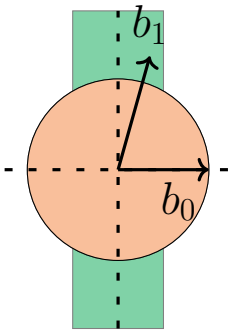


Figure 6.3: Illustration of the Wristwatch Lemma 89. There always exists a basis $[\mathbf{b}_0, \mathbf{b}_1]$ with $\mathbf{b}_0$ a shortest vector, and $\mathbf{b}_1$ in the green strap of width $\frac{1}{2}\|\mathbf{b}_0\|$ (in the direction orthogonal to $\mathbf{b}_0$).

basis is called Lagrange reduced, and visually looks like a 'Wristwatch' basis (see Figure 6.3).

Lemma 89 (Wristwatch Lemma). *Let $\mathcal{L} \subset \mathbb{R}^d$ be a lattice of rank 2, then there exists a $\mathbf{B} = [\mathbf{b}_0, \mathbf{b}_1]$ of $\mathcal{L}$ that satisfies*

1. $\|\mathbf{b}_0\| \leq \|\mathbf{b}_1\|$, *and*
2. $|\langle \mathbf{b}_0, \mathbf{b}_1 \rangle| \leq \frac{1}{2}\|\mathbf{b}_0\|^2$ *(size-reduction).*

In particular this implies that $\|\mathbf{b}_0\| = \lambda_1(\mathcal{L})$ and $\|\tilde{\mathbf{b}}_0\| \leq \sqrt{4/3} \cdot \|\tilde{\mathbf{b}}_1\|$. We call such a basis Lagrange reduced.

There is an efficient algorithm to obtain such a basis.

Lemma 90. *On input a basis of a rank 2 lattice $\mathcal{L}$, Algorithm 8 terminates and returns a Lagrange reduced basis $[\mathbf{b}_0, \mathbf{b}_1]$ of $\mathcal{L}$, i.e. a basis satisfying the condition in Lemma 89.*

Proof. Swapping basis vectors, and subtracting integer multiples of other basis vectors, are all unimodular operations. Therefore the returned basis is still a basis of the input lattice. This also implies that $\mathbf{b}_0, \mathbf{b}_1$ are nonzero throughout the algorithm.

The algorithm terminates in a finite number of steps given that the norm of $\|\mathbf{b}_0\|$ strictly decreases in each iteration, and because by a packing argument there are only a finite number of lattice points in

Algorithm 8: Lagrange reduction

Input : A rank 2 basis $[\mathbf{b}_0, \mathbf{b}_1]$.
Input : A Lagrange reduced basis of the input lattice.

repeat
  swap $\mathbf{b}_0 \leftrightarrow \mathbf{b}_1$
  $k \leftarrow \left\lfloor \frac{\langle \mathbf{b}_0, \mathbf{b}_1 \rangle}{\langle \mathbf{b}_0, \mathbf{b}_0 \rangle} \right\rceil$
  $\mathbf{b}_1 \leftarrow \mathbf{b}_1 - k \cdot \mathbf{b}_0$ `// size-reduction`
until $\|\mathbf{b}_0\| \leq \|\mathbf{b}_1\|$
return $[\mathbf{b}_0, \mathbf{b}_1]$

$\mathcal{L}$ of norm at most some bound. A closer inspection shows that the number of iterations is of order at most $O(\log(\|\mathbf{b}_0\|))$.

Now let us consider the two conditions of Lemma 89. The second condition $\|\mathbf{b}_0\| \leq \|\mathbf{b}_1\|$ is true by the termination condition of the repeat loop. The size-reduction operation just before that makes sure that the first condition $|\langle \mathbf{b}_0, \mathbf{b}_1 \rangle| \leq \frac{1}{2}\|\mathbf{b}_0\|^2$ is also true. □

Proof Lemma 89. The existance of such a basis follows directly from Lemma 90. Let $[\mathbf{b}_0, \mathbf{b}_1]$ be a Lagrange reduced basis. Consider any nonzero lattice vector $\mathbf{v} = x\mathbf{b}_0 + y\mathbf{b}_1 \in \mathcal{L}$ with $(0,0) \neq (x,y) \in \mathbb{Z}^2$. We want to show that $\|\mathbf{v}\| \geq \|\mathbf{b}_0\|$. When $y = 0$ this is trivial, and when $x = 0$ it follows from $\|\mathbf{b}_1\| \geq \|\mathbf{b}_0\|$. For the case $x \neq 0$ and $y \neq 0$ we have

$$\begin{aligned}\|\mathbf{v}\|^2 &= x^2 \cdot \|\mathbf{b}_0\|^2 + y^2 \cdot \|\mathbf{b}_1\|^2 + 2xy\langle \mathbf{b}_0, \mathbf{b}_1 \rangle \\ &\geq (x^2 + y^2 - |xy|)\|\mathbf{b}_0\|^2 \geq \min\{x^2, y^2\} \cdot \|\mathbf{b}_0\|^2 \geq \|\mathbf{b}_0\|^2,\end{aligned}$$

and thus $\|\mathbf{b}_0\| = \lambda_1(\mathcal{L})$. For the GSO norms $\|\tilde{\mathbf{b}}_0\|, \|\tilde{\mathbf{b}}_1\|$ we have

$$\|\mathbf{b}_0\|^2 \leq \|\mathbf{b}_1\|^2 = \|\tilde{\mathbf{b}}_1\|^2 + |\langle \mathbf{b}_0, \mathbf{b}_1 \rangle|^2 / \|\mathbf{b}_0\|^2 \leq \|\tilde{\mathbf{b}}_1\|^2 + \frac{1}{4}\|\mathbf{b}_0\|^2,$$

and thus $\|\tilde{\mathbf{b}}_0\| = \|\mathbf{b}_0\| \leq \sqrt{4/3} \cdot \|\tilde{\mathbf{b}}_1\|$. □

6.3.2 LLL

If all the projected sublattice bases $\mathbf{B}_{[i:i+2)}$ are Lagrange reduced, then we obtain $\|\tilde{\mathbf{b}}_{i+1}\| \geq \sqrt{3/4}\|\tilde{\mathbf{b}}_i\|$ for all $i = 0, \ldots, n-2$. In other

words, the profile cannot decrease too quickly. Simply stated the LLL algorithm proceeds as follows: if any local basis $\mathbf{B}_{[i:i+2)}$ is not Lagrange reduced, then reduce it. Clearly if the algorithm terminates all local bases are Lagrange reduced. To show that it terminates in a polynomial number of iterations we need to slightly relax the local Lagrange condition $\|\pi_i(\mathbf{b}_i)\| \leq \|\pi_i(\mathbf{b}_{i+1})\|$.

Definition 91 (LLL reduced). *A lattice basis* $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}]$ *is called* δ*-LLL reduced, for* $\delta \in (\frac{1}{4}, 1]$*, if*

1. *The Lovász Condition* $\delta\|\pi_i(\mathbf{b}_i)\|^2 \leq \|\pi_i(\mathbf{b}_{i+1})\|^2$ *holds for all* $0 \leq i < n-1$*, and*

2. *it is size-reduced.*

We call $\delta \in (\frac{1}{4}, 1]$ the reduction parameter. Note that $\delta = 1$ represents exact Lagrange reduction, but for this value we cannot show that the algorithm terminates in polynomial time. A δ-LLL reduced bases does not decrease too steeply, i.e. combining the two conditions we obtain

$$\|\tilde{\mathbf{b}}_{\kappa+1}\| \geq \sqrt{\delta - \frac{1}{4}}\|\tilde{\mathbf{b}}_{\kappa}\|,$$

for all $\kappa = 0, \ldots, n-2$. This directly has implications for the length of the basis vectors.

Corollary 92 (LLL results). *Let* $\delta \in (\frac{1}{4}, 1]$*. For a* δ*-LLL reduced basis* $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ *of the lattice* $\mathcal{L}$*, we have*

$$\|\mathbf{b}_0\| \leq (\delta - \tfrac{1}{4})^{-(n-1)/2} \cdot \lambda_1(\mathcal{L})$$
$$\|\mathbf{b}_0\| \leq (\delta - \tfrac{1}{4})^{-(n-1)/4} \cdot \mathrm{vol}(\mathcal{L})^{1/n}$$

Proof. Let $\gamma := \sqrt{1/(\delta - \frac{1}{4})}$. By combining the LLL conditions we have

$$\|\mathbf{b}_0\| = \|\tilde{\mathbf{b}}_0\| \leq \gamma \cdot \|\tilde{\mathbf{b}}_1\| \leq \ldots \gamma^{n-1} \cdot \|\tilde{\mathbf{b}}_{n-1}\|.$$

In particular we have $\|\mathbf{b}_0\| \leq \gamma^{n-1} \cdot \min_i \|\tilde{\mathbf{b}}_i\|$, and the first statement follows from the fact that $\min_i \|\tilde{\mathbf{b}}_i\| \leq \lambda_1(\mathcal{L})$. For the second statement

we have

$$\det(\mathcal{L})^{1/n} = \left(\prod_{i=0}^{n-1} \|\tilde{\mathbf{b}}_i\|\right)^{1/n} \geq \left(\prod_{i=0}^{n-1} \|\tilde{\mathbf{b}}_0\| \cdot \gamma^i\right)^{1/n} = \|\mathbf{b}_0\| \cdot \gamma^{(n-1)/2}.$$

□

For $\delta = 1$ we have $\|\mathbf{b}_0\| \leq (4/3)^{(n-1)/2} \cdot \mathrm{vol}(\mathcal{L})^{1/n}$. Recall that Hermite's bound says precisely that such a vector exists, i.e., $\lambda_1(\mathcal{L}) \leq (4/3)^{(n-1)/2} \cdot \mathrm{vol}(\mathcal{L})^{1/n}$. As such, the LLL algorithm can be seen as an algorithmic instantiation of Hermite's bound: it does not just show existence, it actually computes a vector of (almost) that length.

6.3.3 LLL algorithm

We will now discuss why the LLL algorithm terminates in a polynomial number of iterations. To show that the LLL algorithm runs in polynomial-time one would additionally have to show that the operations, like computing the GSO (using rationals), run in polynomial-time. We will not cover this, but this is precisely where the global size-reduction of the basis comes into play.

Algorithm 9: LLL reduction

Input : A basis $\mathbf{B} = [\mathbf{b}_0, , \ldots, \mathbf{b}_{n-1}]$ of a lattice $\mathcal{L}$ and a reduction parameter $\delta \in (\frac{1}{4}, 1]$.

Output: A δ-LLL reduced basis of the input lattice.

```
Size-reduce B
while ∃index i that does not satisfy Lovász' condition do
    Let U ∈ GL_2(ℤ) be s.t. B_[i:i+2) · U is Lagrange reduced
     // Using Algorithm 8
    [b_i, b_{i+1}] ← [b_i, b_{i+1}] · U                    // Update B
    Size-reduce B
end
return B
```

To argue that the number of iterations is polynomially bounded in the input size we use a *potential* argument.

Definition 93 (Potential). *For a basis* $\mathbf{B}$*, we define the potential* $\mathcal{D}_{\mathbf{B}}$ *as*

$$\mathcal{D}_{\mathbf{B}} = \prod_{i=1}^{n} \mathrm{vol}(\mathcal{L}_{0:i}) = \prod_{i=0}^{n-1} \|\tilde{\mathbf{b}}_i\|^{n-i}.$$

Lemma 94. *For any integer basis* $\mathbf{B} \in \mathbb{Z}^{m \times n}$ *and reduction parameter* $\delta \in (\frac{1}{4}, 1)$*, Algorithm 9 is correct and terminates after at most*

$$O(n^2 \log(\max_i \|\mathbf{b}_i\|))$$

iterations.

Proof. The correctness is clear from the loop condition.

To show that the number of iterations is polynomially bounded we first argue that the potential decreases significantly during each loop. First note that size-reducing a basis does not change the GSO norms, and thus the size-reduction operations do not change the potential. Now let us consider a loop, where $0 \leq i \leq n-2$ is the index at which Lovász' condition is not satisfied. At the start of the loop we thus have

$$\delta \|\pi_i(\mathbf{b}_i)\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2.$$

Let $[\mathbf{c}_i, \mathbf{c}_{i+1}] = [\mathbf{b}_i, \mathbf{b}_{i+1}] \cdot \mathbf{U}$ be the new basis vectors, $\tilde{\mathbf{c}}_i, \tilde{\mathbf{c}}_{i+1}$ the new GSO vectors, and let $\mathcal{D}_{\mathbf{B}}$ and $\mathcal{D}_{\mathbf{B}'}$ be the old and new potential respectively. The lattice $\mathcal{L}_{[i:i+2)}$ stays the same (and thus its span), so all GSO norms $\|\tilde{\mathbf{b}}_j\|$ for $j \notin \{i, i+1\}$ remain unchanged in this loop. Furthermore we have the invariant

$$\|\tilde{\mathbf{c}}_i\| \cdot \|\tilde{\mathbf{c}}_{i+1}\| = \mathrm{vol}(\mathcal{L}_{[i:i+2)}) = \|\tilde{\mathbf{b}}_i\| \cdot \|\tilde{\mathbf{b}}_{i+1}\|.$$

By Lagrange reduction, the vector $\tilde{\mathbf{c}}_i$ is a shortest vector in $\mathcal{L}_{[i:i+2)}$, and in particular $\|\tilde{\mathbf{c}}_i\| \leq \|\pi_i(\mathbf{b}_{i+1})\|$. To summarize we have

$$\|\tilde{\mathbf{c}}_i\|^2 \leq \|\pi_i(\mathbf{b}_{i+1})\|^2 < \delta \cdot \|\tilde{\mathbf{b}}_i\|^2,$$

and together with the invariant this gives

$$\|\tilde{\mathbf{c}}_i\|^{n-i} \cdot \|\tilde{\mathbf{c}}_{i+1}\|^{n-(i+1)} < \sqrt{\delta} \cdot \|\tilde{\mathbf{b}}_i\|^{n-i} \cdot \|\tilde{\mathbf{b}}_{i+1}\|^{n-(i+1)},$$

and thus $\mathcal{D}_{\mathbf{B}'} < \sqrt{\delta} \cdot \mathcal{D}_{\mathbf{B}}$. So the potential decreases by a constant factor $\frac{1}{2} < \sqrt{\delta} < 1$ each iteration.

To conclude we now have to upper and lower bound the potential. For the lower bound note that for any integer basis $\mathbf{B}$ we have $\mathcal{D}_{\mathbf{B}} \geq 1$. For the upper bound we have

$$\mathcal{D}_{\mathbf{B}} \leq \prod_{i=0}^{n-1} \|\mathbf{b}_i\|^{n-i} \leq B^{\frac{1}{2}n(n+1)},$$

where $B := \max_i \|\mathbf{b}_i\|$. By the constant factor decrease each iteration, and a lower and upper bound of 1 and $B^{\frac{1}{2}n(n+1)}$, there can be at most $O(n^2 \log B)$ iterations. □

We obtained the lower bound $\mathcal{D}_{\mathbf{B}} \geq 1$ by restricting to integer bases. This is not necessary, alternatively by Hermite's inequality or Minkowski's Theorem we could lower bound each volume $\mathrm{vol}(\mathcal{L}_{0:i})$ in terms of $\lambda_1(\mathcal{L}_{0:i}) \geq \lambda_1(\mathcal{L})$, which only depends on the lattice.

The LLL algorithm does not even need a basis, it can also be modified to work directly on the Gram matrix. The upper triangular Cholesky matrix $\mathbf{C}$ from the decomposition $\mathbf{G} = \mathbf{C}^\top\mathbf{C}$ of a Gram matrix plays the role of (basis and) GSO. The Gram matrix is modified by the unimodular transformations as $\mathbf{G}' = \mathbf{U}^\top\mathbf{G}\mathbf{U}$.

A small modification to the LLL algorithm by Pohst [Poh87] allows to remove the requirement that the input vectors are linearly independent. After this modification the input can consists of any number of vectors, and the output will consists of an LLL reduced basis of the lattice generated by the input vectors. In the next section we will use this property to 'insert' short vectors in a basis. Given a basis $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ and a short vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$, we can run the modified LLL reduction algorithm on $[\mathbf{v}, \mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$. Because the first GSO norm can never increase during the LLL reduction algorithm we end up with an LLL reduced basis $\mathbf{B}' = [\mathbf{b}'_0, \ldots, \mathbf{b}'_{n-1}]$ with $\|\mathbf{b}'_0\| \leq \min\{\|\mathbf{b}_0\|, \|\mathbf{v}\|\}$.

A nice property of the LLL algorithm is that it is self-dual: to be more precise, if a basis $\mathbf{B}$ is δ-LLL reduced, then the reversed dual basis $\mathbf{D}$ of $\mathbf{B}$ is after size-reducing also δ-LLL reduced. This means that e.g. Corollary 92 can also be implied to the reverse dual basis $[\mathbf{d}_{n-1}, \ldots, \mathbf{d}_0]$, and thus give a lower bound on $\|\tilde{\mathbf{b}}_{n-1}\| = 1/\|\mathbf{d}_{n-1}\|$.

6.4 BKZ reduction

We have seen the two extreme cases of basis reduction, the HKZ algorithm delivers close to optimal reduction at the cost of solving SVP in the full lattice, and the LLL algorithm that gives exponential approximation factors but runs in polynomial time. We will now discuss a generalisation, that delivers a trade-off between these two extremes.

So how to generalize the LLL and HKZ algorithms? The common theme is that they both rely on an SVP oracle for projected sublattices $\mathcal{L}_{[\kappa:\kappa+\beta)}$, which we call a *block*.

The difference being that HKZ works on blocks $\mathcal{L}_{[\kappa:n)}$ of rank at most n, while the LLL works on blocks $\mathcal{L}_{[\kappa:\kappa+2)}$ of rank 2 (Lagrange reduction). From this perspective the natural generalisation is to work on blocks $\mathcal{L}_{[\kappa:\kappa+\beta)}$ of rank at most β. This is precisely what Block-Korkine-Zolotarev [Sch87] reduction does, and we call the parameter β the *blocksize*.

Definition 95 (BKZ). *A basis $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ is called BKZ-β reduced if it is size-reduced and*

$$\|\tilde{\mathbf{b}}_\kappa\| = \lambda_1(\mathcal{L}_{[\kappa:\min(\kappa+\beta,n))}) \text{ for all } \kappa = 0, \ldots, n-1.$$

For $\beta = 2$ we recover the definition of 1-LLL, and for $\beta = n$ we recover the definition of HKZ. The condition on $\|\tilde{\mathbf{b}}_\kappa\|$ is often relaxed by a factor $1+\epsilon$ for some $\epsilon > 0$ close to 0, both to allow for numerical imprecisions and potential arguments like LLL. This is similar to the relaxation of the Lovász Condition in LLL. In what follows we will ignore this factor.

Similar to LLL a BKZ reduced basis contains a short basis vector.

Lemma 96 (BKZ approximation [Sch94]). *For a BKZ-δ reduced basis $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}]$ of the lattice $\mathcal{L}$, with $2 \leq \beta \leq n$, we have*

$$\|\mathbf{b}_0\| \leq \gamma_\beta^{\frac{n-1}{\beta-1}} \cdot \mathrm{vol}(\mathcal{L})^{1/n},$$

where γ_β is Hermite's constant of rank β.

In addition a Hermite factor bound $\|\mathbf{b}_0\| \leq \sqrt{\gamma_\beta}^{\frac{n-1}{\beta-1}+1} \cdot \mathrm{vol}(\mathcal{L})^{1/n}$ is claimed without proof in [GN08b]. Because $\gamma_\beta \leq O(\beta)$ the bound decreases as the blocksize β increases; a larger blocksize gives a better reduced basis.

6.4.1 BKZ algorithm

The BKZ algorithm (see Algorithm 10) computes a BKZ reduced basis from any other basis. The algorithm greedily attempts to satisfy the BKZ condition at each position by computing a shortest vector in each block $\mathcal{L}_{[\kappa:\min(\kappa+\beta,n))}$, and replacing the basis vector $\mathbf{b}_\kappa$ accordingly. This makes the basis BKZ-β reduced at position κ, but might invalidate the condition at other positions. Applying this once to all positions $\kappa = 0, \ldots, n-2$ is called a *tour*. The BKZ algorithm repeats such tours until the basis remains unchanged and is thus BKZ reduced.

Algorithm 10: The BKZ algorithm.

Input : A lattice basis $\mathbf{B}$, blocksize β.
Output: A BKZ-β reduced basis of the input lattice.

LLL reduce $\mathbf{B}$
while $\mathbf{B}$ *is not BKZ-*β *reduced* **do**
  for $\kappa = 0, \ldots, n-2$ **do** `// A single BKZ-β tour`
    $b \leftarrow \min\{\beta, n-\kappa\}$
    $\mathbf{w} \leftarrow$ a shortest vector in $\mathcal{L}_{[\kappa:\kappa+b)}$
    Lift $\mathbf{w}$ to a full vector $\mathbf{v} \in \mathcal{L}_{[0:\kappa+b)}$ s.t. $\pi_\kappa(\mathbf{v}) = \mathbf{w}$
    Insert $\mathbf{v}$ in $\mathbf{B}$ at position κ using LLL
    LLL reduce $\mathbf{B}_{[0:\kappa+b)}$
  end
end
return B

The BKZ algorithm as stated is not known to terminate in a polynomial number of tours. Terminating variants exist that succeed after about $O(n^2 \log(n)/\beta^2)$ tours [HPS11b; LN20], but the resulting bases are not precisely BKZ-β reduced, and the resulting worst-case bounds on $\|\mathbf{b}_0\|$ are a polynomial factor worse than Lemma 96.

The troubles of proving a polynomial bound on the number of tours, with a potential argument like LLL, seems to be related to the fact that the BKZ algorithm is not self-dual like LLL. There do exists variants of BKZ, like SDBKZ [MW16], that are self-dual, and that have easier provable runtime and quality bounds.

In practice however, not much improvement to the basis profile is attained after say a few dozen tours. The cost of BKZ is thus mainly dominated by the exponential (in β) cost of finding a shortest vector

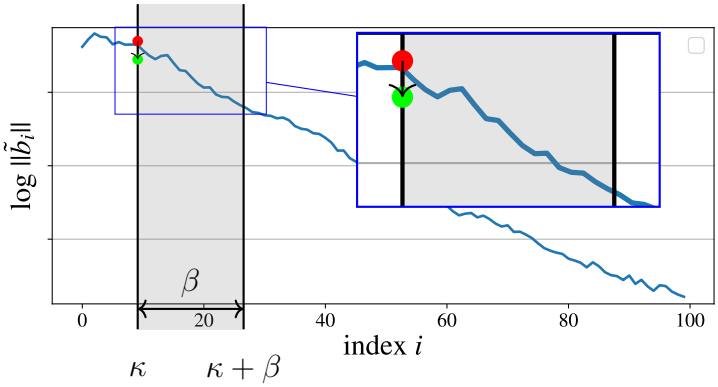


Figure 6.4: The BKZ algorithm: inserting a shortest vector in the projected sublattice $\mathcal{L}_{[\kappa:\kappa+\beta)}$.

in a β-dimensional lattice. Throughout this thesis we will therefore ignore the lattice rank and the number of iterations and cost the BKZ algorithm in terms of the blocksize β.

6.4.2 Progressive BKZ

The cost of a BKZ-β tour quickly grows when increasing β. We thus want to limit the number of tours we have to run at the maximal blocksize β. The better the basis is already reduced, the less tours are needed to reach a (close to) BKZ-β reduced basis. If we first reduce the basis with a lower blocksize $\beta' < \beta$, then afterwards only a few expensive tours are needed. Beyond the number of expensive tours needed this also has benefits for the underlying SVP subroutine. When using (extreme) enumeration a better basis lowers the cost significantly, and when using sieving a better basis improves both the number of dimensions for free and the number of sieving iterations needed during progressive sieving. The idea of *progressive BKZ* (Algorithm 11) is to apply this idea recursively: run only a few tours (say 1 or 2) for increasing $\beta' = 2, 3, \ldots, \beta$.

Algorithm 11: Progressive BKZ.

Input : A lattice basis $\mathbf{B}$, blocksize β, tours $T > 0$.

for $\beta' = 2, 3, \ldots, \beta$ **do**
  Run T BKZ-β' tours.
end
return B

6.4.3 Slide reduction

An alternative generalisation of LLL is the *slide reduction* algorithm. It uses SVP oracles both on primal blocks $\mathcal{L}_{[\kappa:\kappa+\beta)}$, and on dual blocks $\mathcal{L}^*_{[\kappa:\kappa:\beta)}$. This gives more control over the profile and makes slide reduction easier to (provably) analyse than BKZ. For $n \geq 2\beta$ this also leads to slightly better asymptotic bounds on the norm $\|\mathbf{b}_0\|$ of the first basis vector.

Unfortunately, in practice the performance of BKZ seems to be better than slide reduction, and as a result most (concrete) cryptanalysis has been based on the BKZ algorithm. Also, the BKZ algorithm reduces the full bases, which is important for solving certain variants of SVP that are common in cryptography. In contrast, slide reduction focusses mostly on decreasing the norm $\|\mathbf{b}_0\|$ of the first basis vector, e.g. solving approx-SVP. Indeed, recently it was hinted that slide reduction can be competitive with BKZ for approx-SVP [MW16; Wal21]. While it would be interesting to see how the slide reduction algorithm behaves on the lattice problems in this thesis, we keep our focus on the BKZ algorithm which is much better understood from a concrete perspective.

6.5 Behaviour of reduction algorithms

For cryptanalysis it is important to understand the practical behaviour of lattice reduction algorithms. In particular in cryptographic dimensions, in which it is infeasible to obtain experimental evidence. In this section we explain the heuristic models behind the behaviour of HKZ, LLL and BKZ. These models are often build on the Gaussian Heuristic and based on extensive experimental evidence in feasible dimensions.

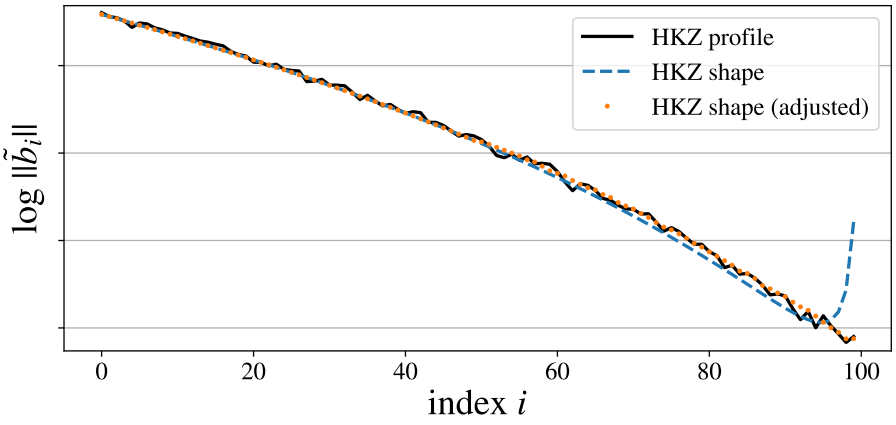


Figure 6.5: A typical HKZ reduced basis of a rank 100 lattice versus the HKZ shape of Definition 97, with and without adjustments for the tail part.

6.5.1 HKZ shape

In Figure 6.5 we see that the typical log-profile of a HKZ reduced random lattice basis is somewhat concave. This shape is correctly predicted by applying the Gaussian Heuristic to the HKZ definition.

Definition 97. *We define the HKZ shape* $(\ell_0, \ldots, \ell_{n-1})$ *of rank* n *by the following sequence*

$$\ell_0 := \mathrm{gh}(n), \qquad \ell_i = \mathrm{gh}(n-i) \cdot \left(\prod_{j<i} \ell_j \right)^{-1/(n-i)}.$$

The above profile shape is for a lattice with volume 1. In general for a HKZ reduced basis $\mathbf{B}$ of rank n the Gaussian Heuristic says that $\log \|\tilde{\mathbf{b}}_i\| \approx \log \ell_i + \frac{1}{n} \log(\det \mathcal{L})$. This estimate is accurate for $i \ll n$, say when $i \leq n - 50$, as can be seen in Figure 6.5.

The tail part of the estimate is inaccurate, because the Gaussian Heuristic gives false predictions in such low dimensions. One way to solve this is to experimentally observe a HKZ profile for a lattice of rank 50, and use that for the last part of the estimate. This adjusted estimate, as shown in Figure 6.5, closely predicts the actual HKZ profile.

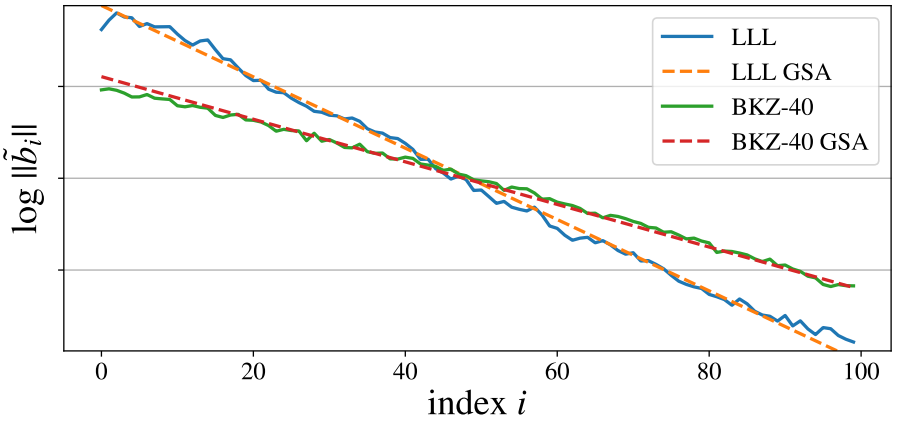


Figure 6.6: Illustration of the Geometric Series Assumption for 100 dimensional LLL and BKZ-40 reduced bases.

6.5.2 Geometric Series Assumption

In Figure 6.6 we see that the typical log-profile of an LLL or BKZ-β reduced basis for $\beta \ll n$ is close to a straight line, i.e. $\|\tilde{\mathbf{b}}_{i+1}\|/\|\tilde{\mathbf{b}}_i\| = \alpha$ for some constant $\alpha > 1$. We can predict the constant by the Gaussian Heuristic, which gives us the Geometric Series Assumption (GSA).

Heuristic 98 (Geometric Series Assumption (GSA)). *Let* $\mathbf{B}$ *be a basis of rank* n *that is BKZ-*β *reduced, then its profile satisfies*

$$\log(\|\tilde{\mathbf{b}}_i\|) = \frac{n-1-2i}{2} \cdot \log(\alpha_\beta) + \frac{\log(\det(\mathbf{B}))}{n},$$

where $\alpha_\beta = \mathrm{gh}(\beta)^{2/(\beta-1)}$.

Justification. Let us zoom in on a BKZ block $\mathcal{L}' = \mathcal{L}_{[\kappa:\kappa+\beta)}$. By the Gaussian Heuristic we have $\lambda = \lambda_1(\mathcal{L}') = \mathrm{gh}(\beta) \cdot \mathrm{vol}(\mathcal{L}')^{1/\beta}$. Under the assumption that the GSO norms decrease exponentially by some factor $\alpha_\beta > 1$ we have $\|\tilde{\mathbf{b}}_{\kappa+i}\| = \lambda/\alpha^i$, and thus $\mathrm{vol}(\mathcal{L}')^{1/\beta} = \lambda/\alpha_\beta^{(\beta-1)/2}$. To conclude the constant $\alpha_\beta > 1$ must satisfy

$$\lambda = \mathrm{gh}(\beta) \cdot \mathrm{vol}(\mathcal{L}') = \mathrm{gh}(\beta) \cdot \lambda/\alpha_\beta^{(\beta-1)/2},$$

from which it follows that $\alpha_\beta = \mathrm{gh}(\beta)^{2/(\beta-1)}$. △

The GSA is reasonably precise for say $\beta \geq 50$ and a not too large blocksize $\beta \ll n$ compared to the lattice rank. For small blocksizes β one can determine the value of α_β experimentally to still get accurate predictions.

However, one should be careful when using the GSA to estimate the start (head) and the end (tail) of the profile. Firstly by definition the last BKZ block $\mathcal{L}_{[n-\beta:n)}$ is HKZ reduced and thus the tail of the profile is slightly concave. Secondly, the Gaussian Heuristic is only an average-case statement, and during BKZ reduction vectors slightly shorter than the prediction can be found. By their shortness the BKZ algorithm is biased towards keeping those vectors and pushing them to the front of the basis. As a result the head of the profile can be a bit lower than predicted by the GSA.

6.5.3 Simulators

While the Geometric Series Assumption gives a good first order estimate of the basis profile after BKZ-reduction, it is known to be inaccurate in small dimensions or when the dimension is only a small multiple of the blocksize due to the head and tail behaviour. Additionally it does not account for the slower convergence when running progressive BKZ with only a few tours. To resolve these problems [CN11] introduced a BKZ simulator based on the Gaussian Heuristic.

This simulator keeps track of the profile $\ell = (\|\tilde{\mathbf{b}}_0\|, \ldots, \|\tilde{\mathbf{b}}_{n-1}\|)$, and runs BKZ tours where instead of computing a shortest vector it just updates

$$\ell_\kappa = \min\left\{\ell_\kappa, \mathrm{gh}(b) \cdot \left(\prod_{i=\kappa}^{\kappa+b-1} \ell_i\right)^{1/b}\right\},$$

for $b = \min\{\beta, n-\kappa\}$ (or experimental values of $\mathrm{gh}(b)$ for $b < 50$), and adjusts the remaining norms $\ell_\kappa, \ldots, \ell_{\kappa+b-1}$ accordingly. Such a simulator predicts correctly both the center (body) and tail part of the profile. This simulator was later refined by a probabilistic variant of the Gaussian Heuristic that can return slightly short vectors with a small probability [YD17; BSW18]. Due to this addition the head behaviour is also captured. In short, these simulators allow for accurate

and efficient predictions of the profile shape for random lattices, even for progressive BKZ with a limited number of tours.

CHAPTER 7

Overstretched NTRU

This chapter is based on the joint work 'NTRU Fatigue: How Stretched is Overstretched', with Léo Ducas, published at Asiacrypt 2021.

7.1 Introduction

One could view lattice reduction, or more specifically the BKZ algorithm, as a way to solve the approximate Shortest Vector Problem, or to create a good basis. However, in practice BKZ goes beyond that and can also recover (hidden) geometric structures of a lattice. For example if a rank d lattice contains an unusually short vector, much shorter than the Gaussian Heuristic prescribes, then the BKZ algorithm recovers this shortest vector with a much smaller blocksize $\beta \ll d$.

Heuristically, this behaviour can be fully explained, and after a line of works [GN08b; AFG13; ADPS16; DDGR20; PV21] we can predict accurately for which blocksize β an unusual short vector is recovered. For most lattice-based cryptosystems, key recovery or decryption can be reduced to finding such an unusually short vector, and

thus these predictions directly give an upper bound on the security of such schemes. In fact, for most schemes and common parameters this is (close to) the best known attack.

One such cryptosystems is the NTRU cryptosystem of Hoffstein, Pipher and Silverman [HPS98; Che+20]. The NTRU secret key consists of polynomials $\mathbf{f}, \mathbf{g} \in \mathbb{Z}[X]/(X^n - 1)$ with n prime, and with small, e.g., ternary, coefficients, and the public key is given by $\mathbf{h} := \mathbf{g}/\mathbf{f} \bmod q$ for some modulus q. The public key and the modulus q allow one to define an 'NTRU lattice' of dimension $d = 2n$, which contains an unusually short vector related to the secret key. Recovery of this vector immediately leads to full key-recovery, and thus the estimated security, for example that of the NIST 3rd round finalist NTRU [Che+20], is directly based on the predictions of how the BKZ algorithms recovers this unusually short vector in the NTRU lattice.

However, only recently, it was discovered that the security of NTRU is in fact more subtle than the problem of finding a single unusually short vector in a lattice; the NTRU lattice contains more structure. The first dent in this status quo came in 2016, from two concurrent works of Albrecht et al., and Cheon et al. [ABD16; CJL16], which exploit the specific algebraic structure of the NTRU lattice to improve upon pure lattice reduction attacks[1]. This approach was shown to be applicable when the modulus q is large enough (say, super-polynomial), a regime coined "overstretched".

Shortly thereafter Kirchner and Fouque [KF17] showed that this improved complexity does *not* require any algebraic structure, and is instead rooted in the purely geometrical fact that the NTRU lattice contains an unusually dense sublattice of large rank, *i.e.* a sublattice of small determinant.[2] Due to this dense sublattice lattice reduction attacks perform much better than initially expected. They also go further in their analysis, and conclude that moduli q as small as $n^{2.783+o(1)}$ already belong to the overstretched regime —for random ternary secrets. In particular, for q larger than this bound, the security of NTRU is significantly less than that of Learning With Errors [Reg04] and of

[1] Though the idea had been inconclusively considered already in 2002 by Gentry, Jonsson, Nguyen Stern and Szydlo as reported in [GS02, Sec. 6].

[2] Note that one may associate a short vector to a dense sublattice of dimension 1.

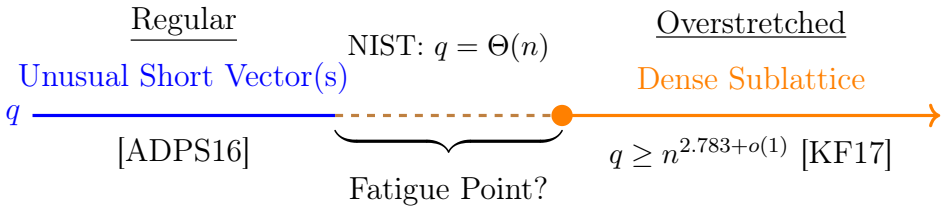


Figure 7.1: A sketch of the regular versus the overstretched regime, and the unknown fatigue point in between.

its Ring variant [SSTX09; LPR13] using similar parameters.[3]

However, it is not so clear from the analysis of Kirchner and Fouque whether this asymptotic quantity $n^{2.783+o(1)}$ is an estimate or merely an upper bound on the *fatigue point*, that is the value of q separating the standard regime from the overstretched regime. Their analysis is based on a lemma of Pataki and Tural [PT08], that constraints the shape of lattice basis in terms of the volume of their sublattices. While it allows to conclude that the dense sublattice must be discovered after reducing the lattice basis beyond these constraints, it does not really explain *how* lattice reduction ends up discovering the dense sublattice, nor does it exclude that the discovery could happen earlier.

So far, it has been generally considered that only advanced schemes —requiring very large q— such as NTRU-based Homomorphic Encryption [BLLN13] or cryptographic multi-linear maps [GGH13] could be affected by this overstretched regime. Yet, because the analysis of Kirchner and Fouque is only asymptotic, and because it may only provide an upper bound on the fatigue point, there is at the moment little documented evidence that the overstretched regime may not in fact extend further down, maybe down to the NTRU encryption scheme itself [HPS98; Che+20]! Admittedly, this seems like a far fetched concern: asymptotically this scheme chooses $q = O(n)$, with a hidden constant between 4 and 5 in practice. However, this scheme being now a finalist of the NIST standardisation process for post-quantum cryptography, it appears rather imperious to refine our understanding of the phenomenon, and to finally close this pending question.

[3] In fact, the presence of n rotations of the secret key already implies a minor security degradation compared to (Ring)-LWE already in the standard regime [MS01; DDGR20].

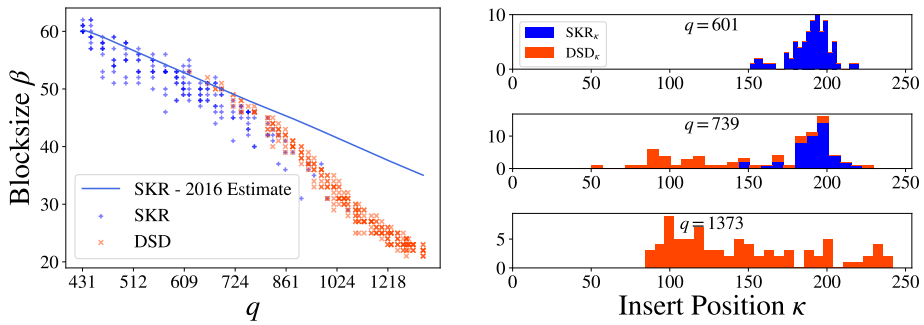


Figure 7.2: Progressive BKZ with 8 tours per blocksize on matrix NTRU instances with parameters $n = 127, \sigma^2 = \frac{2}{3}$ for several moduli q. **Left:** the first blocksize β at which Progressive BKZ detects the Secret Key Recovery (SKR$_\kappa$) or Dense Sublattice Discovery (DSD$_\kappa$) event. We did 10 runs per modulus q. For the 2016-estimates, we use the geometric series assumption (GSA) for the shape of the basis and a probabilistic model for the discovery of the secret vector (see Section 7.2.3). **Right:** the positions κ at which a secret key or dense sublattice vector are detected over 80 runs per modulus.

We found further motivation to go down this rabbit hole by measuring the concrete value of the fatigue point experimentally. Until now, all documented experiments on the overstretched regime [ABD16; KF17; LW20] have focused on rather large values of q, and only used weak lattice reduction (LLL [LLL82], BKZ with blocksize 20): their goal was to demonstrate the claimed general behaviour when parameters are far in the overstretched regime. On the contrary, we focus the attention to the fatigue point for this preliminary experiment. That is, we ran strong reduction (progressive-BKZ [Sch87; AWHT16] up to blocksize 60) until a vector related to the secret key appeared for a range of moduli q. We distinguished the standard regime from the overstretched regime by classifying according to which event occurs first

- Secret Key Recovery (SKR$_\kappa$): a vector as short as a secret key vector is inserted in the basis at any given position κ.
- Dense Sublattice Discovery (DSD$_\kappa$): a vector strictly longer than

the secret key but belonging to the dense sublattice generated by the secret key is inserted in the basis at any given position κ.

The result (Fig. 7.2) is rather striking: for $n = 127$, we start seeing a deviation from the standard regime for q as small as 700, while a naive interpretation of the prediction by Kirchner and Fouque [KF17] would suggest a fatigue point at $q \approx n^{2.783} \approx 700\,000$. We can conclude either that the asymptotic bound is not tight, or that the hidden asymptotic term (the $o(1)$ in $n^{2.783+o(1)}$) is significantly negative in practice. In any case, the bound of Kirchner and Fouque does not seem to provide accurate concrete predictions.

Remark

At this point, we should clarify why the DSD event should essentially be considered a successful attack. First, for q not too much larger than the fatigue point, an SKR event typically quickly follows after the DSD event; what happens is that DSD events cascade, until the full dense sublattice has been extracted: the first half of the reduced basis precisely generates the dense sublattice. Lattice reduction will happen independently on each half of the basis, meaning that the dimension of the search space for the secret key has effectively been halved, and therefore making the problem much easier.

However, as q increases, DSD becomes easier and easier, to the point that it becomes even easier than secret key recovery within the dense sublattice. In other terms, there is a *superstretched* regime for larger q, where DSD does not directly lead to SKR.

Nevertheless, we argue —essentially rephrasing [ABD16]— that the DSD event is typically sufficient for an attack. First, the dense sublattice vector discovered is of length significantly lower than q; in an FHE scheme such as [BLLN13] it is sufficient to decrypt fresh ciphertexts.[4] Secondly, in the case of cyclotomic or circulant NTRU, it is possible to recover the secret key from the dense sublattice by other means than pure lattice reduction; in particular the recent line of work on the principal ideal-SVP [EHKS14; CDPR16; Bia+17] showed that this can be done classically in subexponential time $\exp(\tilde{O}(\sqrt{n}))$ and quantumly in polynomial time.

[4] The secret key being shorter is only required to deal with ciphertexts obtained by homomorphic computation.

7.1.1 Contributions

Having identified precisely what event distinguishes the standard regime of NTRU from its overstretched regime, we may now proceed to a refined analysis, and determine precisely both the fatigue point and the precise cost[5] of attacks in the overstretched regime. The refined analysis in this work diverges from the one of Kirchner and Fouque [KF17] on the following points:

1. we exploit the fact that BKZ runs SVP on large blocks ($\beta \geq 2$) not only to deduce the shape of the basis, but also to actually discover dense sublattice vectors,

2. we do not solely focus on the behaviour at position $\kappa = n - \beta + 1$ out of $d = 2n$ dimensions, but instead predict the most relevant position,

3. we propose an average-case analysis of volumes of the relevant lattices and sublattices, leading to a concrete prediction rather than a worst-case bound,

4. we also validate the intermediate and final predictions quantitatively with extensive experiments.

We note that contributions 1 and 2 alone already give us an important asymptotic result: the fatigue point of NTRU is indeed lower than predicted by Kirchner and Fouque, namely, it should happen at $q = n^{2.484+o(1)}$ instead of $n^{2.783+o(1)}$.

Furthermore, for the concrete average case analysis we differentiates between the *circulant* version of NTRU [HPS98] and its *matrix* version [CG05; Gen+19]. We note minor deviations in the concrete analysis of volumes of relevant sublattices, that on average slightly favours the attacker in the matrix case, but also shows a larger variance in the concrete hardness of the circulant case. The concrete analysis in this work is versatile, as one only has to estimate the volume of the dense sublattice to analyse a different variant.

[5] In this work, we only measure cost of lattice reduction in terms of the required BKZ blocksize; the computational cost of BKZ is essentially an orthogonal question.

In summary: we achieve an explicative and predictive model for the fatigue of NTRU, with concrete predictions confirmed in practice. In particular, the fatigue point is estimated to be at $q \approx 0.004 \cdot n^{2.484}$ for $n > 100$ and ternary errors. All artefacts for experiments and predictions are open-source[6], and are based on the FPLLL and FPyLLL libraries [tea21a; tea21b].

Impact

We wish to clarify that this work *does not* contradict the concrete security of the NTRU candidate to the NIST competition [Che+20]; on the contrary, we close a pending question regarding a potential vulnerability.

Limitation: the lucky-lifts

During the experiments, we also noted rare occurrence of DSD events that qualitatively differ from what we expected. Namely, the vector from the dense sublattice was found at positions κ quite larger than what was predicted by the model, as shown in Fig. 7.11 (a). More remarkable, these vectors were extremely unbalanced: their $2n - \kappa$ last (Gram-Schmidt) coordinates were much smaller than the κ first coordinates (Fig. 7.11 (b)). We call these DSD events lucky-lifts (DSD-LL), while the one we model and mostly observe are called after the Pataki-Tural Lemma (DSD-PT). Despite those two phenomena being very distinct, they nevertheless occured for the same BKZ blocksizes β, at least in the range of parameters we could experiment with.

It could very well be that these rare DSD-LL events are just artefacts of the modest parameters of the experiments and that these events vanish as the dimension grows. Yet, as they seem of a very different nature, a definitive conclusion would require a dedicated study.

7.1.2 Organisation

We introduce some preliminaries, the NTRU lattice, and the state-of-the-art estimates in Section 7.2. In Section 7.3 we introduce the new DSD-PT estimate and give an asymptotic analysis. In Section 7.4 we

[6] Available at: `https://github.com/WvanWoerden/NTRUFatigue`

give an average-case analysis to construct a concrete estimator. In the final Section 7.5 we compare the estimate with experiments.

7.2 The NTRU lattice and estimates

7.2.1 NTRU and lattice attacks

We start with the historical definition of NTRU.

Definition 99 (NTRU). *Let n be prime, q a positive integer and let $\mathbf{f}, \mathbf{g} \in (\mathbb{Z}/q\mathbb{Z})[X]$ be polynomials of degree n with small coefficients sampled from some distribution χ under the condition that $\mathbf{f}$ is invertible in $\mathcal{R}_q := (\mathbb{Z}/q\mathbb{Z})[X]/(X^n - 1)$. The pair $(\mathbf{f}, \mathbf{g})$ forms the secret key, and the public key is defined as $\mathbf{h} := \mathbf{g}/\mathbf{f} \bmod q$. The* NTRU problem *is to recover any rotation $(X^i \cdot \mathbf{f}, X^i \cdot \mathbf{g})$ of the secret key from $\mathbf{h}$.*

For *NTRUencrypt* [HPS98; Che+20] $\mathbf{f}$ and $\mathbf{g}$ have ternary coefficients, with a fixed number of about $n/3$ of each value in $\{-1, 0, 1\}$. For the analysis we consider the case where each coefficient is sampled from a discrete Gaussian over $\mathbb{Z}$ with some variance $\sigma^2 > 0$. For simplicity the ternary case is treated as a discrete Gaussian with variance $\sigma^2 = \frac{2}{3}$.

More generally we consider a matrix description of NTRU where the polynomials are replaced by matrices $\mathbf{F}, \mathbf{G}, \mathbf{H} \in \mathbb{Z}^{n \times n}$ such that $\mathbf{H} := \mathbf{G} \cdot \mathbf{F}^{-1} \bmod q$ [CG05; Gen+19]. Variants of NTRU, e.g., based on different algebraic rings [Ber+20], can be encoded in the structure of the matrices. For example, the original problem can be encoded by setting $\mathbf{F}_{i,j} := f_{(i+j \bmod n)}$ where $\mathbf{f} = \sum_{i=0}^{n-1} f_i X^i$, for each polynomial respectively. We call the original variant *circulant NTRU*, based on the resulting shape of the matrices $\mathbf{F}, \mathbf{G}$, and we treat $\mathbf{f}, \mathbf{g}$ as n-dimensional vectors. We also consider the variant, called *matrix NTRU*, where the matrices $\mathbf{F}, \mathbf{G}$ have no extra structure and the coefficients are independently sampled from a discrete Gaussian.

To reduce the NTRU problem to a lattice problem we define the *NTRU lattice*, which contains a particularly *dense* sublattice generated by the secret key.

Definition 100. *Let $(n, q, \mathbf{F}, \mathbf{G}, \mathbf{H})$ be an NTRU instance. We define the NTRU lattice as*

$$\mathcal{L}^{\mathbf{H},q} := \begin{pmatrix} q\mathbf{I}_n & \mathbf{H} \\ \mathbf{0} & \mathbf{I}_n \end{pmatrix} \cdot \mathbb{Z}^{2n},$$

and its (secret) dense sublattice of rank n by:

$$\mathcal{L}^{\mathbf{GF}} := \mathbf{B}^{\mathbf{GF}} \cdot \mathbb{Z}^n \subset \mathcal{L}^{\mathbf{H},q}, \text{ where } \mathbf{B}^{\mathbf{GF}} := \begin{pmatrix} \mathbf{G} \\ \mathbf{F} \end{pmatrix}.$$

Solving the NTRU problem is equivalent to recovering the dense sublattice basis $\mathbf{B}^{\mathbf{GF}} = [\mathbf{G}; \mathbf{F}]$ up to some permutation of the columns. For uniformity of notation we will denote such a column by $(\mathbf{g}; \mathbf{f})$. These column vectors have a length of about $\|(\mathbf{g}; \mathbf{f})\| \approx \sqrt{2n\sigma^2}$, which for common parameters is much shorter than the expected minimal length $\mathrm{gh}(\mathcal{L}^{\mathbf{H},q}) \approx \sqrt{nq/(\pi e)}$ of the full lattice $\mathcal{L}^{\mathbf{H},q}$ for a truly uniform random $\mathbf{H} \in (\mathbb{Z}/q\mathbb{Z})^{n \times n}$. To recover the secret key we thus have to find these exceptionally short vectors in the full lattice $\mathcal{L}^{\mathbf{H},q}$.

In [CS97] Coppersmith and Shamir showed that we can slightly relax the problem as any small vector from the dense sublattice $\mathcal{L}^{\mathbf{GF}}$ is enough to decode a message. We therefore focus the analysis on the recovery of elements from $\mathcal{L}^{\mathbf{GF}}$, and not (directly) on the full secret basis $\mathbf{B}^{\mathbf{GF}}$. To recover short vectors we resort to lattice reduction.

7.2.2 Lattice reduction on q-ary lattices

While by now the behaviour of BKZ on random lattices is reasonably understood, this is less the case for q-ary lattices (for certain parameters) such as the NTRU lattice $\mathcal{L}^{\mathbf{H},q}$.

Definition 101 (q-ary lattices). *A lattice $\mathcal{L}$ of dimension d is said to be q-ary if for some $q > 0$ we have*

$$q\mathbb{Z}^d \subset \mathcal{L} \subset \mathbb{Z}^d.$$

Note that the first n basis vectors of $\mathcal{L}^{\mathbf{H},q}$ are orthogonal q-vectors $(q, 0, \ldots, 0)$, $(0, q, 0, \ldots, 0), \ldots$, and so the initial basis profile starts with $\|\tilde{\mathbf{b}}_0\| = \cdots = \|\tilde{\mathbf{b}}_{n-1}\| = q$. Additionally after projecting away from these q-vectors, the remaining basis vectors are again orthogonal

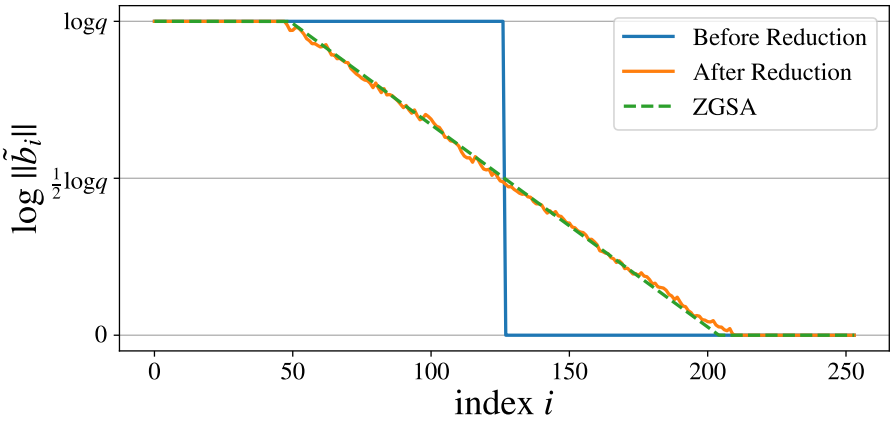


Figure 7.3: The Z-shape of an NTRU-lattice basis with parameters $q = 1031, n = 127$ before and after LLL reduction, versus the ZGSA.

with length 1, and thus we have $\|\tilde{\mathbf{b}}_n\| = \cdots = \|\tilde{\mathbf{b}}_{d-1}\| = 1$. Note that in the BKZ algorithm the length of $\mathbf{b}_0$ can not increase, and is thus always at most q. Also $\mathbf{b}_1$ can not increase in length if $\mathbf{b}_0$ remains unchanged, and so on. For dual-BKZ or the self-dual LLL the profile lengths can not drop below 1 anywhere by the same reasoning. Still LLL and BKZ guarantee that the profile slope in the middle is not too steep. So after LLL reduction the profile must be flat at the start and end, and have a sloped part in the middle, we call this a Z-shape [AD21; AL22]. Because BKZ is not self-dual we do not have any guarantee that the last profile elements do not drop below 1, however we could for example run BKZ only on an appropriate middle context $\mathcal{L}_{[n-m:n+m)}$ to force this behaviour. With this description one would expect the middle part to follow the GSA, leading to an alternative heuristic for q-ary lattices.

Heuristic 102 (Z-shape Geometric Series Assumption (ZGSA)). *Let $\mathbf{B}$ be a basis of a $2n$-dimensional q-ary lattice $\mathcal{L}$ with n q-vectors. After BKZ-β reduction the profile has the following shape:*

$$\|\tilde{\mathbf{b}}_i\| = \begin{cases} q & \text{if } i \leq n-m, \\ \sqrt{q} \cdot \alpha_\beta^{\frac{2n-1-2i}{2}}, & \text{if } n-m < i < n+m-1, \\ 1, & \text{if } i \geq n+m-1, \end{cases}$$

where $\alpha_\beta = \mathrm{gh}(\beta)^{2/(\beta-1)}$, *and* $m = \frac{1}{2} + \frac{\log(q)}{2\log(\alpha_\beta)}$.

Similar to the regular GSA this gives us a good first order estimate, as can be observed in Figure 7.3. Asymptotically setting $\beta = \mathcal{B} \cdot n$ and $q = n^{\mathcal{Q}}$, we obtain $\log(\alpha_\beta) = \frac{\log(n)}{\mathcal{B} \cdot n} + O\left(n^{-1}\right)$, and $m = \frac{1}{2}\mathcal{Q}\mathcal{B} \cdot n + O\left(\frac{n}{\log(n)}\right)$.

7.2.3 Estimates

The main question of this work is to better understand how BKZ recovers the dense sublattice $\mathcal{L}^{\mathbf{GF}}$ from an NTRU lattice $\mathcal{L}^{\mathbf{H},q}$. Several works exist that give estimates on the blocksize β for which BKZ successfully recovers the secret key $(\mathbf{g}, \mathbf{f})$, or more generally a vector from the dense sublattice. We discuss the state-of-the-art estimates, one known as the *2016 Estimate* [ADPS16] with further refinements [DDGR20; PV21], and one by Kirchner and Fouque [KF17].

While the 2016 Estimate already gives a clear explanation *how* BKZ recovers a suitable vector, the Kirchner and Fouque estimate is only based on an impossibility result. To be more precise about what we mean with recovery we define the following two events.

Definition 103 (BKZ Events). *For a BKZ run on an NTRU lattice* $\mathcal{L}$ *with dense sublattice* $\mathcal{L}^{\mathbf{GF}}$ *we define two events:*

1. ***Secret Key Recovery (SKR)****: The first time one the secret keys* $(\mathbf{g}; \mathbf{f})$ *is inserted.*
2. ***Dense Sublattice Discovery (DSD)****: The first time a dense lattice vector* $\mathbf{v} \in \mathcal{L}^{\mathbf{GF}}$ *strictly longer than the secret key(s) is inserted.*

We further specify SKR_κ *and* DSD_κ *when the insertion takes place at position* κ *in the basis.*

2016 Estimate [ADPS16] for SKR

The 2016 Estimate is aimed at the more general problem of detecting an unusually short vector in a lattice. To obtain an estimate for the NTRU problem, and more specifically the SKR event, we apply it to the unusually short vector $(\mathbf{g}; \mathbf{f}) \in \mathcal{L}^{\mathbf{H},q}$.

Heuristic claim 104 (SKR – 2016 Estimate). *Let $\mathcal{L}$ be a lattice of dimension d and let $\mathbf{v} \in \mathcal{L}$ be a unusually short vector $\|\mathbf{v}\| \ll \mathrm{gh}(\mathcal{L})$. Then under the Geometric Series Assumption BKZ recovers $\mathbf{v}$ if*

$$\sqrt{\beta/d} \cdot \|\mathbf{v}\| < \sqrt{\alpha_\beta}^{2\beta-d-1} \cdot \mathrm{vol}(\mathcal{L})^{1/d},$$

where $\alpha_\beta = \mathrm{gh}(\beta)^{2/(\beta-1)}$.

Justification. The left hand side of the inequality is an estimate for $\|\pi_{d-\beta}(\mathbf{v})\|$, while the right hand size is the expected norm of $\tilde{\mathbf{b}}_{d-\beta}$ under the GSA. When the inequality is satisfied we expect that the shortest vector in $\mathcal{L}_{[d-\beta:d)}$ is in fact (a projection of) the unusually short vector, and thus it is inserted by BKZ at position $d-\beta$. See Figure 7.4 for an illustration.

Except for very small blocksizes β, the unusually short vector $\mathbf{v}$ is recovered from its projection $\pi_{d-\beta}(\mathbf{v})$ with high probability, either directly by Babai's nearest plane algorithm, or by later BKZ tours on the blocks $\mathcal{L}_{[d-2\beta+1:d-\beta+1)}, \mathcal{L}_{[d-3\beta+2:d-2\beta+2)}, \ldots$; lifting the vector block by block. △

For q-ary lattices we can easily change the estimate to make use of the ZGSA instead, although for successful blocksizes $\tilde{\mathbf{b}}_{d-\beta}$ will not lie on the flat tail-part, and thus this will not change anything. Additionally for q-ary lattices it can be beneficial to apply the estimate not to the full lattice but on some projected sublattice $\mathcal{L}_{[i:d)}$ for $i \leq n$; the left hand side of the equation is expected to remain unchanged, while the right hand side might increase as $\mathrm{vol}(\mathcal{L})$ loses only a factor q^i and the dimension decreases by i. Note that we do not necessarily have to explicitly let BKZ act on this projected sublattice, as BKZ already does this naturally.

Asymptotics. Consider the NTRU lattice $\mathcal{L}^{\mathbf{H},q}$ and suppose that $q = \Theta(n^{\mathcal{Q}})$, $\|\mathbf{v}\| = \|(\mathbf{g};\mathbf{f})\| = \Theta(n^{\mathcal{S}})$ and $\beta = (\mathcal{B} + o(1))n$. Applying the 2016 Estimate the right hand side of the inequality is minimised when only keeping $k = \min\left((\sqrt{2\mathcal{B}\mathcal{Q}} - 1)n, n\right)$ of the q-vectors, so by applying the estimate to the projected sublattice $\mathcal{L}^{\mathbf{H},q}_{[n-k:2n)}$. For $\mathcal{S} \geq 1$ we have $k = n$, and solving the equation gives $\mathcal{B} = \frac{2}{\mathcal{Q}+2-2\mathcal{S}}$. For $\mathcal{S} < 1$ we have $k = (\sqrt{2\mathcal{B}\mathcal{Q}} - 1)n$, and solving gives $\mathcal{B} = \frac{2\mathcal{Q}}{(\mathcal{Q}+1-\mathcal{S})^2}$. Note in particular that in terms of q we require a blocksize of $\beta = \tilde{\Theta}\left(n/\log(q)\right)$.

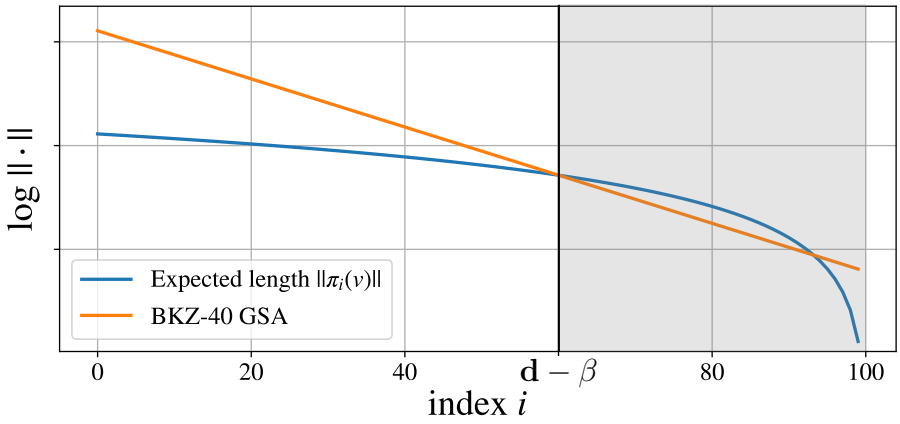


Figure 7.4: Illustration of the 2016 Estimate on a 100-dimensional lattice with an unusually short vector $\mathbf{v}$. Around $\beta = 40$ the projection $\pi_{100-\beta}(\mathbf{v})$ is expected to be the shortest vector in the projected sublattice $\mathcal{L}_{[d-\beta:d)}$, which is thus recovered by the SVP call on this block inside BKZ-β.

Refinements. The 2016 Estimate gives a clear explanation on how and where the secret vector is recovered, namely its projection is found in the last BKZ block. This also allows to further refine the estimate and give concrete predictions. For example by using a BKZ-simulator instead of the GSA, and by accounting for the probability that after the projection $\|\pi_{d-\beta}(\mathbf{v})\|$ has been found, it is successfully lifted to the full vector $\mathbf{v}$. Also instead of working with the expected length of the projection, we can directly model the probability distribution under the assumption that $\mathbf{v}$ is distributed as a Gaussian vector. Such refinements were applied in [DDGR20; PV21], and the resulting concrete predictions match with experiments to recover an unusually short vector. Current simulators for the behaviour of the BKZ algorithm do not account correctly for the Z-shape [AD21]. Therefore, in this work, we will rely on the (Z)GSA for the basis shape, and we adjust the slope to account for the speed of convergence using experimentally determined values. We also use the advanced probabilistic model for the detection and lifting of the short vector.

For NTRU there is not just a single unusually short vector, but there are $n = d/2$ of them, which makes it more likely that at least one

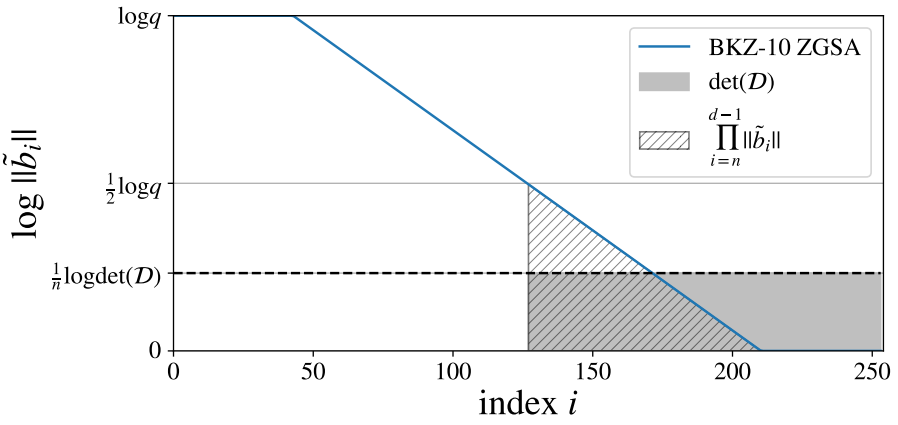


Figure 7.5: Illustration of both sides of the equation when applying Lemma 105 to $\mathcal{D} = \mathcal{L}^{\mathbf{GF}} \subset \mathcal{L}^{\mathbf{H},q}$ and $J = \{n, \ldots, d-1\}$. Following the Pataki and Tural Lemma, the striped area can be at most as large as the solid area. For larger blocksize β, the BKZ-β ZGSA contradicts this, and thus the basis must have deviated from it (and thereby have revealed some information).

of them is recovered. Because the refined concrete estimator already works with a probability distribution, we can easily take multiple vectors into account. The resulting predictions for the SKR event match the experiments reasonably well for smallish q as can be seen in Figure 7.2. For large q, the so-called *overstretched* regime, the estimate is however too pessimistic.

Kirchner–Fouque estimate [KF17] for DSD

In 2016 Albrecht, Bai and Ducas [ABD16] showed that for very large values of q one can mount an algebraic *subfield attack* on the cyclotomic NTRU problem with subexponential or even polynomial complexity. This allowed them to break several homomorphic encryption schemes that relied on NTRU in the overstretched regime.

However soon after, Kirchner–Fouque [KF17] showed that this elaborate algebraic attack was unnecessary: (dual-)BKZ already behaves much better in this regime than the 2016 Estimate predicts,

leading to the same asymptotic improvements. The key idea behind their analysis is that in the overstretched regime the NTRU lattice $\mathcal{L}^{\mathbf{H},q}$ contains an exceptionally dense sublattice $\mathcal{L}^{\mathbf{GF}}$ of low volume. This gives a constraint on the basis profile via the following lemma by Pataki and Tural.

Lemma 105 (Pataki and Tural [PT08]). *Let $\mathcal{L}$ be a d-dimensional lattice with basis $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$. For any k-dimensional sublattice $\mathcal{L}' \subset \mathcal{L}$ we have*

$$\mathrm{vol}(\mathcal{L}') \geq \min_J \prod_{j \in J} \|\tilde{\mathbf{b}}_j\|,$$

where J ranges over the k-size subsets of $\{0, \ldots, d-1\}$.

Applying Lemma 105 to the n-dimensional sublattice $\mathcal{L}^{\mathbf{GF}} \subset \mathcal{L}^{\mathbf{H},q}$, and assuming a non-increasing profile, we obtain an upper bound on the volume of $\mathcal{L}^{\mathbf{H},q}_{[n:2n)}$. See Figure 7.5 for an illustration of these volumes. Assuming the ZGSA the latter volume increases when running BKZ-β for increasing blocksizes, eventually contradicting the upper bound. This allows us to detect if a q-ary lattice is in fact an NTRU lattice, but additionally Kirchner–Fouque argue that BKZ must *somehow* have detected the dense sublattice after this point. Based on this impossibility argument they introduced the following estimate.

Heuristic claim 106 (DSD – Kirchner–Fouque Estimate). *Let $\mathcal{L}^{\mathbf{H},q}$ be an NTRU lattice of dimension $2n$, with dense sublattice $\mathcal{L}^{\mathbf{GF}} \subset \mathcal{L}^{\mathbf{H},q}$. Under the Z-shape Geometric Series Assumption BKZ-β triggers the DSD event if*

$$\mathrm{vol}(\mathcal{L}^{\mathbf{GF}}) < q^{\frac{m-1}{2}} \cdot \alpha_\beta^{-\frac{1}{2}(m-1)^2},$$

where $\alpha_\beta = \mathrm{gh}(\beta)^{2/(\beta-1)}$, and $m = \frac{1}{2} + \frac{\log(q)}{2\log(\alpha_\beta)}$.

To apply this estimate we can bound $\mathrm{vol}(\mathcal{L}^{\mathbf{GF}})$ using the Hadamard inequality by $\|(\mathbf{g}; \mathbf{f})\|^n$. As a first approximation this is reasonably tight because the secret basis $\mathbf{B}^{\mathbf{GF}}$ is close to orthogonal.

Asymptotics. Consider the NTRU lattice $\mathcal{L}^{\mathbf{H},q}$ and suppose that $q = \Theta(n^{\mathcal{Q}})$, $\|(\mathbf{g};\mathbf{f})\| = \Theta(n^{\mathcal{S}})$ and $\beta = (\mathcal{B}+o(1))n$. We apply the Kirchner–Fouque Estimate using that $m \approx \frac{\mathcal{B}\mathcal{Q}}{2}n$ and $\alpha_\beta \approx (\mathcal{B}n)^{1/(\mathcal{B}n)}$.The left hand side of the inequality is bounded by $n^{n\mathcal{S}+o(n)}$ and the right hand side equals $n^{\frac{\mathcal{B}\mathcal{Q}^2}{8}n+o(n)}$; solving gives $\mathcal{B} \geq \frac{8\mathcal{S}}{\mathcal{Q}^2}$. Note that in terms of n and q we require a blocksize of $\beta = \tilde{\Theta}\left(n/\log^2(q)\right)$, improving upon the 2016 Estimate by a factor $\log(q)$. So for large enough q the Kirchner–Fouque Estimate predicts a lower successful blocksize than the 2016 Estimate. We call the value of q for which BKZ starts to behave better than predicted by the 2016 Estimate the *fatigue point*. For the common situation that $\mathcal{S} = \frac{1}{2}$, e.g., when each secret coefficient has standard deviation $\sigma = \Theta(1)$, the Kirchner–Fouque Estimate predicts that the fatigue point lies at some $q \leq n^{2.783+o(1)}$.

7.3 A new dense sublattice discovery estimate

7.3.1 Preliminary experiments

Both the 2016 Estimate and the Kirchner–Fouque Estimate analyse an event that leads to successful recovery of a vector of the dense NTRU sublattice. This only gives an upper bound on the hardness; a different event leading to the recovery might happen at a lower blocksize. Additionally the Kirchner–Fouque Estimate is only based on an impossibility result and gives no explanation as to how BKZ actually recovers a vector from the dense sublattice. In order to derive a tight estimate we first run experiments to track down at which point a dense sublattice vector is actually found during the BKZ tours, i.e., when the DSD_κ event is triggered and at what position. Then we model this event in order to hopefully derive a tight estimate.

We run progressive BKZ on NTRU lattices $\mathcal{L}^{\mathbf{H},q}$ for fixed parameters $n = 127, \sigma^2 = \frac{2}{3}$, and several moduli q. For each BKZ insertion at position κ we check if the inserted vector belongs to the dense sublattice $\mathcal{L}^{\mathbf{GF}}$, and thereby if the SKR_κ or DSD_κ event takes place, after which we stop.

The results are shown in Figure 7.2. We take a closer look at the observed SKR_κ and DSD_κ events and where they are triggered. We

can group the observations in three typical circumstances.

- **SKR-2016**. The SKR_κ event is mostly triggered for small values of q, and this mostly happens at the position $\kappa = 2n - \beta$, so in the last block $[2n - \beta : 2n)$, or slightly earlier. This coincides exactly with the $\mathrm{SKR}_{2n-\beta}$ event as predicted by the 2016 Estimate [ADPS16] with further refinements [AGVW17; DDGR20; PV21].

- **DSD-PT**. The DSD_κ event is mostly triggered at positions $\kappa = n + k - \beta$ for $0 < k \ll n$. The inserted dense vector $\mathbf{v}$ is often significantly longer than the secret key but still shorter than the q-vectors. On closer inspection the projected length $\|\pi_{n+k-\beta}(\mathbf{v})\|$ is close to the expected length $\sqrt{\frac{\beta}{n+k}}\|\mathbf{v}\|$ for all instances, more specifically the length of $\mathbf{v}$ is well balanced over the Gram-Schmidt directions $\tilde{\mathbf{b}}_0, \ldots, \tilde{\mathbf{b}}_{n+k-1}$. We name these events after the Pataki–Tural Lemma (DSD-PT).

- **DSD-LL**. For a few instances the DSD_κ event is triggered at large positions κ, up to $2n - \beta$. The inserted dense vector $\mathbf{v}$ is again significantly longer than the secret key, but it has an unexpectedly short projection $\pi_\kappa(\mathbf{v})$ on the BKZ block $[\kappa : \kappa + \beta)$. We call these events lucky-lifts (DSD-LL).

The DSD-LL event could potentially be explained by the relatively large amount of shortish vectors in the close to orthogonal dense sublattice $\mathcal{L}^{\mathbf{GF}}$ compared to what one would expect based on the Gaussian Heuristic. These many vectors might compensate for the low probability event that: (1) such a long vector has such a short projection, and (2) the projected vector is correctly lifted by Babai's nearest plane algorithm (thus a *lucky lift*). The DSD-LL event remains rare for all parameters we used in the experiments, and the successful blocksizes do not seem to deviate from the DSD-PT events. Although we think this circumstance deserves further analysis we therefore base the estimate on the more common DSD-PT event.

For the DSD-PT event the projected length $\|\pi_{n+k-\beta}(\mathbf{v})\|$ is close to $\sqrt{\frac{\beta}{n+k}}\|\mathbf{v}\|$, and thus the inserted dense vector $\mathbf{v}$ must in fact be (close to) a shortest vector of the intersected sublattice $\mathcal{L}^{\mathbf{H},q}_{[0:n+k)} \cap \mathcal{L}^{\mathbf{GF}}$. If not,

the shortest vector would typically have an even smaller projection and would thus be inserted instead. For ease of analysis we therefore assume that $\mathbf{v}$ is a shortest vector of $\mathcal{L}^{\mathbf{H},q}_{[0:n+k)} \cap \mathcal{L}^{\mathbf{GF}}$. In short the new estimate can be described as follows.

Heuristic claim 107 (DSD-PT estimate). *A tour of BKZ-β triggers the DSD event if*

$$\pi_{n+k-\beta}(\mathbf{v}) < \|\tilde{\mathbf{b}}_{n+k-\beta}\|,$$

where $\mathbf{v}$ is a shortest nonzero vector of $\mathcal{L}^{\mathbf{H},q}_{[0:n+k)} \cap \mathcal{L}^{\mathbf{GF}}$ for some $0 < k \leq n$.

7.3.2 Asymptotic analysis

For $0 \leq r \leq 2n$ denote the (possibly trivial) intersected sublattice by $\mathcal{L}^{\mathbf{GF}}_{\cap[0:r)} := \mathcal{L}^{\mathbf{H},q}_{[0:r)} \cap \mathcal{L}^{\mathbf{GF}}$. To directly apply Claim 107 we are interested in the length of $\mathbf{v}$, and thus the value of $\lambda_1\big(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\big)$. We break down the analysis into several steps. In order to obtain a bound on the first minimum we first compute a bound on the volume of the intersection $\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}$ in terms of the basis profile and the volume of $\mathcal{L}^{\mathbf{GF}}$. Together with the ZGSA and a simple bound for $\mathrm{vol}(\mathcal{L}^{\mathbf{GF}})$ we can then apply Minkowski's bound on the first minimum. By optimising $\kappa = n+k-\beta$ we obtain a new asymptotic estimate.

Intersection.

To understand the behaviour of the volume of the intersected lattice we first need a small technical Lemma.

Lemma 108 ([DDGR20]). *Given a lattice $\mathcal{L}$ with volume $\mathrm{vol}(\mathcal{L})$, and a primitive dual vector $\mathbf{v} \in \mathcal{L}^*$. Let $\mathbf{v}^\perp$ denote the subspace orthogonal to $\mathbf{v}$. Then $\mathcal{L} \cap \mathbf{v}^\perp$ is a (possibly trivial) lattice with volume $\mathrm{vol}(\mathcal{L} \cap \mathbf{v}^\perp) = \|\mathbf{v}\| \cdot \mathrm{vol}(\mathcal{L})$.*

Recall that we defined $\mathrm{vol}(\{0\}) = 1$ for the trivial lattice. The following Lemma generalises the Pataki–Tural Lemma on which the estimate of Kirchner–Fouque is based. More specifically the Pataki–Tural Lemma only considers the case where the intersection is always trivial ($s = 0$).

Lemma 109 (Generalisation of [PT08]). *Let $\mathcal{L}$ be a d-dimensional lattice with basis $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$, and consider the (possibly trivial) sublattice $\mathcal{L}_{[0:s)}$ for $0 \leq s \leq 2n$. For any n-dimensional sublattice $\mathcal{L}' \subset \mathcal{L}$ we have*

$$\mathrm{vol}(\mathcal{L}_{[0:s)} \cap \mathcal{L}') \leq \mathrm{vol}(\mathcal{L}') \cdot \left(\min_J \prod_{j \in J} \|\tilde{\mathbf{b}}_j\| \right)^{-1},$$

where $k := \dim(\mathcal{L}_{[0:s)} \cap \mathcal{L}')$ and J ranges over the $(n-k)$-size subsets of $\{s, \ldots, d-1\}$.

Proof. We write $\mathcal{L}'_{\cap[0:r)} := \mathcal{L}_{[0:r)} \cap \mathcal{L}'$. For $j = k, \ldots, n$ we define $s_j \in \{s, \ldots, d\}$ as the maximal index such that $\dim\left(\mathcal{L}'_{\cap[0:s_j)}\right) = j$, i.e., we obtain the following strict chain of sublattices:

$$\mathcal{L}'_{\cap[0:s)} = \mathcal{L}'_{\cap[0:s_k)} \subsetneq \mathcal{L}'_{\cap[0:s_{k+1})} \subsetneq \cdots \subsetneq \mathcal{L}'_{\cap[0:s_n)} = \mathcal{L}'.$$

Fix $j \in \{k, \ldots, n-1\}$. Because the basis vectors $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$ are linearly independent we have that $\mathcal{L}'_{\cap[0:s_{(j+1)})} = \mathcal{L}'_{\cap[0:s_j+1)}$. This allows us to focus on the volume decrease from index $s_j + 1$ to s_j, for which we know that

$$\mathcal{L}'_{\cap[0:s_j)} = \mathcal{L}'_{\cap[0:s_j+1)} \cap (\tilde{\mathbf{b}}_{s_j})^{\perp},$$

where $(\tilde{\mathbf{b}}_{s_j})^{\perp}$ denotes the subspace orthogonal to $\tilde{\mathbf{b}}_{s_j}$. The corresponding dual basis of $\mathbf{b}_0, \ldots, \mathbf{b}_{s_j}$ contains a dual vector $\mathbf{d} \in \mathcal{L}^*_{[0:s_j+1)}$ of length $\left\|\tilde{\mathbf{b}}_{s_j}\right\|^{-1}$ with $\mathrm{span}(\mathbf{d}) = \mathrm{span}(\tilde{\mathbf{b}}_{s_j})$. Let π be the orthogonal projection onto $\mathrm{span}\left(\mathcal{L}'_{\cap[0:s_j+1)}\right)$, then $\pi(\mathbf{d}) \in \left(\mathcal{L}'_{\cap[0:s_j+1)}\right)^*$, and

$$\mathcal{L}'_{\cap[0:s_j)} = \mathcal{L}'_{\cap[0:s_j+1)} \cap \mathbf{d}^{\perp} = \mathcal{L}'_{\cap[0:s_j+1)} \cap \pi(\mathbf{d})^{\perp}.$$

Let $m \in \mathbb{Z}_{\geq 1}$ be such that $\pi(\mathbf{d})/m$ is primitive w.r.t. $\left(\mathcal{L}'_{\cap[0:s_j+1)}\right)^*$, then by Lemma 108 we obtain:

$$\begin{aligned} \mathrm{vol}\left(\mathcal{L}'_{\cap[0:s_j)}\right) &= \mathrm{vol}\left(\mathcal{L}'_{\cap[0:s_j+1)}\right) \cdot \|\pi(\mathbf{d})/m\| \\ &\leq \mathrm{vol}\left(\mathcal{L}'_{\cap[0:s_{j+1})}\right) \cdot \|\tilde{\mathbf{b}}_{s_j}\|^{-1}. \end{aligned}$$

We conclude by chaining the above inequality for $j = k, \ldots, n-1$. □

Before recovering a dense lattice vector we heuristically assume that there is no special relation between the current lattice basis and the dense sublattice. More specific we can consider that the span of $\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}$ and that of $\mathcal{L}^{\mathbf{GF}}$ behave like random n-dimensional subspaces, and thus they have a trivial intersection with high probability in the $2n$-dimensional space. As a direct result we expect that $\dim\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right) = k$ for $k = 0, \ldots, n$. Applying this to Lemma 109 we obtain the following corollary.

Corollary 110. *Let $\mathcal{L}^{\mathbf{H},q}$ be an NTRU lattice with dense sublattice $\mathcal{L}^{\mathbf{GF}}$ of dimension n, if $\dim\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right) = k$ for some $k \geq 0$, then*

$$\operatorname{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right) \leq \operatorname{vol}\left(\mathcal{L}^{\mathbf{GF}}\right) \cdot \left(\prod_{j=n+k}^{d-1} \|\tilde{\mathbf{b}}_j\|\right)^{-1}.$$

Note that Corollary 110 already shows that the new estimate can not be worse than the Kirchner–Fouque Estimate. Namely if the Kirchner–Fouque Estimate is triggered, then for intersection dimension $k = 1$ the right hand side is smaller than $\|\tilde{\mathbf{b}}_n\|$. Assuming a non-decreasing profile we then have $\lambda_1\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+1)}\right) = \operatorname{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+1)}\right) \leq \|\tilde{\mathbf{b}}_n\| \leq \left\|\tilde{\mathbf{b}}_{n+1-\beta}\right\|$, which implies that BKZ-β would find a dense sublattice vector in the block $\mathcal{L}_{[n+1-\beta:n+1)}$ (or earlier).

Volume dense sublattice.

To use Corollary 110 we also need to bound the volume of the dense sublattice $\mathcal{L}^{\mathbf{GF}}$. Because the secret basis is close to orthogonal the Hadamard Inequality $\operatorname{vol}(\mathcal{L}^{\mathbf{GF}}) \leq \|(\mathbf{g}; \mathbf{f})\|^n$ is sufficient as a first order approximation.

Conclusion.

To obtain a heuristic asymptotic estimate we will assume that before finding a dense lattice vector the basis follows the ZGSA shape.

Heuristic claim 111. *The BKZ algorithm with blocksize $\beta = \mathcal{B}n$ applied to an NTRU instance with parameters $q = \Theta(n^{\mathcal{Q}})$, $\|(\mathbf{g}; \mathbf{f})\| =$*

$O(n^{\mathcal{S}})$ triggers the DSD event if

$$\mathcal{B} = \frac{8\mathcal{S}}{\mathcal{Q}^2 + 1} + o(1).$$

Justification. By the Hadamard Inequality we have $\log(\mathrm{vol}(\mathcal{L}^{\mathbf{GF}})) \leq \mathcal{S}n\log(n) + O(n)$. Let $k := \mathcal{K}n > 0$ for some constant $0 < \mathcal{K} \leq 1$. Heuristically we expect that $\dim(\mathcal{L}^{\mathbf{H},q}_{[0:n+k)} \cap \mathcal{L}^{\mathbf{GF}}) = k$, and thus by Corollary 110 and by assuming the ZGSA we obtain a bound on the volume of the intersected sublattice:

$$\begin{aligned}
&\log(\mathrm{vol}(\mathcal{L}^{\mathbf{H},q}_{[0:n+k)} \cap \mathcal{L}^{\mathbf{GF}})) \\
&\leq \mathcal{S}n\log(n) - \frac{1}{2}\sum_{i=n+k}^{n+m-1}\left(Q + \frac{2n-1-2i}{\mathcal{B}n}\right)\log(n) + O(n) \\
&= \mathcal{S}n\log(n) - \frac{(\mathcal{B}\mathcal{Q}-2\mathcal{K})^2}{8\mathcal{B}}n\log(n) + O(n).
\end{aligned}$$

By Minkowski's bound we bound the first minimum using the above volume

$$\begin{aligned}
\log(\lambda_1(\mathcal{L}^{\mathbf{H},q}_{[0:n+k)} \cap \mathcal{L}^{\mathbf{GF}})) &\leq \frac{1}{2}\log(\mathcal{K}n) \\
&\quad + \frac{\log(\mathrm{vol}(\mathcal{L}^{\mathbf{H},q}_{[0:n+k)} \cap \mathcal{L}^{\mathbf{GF}}))}{\mathcal{K}n} + O(1) \\
&\leq \left(-\frac{(\mathcal{B}\mathcal{Q}-2\mathcal{K})^2}{8\mathcal{B}\mathcal{K}} + \frac{\mathcal{S}}{\mathcal{K}} + \frac{1}{2}\right)\log(n) + O(1).
\end{aligned}$$

After projecting onto the block $[n+k-\beta : n+k)$ the above short vector does not increase in length.[7] BKZ detects the projected dense lattice vector in this block if the length is less than $\left\|\tilde{\mathbf{b}}_{n+k-\beta}\right\| = \left(\frac{1}{2}\mathcal{Q} + \frac{\mathcal{B}-\mathcal{K}}{\mathcal{B}}\right)\log(n) + O(1)$. Solving for $\mathcal{B}$ shows that this is the case when

$$\mathcal{B} \geq \frac{2\sqrt{(2\mathcal{S}-\mathcal{K})^2 + \mathcal{K}^2\mathcal{Q}^2} + 2(2\mathcal{S}-\mathcal{K})}{\mathcal{Q}^2} + o(1).$$

[7] One may also be concerned that the short vector would collapse to the zero vector $\mathbf{0}$ after projection onto the block $[n+k-\beta : n+k)$, but this becomes increasingly unlikely as the dimension β of the block grows.

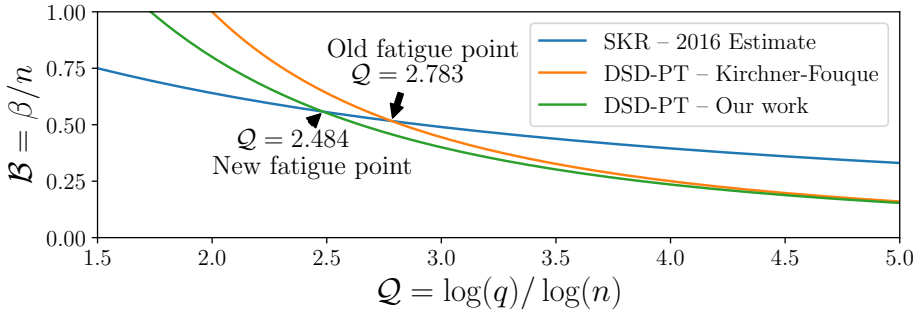


Figure 7.6: Comparison of asymptotic estimates and new fatigue point for $n \to \infty$ when the secret key coefficients have standard deviation $\sigma = \Theta(1)$.

When $\mathcal{K} = \frac{4\mathcal{S}}{\mathcal{Q}^2+1}$ the right hand side is minimised and we obtain that BKZ detects the projected dense lattice vector when $\mathcal{B} \geq \frac{8\mathcal{S}}{\mathcal{Q}^2+1}$, which concludes the claim. This routine computation can be verified symbolically via the sage notebook `claim3_5.ipynb`[8]. △

The new estimate gives an asymptotic improvement over the estimate by Kirchner and Fouque ($\frac{8\mathcal{S}}{\mathcal{Q}^2}$). Asymptotically the optimal position is at $\kappa = n + k - \beta \approx n - \frac{1}{2}\beta$. Interestingly, if we do not optimize k and only consider $k = O(1)$ we obtain the same asymptotic estimate as Kirchner and Fouque, which again emphasizes that we generalised their analysis.

For the fatigue point we compare the relative blocksize of $\frac{8\mathcal{S}}{\mathcal{Q}^2+1}$ to that of the 2016 Estimate given by $\frac{2\mathcal{Q}}{(\mathcal{Q}+1-\mathcal{S})^2}$ for $\mathcal{S} < 1$ and by $\frac{2}{\mathcal{Q}+2-2\mathcal{S}}$ for $\mathcal{S} \geq 1$. For ternary secrets ($\mathcal{S} = \frac{1}{2}$) this narrows down the fatigue point from $q \leq n^{2.783+o(1)}$ to $q = n^{2.484+o(1)}$ compared to the Kirchner–Fouque Estimate. This is still far above the (sub)linear parameters used for NTRU encryption schemes, and thus asymptotically we can close the pending question if these parameters fall in the weaker overstretched regime or not. In practice however we do observe fatigue points that are significantly lower than the naive value of $q = n^{2.484}$, which motivates a concrete analysis with concrete predictions.

[8] Available at: https://github.com/WvanWoerden/NTRUfatigue

7.4 A concrete average-case analysis

In this section we consider a concrete analysis of the new DSD-PT estimate, based on simple heuristics, to better predict the behaviour in practice, and to show that the analysis matches experiments and is thus likely to be tight. The first order asymptotics shown in Section 7.3.2 will remain unchanged, but the differences are significant for practical parameters. Again we split the analysis into several steps, but now derive heuristic expectations instead of loose upper bounds.

We assume that lattice vectors we encounter follow the Gaussian heuristic, and thus in particular that vectors are spherically distributed after normalisation. When projecting such vectors to a lower dimension they become shorter. The following Lemma shows how much shorter we expect them to become.

Lemma 112. *Let $\mathbf{x} \in \mathcal{S}^{d-1}$ follow a spherical distribution, and let $\pi_V : \mathbb{R}^d \to V$ be a projection to some subspace $V \subset \mathbb{R}^d$ of rank $k > 0$, then*

$$\mathbb{E}[\log(\|\pi_V(\mathbf{x})\|)] = \frac{1}{2}(\psi(k/2) - \psi(d/2)),$$

where ψ is the digamma function.

Proof. Let $X_0, \ldots, X_{d-1}$ be standard normal random variables, then the vector $\mathbf{x} = (x_0, \ldots, x_{d-1})$, with $x_j = X_j / \sqrt{\sum_{i=0}^{d-1} X_i^2}$, is spherically distributed. Without loss of generality we can assume that π_V projects onto the first k-coordinates. Then we conclude by Lemma 53 that

$$\begin{aligned}
\mathbb{E}[\log(\|\pi_V(\mathbf{x})\|)] &= \frac{1}{2}\mathbb{E}\left[\log\left(\frac{\sum_{i=0}^{k-1} X_i^2}{\sum_{i=0}^{d-1} X_i^2}\right)\right] \\
&= \frac{1}{2}\mathbb{E}\left[\log\left(\sum_{i=0}^{k-1} X_i^2\right)\right] - \frac{1}{2}\mathbb{E}\left[\log\left(\sum_{i=0}^{d-1} X_i^2\right)\right] \\
&= \frac{1}{2}(\psi(k/2) - \psi(d/2)).
\end{aligned}$$

□

7.4.1 Intersection

We start by giving a concrete average-case estimate for the intersection volumes. Assuming that projections behave as random we obtain the following concrete estimate.

Heuristic claim 113. *Let $\mathcal{L}$ be a $2n$-dimensional NTRU lattice with dense sublattice $\mathcal{L}^{\mathbf{GF}}$, before the DSD event is triggered we have for $k = 1, \ldots, n$ that* $\dim\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right) = k$, *and*

$$\mathbb{E}[\log \mathrm{vol}(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)})] = \log \mathrm{vol}(\mathcal{L}^{\mathbf{GF}}) - \left(\sum_{j=n+k}^{2n-1} \log\|\tilde{\mathbf{b}}_j\|\right) + \sum_{l=k+1}^{n} \psi\left(\frac{l}{2}\right) - \psi\left(\frac{n+l}{2}\right) + \frac{\zeta'(l)}{\zeta(l)},$$

where $\zeta(l) := \sum_{m=1}^{\infty} \frac{1}{m^l}$ is the Riemann zeta function and $\zeta'(l) := \sum_{m=1}^{\infty} \frac{\log(m)}{m^l}$ its derivative.

Justification. We follow the proof of Lemma 109. It is tight except for the length decrease from $\|\mathbf{d}\|$ to the projected and primitive vector $\|\pi(\mathbf{d})\|/m$. Note that when obtaining $\log \mathrm{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+l-1)}\right)$ from $\log \mathrm{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+l)}\right)$ for some $l = k+1, \ldots, n$, the dual vector $\mathbf{d}$ lives in a $(n+l)$-dimensional space and is projected to an l-dimensional space. Heuristically we assume that the normalisation of $\mathbf{d}$ is spherically distributed (or that π projects to a random l-dimensional subspace). By Lemma 112 the log-expected decrease in length from this projection then equals

$$\mathbb{E}\left[\log(\pi(\|\mathbf{d}\|)) - \log(\|\mathbf{d}\|)\right] = \psi\left(\frac{l}{2}\right) - \psi\left(\frac{n+l}{2}\right).$$

To conclude we also have to include the primitivity of $\pi(\mathbf{d})$ and thus the log-expectation of $m \geq 1$ such that $\pi(\mathbf{d})/m$ is primitive. For any basis $\mathbf{d}_0, \ldots, \mathbf{d}_{l-1}$ and $\pi(\mathbf{d}) = \sum_{i=0}^{l-1} x_i \mathbf{d}_i$, the scaling is given by $m = \gcd(x_0, \ldots, x_{l-1})$. Heuristically we assume that the absolute coefficients $|x_0|, \ldots, |x_{l-1}|$ are random integers in the interval $\{1, \ldots, B\}$

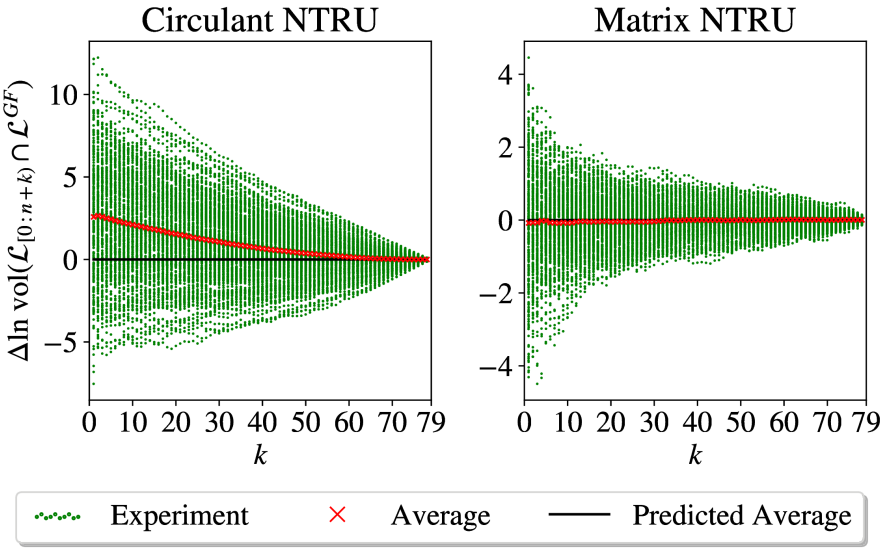


Figure 7.7: Experimental values of $\log \mathrm{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right)$ versus Claim 113 for circulant and matrix NTRU respectively. For each variant we used 256 LLL reduced NTRU lattices with parameters $q = 257, n = 79, \sigma^2 = \frac{2}{3}$ and computed the intersection for each k.

and we let $B \to \infty$. For $l \geq 2$ we have (see e.g., [DE+04])

$$\mathbb{P}_{\mathbf{x}\in\{1,\ldots,B\}^l}\left[\gcd(x_0,\ldots,x_{l-1}) = m\right] = \frac{1}{\zeta(l)} \cdot \frac{1}{m^l} + O(\log(B)/(Bm^{l-1})),$$

where the Riemann zeta function $\zeta(l) = \sum_{m=1}^{\infty} \frac{1}{m^l}$ is just the normalisation factor. From this we conclude that

$$\begin{aligned} &\lim_{B\to\infty} \mathbb{E}_{\mathbf{x}\in\{1,\ldots,B\}^l}[\log \gcd(x_0,\ldots,x_{l-1})] \\ &= \lim_{B\to\infty} \frac{1}{\zeta(l)} \sum_{m=1}^{B} \left[\frac{\log(m)}{m^l} + O\left(\frac{\log(m)\log(B)}{Bm^{l-1}}\right)\right] = -\frac{\zeta'(l)}{\zeta(l)} \end{aligned}$$

for $l \geq k+1 \geq 2$. △

Validation.

To validate Claim 113 we computed the actual intersection volumes $\mathrm{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right)$ for LLL reduced NTRU instances. We observed here, and also in further experiments, that the assumption on the dimension $\dim\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right) = k$ holds before we get close to triggering the DSD event. Figure 7.7 shows that the prediction perfectly matches the experiments for matrix NTRU. For circulant NTRU we see both that the expectation is slightly off and that the variance is much higher. The higher variance can be explained from the fact that the projections are very much dependent due to the circulant structure; in fact a closer inspection shows that for k close to n the differences with the prediction are highly correlated. We were not able to explain the error in the predicted expectation, but it seems to be caused by the circulant structure in combination with the Z-shape: the error decreased and eventually disappeared for large values of q and σ, for which the Z-shape disappeared (and before the DSD event was triggered). A maximal log-error of 2.5 is reached at $k = 1$. Note that a log-error of ϵ on $\mathrm{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right)$ translate into a factor of $e^{\epsilon/k}$ on the predicted length for the shortest vector. Except for very small k, this error appears benign.

7.4.2 Dense sublattice

In this section we give a concrete estimate for the expected volume of the dense NTRU sublatice $\mathcal{L}^{\mathbf{GF}}$. Directly from the construction we obtain a basis $[\mathbf{G}; \mathbf{F}]$ of $\mathcal{L}^{\mathbf{GF}}$, with $\mathbf{F}$ invertible. We consider two cases, that of regular NTRU, where $\mathbf{F}$ and $\mathbf{G}$ are circulant matrices, and that of matrix NTRU, where all entries are independently sampled. For both constructions the entries are sampled from independent discrete Gaussians over $\mathbb{Z}$, with some standard deviation $\sigma > 0$. As the only heuristic we assume that the individual entries in fact follow a *continuous* Gaussian instead of the discrete one.

Matrix NTRU.

We start with matrix NTRU, where we heuristically assume that all $2n \times n$ coefficients of the basis $[\mathbf{G}; \mathbf{F}]$ are sampled according to inde-

pendent continuous Gaussians with standard deviation σ. Under this heuristic we can derive an exact expression for the expected log-volume of the dense sublattice.

Lemma 114. *Let* $[\mathbf{G}; \mathbf{F}]$ *be a basis of the lattice* $\mathcal{L}^{\mathbf{GF}}$ *where all sampled entries are i.i.d. continous Gaussians with standard deviation* $\sigma > 0$*, then*

$$\mathbb{E}[\log(\mathrm{vol}(\mathcal{L}^{\mathbf{GF}}))] = \frac{1}{2}n\left(\log(2\sigma^2) - \psi(n)\right) + \sum_{i=0}^{n-1}\psi\left(\frac{2n-i}{2}\right).$$

Proof. By Lemma 53 the log-expectation of the norm of each basis element equals $(\ln(2\sigma^2) + \psi(n))/2$. Note that the i-th Gram-Schmidt vector $\tilde{\mathbf{b}}_i$ is obtained after projecting the i-th basis vector orthogonally away from an i-dimensional subspace, and thus onto a $2n-i$ dimensional subspace. However after normalisation the basis vectors follow a spherical distribution and thus by Lemma 112 we have

$$\begin{aligned}\mathbb{E}[\log\|\tilde{\mathbf{b}}_i\|] &= (\ln(2\sigma^2) + \psi(n))/2 + \psi\left(\frac{2n-i}{2}\right) - \psi(n)\\ &= (\ln(2\sigma^2) - \psi(n))/2 + \psi\left(\frac{2n-i}{2}\right).\end{aligned}$$

We conclude by noting that $\mathbb{E}[\log(\mathrm{vol}(\mathcal{L}^{\mathbf{GF}}))] = \sum_{i=0}^{n-1}\mathbb{E}[\log\|\tilde{\mathbf{b}}_i\|]$. □

Circulant NTRU.

For circulant NTRU both $\mathbf{G}$ and $\mathbf{F}$ in the basis $[\mathbf{G}; \mathbf{F}]$ are circulant matrices. Again we replace discrete with continuous Gaussians. The eigenvalues and eigenvectors of a circulant matrix are well known and we use this to obtain an exact expression for the expected volume of the dense sublattice.

Lemma 115. *Let* $[\mathbf{G}; \mathbf{F}]$ *be a basis of the lattice* $\mathcal{L}^{\mathbf{GF}}$ *where* $\mathbf{G}, \mathbf{F}$ *are circulant and all sampled entries are i.i.d. continous Gaussians with with standard deviation* $\sigma > 0$*, then*

$$\mathbb{E}[\log(\mathrm{vol}(\mathcal{L}^{\mathbf{GF}}))] = \frac{1}{2}n\left(\log(2n\sigma^2) + \psi(1)\right) + \frac{1}{2}(n-1)(1-\log(2)).$$

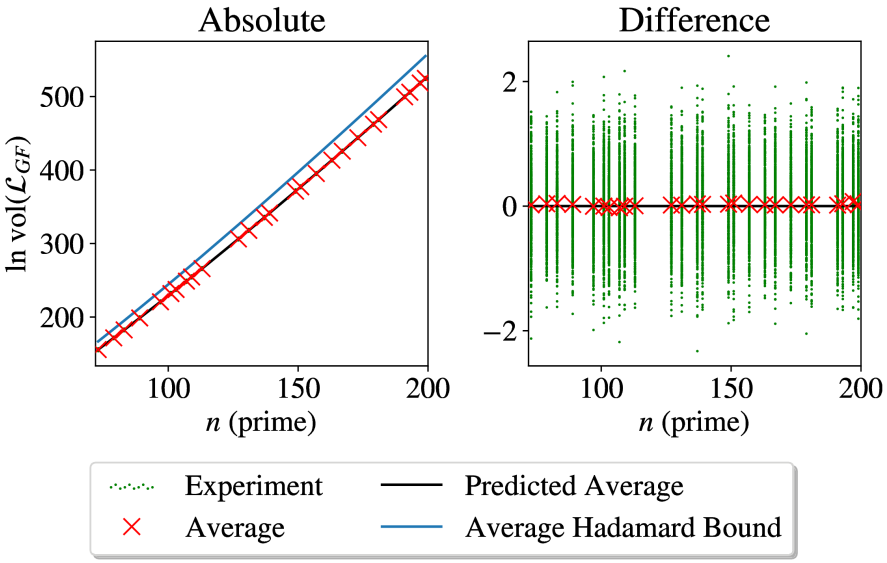


Figure 7.8: Experimental values of $\log(\mathrm{vol}(\mathcal{L}^{\mathbf{GF}}))$ versus Lemma 114 for matrix NTRU with discrete Gaussians and variance $\sigma^2 = \frac{2}{3}$. For each parameter n we generated 512 instances.

Proof. For $n \times n$ circulant matrices $\mathbf{G}, \mathbf{F}$ the eigenvectors are identical and given by $\mathbf{v}_j := (1, \omega^j, \omega^{2j}, \omega^{(n-1)j})$ for $j = 0, \ldots, n-1$, where $\omega := e^{2\pi i/n} \in \mathbb{C}$ is a primitive n-th root of unity. Suppose that the circulant matrix $\mathbf{G}$ is generated by the vector $\mathbf{c} = (c_0, \ldots, c_{n-1})$, then the corresponding eigenvalues are given by the DFT coefficients of $\mathbf{c}$, namely $\lambda_j := c_0 + c_{n-1}\omega^j + \ldots + c_1\omega^{(n-1)j}$. We have that $\lambda_0 = \sum_{j=0}^{n-1} c_j$, and thus λ_0 follows a Gaussian distribution with variance $n\sigma^2$, and in particular $\lambda_0^2 \sim \chi^2_{1,n\sigma^2}$. Additionally for $j = 1, \ldots, n-1$ we can write $\lambda_j = X + i \cdot Y \in \mathbb{C}$ where $X, Y \in \mathbb{R}$ are both linear combinations of the c_i's and thus (X, Y) follows a jointly Gaussion distribution. A simple computation shows that X and Y both have variance $n\sigma^2/2$ and that they are uncorrelated, which for Gaussians implies that they are independent [PP02, p. 212]. So $|\lambda_j|^2 = X^2 + Y^2 \sim \chi^2_{2,n\sigma^2/2}$. Note that all circulant matrices have the same eigenvectors and thus the squared singular values of the concatenation of two circulant matrices are the sum of the squared absolute eigenvalues. So $[\mathbf{G}; \mathbf{F}]$ has one squared singular value s_0^2 distributed as $\chi^2_{1,n\sigma^2} + \chi^2_{1,n\sigma^2} = \chi^2_{2,n\sigma^2}$, and $n-1$

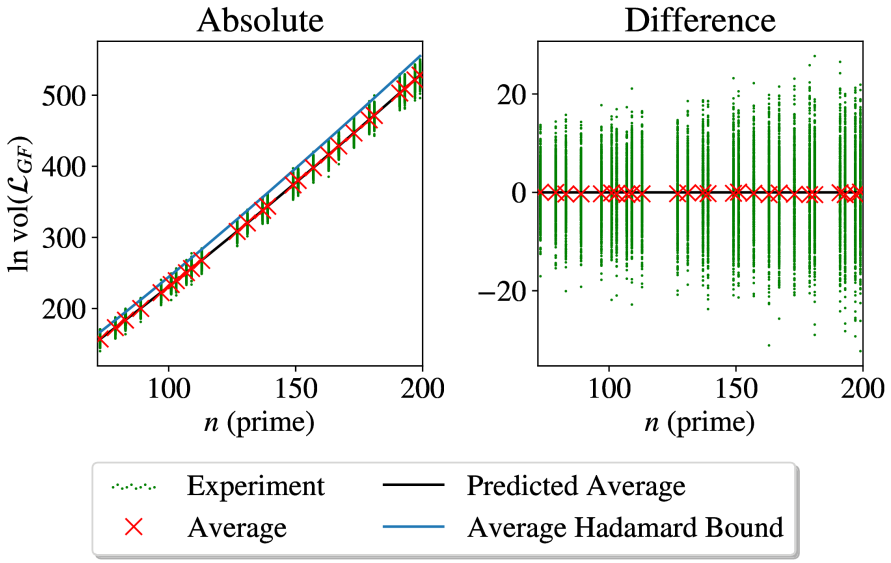


Figure 7.9: Experimental values of $\log(\mathrm{vol}(\mathcal{L}^{\mathbf{GF}}))$ versus Lemma 115 for circulant NTRU with discrete Gaussians and variance $\sigma^2 = \frac{2}{3}$. For each parameter n we generated 512 instances.

squared singular values $s_1^2, \ldots, s_{n-1}^2$ distributed as $\chi^2_{2,n\sigma^2/2} + \chi^2_{2,n\sigma^2/2} = \chi^2_{4,n\sigma^2/2}$. By Lemma 53 they have a log-expectation of

$$\mathbb{E}[\log s_0^2] = \log(2n\sigma^2) + \psi(1), \text{ and } \mathbb{E}[\log s_j^2] = \log(n\sigma^2) + \psi(2)$$

for $j = 1, \ldots, n-1$. We conclude by noting that

$$\mathbb{E}[\log(\mathrm{vol}(\mathcal{L}^{\mathbf{GF}}))] = \frac{1}{2}\sum_{i=0}^{n-1} \log(s_i^2).$$

□

Validation.

To validate the concrete estimate for $\mathrm{vol}(\mathcal{L}^{\mathbf{GF}})$ we generated the NTRU sublattice for several dimensions and computed its volume. We sample the secret coefficients following a discrete Gaussian with variance $\sigma^2 = \frac{2}{3}$ and ran experiments for both matrix NTRU and circulant

NTRU. In Figures 7.8 and 7.9 we see that the predictions from Lemmas 114 and 115 perfectly fit the observed volumes in all dimensions. We do note that the variance is quite significant for the circulant case, but it can be fully explained by the computed eigenvalue distributions in the proof of Lemma 115. We will later see that this high variance has a large influence on the successful BKZ blocksize (Section 7.5.1, Figure 7.11).

7.4.3 Further refinements

We discuss some further refinements, some of which were already successfully applied to the 2016 Estimate [AGVW17; DDGR20; PV21].

Gaussian Heuristic. For the asymptotic analysis we used Minkowski's bound to estimate the length $\lambda_1\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right)$ in terms of the volume $\mathrm{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right)$. A natural way to obtain a concrete estimate for the expected minimal length is by assuming that the intersection $\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}$ follows the Gaussian Heuristic and thus for the prediction we assume that

$$\lambda_1\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right) = \mathrm{gh}(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}) \approx \sqrt{k/(2\pi e)} \cdot \mathrm{vol}\left(\mathcal{L}^{\mathbf{GF}}_{\cap[0:n+k)}\right)^{1/k}.$$

We should however be careful with this assumption, as in fact it is false for $k = n$. E.g., the above predicts that $\lambda_1(\mathcal{L}^{\mathbf{GF}}) \approx \sqrt{n/(2\pi e)} \cdot \sqrt{2n\sigma^2}$, while we know that $\lambda_1(\mathcal{L}^{\mathbf{GF}}) = \|(\mathbf{g};\mathbf{f})\| \approx \sqrt{2n\sigma^2}$, a factor $\Theta(\sqrt{n})$ shorter than predicted. The reason for this is that the dense sublattice is up to rotation and scaling very similar to the orthogonal lattice $\mathbb{Z}^n$, precisely the lattice for which it is well known that the Gaussian Heuristic is false. For small $k \ll n$ we do observe that the intersected lattice $\mathcal{L}^{\mathbf{GF}}_{\cap[0:s)}$ follows the Gaussian Heuristic; the orthogonal structure seems to be broken by the intersection. However we do not have a clear idea how large k can become before the orthogonal structure returns and the minimal length stops following the prediction from the Gaussian Heuristic. We think this behaviour deserves some further investigation, e.g., if the transition is very sudden or not, and we leave it as an open problem. This near-orthogonality of $\mathcal{L}^{\mathbf{GF}}_{\cap[0:s)}$ may be critical to model the DSD-LL events.

Probabilities. So far we have only considered expectations of volumes and projections. While this is enough to give a rough concrete estimate we want to be more precise. Success probabilities can accumulate up over multiple BKZ blocks and (progressive) tours, possibly leading to success at much lower blocksizes than the rough estimate. We continue using the expected values for the volume of the dense sublattice and the intersection volumes to obtain the expected length $\lambda_1\big(\mathcal{L}^{\mathbf{GF}}_{\cap[0:s)}\big)$ of the dense sublattice vector via the Gaussian Heuristic. However we then model the short dense sublattice vector $\mathbf{v} \in \mathcal{L}^{\mathbf{GF}}_{\cap[0:s)}$ as an s-dimensional Gaussian vector with the same expected length; allowing us to compute the exact probability that $\big\|\pi_{s-\beta}(\mathbf{v})\big\| \leq \big\|\tilde{\mathbf{b}}_{s-\beta}\big\|$ using the CDF of the chi-square distribution with β degrees of freedom.

Up to now we have ignored the probability that after $\pi_{s-\beta}(\mathbf{v})$ is inserted, it is also correctly lifted to the full vector $\mathbf{v}$ by later BKZ tours. While this almost always happens for higher blocksizes, it is not so likely for lower blocksizes, and ignoring this leads to overly optimistic predictions. For BKZ-β to successfully lift or *eventually* pull the vector $\mathbf{v}$ to the front it should also satisfy $\|\pi_i(\mathbf{v})\| \leq \|\tilde{\mathbf{b}}_i\|$ for all $i = s-2\beta+1, s-3\beta+2, \ldots$. These conditions are not independent which makes them hard to compute exactly. We simplify the computation by only considering the dependence for consecutive positions $i, i-\beta+1$ as done in [DDGR20]. We iteratively run the estimator for progressive $\beta = 2, 3, \ldots$ and take account of all probabilities assuming that all tours behave completely independently. The new concrete estimate will be the expected successful blocksize. Additionally this allows us to combine both the (probabilistic) SKR 2016 Estimate and the new DSD-PT estimate in a single estimator. With some more administration we can also predict the distribution of the successful location κ, and predict the probability that the SKR event happens before the DSD event.

BKZ shape for low blocksizes. While the formulas for the (Z)GSA slope α_β and the expected first minimum $\mathrm{gh}(\beta)$ convert to the experimental values for large blocksizes of say $\beta \geq 50$, they are not as accurate for small β. As expected the convergence is worse for progressive BKZ when we only use a few tours of each blocksize. We ran some experiment on random low dimensional q-ary lattices to obtain practical estimates for $\mathrm{gh}(\beta)$ with $\beta \leq 50$. Earlier works about the

2016 Estimate resorted to BKZ simulators to predict the BKZ shape, which account for the number of tours and also the special shape of the head and tail that do not perfectly follow the GSA shape. Together with the earlier mentioned refinements this resulted in very precise predictions [DDGR20; PV21]. However how BKZ acts on a Z-shaped basis is much less understood [AD21] and as of yet there are no accurate BKZ simulators. Understanding the behaviour and creating an accurate simulator would be very interesting, but is out of the scope of this work. We continue using the ZGSA, but we resort to experimental values for α_β obtained by running BKZ on random q-ary lattices for large q. To remain consistent we also do not use a simulator for the GSA shape, and accept the small discrepancy between the predictions and practical experiments.

7.5 Experimental validation

7.5.1 Successful blocksize

We start with comparing the concrete predictions to the preliminary experiment from Section 7.3.1. We ran progressive BKZ with 8 tours on matrix NTRU instances with parameters $n = 127, \sigma^2 = \frac{2}{3}$ for several moduli q. In Figure 7.10 we show the blocksizes at which the SKR or DSD event is first detected, and compare them to the concrete estimator. We ran the estimator three times for each modulus q: only accounting for SKR, only accounting for DSD-PT, and accounting for both. Note that the combined estimate can be strictly lower than both the first two because the probabilities to succeed accumulate over both events. We calibrated the values of α_β by running the same BKZ routine on $(2 \cdot 127)$-dimensional q-ary lattices with $q \approx 2^{20}$.

We observe that the experiments match the estimates reasonably well, with an average blocksize error of less than 2 for the DSD events and less than 3 for the SKR events. We shortly discuss potential sources of the small errors error.

- We do not actually run the classical BKZ algorithm, but the BKZ 2.0 algorithm as it is more feasible to run for large blocksizes. One part of the latter algorithm is that in each BKZ block $[\kappa : \kappa + \beta)$ the last $\beta - 1$ vectors are randomised before finding a

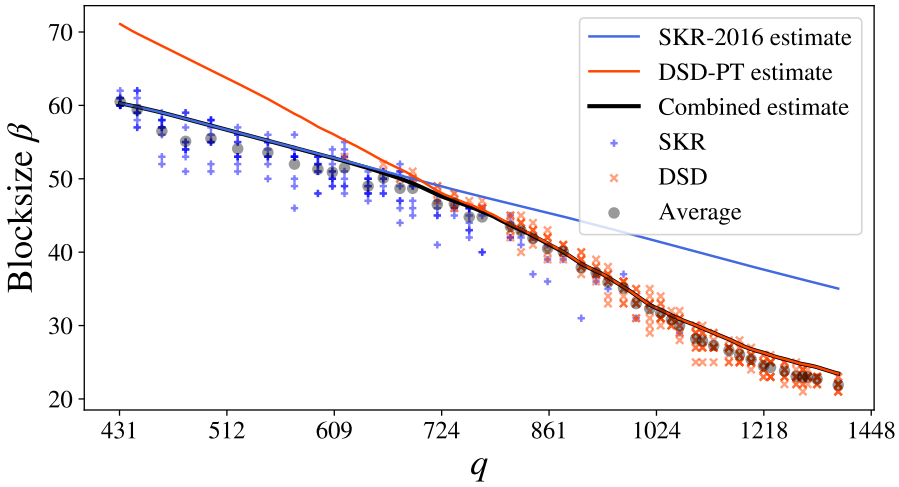


Figure 7.10: Experiment versus prediction for progressive BKZ with 8 tours on matrix NTRU instances with parameters $n = 127, \sigma^2 = \frac{2}{3}$ for several moduli q. We did 10 runs per modulus q.

short projected vector. This temporarily breaks the GSA shape and results a small 'bump' in the profile that is pushed to the right during a tour. On average we measured at the SKR events a log-increase of 0.048 on the value of $\|\tilde{\mathbf{b}}_\kappa\|$ compared to the GSA (while the rest of the basis matches very closely). Although anecdotal, adjusting the estimator with this offset of 0.048 resulted in very close predictions for the SKR events.

- For small blocksizes $\beta \leq 30$ we see that the DSD-PT estimate is slightly pessimistic compared to the experiments. However the successful profile slope α_β (computed from the profile at the moment of detection) does closely match the predicted slope $\alpha_{\beta_{pred}}$, pointing to a wrong calibration of the slope parameter for very low blocksizes. Note that the non-flat part of the Z-shape in the experiments has size less than the $2 \cdot 127$ dimensional lattice used for calibration, which plausibly explain why the slope converges quicker than expected.

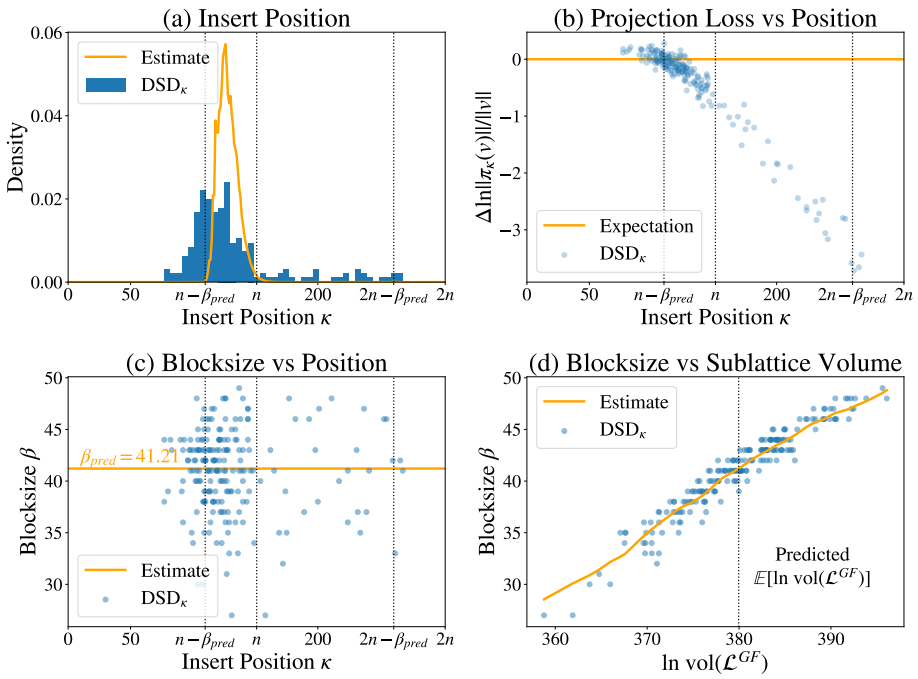


Figure 7.11: Results from running progressive BKZ with 8 tours on 200 circulant NTRU instances in the overstretched regime with parameters $n = 151, q = 2003$ and $\sigma^2 = \frac{2}{3}$. (a) Distribution of κ at which the DSD$_\kappa$ event is triggered. (b) The log-length decrease from $\|\mathbf{v}\|$ to $\|\pi_\kappa(\mathbf{v})\|$ normalised by the expected decrease (Lemma 112). (c) Successful blocksize β versus the position κ, and (d) versus the volume $\mathrm{vol}(\mathcal{L}^{\mathbf{GF}})$.

7.5.2 Detailed behaviour

Because the estimator computes actual probabilities we can compare the predictions to the experiments on a deeper level. We ran 200 experiments on circulant NTRU instances with parameters $n = 151, q = 2003$ and $\sigma^2 = \frac{2}{3}$. These parameters fall into the overstretched regime. The results, compared to the estimator, are shown in Figure 7.11. The average successful blocksize over all 200 instances is $\beta_{\mathrm{avg}} = 40.99$, which is close to the estimated value of $\beta_{\mathrm{pred}} = 41.21$. We note however that the average squared error is as large as 16.9; the successful blocksizes range from 27 to 49 for identical initial parameters. Looking at Figure 7.11(d) we see that this can mostly be

explained by the large variance of the volume of the dense sublattice $\mathrm{vol}(\mathcal{L}^{\mathbf{GF}})$, as earlier noticed in Figure 7.9. Adjusting the estimate with the concrete volume (instead of the average case prediction) decreases the average squared error all the way down to 0.83. E.g., the large average squared error is not an imprecision of the estimator, but an inherent high variance in the hardness of circulant NTRU instances. In many schemes this variance is significantly reduced by sampling the secret keys $\mathbf{f}, \mathbf{g}$ under some fixed constraints on the lengths $\|\mathbf{f}\|, \|\mathbf{g}\|$ or even stronger restrictions.

We now take a look at the positions κ at which the DSD_κ events took place. In Figure 7.11(a) we see that most events (169/200) happen for $\kappa \leq n$, as predicted by the DSD-PT estimate. We see in Figure 7.11(b) that for these instances the projection $\pi_\kappa(\mathbf{v})$ of the dense sublattice vector $\mathbf{v}$ is also not much shorter than expected, which matches the DSD-PT event as defined in Section 7.3.1. For the remaining events (31/200) with $\kappa > n$ we note that κ seems to be evenly distributed over $n+1, \ldots, 2n-\beta$, while the projected length becomes increasingly smaller than expected for increasing κ.

These DSD-LL events do not seem properly predicted by the model. Indeed, looking at κ around $2n-\beta$ the model predicts that the probability that a random vector has such a short projection is as small as 10^{-55}, and the probability that such a vector would correctly be lifted by Babai's nearest plane algorithm is even smaller (see script `lucky_lift.sage`). Interestingly however is that the transition between DSD-PT and DSD-LL events is rather continuous, and it is not so clear where to put the threshold between these two events.

In Figure 7.11(c) we see no clear difference between the successful blocksizes at positions $\kappa \leq n$ versus those at $\kappa > n$, and their average of 41.1 and 40.4 respectively. This suggests that, at least for these parameters, the DSD-LL events do not significantly contribute to making the attack cheaper. This was also the case for all other experiments we ran with different parameters.

In Figure 7.11(a) we also show the predicted distribution of the event positions κ. We correctly predicted the peak around $\kappa \approx n - \frac{1}{2}\beta_{\mathrm{pred}}$, and that $\kappa \leq n$ for the DSD-PT events. However the prediction is much more concentrated than the experimentally observed events, and neglects to notice the events at $\kappa \leq n - \frac{1}{2}\beta_{\mathrm{pred}}$. We give two possible explanations for this.

- The estimate assumes an average-case dense sublattice volume, ignoring the high variance of the dense sublattice volume. Adjusting for this using the real volumes makes the prediction less concentrated and closer to the observed events. However this still not captures all events at $\kappa \leq n - \frac{1}{2}\beta_{\text{pred}}$.
- The experiments and the estimator inherently measure two different things. The experiment detects a DSD_κ event if the projection $\pi_\kappa(\mathbf{v})$ is inserted and is lifted to a dense sublattice vector $\mathbf{v}$ by Babai's nearest plane algorithm. The estimator however estimates the probability that the projection is short enough and that the projection is *eventually* lifted to the dense sublattice vector $\mathbf{v}$ by later tours of the BKZ algorithm. Therefore a DSD-PT event predicted at position κ might experimentally only be detected one tour later at $\kappa - \beta + 1$ or at an even lower position. This could potentially be resolved by also taking into account the direct lift of Babai's nearest plane in the estimator, but note that for the eventual goal of estimating the successful blocksize this will not matter.

7.5.3 Fatigue point

The concrete estimator follows the experiments reasonably well and thus we can use it to estimate the concrete fatigue point for dimensions that are not feasible in practice. To verify the estimate of the fatigue point we also did some experiments in dimensions that are still feasible. For this we ran a *soft* binary search, only decreasing the interval length by 3/4 so as not view a probabilistic result as a definitive answer. More specifically, starting with a range of $[q_{\min}, q_{\max}]$ we ran an experiment for a prime $q \approx (q_{\min} + q_{\max})/2$. If it succeeds with an SKR event we update $q_{\min}$ to $(q_{\min} + q)/2 + 1$, if it succeeds with a DSD event we update $q_{\max}$ to $(q_{\max} + q)/2 - 1$. We repeat this until the interval does not contain any prime and we return $(q_{\min} + q_{\max})/2$ as a rough estimate of the fatigue point. We averaged this over 20 experiments for each parameter n. We chose for matrix NTRU because of the lower variance in the hardness of these instances.

We compared this to the prediction. Because the estimator accounts for probabilities of events, we can predict for which value of q about 50% of the instances succeeds with a DSD event. Because it

would be unreasonable to calibrate the low blocksize slope values α_β for each dimension we reused those of the $2 \cdot 127$ dimensional q-ary lattice from an earlier experiment. This might make the estimates a bit less precise for $n \ll 127$, and $n \gg 127$ if the successful blocksize is small around the fatigue point.

The results are shown in Figure 7.12 and plotted against $Cn^{2.484}$ for several constants C. Remarkably the experiments and concrete predictions closely follow the asymptotics already for reasonably small values of n. A loglog-linear regression of the 50% DSD-PT estimate over all primes $199, \ldots, 499$ gives $0.0034 \cdot n^{2.506}$. Restricting the exponent to 2.484 gives $0.0038 \cdot n^{2.484}$ with a log-standard deviation of only 0.006.

The experimental average appears slightly higher than the estimator prediction for 50% DSD - 50% SKR. The main reason for this seems to be that the estimator is slightly pessimistic for detecting the SKR event, as already observed and explained in Section 7.5.1. Another small detail is that the binary search is slightly biased to higher values of q because at each iteration we pick the *next* prime after

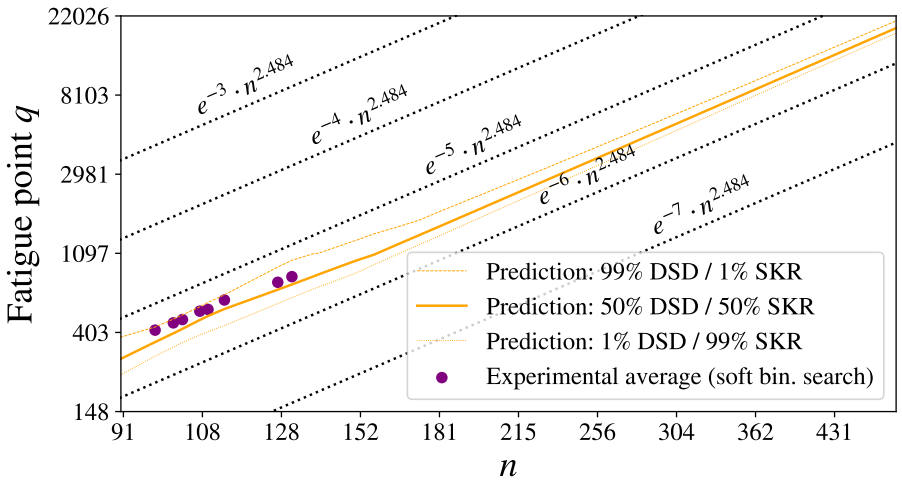


Figure 7.12: Concrete fatigue point versus asymptotics using progressive BKZ with 8 tours on matrix NTRU instances with variance $\sigma^2 = \frac{2}{3}$. The 0.5 percentile line shows for which q we estimate that the DSD event is triggered before the SKR event for about 50% of the instances.

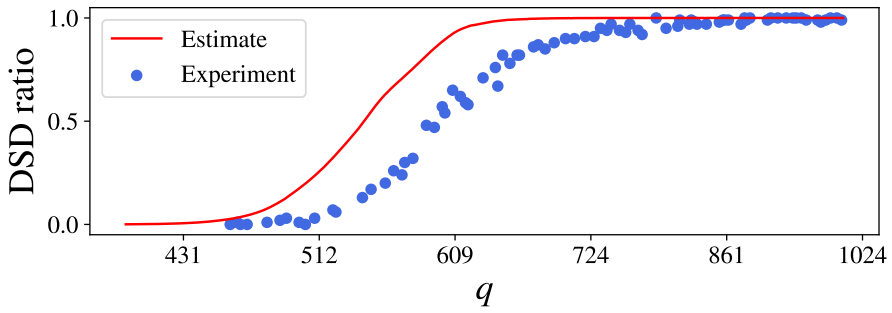


Figure 7.13: Experiment versus prediction for progressive BKZ with 8 tours on matrix NTRU instances with parameters $n = 113, \sigma^2 = \frac{2}{3}$ for several moduli q. We did 100 runs per modulus q and the plot shows the ratio of these runs succeeding with a DSD event (before an SKR event).

$(q_{\min} + q_{\max})/2$.

7.5.4 Zoom on the fatigue point: a smooth probabilistic transition

We take a closer look at the transition from the non-overstretched to the overstretched regime. For this we ran several experiments on matrix NTRU instances with parameters $n = 113, \sigma^2 = \frac{2}{3}$ for several moduli q, with 100 runs each. We compare the DSD success ratio with the probabilistic concrete estimate. The results are shown in Figure 7.13. Just as in Figure 7.12 we see a shift between the experiment and prediction, which can again be explained by the SKR estimator being too pessimistic. Note however that while the discrepancy looks significant in this zoomed plot, it only emphasises a small error of about 2 block sizes between the experiments and the predictions. Ignoring this shift the shape of the predicted transition matches the experiments very well.

CHAPTER 8

Basis Reduction for Binary Codes

This chapter is an abbreviated version of the joint work 'An Algorithmic Reduction Theory for Binary Codes: LLL and more', with Thomas Debris-Alazard and Léo Ducas, published in IEEE Transactions on Information Theory.

8.1 Introduction

Codes and lattices share many mathematical similarities. A (linear) code $\mathcal{C} \subset \mathbb{F}_q^n$ is defined as a subspace of a vector space over a finite field, typically endowed with the Hamming metric, while a lattice $\mathcal{L} \subset \mathbb{R}^n$ is a discrete subgroup of a Euclidean vector space. They both found similar applications in information theory and computer sciences. For example, both can be used to perform error correction; on digital channels for codes, and on analogue channels for lattices.

Both objects also found applications in cryptography. Cryptosystems can be built relying either on the hardness of finding a close codeword or a close lattice point from a given target. And just as

for lattices, cryptography based on codes appears to be resistant to quantum computing.

The set of techniques for attacking those problems also have similarities, and some algorithms have been transferred in each direction: for example the Blum-Kalai-Wasserman [BKW03] algorithm has been adapted from codes to lattices [Alb+15], while the introduction of locally-sensitive hashing in code cryptanalysis [MO15] shortly followed its introduction in lattice cryptanalysis [Laa15].

It is therefore very natural to question whether all techniques used for codes have also been considered for lattices, and reciprocally. Beyond scientific curiosity, this approach can hint us at how complete each state of the art is, and therefore, how much trust we should put into cryptography based on codes and cryptography based on lattices.

Comparing both states of the art, it appears that there is a major lattice algorithmic technique that has no clear counterpart for codes, namely, *basis reduction*. Recall from Chapter 6 that lattice reduction attempts to find a basis with good geometric properties; in particular its vectors should be rather short and orthogonal to each others.

More specifically, the basis defines, via Gram-Schmidt Orthogonalization, a fundamental domain (or tiling) of the space, and a corresponding decoding algorithm. Decoding with this algorithm is the most favourable when these tiles are close to being square, *i.e.* when the Gram-Schmidt lengths are *balanced*. Basis reduction algorithms such as LLL aim at making the Gram-Schmidt lengths more balanced.

Certainly, the problem of finding short codewords has also been intensively studied in cryptanalysis with the Information Set Decoding (ISD) literature [Pra62; LB88; Ste88; Dum91; MMT11; BJMM12; MO15; BM18], but notions of basis reduction for lattices are more subtle than containing short vectors; as discussed above, a more relevant objective is to balance the Gram-Schmidt norms. There seem to be no analogue notions of Gram-Schmidt norms and basis reduction for codes, or at least they are not explicit nor associated with reduction algorithms. We are also unaware of any study of how such reduced bases would help with decoding tasks.

This observation leads to two questions. Is there an algorithmic reduction theory for codes, analogue to the one of lattices? And if so, can it be useful for decoding tasks?

8.1.1 Contributions

In this chapter we answer both questions positively, and set the foundation of an algorithmic reduction theory for codes. More specifically, we propose as main contributions:

1. the notion of an epipodal matrix $\mathbf{B}^+$ of the basis $\mathbf{B}$ of a binary code $\mathcal{C} \subset \mathbb{F}_2^n$ (depicted in Figure 8.1), playing a role analogue to the Gram-Schmidt Orthogonalisation $\tilde{\mathbf{B}}$ of a lattice basis,

2. a fundamental domain (or tiling) of $\mathcal{C}$ over $\mathbb{F}_2^n$ associated to this epipodal matrix, as an analogue to the rectangle parallelepipedic tiling for lattices,

3. a polynomial time decoding algorithm (SizeRed) effectively reducing points to this fundamental region, analogue to the Nearest Plane algorithm popularised by Babai [Bab86],

4. a relation between the geometric quality of the fundamental domain and the success probability for decoding a random error to the balance of the lengths of the epipodal vectors,

5. an adaptation of the seminal LLL reduction algorithm [LLL82] from lattices to codes, providing in polynomial time a basis with some epipodal length balance guarantees. Interestingly, this LLL algorithm for codes appears to be an algorithmic realisation of the classic bound of Griesmer [Gri60], in the same way that LLL for lattices realizes Hermite's bound.

These contributions establish an initial dictionary between reduction for codes and for lattices, summarised in Table 8.1.

Open Artefacts: Source code (`c++` kernel, with a `python` interface)[1].

8.1.2 Organisation

We introduce some preliminaries on binary linear codes in Section 8.2. In Section 8.3 we introduce the notion of orthopodality for binary vectors, and use this to define (orthopodal) projections and a

[1] Available at `https://github.com/lducas/CodeRed/`.

Table 8.1: A Lattice-Code Dictionary.

	Lattice $\mathcal{L} \subset \mathbb{R}^n$	Code $\mathcal{C} \subset \mathbb{F}_2^n$
Ambient Space	$\mathbb{R}^n$	$\mathbb{F}_2^n$
Metric	Euclidean $\|\mathbf{x}\|^2 = \sum x_i^2$	Hamming $\|\mathbf{x}\| = \#\{i \mid x_i \neq 0\}$
Support (element) Support (lattice/code)	$\mathbb{R} \cdot \mathbf{x}$ $\mathrm{Span}_{\mathbb{R}}(\mathcal{L})$	$\{i \mid x_i \neq 0\}$ $\{i \mid \exists \mathbf{c} \in \mathcal{C} \text{ s.t. } c_i \neq 0\}$
Sparsity Effect. Sparsity	$\det(\mathcal{L})$ $\det(\mathcal{L})$	2^{n-k} $2^{\#\mathrm{S}(\mathcal{C})-k}$
$\mathbf{x} \perp \mathbf{y}$ Projection $\pi_{\mathbf{x}}$	Orthogonality $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ Ortho. Project. onto $\mathbf{x}$ $\mathbf{y} \mapsto \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\langle \mathbf{x}, \mathbf{x} \rangle} \cdot \mathbf{x}$	Orthopodality $\mathrm{S}(\mathbf{x}) \cap \mathrm{S}(\mathbf{y}) = \emptyset$ Punct. pattern $\mathbf{x}$ $\mathbf{y} \mapsto \mathbf{y} \wedge \mathbf{x}$
Auxiliary matrix Basis profile ℓ	Gram-Schmidt Orth. $\tilde{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j<i} \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle} \cdot \tilde{\mathbf{b}}_j$ $\ell_i = \|\tilde{\mathbf{b}}_i\|$	Epipodal matrix $\mathbf{b}_i^+ = \mathbf{b}_i \wedge \overline{(\mathbf{b}_1^+ \vee \cdots \vee \mathbf{b}_{i-1}^+)}$ $\ell_i = \|\mathbf{b}_i^+\|$
Fundamental domain $\mathcal{F}(\mathbf{B})$ Error correct. radius Average deco. dist. Worst deco. dist. Favourable decoding	Parallelepiped $\mathcal{P}(\tilde{\mathbf{B}})$ $\left\{\mathbf{x} \,\middle|\, \forall i \ \lvert\langle \mathbf{x}, \tilde{\mathbf{b}}_i \rangle\rvert \leq \frac{\ell_i^2}{2}\right\}$ $\min_i \ell_i^2/2$ $\sqrt{\frac{1}{12}\sum_i \ell_i^2}$ $\sqrt{\frac{1}{2}\sum_i \ell_i^2}$ balanced ℓ_i's	Prod. of Hamming balls[1] $\left\{\mathbf{x} \,\middle|\, \forall i \ \lvert\mathbf{x} \wedge \mathbf{b}_i^+\rvert \leq \frac{\ell_i}{2}\right\}$ $\min_i \lfloor (\ell_i - 1)/2 \rfloor$ $\approx \frac{n}{2} - \frac{1}{\sqrt{\pi}} \sum_i \sqrt{\lceil \ell_i/2 \rceil}$ $\sum_i \lfloor \ell_i/2 \rfloor$ balanced and odd ℓ_i's
Basis inequality Invariant	$\prod \|\mathbf{b}_i\| \geq \det(\mathcal{L})$ $\prod \|\tilde{\mathbf{b}}_i\| = \det(\mathcal{L})$	$\sum \lvert\mathbf{b}_i\rvert \geq \#\mathrm{S}(\mathcal{C})$ $\sum \lvert\mathbf{b}_i^+\rvert = \#\mathrm{S}(\mathcal{C})$
LLL balance LLL first length Corresponding bound	$\ell_i \leq \sqrt{4/3} \cdot \ell_{i+1}$ $\ell_1 \leq (\frac{4}{3})^{\frac{n-1}{2}} \det(\mathcal{L})^{\frac{1}{n}}$ Hermite's	$1 \leq \ell_i \leq 2 \cdot \ell_{i+1}$ $\ell_1 - \frac{\lceil \log_2 \ell_1 \rceil}{2} \leq \frac{n-k}{2} + 1$ Griesmer's [Gri60]

[1]This is not exactly correct when some epipodal length $\lvert\mathbf{b}_i^+\rvert$ are even. See tie-breaking in Section 8.4.

code analogue to the Gram-Schmidt basis. In Section 8.4 we give a translation of Babai's Nearest Plane algorithm to binary linear codes, and explain how its performance depends on the basis profile. In Section 8.5 we give a translation of the LLL algorithm to binary linear codes, and show how it results in a reasonably balanced basis profile. In the last Section 8.6 we discuss some future research directions.

8.2 Binary linear codes

We give a short introduction on the Hamming metric, binary linear codes, and how they relate to lattices.

Binary vector space

While lattices live in the continuous Euclidean vector space $\mathbb{R}^n$, codes live in the discrete non-euclidean vector space $\mathbb{F}_q^n$ for some prime q. In this work we will only consider the binary case $q = 2$, as it allows us to explain our general ideas, without too many technical difficulties. We identify $\mathbb{F}_2$ with the binary set $\{0, 1\}$, and thus interpret elements of $\mathbb{F}_2^n$ as binary vectors. We will use the standard boolean notations $\overline{\mathbf{x}}$, $\mathbf{x} \oplus \mathbf{y}$, $\mathbf{x} \wedge \mathbf{y}$, $\mathbf{x} \vee \mathbf{y}$, for respectively the bitwise NOT, the bitwise XOR (vector addition over $\mathbb{F}_2^n$), the bitwise AND, and the bitwise OR.[2] In contrast to most code-based literature the vectors $\mathbf{x} \in \mathbb{F}_2^n$ should be interpreted as column vectors, following the overall notation in this thesis.

Hamming metric

The most common metric in the code-based literature is the Hamming metric. The support $\mathrm{S}(\mathbf{x})$ of a vector $\mathbf{x} \in \mathbb{F}_2^n$ is the set of indices of its nonzero coordinates, and its Hamming weight $|\mathbf{x}| \in [\![0, n]\!]$ is the cardinality of its support:

$$\mathrm{S}(\mathbf{x}) := \{i \in [\![1, n]\!] \mid x_i \neq 0\}, \quad |\mathbf{x}| := \#\mathrm{S}(\mathbf{x}).$$

[2] In the code-based cryptography literature, for instance [COT16], the bitwise AND is often interpreted algebraically as a *star-product* or *Schur-product*, denoted $\odot$ or $\star$. We found the boolean notations more adapted to our geometric purposes, especially given that the bitwise OR also plays an important role.

The Hamming distance between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ is likewise given by $|\mathbf{x} \oplus \mathbf{y}| \in [\![0, n]\!]$. We will denote by $\mathcal{S}_w^n$ the Hamming sphere and by $\mathcal{B}_w^n$ the Hamming ball of radius w over $\mathbb{F}_2^n$, namely:

$$\mathcal{S}_w^n := \{\mathbf{x} \in \mathbb{F}_2^n : |\mathbf{x}| = w\}, \ \mathcal{B}_w^n := \{\mathbf{x} \in \mathbb{F}_2^n : |\mathbf{x}| \leq w\}.$$

Binary linear codes

A binary linear code $\mathcal{C}$ of length n and dimension k — for short, an $[n, k]$-code — is a subspace of $\mathbb{F}_2^n$ of dimension k. The ratio $R = k/n$ is called the *rate* of the code. Every linear code can be described either by a set of linearly independent generators (basis representation) or by a system of modular equations (parity-check representation). We will mostly consider the first representation for our purpose.

To build an $[n, k]$-code we may take any set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k \in \mathbb{F}_2^n$ which are *linearly independent* and define:

$$\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_k) := \left\{ \sum_{i=1}^{k} z_i \mathbf{b}_i : z_i \in \mathbb{F}_2 \right\} \qquad \text{(Basis representation)}$$

We say that $\mathbf{b}_1, \ldots, \mathbf{b}_k$ is a basis for the code $\mathcal{C} = \mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$. Alternatively we will call the matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k] \in \mathbb{F}_2^{n \times k}$ a basis or a generating matrix of the code $\mathcal{C} = \mathcal{C}(\mathbf{B})$.

Properties

An element $\mathbf{c} \in \mathcal{C}$ of a code $\mathcal{C}$ is called a *codeword*. The support of the code is defined as the union of the supports of all its codewords, which also implies the definition of an *effective length* $|\mathcal{C}| \leq n$ of a $[n, k]$-code $\mathcal{C}$:

$$\mathrm{S}(\mathcal{C}) := \bigcup_{\mathbf{c} \in \mathcal{C}} \mathrm{S}(\mathbf{c}), \quad |\mathcal{C}| := \#\mathrm{S}(\mathcal{C}).$$

Indeed, the code length n defined by the ambient space is rather extrinsic information, in particular extending a code $\mathcal{C}$ by padding a 0 to all codewords does not affect the geometry of the code, but it does affect the apparent length n. To avoid unnecessary technicalities we assume in the rest of this chapter that each code has full support, i.e., $|\mathcal{C}| = n$.

Note that in contrast to lattices, the span of a code basis is automatically discrete due to its ambient space $\mathbb{F}_2^n$, while for lattice we need to restrict to the $\mathbb{Z}$-span to obtain the discreteness inside $\mathbb{R}^n$. Similar to lattices, the discreteness allows us to define a first minimum, or minimal distance $d_{\min}(\mathcal{C})$ of a code, as the shortest Hamming weight of nonzero codewords, namely:

$$d_{\min}(\mathcal{C}) := \min\{|\mathbf{c}| : \mathbf{c} \in \mathcal{C} \text{ and } \mathbf{c} \neq \mathbf{0}\}.$$

Hard problems

The problem of finding such a minimum length codeword, the analogue of the Shortest Vector Problem, is known as the Minimum Codeword Problem.

Definition 116 (Minimum Codeword Problem (MCP)). *Given a basis $\mathbf{B}$ of a binary $[n,k]$-code $\mathcal{C}$, compute a nonzero codeword $\mathbf{x} \in \mathcal{C}$ of minimum weight, i.e., such that $|\mathbf{x}| = d_{\min}(\mathcal{C})$.*

Similarly the Closest Vector Problem has a direct analogue in codes, usually named the Nearest Codeword Problem.

Definition 117 (Nearest Codeword Problem (NCP)). *Given a basis $\mathbf{B}$ of a binary $[n,k]$-code $\mathcal{C}$, and a target $\mathbf{t} \in \mathbb{F}_2^n$, compute a codeword $\mathbf{x} \in \mathcal{C}$ of minimum distance to $\mathbf{t}$, i.e., such that $|\mathbf{t} \oplus \mathbf{x}|$ is minimal.*

Alternatively this is called *decoding* the code $\mathcal{C}$.

Systematic form

An usual way in code-based cryptography to decode random codes is to use bases in *systematic form*. A basis $\mathbf{B}$ of an $[n,k]$-code is said to be in *systematic form* or in *reduced row echelon form* if up to a permutation of its rows $\mathbf{B} = (\mathbf{I}_k; \mathbf{B}')$ where $\mathbf{I}_k$ denotes the identity of size $k \times k$ and $\mathbf{B}' \in \mathbb{F}_2^{(n-k)\times k}$. Such bases can be produced from any basis by doing a Gaussian elimination over the columns. Choosing the set of pivots iteratively (possibly at random among available ones), this can be done in time $O(nk^2)$.

8.3 Orthopodality and the epipodal matrix

Let us start by recalling the standard definition of Gram-Schmidt Orthogonalisation (GSO) over Euclidean vector spaces. Given a basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of the Euclidean space $(\mathbb{R}^n, \|\cdot\|)$ its Gram-Schmidt orthogonalisation $(\tilde{\mathbf{b}}_1; \ldots; \tilde{\mathbf{b}}_n)$ is defined inductively by:

$$\tilde{\mathbf{b}}_i := \pi_i(\mathbf{b}_i) \quad \text{where } \pi_i : \mathbf{x} \mapsto \mathbf{x} - \sum_{j<i} \frac{\langle \mathbf{x}, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle} \cdot \tilde{\mathbf{b}}_j.$$

The map π_i denotes the orthogonal projection onto the orthogonal of the space generated by vectors $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ in $\mathbb{R}^n$. While the GSO of the basis of a lattice *is not* itself a basis of that lattice, it is a central object in the reduction theory of lattices, and in lattice reduction algorithms. This section is dedicated to the construction of an analogue object for bases of binary linear codes.

8.3.1 An orthogonality notion for binary vectors

In the case of Euclidean vector spaces $\mathbb{R}^n$, orthogonality can be defined via the standard inner-product, namely $\mathbf{x} \perp \mathbf{y} : \langle \mathbf{x}, \mathbf{y} \rangle = 0$ and so orthogonal projections onto the line spanned by $\mathbf{x}$ as $\pi_{\mathbf{x}}(\mathbf{y}) := \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\langle \mathbf{x}, \mathbf{x} \rangle} \mathbf{x}$. Orthogonality also provides Pythagorian additivity:

$$\|\mathbf{x} \oplus \mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2.$$

The space $\mathbb{F}_2^n$ is also endowed with an inner-product, but it does not lead to a geometrically meaningful notion of orthogonality. For instance for $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$, $\langle \mathbf{x}, \mathbf{y} \rangle = 0 \pmod 2$ does not seem to imply anything similar to Pythagorian additivity. However we note that, by definition of the Hamming weight we do have:

$$|\mathbf{x} \oplus \mathbf{y}| = |\mathbf{x}| + |\mathbf{y}| \iff \mathrm{S}(\mathbf{x}) \cap \mathrm{S}(\mathbf{y}) = \emptyset.$$

In fact, we even have the identity:

$$|\mathbf{x} \oplus \mathbf{y}| = |\mathbf{x}| + |\mathbf{y}| - 2|\mathbf{x} \wedge \mathbf{y}|$$

for $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$, which should be read as an analogue of the Euclidean identity:

$$\|\mathbf{x} - \mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\langle \mathbf{x}, \mathbf{y} \rangle.$$

This suggests to define $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ to be *orthopodal* if their supports are disjoint, that is, defining the orthogonality relation as:

$$\mathbf{x} \perp \mathbf{y} \iff \mathbf{x} \wedge \mathbf{y} = \mathbf{0}.$$

We can then associate convenient notions of projections *onto* (the support of) $\mathbf{x}$ and *orthopodally to* (the support of) $\mathbf{x}$ as follows:

$$\pi_{\mathbf{x}} : \mathbf{y} \mapsto \mathbf{y} \wedge \mathbf{x}, \quad \pi_{\mathbf{x}}^{\perp} : \mathbf{y} \mapsto \mathbf{y} \wedge \overline{\mathbf{x}} = \pi_{\overline{\mathbf{x}}}(\mathbf{y})$$

Such transformations are certainly not new to codes, and known in the literature as *puncturing* [MS77, Ch.1 §9]. However, we are here especially interested in their geometric virtues. They do satisfy similar properties as their Euclidean analogues: $\pi_{\mathbf{x}}$ is linear, idempotent, it fixes $\mathbf{x}$, it does not increase length (Hamming weight), and together with $\pi_{\mathbf{x}}^{\perp}$ yields an orthogonal decomposition. More formally:

Lemma 118. *For any $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_2^n$ it holds that:*

$$\begin{aligned} \pi_{\mathbf{x}}(\mathbf{y} \oplus \mathbf{z}) = \pi_{\mathbf{x}}(\mathbf{y}) \oplus \pi_{\mathbf{x}}(\mathbf{z}), \quad & \pi_{\mathbf{x}}^2(\mathbf{y}) = \pi_{\mathbf{x}}(\mathbf{y}), \\ \pi_{\mathbf{x}}^{\perp}(\mathbf{x}) = \mathbf{0}, \quad & \pi_{\mathbf{x}}(\mathbf{x}) = \mathbf{x}, \\ \pi_{\mathbf{x}}^{\perp}(\mathbf{y}) \perp \mathbf{x}, \quad & |\pi_{\mathbf{x}}(\mathbf{y})| \leq |\mathbf{y}|, \\ \pi_{\mathbf{x}}(\mathbf{y}) \perp \pi_{\mathbf{x}}^{\perp}(\mathbf{y}), \quad & \pi_{\mathbf{x}}(\mathbf{y}) \oplus \pi_{\mathbf{x}}^{\perp}(\mathbf{y}) = \mathbf{y}. \end{aligned}$$

The proof is immediate by boolean algebra. Furthermore, and unlike their Euclidean analogues: they always commute and their compositions can be compactly represented.

Lemma 119. *For any $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ it holds that:*

$$\pi_{\mathbf{x}} \circ \pi_{\mathbf{y}} = \pi_{\mathbf{y}} \circ \pi_{\mathbf{x}} = \pi_{\mathbf{x} \wedge \mathbf{y}},$$

$$\pi_{\mathbf{x}}^{\perp} \circ \pi_{\mathbf{y}}^{\perp} = \pi_{\mathbf{y}}^{\perp} \circ \pi_{\mathbf{x}}^{\perp} = \pi_{\mathbf{x} \vee \mathbf{y}}^{\perp}.$$

The proof is also immediate by boolean algebra. We therefore extend the notation $\pi_S^{\perp}$ to sets $S \subseteq \mathbb{F}_2^n$ to denote the projection orthopodally to the support of S, namely $\pi_{\mathbf{x}}^{\perp}$ where $\mathbf{x} = \bigvee_{\mathbf{s} \in S} \mathbf{s}$. This compact representation will allow for various algorithmic speed-ups.

8.3.2 Epipodal matrix

We are now fully equipped to define an analogue of the Gram-Schmidt Orthogonalisation process over real matrices to the case of binary matrices. An important remark is that this analogue notion given below does not preserve the $\mathbb{F}_2$-span of partial bases, which may appear as breaking the analogy with the GSO over the reals which precisely preserves $\mathbb{R}$-span of those partial bases. This is in fact not the right analogy for our purpose, noting that the GSO does not preserve the $\mathbb{Z}$-spans of those partial bases. The proper lattice to code translation (Table 8.1) associates $\mathbb{Z}$-spans (*i.e.* lattices) to $\mathbb{F}_2$-spans (*i.e.* codes), and $\mathbb{R}$-spans to *supports*.

Definition 120 (Epipodal matrix). *Let $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k] \in \mathbb{F}_2^{n \times k}$ be a binary matrix. The i-th projection associated to this matrix is defined as $\pi_i := \pi^{\perp}_{\{\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}\}}$ where π_1 denotes the identity. Equivalently,*

$$\begin{array}{rccc} \pi_i : & \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ & \mathbf{c} & \longmapsto & \mathbf{c} \wedge \overline{(\mathbf{b}_1 \vee \cdots \vee \mathbf{b}_{i-1})}. \end{array}$$

The i-th epipodal vector is then defined as:

$$\mathbf{b}_i^+ := \pi_i(\mathbf{b}_i),$$

and the matrix $\mathbf{B}^+ := [\mathbf{b}_1^+, \ldots, \mathbf{b}_k^+] \in \mathbb{F}_2^{n \times k}$ is called the epipodal matrix of $\mathbf{B}$.

Note that the definition does not need to be restricted to full rank matrices. The i-th epipodal vector should be interpreted as the support increment from the code $\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$ to $\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_i)$. The epipodal matrix enjoys the following properties, analogue to the GSO.

Lemma 121 (Properties of Epipodal Matrices). *For any binary matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k] \in \mathbb{F}_2^{n \times k}$, its epipodal matrix $\mathbf{B}^+ = [\mathbf{b}_1^+, \ldots, \mathbf{b}_k^+]$ satisfies:*

1. *The epipodal vectors are pairwise orthopodal:*

$$\forall i \neq j, \quad \mathbf{b}_i^+ \perp \mathbf{b}_j^+. \tag{8.1}$$

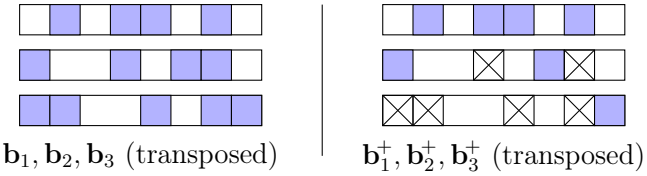


Figure 8.1: The (transposed) basis $\mathbf{B}$ of a $[8, 3]$-code, and its associated epipodal matrix $\mathbf{B}^+$.

2. *For all $i \leq k$, $(\mathbf{b}_1, \ldots, \mathbf{b}_i)$ and $[\mathbf{b}_1^+, \ldots, \mathbf{b}_i^+]$ have the same supports, that is:*

$$\bigcup_{j \leq i} \mathrm{S}(\mathbf{b}_j^+) = \bigcup_{j \leq i} \mathrm{S}(\mathbf{b}_j), \text{ or equivalently, } \bigvee_{j \leq i} \mathbf{b}_j^+ = \bigvee_{j \leq i} \mathbf{b}_j. \quad (8.2)$$

Proof. For any i, we have by definition that $\mathbf{b}_i^+ = \pi_i(\mathbf{b}_i) = \mathbf{b}_j \wedge \overline{(\mathbf{b}_1 \vee \cdots \vee \mathbf{b}_{i-1})}$, so it holds that $\mathrm{S}(\mathbf{b}_i^+) \subset \mathrm{S}(\mathbf{b}_i)$, and that $\mathrm{S}(\mathbf{b}_i^+) \cap \mathrm{S}(\mathbf{b}_j) = \emptyset$ for any $j < i$. Therefore $\mathrm{S}(\mathbf{b}_j^+) \cap \mathrm{S}(\mathbf{b}_i^+) = \emptyset$, that is $\mathbf{b}_i^+ \perp \mathbf{b}_j^+$. For the second item, rewrite $\mathbf{b}_j^+ = \mathbf{b}_j \wedge \bigwedge_{i<j} \overline{\mathbf{b}_i}$ and conclude by induction.

□

Furthermore, one may note that epipodal vectors satisfy a similar induction to the one of the GSO over the reals:

$$\text{GSO: } \tilde{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j<i} \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle} \cdot \tilde{\mathbf{b}}_j.$$

$$\text{Epipodal Matrix: } \mathbf{b}_i^+ = \mathbf{b}_i \oplus \sum_{j<i} \mathbf{b}_i \wedge \mathbf{b}_j^+,$$

However following this induction leads to perform $O(k^2)$ vector operations. In the case of the epipodal matrix, the computation can be sped-up to $O(k)$ vector operations using cumulative support vectors $\mathbf{s}_i$:

$$\mathbf{s}_0 = \mathbf{0}, \quad \mathbf{s}_i = \mathbf{s}_{i-1} \vee \mathbf{b}_i, \quad \mathbf{b}_i^+ = \mathbf{b}_i \wedge \overline{\mathbf{s}_{i-1}}.$$

The epipodal matrix of the basis of a code also enjoys an analogue invariant to the GSO for lattices. The GSO $(\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_k)$ of a lattice $\mathcal{L}$ is not generally a basis of $\mathcal{L}$ but it verifies the following invariant $\prod_{i=1}^{k} \|\tilde{\mathbf{b}}_i\| = \det(\mathcal{L})$.

Corollary 122 (Length Invariant). *For any basis* $[\mathbf{b}_1, \dots, \mathbf{b}_k]$ *of an* $[n, k]$*-code* $\mathcal{C}$*:*

$$\sum_{i=1}^{k} |\mathbf{b}_i^+| = |\mathcal{C}|.$$

Proof. Using equation (8.2) we have $\bigcup_{j \le n} \mathrm{S}(\mathbf{b}_j^+) = \bigcup_{j \le n} \mathrm{S}(\mathbf{b}_j) = \mathrm{S}(\mathcal{C})$, and according to (8.1) this union is disjoint. □

8.4 Size-reduction and its fundamental domain

While the GSO of a lattice basis is not itself a basis of that lattice, it is a central notion to define what a "good" basis is. For example, because of the invariant $\prod \|\tilde{\mathbf{b}}_i\| = \det(\mathcal{L})$, and because $\tilde{\mathbf{b}}_1 = \mathbf{b}_1$, making the first vector of a basis short means that other Gram-Schmidt lengths must grow.

The notion of quality of a basis should more specifically be linked to what we can do algorithmically with it. In the cases of lattices, the GSO of a basis allows to *tile* the space. More formally, if $\mathbf{B}$ is the basis of a full rank lattice $\mathcal{L} \subset \mathbb{R}^n$, one can define a fundamental domain of the translation action of $\mathcal{L}$ over $\mathbb{R}^n$ (*i.e.* a set of representatives of the quotient $\mathbb{R}^n/\mathcal{L}$) by the following rectangle parallelepiped:

$$\mathcal{P}(\tilde{\mathbf{B}}) := \left\{ \sum x_i \tilde{\mathbf{b}}_i \;\middle|\; -\tfrac{1}{2} \le x_i < \tfrac{1}{2} \right\} = \left[-\frac{1}{2}, \frac{1}{2} \right)^k \cdot \tilde{\mathbf{B}}$$

Furthermore, there is a polynomial time algorithm that effectively reduces points $\mathbf{x} \in \mathbb{R}^n$ modulo $\mathcal{L}$ to this parallelepiped, namely *size-reduction* [LLL82], also known as the *Nearest Plane Algorithm* [Bab86]. This parallelepiped has inner radius $r_{\text{in}} = \min \|\tilde{\mathbf{b}}_i\|/2$ and outer square radius $r_{\text{out}}^2 = \frac{1}{4} \sum \|\tilde{\mathbf{b}}_i\|^2$. This means that size-reduction can, in the worst case, find a close lattice vector at distance r_{out}, and correctly decode all errors of length up to r_{in}. One can also establish that the average squared distance of the decoding of a random coset is $\frac{1}{12} \sum \|\tilde{\mathbf{b}}_i\|^2$.

This Section is dedicated to an equivalent size-reduction algorithm for binary codes, and to the study of its associated fundamental domain.

8.4.1 Size-reduction: definition and algorithm

Let us start by defining size-reduction and its associated fundamental domain. A first technical detail is that in the case of codes, it is not given that the epipodal vectors are nonzero, a minor difference with the Gram-Schmidt Orthogonalisation for bases in a real vector space. We restrict our attention to *proper bases*.

Definition 123 (Proper bases). *A basis* $\mathbf{B}$ *is proper if all epipodal vectors* $\mathbf{b}_i^+$ *are nonzero.*

Note for example that bases in systematic form are proper bases: proper bases do exist for all codes, and can be produced from any basis in polynomial time.

A more annoying hiccup is that we may need to handle ties to prevent the size-reduction tiles from overlapping; in the case of lattices these might as well be ignored as the difference between $[-\frac{1}{2}, \frac{1}{2}]^n$ and $[-\frac{1}{2}, \frac{1}{2})^n$ has zero measure. This issue arises when an epipodal vector $\mathbf{p}$ has even weight, if we are reducing some $\mathbf{y} \in \mathbb{F}_2^n$ such that $|\mathbf{y} \wedge \mathbf{p}| = |\mathbf{p}|/2 = |(\mathbf{y} \oplus \mathbf{p}) \wedge \mathbf{p}|$. We (arbitrarily) use the first epipodal coordinate to break such ties:

$$\mathrm{TB}_{\mathbf{p}}(\mathbf{y}) = \begin{cases} 0 & \text{if } |\mathbf{p}| \text{ is odd,} \\ 0 & \text{if } y_j = 0 \text{ where } j = \min(\mathrm{S}(\mathbf{p})), \\ 1/2 & \text{otherwise.} \end{cases}$$

Definition 124 (size-reduction). *Let* $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ *be a basis of an* $[n, k]$*-code. The Size-Reduced region relative to* $\mathbf{B}$ *is defined as:*

$$\mathcal{F}(\mathbf{B}^+) := \left\{ \mathbf{y} \in \mathbb{F}_2^n \ : \ \forall i \in [\![1, k]\!], \ |\mathbf{y} \wedge \mathbf{b}_i^+| + \mathrm{TB}_{\mathbf{b}_i^+}(\mathbf{y}) \leq \frac{|\mathbf{b}_i^+|}{2} \right\}.$$

Vectors in this region are said to be size-reduced with respect to the basis $\mathbf{B}$*.*

Furthermore, as for lattices, we have an efficient size-reduction algorithm that reduces any target to this region.

Proposition 125. *Algorithm 12 is correct and runs in polynomial time.*

Algorithm 12: SizeRed($\mathbf{B}, \mathbf{y}$) Size-reduce $\mathbf{y}$ with respect to $\mathbf{B}$

Input : A basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k] \in \mathbb{F}_2^{k \times n}$ and a target $\mathbf{y} \in \mathbb{F}_2^n$
Output: $\mathbf{e} \in \mathcal{F}(\mathbf{B}^+)$ such that $\mathbf{e} \oplus \mathbf{y} \in \mathcal{C}(\mathbf{B})$

```
e ← y
for i = k down to 1 do
    if |e ∧ b_i^+| + TB_{b_i^+}(e) > |b_i^+|/2 then
        e ← e ⊕ b_i
    end
end
return e
```

Proof. First note that $\mathbf{e} \oplus \mathbf{y} \in \mathcal{C}(\mathbf{B})$ is a loop invariant, as we only add basis vectors $\mathbf{b}_i$ to $\mathbf{e}$. Therefore all we need to show is that $\mathbf{e} \in \mathcal{F}(\mathbf{B}^+)$. Note that the loop at step i enforces $|\mathbf{e} \wedge \mathbf{b}_i^+| + \mathrm{TB}_{\mathbf{b}_i^+}(\mathbf{e}) \leq |\mathbf{b}_i^+|/2$. Furthermore, this constraint is maintained by subsequent loop iterations of index $j < i$ since $\mathbf{b}_j \wedge \mathbf{b}_i^+ = \mathbf{0}$. □

Proposition 126 (Fundamental Domain). *Let $\mathbf{B} := [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ be a proper basis of an $[n, k]$-code $\mathcal{C}$. Then $\mathcal{F}(\mathbf{B}^+)$ is a fundamental domain for $\mathcal{C}$, that is:*

1. *$\mathcal{F}(\mathbf{B}^+)$ is $\mathcal{C}$-packing:*

$$\forall \mathbf{c} \in \mathcal{C} \setminus \{\mathbf{0}\}, \quad (\mathbf{c} + \mathcal{F}(\mathbf{B}^+)) \cap \mathcal{F}(\mathbf{B}^+) = \emptyset,$$

2. *$\mathcal{F}(\mathbf{B}^+)$ is $\mathcal{C}$-covering:*

$$\mathcal{C}(\mathbf{B}) + \mathcal{F}(\mathbf{B}^+) = \mathbb{F}_2^n.$$

Proof. Let us start by proving that $\mathcal{F}(\mathbf{B}^+)$ is $\mathcal{C}(\mathbf{B})$-packing. Let $\mathbf{c} = \sum_i x_i \mathbf{b}_i \in \mathcal{C}(\mathbf{B})$ and $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{F}(\mathbf{B}^+)$ such that $\mathbf{c} = \mathbf{y}_1 \oplus \mathbf{y}_2$. By definition of the orthopodalization $\mathbf{c} = \sum_i x_i \mathbf{b}_i^+ \oplus \sum_{j<i} x_i \mathbf{b}_i \wedge \mathbf{b}_j^+$ which gives:

$$\forall \ell \in [\![1, k]\!], \quad \mathbf{c} \wedge \mathbf{b}_\ell^+ = x_\ell \mathbf{b}_\ell^+ \oplus \sum_{i>\ell} x_i \mathbf{b}_i \wedge \mathbf{b}_\ell^+.$$

Suppose by contradiction that $\mathbf{c} \neq \mathbf{0}$. Let j be the largest index such that $x_j \neq 0$. Then $\mathbf{c} \wedge \mathbf{b}_j^+ = \mathbf{b}_j^+$. As $\mathbf{c} = \mathbf{y}_1 \oplus \mathbf{y}_2$ where $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{F}(\mathbf{B}^+)$,

$$|\mathbf{c} \wedge \mathbf{b}_j^+| \leq |\mathbf{y}_1 \wedge \mathbf{b}_j^+| + |\mathbf{y}_2 \wedge \mathbf{b}_j^+| < |\mathbf{b}_j^+|,$$

which is a contradiction. Therefore $\mathbf{c} = \mathbf{0}$ which shows that $\mathcal{F}(\mathbf{B}^+)$ is $\mathcal{C}(\mathbf{B})$-packing. The $\mathcal{C}(\mathbf{B})$-covering property follows from the fact that Algorithm 12 is correct as proven in Proposition 125. □

8.4.2 Decoding performance of size-reduction

Given any fundamental domain $\mathcal{F}$ of an $[n, k]$-code $\mathcal{C}(\mathbf{B})$ and a corresponding reduction algorithm one can consider the decoding performance. Such a reduction algorithm would reduce a target $\mathbf{y} \in \mathbb{F}_2^n$ to the (unique) error $\mathbf{e} \in (\mathbf{y} + \mathcal{C}) \cap \mathcal{F}$, and thereby finding a close codeword $\mathbf{c} = \mathbf{y} \oplus \mathbf{e}$ to $\mathbf{y}$. A uniformly random target in $\mathbb{F}_2^n$ is uniformly distributed over the fundamental domain after reduction and thus the quality of decoding fully dependents on the geometric properties of the fundamental domain. For example, the expected Hamming distance is equal to the expected weight $\mathbb{E}[|\mathcal{U}(\mathcal{F})|]$. And, more precisely, decoding a random target, to a codeword at Hamming distance w, succeeds with probability:

$$p_{\leq w}^{\text{rand}}(\mathcal{F}) = \frac{\#(\mathcal{F} \cap \mathcal{B}_w^n)}{\#\mathcal{F}} = \frac{\#(\mathcal{F} \cap \mathcal{B}_w^n)}{2^{n-k}}.$$

The minimal distance $d := d_{\min}(\mathcal{C})$ of a random code $\mathcal{C}$ is expected to be very close to the Gilbert-Varshamov bound [OS09, §3.2, Definition 1], and a uniformly random target lies almost always at distance $\approx d$ (see [MO15, §2] for a justification). Therefore in this setting random decoding is mostly interesting for $w \geq d$, as otherwise no solution is expected to exist.

Another regime we can consider is unique decoding, where we have the guarantee that our target has a unique codeword at distance at most w. For general codes this is the case for half distance decoding up to weight $w < d/2$. If the error is uniform over $\mathcal{B}_w^n$, we obtain a success probability of

$$p_{\leq w}^{\text{uniq}}(\mathcal{F}) = \frac{\#\mathcal{F} \cap \mathcal{B}_w^n}{\#\mathcal{B}_w^n}.$$

For random codes a target at distance at most $w < d$ (instead of $d/2$) is almost always uniquely decodable and as a result the success probability is also close to the above quantity. For cryptanalytic purposes it is assumed to be identical [OS09, page 3.3].

Let us now consider three such fundamental domains, an optimal one, the one corresponding to an algorithm by Prange [Pra62], and finally the size-reduction domain $\mathcal{F}(\mathbf{B}^+)$.

Maximum likelihood decoding

An optimal decoder, also know as a maximum likelihood decoder, always decodes to a nearest codeword. Implicitly this corresponds to a fundamental domain $\mathcal{F}$ where each coset representative has minimal weight. This implies that the random decoding probability $p^{\text{rand}}_{\leq w}(\mathcal{F})$ hits the probability that a uniformly random target lies at distance at most w to the code, and a perfect unique decoding probability of $p^{\text{uniq}}_{\leq w}(\mathcal{F}) = 1$. For lattices the analogue fundamental domain is unique up to the boundary and is known as the *Voronoi Domain* of a lattice. Unfortunately, reducing a target to this fundamental domain is in general hard for both lattices and codes, and takes exponential time [Pra62; MV13; DLW19].

Prange's fundamental domains

Given a basis $\mathbf{B}$ in systematic form, a more common decoding algorithm, namely the Prange algorithm [Pra62], is to assume an error of $\mathbf{0}$ on the k pivot positions of the systematic form. This induces a fundamental domain, but of geometric shape $\mathbb{F}_2^{n-k} \times \{0\}^k$ instead of $\mathcal{F}(\mathbf{B}^+)$. This leads respectively to an expected error weight of $(n-k)/2$, and random and unique decoding probabilities of:

$$p^{\text{rand}}_{\leq w}(\mathbb{F}_2^{n-k} \times \{0\}^k) = \frac{\#\mathcal{B}_w^{n-k}}{2^{n-k}}, \quad \text{and} \quad p^{\text{uniq}}_{\leq w}(\mathbb{F}_2^{n-k} \times \{0\}^k) = \frac{\#\mathcal{B}_w^{n-k}}{\#\mathcal{B}_w^n}.$$

Size-reduction

In the case of lattices, size-reduction gives a fundamental domain that can be written as a direct sum of segments $\mathcal{P}(\tilde{\mathbf{B}}) = \prod_i [-1/2, 1/2) \cdot \tilde{\mathbf{b}}_i$ where the $\tilde{\mathbf{b}}_i$'s are the GSO of the lattice basis. The expected squared

decoding error $\frac{1}{12}\sum\|\tilde{\mathbf{b}}_i\|^2$ is simply the sum of the expected squared error on each segment, and only depends on the Gram-Schmidt profile $\|\tilde{\mathbf{b}}_1\|,\ldots,\|\tilde{\mathbf{b}}_k\|$.

We proceed similarly for the Size-Reduced region $\mathcal{F}(\mathbf{B}^+)$. The role of the segment is taken over by the *fundamental ball* $\mathcal{E}^p$ of length $p>0$ as:

$$\mathcal{E}^p := \left\{\mathbf{y}\in\mathbb{F}_2^p \ :\ |\mathbf{y}| + \mathrm{TB}_{(1,\ldots,1)}(\mathbf{y}) \le p/2\right\}.$$

It should be thought of as the canonical fundamental domain of the $[p,1]$-code $\mathcal{C}=\mathbb{F}_2\cdot(1,1,\ldots,,1)$ inside $\mathbb{F}_2^p$ (the repetition code of length p). To each proper epipodal vector $\mathbf{b}_i^+$ we assign an *epipodal ball* $\mathcal{E}^{|\mathbf{b}_i^+|}$, and this allows to rewrite the fundamental domain $\mathcal{F}(\mathbf{B}^+)$ as a direct product of balls via the isometry:

$$\mathcal{F}(\mathbf{B}^+) \xrightarrow{\sim} \prod_{i=1}^{k} \mathcal{E}^{|\mathbf{b}_i^+|} : \quad \mathbf{y}\mapsto \left(\mathbf{y}|_{\mathrm{S}(\mathbf{b}_1^+)},\ldots,\mathbf{y}|_{\mathrm{S}(\mathbf{b}_k^+)}\right).$$

The latter object only depends on the epipodal lengths $|\mathbf{b}_1^+|,\ldots,|\mathbf{b}_k^+|$ which we call the *profile* $(\ell_i := |\mathbf{b}_i^+|)_i$ of the basis $\mathbf{B}$.

We can now proceed to analyse the Hamming weight of uniformly random targets in the domain by looking at their weight in each local fundamental ball $\mathcal{E}^{|\mathbf{b}_i^+|}$. Let $W_p := |\mathcal{U}(\mathcal{E}^p)|$ for $p>0$ be the distribution of the Hamming weight of uniformly drawn targets in the fundamental ball $\mathcal{E}^p$. This weight distribution has the following probabilities for integer weight $w\ge 0$:

$$\Pr\left[W_p=w\right] = \begin{cases} 0 & \text{if } w>p/2 \text{ or } w<0,\\ \frac{\binom{p}{p/2}}{2^p} & \text{if } w=p/2,\\ \frac{\binom{p}{w}}{2^{p-1}} & \text{otherwise,}\end{cases}$$

and its expectation is given by:

$$\mathbb{E}\left[W_p\right] = \frac{p}{2} - \left\lceil\frac{p}{2}\right\rceil\cdot\binom{p}{\lfloor\frac{p}{2}\rfloor}\cdot 2^{-p} = \frac{p}{2} - \sqrt{\frac{p}{2\pi}} + \Theta(1/\sqrt{p}) \qquad (8.3)$$

which is bounded by $\frac{p-1}{2}$. This bound is strict for $p\ge 3$, which will give us a gain over the Prange decoder [Pra62]. Analogues to the lattice case the expected Hamming weight of a uniformly random target in $\mathcal{E}^p$ is given by the sum of the local expectations $\sum_{i=1}^{k}\mathbb{E}\left[W_{\ell_i}\right]$.

To be more precise, for a basis $\mathbf{B}$ with profile $\ell = (\ell_1, \dots, \ell_k)$, the weight distribution $W(\mathbf{B}) := |\mathcal{U}(\mathcal{F}(\mathbf{B}^+))|$ of the size-reduction algorithm, simply denoted by $W(\ell)$, is given by a convolution of the local weight distributions $W_{\ell_1}, \dots, W_{\ell_k}$:

$$\Pr[W(\mathbf{B}) = w] = \sum_{\sum_i w_i = w} \left(\prod_{i=1}^{k} \Pr[W_{\ell_i} = w_i] \right).$$

The whole distribution can efficiently (in time polynomial in n) be computed by iterated convolutions, as its support $[\![0, n]\!]$ is discrete and small (see `weights.py`).

8.4.3 Comparing profiles for size-reduction decoding

We have shown that the geometric shape of the fundamental domain $\mathcal{F}(\mathbf{B}^+)$ depends fully on the profile $\ell_1, \dots, \ell_k$. For any profile we have $\mathbb{E}[W(\ell)] = \sum_{i=1}^{k} \mathbb{E}[W_{\ell_i}] \leq \sum_{i=1}^{k} \frac{\ell_i - 1}{2} = (n-k)/2$, with a strict inequality if $|\ell_i| \geq 3$ for any $i \in [\![1, k]\!]$, and thus we improve on the fundamental domain induced by Prange. But what is actually a good profile?

Let us first focus on the expected error weight. By eq. (8.3) the expectation roughly equals

$$\mathbb{E}[W(\ell)] \approx \sum_{i=1}^{k} \frac{\ell_i}{2} - \sqrt{\frac{\ell_i}{2\pi}} = n/2 - \frac{1}{\sqrt{2\pi}} \sum_{i=1}^{k} \sqrt{\ell_i},$$

or more precisely we have the following statement, which follows from the inequalities $\frac{4^k}{\sqrt{\pi(k+1)}} \leq \binom{2k}{k} \leq \frac{4^k}{\sqrt{\pi k}}$.

Lemma 127. *Given a proper basis $\mathbf{B}$ of an $[n, k]$-code with profile $\ell = (\ell_1, \dots, \ell_k)$ we have:*

$$\frac{1}{\sqrt{\pi}} \sum_{i=1}^{k} \sqrt{\frac{\left\lceil \frac{\ell_i}{2} \right\rceil^2}{\left\lceil \frac{\ell_i}{2} \right\rceil + 1}} \leq \frac{n}{2} - \mathbb{E}[W(\mathbf{B})] \leq \frac{1}{\sqrt{\pi}} \sum_{i=1}^{k} \sqrt{\left\lceil \frac{\ell_i}{2} \right\rceil}.$$

Since $x \mapsto \sqrt{x}$ is concave, Lemma 127 (ignoring the rounding) suggest that the expected error is minimised when the ℓ_i are the most

balanced. Again a similar phenomenon is well known in the case of lattices: on random inputs, size-reduction produces vectors with an expected squared length of $\frac{1}{12}\sum \|\tilde{\mathbf{b}}_i\|^2$; under the invariant $\prod \|\tilde{\mathbf{b}}_i\| = \det(\mathcal{L})$ the expectation is minimised for a basis with a balanced profile $\|\tilde{\mathbf{b}}_1\| = \|\tilde{\mathbf{b}}_2\| = \cdots = \|\tilde{\mathbf{b}}_k\|$.

However, the quantity $\mathbb{E}[W(\mathbf{B})]$ discussed above does not necessarily reflect the quality of the basis for all relevant algorithmic tasks. For example, if one wishes to decode errors of weight at most w with a 100% success probability, it is necessary and sufficient that $w < \min_i \ell_i/2$.

We therefore propose the following partial ordering on profiles that is meant to account that a profile is better than another for the mentioned natural decoding tasks; as a counterpart, this is only a partial ordering and two profiles may simply be incomparable.

Definition 128 (Comparing Profiles). *We define a partial ordering $(\mathcal{L}_{n,k}, \preceq)$ on the set of proper profiles $\mathcal{L}_{n,k}$ of $[n,k]$-codes by:*

$$\ell \preceq \ell' \iff \Pr[W(\ell) \leq w] \geq \Pr[W(\ell') \leq w] \text{ for all } w \in [\![0, n]\!].$$

This also defines an equivalence relation $\asymp$ on $\mathcal{L}_{n,k}$. We call a profile ℓ better than ℓ' if $\ell \preceq \ell'$. We call ℓ strictly better than ℓ' and write $\ell \prec \ell'$ if $\ell \preceq \ell'$ and $\ell \not\asymp \ell'$. We call ℓ, ℓ' incomparable and write $\ell \nparallel \ell'$ if $\ell \not\preceq \ell'$ and $\ell \not\succeq \ell'$.

Let us first justify the relevance of this partial ordering for random decoding and unique decoding for an $[n,k]$-code $\mathcal{C}(\mathbf{B})$ with profile ℓ.

Random decoding

The probability to successfully decode a random target up to an error of weight at most w can directly be expressed as $\Pr[W(\ell) \leq w]$. For a better profile we see that this probability will also be higher. The expected distance is equal to $\mathbb{E}[W(\ell)]$. By noting that $\mathbb{E}[W(\ell)] = n - \sum_{w=0}^{n} \Pr[W(\ell) \leq w]$ we see that a better profile also implies a lower expected distance.

Unique decoding

For unique decoding up to weight $w < d_{\min}(\mathcal{C}(\mathbf{B}))/2$ we can also rewrite the success probability in terms of the weight distribution as:

$$\frac{\#\mathcal{F}(\mathbf{B}^+) \cap \mathcal{B}_w^n}{\#\mathcal{B}_w^n} = \frac{2^{n-k} \cdot \Pr[W(\ell) \leq w]}{\sum_{i=0}^{w} \binom{n}{i}}.$$

Again we see that a better profile gives a higher success probability.

The more balanced, the better

Now that we have argued that the ordering of Definition 128 is relevant, let us show that, indeed, balanced profiles are preferable. As we will see, this rule of thumb is in fact imperfect, and only apply strictly to profiles that share the same parities $(\wp_i := \ell_i \bmod 2)_i$.

Lemma 129 (Profile Relations). *The partial ordering $\preceq$ has the following properties:*

(1) If ℓ is a permutation of ℓ' then $\ell \asymp \ell'$.

(2) if $\ell_1 \preceq \ell_1'$ and $\ell_2 \preceq \ell_2'$, then $(\ell_1|\ell_2) \preceq (\ell_1'|\ell_2')$.

(3) If $3 \leq x \leq y+1$, then $(x, y) \prec (x-2, y+2)$.

Proof. (1) follows from the fact that the geometric properties of the size-reduced region fully depend on the values of the profile (and not their ordering). For (2) note that $\ell_i \preceq \ell_i'$ implies the existence of a one-to-one map f_i from the size-reduction domain $\mathcal{F}(\ell_i)$ to $\mathcal{F}(\ell_i')$ that is non-decreasing in weight for $i = 1, 2$. The product map $f_1 \times f_2$ is then a one-to-one map from $\mathcal{F}(\ell_1|\ell_2)$ to $\mathcal{F}(\ell_1'|\ell_2')$ that is non-decreasing in weight, which implies that $(\ell_1|\ell_2) \preceq (\ell_1'|\ell_2')$. For the technical proof of (3) see the full version of this work[3].

□

[3] Available at https://eprint.iacr.org/2020/869.

An odd game of parity

Although for fixed parity a balanced profile is always better than an unbalanced one there are some exceptions to this rule when dropping the parity constraint. For example, looking at Lemma 127 one can notice that, at least for average decoding distances, odd values in a profile are preferable to even values. One can show that a slight unbalance with odd coefficients is preferable for all purpose to a perfect even balance:

$$(x-1, x+1) \preceq (x, x) \quad \text{for all even } x \geq 2.$$

This is an artefact of the need to tie-break certain size-reductions with respect to epipodal vectors of even length.

8.4.4 Basis size-reduction

To complete the analogy with the lattice literature, let us now adapt the notion of size-reduction for a basis. We call a basis size-reduced if each basis vector is size-reduced with respect to all previous basis vectors.

Definition 130. *We call a proper basis* $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_k]$ *size-reduced if* $\mathbf{b}_i \in \mathcal{F}([\mathbf{b}_1, \dots, \mathbf{b}_{i-1}]^+)$ *for all* $1 < i \leq k$.

Size-reduction for a basis allows to control the basis vector lengths with the epipodal lengths as follows.

Proposition 131. *Let* $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_k]$ *be a size-reduced basis with epipodal lengths* $\ell_i = |\mathbf{b}_i^+|$. *Then, for all* $i \leq n$ *it holds that* $|\mathbf{b}_i| \leq \ell_i + \sum_{j<i} \lfloor \ell_j/2 \rfloor$ *for all* $i \leq k$.

Perhaps surprisingly, this global notion of size-reduction will not be required in the LLL algorithm for codes discussed in the next Section 8.5.2, which is a first deviation from the original LLL algorithm for lattices [LLL82]. However it can still be useful to adapt more powerful reduction algorithms such as deepLLL [SE94; FSW14].

Proposition 132. *Algorithm 13 is correct and runs in polynomial time.*

Algorithm 13: SizeRedBasis($\mathbf{B}, \mathbf{y}$) Size-reduce the basis $\mathbf{B}$

Input : A proper basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k] \in \mathbb{F}_2^{k \times n}$ of a code $\mathcal{C}$
Output: A size-reduced basis of $\mathcal{C}(\mathbf{B})$ with the same epipodal matrix as $\mathbf{B}$.

```
for i = 2 to k do
    b_i ← SizeRed([b_1, ..., b_{i-1}], b_i)
end
return [b_1, ..., b_k]
```

Proof. The polynomial claim immediately follows from Proposition 125. Secondly note that vectors $\mathbf{b}_i$'s form a basis of the code $\mathcal{C}$ given as input, and this is a loop invariant. Indeed, in any step i of the algorithm only codewords from the sub-code $\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$ are added to $\mathbf{b}_i$. Furthermore, this does not affect $\mathbf{b}_i^+ := \pi_i(\mathbf{b}_i)$. The loop at step i enforces that $\mathbf{b}_i \in \mathcal{F}([\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}]^+)$ and this constraint is maintained as $\mathbf{b}_1, \ldots, \mathbf{b}_i$ are unchanged by all later steps. □

8.5 LLL for binary codes

In the previous section, we have seen that the geometric quality of the fundamental domain $\mathcal{F}(\mathbf{B}^+)$ solely depends upon the epipodal lengths $\ell_i := |\mathbf{b}_i^+|$: the more balanced, the better, both for finding close codewords of random words, and for decoding random errors. This situation is in perfect analogy with the situation in lattices. We therefore turn to the celebrated LLL [LLL82] algorithm for lattice reduction, which aims precisely at balancing the profile $(\ell_i)_i$.

The LLL algorithm can be interpreted as an algorithmic version of the so-called Hermite's bound on the minimal length of an n-dimensional vector [GHKN06]. Again, the analogy between codes an lattices stands: the LLL reduction for codes turns out to be an algorithmic version of Griesmer's bound [Gri60].

Certainly, Griesmer's bound [Gri60] is far from tight in all regimes for the parameters of the code, as it is already the case with Hermite's bound for lattices which is exponentially weaker than Minkowski's bound. Griesmer's bound and Hermite's bound virtues reside in the algorithm underlying their proofs.

8.5.1 Griesmer's bound and LLL reduction

In this section, we revisit the classical Griesmer's bound and its proof from the perspective of reduction theory, that is we will re-interpret its proof in terms of the epipodal matrix and in particular its profile. The proof we propose is admittedly a bit less direct than the original; our purpose is to dissect this classic proof, and extract an analogue to LLL reduction for codes.

Theorem 133 (Griesmer Bound [Gri60]). *For any $[n,k]$-code of minimal distance $d := d_{\min}(\mathcal{C})$, it holds that:*

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil .$$

In particular, if $k-1 \geq \log_2(d)$, it holds that $d - \frac{\lceil \log_2(d) \rceil}{2} \leq \frac{n-k}{2} + 1$.

The latter inequality follows from the first by setting $r = \lceil \log_2(d) \rceil$ as follows:

$$\begin{aligned} n &\geq d \sum_{i=0}^{r-1} \frac{1}{2^i} + \sum_{i=r}^{k-1} 1 \\ &= 2d(1 - \frac{1}{2^r}) + (k-r) \\ &\geq 2d - 2 + k - r. \end{aligned}$$

Definition 134 (Griesmer-reduced basis). *A basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ of an $[n,k]$-code is said to be* Griesmer*-reduced if $\mathbf{b}_i^+$ is a shortest nonzero codeword of the projected subcode $\pi_i(\mathcal{C}(\mathbf{b}_i, \ldots, \mathbf{b}_k))$ for all $i \in [\![1,k]\!]$.*

This definition is a direct analogue of the so-called Hermite-Korkine-Zolotarev (HKZ) reduction for lattice bases. Note that the existence of such a basis is rather trivial by construction: choose $\mathbf{b}_1$ as a shortest nonzero vector and so forth. The only minor difficulty is showing that the projected codes $\pi_i(\mathcal{C}(\mathbf{b}_i, \ldots, \mathbf{b}_k))$ are non-trivial, which can be done by resorting to Singleton's bound. In particular, Griesmer-reduced bases are proper bases.

Lemma 135 ([HP10, Corollary 2.7.2]). *Let $\mathcal{C}$ be an $[n,k]$-code and $\mathbf{c}$ be a codeword of weight $d_{\min}(\mathcal{C})$. Then $\mathcal{C}' := \pi_{\mathbf{c}}^{\perp}(\mathcal{C}) = \mathcal{C} \wedge \overline{\mathbf{c}}$ satisfies:*

1. *$|\mathcal{C}'| = n - d_{\min}(\mathcal{C})$ and its dimension is $k-1$,*
2. *$d_{\min}(\mathcal{C}') \geq \lceil d_{\min}(\mathcal{C})/2 \rceil$.*

Therefore with the first point of this lemma we can prove by induction on k that there exists for any $[n,k]$-code $\mathcal{C}$ a Griesmer-reduced basis. Let $[\mathbf{b}_1, \ldots, \mathbf{b}_k]$ be such a basis and let $\ell_i := |\mathbf{b}_i^+|$. From definition of Griesmer-reduced bases and the previous lemma we deduce that $\ell_{i+1} \geq \lceil \ell_i/2 \rceil$. In other words, the profile $(\ell_i)_i$ is somewhat controlled: it *does not decrease too fast*. To prove Griesmer's bound it remains to chain those inequalities and to sum them up to obtain:

$$n \geq |\mathcal{C}| = \sum_{i=1}^{k} \ell_i \geq \sum_{i=0}^{k-1} \left\lceil \frac{\ell_1}{2^i} \right\rceil = \sum_{i=0}^{k-1} \left\lceil \frac{d_{\min}(\mathcal{C})}{2^i} \right\rceil \tag{8.4}$$

The proof of Lemma 135 proceeds by a *local* minimality argument, namely it looks at the first two vectors $\mathbf{b}_1, \mathbf{b}_2$. It shows that the support of $\mathbf{b}_1$ is at most $2/3$ of the support of $\mathcal{C}(\mathbf{b}_1, \mathbf{b}_2)$:

$$|\mathbf{b}_1| \leq \tfrac{2}{3} \cdot \big|\mathcal{C}(\mathbf{b}_1, \mathbf{b}_2)\big|.$$

The proof is rather elementary as the code $\mathcal{C}(\mathbf{b}_1, \mathbf{b}_2)$ has only 3 nonzero codewords to consider: $\mathbf{b}_1, \mathbf{b}_2$ and $\mathbf{b}_1 \oplus \mathbf{b}_2$. What we should note here is that the notion of Griesmer-reduction is stronger than what is actually used by the proof: indeed, we only need the much weaker property that $\mathbf{b}_1$ is a shortest codeword of the 2-dimensional subcode $\mathcal{C}(\mathbf{b}_1, \mathbf{b}_2)$, and so forth inductively. This relaxation gives us an analogue of the LLL reduction for linear codes.

Definition 136 (LLL reduced basis). *A basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ of an $[n,k]$-code is said to be* LLL*-reduced if it is a proper basis, and if $\mathbf{b}_i^+$ is a shortest nonzero codeword of the projected subcode $\pi_i(\mathcal{C}(\mathbf{b}_i, \mathbf{b}_{i+1}))$ for all $i \in [\![1, k-1]\!]$.*

Note that a Griesmer-reduced basis is an LLL reduced basis, and the same holds for lattices: an HKZ reduced lattice basis is also LLL reduced. Indeed, if $\mathbf{b}_i^+$ is a shortest codeword of $\pi_i(\mathcal{C}(\mathbf{b}_i, \ldots, \mathbf{b}_n))$ it is also a shortest vector of the subcode $\pi_i(\mathcal{C}(\mathbf{b}_i, \mathbf{b}_{i+1}))$.

Having identified this weaker yet sufficient notion of reduction, we can finalize the proof of Griesmer's bound, in a reduction-theoretic fashion.

Lemma 137. *Let* $[\mathbf{b}_1, \dots, \mathbf{b}_k]$ *be an* LLL*-reduced basis, and let* $\ell_i = |\mathbf{b}_i^+|$ *for* $i \leq k$*. Then we have,*

$$\forall i \in [\![1, k]\!], \quad \ell_{i+1} \geq \left\lceil \frac{\ell_i}{2} \right\rceil .$$

Proof. We start by noting that LLL reduced bases are proper bases by definition, hence every projected subcode $\mathcal{C}_i := \pi_i(\mathcal{C}(\mathbf{b}_i, \mathbf{b}_{i+1})) = \mathcal{C}(\mathbf{b}_i^+, \pi_i(\mathbf{b}_{i+1}))$ has dimension 2 and support size $\ell_i + \ell_{i+1}$.

Let us denote by $\mathbf{x} = \mathbf{b}_i^+$, $\mathbf{y} = \pi_i(\mathbf{b}_{i+1})$ and $\mathbf{z} = \mathbf{y} \oplus \mathbf{x}$ the three nonzero codewords of $\mathcal{C}_i$, and remark that $|\mathbf{x}| = \ell_i$, $|\mathbf{z}| = |\mathbf{x}| + |\mathbf{y}| - 2|\mathbf{x} \wedge \mathbf{y}|$ and $|\mathbf{x} \wedge \mathbf{y}| = |\mathbf{y}| - \ell_{i+1}$. This gives $|\mathbf{x}| + |\mathbf{y}| + |\mathbf{z}| = 2(\ell_i + \ell_{i+1})$, [4] and because $\mathbf{x}$ is the shortest codeword among $\mathbf{x}, \mathbf{y}, \mathbf{z}$, we conclude with:

$$\ell_i = |\mathbf{x}| \leq \tfrac{1}{3}\big(|\mathbf{x}| + |\mathbf{y}| + |\mathbf{z}|\big) \leq \tfrac{2}{3}(\ell_i + \ell_{i+1}).$$

□

We can now reformulate Griesmer bound, while making the underlying reduction notion explicit.

Theorem 138 (Griesmer bound, revisited). *Let* $[\mathbf{b}_1, \dots, \mathbf{b}_k]$ *be a basis of a (linear, binary)* $[n, k]$*-code* $\mathcal{C}$ *that is* LLL*-reduced. Then,*

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{\ell_1}{2^i} \right\rceil , \tag{8.5}$$

where $\ell_1 := |\mathbf{b}_1| \geq d_{\min}(\mathcal{C})$*. In particular, if* $k - 1 \geq \log_2(d_{\min}(\mathcal{C}))$*, it holds that* $\ell_1 - \frac{\lceil \log_2(\ell_1) \rceil}{2} \leq \frac{n-k}{2} + 1$*. Moreover, every binary linear code admits an* LLL*-reduced basis.*

Proof. The inequalities follow from (8.4), while the existence of an LLL reduced basis follows from the fact that Griesmer-reduced bases are LLL reduced. □

[4] Alternatively, one could have invoked the more general fact that the average weights $2^{-k} \sum |\mathbf{c}|$ over a linear code of dimension k is half of its support size $|\mathcal{C}|/2$.

Tightness

A first remark is that the local bound $\ell_{i+1} \geq \left\lceil \frac{\ell_i}{2} \right\rceil$ is tight; it is reached by the $[3, 2]$-code $\mathcal{C} = \{(000), (101), (110), (011)\}$, and more generally by $[n, 2]$-codes for any $n \geq 3$ following a similar pattern. Griesmer's bound is also reached globally and thus we know inputs that give the worst case of our LLL algorithm (which computes efficiently LLL reduced bases of a code), *i.e.* the largest ℓ_1. For instance there are the simplex codes[5] [MS77, Ch.1, §9] and the Reed-Muller codes of order one [Ree53; Mul54] which are respectively $[2^m - 1, m]$-codes and $[2^m, m + 1]$-codes.

Generalisations

The discussion above shows that one can think of Griesmer's bound as an inequality relating codes of dimension k to codes of dimension 2, in the same way that Hermite related lattices of rank n to lattices of rank 2 via Hermite's inequality on the eponymous constants $\gamma_n \leq \gamma_2^{n-1}$.

This type of reasoning can be generalised to relate other quantities. In the literature on lattices, those are known as Mordell's inequalities [GHKN06; GN08a]. These bounds also have underlying algorithms, namely block-reduction algorithm such as BKZ [Sch87] and Slide [GHKN06]. Translating those bounds and their associated algorithms from lattices to codes appears as a very interesting research direction.

Beyond algorithms based on finding shortest vectors of projected sublattices, we also note that some algorithms consider the dense sublattice problem [DM13; LN14]. According to [Bog01], the analogy should be made with the notion of *higher weight* [Wei91; TV95].

Comparison with other code-based bounds

Before presenting our LLL algorithm let us quickly compare in Table 8.2 Griesmer's bound (and thus the bound reached by the LLL algorithm) to classic bounds from coding theory: Singleton's and Hamming's. One can consult [HP10] for their proofs.

An important remark is that until now, only the Singleton bound was algorithmic while Griesmer's bound was seen as an extension of

[5] The simplex code is defined as the dual of the Hamming code.

Bound	Concrete	Asymptotic	Poly. Algo.
Singleton's	$d \leq n - k + 1$	$\delta \leq 1 - R$	YES
Hamming's	$2^k \sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i} \leq 2^n$	$R \leq 1 - h\left(\frac{\delta}{2}\right)$	NO
Griesmer's	$d - \frac{\log_2 d}{2} \leq \frac{n-k}{2} + 1$	$\delta \leq \frac{1-R}{2}$	**Now, YES**

Table 8.2: Bounds on $d = d_{\min}(\mathcal{C})$. Asymptotic form is given for a fixed rate $R = k/n \in [0, 1]$, $\delta := d/n$ and $n \to \infty$. In Hamming's constant, $h(x) := -x \log_2(x) - (1-x) \log_2(1-x)$ denotes the so-called binary entropy of x.

it but not algorithmic. For an $[n, k]$-code, Singleton's bound states that $d_{\min}(\mathcal{C}) \leq n - k + 1$. The underlying algorithm of this bound simply consists in putting the basis in systematic form, namely to reduce in row echelon form the basis, to get a short codeword. It may be argued that for random codes this bound is far from tight. Indeed, the systematic form in fact produces codewords of average length $\frac{n-k}{2} + 1$ (this is exactly what Prange algorithm [Pra62] does). While this seems better than Griesmer's bound, the LLL algorithm gives a codeword of length at most $\frac{n-k}{2} + \log_2 n$ but in the *worst-case*.

This concludes the translation of all the notions at hands from lattices to codes. We summarize them as a dictionary in Table 8.1.

8.5.2 An LLL reduction algorithm for codes

In the above subsection, we have defined LLL reduced bases and have shown that they exist by constructing a basis with an even stronger reduction property. However, such a construction requires to solve the shortest codeword problem, a problem known to be NP-hard [Var97], and the best known algorithm have exponential running time in n or in k, at least for constant rates $R = k/n$.

In other words, we have shown existence of LLL reduced bases (*local minimality*) by a *global minimality* argument, which would translate into an algorithm with exponential running time. Instead, we can show their existence by a *descent* argument, and this proof translates to a polynomial time algorithm, the *LLL algorithm for binary codes*.

The strategy to produce LLL reduced bases is very simple, and essentially the same as in the case of lattices [LLL82]: if $\pi_i(\mathbf{b}_i)$ is not a shortest nonzero codeword of $\pi_i(\mathcal{C}(\mathbf{b}_i, \mathbf{b}_{i+1}))$ for some i, then apply a change of basis on $\mathbf{b}_i, \mathbf{b}_{i+1}$ so that it is. Such a transformation may break the same property for nearby indices $i-1$ and $i+1$, however, we will show that, overall, the algorithm still makes progress.

There are two technical complications of the original LLL [LLL82] that can be removed in the case of codes. The first is that we do not need a global size-reduction on the basis; this step of LLL does not affect the Gram-Schmidt vectors themselves, but is needed for numerical stability issues, which do not arise over the finite field $\mathbb{F}_2$. Secondly, we do not need to introduce a small approximation term $\varepsilon > 0$ to prove that the algorithm terminates in polynomial time, thanks to the discreteness of epipodal lengths.

Algorithm 14: LLL($\mathbf{B}$) — LLL reduce the basis $\mathbf{B}$

Input : A proper basis $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_k] \in \mathbb{F}_2^{k \times n}$ of a code $\mathcal{C}$
Output: An LLL reduced basis for $\mathcal{C}$

while $\exists i \in [\![0, k-1]\!]$ *s.t.* $\min\left(\left|\pi_i(\mathbf{b}_{i+1})\right|, \left|\mathbf{b}_i^+ \oplus \pi_i(\mathbf{b}_{i+1})\right|\right) < \left|\mathbf{b}_i^+\right|$ **do**
  if $\left|\pi_i(\mathbf{b}_{i+1}) \wedge \mathbf{b}_i^+\right| + \mathrm{TB}_{\mathbf{b}_i^+}\left(\pi_i(\mathbf{b}_{i+1})\right) > \left|\mathbf{b}_i^+\right|/2$ **then**
    $\mathbf{b}_{i+1} \leftarrow \mathbf{b}_{i+1} \oplus \mathbf{b}_i$ `// Local size-reduction`
  end
  $\mathbf{b}_i \leftrightarrow \mathbf{b}_{i+1}$ `// Swap`
end
return B

Theorem 139. *Algorithm 14 is correct and its running time is polynomial; more precisely on input an $[n,k]$-code it performs at most kn vector operations over $\mathbb{F}_2^n$.*

It will be a consequence of the two following lemmata. Let us start with correctness.

Lemma 140 (Correctness). *If the* LLL *algorithm terminates then it outputs an* LLL*-reduced basis for the code spanned by the basis given as input.*

Proof. Note that vectors $\mathbf{b}_i$'s form a basis of the code $\mathcal{C}$ given as input and this is a loop invariant. The exit condition ensures that the basis is indeed LLL reduced, at least if it is proper. So it suffices to show that properness is also a loop invariant.

Assume that the basis is proper ($\ell_j \geq 1$ for all j) as we enter the loop at index i. The local size-reduction step does not affect epipodal lengths. The swap only affects ℓ_i and ℓ_{i+1}, and leaves $\ell_i + \ell_{i+1}$ unchanged. The epipodal length ℓ_i decreases, but remains nonzero since $\mathbf{b}_i^+$ is a shortest-codeword of a 2-dimensional code by construction. The epipodal length ℓ_{i+1} can only increase. □

The key-ingredient to prove that LLL terminates lies in the construction of a potential: a quantity that only decreases during the algorithm, and is lower bounded.

Definition 141 (Potential). *Let* $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ *be a basis of an* $[n, k]$*-code. The potential of* $\mathbf{B}$*, denoted* $\mathcal{D}_\mathbf{B}$*, is defined as:*

$$\mathcal{D}_\mathbf{B} := \sum_{i=1}^{k} (k - i + 1)\ell_i = \sum_{i=1}^{k} \left(\sum_{j=1}^{i} \ell_j \right) = \sum_{i=1}^{k} \mathcal{D}_{\mathbf{B},i},$$

where $\mathcal{D}_{\mathbf{B},i} := |\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_i)|$ *and* $\ell_i := |\mathbf{b}_i^+|$.

Each time LLL makes a swap, the potential decreases at least by one as shown in the proof of the following lemma. Furthermore, this quantity is always positive. Therefore the number of iterations is upper-bounded by the initial value of $\mathcal{D}_\mathbf{B}$.

Lemma 142 (Termination). *The number of iterations in Algorithm 14 on input* $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ *is upper bounded by,*

$$\min \left\{ kn, \frac{k(k+1)}{2} \max_i |\mathbf{b}_i^+| \right\}.$$

Proof. The potential $\mathcal{D}_\mathbf{B}$ only changes during the swap step. Let us show that it decreases by at least one at each swap step. Suppose there is a swap at the index i. Let $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ be the values of the basis vectors after the **if** loop and just before the swap step. Here, the **if** loop ensures:

$$|\mathbf{b}_i^+ \wedge \pi_i(\mathbf{b}_{i+1})| \leq \frac{|\mathbf{b}_i^+|}{2}. \tag{8.6}$$

Now, whether or not the **if** loop was executed we have that:

$$\min\left(\left|\pi_i(\mathbf{b}_{i+1})\right|, \left|\mathbf{b}_i^+ \oplus \pi_i(\mathbf{b}_{i+1})\right|\right) < \left|\mathbf{b}_i^+\right|. \tag{8.7}$$

Therefore, combining Equations (8.6) and (8.7) with $\left|\mathbf{b}_i^+ \oplus \pi_i(\mathbf{b}_{i+1})\right| = |\mathbf{b}_i^+| + |\pi_i(\mathbf{b}_{i+1})| - 2|\mathbf{b}_i^+ \wedge \pi_i(\mathbf{b}_{i+1})|$ leads to:

$$|\pi_i(\mathbf{b}_{i+1})| < |\mathbf{b}_i^+|. \tag{8.8}$$

Now during the **while** execution only $\mathcal{D}_{\mathbf{B},i}$ is modified. Let $\mathcal{C}_i'$ be the new partial code $\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_i)$ after the swap, and $\mathcal{D}'_{\mathbf{B},i}$ be the new values of $\mathcal{D}_{\mathbf{B},i}$. We have,

$$\begin{aligned}
\mathcal{D}'_{\mathbf{B},i} - \mathcal{D}_{\mathbf{B},i} &= |\mathcal{C}_i'| - |\mathcal{C}_i| \\
&= \sum_{j=1}^{i-1} |\mathbf{b}_j^+| + |\pi_i(\mathbf{b}_{i+1})| - \sum_{j=1}^{i-1} |\mathbf{b}_j^+| - |\mathbf{b}_i^+| \\
&= |\pi_i(\mathbf{b}_{i+1})| - |\mathbf{b}_i^+| < 0
\end{aligned}$$

where for the inequality we used Equation (8.8). Potentials $\mathcal{D}_{\mathbf{B},i}$ are integers. Therefore at each iteration $\mathcal{D}_{\mathbf{B}}$ decreases by at least one. Let $\mathcal{D}_{\mathbf{B}}^{(0)}$ be the initial value of the potential $\mathcal{D}_{\mathbf{B}}$. We conclude with:

$$\mathcal{D}_{\mathbf{B}}^{(0)} = \sum_{i=1}^{k} (k-i+1)|\mathbf{b}_i^+| \leq \min\left\{kn, \frac{k(k+1)}{2} \max_i |\mathbf{b}_i^+|\right\},$$

where for the first bound we use that $\sum_i |\mathbf{b}_i^+| = |\mathcal{C}| \leq n$.

□

8.6 Perspectives

This work brings codes and lattices closer to each other by enriching the existing dictionary (Table 8.1); we hope that it can enable more transfer of techniques between those two research areas, and list some research directions.

Generalisations

In principle, the definitions, theorems and algorithms of this article should be generalizable to codes over $\mathbb{F}_q$ endowed with the Hamming

metric, with the minor inconvenience that one may no longer conflate words over $\mathbb{F}_q$ with their binary support vector. Some algorithms may see their complexity grow by a factor $\Theta(q)$, meaning that the algorithms remain polynomial-time only for $q = n^{O(1)}$. It is natural to hope that such a generalised LLL would still match Griesmer [Gri60] bound for $q > 2$. However, we expect that the analysis of the fundamental domain of Section 8.4 would become significantly harder to carry out.

Another natural generalisation to aim for would be codes constructed with a different metric, in particular codes endowed with the rank metric [Gab85; Gab95]. In this case codes are subspaces of $\mathbb{F}_{q^m}$ endowed with the rank metric; the weight of a codeword $\mathbf{x} \in \mathbb{F}_{q^m}^n$ is the rank of its matrix representation over $\mathbb{F}_q$ (which is a matrix of size $m \times n$). While the support of a codeword with the Hamming metric is the set of its nonzero coordinates, the support of $\mathbf{x} = (x_1, \cdots, x_n) \in \mathbb{F}_{q^m}^n$ is the $\mathbb{F}_q$-subspace of $\mathbb{F}_{q^m}$ that the x_i's generate, namely $\{\sum_i \lambda_i x_i : \lambda_i \in \mathbb{F}_q\}$. We believe that this work can be generalised in this case, in particular the notion of epipodal matrices. However there are some difficulties to overcome. In particular, projecting one support orthopodally to another one is not canonical.

Cryptanalysis

In the full version of this work we introduce a hybrid (LeeBrickellBabai) of the Lee-Brickell information set decoding algorithm [LB88], and the size-reduction algorithm (after LLL). We show heuristically that for common decoding parameters this leads to a small polynomial (in n) speed-up over Lee-Brickell. This contribution is meant to show that this algorithmic reduction theory for codes is compatible with existing techniques, and can, in principle bring improvements. By itself, this hybrid LeeBrickellBabai algorithm with LLL preprocessing is only tenuously faster than the original algorithm. Time-memory trade-offs such as [Ste88; Dum91; MMT11; BJMM12; MO15; BM18] admittedly provide much more substantial speed-ups in theory, and currently hold the records in practice [ALL19].

However, the lattice literature has much stronger reduction algorithms to offer than LLL [Sch87; GHKN06; GN08a; LN14; DM13]; this work opens their adaptation to codes as a new research area, together

with the study of their cryptanalytic implications. Furthermore, it is not implausible that reduction techniques may be compatible with memory intensive techniques [Ste88; Dum91; MO15]; this is the case in the lattice cryptanalysis literature [Duc18].

Further algorithmic translations

Still based around the same fundamental domain, a central algorithm for lattices is the Branch-and-Bound enumeration algorithm of Finkle and Pohst [FP85], which has been the object of numerous variations for heuristic speed-ups [GNR10]. While the hybrid algorithm LeeBrickellBabai may be read as an analogue of the random sampling algorithm of Schnorr [Sch03; AN17], a more general study of enumeration techniques for codes would be interesting.

Both for codes and lattices, there are other natural fundamental domains than the size-reduction studied in this paper. For lattices we have the (non-rectangle) parallelepiped $\mathcal{P}(\mathbf{B})$ provided with the so-called "simple rounding" algorithm $\mathbf{x} \mapsto \mathbf{x} - \lfloor \mathbf{x} \cdot \mathbf{B}^{-1} \rceil \cdot \mathbf{B}$ [Bab86]. For codes we have a domain of the form $\mathbb{F}_2^{n-k} \times \{0\}^k$ for each information set $\mathcal{I} \subset [\![1, n]\!]$ of size k given by Prange's algorithm [Pra62]. It is tempting to think they could be in correspondence in a unified theory for codes and lattices.

Another fundamental domain of interest is the Voronoi domain, which is naturally defined for both codes and lattices. In the case of lattices there are algorithms associated with it known as iterative slicers. Rather than operating with a basis, the provable versions of this algorithm operates with the (exponentially large) set of *Voronoi-relevant* vectors [MV13], while heuristic variants can work with a (smaller, but still exponential) set of short vectors (see chapter 5). We are not aware of similar approaches in the code literature.

Cryptographic design

Some of the developed notions could have application in cryptographic constructions as well, in particular for trapdoor sampling [GPV08; DST19]. Indeed, the Gaussian sampling algorithm of [GPV08] is merely a careful randomisation of the size-reduction algorithm, and the variant of Peikert [Pei10] is a randomisation of the "simple rounding" algorithm discussed above. It requires knowing a basis with a

good profile as a trapdoor. While the construction of a sampleable trapdoor function has finally been realised [DST19], the method and underlying problem used are rather ad-hoc. We note in particular that the underlying generalized $(U, U+V)$-codes admit bases with a peculiar profile, which may explain their fitness for trapdoor sampling. The algorithmic reduction theory proposed in this work appears as the natural point of view to approach and improve trapdoor sampling for codes.

Bounds

Beyond cryptography, this reduction theory may be of interest to establish new bounds for codes. In particular we emphasize the notion of *higher weight* [Wei91; TV95] as an analogue of the notion of the density of sub-lattices [Bog01]; the latter are subject to the so-called Rankin-bound, generalizing Hermite's bound on the minimal distance of a lattice.

Duality

Also on a theoretical level, one intriguing question is how this reduction theory interacts with the notion of duality for codes. In particular, for lattices, the dual of an LLL reduced basis of the primal lattice is (essentially) an LLL reduced basis of the dual lattice. One could wonder whether this also holds for codes; however, while there is a notion of a dual code, their doesn't seem to be a 1-to-1 correspondence between primal and dual bases. The notion of orthopodality does allow to introduce (non-unique) dual bases for codes, with similar geometric properties as the (reversed) dual basis. For lattices this leads to a correpondence $\|\tilde{\mathbf{b}}_i\| = \|\tilde{\mathbf{d}}_i\|^{-1}$; for codes the similar concept seems to be the correspondence between the $[|\mathbf{b}_i^+|, 1]$ repetition code generated by $\mathbf{b}_i^+$, and the $[|\mathbf{b}_i^+|, |\mathbf{b}_i^+| - 1]$ even code generated by some $\mathbf{d}_{j_1}^+, \ldots, \mathbf{d}_{j_{|\mathbf{b}_i^+|-1}}^+$. The construction allows for some choices, such as which basis to use for the even code. Under some light conditions this gives for $\ell_i \geq 2$ a primal-dual profile correspondence of $(\ell_i) \leftrightarrow (2, 1, \ldots, 1)$ with $\ell_i - 2$ ones. We leave a further investigation of the concept of a dual basis, and self-duality of the presented LLL algorithm to future work.

Another remark is that it may also be natural to consider Branch-and-Bound enumeration algorithms working with a reduced basis of the dual code rather than a basis of the primal code, at least if self-duality can not be established. This may be advantageous in certain regimes.

Part IV

The Lattice Isomorphism Problem, Remarkable Lattices and Cryptography

CHAPTER 9

The Lattice Isomorphism Problem

This chapter gives an introduction to the Lattice Isomorphism Problem. It contains results from the joint work 'A canonical form for positive definite matrices', with Mathieu Dutour Sikirić, Anna Haensch, John Voight, published at ANTS 2020. In addition it shows direct applications of this work to ongoing joint work on Perfect form and C-type enumeration, with Mathieu Dutour Sikirić.

9.1 Introduction

When are two lattices the same? Two distinct bases $\mathbf{B}, \mathbf{B}' \in \mathcal{GL}_n(\mathbb{R})$ can generate exactly the same lattice $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}')$, and they do if and only if $\mathbf{B}' = \mathbf{B} \cdot \mathbf{U}$ for some unimodular matrix $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$. What about a lattice $\mathcal{L}$ and a rotation $\mathcal{L}'$ of the same lattice? These lattices are not (necessarily) the same as sets, but they do carry the same geometric information. We call such lattices isomorphic. More precisely we say that a lattice $\mathcal{L}$ and a lattice $\mathcal{L}'$ are *isomorphic* if

there is an orthonormal transformation $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$, i.e., an isometry, such that $\mathcal{L}' = \mathbf{O} \cdot \mathcal{L} = \{\mathbf{Ov} : \mathbf{v} \in \mathcal{L}\}$.

The study of isomorphisms between lattices is as old as the use of lattices, for example by Gauss, Lagrange, Minkowski and Voronoi from the 18th century to the beginning of the 20th century. Isomorphisms naturally arise in the search for interesting lattices[1], as a lattice is only really 'new' if it is not already isomorphic to a known one. Furthermore, isomorphisms between a lattice and itself, enable a study of the symmetries of a lattice.

Computing an isometry $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$ between two isomorphic lattices, known as the Lattice Isomorphism Problem (LIP), or even deciding if two lattices are isomorphic, is seemingly a hard problem. Almost all algorithms require as a first step the enumeration of (many) short lattice vectors, which in itself is a hard problem. This includes the best proven algorithm, that runs in time $n^{O(n)}$ [HR14], as well as the best practical algorithms [PP85; PS97]. Moreover among the interesting lattices there are many with a high kissing number, and LIP seems even harder for those. For practical computations we are thus restricted to somewhat low dimensions, and this chapter must be considered in this context. For higher dimensions the hardness of LIP directly motivates its use in cryptography, and in Chapter 10 we show how to leverage LIP to construct a new type of lattice-based cryptography.

In this chapter we give an overview of the literature on LIP. First we show that this problem is more natural to state in the quadratic form setting. Next, we discuss the presence of natural invariants, that can be used to answer LIP in the negative, or guide the search for an isomorphism. We show how so-called characteristic sets play a fundamental role in solving LIP, and even in constructing a canonical representative inside an isomorphism class. We end with some interesting applications of this canonical representative, such as computationally solving the lattice packing problem by perfect form enumeration.

[1] Examples are lattices that are good packings, coverings or have a high kissing number, perfect lattices or those arising from number theory.

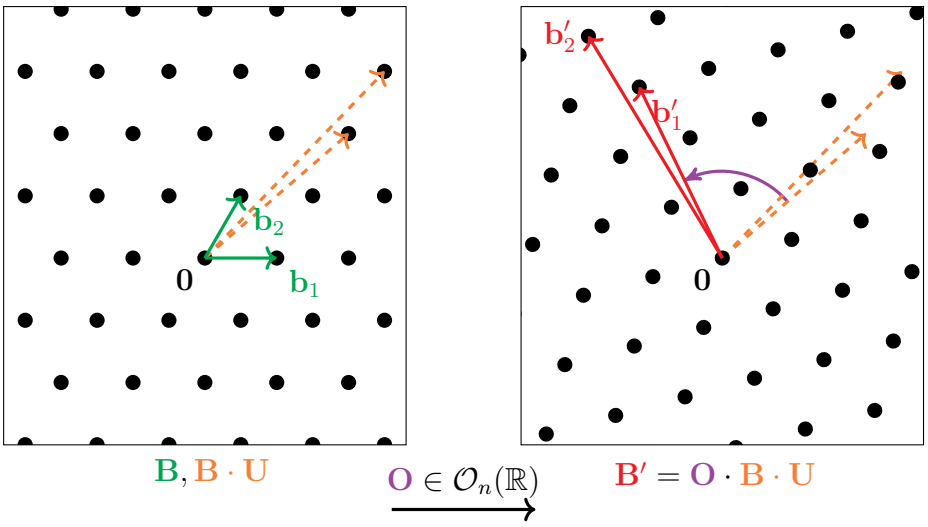


Figure 9.1: Illustration of two isomorphic lattices.

9.2 LIP & quadratic forms

Abstractly, the set of full rank, n-dimensional lattices can be thought of as the homogeneous space[2] $\mathcal{GL}_n(\mathbb{R})/\mathcal{GL}_n(\mathbb{Z})$: two bases $\mathbf{B}, \mathbf{B}' \in \mathcal{GL}_n(\mathbb{R})$ generate the same lattice if and only if there exists a unimodular matrix $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{B}' = \mathbf{BU}$. Two lattices $\mathcal{L}, \mathcal{L}'$ are isomorphic if and only if there exists an orthonormal transformation $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$ such that $\mathcal{L}' = \mathbf{O} \cdot \mathcal{L}$. The set of lattices up to isomorphism can thus be thought of as the double quotient

$$\mathcal{O}_n(\mathbb{R}) \backslash \mathcal{GL}_n(\mathbb{R}) / \mathcal{GL}_n(\mathbb{Z}).$$

Reconstructing equivalence in this double quotient is known as the Lattice Isomorphism Problem (LIP).

Definition 143 (LIP, lattice version). *Given bases $\mathbf{B}, \mathbf{B}' \subset \mathbb{R}^n$ of two isomorphic lattices, compute an orthonormal transformation $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$, and an unimodular transformation $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{B}' = \mathbf{OBU}$.*

[2] This quotient should read as the quotient of a *set* by the action of group, and not a group quotient. Indeed $\mathcal{GL}_n(\mathbb{Z})$ is *not* a normal subgroup of $\mathcal{GL}_n(\mathbb{R})$ for $n > 1$.

If either $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$ or $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ is known, recovering the other reduces to a matrix inversion. The presence of *both* the unknown orthonormal and the unknown unimodular transformation is what makes LIP seemingly a hard problem. In other words, reconstructing (or even testing) equivalence in either quotient $\mathcal{GL}_n(\mathbb{R})/\mathcal{GL}_n(\mathbb{Z})$ or $\mathcal{O}_n(\mathbb{R})\backslash\mathcal{GL}_n(\mathbb{R})$ is easy, doing so in the double quotient appears to be hard.

The real-valued coordinates of the basis and orthonormal transformation can be inconvenient and inefficient to work with. We can alleviate some of these concerns by moving to the (equivalent) quadratic form setting, where instead of a basis $\mathbf{B}$ we focus on the Gram matrix $\mathbf{Q} = \mathbf{B}^\top\mathbf{B} = (\langle \mathbf{b}_i, \mathbf{b}_j \rangle)_{i,j}$.

Quadratic forms and arithmetical equivalence

The idea of the Quadratic Form point of view on LIP is to consider the quotient in the opposite order than in the lattice point of view: first on the left by $\mathcal{O}_n(\mathbb{R})$ and then only on the right by $\mathcal{GL}_n(\mathbb{Z})$.

Definition 144 (Quadratic Form). *A quadratic form of rank n is a positive definite real*[3] *symmetric matrix $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$.*

Quadratic forms can be thought of as bases modulo rotation; they realize the quotient $\mathcal{O}_n(\mathbb{R})\backslash\mathcal{GL}_n(\mathbb{R})$. More precisely, consider the surjective Gram map $\gamma : \mathcal{GL}_n(\mathbb{R}) \to \mathcal{S}_n^{>0}(\mathbb{R})$ sending a lattice basis $\mathbf{B}$ to the quadratic form $\mathbf{Q} = \mathbf{B}^\top\mathbf{B}$. The preimages of $\gamma(\mathbf{B})$ are precisely the $\mathbf{OB}$ for $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$.

Lemma 145. *The Gram map $\gamma : \mathcal{GL}_n(\mathbb{R}) \to \mathcal{S}_n^{>0}(\mathbb{R})$ induces a bijection*

$$\begin{aligned} \mathcal{O}_n(\mathbb{R})\backslash\mathcal{GL}_n(\mathbb{R}) &\longleftrightarrow \mathcal{S}_n^{>0}(\mathbb{R}), \\ \mathcal{O}_n(\mathbb{R}) \cdot \mathbf{B} &\longmapsto \mathbf{B}^\top\mathbf{B}. \end{aligned}$$

Proof. Because $(\mathbf{OB})^\top(\mathbf{OB}) = \mathbf{B}^\top\mathbf{O}^\top\mathbf{OB} = \mathbf{B}^\top\mathbf{B}$ the map is well defined. The matrix $\mathbf{B}^\top\mathbf{B}$ is symmetric and positive definite as for any

[3] In constrast to the standard definition, our quadratic form is always real and positive definite. Furthermore we define the quadratic form as a symmetric matrix, and not by the associated degree two polynomial $f(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top\mathbf{Q}\mathbf{y}$.

nonzero $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ the vector $\mathbf{Bx}$ is nonzero, and thus $\mathbf{x}^\top(\mathbf{B}^\top\mathbf{B})\mathbf{x} = \|\mathbf{Bx}\|^2 > 0$. The map is injective because if for any $\mathbf{A}, \mathbf{B} \in \mathcal{GL}_n(\mathbb{R})$ we have $\mathbf{A}^\top\mathbf{A} = \mathbf{B}^\top\mathbf{B}$, then $(\mathbf{BA}^{-1})^\top(\mathbf{BA}^{-1}) = \mathbf{I}_n$ and thus $\mathbf{BA}^{-1} \in \mathcal{O}_n(\mathbb{R})$. The map is surjective because for every quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$ the Cholesky decomposition gives a unique upper-triangular lattice basis $\mathbf{B_Q}$ with positive diagonal such that $\mathbf{Q} = \mathbf{B_Q^\top B_Q}$. □

For a lattice basis $\mathbf{B}$ the Gram matrix $\mathbf{Q} = \mathbf{B}^\top\mathbf{B}$ naturally gives a quadratic form. We can now translate the notions we have for lattices to those of quadratic forms. In the quadratic form setting lattice vectors $\mathbf{Bx} \in \mathbb{R}^n$ are represented by their integral basis coefficients $\mathbf{x} \in \mathbb{Z}^n$. The inner product with respect to a quadratic form is naturally given by $\langle \mathbf{x}, \mathbf{y} \rangle_\mathbf{Q} := \mathbf{x}^\top\mathbf{Qy}$, and the norm by $\|\mathbf{x}\|_\mathbf{Q}^2 := \mathbf{x}^\top\mathbf{Qx}$. Note that this perfectly coincides with the geometry between the original lattice vectors. We denote the $\mathbf{Q}$-ball of radius r by $\mathcal{B}_\mathbf{Q}(r) := \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_\mathbf{Q} \leq r\}$. Translating the lattice definition, one gets the *first minimum*[4] $\lambda_1(\mathbf{Q})$ defined by

$$\lambda_1(\mathbf{Q}) := \min_{\mathbf{x} \in \mathbb{Z}^n \setminus \{0\}} \|\mathbf{x}\|_\mathbf{Q},$$

and more generally the i-th succesive minimum $\lambda_i(\mathbf{Q})$ defined as the smallest $r > 0$ such that $\mathbb{Z}^n \cap \mathcal{B}_\mathbf{Q}(r)$ spans a space of dimension at least i.

In this realization $\mathcal{S}_n^{>0}(\mathbb{R})$ of the quotient $\mathcal{O}_n(\mathbb{R})\backslash\mathcal{GL}_n(\mathbb{R})$, the (right) action of $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ is given by $\mathbf{Q} \mapsto \mathbf{U}^\top\mathbf{QU}$. We may now rephrase lattice isomorphisms in terms of quadratic forms, moving the real-valued orthonormal transform $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$ out of the picture.

Definition 146 (Arithmetical Equivalence). *Two quadratic forms $\mathbf{Q}, \mathbf{Q}' \in \mathcal{S}_n^{>0}(\mathbb{R})$ are called equivalent, denoted $\mathbf{Q} \cong \mathbf{Q}'$, if there exists a unimodular $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}' = \mathbf{U}^\top\mathbf{QU}$.*

For two equivalent forms $\mathbf{Q}, \mathbf{Q}'$ we denote the set of all such unimodular transformations by $\mathrm{Isom}(\mathbf{Q}, \mathbf{Q}') := \{\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z}) : \mathbf{Q}' = \mathbf{U}^\top\mathbf{QU}\}$.

Lemma 147. *Two lattice bases $\mathbf{B}, \mathbf{B}'$ are isomorphic if and only if their Gram matrices are equivalent.*

[4] In the setting of quadratic forms the first minimum often refers to the squared norm, i.e., the value $\lambda_1(\mathbf{Q})^2$. We chose to stay in line with the lattice definition.

Proof. Let $\mathbf{B}, \mathbf{B}' \in \mathcal{GL}_n(\mathbb{R})$ be isomorphic bases, i.e., there exists an $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$, and $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{B}' = \mathbf{OBU}$, then for $\mathbf{Q}' = \mathbf{B}'^\top \mathbf{B}'$ and $\mathbf{Q} := \mathbf{B}^\top \mathbf{B}$ we have:

$$\mathbf{Q}' := (\mathbf{B}')^\top \mathbf{B}' = \mathbf{U}^\top \mathbf{B}^\top \mathbf{O}^\top \mathbf{OBU} = \mathbf{U}^\top \mathbf{B}^\top \mathbf{BU} = \mathbf{U}^\top \mathbf{QU},$$

and thus $\mathbf{Q}'$ and $\mathbf{Q}$ are equivalent.

For the other direction, assume that $\mathbf{Q}' = \mathbf{U}^\top \mathbf{QU}$ for some $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$. This implies that $(\mathbf{B}')^\top \mathbf{B}' = (\mathbf{BU})^\top (\mathbf{BU})$, and thus by Lemma 145 there exists some $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$ such that $\mathbf{B}' = \mathbf{OBU}$ and thus $\mathbf{B}$ and $\mathbf{B}'$ are isomorphic. □

We denote the equivalence class, or orbit, of a quadratic form $\mathbf{Q}$ by $[\mathbf{Q}] := \{\mathbf{U}^\top \mathbf{QU} : \mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})\}$. Many remarkable lattices attain a rational-valued Gram matrix $\mathbf{Q}$, removing the need for real-valued or approximate arithmetic. We will often restrict ourself to the set of integer-valued (positive-definite) quadratic forms denoted by $\mathcal{S}_n^{>0}(\mathbb{Z})$.

Remark 148 (Isometry). Generally the term *isometry* is used for a distance preserving bijective map between normed vector spaces, and *linear isometry* for norm preserving bijective linear maps. In this thesis we simply refer to *isometry* for any bijective inner product preserving map between (subsets of) inner product spaces. In particular for two equivalent forms $\mathbf{Q}, \mathbf{Q}' \in \mathcal{S}_n^{>0}(\mathbb{R})$, any unimodular $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}' = \mathbf{U}^\top \mathbf{QU}$ corresponds to an isometry $\mathbf{x} \mapsto \mathbf{U}^{-1}\mathbf{x}$ between $\mathbb{Z}^n \subset \mathbb{R}^n$ w.r.t. (the inner product given by) $\mathbf{Q}$ and $\mathbb{Z}^n \subset \mathbb{R}^n$ w.r.t. $\mathbf{Q}'$.

The Lattice Isomorphism Problem, quadratic form formulation

The Lattice Isomorphism Problem can now be restated. We start by properly defining the worst-case problems, in both a search and distinguishing variant.

Definition 149 (Worst-case search LIP: wc-sLIP$^{\mathbf{Q}}$). *For a quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}$ the problem* wc-sLIP^{Q} *is, given any quadratic form $\mathbf{Q}' \in [\mathbf{Q}]$, to compute a unimodular $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}' = \mathbf{U}^\top \mathbf{QU}$.*

Note that the problem is equivalent to the original LIP problem as we can still extract an orthonormal transformation by computing $\mathbf{O} = \mathbf{B}'(\mathbf{BU})^{-1}$. We also consider a *distinguishing* variant of LIP, denoted wc-ΔLIP. It is not to be confused with the *decisional* version of LIP (which we will refer to as dLIP).[5]

Definition 150 (Worst-case distinguishing LIP: wc-$\Delta\mathrm{LIP}^{\mathbf{Q}_0,\mathbf{Q}_1}$). *For two quadratic forms $\mathbf{Q}_0, \mathbf{Q}_1 \in \mathcal{S}_n^{>0}$ the problem* wc-$\Delta\mathrm{LIP}^{\mathbf{Q}_0,\mathbf{Q}_1}$ *is, given any quadratic form $\mathbf{Q}' \in [\mathbf{Q}_b]$, where $b \in \{0,1\}$ is a uniform random bit, to recover b*[6].

The distinguishing problem wc-$\Delta\mathrm{LIP}^{\mathbf{Q}_0,\mathbf{Q}_1}$ is worst-case w.r.t. the choice of $\mathbf{Q}'$, but also the fact that the challenge $b \in \{0,1\}$ is sampled uniformly is worst-case: any additional bias would only make the problem easier for an attacker.

Automorphisms

Isomorphisms between a lattice and itself are called automorphisms. They represent the internal symmetry of the lattice. In the quadratic form setting, these are exactly the unimodular transformations that act invariantly, i.e., thet form the stabilizer group of $\mathbf{Q}$ under the action of $\mathcal{GL}_n(\mathbb{Z})$. This group is called the automorphism group of $\mathbf{Q}$.

Definition 151 (Automorphisms). *For a quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$, the automorphism group* $\mathrm{Aut}(\mathbf{Q})$ *of $\mathbf{Q}$ is defined by*

$$\mathrm{Aut}(\mathbf{Q}) := \{\mathbf{V} \in \mathcal{GL}_n(\mathbb{Z}) : \mathbf{V}^\top \mathbf{Q} \mathbf{V} = \mathbf{Q}\}.$$

The automorphism group is finite, as will become clear in Section 9.5. For any $\mathbf{Q}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U} \in [\mathbf{Q}]$ we have the natural stabilizer conjugacy $\mathrm{Aut}(\mathbf{Q}') = \mathbf{U}^{-1} \mathrm{Aut}(\mathbf{Q}) \mathbf{U}$. Additionally, we obtain a bijection between the right cosets $\mathrm{Aut}(\mathbf{Q}) \setminus \mathcal{GL}_n(\mathbb{Z})$ and the orbit $[\mathbf{Q}]$, and thus

[5] In $\mathrm{dLIP}^{\mathbf{Q}_0}$ one is given an arbitrary $\mathbf{Q}'$ and must decide whether $\mathbf{Q}'$ belongs to $[\mathbf{Q}_0]$. The distinguishing version is potentially easier in that $\mathbf{Q}'$ is promised to belong to either $[\mathbf{Q}_0]$ or $[\mathbf{Q}_1]$ for some known fixed $[\mathbf{Q}_1]$.

[6] The distinguishing problem should be interpreted as a game: a challenger samples $b \in \{0,1\}$ uniformly and responds with any $\mathbf{Q}' \in [\mathbf{Q}_b]$, the prover should respond with a bit b' and wins if $b = b'$. The prover solves the problem if it can win with non-negligible advantage (over the trivial win probability of 0.5).

for any isomorphism $\mathbf{U} \in \mathrm{Isom}(\mathbf{Q}, \mathbf{Q}')$, the full set of isomorphisms is given by $\mathrm{Isom}(\mathbf{Q}, \mathbf{Q}') = \mathrm{Aut}(\mathbf{Q}) \cdot \mathbf{U}$.

9.3 Invariants

Equivalent quadratic forms $\mathbf{Q}, \mathbf{Q}' := \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ (for some $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$) share many common properties, and these invariants can be used to decide that two quadratic forms cannot be equivalent, or can guide the search for an isomorphism. We start with some arithmetic invariants that are efficiently computable, and then discuss some geometric invariants which are only efficiently computable in low dimensions. We restrict ourself to integral quadratic forms $\mathbf{Q}, \mathbf{Q}' \in \mathcal{S}_n^{>0}(\mathbb{Z})$, as those are common in practice and allow for the most invariants.

9.3.1 Arithmetic invariants

Firstly we have $\det(\mathbf{U}) = \pm 1$, and thus for two equivalent quadratic forms we have

$$\det(\mathbf{Q}') = \det(\mathbf{U}^\top) \det(\mathbf{Q}) \det(\mathbf{U}) = \det(\mathbf{Q}).$$

Secondly because $\mathbf{U}$ and $\mathbf{U}^{-1}$ are both integral, the quantity $\gcd(\mathbf{Q}) := \gcd\{\mathbf{Q}_{ij} : 1 \le i, j \le n\}$ also gives an invariant.

A third and less obvious invariant is the parity of the quadratic form. The notion is standard for unimodular lattices: it is called even if all norms are even, and odd otherwise. More generally, writing $\|\mathbf{x}\|_{\mathbf{Q}}^2 = \sum_i \mathbf{Q}_{ii} x_i^2 + 2 \sum_{i<j} x_j \mathbf{Q}_{ij} x_i$ one gets that $\gcd\{\|\mathbf{x}\|_{\mathbf{Q}}^2 : x \in \mathbb{Z}^n\} \in \{1, 2\} \cdot \gcd(\mathbf{Q})$. We call this factor $\mathrm{par}(\mathbf{Q}) \in \{1, 2\}$ the parity of $\mathbf{Q}$. It is also efficiently computable by noting that $\mathrm{par}(\mathbf{Q}) = \gcd(\{\mathbf{Q}_{ii} : 1 \le i \le n\} \cup \{2 \gcd(\mathbf{Q})\}) / \gcd(\mathbf{Q})$.

These invariants are all a bit ad hoc, and possibly incomplete. All these invariants (and more) are captured by the following.

Weaker equivalence (genus)

The study of integral equivalence of quadratic forms is classically approached via weaker notions, namely, equivalence over larger rings [CS13, Ch. 15, Sec. 4] $R \supset \mathbb{Z}$. In particular we consider $R =$

$\mathbb{R}, \mathbb{Q}, \mathbb{Q}_p, \mathbb{Z}_p$, where $\mathbb{Q}_p$ denotes the rational and $\mathbb{Z}_p$ the integer p-adic numbers for a prime p. For any ring R we denote the equivalence class of a quadratic form $\mathbf{Q}$ by $[\mathbf{Q}]_R := \{\mathbf{U}^\top \mathbf{Q}\mathbf{U} : \mathbf{U} \in \mathcal{GL}_n(R)\}$. These equivalences are coarser than integral equivalence, $[\mathbf{Q}] = [\mathbf{Q}'] \Rightarrow [\mathbf{Q}]_R = [\mathbf{Q}']_R$. The R-equivalence class is thus an invariant under integral equivalence, which, in contrast to $R = \mathbb{Z}$, might be efficiently computable.

Over the reals one can always diagonalize a quadratic form $\mathbf{D} := \mathbf{U}^\top \mathbf{Q}\mathbf{U}$ with $\mathbf{U} \in \mathcal{GL}_n(\mathbb{R})$ such that $\mathbf{D}_{ii} \in \{-1, 0, 1\}$. Let $n_j = \#\{i : \mathbf{D}_{ii} = j\}$ for $j\{-1, 0, 1\}$ denote the number of occurrences of j on the diagonal. Then (n_1, n_0, n_{-1}), also called the *signature*, uniquely represents the $\mathbb{R}$-equivalence class. A positive definite quadratic form of rank n always has a signature of $(n, 0, 0)$, and thus all quadratic forms of fixed rank that we consider fall in the same class.

We now turn our focus to the p-adic integral equivalence for all primes p. For (positive definite) quadratic forms this equivalence over all primes p is nicely summarized as the genus.

Definition 152 (Genus). *The genus* $\text{genus}(\mathbf{Q})$ *of a (positive definite) quadratic form* $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ *is defined as the collection of* p*-adic integral equivalence classes* $([\mathbf{Q}]_{\mathbb{Z}_p})_p$ *for all primes* p*.*

Two quadratic forms $\mathbf{Q}, \mathbf{Q}' \in \mathcal{S}_n^{>0}(\mathbb{Z})$ have the same genus if and only if they are p-adic integral equivalent for all primes p. In general, the genus is defined to also cover $\mathbb{R}$-equivalence, but for positive definite forms that would be redundant. The genus turns out to cover all arithmetic invariants mentioned so far, while, as we see later, still being efficient to compute.

Theorem 153 (Hasse-Minkowski). *If two (positive definite) quadratic forms are* p*-adic equivalent for all primes* p*, then they are equivalent over the rationals, and their invariants* $\det, \gcd$*, and* par *match.*

Proof. Note that $\mathbb{Z}_p \subset \mathbb{Q}_p$, and that all positive definite quadratic forms of the same rank are $\mathbb{R}$-equivalent. So the first statement follows from the Hasse-Minkowski theorem. The second part of the proof follows from Remark 155. □

We shortly discuss how p-adic integral equivalence can be classified efficiently by a complete system of invariants by Conway [CS13, Ch.15]. Let $p \geq 2$ be any prime. By appropriate (near) diagonalizing any form $\mathbf{Q}$ can be decomposed over the p-adic integers as a block diagonal matrix

$$\mathbf{Q} = \mathbf{Q}_1 \oplus p\mathbf{Q}_p \oplus p^2\mathbf{Q}_{p^2} \oplus \cdots \oplus q\mathbf{Q}_q \oplus \cdots , \tag{9.1}$$

in which each $\mathbf{Q}_q$ is a p-adic integral form whose determinant is coprime to p. For an odd prime $p \neq 2$ the (finite) set of values of q occurring in eq. (9.1), together with the dimensions $n_q := \dim(\mathbf{Q}_q)$ and signs

$$\epsilon_q := \left(\frac{\det(\mathbf{Q}_q)}{p}\right),$$

form a complete set of invariants for the p-adic integral equivalence of $\mathbf{Q}$.

Theorem 154 ([CS13, Ch. 15, Theorem 9]). *For any odd prime $p \neq 2$, two quadratic forms $\mathbf{Q}, \mathbf{Q}'$ are equivalent over the p-adic integers if and only if they have the same invariants n_q, ϵ_q for each power q of p.*

For any odd prime $p \nmid \det(\mathbf{Q})$ we have $n_q = n$, and ϵ_q is fully determined by the determinant $\det(\mathbf{Q})$. As such primes can be ignored, we only have to consider the finite number of odd primes $p \mid \det(\mathbf{Q})$. For such odd primes p there are at most n powers $q_1, \ldots, q_k \leq \det(\mathbf{Q})$ of p with a non-trivial block dimension $n_{q_i} > 0$, and thus $\{(q, n_{q_i}, \epsilon_{q_i})\}_{i=1,\ldots,k}$ determines canonically the p-adic integral equivalence. So the genus (except for $p = 2$) can be described in a finite and canonical way.

For $p = 2$ there are some additional complexities. For example for $\mathbf{Q}_q$ with q a power of 2, the existence (or not) of an odd entry on the diagonal gives an additional invariant, which is related to the earlier defined parity of a quadratic form. If there is an odd entry on the diagonal, the trace t_q mod 8 of $\mathbf{Q}_q$, better known as the oddity, is needed to completely specify the 2-adic equivalence. These oddities are not an invariant, but together with the other invariants they can efficiently be turned into a canonical description (see [CS13, Ch. 15, Sec. 7].

Maybe unsurprisingly, the genus covers all the previously defined invariants.

Remark 155 (Genus invariants). The genus of a quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ completely determines the det, gcd and par of $\mathbf{Q}$.

Proof. For any p and the p-adic decomposition eq. (9.1) the maximal power of p dividing the determinant $\det(\mathbf{Q})$ is given by $\prod_{q=p^i} q^{n_q}$. The maximal power of p dividing $\gcd(\mathbf{Q})$ is given by the minimal $q = p^i$ such that $n_q \geq 1$. The parity is fully determined by the decomposition at $p = 2$. Namely, let $q = 2^i$ be the minimal power of 2 such that $n_q \geq 1$, then $\mathrm{par}(\mathbf{Q}) = 1$ if and only if there is an odd value on the diagonal of a matrix representing $\mathbf{Q}_q$. □

The hull

In the literature for linear code equivalence, a relevant notion is that of the efficiently computable hull $C \cap C^\perp$ of a code $C \subset \mathbb{F}_q^n$. Properties such as the rank of the hull are invariant under equivalence, and a small rank even allows to efficiently find the isometry [Sen00].

For a lattice $\mathcal{L}$ and its dual $\mathcal{L}^*$ we could define the hull as $\mathcal{L} \cap \mathcal{L}^*$. For integral lattices (i.e., if the associated quadratic form is integral) the hull does not present us with new information, since we always have $\mathcal{L} \subset \mathcal{L}^*$ and thus $\mathcal{L} \cap \mathcal{L}^* = \mathcal{L}$. We could generalize the definition to consider $\mathcal{L} \cap (k \cdot \mathcal{L}^*)$ for rational $k \in \mathbb{Q}_{\neq 0}$. However, even for such a generalized construction the genus of the resulting lattice is completely determined by the genus of the original lattice $\mathcal{L}$ [DG22]. To conclude, the (generalized) hull does not appear to give us any new arithmetic invariant.

One possibility is that the generalized hull could have certain geometric properties that could be exploited (more so than the original lattice or its dual). This is a question that deserves further investigation, but is beyond the scope of this work.

9.3.2 Geometric invariants

The defining property of a unimodular transformation $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ is that it gives a bijection $\mathbb{Z}^n \to \mathbb{Z}^n$ by $\mathbf{x} \mapsto \mathbf{U}\mathbf{x}$ (or $\mathbf{x} \mapsto \mathbf{U}^{-1}\mathbf{x}$). With respect to the quadratic forms $\mathbf{Q}, \mathbf{Q}' := \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ this even gives an isometry (from $\mathbf{Q}'$ to $\mathbf{Q}$) as

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}'} = \mathbf{x}^\top \mathbf{Q}' \mathbf{y} = \mathbf{x}^\top \mathbf{U}^\top \mathbf{Q} \mathbf{U} \mathbf{y} = \langle \mathbf{U}\mathbf{x}, \mathbf{U}\mathbf{y} \rangle_{\mathbf{Q}} \text{ for } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n.$$

This isometry results in several natural geometric invariants related to the norms and inner products of integral vectors. We have already seen some, namely the first minimum $\lambda_1(\mathbf{Q})$ and the i-th successive minimum $\lambda_i(\mathbf{Q})$. More geometric invariants can be defined, such as the kissing number $\kappa(\mathbf{Q}) = |\operatorname{Min}(\mathbf{Q})|$ where

$$\operatorname{Min}(\mathbf{Q}) := \{\mathbf{x} \in \mathbb{Z}^n : \|\mathbf{x}\|_{\mathbf{Q}} = \lambda_1(\mathbf{Q})\},$$

and more generally the (formal) Theta-series $\Theta_{\mathbf{Q}}(q) = \sum_{\ell \geq 0} N_\ell q^\ell$ associated to $\mathbf{Q}$, where $N_\ell = |\{\mathbf{x} \in \mathbb{Z}^n : \|\mathbf{x}\|_{\mathbf{Q}}^2 = \ell\}|$. Going back to the inner products one could also consider the (multi-)set of all pairwise inner products $\{\langle \mathbf{x}, \mathbf{y} \rangle : \mathbf{x}, \mathbf{y} \in S\}$, where e.g., $S = \operatorname{Min}(\mathbf{Q})$ or $S = \{\mathbf{x} \in \mathbb{Z}^n : \|\mathbf{x}\|_{\mathbf{Q}}^2 = \ell\}$. One could also consider pairwise inner products between two of such sets that are distinct.

Two equivalent forms also share symmetries $\operatorname{Aut}(\mathbf{Q}') \cong \operatorname{Aut}(\mathbf{Q})$, which for example implies that $|\operatorname{Aut}(\mathbf{Q}')| = |\operatorname{Aut}(\mathbf{Q})|$.

The computation of all these geometric invariants appears to involve finding or even enumerating short vectors, or computing automorphisms; in particular they seem hard to compute in general.

9.4 Characteristic sets

To solve the Lattice Isomorphism Problem we need to recover a right unimodular transformation $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$. Since the group $\mathcal{GL}_n(\mathbb{Z})$ of such transformations is infinite, it is a priori not clear how even a brute-force approach would work. Instead, we follow the framework introduced by Plesken and Souvignier [PS97] based on so-called *characteristic sets*.

Definition 156 (Characteristic Set). *A characteristic vector set function $\mathcal{V}$ is a map that assigns to every $n \geq 1$ and quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$, a finite subset of vectors $\mathcal{V}(\mathbf{Q}) \subset \mathbb{Z}^n$ such that*

(i) $\mathcal{V}(\mathbf{Q})$ generates $\mathbb{Z}^n$ (as a $\mathbb{Z}$-module); and

(ii) for all $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$, we have $\mathbf{U}^{-1}\mathcal{V}(\mathbf{Q}) = \mathcal{V}(\mathbf{U}^\top \mathbf{Q} \mathbf{U})$.

The transformation property (ii) can be interpreted as a canonicity property under equivalence of quadratic forms, or stated differently,

that the map $\mathcal{V}$ is equivariant (for each $n \geq 1$). For example the minimal vectors function $\mathbf{Q} \mapsto \mathrm{Min}(\mathbf{Q})$ satisfies this property; minimal vectors are mapped to minimal vectors by any isometry. Characteristic sets allow us to limit the search space to a finite one; as, by property (ii), each solution of LIP between two quadratic forms $\mathbf{Q}, \mathbf{Q}'$ induces an isometry between their characteristic sets $\mathcal{V}(\mathbf{Q}), \mathcal{V}(\mathbf{Q}')$. Property (i) is necessary to obtain the correspondence in the other direction.

Lemma 157 (Isometries). *Let $\mathcal{V}$ be a characteristic function, and let $\mathbf{Q}, \mathbf{Q}' \in \mathcal{S}_n^{>0}(\mathbb{R})$ be quadratic forms. There is a bijection*

$$\begin{aligned} \mathrm{Isom}(\mathbf{Q}, \mathbf{Q}') &\leftrightarrow \mathrm{Isom}(\mathcal{V}(\mathbf{Q}), \mathcal{V}(\mathbf{Q}')), \\ \mathbf{U} &\mapsto (\psi_{\mathbf{U}} : \mathbf{x} \mapsto \mathbf{U}^{-1}\mathbf{x}), \end{aligned}$$

where $\mathrm{Isom}(\mathcal{V}(\mathbf{Q}), \mathcal{V}(\mathbf{Q}'))$ *is the set of isometries between* $\mathcal{V}(\mathbf{Q})$ *w.r.t.* $\mathbf{Q}$*, and* $\mathcal{V}(\mathbf{Q}')$ *w.r.t.* $\mathbf{Q}'$*. If* $\mathbf{Q} = \mathbf{Q}'$*, then* $\mathrm{Isom}(\mathbf{Q}, \mathbf{Q}') = \mathrm{Aut}(\mathbf{Q})$ *and* $\mathrm{Isom}(\mathcal{V}(\mathbf{Q}), \mathcal{V}(\mathbf{Q}'))$ *are groups under function composition and the bijection is an isomorphism.*

Proof. Let $\mathbf{U} \in \mathrm{Isom}(\mathbf{Q}, \mathbf{Q}')$ be such that $\mathbf{Q}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$. Then by definition of the characteristic set we have $\mathcal{V}(\mathbf{Q}') = \mathbf{U}^{-1}\mathcal{V}(\mathbf{Q})$, and thus $\psi_{\mathbf{U}}$ induces a bijection from $\mathcal{V}(\mathbf{Q})$ to $\mathcal{V}(\mathbf{Q}')$. Furthermore for any $\mathbf{x}, \mathbf{y} \in \mathcal{V}(\mathbf{Q})$ we have $\langle \psi_{\mathbf{U}}(\mathbf{x}), \psi_{\mathbf{U}}(\mathbf{y}) \rangle_{\mathbf{Q}'} = \langle \mathbf{U}^{-1}\mathbf{x}, \mathbf{U}^{-1}\mathbf{y} \rangle_{\mathbf{U}^\top \mathbf{Q} \mathbf{U}} = \langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}}$, and thus $\psi_{\mathbf{U}}$ is actually an isometry.

For the injectivity let $\mathbf{U}, \mathbf{V} \in \mathrm{Isom}(\mathbf{Q}, \mathbf{Q}')$ be such that $\psi_{\mathbf{U}} = \psi_{\mathbf{V}}$. This implies that $\mathbf{U}^{-1}\mathbf{x} = \mathbf{V}^{-1}\mathbf{x}$ for all $\mathbf{x} \in \mathcal{V}(\mathbf{Q})$, and because $\mathcal{V}(\mathbf{Q})$ is of full rank we have $\mathbf{U}^{-1} = \mathbf{V}^{-1}$, and thus $\mathbf{U} = \mathbf{V}$.

For the surjectivity let $\psi : \mathcal{V}(\mathbf{Q}) \to \mathcal{V}(\mathbf{Q}')$ be an isometry. Both characteristic sets generate $\mathbb{Z}^n$ as a $\mathbb{Z}$-module, and thus in particular we can find n $\mathbb{R}$-linear independent vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{V}(\mathbf{Q})$. Because ψ is an isometry, the vectors $\psi(\mathbf{x}_1), \ldots, \psi(\mathbf{x}_n)$ must also be linearly independent, and thus we obtain a *unique* linear map $\mathbf{U} \in \mathcal{GL}_n(\mathbb{R})$ such that $\psi(\mathbf{x}_i) = \psi_{\mathbf{U}}(\mathbf{x}_i)$ for all $i = 1, \ldots, n$. Because ψ is an isometry we have for any $\mathbf{y} \in \mathcal{V}(\mathbf{Q})$ the n independent linear equations $\langle \psi(\mathbf{y}), \psi_{\mathbf{U}}(\mathbf{x}_i) \rangle_{\mathbf{Q}'} = \langle \mathbf{y}, \mathbf{x}_i \rangle_{\mathbf{Q}}$ for $i = 1, \ldots, n$, with the unique solution $\psi(\mathbf{y}) = \psi_{\mathbf{U}}(\mathbf{y})$. So $\psi = \psi_{\mathbf{U}}$ is in fact a linear isometry represented by $\mathbf{U}$, and because both characteristic sets generate $\mathbb{Z}^n$ as a $\mathbb{Z}$-module, we have $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ by definition. From the isometry we obtain that $\mathbf{e}_i^\top \mathbf{U}^\top \mathbf{Q} \mathbf{U} \mathbf{e}_j = \mathbf{e}_i^\top \mathbf{Q}' \mathbf{e}_j$ for all pairs of standard unit

vectors $\mathbf{e}_i, \mathbf{e}_j$ with $i, j = 1, \ldots, n$, which in particular gives us that $\mathbf{Q}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$. So $\mathbf{U} \in \mathrm{Isom}(\mathbf{Q}, \mathbf{Q}')$, and it has image $\psi = \psi_{\mathbf{U}}$.

If $\mathbf{Q} = \mathbf{Q}'$, then both sides are groups under composition. For $\mathbf{U}, \mathbf{V} \in \mathrm{Isom}(\mathbf{Q}, \mathbf{Q}) = \mathrm{Aut}(\mathbf{Q})$ their composition as automorphisms $\mathbf{V} \circ \mathbf{U}$ is represented by $\mathbf{UV} \in \mathrm{Aut}(\mathbf{Q})$. Then for $\mathbf{x} \in \mathcal{V}(\mathbf{Q})$ we have

$$\psi_{\mathbf{V} \circ \mathbf{U}}(\mathbf{x}) = \psi_{\mathbf{UV}}(\mathbf{x}) = (\mathbf{UV})^{-1}\mathbf{x} = \mathbf{V}^{-1}\mathbf{U}^{-1}\mathbf{x} = (\psi_{\mathbf{V}} \circ \psi_{\mathbf{U}})(\mathbf{x}),$$

and thus we have an isomorphism. □

We will now discuss some examples of characteristic vector set functions.

9.4.1 A set based on short vectors

As mentioned before the map $\mathbf{Q} \mapsto \mathrm{Min}(\mathbf{Q})$ satisfies the desired transformation property of a characteristic vector set function. However, two problems remain for using $\mathrm{Min}(A)$ as a characteristic vector set:

PB1. If $n \geq 2$, then $\mathrm{span}(\mathrm{Min}(A))$ may not have rank n.

PB2. If $n \geq 5$, then $\mathrm{span}(\mathrm{Min}(A))$ may have rank n, but may not generate $\mathbb{Z}^n$ as a $\mathbb{Z}$-module.

Thus we have to consider larger sets of short lattice vectors. For $\lambda > 0$, let

$$\mathrm{Min}(\mathbf{Q}, \lambda) := \left\{ \mathbf{x} \in \mathbb{Z}^n \setminus \{\mathbf{0}\} : \|\mathbf{x}\|_{\mathbf{Q}} \leq \lambda \right\}, \tag{9.2}$$

be the set of all nonzero integer vectors up to length λ w.r.t. $\mathbf{Q}$. In particular we have $\mathrm{Min}(\mathbf{Q}) = \mathrm{Min}(\mathbf{Q}, \lambda_1(\mathbf{Q}))$.

Now we need to pick $\lambda > 0$ large enough such that $\mathrm{Min}(\mathbf{Q}, \lambda)$ generates $\mathbb{Z}^n$. The implementations `AUTO/ISOM` by Plesken and Souvignier [PS97], pick λ^2 equal to $\mathrm{maxdiag}(\mathbf{Q}) := \max \mathbf{Q}_{ii}$, to compute automorphisms. The set $\mathrm{Min}(\mathbf{Q}, \sqrt{\mathrm{maxdiag}(\mathbf{Q})})$ contains the standard unit vectors, and thus generates $\mathbb{Z}^n$. One could first use lattice reduction techniques to limit the size of $\mathrm{maxdiag}(\mathbf{Q})$, to prevent large sets. Such a set is enough to compute automorphisms, but can be problematic for equivalence, e.g., two forms $\mathbf{Q}, \mathbf{Q}'$ can be equivalent but satisfy $\mathrm{maxdiag}(\mathbf{Q}) \neq \mathrm{maxdiag}(\mathbf{Q}')$. For a particular LIP instance this can be solved by picking the bound $\lambda^2 = \max\{\mathrm{maxdiag}(\mathbf{Q}), \mathrm{maxdiag}(\mathbf{Q}')\}$. However, such a choice is still not canonical (something we will care

about in Section 9.6), and would in particular break the transformation property. Additionally this may lead to much larger sets than necessary.

To prevent these problems we can use a more reliable vector set, namely by picking the smallest bound λ, such that $\mathrm{Min}(\mathbf{Q}, \lambda)$ still spans $\mathbb{Z}^n$.

$$\begin{aligned} \mathcal{V}_{\mathrm{ms}}(\mathbf{Q}) &:= \mathrm{Min}(\mathbf{Q}, \lambda_{\min}), \text{ where} \\ \lambda_{\min} &:= \min\{\lambda > 0 : \mathrm{span}_{\mathbb{Z}}(\mathrm{Min}(\mathbf{Q}, \lambda)) = \mathbb{Z}^n\}. \end{aligned} \tag{9.3}$$

Lemma 158. *The map $\mathbf{Q} \mapsto \mathcal{V}_{\mathrm{ms}}(\mathbf{Q})$ is a characteristic vector set function.*

This characteristic set function often works great in practice[7], but in general we do not have any bound on the size of $|\mathcal{V}_{\mathrm{ms}}(\mathbf{Q})|$, as the following example shows.

Example 159. The quadratic form $\mathbf{Q}_\lambda = \begin{pmatrix} 1 & 0 \\ 0 & \lambda^2 \end{pmatrix}$ for $\lambda \geq 1$ gives

$$\mathcal{V}_{\mathrm{ms}}(\mathbf{Q}_\lambda) = \{\pm e_2\} \cup \{\pm e_1, \pm 2e_1, \ldots, \pm \lfloor \lambda \rfloor e_1\}.$$

One way to solve the particular example above is by also removing all non-primitive vectors from the set, but this is not enough to solve the general problem of obtaining vector sets of unbounded size (under a fixed dimension). To overcome this problem, we introduce a characteristic vector set, although somewhat impractical, that does come with bounds on the number of vectors, only depending on the dimension n.

9.4.2 A set based on Voronoi-relevant vectors

A well-known geometric shape associated to lattices, extensively discussed in Chapter 5, is the *Voronoi cell*. The Voronoi cell is the set of all points closer to 0 with respect to $\mathbf{Q}$ than to any other integer point. For a form $\mathbf{Q}$, the (closed) Voronoi cell is the intersection of half-spaces

$$\mathrm{Vor}(\mathbf{Q}) := \bigcap_{\mathbf{x} \in \mathbb{Z}^n \setminus \{0\}} H_{\mathbf{Q},\mathbf{x}}, \tag{9.4}$$

[7] Practical for computational algebra purposes in low dimensions of say $n \leq 30$. Not for cryptographic lattices of large dimension.

with $H_{\mathbf{Q},\mathbf{x}} := \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{y}\|_{\mathbf{Q}} \leq \|\mathbf{y} - \mathbf{x}\|_{\mathbf{Q}}\}$. However, almost all vectors in this intersection are superfluous, and we only consider the finite set of *Voronoi-relevant vectors* $\mathcal{V}_{\mathrm{vor}}(\mathbf{Q})$, i.e., the (unique) minimal set of vectors such that

$$\mathrm{Vor}(\mathbf{Q}) = \bigcap_{\mathbf{x} \in \mathcal{V}_{\mathrm{vor}}(\mathbf{Q})} H_{\mathbf{Q},\mathbf{x}}. \tag{9.5}$$

Each Voronoi-relevant vector corresponds to exactly one facet of the Voronoi cell. Using, an algorithm by Micciancio and Voulgaris [MV13], we can compute the set of Voronoi-relevant vectors in time $2^{2n+o(n)}$.

Lemma 160. *The following statements hold*

(i) The map $\mathbf{Q} \mapsto \mathcal{V}_{\mathrm{vor}}(\mathbf{Q})$ *is a characteristic vector set function;*

(ii) We have $\#\mathcal{V}_{\mathrm{vor}}(\mathbf{Q}) \leq 2 \cdot (2^n - 1)$.

Proof. Property (ii) of a characteristic vector set for $\mathcal{V}_{\mathrm{vor}}$ follows from the geometric definition, fully independent of the basis. For property (i), note that for any nonzero $\mathbf{x} \in \mathbb{Z}^n$, we have $\mathbf{x} \notin \mathrm{Vor}(\mathbf{Q})$, and thus by definition there is a Voronoi-relevant vector $\mathbf{v} \in \mathcal{V}_{\mathrm{vor}}(\mathbf{Q})$ such that $\mathbf{x} - \mathbf{v}$ lies strictly closer to 0 with respect to $\mathbf{Q}$. Repeating this (a finite number of time by a packing argument) we eventually end up at 0 and thus $\mathbf{x}$ is the sum of Voronoi-relevant vectors. The second statement was already proven by Minkowski in 1897 [Min97]. The idea is that each Voronoi-relevant vector $\mathbf{v}$ has precisely two closests vectors in $2\mathcal{L}$, namely $\mathbf{0}$ and $2\mathbf{v}$. This implies that each $\pm\mathbf{v}$ (up to sign) represents a distinct and nonzero coset modulo $2\mathcal{L}$, so there can be at most $2 \cdot (|\mathcal{L}/(2\mathcal{L})| - 1)$ Voronoi-relevant vectors. □

Although this characteristic vector set has great theoretical bounds, we refrain from using it in practice: most lattices actually attain the worst-case $2 \cdot (2^n - 1)$ Voronoi bound [Vor08; Vor09], whereas constructions based on short vectors often beat the theoretical worst-case bounds and give much smaller vector sets in practice.

9.5 Solving LIP

We discuss several algorithms to solve the Lattice Isomorphism Problem. We start with some practical approaches based on the earlier

introduced characteristic vector sets, and then discuss some theoretical results.

9.5.1 Characteristic set approach

Recall from Lemma 157 that solving LIP between two quadratic forms $\mathbf{Q}, \mathbf{Q}'$ is equivalent to finding an isometry between the finite sets $\mathcal{V}(\mathbf{Q})$ and $\mathcal{V}(\mathbf{Q}')$, for any characteristic set function $\mathbf{Q} \mapsto \mathcal{V}(\mathbf{Q})$.

A generic approach, that we formalize in [DHVW20], of finding such isometries is to construct two complete weighted graphs for $\mathbf{Q}$ and $\mathbf{Q}'$ respectively, where the nodes are the elements from the characteristic vector set, and each edge $(\mathbf{x}, \mathbf{y})$ gets weight $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}}$ or $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}'}$ respectively.

Definition 161. *Let $\mathcal{V}$ be a characteristic vector set function, and let $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$ be a quadratic form. We define the complete weighted graph $G_{\mathbf{Q},\mathcal{V}} = (V, w)$ as the graph with nodes $V := \mathcal{V}(\mathbf{Q})$ and weight function*

$$w(\mathbf{x}, \mathbf{y}) := \langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}} \text{ for all } \mathbf{x}, \mathbf{y} \in \mathcal{V}(\mathbf{Q}).$$

Using these graphs turns the problem of quadratic form equivalence into one of graph isomorphism. Recall that two complete weighted graphs $G = (V, w)$ and $G' = (V', w')$ are isomorphic $G \cong G'$ if there exists a bijection $\psi : V \to V'$, which keeps the edge weights intact, i.e., $w'(\psi(x), \psi(y)) = w(x, y)$ for each edge $(x, y) \in V^2$. We denote the set of all such bijections by $\mathrm{Isom}(G, G')$.

Lemma 162. *Let $\mathcal{V}$ be a characteristic vector set function, and let $\mathbf{Q}, \mathbf{Q}' \in \mathcal{S}_n^{>0}(\mathbb{R})$ be two quadratic forms, then $\mathbf{Q}$ and $\mathbf{Q}'$ are equivalent if and only if $G_{\mathbf{Q},\mathcal{V}}$ and $G_{\mathbf{Q}',\mathcal{V}}$ are isomorphic. In particular there exists a bijection*

$$\mathrm{Isom}(\mathbf{Q}, \mathbf{Q}') \leftrightarrow \mathrm{Isom}(G_{\mathbf{Q},\mathcal{V}}, G_{\mathbf{Q}',\mathcal{V}}).$$

If $\mathbf{Q} = \mathbf{Q}'$, then $\mathrm{Isom}(\mathbf{Q}, \mathbf{Q}') = \mathrm{Aut}(\mathbf{Q})$ and $\mathrm{Isom}(G_{\mathbf{Q},\mathcal{V}}, G_{\mathbf{Q}',\mathcal{V}})$ are groups under function composition and the bijection is an isomorphism.

Proof. By Lemma 157 we only have to show that there exists a bijection between $\mathrm{Isom}(\mathcal{V}(\mathbf{Q}), \mathcal{V}(\mathbf{Q}'))$ and $\mathrm{Isom}(G_{\mathbf{Q},\mathcal{V}}, G_{\mathbf{Q}',\mathcal{V}})$. Because all pairwise inner products (and norms) are encoded in the graph, these

two sets are the same by definition, so the bijection (and isomorphism) is trivial. □

So to solve LIP, we can use heavily optimized graph isomorphism algorithms on the resulting graphs, such as `Nauty`, `Traces` [MP14] and `Bliss` [JK07]. These are backtrack algorithms, that try to efficiently enumerate all possible node mappings that keep the edge weights invariant. They make heavy use of graph invariants to limit the search tree, such as the degree of nodes, or the degree of all neighbours. These algorithms do not directly work on weighted graphs, but in the `Nauty` manual [McK07] there are generic transformations from weighted graphs to (somewhat larger) unweighted graphs that keep the isomorphisms intact. Depending on the size of the characteristic set $\mathcal{V}(\mathbf{Q})$, this approach can be very practical, and a public implementation by Dutour Sikirić can be found at [Dut22]. On the asymptotic side, the graph isomorphism algorithm by Babai [Bab16], gives an algorithm that runs in time quasi-polynomial in the characteristic vector set size $|\mathcal{V}(\mathbf{Q})|$. However, this time may be as large as $\exp(n^{O(1)})$ given that $|\mathcal{V}(\mathbf{Q})|$ itself may be as large as $\exp(\Theta(n))$, for example, when using Voronoi-relevant vectors.

The approach of Plesken and Pohst [PP85], later improved by Plesken and Souvignier [PS97], can be seen as a specialized variant of the above approach, where more (geometric) invariants are considered to improve the search, and possibly without explicitly constructing the full graph. Implementations by Souvignier for automorphisms and equivalence testing are available under the names `AUTO` and `ISOM` respectively[8].

9.5.2 A provable algorithm

The best asymptotic algorithm is one by Haviv and Regev [HR14] that runs in time and space $n^{O(n)}$, and returns all solutions to a LIP instance. As the lattice $\mathbb{Z}^n$ has $2^n \cdot n! = n^{O(n)}$ automorphisms, this result is in some sense optimal. A major open question is, whether there exists a single exponential time algorithm to find only a single solution to LIP.

[8] The `AUTO` and `ISOM` programs are available through MAGMA. The implementation has also been ported to PARI/GP (and thus SageMath).

The algorithm by Haviv and Regev requires short primal as well as short dual vectors. Let $\mathbf{Q}, \mathbf{Q}' \in \mathcal{S}_n^{>0}(\mathbb{R})$ be two equivalent quadratic forms. They first consider the case that $\mathrm{Min}(\mathbf{Q})$ (and $\mathrm{Min}(\mathbf{Q}')$) are full rank, i.e., $\mathrm{span}(\mathrm{Min}(\mathbf{Q})) = \mathbb{R}^n$. Fundamental to their algorithm is the so-called isolation lemma.

Lemma 163 (Isolation Lemma [HR14], informal). *there exists a relatively short dual vector $\mathbf{v} \in \mathbb{Z}^n$ (in the sense of $\|\mathbf{v}\|_{\mathbf{Q}^{-1}}$), that uniquely defines n linearly independent vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathrm{Min}(\mathbf{Q})$. These vectors are defined as follows:*
for every $1 \leq j \leq n$, the minimum standard inner product[9] $\langle \mathbf{v}, \mathbf{x} \rangle$ with vectors $\mathbf{x} \in \mathrm{Min}(\mathbf{Q}) \setminus \mathrm{span}(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1})$ is uniquely achieved by $\mathbf{x}_j$.

After such an isolating vector $\mathbf{v} \in \mathbb{Z}^n$ is found for $\mathrm{Min}(\mathbf{Q})$ by enumerating short dual vectors, we do the same for $\mathrm{Min}(\mathbf{Q}')$. Once the correct image of $\mathbf{v}$ in $\mathrm{Min}(\mathbf{Q}')$ is found by a similar enumeration (which happens because short dual vectors are mapped to short dual vectors), we obtain another unique chain of n independent vectors $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathrm{Min}(\mathbf{Q}')$, and the unimodular transformation $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ can be recovered from the mapping $\mathbf{y}_i \mapsto \mathbf{x}_i$. To provably find such an isolating short dual vector we need to enumerate all dual vectors up to some norm $n^{O(1)} \cdot \lambda_1(\mathbf{Q}^{-1})$, of which there can be at most $n^{O(n)}$. Running such an enumeration can also be done in time $n^{O(n)}$, and space $\exp(O(n))$ (to store $\mathrm{Min}(\mathbf{Q}_1), \mathrm{Min}(\mathbf{Q}_2)$).

For the general case Regev and Haviv note that one can first solve LIP for the sublattices in the span of $\mathrm{Min}(\mathbf{Q})$ and $\mathrm{Min}(\mathbf{Q}')$ respectively, and (recursively) for the lattices projected away from that span. The resulting transformations can then be combined if compatible.

While the algorithm is useful as an asymptotic result, the (proven) constant in the exponent of $n^{O(n)}$ is quite large, which makes the algorithm seemingly impractical. We do not know of any implementation of this algorithm, and therefore if the practical complexity might be better.

[9] The standard inner product $\langle \mathbf{v}, \mathbf{x}_i \rangle$ has the same value as the inner product between the primal lattice vector $\mathbf{B}\mathbf{x}_j$ and the dual lattice vector $(\mathbf{B}^\top)^{-1}\mathbf{v}$ for any full rank basis $\mathbf{B}$.

9.5.3 The case of $\mathbb{Z}^n$, an approach by Szydlo

In 2003 Szydlo [Szy03] showed a search LIP to decision LIP reduction for the special case of the integer lattice $\mathbb{Z}^n$. Note that for $\mathbb{Z}^n$ the standard basis gives the quadratic form $\mathbf{I}_n$, so the search problem becomes: given $\mathbf{U}^\top\mathbf{U}$ for some unimodular matrix $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$, recover $\mathbf{U}$ (up to $\mathrm{Aut}(\mathbf{I}_n)$; i.e., all signed permutations). He shows how to solve this search problem by querying a decisional LIP oracle on lattices that are very similar to $\mathbb{Z}^n$.

Additionally, Szydlo gives a heuristic algorithm to instantiate such a decision oracle for these specific cases, that requires to sample vectors of length $O(\sqrt{n})$. With the current reduction techniques, however, it is easier to find the unusually short vectors of length 1, in any rotation of $\mathbb{Z}^n$, than to enumerate vectors of length $O(\sqrt{n})$. So even if this heuristic algorithm works, it is worse than solving the search problem directly.

An interesting open question is, whether there exists a more general search to decision reduction for LIP.

9.6 A canonical function

Using the algorithm in Section 9.5, we can compute the automorphism group, and check for equivalence between two quadratic forms of low dimension. However, in practice LIP often appears in the following setting: we have many quadratic forms $\mathbf{Q}_1, \ldots, \mathbf{Q}_N \in \mathcal{S}_n^{>0}$, and we wish to identify them up to equivalence. For example, this occurs during the enumeration (up to equivalence) of quadratic forms of low rank with special properties, such as unimodular forms, perfect forms and more. Since a naive application of an equivalence algorithm requires $O(N^2)$ equivalence tests (in the worse case), this can quickly become a bottleneck. The number of tests can be somewhat lowered if many of the invariants presented in Section 9.3 differ, which may or may not be the case.

The solution we present in this section solves the above problem by introducing a canonical form $\mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}(\mathbf{Q}) \in \mathcal{S}_n^{>0}$ for $\mathbf{Q} \in \mathcal{S}_n^{>0}$. This canonical form should satisfy the following two requirements.

Definition 164. *We say* $\mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})} : \mathcal{S}_n^{>0} \to \mathcal{S}_n^{>0}$ *is a canonical form function if it satisfies the following two requirements:*

1. *For every* $\mathbf{Q} \in \mathcal{S}_n^{>0}$, $\mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}(\mathbf{Q})$ *is equivalent to* $\mathbf{Q}$*; and*

2. *for every* $\mathbf{Q} \in \mathcal{S}_n^{>0}$ *and* $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$, $\mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}(\mathbf{U}^\top \mathbf{Q} \mathbf{U}) = \mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}(\mathbf{Q})$.

We call the output $\mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}(\mathbf{Q})$ of $\mathbf{Q}$ the canonical form of $\mathbf{Q}$ (assuming that the function is fixed). A canonical form can be seen as a canonical representative of the equivalence class $[\mathbf{Q}]$, and thus two quadratic forms are equivalent if and only if their canonical forms are identical. Combining a canonical form with a hash table, the identification of equivalence classes in a list of N quadratic forms then takes only N canonical form computations (and N hash table lookups), and thus has the potential to be much faster. In Section 9.7 we present an application with $N \geq 10^9$ quadratic forms that became feasible largely due to the introduction of a canonical form function.

Recall from Lemma 162 that we used a characteristic vector set function $\mathcal{V}$ to turn the question of quadratic form equivalence $\mathbf{Q} \cong \mathbf{Q}'$ into one of graph isomorphism $G_{\mathbf{Q},\mathcal{V}} \cong G_{\mathbf{Q}',\mathcal{V}}$. For graphs there is already a long line of research into canonical functions $\mathrm{Can}_{\mathrm{Sym}_m}$ (e.g., [BKL80; McK81; BL83; FSS83; JK07; MP14; Bab19]). The main idea is to apply a canonical function to the graph $G_{\mathbf{Q},\mathcal{V}}$, and then lift the result back.

For simplicity we limit ourself to the set $\mathcal{G}$ of weighted graphs with (ordered) nodes $[m] = \{1, \ldots, m\}$ for some $m \geq 1$, such that automorphisms and isomorphisms can be represented by permutations. For a graph $G = ([m], w) \in \mathcal{G}$ with $m \geq 1$ nodes and weight function w, and a permutation $\pi \in \mathrm{Sym}_m$ we denote by $\pi(G) = ([m], w') \in \mathcal{G}$ the graph with weight function w' given by $w'(i,j) := w(\pi^{-1}(i), \pi^{-1}(j))$ for all $i, j = 1, \ldots, m$. Two graphs $G, G' \in \mathcal{G}$ with $m \geq 1$ nodes are called isomorphic $G \cong G'$ if there exists a $\pi \in \mathrm{Sym}_m$ such that $\pi(G) = G'$.

Definition 165. *For* $m \geq 1$ *we say that* $\mathrm{Can}_{\mathrm{Sym}_m} : \mathcal{G} \to \mathcal{G}$ *is a canonical graph ordering if it satisfies the following requirement:*

1. *For every* $G \in \mathcal{G}$, $\mathrm{Can}_{\mathrm{Sym}_m}(G)$ *is isomorphic to* G*; and*

2. *for every* $G \in \mathcal{G}$ *with* $m \geq 1$ *nodes and* $\pi \in \mathrm{Sym}_m$, $\mathrm{Can}_{\mathrm{Sym}_m}(\pi(G)) = \mathrm{Can}_{\mathrm{Sym}_m}(G)$.

In fact, the graph isomorphism implementations mentioned in Section 9.5, solve the graph isomorphism problem by computing a canonical graph ordering on both graphs. On the theoretical side there also has been some effort by Babai [Bab19] to turn the quasi-polynomial (in m) graph isomorphism algorithm [Bab16] into a canonical graph ordering algorithm.

Remark 166. Given any vector set function $\mathcal{V}$ and canonical graph ordering $\mathrm{Can}_{\mathrm{Sym}_m}$, the map $\mathbf{Q} \mapsto \mathrm{Can}_{\mathrm{Sym}_m}(G_{\mathbf{Q},\mathcal{V}})$ already defines a canonical function, in the sense that the output on two forms is identical if and only if the forms are equivalent. However this graph can be quite large, and the canonical quadratic form allows a more compact representation.

We now construct a canonical form function by lifting the canonical graph ordering back to the quadratic form setting. For this we first need another type of canonization algorithm: the Hermite Normal Form.

Definition 167 (Hermite Normal Form (HNF)). *The Hermite Normal Form of an integer matrix* $\mathbf{M} \in \mathbb{Z}^{n \times m}$ *of row rank* $r \leq n$ *is the unique matrix* $\mathbf{H} \in \mathbb{Z}^{n \times m}$ *for which there exists a* $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$, *such that* $\mathbf{M} = \mathbf{UH}$, *and moreover*

1. *The first* r *rows of* $\mathbf{H}$ *are nonzero and the remaining rows are zero; and*

2. *for* $1 \leq i \leq r$, *if* h_{i,j_i} *is the first nonzero entry in row* i, *then* $j_1 < \ldots < j_r$; *and*

3. *for* $1 \leq i \leq r$, *we have* $h_{i,j_i} > 0$; *and*

4. *for* $1 \leq k < i \leq r$, *we have* $0 \leq h_{k,j_i} < h_{i,j_i}$.

The HNF of a matrix is computable in polynomial time [KB79; MW01]. The HNF can be seen as a canonical function of the left action of $\mathcal{GL}_n(\mathbb{Z})$ given by multiplication on integer matrices $\mathbb{Z}^{n \times m}$.

In particular it turns any (transpose) basis of a lattice into a (transpose) canonical basis. Note however that it is only canonical up to the action of $\mathcal{GL}_n(\mathbb{Z})$, and not with respect to the action of $\mathcal{O}_n(\mathbb{R})$. Therefore it cannot directly be used to solve LIP. Furthermore, given that $\mathcal{V}(\mathbf{U}^\top \mathbf{Q}\mathbf{U}) = \mathbf{U}^{-1}\mathcal{V}(\mathbf{Q})$, one might be temped to apply HNF directly to the characteristic set to make the action of $\mathcal{GL}_n(\mathbb{Z})$ canonical, however, the output of the HNF does depend on the order of the characteristic vectors, i.e., it is not simultaneously canonical with respect to the permutation action of Sym_m. Obtaining a canonical ordering is precisely where we use the canonical graph function.

Algorithm 15: A canonical form function $\mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}$.

Given : A characteristic set function $\mathcal{V}$, and a canonical graph function $\mathrm{Can}_{\mathrm{Sym}_m}$.
Input : A quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}$.
Output: A canonical representative $\mathbf{Q}' \in [\mathbf{Q}]$.

1 Compute the characteristic vector set $\mathcal{V}(\mathbf{Q})$, $m \leftarrow |\mathcal{V}(\mathbf{Q})|$
2 Construct the weighted graph $G_{\mathbf{Q},\mathcal{V}}$
3 Run $\mathrm{Can}_{\mathrm{Sym}_m}$ on $G_{\mathbf{Q},\mathcal{V}}$ to obtain a canonical ordering $\mathbf{v}_1, \ldots, \mathbf{v}_m$ of $\mathcal{V}(\mathbf{Q})$
4 $\mathbf{M} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_m) \in \mathbb{Z}^{n \times m}$
5 Compute $\mathbf{U}_{\mathcal{V}(\mathbf{Q})} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{M} = \mathbf{U}_{\mathcal{V}(\mathbf{Q})}\mathbf{H}$ is the unique HNF decomposition of $\mathbf{M}$
6 **return** $\mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}(\mathbf{Q}) = \mathbf{U}_{\mathcal{V}(\mathbf{Q})}^\top \mathbf{Q}\mathbf{U}_{\mathcal{V}(\mathbf{Q})}$

Proposition 168. *Algorithm 15 is a canonical form function.*

Proof. We first show that the algorithm is well defined, i.e., that it does not depend on the initial ordering of the characteristic vector set $\mathcal{V}(\mathbf{Q})$. Note that the output ordering of $\mathrm{Can}_{\mathrm{Sym}_m}$ is unique up to $\mathrm{Aut}(G_{\mathbf{Q},\mathcal{V}})$, and thus by Lemma 162 the ordering of the characteristic vectors is unique up to $\mathrm{Aut}(\mathbf{Q})$. For any such different ordering $\mathbf{S}\mathbf{v}_1, \ldots, \mathbf{S}\mathbf{v}_m$ for $\mathbf{S} \in \mathrm{Aut}(\mathbf{Q})$ would lead to the matrix $\mathbf{S}\mathbf{U}_{\mathcal{V}(\mathbf{Q})}$, and thus $\mathbf{U}_{\mathcal{V}(\mathbf{Q})}$ is well-defined as a coset representative in $\mathrm{Aut}(\mathbf{Q})\backslash\mathcal{GL}_n(\mathbb{Z})$. As $\mathbf{Q}$ is by definition invariant under $\mathrm{Aut}(\mathbf{Q})$ the output form is thus well-defined.

We now verify the two properties in Definition 164. The first property is clear by definition. For the second property note that for any $\mathbf{V} \in \mathcal{GL}_n(\mathbb{Z})$, we have

$$\mathbf{U}_{\mathcal{V}(\mathbf{V}^\top \mathbf{Q}\mathbf{V})} \equiv \mathbf{U}_{\mathbf{V}^{-1}\mathcal{V}(\mathbf{Q})} \equiv \mathbf{V}^{-1}\mathbf{U}_{\mathcal{V}(\mathbf{Q})} \in \mathrm{Aut}(\mathbf{V}^\top \mathbf{Q}\mathbf{V}) \setminus \mathcal{GL}_n(\mathbb{Z}),$$

and thus $\mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}(\mathbf{V}^\top \mathbf{Q}\mathbf{V}) = \mathrm{Can}_{\mathcal{GL}_n(\mathbb{Z})}(\mathbf{Q})$ as desired. Here the first equivalence stems from the transformation property of the characteristic set, and the second equivalence stems from the uniqueness of the HNF. □

9.7 Applications to perfect form enumeration

The technique to construct a canonical form from graphs is quite versatile and has many applications. For example, a similar function can be constructed for symplectic groups [DHVW20], or for the equivalence of C-type domains [DMW22]. Beyond testing for equivalence, a canonical function also gives a deterministic naming scheme for lattices (for a fixed function). In this section, we dive in a bit deeper on applying the canonical form function for perfect form enumeration, and we discuss a further improvement that gives a 30-fold speed-up in practice.

9.7.1 Background on perfect lattices and forms

The search for perfect forms is motivated by the Lattice Packing Problem (LPP): how to pack n-dimensional balls in $\mathbb{R}^n$ in a lattice pattern, such that their density, the proportion of $\mathbb{R}^n$ they fill, is maximized? So far LPP has only been solved up to dimension 8 and in dimension 24 (by the remarkable Leech lattice). It is known that every optimal lattice packing is attained by a perfect lattice, and up to similarity there are only a finite number of them in each dimension. Therefore, to solve LPP, one can simply enumerate all perfect lattices.

From a quadratic form perspective the goal of LPP is to find a form $\mathbf{Q} \in \mathcal{S}_n^{>0}$ that maximizes $\lambda_1(\mathbf{Q})^2/\det(\mathbf{Q})^{1/n}$. This is a search problem over the cone of (positive definite) quadratic forms $\mathcal{S}_n^{>0}$. As the objective function is invariant under scaling we can restrict the

space to those forms with $\lambda_1(\mathbf{Q})^2 \geq 1$, which defines the Ryshkov polyhedra

$$\mathcal{P} := \{\mathbf{Q} \in \mathcal{S}_n^{>0} : \lambda_1(\mathbf{Q})^2 \geq 1\}.$$

The inequality $\lambda_1(\mathbf{Q})^2 \geq 1$ is equivalent to the infinite set of (linear) inequalities $\mathbf{x}^\top \mathbf{Q}\mathbf{x} \geq 1$ for all nonzero $\mathbf{x} \in \mathbb{Z}^n$, which implies that the Ryshkov polyhedra is an intersection of infinitely many halfspaces. Over the Ryshkov polyhedra we now only have to minimize the determinant. Minkowski showed that the function $\mathbf{Q} \mapsto \det(\mathbf{Q})^{1/n}$ is concave on $\mathcal{S}_n^{>0}$, which implies that the optima lie at the vertices of the Ryshkov polyhedra, and we call the quadratic forms that correspond to these vertices *perfect*. Up to equivalence (and scaling) there are only a finite number of distinct perfect forms. In fact, we have an upper bound of $\exp(O(n^2))$ on the number of equivalence classes [Woe20]. Voronoi's Algorithm [Vor08] explores the Ryshkov polyhedra up to equivalence, and thereby enumerates all perfect forms in a fixed dimension. It has been successfully used to find all perfect forms in dimensions up to 8, with respectively $1, 1, 1, 2, 3, 7, 33, 10916$ equivalence classes in each dimension.

While in dimension 8 the main bottleneck was due to high degeneracy in the Ryshkov polyhedra, we have an additional problem in dimension 9: the large number of perfect forms. With possibly billions of non-equivalent perfect forms we require a very efficient canonical function to quickly check if an encountered perfect form is already known or not. Previous enumeration attempts in dimension 9, stranded at only 500.000 [SSV07] or 25 million [Woe18] forms respectively, with equivalence computations as the main bottleneck.

9.7.2 An improved canonical function

Ignoring the few highly degenerate vertices (perfect forms), the main cost of enumerating all 9-dimensional perfect forms is the computation of canonical forms. Any speed-up in this computation directly translates to a significantly reduced running time to finish the enumeration. We discuss one trick to achieve this. For perfect forms the most costly step in Algorithm 15 is that of computing the canonical graph function, e.g., using `Nauty`, `Traces` [MP14] or `Bliss` [JK07]. The runtime of these algorithms depends on many factors, but when restricting to our application the runtime mostly seems to depend on

the number of vertices $|\mathcal{V}(\mathbf{Q})|$, and the number of distinct weights, i.e., $|\{\langle \mathbf{x}, \mathbf{y}\rangle_{\mathbf{Q}} : \mathbf{x}, \mathbf{y} \in \mathcal{V}(\mathbf{Q})\}|$.

In order to improve the algorithm we thus have to try to limit the size of this graph. The minimal vectors of a perfect form often (except for a few cases) already span $\mathbb{Z}^n$, and thus we have $\mathcal{V}(\mathbf{Q}) := \mathcal{V}_{\text{ms}}(\mathbf{Q}) = \text{Min}(\mathbf{Q})$. Reducing this set any further seems hard (and is actually impossible if the automorphism group acts transitively). Note, however, that we have the sign symmetry $\mathbf{x} \in \mathcal{V}_{\text{ms}}(\mathbf{Q}) \Leftrightarrow -\mathbf{x} \in \mathcal{V}_{\text{ms}}(\mathbf{Q})$. What happens if we ignore the signs?

Similarly to $G_{\mathbf{Q},\mathcal{V}}$ we can construct the *absolute* graph $G^{\text{abs}}_{\mathbf{Q},\mathcal{V}}$, where each node corresponds to some $\pm\mathbf{x} \in \mathcal{V}(\mathbf{Q})$, and the weight function is (well-)defined as $w(\pm\mathbf{x}, \pm\mathbf{y}) := |\langle \mathbf{x}, \mathbf{y}\rangle_{\mathbf{Q}}|$.

Lemma 169. *For any perfect form $\mathbf{Q} \in \mathcal{S}_n^{>0}$, and the characteristic set function $\mathcal{V}_{\text{ms}}$, we have an injective homomorphism*

$$\begin{aligned} \text{Aut}(G_{\mathbf{Q},\mathcal{V}_{\text{ms}}})/\langle \pm\mathbf{I}_n\rangle &\hookrightarrow \text{Aut}(G^{abs}_{\mathbf{Q},\mathcal{V}_{\text{ms}}}), \\ \pi &\mapsto \pi^{abs}, \end{aligned}$$

where $-\mathbf{I}_n$ sends $\mathbf{x} \mapsto -\mathbf{x}$ for all $\mathbf{x} \in \mathcal{V}_{\text{ms}}$.

Proof. Let $\pi \in \text{Aut}(G_{\mathbf{Q},\mathcal{V}_{\text{ms}}})$. The inner product constraints enforce that if π maps $\mathbf{x} \mapsto \mathbf{y}$, then $-\mathbf{x} \mapsto -\mathbf{y}$ for $\mathbf{x}, \mathbf{y} \in \mathcal{V}_{\text{ms}}$. So π induces a permutation π^{abs} on $G^{\text{abs}}_{\mathbf{Q},\mathcal{V}_{\text{ms}}}$ by mapping pairs to pairs $\pm\mathbf{x} \mapsto \pm\mathbf{y}$. Because π is invariant with respect to the inner products, π^{abs} is clearly invariant with respect to the absolute invariants, and thus $\pi^{\text{abs}} \in \text{Aut}(G^{\text{abs}}_{\mathbf{Q},\mathcal{V}_{\text{ms}}})$.

For the injectivity we have to show that the kernel is $\langle \pm\mathbf{I}_n\rangle$. First suppose (as we will later prove) that the graph $G_{\mathbf{Q},\mathcal{V}_{\text{ms}}}$ is connected if we remove all edges of weight 0 . Let $\pi \in \text{Aut}(G_{\mathbf{Q},\mathcal{V}_{\text{ms}}})$ be such that π^{abs} is trivial. Every $\mathbf{x} \in \mathcal{V}_{\text{ms}}$ is mapped to either $\mathbf{x}$ or $-\mathbf{x}$. However any pair $\mathbf{x}, \mathbf{y}$ is connected by some path with nonzero (signed) inner products in $G_{\mathbf{Q},\mathcal{V}_{\text{ms}}}$, and thus any choice for $\mathbf{x} \mapsto \mathbf{x}$ or $\mathbf{x} \mapsto -\mathbf{x}$ determines the signs for all other elements. So there are only two choices for π, and clearly the identity map $\mathbf{I}_n$ and the negation map $-\mathbf{I}_n$ satisfy this.

What remains is to shoe that the graph $G_{\mathbf{Q},\mathcal{V}_{\text{ms}}}$ is connected. We have $\text{Min}(\mathbf{Q}) \subset \mathcal{V}_{\text{ms}}$, and by definition of perfect, the set $\{\mathbf{x}\mathbf{x}^\top : \mathbf{x} \in \text{Min}(\mathbf{Q})\}$ has full rank $\frac{1}{2}n(n+1)$ inside the space of $n \times n$ symmetric

matrices. Assume for contradiction that the graph $G_{\mathbf{Q},\mathcal{V}_{\text{ms}}}$ is disconnected if we remove the edges of weight 0. This implies that we can decompose $\mathcal{V}_{\text{ms}} = V_1 \sqcup V_2$ into two disjunct non-empty sets such that $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}} = 0$ for all pairs $(\mathbf{x}, \mathbf{y}) \in V_1 \times V_2$. Let $1 \leq r_i := \text{rk}(V_i)$ be the rank of the subspace spanned by V_i inside $\mathbb{R}^n$, and note that the pairwise orthogonality gives $r_1 + r_2 \leq n$. By the (strict) convexity of the map $c \mapsto c(c+1)$ for $c > 0$ we then obtain the following contradiction

$$\frac{1}{2}n(n+1) \leq \sum_{i=1}^{2} \text{rk}\{\mathbf{x}\mathbf{x}^{\top} : \mathbf{x} \in V_i\} \leq \sum_{i=1}^{2} \frac{1}{2} r_i(r_i+1) < \frac{1}{2}n(n+1),$$

and thus the graphs are connected. □

If the above injective morphism is in fact an isomorphism, and we have an efficient way to lift π^{abs} back to π (up to sign), then we could simply use $G^{\text{abs}}_{\mathbf{Q},\mathcal{V}}$ instead of $G_{\mathbf{Q},\mathcal{V}}$, with half as many nodes and almost half as many distinct weights. In fact an algorithm for the inverse map immediately follows from the proof of Lemma 9.7.2: fix a single sign choice, and determine the other signs uniquely via a spanning tree of the graph.

In general however, removing the signs could also introduce more automorphisms, so we have to check if this is the case or not. The strategy is as follows, first we compute the canonical function using $G^{\text{abs}}_{\mathbf{Q},\mathcal{V}}$. As a cheap by-product of the canonical graph algorithm we also obtain generators for $\text{Aut}(G^{\text{abs}}_{\mathbf{Q},\mathcal{V}})$, and we check if each of these (unsigned) generators π^{abs} correctly lifts to a (signed) automorphism π of $G_{\mathbf{Q},\mathcal{V}}$. If this is the case then we are done, if not then we recompute the full canonical function using $G_{\mathbf{Q},\mathcal{V}}$. Our hope is that the latter almost never happens, and thus we can rely on a much smaller, and thus more efficient, canonical graph computation.

9.7.3 Results

We implemented a canonical function, highly optimized for the perfect form use-case. As a characteristic function we used $\mathcal{V}_{\text{ms}}(\mathbf{Q})$, which in the case of perfect forms is often just $\text{Min}(\mathbf{Q})$ and thus quite small. For the canonical graph function we implemented both `BLISS` [JK07] and `Traces` [MP14].

	Graph Variant		
Canonical Library	Signed	Absolute	speed-up
`BLISS` [JK07]	3558 ± 613 µs	398 ± 129 µs	8.94×
`Traces` [MP14]	601 ± 84 µs	223 ± 36 µs	2.70×

Table 9.1: The average number of microseconds and the standard deviation per full canonical form computation on the 524.288 9-dimensional perfect forms found by [SSV07]. The absolute graph timings include the check and fallback to the signed case if needed.

The timings for this highly optimized implementation, and the additional speed-up of the absolute graph improvement, are shown in Table 9.1. The low dimension allowed to implement most steps of the algorithm with primitive types (no extended precision), which gave a large speed-up over the original generic implementation. In addition, depending on the canonical graph library we obtain a speed-up of 2.7 to 8.94 using the absolute graph approach, even including the fallback to the original case if needed. The generic implementation from the original paper (using `Traces`) took an average of 5941µper canonical form on the same dataset and same hardware [DHVW20; Dut22]; in total we obtain a 27-fold speed-up over this implementation.

Exploring all perfect forms in dimension 9 requires roughly $2 \cdot 10^{11}$ canonical form computations. Based on the experiments, the extra improvement thus reduced the running time spend on this part of the algorithm from roughly 330.000 core-hours to less than 12.500 core-hours, making (that part of the) computation feasible on a high-end server or a small cluster.

Finishing the full enumeration is an ongoing joint project with Mathieu Dutour Sikirić. The canonical form function allowed us to find (probably) all perfect forms, with only a few exceptionally hard cases yet to treat.

Lemma 170. *There are at least* 2,237,251,033 *non-similar perfect forms in dimension* 9.

Conjecturally the above inequality is tight and we have found all perfect forms in dimension 9, of which the densest corresponds to the

laminated lattice Λ_9. To be certain about this we still have to enumerate all neighbours of a few perfect forms (with many neighbours).

The same idea, using the absolute graph canonical function, has been used to make the enumeration of all 55,083,358 C-type domains in dimension 6 feasible [DMW22].

CHAPTER 10

Remarkable Lattices & Cryptography

This chapter is based on the joint work 'On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography', with Léo Ducas, published at Eurocrypt 2022. The last section summarizes the follow-up joint work 'HAWK: Module LIP makes Lattice Signatures Fast, Compact and Simple', with Léo Ducas, Eamonn W. Postlethwaite and Ludo Pulles.

10.1 Introduction

The central idea in lattice-based cryptography is the concept of a bad basis acting as a public key, and a good basis of the same lattice acting as a secret key. With a good basis one can efficiently decode or Gaussian sample at low width, while with a bad basis this is hard. This discrepancy in hardness allows to create a backdoor that can be used for encryption schemes, signature schemes and more.

The creation of such a basis key-pair is precisely where the hardness assumptions such as NTRU [HPS98] and the Learning with Error (LWE) problem [Reg09] come in. They allow us to securely generate a somewhat random lattice along with a (partial) good basis. These assumptions are very versatile and allow to instantiate many different kinds of cryptographic schemes. However, from a decoding perspective, these good bases can only decode up to a radius $\Omega(\sqrt{n})$ from the optimal Minkowski bound, as they essentially reduce to the case of the trivial lattice $\mathbb{Z}^n$. Due to this non-optimal decoding distance we can break these schemes by solving SVP in some dimension $\beta \leq n/2+o(n)$, which forces the dimension to be at least 2 times larger than a direct SVP attack would imply.

In contrast, there do exist many remarkable lattices for which we can efficiently decode at much lower approximation factors. For example, Ducas and Pierrot [DP19] showed that the Chor-Rivest cryptosystem [CR88] was implicitly relying on a family of lattices for which it is possible to efficiently decode errors up to a radius within a factor of $O(\log n)$ from optimal. Recently, lattice families reaching a factor of $O(\sqrt{\log n})$ were found [MP21; BP22]. We cannot use such remarkable lattice directly, everyone, including an adversary, can decode in them. To use such remarkable lattices in cryptographic protocols we need a way to hide their remarkable structure, without destroying their decoding capabilities.

At repeated occasions [CR88; Len91; OTU00; SCM01; LLXY20], and over more than 30 years, exactly this has been tried, for example by porting the original public-key encryption scheme of McEliece [McE78] from codes to lattices: instead of giving a public bad basis of the lattice $\mathcal{L} \subset \mathbb{R}^n$ itself, one gives a bad basis of the permuted lattice $\pi(\mathcal{L}) := \{(y_{\pi^{-1}(i)})_i \in \mathbb{R}^n : \mathbf{y} \in \mathcal{L}\}$ for some secret random permutation π. One can efficiently decode in $\pi(\mathcal{L})$ via $\mathcal{L}$ (only) when π is known. Unfortunately this code-style way of hiding the lattice is also susceptible for code-style attacks, leading to subexponential attacks [CR88; Lap21; Odl90]; either by Information Set Decoding techniques or by guessing or brute-forcing some coordinates. Due to these attacks, this approach has not been very popular lately.

These attacks are enabled by the fact that these schemes do barely more than re-randomize the lattice by applying a permutation of the coordinates. Such permutations are isometries, i.e., lattice isomor-

phisms, but those are not the only ones. Since the isometry group $\mathcal{O}_n(\mathbb{R})$ acting on lattices is much larger than the one acting on codes, applying a random isometry from this larger group should convincingly thwart those code-style attacks: the canonical coordinate system becomes irrelevant.

All these remarks point toward the Lattice Isomorphism Problem (LIP) as a potential theoretical platform for finally getting this natural approach properly formalized, and hopefully, truly "lattice-based" in the cryptanalytic sense: the best known attack should be based on generic lattice reduction algorithms such as LLL [LLL82] and BKZ [Sch87]. As seen in Section 9.5 the current state of the art on LIP supports this hypothesis: all known algorithms [PP85; PS97; HR14; DHVW20] rely on finding short vectors. This is the case even for algorithms specialized to the trivial lattice $\mathbb{Z}^n$ [Szy03]. Experimental studies [BM21] do show that the basis randomization step requires care, in order to not introduce any weak instances.

While instantiating LIP with $\mathbb{Z}^n$ may already give rise to secure cryptosystems, the end goal of this work is to enable lattice-based cryptosystems that could be be even more secure than those based on LWE and SIS in similar dimensions, by instantiating the constructed schemes with remarkably decodable lattices.

10.1.1 Contributions

We introduce a formal and convenient framework for LIP-based cryptography, from which we build three cryptographic schemes: an identification scheme based on search-LIP (sLIP), a (passively secure) Key Encapsulation Mechanism (KEM) based on distinguish-LIP (ΔLIP), and a signature scheme also based on ΔLIP. In more detail:

- In Chapter 9 we discussed LIP, recalled the quadratic form formalism, and rephrased the LIP problem in terms of quadratic forms to conveniently avoid real numbers. Here, with the use of Gaussian Sampling [GPV08; Pei10], we define an average-case distribution for LIP and establish a worst-case to average-case reduction within an isomorphism class (Section 10.2). This addresses the concerns raised by Blanks and Miller [BM21] and formalizes their heuristic countermeasure.

- The above cryptographic foundations are directly inspired by the Zero-Knowledge proof of lattice non-isomorphism of Haviv and Regev [HR14]. We further extend on their techniques by proposing a Zero-Knowledge proof of knowledge (ZKPoK) of a lattice isomorphism (Section 10.3). This directly implies the existence of an identification scheme based on sLIP.

- We propose a KEM scheme (Section 10.4) and a hash-then-sign signature scheme (Section 10.5), both based on ΔLIP. Perhaps surprisingly, and unlike the original scheme of McEliece for codes, we circumvent the additional assumption that decoding a certain class of random lattices is hard. This is done via a lossyness argument [PW11] for the KEM, and a dual argument for the signature scheme.

- We review the state of the art for solving LIP (Section 10.6). In particular we note that all known algorithms require lattice reduction, and we quantify the required approximation factor.

- We discuss natural instantiations for each scheme given any remarkable lattice in Section 10.7. This section handles the construction of the auxiliary lattice appearing in ΔLIP for the lossyness arguments to get through.

10.1.2 Potential advantages

The KEM. To instantiate the KEM, consider a lattice $\mathcal{L}$ (w.l.o.g., of volume 1) such that:

- the minimal distance is within a factor f from Minkowski's bound: $\lambda_1(\mathcal{L}) \geq \Omega(\sqrt{n}/f)$,

- there exists an efficient algorithm that can decode errors in $\mathcal{L}$ up to radius $\rho < \lambda_1(\mathcal{L})/2$ within a factor f' from Minkowski's bound: $\rho \geq \Omega(\sqrt{n}/f')$.[1]

- the dual minimal distance is within a factor f^* from Minkowski's bound: $\lambda_1(\mathcal{L}^*) \geq \Omega(\sqrt{n}/f^*)$.

[1] Note that uniqueness of decoding implies $f' \geq 2f$.

Then, the instantiated KEM appears to resist concrete lattice reduction attacks down to an approximation factor $O(\max(f, f^*, f'))$. For a security reduction to ΔLIP we have to create a pair of lattices, which increases the approximation factor to $O(\max(f, f^*) \cdot f^* \cdot f')$. More specifically, it's security is based on ΔLIP for two lattices whose primal and dual first minima are all within a factor $O(\max(f, f^*) \cdot f^* \cdot f')$ from Minkowski's bound.

The trivial lattice $\mathbb{Z}^n$ gives all three factors f, f', f^* of the order $\Theta(\sqrt{n})$. The Barnes-Wall lattice improves all three factors down to $\Theta(\sqrt[4]{n})$ [MN08].

The endgame would be to instantiate with lattices for which all three factors f, f', f^* would be very small. In particular, one would naturally turn to recent work on decoding the Chor-Rivest lattices [CR88; DP19; LLXY20; Lap21] and the Barnes-Sloane lattices [MP21] giving $f = \text{polylog}(n)$ and $f' = \text{polylog}(n)$, but unfortunately their dual gaps are not that good: $f^* \geq \Theta(\sqrt{n})$. Indeed, both Chor-Rivest and Barnes-Sloane are integer lattices $\mathcal{L} \subset \mathbb{Z}^n$ with single exponential volume $\det(\mathcal{L}) = c^n$: their duals $\mathcal{L}^*$ have a Minkowski bound of $\Theta(\sqrt{n}/\det(\mathcal{L})^{1/n}) = \Theta(\sqrt{n})$, but contain all of $\mathbb{Z}^n \subset \mathcal{L}^*$, including vectors of norm 1.

Note, nevertheless that there is no geometric impossibility to the existence of the desired remarkably decodable lattice: random lattices have $f = O(1)$ and $f^* = O(1)$; so unique decoding is in principle possible down to $f' = f/2 = O(1)$. Unfortunately the the best known decoding algorithm for such lattices takes exponential time.

The signature scheme. The same principle also applies to our signature scheme, but this time with respect to Gaussian sampling rather than decoding: lattices with tight sampling (and large dual minimal distance) would lead to a scheme resisting attacks down to very small approximation factors. Unfortunately, even ignoring the constraint on the dual lattice, we do not know of any lattice much better than $\mathbb{Z}^n$ for efficient Gaussian sampling. However, instantiated with $\mathbb{Z}^n$ our scheme still has an interesting feature: not having to deal with any Gram-Schmidt or Cholesky matrices over the real numbers. This may be a worthy practical advantage over currently proposed hash-then-sign signature schemes [GPV08].

	Primal	Dual	Decoding
Decodable Lattice	f	f^*	f'
Random Lattice	$\Theta(1)$	$\Theta(1)$	$2^{\Theta(n)}$
$\mathbb{Z}^n$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$
NTRU [HPS98]	$\Omega(\alpha)$	$\Omega(\alpha)$	$\Omega(n/\alpha)$
LWE [Ajt99; AP11; MP12]	$\Omega(1)$	$\Omega(\alpha)$	$\Omega(n/\alpha)$
Prime Lattice [CR88; DP19]	$\Theta(\log n)$	$\Omega(\sqrt{n})$	$\Theta(\log n)$
Barnes-Sloane [MP21]	$\Theta(\sqrt{\log n})$	$\Omega(\sqrt{n})$	$\Theta(\sqrt{\log n})$
Reed-Solomon [BP22]	$\Theta(\sqrt{\log n})$	$\Omega(\sqrt{n})$	$\Theta(\sqrt{\log n})$
Barnes-Wall [MN08]	$\Theta(\sqrt[4]{n})$	$\Theta(\sqrt[4]{n})$	$\Theta(\sqrt[4]{n})$

Table 10.1: Some decodable lattices and their primal gap $f = \mathrm{gh}(\mathcal{L})/\lambda_1(\mathcal{L})$, dual gap $f^* = \mathrm{gh}(\mathcal{L}^*)/\lambda_1(\mathcal{L}^*)$ and efficient decoding gap $f' = \mathrm{gh}(\mathcal{L})/\rho$. We have $1 \leq \alpha \leq n$.

The identification scheme. Because sLIP seems super-exponentially hard in the dimension for well chosen lattices (i.e., that have a large kissing number), it might be secure to instantiate our ZKPoK with a rather small lattice dimension, maybe down to about a hundred (see the challenge in Table 10.2). This is more a theoretical curiosity than a practical advantage —the protocol still needs soundness amplification, where each round requires exchanging $\tilde{O}(n^2)$ bits.

10.1.3 Open questions

A KEM with polylog*-approximation factor security.* Is there any family of lattices that can be efficiently decoded within a polylog factor from Minkowski's bound such as [CR88; DP19; LLXY20; Lap21; MP21], but whose dual would also have an equally large minimal distance?

Tight Gaussian Sampling for signatures. Is there any family of lattices $\mathcal{L}$ (of volume 1) in which one can efficiently sample a discrete Gaus-

sian with small parameter $\sigma < o(\sqrt{n})$, if not $\sigma = \text{polylog}(n)$ (with exponential smoothing $\sigma > \eta_{2^{-n}}(\mathcal{L})$)? And if so, do they and their dual have a large minimal distance? Note that quantumly, this question is related to the previous one via the reduction of Regev [Reg09]: decoding in the primal for a large radius gives Gaussian sampling in the dual for a small width. But a classical algorithm would be much preferable.

Concrete instantiation with simple lattices. Instantiated with $\mathbb{Z}^n$, our signature scheme has the advantage of not requiring any Gram-Schmidt or Cholesky decomposition, contrary to existing hash-then-sign signature schemes, and may therefore be of practical interest. It could also be reasonable to instantiate our KEM with the lattice of Barnes and Wall, by using the decoder of Micciancio and Nicolesi [MN08].

Module-LIP. Lastly, it also seems natural to explore structured variants of LIP, where both the lattice and the isometry should be structured. We note that for every ideal lattice in complex-multiplication number fields, a classical polynomial time algorithm for LIP is known [GS02; LS14]. Could the module variant be secure? Can our constructions gain a linear factor on key sizes from such a variant? And are there remarkably decodable lattices that are also modules over certain number fields? For example the repeated-difference lattices (Craig's lattices [CS13]) are ideal lattices in cyclotomic number fields with large minimal distances, but a polynomial decoding algorithm for them remains to be discovered.

Remark 171. The last two open questions were partially answered positively in a follow-up work [DPPW22]. In this work we build a competitive signature scheme, named Hawk, based on the simple lattice $\mathbb{Z}^n$, instantiated as a rank 2 module. In Section 10.8 we give a short summary of this work, and show that it improves upon Falcon in several ways.

10.2 LIP and self-reducibility

In this section we lay the foundation for using the Lattice Isomorphism Problem in cryptography. We present an average-case distribution for any quadratic form equivalence class, show how to sample from it, and conclude with a worst-case to average-case reduction. Note that the worst-case to average-case reduction is realized *within* an equivalence class $[\mathbf{Q}] = \{\mathbf{U}^\top \mathbf{Q}\mathbf{U} : \mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})\}$. But first we start with some adaptation of discrete Gaussian sampling to the quadratic form setting.

10.2.1 Discrete Gaussians and sampling

Discrete Gaussian sampling has been fundamental to the development of lattice based cryptography, by allowing to return short or nearby lattice vectors without leaking information about the secret key [GPV08]. We rephrase the relevant definitions and propositions in the quadratic form language.

Distribution. For any quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}$ we define the Gaussian function on $\mathbb{R}^n$ with parameter $s > 0$ and center $\mathbf{c} \in \mathbb{R}^n$ by

$$\forall \mathbf{x} \in \mathbb{R}^n, \rho_{\mathbf{Q},s,\mathbf{c}}(\mathbf{x}) := \exp(-\pi \left\|\mathbf{x} - \mathbf{c}\right\|_{\mathbf{Q}}^2 / s^2).$$

The discrete Gaussian distribution is obtained by restricting the continuous Gaussian distribution to a discrete lattice. In the quadratic form setting the discrete lattice will always be $\mathbb{Z}^n$, but with the geometry induced by the quadratic form. For any quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}$ we define the discrete Gaussian distribution $\mathcal{D}_{\mathbf{Q},s,\mathbf{c}}$ with center $\mathbf{c} \in \mathbb{R}^n$ and parameter $s > 0$ by

$$\Pr_{X \sim \mathcal{D}_{\mathbf{Q},s,\mathbf{c}}}[X = \mathbf{x}] := \frac{\rho_{\mathbf{Q},s,\mathbf{c}}(\mathbf{x})}{\rho_{\mathbf{Q},s,\mathbf{c}}(\mathbb{Z}^n)} \text{ if } \mathbf{x} \in \mathbb{Z}^n, \text{ and 0 otherwise.}$$

If the center $\mathbf{c}$ is not denoted we have $\mathbf{c} = \mathbf{0}$. An important property of the discrete gaussian distribution is the smoothing parameter, i.e., how much gaussian noise $s > 0$ is needed to 'smooth out' the discrete structure.

Definition 172 (Smoothing Parameter). *For a quadratic form* $\mathbf{Q} \in \mathcal{S}_n^{>0}$ *and* $\epsilon > 0$ *we define the smoothing parameter* $\eta_\epsilon(\mathbf{Q})$ *as the minimal* $s > 0$ *such that* $\rho_{\mathbf{Q}^{-1},1/s}(\mathbb{Z}^n) \leq 1 + \epsilon$.

The smoothing parameter is a central quantity for gaussians over lattice, for example it permits to control the variations of $\rho_{\mathbf{Q},s,\mathbf{c}}(\mathbb{Z}^n)$ is over all centers $\mathbf{c}$.

Lemma 173 ([MR07]). *For any quadratic form* $\mathbf{Q} \in \mathcal{S}_n^{>0}$, $\epsilon > 0$, *center* $\mathbf{c} \in \mathbb{R}^n$ *and parameter* $s > \eta_\epsilon(\mathbf{Q})$ *we have:*

$$(1-\epsilon)\frac{s^n}{\sqrt{\det(\mathbf{Q})}} \leq \rho_{\mathbf{Q},s,\mathbf{c}}(\mathbb{Z}^n) \leq (1+\epsilon)\frac{s^n}{\sqrt{\det(\mathbf{Q})}}.$$

Note that the smoothing parameter $\eta_\epsilon(\mathbf{Q})$ is an invariant property of the similarity class $[\mathbf{Q}]$, and so we might also denote $\eta_\epsilon([\mathbf{Q}])$ for a similarity class. While computing or even approximating the exact smoothing parameter is hard, we can obtain sufficient bounds via the dual form.

Lemma 174 (Smoothing bound [MR07]). *For any* $\epsilon > 0$ *and any quadratic form* $\mathbf{Q} \in \mathcal{S}_n^{>0}$ *we have* $\eta_{2^{-n}}(\mathbf{Q}) \leq \sqrt{n}/\lambda_1(\mathbf{Q}^{-1})$, *and*

$$\eta_\epsilon(\mathbf{Q}) \leq \|\tilde{\mathbf{B}}_{\mathbf{Q}}\| \cdot \sqrt{\ln(2n(1+1/\epsilon))/\pi}.$$

Above the smoothing parameter the discrete gaussian distribution is in some sense 'well behaved' and we have the following tailbound that one would expect from a Gaussian distribution.

Lemma 175 (Tailbound [MR07, Lemma 4.4]). *For any quadratic form* $\mathbf{Q} \in \mathcal{S}_n^{>0}$, $\epsilon \in (0,1)$, *center* $\mathbf{c} \in \mathbb{R}^n$ *and parameter* $s \geq \eta_\epsilon(\mathbf{Q})$, *we have*

$$\Pr_{\mathbf{x}\sim\mathcal{D}_{\mathbf{Q},s,\mathbf{c}}}[\|\mathbf{x}-\mathbf{c}\|_{\mathbf{Q}} > s\sqrt{n}] \leq \frac{1+\epsilon}{1-\epsilon}\cdot 2^{-n}.$$

A constant factor above the smoothing parameter we can furthermore lower bound the min-entropy of the distribution.

Lemma 176 (Min-entropy [PR06]). *For any quadratic form* $\mathbf{Q} \in \mathcal{S}_n^{>0}$, *positive* $\epsilon > 0$, *center* $\mathbf{c} \in \mathbb{R}^n$, *parameter* $s \geq 2\eta_\epsilon(\mathbf{Q})$, *and for every* $\mathbf{x} \in \mathbb{Z}^n$, *we have*

$$\Pr_{X\sim\mathcal{D}_{\mathbf{Q},s,\mathbf{c}}}[X = \mathbf{x}] \leq \frac{1+\epsilon}{1-\epsilon}\cdot 2^{-n}.$$

Gaussian Sampling. While the discrete Gaussian distribution already is an important theoretical tool, for many practical purposes we want to actually sample (close to) the distribution in an efficient manner. In their breakthrough work Gentry et al. [GPV08] showed that Klein's [Kle00] randomized Babai's nearest plane algorithm does exactly that. Given a lattice basis one can sample statistically close to the discrete Gaussian distribution with parameters depending on the shortness of the (Gram-Schmidt) basis; a better reduced basis allows for a lower Gaussian width s. To simplify later proofs we use an exact sampling algorithm by Brakerski et al. [Bra+13].

Lemma 177 (Discrete Sampling [Bra+13, Lemma 2.3]). *There is a polynomial-time algorithm* $\mathbf{DiscreteSample}(\mathbf{Q}, s, \mathbf{c})$ *that given a quadratic form* $\mathbf{Q} \in \mathcal{S}_n^{>0}$*, center* $\mathbf{c} \in \mathbb{R}^n$*, and a parameter* $s \geq \|\tilde{\mathbf{B}}_{\mathbf{Q}}\| \cdot \sqrt{\ln(2n+4)/\pi}$*, returns a sample distributed as* $\mathcal{D}_{\mathbf{Q},s,\mathbf{c}}$*.*

10.2.2 An average-case distribution

First, we define our average-case distribution within an equivalence class $[\mathbf{Q}]$, which can be seen as an extension of the techniques used by Haviv and Regev [HR14] to show that LIP lies in SZK. While in their work they use a discrete Gaussian sampler [GPV08] to sample a generating set of the lattice, we extend this by a linear algebra step that returns a canonically distributed lattice basis — or, in our case, a quadratic form.

In hindsight, this algorithm appears very similar to the heuristic approach of [BM21], but the use of Gaussian sampling formally guarantees that the output distribution solely depends on the lattice and not on the specific input basis — or, in our case, depends only on the class of the input quadratic form.

We start with the linear algebra step, that, given a quadratic form and a set of short vectors of full rank, returns a well-reduced equivalent form.

Lemma 178 (Adapted from [MG02, Lemma 7.1]). *There is a polynomial time algorithm* $(\mathbf{R}, \mathbf{U}) \leftarrow \mathbf{Extract}(\mathbf{Q}, \mathbf{Y})$ *that on input a quadratic form* $\mathbf{Q}$*, and vectors* $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_m) \in \mathbb{Z}^{n \times m}$ *such that*

$\mathrm{rk}(\mathcal{L}(\mathbf{Y})) = n$, *outputs a transformation* $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ *and a quadratic form* $\mathbf{R} = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ *equivalent to* $\mathbf{Q}$ *such that* $\left\|\tilde{\mathbf{B}}_{\mathbf{R}}\right\| \leq \max_i \|\mathbf{y}_i\|_{\mathbf{Q}}$.

Proof. First let $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ be the unique transformation such that $\mathbf{T} = \mathbf{U}^{-1}\mathbf{Y}$ is the canonical upper-diagonal Hermite Normal Form of $\mathbf{Y}$. Let $\mathbf{R} = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$ and note that $\mathbf{R}$ is equivalent to $\mathbf{Q}$. Denote the column vectors of $\mathbf{U}$ by $\mathbf{u}_1, \ldots, \mathbf{u}_n$. Let $p_1, \ldots, p_n$ be the pivot columns of $\mathbf{T}$. Because $\mathbf{T}$ is upper triangular and in Hermite Normal Form we have $\mathbf{y}_{p_i} = \sum_{j=1}^{i} \mathbf{T}_{j,p_i} \mathbf{u}_j$, where $\mathbf{T}_{i,p_i} \geq 1$. In particular we have that $\mathrm{span}(\mathbf{y}_{p_i}, \ldots, \mathbf{y}_{p_k}) = \mathrm{span}(\mathbf{u}_1, \ldots, \mathbf{u}_k)$. Let $\tilde{\mathbf{y}}_{p_i}$ and $\tilde{\mathbf{u}}_i$ be the i-th Gram-Schmidt vector of $(\mathbf{y}_{p_1}, \ldots, \mathbf{y}_{p_n})$ and $\mathbf{U}$ respectively w.r.t. $\mathbf{Q}$. Note that $\tilde{\mathbf{y}}_{p_i} = \mathbf{T}_{i,p_i} \cdot \tilde{\mathbf{u}}_i$, and thus $\|\tilde{\mathbf{u}}_i\|_{\mathbf{Q}} = \|\tilde{\mathbf{y}}_{p_i}\|_{\mathbf{Q}} / \mathbf{T}_{i,p_i} \leq \|\tilde{\mathbf{y}}_{p_i}\|_{\mathbf{Q}} \leq \|\mathbf{y}_{p_i}\|_{\mathbf{Q}}$. We conclude by $\left\|\tilde{\mathbf{B}}_{\mathbf{R}}\right\| = \max_i \|\tilde{\mathbf{u}}_i\|_{\mathbf{Q}} \leq \max_i \|\mathbf{y}_i\|_{\mathbf{Q}}$. □

For our final distribution to be well-defined we need that the extracted quadratic form only depends on the geometry of the input vectors, and not on the particular representative $\mathbf{Q}$.

Lemma 179. *Let* $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_m) \in \mathbb{Z}^{n \times m}$ *have full rank* n. *If* $(\mathbf{R}, \mathbf{U}) \leftarrow \mathbf{Extract}(\mathbf{Q}, \mathbf{Y})$, *and for some unimodular* $\mathbf{V} \in \mathcal{GL}_n(\mathbb{Z})$ *we have* $(\mathbf{R}', \mathbf{U}') \leftarrow \mathbf{Extract}(\mathbf{V}^\top \mathbf{Q} \mathbf{V}, \mathbf{V}^{-1}\mathbf{Y})$, *then* $\mathbf{R}' = \mathbf{R}$, *and* $\mathbf{U}' = \mathbf{V}^{-1} \cdot \mathbf{U}$.

Proof. From the canonicity of the Hermite Normal Form we immediately obtain that $(\mathbf{U}')^{-1}\mathbf{V}^{-1}\mathbf{Y} = \mathbf{T} = \mathbf{U}^{-1}\mathbf{Y}$, and thus $\mathbf{U}' = \mathbf{V}^{-1} \cdot \mathbf{U}$. It follows that $\mathbf{R}' = (\mathbf{V}^{-1} \cdot \mathbf{U})^\top \mathbf{V}^\top \mathbf{Q} \mathbf{V} (\mathbf{V}^{-1} \cdot \mathbf{U}) = \mathbf{U}^\top \mathbf{Q} \mathbf{U} = \mathbf{R}$. □

Now we can define our average-case distribution for a Gaussian parameter $s > 0$.

Definition 180. *Given a quadratic form* $\mathbf{Q} \in \mathcal{S}_n^{>0}$, *we define the Gaussian form distribution* $\mathcal{D}_s([\mathbf{Q}])$ *over* $[\mathbf{Q}]$ *with parameter* $s > 0$ *algorithmically as follows:*

1. *Let* $C := 1 - (1 + e^{-\pi})^{-1} > 0$, *and* $m := \left\lceil \frac{2n}{C} \right\rceil$. *Sample* m *vectors* $(\mathbf{y}_1, \ldots, \mathbf{y}_m) =: \mathbf{Y}$ *from* $\mathcal{D}_{\mathbf{Q},s}$. *Repeat until their span has full rank* n.
2. $(\mathbf{R}, \mathbf{U}) \leftarrow \mathbf{Extract}(\mathbf{Q}, \mathbf{Y})$.
3. *Return* $\mathbf{R}$.

Proof. We have to show that the distribution is well-defined over the equivalence class $[\mathbf{Q}]$, i.e., for any input representative $\mathbf{Q}' \in [\mathbf{Q}]$ the output distribution should be identical. Let $\mathbf{Q}' = \mathbf{V}^\top \mathbf{Q}\mathbf{V} \in [\mathbf{Q}]$. Note that step 1 only depends on the geometry of the vectors w.r.t. $\mathbf{Q}'$. Therefore if step 1 on input $\mathbf{Q}$ finishes with $\mathbf{Y}$, then on input $\mathbf{Q}'$ it finishes step 1 with $\mathbf{Y}' = \mathbf{V}^{-1}\mathbf{Y}$ with the same probability. By Lemma 179 step 2 extracts the same quadratic form $\mathbf{R}$ in both cases. So the distribution is independent of the input representative $\mathbf{Q}' \in [\mathbf{Q}]$, and thus it is well-defined over $[\mathbf{Q}]$. □

Given the algorithmic definition of $\mathcal{D}_s([\mathbf{Q}])$, an efficient sampling algorithm follows with only a few adaptations. Firstly, we need to efficiently sample from $\mathcal{D}_{\mathbf{Q},s}$ which puts some constraints on the parameter s depending on the reducedness of the representative $\mathbf{Q}$. Secondly, we need to show that step 1 does not have to be repeated very often. For this we require the additional constraint $s \geq \lambda_n(\mathbf{Q})$.

Algorithm 16: Sampling from $\mathcal{D}_s([\mathbf{Q}])$.

Data: A quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z}^n)$, and a parameter $s \geq \max\{\lambda_n(\mathbf{Q}), \|\tilde{\mathbf{B}}_\mathbf{Q}\| \cdot \sqrt{\ln(2n+4)/\pi}\}$.
Result: Sample $\mathbf{R} = \mathbf{U}^\top \mathbf{Q}\mathbf{U}$ from $\mathcal{D}_s([\mathbf{Q}])$, with a transformation $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$.

1 $C \leftarrow 1 - (1 + e^{-\pi})^{-1}$, $m \leftarrow \lceil \frac{2n}{C} \rceil$;
2 **do**
3 Sample $\mathbf{y}_1, \ldots, \mathbf{y}_m \leftarrow \mathcal{D}_{\mathbf{Q},s}$; // Using Lemma 177
4 $\mathbf{Y} \leftarrow (\mathbf{y}_1, \ldots, \mathbf{y}_m)$;
5 **while** $\mathrm{rk}(\mathbf{Y}) < n$;
6 $(R, U) \leftarrow$ **Extract**$(\mathbf{Q}, \mathbf{Y})$;

Lemma 181. *For any quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z})$, and parameter*

$$s \geq \max\{\lambda_n(\mathbf{Q}), \|\tilde{\mathbf{B}}_\mathbf{Q}\| \cdot \sqrt{\ln(2n+4)/\pi}\},$$

Algorithm 16 runs in expected polynomial time and returns $(\mathbf{R}, \mathbf{U}) \in [\mathbf{Q}] \times \mathcal{GL}_n(\mathbb{Z})$, where $\mathbf{R}$ is a sample from $\mathcal{D}_s([\mathbf{Q}])$, and $\mathbf{U}$, conditioned on $\mathbf{R}$, is uniform over $\mathrm{Isom}(\mathbf{Q}, \mathbf{R})$. In particular $\mathbf{R} = \mathbf{U}^\top \mathbf{Q}\mathbf{U}$.

Proof. By Lemmas 177 and 178 every step in Algorithm 16 runs in polynomial time. What remains is to show that the number of iterations is polynomially bounded in expectation. Let the random variable T be the number of iterations before a set of full rank vectors is found, we have to bound $\mathbb{E}[T]$.

If $\mathrm{rk}(\mathbf{y}_1, \ldots, \mathbf{y}_i) < n$, then because $s \geq \lambda_n(\mathbf{Q})$ we have by [HR14, Lemma 5.1] that every newly sampled vector $\mathbf{y}_{i+1} \leftarrow \mathcal{D}_{\mathbf{Q},s}$ is not in the span of $\mathbf{y}_1, \ldots, \mathbf{y}_i$ with constant probability at least $C := 1 - (1 + e^{-\pi})^{-1} > 0$. Let $m := \left\lceil \frac{2n}{C} \right\rceil$. The failure probability p_{fail} of not finding n linearly independent vectors is upper bounded by a binomial experiment with success probability C, where we reach at most $n-1$ wins in m trials. By Hoeffding's inequality we have the tail bound $p_{\text{fail}} \leq \exp(-2m \cdot (C - \frac{n-1}{m})^2) \leq \exp(-mC) \leq \exp(-2n) \leq e^{-2}$. The expected number of iterations $\mathbb{E}[T]$ is then bounded by the mean of a geometric distribution with success probability $1 - p_{\text{fail}}$, i.e.,

$$\mathbb{E}[T] \leq 1/(1 - p_{\text{fail}}) \leq 1/(1 - e^{-2}) < 2.$$

Suppose that the algorithm runs and finishes with a full rank matrix $\mathbf{Y}$, and returns $(\mathbf{R}, \mathbf{U}) \leftarrow \mathbf{Extract}(\mathbf{Q}, \mathbf{Y})$. For any automorphism $\mathbf{V} \in \mathrm{Aut}(\mathbf{Q})$, i.e., such that $\mathbf{V}^\top \mathbf{Q} \mathbf{V} = \mathbf{Q}$, it would have been just as likely that the final full rank matrix equalled $\mathbf{VY}$, because the samples from $\mathcal{D}_{\mathbf{Q},s}$ and the stopping condition only depend on the geometry of the vectors w.r.t. $\mathbf{Q}$. Then, by Lemma 179, we have:

$$\mathbf{Extract}(\mathbf{Q}, \mathbf{VY}) = \mathbf{Extract}((\mathbf{V}^{-1})^\top \mathbf{Q} \mathbf{V}^{-1}, \mathbf{VY}) = (\mathbf{R}, \mathbf{VU}),$$

and thus the algorithm would have returned $\mathbf{VU}$ with the same probability as $\mathbf{U}$, which makes the returned transformation uniform over the set of isomorphisms $\{\mathbf{VU} : \mathbf{V} \in \mathrm{Aut}(\mathbf{Q})\}$ from $\mathbf{Q}$ to $\mathbf{R}$. □

For (exponentially) large parameters s we can always efficiently sample from the average-case distribution by first LLL reducing the representative.

Lemma 182. *Given any quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ we can sample from $\mathcal{D}_s([\mathbf{Q}])$ (together with a transformation) in polynomial time for $s \geq 2^{\Theta(n)} \cdot \lambda_n([\mathbf{Q}])$.*

Proof. Run the LLL algorithm on $\mathbf{Q}$ to obtain a representative $\mathbf{Q}' \in [\mathbf{Q}]$ for which $\|\tilde{\mathbf{B}}_{\mathbf{Q}'}\| \leq 2^{\Theta(n)} \cdot \lambda_n([\mathbf{Q}])$. Then apply Lemma 181 on $\mathbf{Q}'$.

For a sample $\mathbf{Q}''$, combining the unimodular transformations obtained from LLL and the sampling gives us a unimodular $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}'' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$. □

Lemma 183. *For any quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}$, parameter $\epsilon \in (0,1)$, and $s \geq \max\{\lambda_n(\mathbf{Q}), \eta_\epsilon(\mathbf{Q})\}$, we have*

$$\Pr_{\mathbf{Q}' \sim \mathcal{D}_s([\mathbf{Q}])}[\|\tilde{\mathbf{B}}_{\mathbf{Q}'}\| > s\sqrt{n}] \leq \frac{1+\epsilon}{1-\epsilon} \cdot 100n \cdot 2^{-n}.$$

Proof. Given full rank vectors $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_m) \in \mathbb{Z}^{n \times m}$ the extractor returns a quadratic form $\mathbf{Q}'$ such that $\|\tilde{\mathbf{B}}_{\mathbf{Q}'}\| \leq \max_i \|\mathbf{y}_i\|_{\mathbf{Q}}$ and thus we can just focus on the norms $\|\mathbf{y}_i\|_{\mathbf{Q}}$ of the sampled vectors. Let the random variable $T \geq 1$ be the number of iterations before a set of full rank vectors is found. From the proof of Lemma 181 we have $\mathbb{E}[T] \leq 2$. After t iterations we have sampled $t \cdot m$ vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{t \cdot m}$, and by Lemma 175 we have $\|\mathbf{x}_i\| > s\sqrt{n}$ with probability at most $(1+\epsilon)/(1-\epsilon) \cdot 2^{-n}$ for each of them. By a union bound we conclude

$$\begin{aligned}
\Pr\left[\max_{1 \leq i \leq T \cdot m} \|\mathbf{y}_i\|_{\mathbf{Q}} > s\sqrt{n}\right] &= \sum_{t=1}^{\infty} \Pr[T = t] \cdot \Pr[\max_{1 \leq i \leq t \cdot m} \|\mathbf{y}_i\|_{\mathbf{Q}} > s\sqrt{n}] \\
&\leq \underbrace{\sum_{t=1}^{\infty} \Pr[T = t] \cdot t}_{\mathbb{E}[T]} \cdot m \cdot \frac{1+\epsilon}{1-\epsilon} \cdot 2^{-n} \\
&\leq 2 \cdot \left\lceil \tfrac{2n}{C} \right\rceil \cdot \frac{1+\epsilon}{1-\epsilon} \cdot 2^{-n} \leq 100n \cdot \frac{1+\epsilon}{1-\epsilon} \cdot 2^{-n}.
\end{aligned}$$

□

10.2.3 Average case LIP

The above definition of a distribution over a class which is efficiently sampleable from any representative of that class leads us to a natural average-case version of both versions of LIP. It is parametrized by a width parameter $s > 0$.

Definition 184 (Average-case search LIP: ac-sLIP$_s^{\mathbf{Q}}$). *For $s > 0$ and a quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}$, the problem* ac-sLIP$_s^{\mathbf{Q}}$ *is, given a quadratic form sampled as $\mathbf{Q}' \leftarrow \mathcal{D}_s([\mathbf{Q}])$, to compute a unimodular $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}' = \mathbf{U}^\top \mathbf{Q} \mathbf{U}$.*

Definition 185 (Average-case distinguishing LIP: ac-$\Delta\text{LIP}_s^{\mathbf{Q}_0,\mathbf{Q}_1}$). *For two quadratic forms $\mathbf{Q}_0, \mathbf{Q}_1 \in \mathcal{S}_n^{>0}$ and parameter $s > 0$ the problem* ac-$\Delta\text{LIP}_s^{\mathbf{Q}_0,\mathbf{Q}_1}$ *is, given a quadratic form sampled as $\mathbf{Q}' \leftarrow \mathcal{D}_s([\mathbf{Q}_b])$ where $b \in \{0,1\}$ is a uniform random bit, to recover b.*

Trivially the average-case variants can be reduced to their respective worst-case variants. In the following section we show that the reverse is also true.

10.2.4 A worst-case to average-case reduction

In general, lattice problems become easier when given a short basis and harder when given a long basis. Similarly, one would expect that ac-$\text{sLIP}_s^{\mathbf{Q}}$ and ac-$\Delta\text{LIP}_s^{\mathbf{Q}_0,\mathbf{Q}_1}$ become harder when the parameter $s > 0$ increases. In fact, when s is large enough, the average-case problem becomes at least as hard as any worst-case instance, making the average-case and worst-case problems equivalent.

Lemma 186 (ac-$\text{sLIP}_s^{\mathbf{Q}} \geq$ wc-$\text{sLIP}^{\mathbf{Q}}$ for large s). *Given an oracle that solves* ac-$\text{sLIP}_s^{\mathbf{Q}}$ *for some $s \geq 2^{\Theta(n)} \cdot \lambda_n(\mathbf{Q})$ in time T_0 with probability $\epsilon > 0$, we can solve* wc-$\text{sLIP}^{\mathbf{Q}}$ *with probability at least ϵ in time $T = T_0 + poly(n, \log s)$.*

Proof. Given any (worst-case) instance $\mathbf{Q}' \in [\mathbf{Q}]$, apply Lemma 182 to sample $\mathbf{Q}'' \leftarrow \mathcal{D}_s([\mathbf{Q}])$ for some $s \geq 2^{O(n)} \cdot \lambda_n([\mathbf{Q}])$, together with a $\mathbf{U}''$ such that $\mathbf{Q}'' = \mathbf{U}''^{\top}\mathbf{Q}'\mathbf{U}''$. Note that $\mathcal{D}_s([\mathbf{Q}]) = \mathcal{D}_s([\mathbf{Q}'])$; we can therefore apply our ac-$\text{sLIP}_s^{\mathbf{Q}}$-oracle once to $\mathbf{Q}''$ and obtain $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}'' = \mathbf{U}^{\top}\mathbf{Q}\mathbf{U}$. Now for $\mathbf{U}' := \mathbf{U}\mathbf{U}''^{-1} \in \mathcal{GL}_n(\mathbb{Z})$ we have:

$$\mathbf{U}'^{\top}\mathbf{Q}\mathbf{U}' = (\mathbf{U}''^{-1})^{\top}\mathbf{U}^{\top}\mathbf{Q}\mathbf{U}\mathbf{U}''^{-1} = (\mathbf{U}''^{-1})^{\top}\mathbf{Q}''\mathbf{U}''^{-1} = \mathbf{Q}'.$$

So given an ac-$\text{sLIP}_s^{\mathbf{Q}}$-oracle we can solve wc-$\text{sLIP}^{\mathbf{Q}}$. □

To allow for more efficient schemes we would like to decrease the parameter $s > 0$ in the worst-case to average-case reduction. We can do so at the cost of a stronger lattice reduction algorithm than LLL.

Lemma 187. *Given an oracle that solves* ac-sLIP$_s^{\mathbf{Q}}$ *for some* $s \geq \lambda_n(\mathbf{Q})$ *in time* T_0 *with probability* $\epsilon > 0$*, we can solve* wc-sLIP$^{\mathbf{Q}}$ *with probability at least* $\frac{1}{2}$ *in time*

$$T = \frac{1}{\epsilon}(T_0 + \mathrm{poly}(n, \log s)) + C\left(n, \frac{s}{\lambda_n(\mathbf{Q}) \cdot \sqrt{\ln(2n+4)/\pi}}\right),$$

where $C(n, f)$ *is the cost of solving the Shortest Independent Vector Problem in dimension* n *(SIVP, [Reg09]) within an approximation factor of* f*.*

Proof. Since the f-approx-SIVP oracle returns n linearly independent vectors of norm at most $f \cdot \lambda_n(\mathbf{Q})$, we can construct an equivalent form $\mathbf{Q}' \in [\mathbf{Q}]$ with $\|\tilde{\mathbf{B}}_{\mathbf{Q}'}\| \leq f \cdot \lambda_n(\mathbf{Q})$, using Lemma 178. We do this once at cost $C(n, f)$. For $f := s/(\lambda_n(\mathbf{Q}) \cdot \sqrt{\ln(2n+4)/\pi})$ we obtain that $s \geq \|\tilde{\mathbf{B}}_{\mathbf{Q}'}\| \cdot \sqrt{\ln(2n+4)/\pi}$. Therefore using $\mathbf{Q}'$ we can sample efficiently from $\mathcal{D}_s([\mathbf{Q}])$. The rest of the proof is similar to that of Lemma 186. Additionally the reduction succeeds with some probability $\epsilon > 0$, so we need to repeat it $\frac{1}{\epsilon}$ times to obtain a success probability of at least $\frac{1}{2}$. Note that each additional sample can be computed in polynomial time from the same representative $\mathbf{Q}'$. □

Remark 188. Note that the overhead is entirely additive, in particular it does not suffer from the $\frac{1}{\epsilon}$ amplification. So, despite the reduction not being polynomial time, one can still afford extraordinary large overheads. Concretely, for example, a hardness of 2^{100} for f-SIVP would barely affect the hardness reduction, if the hardness of ac-sLIP is 2^{128}. This situation is quite different from the usual inefficient reductions found in the literature, where the overhead is multiplicative.

In Lemma 187, the SIVP oracle can be instantiated by a variant of the BKZ algorithm [Sch87]. With a sub-linear blocksize of $\beta := n/\log(n)$ we could decrease s to a quasi-polynomial factor $\exp(\log^2(n)) \cdot \lambda_n(\mathbf{Q})$, with only a subexponential additive cost to the reduction. For security based on exponential hardness (e.g., $T_0/\epsilon = \exp(\Omega(n))$) this would still be meaningful, while the lower parameters imply a more efficient scheme with only a poly-logarithmic bitlength for the integer entries of the manipulated matrices.

Going down to polynomial factors $s = \mathrm{poly}(n) \cdot \lambda_n(\mathbf{Q})$ (and hence single logarithmic integer bitlength) would require a linear blocksize $\beta := \Theta(n)$, and an exponential cost 2^{cn}. For small constants $c > 0$ such that cn is smaller than the security parameter the reduction would still be meaningful. However, for provable algorithms this constant c is generally too large to be useful. As we want to keep our reduction non-heuristic in this initial work, we will leave this regime for further research.

Using a similar strategy, one can also establish a worst-case to average-case reduction for ΔLIP. Note that, because it is a distinguishing problem, the advantage amplification requires $O(1/\alpha^2)$ calls to the average-case oracle.

Lemma 189 (ac-$\Delta\mathrm{LIP}_s^{\mathbf{Q}_0,\mathbf{Q}_1} \geq$ wc-$\Delta\mathrm{LIP}^{\mathbf{Q}_0,\mathbf{Q}_1}$ for large s). *Given an oracle that solves* ac-$\Delta\mathrm{LIP}_s^{\mathbf{Q}_0,\mathbf{Q}_1}$ *for some*

$$s \geq 2^{\Theta(n)} \cdot \max\{\lambda_n(\mathbf{Q}_0), \lambda_n(\mathbf{Q}_1)\}$$

in time T_0 with advantage $\alpha > 0$, we can solve wc-$\Delta\mathrm{LIP}^{\mathbf{Q}_0,\mathbf{Q}_1}$ *with advantage α in time $T = T_0 + poly(n, \log s)$.*

Lemma 190. *Given an oracle that solves* ac-$\Delta\mathrm{LIP}_s^{\mathbf{Q}_0,\mathbf{Q}_1}$ *in time T_0 for some $s \geq \max\{\lambda_n(\mathbf{Q}_0), \lambda_n(\mathbf{Q}_1)\}$ with advantage $\alpha > 0$, we can solve* wc-$\Delta\mathrm{LIP}^{\mathbf{Q}_0,\mathbf{Q}_1}$ *with advantage at least $\frac{1}{4}$ in time*

$$\begin{aligned} T =& \frac{1}{\alpha^2}(T_0 + \mathrm{poly}(n, \log s)) \\ &+ C\left(n, \frac{s}{\max\{\lambda_n(\mathbf{Q}_0), \lambda_n(\mathbf{Q}_1)\} \cdot \sqrt{\ln(2n+4)/\pi}}\right), \end{aligned}$$

where $C(n, f)$ is the cost of solving the Shortest Independent Vector Problem in dimension n (SIVP, [Reg09]) within an approximation factor of f.

10.3 Zero Knowledge Proof of Knowledge

Recall from Section 2.7.2 that a ZKPoK protocol allows one to prove knowledge of a solution to some problem without revealing the solution. In our case, given public quadratic forms $\mathbf{Q}_0, \mathbf{Q}_1 \in \mathcal{S}_n^{>0}(\mathbb{Z})$

we want to show knowledge of a LIP solution $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}_1 = \mathbf{U}^\top \mathbf{Q}_0 \mathbf{U}$, without revealing any information about $\mathbf{U}$. The protocol is defined in Figure 10.1.

On a high level, the protocol of Haviv and Regev [HR14], as well as ours, is very similar to protocols for other types of isomorphisms, in particular protocols for graph isomorphism [GMW91] and for code isomorphism [BMPS20].

A notable difference, however, is that both these type of protocols [GMW91; BMPS20] rely on the action of a finite group (permutations), allowing to show zero-knowledgness by uniformity of the distribution over an orbit. Since in our case, the group acting $\mathcal{GL}_n(\mathbb{Z})$ is not finite, not even compact, it cannot admit such uniform distribution. It is perhaps surprising to see that uniformity is in fact not required; our earlier defined average-case distribution can be used instead.

10.3.1 The Σ-protocol

Besides efficiency we have to check the standard properties of a sigma protocol as defined in Section 2.7.2: completeness, special soundness and honest-verifier zero-knowledge.

Efficiency and completeness. To show the efficiency of the prover we have to check that Algorithm 16 runs in polynomial time. Indeed, by Lemma 181, this is the case because

$$s \geq \max\left\{\lambda_n([\mathbf{Q}_0]), \|\tilde{\mathbf{B}}_{\mathbf{Q}_0}\| \cdot \sqrt{\ln(2n+4)/\pi}\right\}.$$

To show the efficiency of the verifier we have to show that the check $\mathbf{W} \in \mathcal{GL}_n(\mathbb{Z})$ can be done efficiently. This is the case, since $\mathbf{W} \in \mathcal{GL}_n(\mathbb{Z})$ if and only if $\mathbf{W}$ is integral and $\det(\mathbf{W}) = \pm 1$, both of which are easy to check in polynomial time.

To prove completeness of the protocol Σ, we have to show that the verifier accepts, whenever the prover executes the protocol honestly. If the prover is honest then we have $\mathbf{W} := \mathbf{U}^{-c} \cdot \mathbf{V} \in \mathcal{GL}_n(\mathbb{Z})$ because $\mathbf{U}$ and $\mathbf{V}$ are both unimodular by definition. Additionally we have

$$\mathbf{Q}' = \mathbf{V}^\top \mathbf{Q}_0 \mathbf{V} = \underbrace{(\mathbf{V}^\top (\mathbf{U}^{-c})^\top)}_{\mathbf{W}^\top} \underbrace{((\mathbf{U}^c)^\top \mathbf{Q}_0 \mathbf{U}^c)}_{\mathbf{Q}_c} \underbrace{(\mathbf{U}^{-c}\mathbf{V})}_{\mathbf{W}} = \mathbf{W}^\top \mathbf{Q}_c \mathbf{W},$$

Zero Knowledge Proof of Knowledge Σ

Consider two equivalent public quadratic forms $\mathbf{Q}_0, \mathbf{Q}_1 \in \mathcal{S}_n^{>0}(\mathbb{Z})$ and a secret unimodular $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}_1 = \mathbf{U}^\top \mathbf{Q}_0 \mathbf{U}$. Given the public parameter

$$s \geq \max\left\{\lambda_n([\mathbf{Q}_0]), \max\left\{\|\tilde{\mathbf{B}}_{\mathbf{Q}_0}\|, \|\tilde{\mathbf{B}}_{\mathbf{Q}_1}\|\right\} \cdot \sqrt{\ln(2n+4)/\pi}\right\},$$

we define the following protocol Σ that gives a zero-knowledge proof of knowledge of an isomorphism between $\mathbf{Q}_0$ and $\mathbf{Q}_1$:

Prover		Verifier
Sample $\mathbf{Q}' \leftarrow \mathcal{D}_s([\mathbf{Q}_0])$ by Alg. 16, together with $\mathbf{V}$ s.t. $\mathbf{Q}' = \mathbf{V}^\top \mathbf{Q}_0 \mathbf{V}$		
	$\xrightarrow{\mathbf{Q}'}$	
		Sample $c \leftarrow \mathcal{U}(\{0,1\})$
	$\xleftarrow{c}$	
Compute $\mathbf{W} = \mathbf{U}^{-c} \cdot \mathbf{V}$		
	$\xrightarrow{\mathbf{W}}$	
		Check if $\mathbf{W} \in \mathcal{GL}_n(\mathbb{Z})$, and $\mathbf{Q}' = \mathbf{W}^\top \mathbf{Q}_c \mathbf{W}$.

Figure 10.1: A zero knowledge proof of knowledge sigma protocol for knowledge of a solution to search LIP.

and thus the verifier accepts.

Special Soundness. For special soundness, we have to show that two accepting conversations with the same initial message, but a different challenge, allow to compute an isomorphism from $\mathbf{Q}_0$ to $\mathbf{Q}_1$. Suppose we have two such accepting conversations $(\mathbf{Q}', 0, \mathbf{W}_0)$ and $(\mathbf{Q}', 1, \mathbf{W}_1)$ of Σ where the first message $\mathbf{Q}'$ is identical. The acceptance implies that $\mathbf{W}_0, \mathbf{W}_1 \in \mathcal{GL}_n(\mathbb{Z})$ and $\mathbf{W}_0^\top \mathbf{Q}_0 \mathbf{W}_0 = \mathbf{Q}' = \mathbf{W}_1^\top \mathbf{Q}_1 \mathbf{W}_1$. Therefore

$\mathbf{U}' := \mathbf{W}_0\mathbf{W}_1^{-1} \in \mathcal{GL}_n(\mathbb{Z})$ gives an isomorphism from $\mathbf{Q}_0$ to $\mathbf{Q}_1$, as

$$\begin{aligned}\mathbf{U}'^{\top}\mathbf{Q}_0\mathbf{U}' &= (\mathbf{W}_1^{-1})^{\top}(\mathbf{W}_0^{\top}\mathbf{Q}_0\mathbf{W}_0)\mathbf{W}_1^{-1} \\ &= (\mathbf{W}_1^{-1})^{\top}(\mathbf{W}_1^{\top}\mathbf{Q}_1\mathbf{W}_1)\mathbf{W}_1^{-1} = \mathbf{Q}_1.\end{aligned}$$

We conclude that Σ has the special soundness property.

Honest-verifier zero-knowledge. To show that a honest verifier cannot learn anything from the interaction, we must show that there exists an efficient simulator that can produce accepting conversations following the same distribution, but *without* knowledge of a secret isomorphism between $\mathbf{Q}_0$ and $\mathbf{Q}_1$. We create such a simulator that given the public input $\mathbf{Q}_0, \mathbf{Q}_1$, outputs an accepting conversation with the same probability distribution as it would have been between a honest prover and verifier. Note that the first message $\mathbf{Q}'$ is always distributed as $\mathcal{D}_s([\mathbf{Q}_0])$, the challenge c as $\mathcal{U}(\{0,1\})$, and $\mathbf{V}$ is uniform over the set of isomorphisms from $\mathbf{Q}_0$ to $\mathbf{Q}'$, by Lemma 181. Because $\mathbf{U}$ is an isomorphism from $\mathbf{Q}_0$ to $\mathbf{Q}_1$ we have, given the challenge c, that $\mathbf{W} = \mathbf{U}^{-c}\cdot\mathbf{V}$ is uniform over the set of isomorphisms from $\mathbf{Q}_c$ to $\mathbf{Q}'$.

To simulate this we first sample the uniformly random challenge $c \leftarrow \mathcal{U}(\{0,1\})$. If $c = 0$ we can proceed the same as in Σ itself, e.g., sample $\mathbf{Q}' \leftarrow \mathcal{D}_s([\mathbf{Q}_0])$ using Algorithm 16, together with a $\mathbf{V}$ such that $\mathbf{Q}' = \mathbf{V}^{\top}\mathbf{Q}_0\mathbf{V}$, and set $\mathbf{W} := \mathbf{V}$. The final conversation $(\mathbf{Q}', 0, \mathbf{W})$ is accepting and follows by construction the same distribution as during an honest execution conditioned on challenge $c = 0$.

If $c = 1$ we use the fact that $[\mathbf{Q}_0] = [\mathbf{Q}_1]$, and that we can use Algorithm 16 with representative $\mathbf{Q}_1$ as input instead of $\mathbf{Q}_0$. So again we obtain $\mathbf{Q}' \leftarrow \mathcal{D}_s([\mathbf{Q}_1]) = \mathcal{D}_s([\mathbf{Q}_0])$ following the same distribution, but now together with a unimodular $\mathbf{W} \in \mathcal{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}' = \mathbf{W}^{\top}\mathbf{Q}_1\mathbf{W}$. The conversation $(\mathbf{Q}', 1, \mathbf{W})$ is accepting by construction, and $\mathbf{Q}'$ follows the same distribution $\mathcal{D}_s([\mathbf{Q}_0])$. Additionally by Lemma 181 the transformation $\mathbf{W}$ is indeed uniform over the set of isomorphisms from $\mathbf{Q}_1$ to $\mathbf{Q}'$.

We conclude that Σ has the honest-verifier zero-knowledge property.

10.3.2 Identification scheme

The Zero Knowledge Proof of Knowledge in the previous section is worst-case in the sense that, given any two equivalent forms $\mathbf{Q}_0, \mathbf{Q}_1 \in \mathcal{S}_n^{>0}(\mathbb{Z})$ and a secret isomorphism $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ from $\mathbf{Q}_0$ to $\mathbf{Q}_1$, we can show knowledge of such an isomorphism. However, to turn this Σ-protocol into an identification scheme (see [Dam02]), we need to define a distribution of $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$ (or alternatively of $\mathbf{Q}_1$ w.r.t $\mathbf{Q}_0$). Finding an isomorphism between $\mathbf{Q}_0$ and $\mathbf{Q}_1$ is at most as hard as solving either ac-sLIP$_s^{\mathbf{Q}_0}$ or ac-sLIP$_s^{\mathbf{Q}_1}$ for parameter s as in Σ. Therefore, a natural choice is to have $\mathbf{Q}_1$ distributed according to $\mathcal{D}_{s'}([\mathbf{Q}_0])$ for some parameter $s' \geq \max\{\lambda_n([\mathbf{Q}_0]), \|\tilde{\mathbf{B}}_{\mathbf{Q}_0}\| \cdot \sqrt{\ln(2n+4)/\pi}\}$, which we can efficiently sample from using Algorithm 16. The security of our identification scheme is then solely based on the hardness of ac-sLIP$_{s'}^{\mathbf{Q}_0}$.

10.4 Key Encapsulation Mechanism

In this section we construct a Key Encapsulation Mechanism (KEM) with a security proof based on the hardness of ΔLIP. See Figure 10.2 for the scheme. In short, we will need a quadratic form $\mathbf{S}$ along with an efficient decoder up to some radius $\rho < \lambda_1(\mathbf{S})/2$. The public key will consist of a long equivalent form $\mathbf{P} := \mathbf{U}^\top \mathbf{S} \mathbf{U} \leftarrow \mathcal{D}_s([\mathbf{S}])$, while the unimodular transformation $\mathbf{U}$ will be the secret key. Knowledge of the transformation $\mathbf{U}$ allows to decode w.r.t. $\mathbf{P}$ via $\mathbf{S}$; without any loss in decoding performance. The generated key will be a random error $\mathbf{e}$ of norm $\|\mathbf{e}\|_{\mathbf{P}} \leq \rho$, and it can be encapsulated as the syndrome $\overline{\mathbf{e}} := \mathbf{e} \bmod \mathbb{Z}^n \in [0,1)^n$. The receiver with knowledge of the secret transformation $\mathbf{U}$ can recover $\mathbf{e}$ by decoding via $\mathbf{S}$. The correctness follows from the fact that the decoding is unique due to $\rho < \lambda_1(\mathbf{S})/2$.

For the security we assume that it is (computationally) hard to differentiate between $\mathbf{P} \leftarrow \mathcal{D}_s([\mathbf{S}])$ and some random sample $\mathbf{R} \leftarrow \mathcal{D}_s([\mathbf{Q}])$ from a special class $[\mathbf{Q}]$, corresponding to a lattice admitting a dense sublattice. This assumption allows us to replace $\mathbf{P}$ by $\mathbf{R}$ in the security proof, which completely breaks the uniqueness of the decoding. That is, the syndrome $\overline{\mathbf{e}}$ has many (say $\exp(\Omega(\lambda))$) nearby points w.r.t. $\mathbf{R}$, and retrieving the exact original point becomes statistically hard.

Key Encapsulation Scheme

Let $\rho < \lambda_1(\mathbf{S})/2$ and let $\mathbf{S} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ be a quadratic form with an efficient decoder **Decode** with decoding radius ρ. Let $\mathcal{E} : \frac{1}{q}\mathbb{Z}^n \times \{0,1\}^z \to \{0,1\}^\ell$ be a $(\ell, \mathrm{negl}(n))$-extractor for some $\ell = \Theta(n)$. Given the public parameters

$$s \geq \max\{\lambda_n(\mathbf{S}), \|\tilde{\mathbf{B}}_\mathbf{S}\| \cdot \sqrt{\ln(2n+4)/\pi}\}, \text{ and}$$
$$q := \left\lceil \frac{s \cdot n}{\rho} \cdot \sqrt{\ln(2n+4)/\pi} \right\rceil,$$

we define the KEM $\mathcal{K} := (\mathbf{Gen}, \mathbf{Encaps}, \mathbf{Decaps})$ as follows:

- $(pk, sk) \leftarrow \mathbf{Gen}(1^n)$: on input 1^n do:
  1. Sample $\mathbf{P} \leftarrow \mathcal{D}_s([\mathbf{S}])$ using Alg. 16, together with $\mathbf{U}$ such that $\mathbf{P} = \mathbf{U}^\top \mathbf{S} \mathbf{U}$.
  2. Output $(pk, sk) = (\mathbf{P}, \mathbf{U})$.
- $(c, k) \leftarrow \mathbf{Encaps}(pk)$: on input a public key $\mathbf{P} = pk$ do:
  1. Sample $\mathbf{e} \leftarrow \frac{1}{q}\mathcal{D}_{\mathbf{P}, q\rho/\sqrt{n}} \in \frac{1}{q}\mathbb{Z}^n$ using Lemma 177.
  2. Compute $\mathbf{c} \leftarrow \mathbf{e} \bmod \mathbb{Z}^n$ s.t. $\mathbf{c} \in \mathbb{T}_q^n = \{0, \frac{1}{q}, \ldots, \frac{q-1}{q}\}^n$.
  3. Sample a random extractor seed $Z \leftarrow \{0,1\}^z$.
  4. Compute $k \leftarrow \mathcal{E}(\mathbf{e}, Z)$.
  5. Output (c, k) where $c := (\mathbf{c}, Z)$.
- $k \leftarrow \mathbf{Decaps}(sk, c)$: on input $c = (\mathbf{c}, Z)$ and a secret key $\mathbf{U} := sk$ do:
  1. Compute $\mathbf{y} \leftarrow \mathbf{Decode}(\mathbf{S}, \mathbf{Uc})$ s.t. $\|\mathbf{y} - \mathbf{Uc}\|_\mathbf{S} \leq \rho$, output $\perp$ on failure.
  2. Compute $k \leftarrow \mathcal{E}(\mathbf{c} - \mathbf{U}^{-1}\mathbf{y}, Z)$.
  3. Output k.

Figure 10.2: A key encapsulation scheme based on LIP.

Efficiency and correctness. We consider the efficiency and correctness of the KEM $\mathcal{K} := (\mathbf{Gen}, \mathbf{Encaps}, \mathbf{Decaps})$ instantiated with quadratic form $\mathbf{S} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ and public parameter

$$s \geq \max\{\lambda_n(\mathbf{S}), \|\tilde{\mathbf{B}}_{\mathbf{S}}\| \cdot \sqrt{\ln(2n+4)/\pi}\}.$$

By the above constraint on s, Algorithm 16 will run in polynomial-time by Lemma 181. Furthermore by Lemma 183 we have with overwhelming probability that

$$q\rho/\sqrt{n} \geq s\sqrt{n} \cdot \sqrt{\ln(2n+4)/\pi} \geq \|\tilde{\mathbf{B}}_{\mathbf{P}}\| \cdot \sqrt{\ln(2n+4)/\pi},$$

and thus we can efficiently sample from $\mathcal{D}_{\mathbf{P},q\rho/\sqrt{n}}$ by Lemma 177.

In order to prove correctness, note that in the key encapsulation algorithm the sampled error $\mathbf{e}$ has norm at most $\|\mathbf{e}\|_{\mathbf{P}} \leq \rho$ except with negligible probability by Lemma 175. We denote the encapsulated key by $k := \mathcal{E}(\mathbf{e}, Z)$, where Z denotes the randomness extractor's seed. Because $\rho < \lambda_1(\mathbf{S})/2$ the vector $\mathbf{x} := \mathbf{c} - \mathbf{e} \in \mathbb{Z}^n$ is the unique closest vector to $\mathbf{c}$ with respect to $\mathbf{P}$, which makes $\mathbf{U}\mathbf{x}$ the unique closest vector to $\mathbf{U}\mathbf{c}$ with respect to $\mathbf{S} = (\mathbf{U}^{-1})^\top \mathbf{P} \mathbf{U}^{-1}$. In the decapsulation the decoder computes the unique vector $\mathbf{y}$ at distance at most ρ from $\mathbf{U}\mathbf{c}$, which implies that $\mathbf{y} = \mathbf{U}\mathbf{x}$. So indeed the output $k' := \mathcal{E}(\mathbf{c} - \mathbf{U}^{-1}\mathbf{y}, Z) = \mathcal{E}(\mathbf{c} - \mathbf{x}, Z) = \mathcal{E}(\mathbf{e}, Z) = k$ equals the encapsulated key with overwhelming probability.

CPA security. To show that our KEM is CPA-secure we fall back to a lossy trapdoor argument as in [PW11]. Under the hardness of decisional LIP we can replace our unique ρ-decodable quadratic form by one that is far from uniquely decodable. For the latter it is enough to have a dense sublattice.

Lemma 191. *Let $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ be a quadratic form with a rank r sublattice $\mathbf{D} \cdot \mathbb{Z}^r \subset \mathbb{Z}^n$. For positive $\epsilon > 0$, center $\mathbf{c} \in \mathbb{R}^n$, parameter $\rho \geq \sqrt{n} \cdot 2\eta_\epsilon([\mathbf{D}^\top \mathbf{Q} \mathbf{D}])$, and for every $\mathbf{x} \in \mathbb{Z}^n$ we have*

$$\Pr_{X \sim \mathcal{D}_{\mathbf{Q},\rho/\sqrt{n},\mathbf{c}}}[X = \mathbf{x}] \leq \frac{1+\epsilon}{1-\epsilon} \cdot 2^{-r}.$$

Proof. Let $\mathbf{y} := \mathbf{x} - \mathbf{c} \in \mathbb{R}^n$, and decompose $\mathbf{y} =: \mathbf{y}_{\mathbf{D}} + \mathbf{y}_{\mathbf{D}^\perp}$ where $\mathbf{y}_{\mathbf{D}} \in \mathrm{span}(\mathbf{D}\mathbb{Z}^r)$, and $\mathbf{y}_{\mathbf{D}^\perp}$ is orthogonal to $\mathbf{y}_{\mathbf{D}}$ w.r.t $\mathbf{Q}$. Then, writing

$s := \rho/\sqrt{n}$, we have

$$\Pr_{X\sim\mathcal{D}_{\mathbf{Q},s,\mathbf{c}}}[X=\mathbf{x}] = \frac{\rho_{\mathbf{Q},s,\mathbf{c}}(\mathbf{x})}{\rho_{\mathbf{Q},s,\mathbf{c}}(\mathbb{Z}^n)} = \frac{\rho_{\mathbf{Q},s}(\mathbf{y})}{\rho_{\mathbf{Q},s}(\mathbf{y}+\mathbb{Z}^n)} \leq \frac{\rho_{\mathbf{Q},s}(\mathbf{y})}{\rho_{\mathbf{Q},s}(\mathbf{y}+\mathbf{D}\mathbb{Z}^r)}$$
$$= \frac{\rho_{\mathbf{Q},s}(\mathbf{y}_{\mathbf{D}^\perp})\cdot\rho_{\mathbf{Q},s}(\mathbf{y}_{\mathbf{D}})}{\rho_{\mathbf{Q},s}(\mathbf{y}_{\mathbf{D}^\perp})\cdot\rho_{\mathbf{Q},s}(\mathbf{y}_{\mathbf{D}}+\mathbf{D}\mathbb{Z}^r)} = \frac{\rho_{\mathbf{Q},s}(\mathbf{y}_{\mathbf{D}})}{\rho_{\mathbf{Q},s}(\mathbf{y}_{\mathbf{D}}+\mathbf{D}\mathbb{Z}^r)}.$$

Since we can write $\mathbf{y}_{\mathbf{D}} = \mathbf{D}\mathbf{z}$ for some $\mathbf{z}\in\mathbb{R}^r$, the above equals $\Pr_{X\sim\mathcal{D}_{\mathbf{D}^\top\mathbf{Q}\mathbf{D},s,\mathbf{z}}}[X=\mathbf{0}]$, which, by Lemma 176, is bounded by $\frac{1+\epsilon}{1-\epsilon}\cdot 2^{-r}$ because $s \geq 2\eta_\epsilon([\mathbf{D}^\top\mathbf{Q}\mathbf{D}])$.

□

Theorem 192. *We consider the KEM* $\mathcal{K} := (\mathbf{Gen}, \mathbf{Encaps}, \mathbf{Decaps})$ *instantiated with quadratic form* $\mathbf{S}\in\mathcal{S}_n^{>0}(\mathbb{Z})$*, decoding radius* ρ*, and public key parameter* $s>0$*. Let* $\mathbf{Q}\in\mathcal{S}_n^{>0}(\mathbb{Z})$ *be a quadratic form with a dense rank* $r=\Theta(n)$ *sublattice* $D\mathbb{Z}^r\subset\mathbb{Z}^n$*, in particular such that* $\eta_{\frac{1}{2}}(\mathbf{D}^\top\mathbf{Q}\mathbf{D}) \leq \rho/(2\sqrt{n})$*. Then* $\mathcal{K}$ *is CPA-secure if* ac-ΔLIP$_s^{\mathbf{S},\mathbf{Q}}$ *is hard.*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary. We present two games $\mathbf{Game}_1$ and $\mathbf{Game}_2$, where $\mathbf{Game}_1$ is the regular CPA-security game with the original scheme, and $\mathbf{Game}_2$ is almost identical but with the only change that the public key is drawn from $\mathcal{D}_s([\mathbf{Q}])$ instead of $\mathcal{D}_s([\mathbf{S}])$. By the hardness of ac-$\Delta$LIP$_s^{\mathbf{S},\mathbf{Q}}$ the two games are computationally indistinguishable, and due to the dense sublattice we can conclude that winning $\mathbf{Game}_2$ with a non-negligible advantage is statistically impossible.

Let the key-size $\ell=\Theta(n)$ be such that $\ell \leq r-\log_2(3)$. The original KEM CPA game $\mathbf{Game}_1$ is as follows [KL20]:

- $\mathbf{Gen}(1^n)$ is run to obtain a public key $pk=\mathbf{P}$. Then $\mathbf{Encaps}(pk)$ is run to generate (c,k) with $k\in\{0,1\}^\ell$.
- A uniform bit $b\in\{0,1\}$ is chosen. If $b=0$, set $\hat{k}:=k$, if $b=1$, choose a uniform $\hat{k}\in\{0,1\}^\ell$.
- Given $(pk, c=(\mathbf{c},Z), \hat{k})$ the adversary $\mathcal{A}$ wins the experiment if b is guessed correctly.

The only difference between $\mathbf{Game}_1$ and $\mathbf{Game}_2$ is that in $\mathbf{Game}_2$ we sample the public key $\mathbf{P}$ from $\mathcal{D}_s([\mathbf{Q}])$ instead of $\mathcal{D}_s([\mathbf{S}])$. Note that $\mathbf{Game}_1$ and $\mathbf{Game}_2$ both only use public information and thus by the hardness of ac-$\Delta\mathrm{LIP}_s^{\mathbf{S},\mathbf{Q}}$ the two are computationally indistinguishable by $\mathcal{A}$.

Now we take a look at $\mathbf{Game}_2$. Consider the output $(c = (\mathbf{c}, Z), k) \leftarrow \mathbf{Encaps}(pk)$ where $pk := \mathbf{Q}' \in [\mathbf{Q}]$. For any fixed $\mathbf{c}$ we have by construction that $k := \mathcal{E}(\mathbf{e}, Z)$, where $\mathbf{e} \leftarrow \frac{1}{q}\mathcal{D}_{\mathbf{Q}',q\rho/\sqrt{n}}$ under the condition that $\mathbf{e} = \mathbf{c} \bmod \mathbb{Z}^n$. Equivalently we could say that $\mathbf{e} \leftarrow \mathbf{c} - \mathcal{D}_{\mathbf{Q}',\rho/\sqrt{n},\mathbf{c}}$, then, by Lemma 191, we know that $\mathbf{e}$ has a min-entropy of at least $r - \log_2(3) \geq l$, and thus $k := \mathcal{E}(\mathbf{e}, Z) \in \{0,1\}^\ell$ is negligibly close to uniform and independent of c. So, in $\mathbf{Game}_2$ we have that $\hat{k}$ is negligibly close to uniform, independent of c and the choice of $b \in \{0,1\}$, making it impossible for $\mathcal{A}$ to guess b with non-negligible advantage.

□

10.5 Signature scheme

Similar to the Key Encapsulation Mechanism we propose in Figure 10.3 a *hash-then-sign* signature scheme based on ΔLIP. The main requirement is a quadratic form $\mathbf{S}$ along with an efficient discrete Gaussian sampling algorithm of smallish width $\rho/\sqrt{n} \geq \eta_{2^{-\Theta(n)}}(\mathbf{S})$.

Again the public key will consist of some lesser reduced form $\mathbf{P} := \mathbf{U}^\top\mathbf{S}\mathbf{U} \leftarrow \mathcal{D}_s([\mathbf{S}])$ equivalent to $\mathbf{S}$, where the unimodular transformation $\mathbf{U}$ will form the secret key. To sign a message we use a full domain hash to obtain a uniform coset $\mathbf{t} + \mathbb{Z}^n$, The signature then consists of a nearby vector $\boldsymbol{\sigma} \leftarrow \mathcal{D}_{\mathbf{P},\rho/\sqrt{n},\mathbf{t}}$ w.r.t. the form $\mathbf{P}$. The nearby vector is obtained via $\mathbf{S}$ by using the secret transformation $\mathbf{U}$.

The security assumption is similar, but in some way dual to that of the KEM. Again assume that it is computationally hard to differentiate between $\mathbf{P}$ and some special class of forms $[\mathbf{Q}]$; however in this case $\mathbf{Q}$ must admit a sparse projection (equivalently, their dual should contain a dense lattice). The sparsity implies that a uniformly random target $\mathbf{t}$ does not have a nearby vector with overwhelming probability, making the signage vacuously hard.

Signature Scheme

Let $\mathbf{S} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ be a quadratic form together with a sampling algorithm **DiscreteSample** that allows to sample statistically close to $\mathcal{D}_{\mathbf{P},\rho/\sqrt{n}}(\mathbf{t}+\mathbb{Z}^n)$ for some parameter $\rho/\sqrt{n} \geq \eta_{2^{-\Theta(n)}}([\mathbf{S}])$ and any target $\mathbf{t} \in \mathbb{T}_q^n$. Let $\mathcal{H} : \mathcal{M} \to \mathbb{T}_q^n$ be a full domain hash function (modeled as a random oracle). Given the public parameters

$$s \geq \max\{\lambda_n(\mathbf{S}), \|\tilde{\mathbf{B}}_{\mathbf{S}}\| \cdot \sqrt{\ln(2n+4)/\pi}\}, \text{ and}$$
$$q := \left\lceil \frac{s \cdot n}{\rho} \cdot \sqrt{\ln(2n+4)/\pi} \right\rceil,$$

we define the signature scheme $\mathcal{S} := (\mathbf{Gen}, \mathbf{Sign}, \mathbf{Verify})$ as follows:

- $(pk, sk) \leftarrow \mathbf{Gen}(1^n)$: on input 1^n do:
  1. Sample $\mathbf{P} \leftarrow \mathcal{D}_s([\mathbf{S}])$ using Alg. 16, together with $\mathbf{U}$ s.t. $\mathbf{P} = \mathbf{U}^\top \mathbf{S} \mathbf{U}$.
  2. Output $(pk, sk) = (\mathbf{P}, \mathbf{U}) \in \mathcal{S}_n^{>0}(\mathbb{Z}) \times \mathcal{GL}_n(\mathbb{Z})$.
- $\boldsymbol{\sigma} \leftarrow \mathbf{Sign}(sk, m)$: on input a message m and a secret key $\mathbf{U} := sk$ do:
  1. Compute $\mathbf{t} \leftarrow \mathcal{H}(m)$.
  2. Sample $\boldsymbol{\sigma}' \leftarrow \mathcal{D}_{\mathbf{S},\rho/\sqrt{n},\mathbf{U}\mathbf{t}}$ using **DiscreteSample**.
  3. Compute $\boldsymbol{\sigma} \leftarrow \mathbf{U}^{-1}\boldsymbol{\sigma}'$.
  4. Output $\boldsymbol{\sigma} \in \mathbb{Z}^n$.
- $b := \mathbf{Verify}(pk, m, \boldsymbol{\sigma})$: on input a public key $\mathbf{P} = pk$, a message m and a signature $\boldsymbol{\sigma}$ do:
  1. Compute $\mathbf{t} \leftarrow \mathcal{H}(m)$.
  2. If $\boldsymbol{\sigma} \in \mathbb{Z}^n$, and $\|\mathbf{t} - \boldsymbol{\sigma}\|_{\mathbf{P}} \leq \rho$, output $b = 1$.
  3. Otherwise, output $b = 0$.

Figure 10.3: A signature scheme based on LIP.

Correctness. To prove correctness, we mainly have to check that the returned signature $\boldsymbol{\sigma} \in \mathbb{Z}^n$ is indeed close to $\mathbf{t} := \mathcal{H}(m)$ w.r.t $\mathbf{P}$. Because $\mathbf{P} = \mathbf{U}^\top \mathbf{S}\mathbf{U}$ we have:

$$\|\boldsymbol{\sigma} - \mathbf{t}\|_{\mathbf{P}} = \|\mathbf{U}(\boldsymbol{\sigma} - \mathbf{t})\|_{\mathbf{S}} = \|\boldsymbol{\sigma}' - \mathbf{U}\mathbf{t}\|_{\mathbf{S}},$$

and by Lemma 175 we have with overwhelming probability that

$$\|\boldsymbol{\sigma} - \mathbf{t}\|_{\mathbf{P}} = \|\boldsymbol{\sigma}' - \mathbf{U}\mathbf{t}\|_{\mathbf{S}} \leq \rho/\sqrt{n} \cdot \sqrt{n} = \rho,$$

concluding the correctness.

Security. For the security proof we first consider a class of quadratic forms for which the signage is vacuously hard, e.g., for a random target $\mathbf{t} \in \mathbb{R}^n/\mathbb{Z}^n$ there exists no nearby vector.

Lemma 193. *Let $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ be a quadratic form with a dense rank k sublattice $\mathbf{D}\mathbb{Z}^k \subset \mathbb{Z}^n$, in particular such that $\rho/\sqrt{k} \leq 1/(\sqrt{8\pi e} \cdot \det(\mathbf{D}^\top\mathbf{Q}\mathbf{D})^{1/2k})$. Then for the dual form $\mathbf{Q}^{-1}$ we have*

$$\Pr_{\mathbf{t}\sim\mathcal{U}([0,1]^n)}[|(\mathbf{t} + \mathcal{B}^n_{\mathbf{Q}^{-1},\rho}) \cap \mathbb{Z}^n| \geq 1] \leq 2^{-k}.$$

Proof. Let $V := \operatorname{span}(\mathbf{D}) \subset \mathbb{R}^n$ such that the orthogonal projection w.r.t. $\mathbf{Q}^{-1}$ of $\mathbb{Z}^n$ onto V defines a projected lattice $\mathbf{C}\mathbb{Z}^k := \pi_{\mathbf{Q}^{-1},V}(\mathbb{Z}^n)$ of rank k, with $\det(\mathbf{C}^\top\mathbf{Q}^{-1}\mathbf{C}) \geq 1/\det(\mathbf{D}^\top\mathbf{Q}\mathbf{D})$. Because a projection is non-increasing in length we have

$$\begin{aligned}&\Pr_{\mathbf{t}\sim\mathcal{U}(\mathbb{R}^n/\mathbb{Z}^n)}[|(\mathbf{t} + \mathcal{B}^n_{\mathbf{Q}^{-1},\rho}) \cap \mathbb{Z}^n| \geq 1]\\ &\leq \Pr_{\mathbf{t}\sim\mathcal{U}(\mathbb{R}^k/\mathbb{Z}^k)}[|(\mathbf{t} + \mathcal{B}^k_{\mathbf{C}^\top\mathbf{Q}^{-1}\mathbf{C},\rho}) \cap \mathbb{Z}^n| \geq 1] = (*).\end{aligned}$$

Then using Markov's inequality we can bound the above by

$$\begin{aligned}(*) \leq \mathbb{E}_{\mathbf{t}\sim\mathcal{U}(\mathbb{R}^k/\mathbb{Z}^k)}[|(\mathbf{t} + \mathcal{B}^k_{\mathbf{C}^\top\mathbf{Q}^{-1}\mathbf{C},\rho}) \cap \mathbb{Z}^n|] &= \frac{\operatorname{Vol}_{\mathbf{C}^\top\mathbf{Q}^{-1}\mathbf{C}}(\mathcal{B}^k_{\mathbf{C}^\top\mathbf{Q}^{-1}\mathbf{C},\rho})}{\operatorname{Vol}_{\mathbf{C}^\top\mathbf{Q}^{-1}\mathbf{C}}(\mathbb{R}^k/\mathbb{Z}^k)}\\ &\leq \frac{(2\pi e/k)^{k/2} \cdot \rho^k}{\sqrt{\det(\mathbf{C}^\top\mathbf{Q}^{-1}\mathbf{C})}} \leq 2^{-k}.\end{aligned}$$

□

Theorem 194. *We consider the signature scheme* $\mathcal{S} := (\mathbf{Gen}, \mathbf{Sign}, \mathbf{Verify})$ *instantiated with quadratic form* $\mathbf{S} \in \mathcal{S}_n^{>0}(\mathbb{Z})$*, sampling parameter* ρ*, and public key parameter* $s > 0$*. Let* $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z})$ *be a quadratic form with a dense rank* $k = \Theta(n)$ *sublattice* $\mathbf{D}\mathbb{Z}^k \subset \mathbb{Z}^n$*, in particular such that* $2\rho/\sqrt{k} \leq (\sqrt{8\pi e} \cdot \det(\mathbf{D}^\top \mathbf{Q} \mathbf{D}^\top)^{1/k})^{-1}$*. Then* $\mathcal{S}$ *is EUF-CMA secure if* ac-$\Delta\mathrm{LIP}_s^{\mathbf{S},\mathbf{Q}^{-1}}$ *is hard.*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary. We present three games $\mathbf{Game}_1, \mathbf{Game}_2, \mathbf{Game}_3$ where $\mathbf{Game}_1$ is the regular EUF-CMA game with the original scheme, $\mathbf{Game}_2$ reprograms the random oracle to generate valid signatures without knowledge of the secret key, and $\mathbf{Game}_3$ samples the public key from $[\mathbf{Q}^{-1}]$ instead of $[\mathbf{S}]$. By a standard smoothness argument the adversary's view of $\mathbf{Game}_1$ and $\mathbf{Game}_2$ is statistically indistinguishable, and $\mathbf{Game}_2$ and $\mathbf{Game}_3$ are indistinguishable by the hardness of ac-$\Delta\mathrm{LIP}_s^{\mathbf{S},\mathbf{Q}^{-1}}$. Then we conclude by Lemma 193 that the problem of forging a signature in $\mathbf{Game}_3$ is statistically hard.

The original EUF-CMA game $\mathbf{Game}_1$ is as follows [KL20]:

- $\mathbf{Gen}(1^n)$ is run to obtain keys $(pk = \mathbf{P}, sk = \mathbf{U})$.
- Adversary $\mathcal{A}$ is given $pk = \mathbf{P}$ and access to an oracle $\mathbf{Sign}(sk, \cdot)$. The adversary then outputs $(m, \boldsymbol{\sigma})$ where m was not queried before to the oracle.
- $\mathcal{A}$ succeeds if and only if $\mathbf{Verify}(pk, m, \boldsymbol{\sigma}) = 1$.

To show that our signature scheme $\mathcal{S}$ is EUF-CMA secure we have to show that $\mathbf{Game}_1$ succeeds only with negligible probability. We assume that the adversary queries the oracle on $l = \mathrm{poly}(n)$ distinct[2] message $m_1, \ldots, m_l$. In $\mathbf{Game}_1$ the secret key is used to obtain a valid signature $(m_i, \boldsymbol{\sigma}_i)$ where $\boldsymbol{\sigma}_i \leftarrow \mathcal{D}_{\mathbf{P},\rho/\sqrt{n},\mathcal{H}(m_i)}$. In $\mathbf{Game}_2$ instead we first sample a random error $\mathbf{e}_i \leftarrow \frac{1}{q} \cdot \mathcal{D}_{\mathbf{P},q\rho/\sqrt{n}}$. By Lemma 183 we have $q\rho/\sqrt{n} \geq \|\tilde{\mathbf{B}}_{\mathbf{P}}\| \cdot \sqrt{\ln(2n+4)/\pi}$ with overwhelming probability, and thus by Lemma 177 we can do the sampling without using the secret key. Then we reprogram the random oracle such that $\mathcal{H}(m_i) := \mathbf{t}_i = \mathbf{e} \bmod \mathbb{Z}^n \in \mathbb{T}_q$, and return the signature pair $(m_i, \boldsymbol{\sigma}_i := \mathbf{t}_i - \mathbf{e}_i)$. Note that the probability that $\mathbf{t}_i$ equals any target $\mathbf{t} \in \mathbb{T}_q^n$ is proportional

[2] this can be enforced by salting messages or by derandomization.

to $\rho_{\mathbf{P},\rho/\sqrt{n},\mathbf{t}}(\mathbb{Z}^n)$. So $\mathbf{t}_i$ is close to uniform by Lemma 173 because $\rho/\sqrt{n} \geq \eta_{2^{-\Theta(n)}}([\mathbf{S}]) = \eta_{2^{-\Theta(n)}}([\mathbf{P}])$, and thus the random oracle is still simulated correctly. Additionally the conditional probability of $\boldsymbol{\sigma}_i$ conditioned on $\mathbf{t}_i$ is exactly the same as in $\mathbf{Game}_1$, so we can conclude that $\mathbf{Game}_1$ and $\mathbf{Game}_2$ are statistically indistinguishable from the adversary's point of view.

The only difference between $\mathbf{Game}_2$ and $\mathbf{Game}_3$ is that in $\mathbf{Game}_3$ we sample the public key $\mathbf{P}$ from $\mathcal{D}_s([\mathbf{Q}^{-1}])$ instead of $\mathcal{D}_s([\mathbf{S}])$. Note that $\mathbf{Game}_2$ and $\mathbf{Game}_3$ both only use public information and thus by the hardness of ac-$\Delta\mathrm{LIP}_s^{\mathbf{S},\mathbf{Q}^{-1}}$ the two are computationally indistinguishable.

To conclude note that for any message m we obtain a random target $\mathbf{t} := \mathcal{H}(m) \in \mathbb{T}_q^n$. Let $\mathbf{e}'$ be uniform over the Babai nearest plane region defined by $\mathbf{P}$, then $\|\mathbf{e}'\|_{\mathbf{P}} \leq \frac{\sqrt{n}}{2}\|\tilde{\mathbf{B}}_{\mathbf{P}}\|$, and $\mathbf{t}' := \mathbf{t} + \frac{1}{q}\mathbf{e}'$ is uniform over $\mathbb{R}^n/\mathbb{Z}^n$. By Lemma 193 the uniformly random target $\mathbf{t}'$ lies at distance at least 2ρ from $\mathbb{Z}^n$ w.r.t. $\mathbf{P}$ with overwhelming probability. So for $\mathbf{t}$ we have with overwhelming probability that:

$$\begin{aligned}\mathrm{dist}_{\mathbf{P}}(\mathbf{t},\mathbb{Z}^n) &\geq \mathrm{dist}_{\mathbf{P}}(\mathbf{t}',\mathbb{Z}^n) - \left\|\frac{1}{q}\mathbf{e}'\right\|_{\mathbf{P}} \geq 2\rho - \frac{\sqrt{n}\cdot\|\tilde{\mathbf{B}}_{\mathbf{P}}\|}{2q} \\ &\geq 2\rho - \rho/(2\sqrt{\ln(2n+4)/\pi}) > \rho.\end{aligned}$$

Therefore it is statistically impossible for the adversary to return a valid signature for m, and thus to win $\mathbf{Game}_3$. □

10.6 Cryptanalysis

Equivalent quadratic forms $\mathbf{Q}, \mathbf{Q}' := \mathbf{U}^t\mathbf{Q}\mathbf{U}$ (for some $\mathbf{U} \in \mathcal{GL}_n(\mathbb{Z})$) share many common properties, and these invariants can be used to decide that two quadratic forms cannot be equivalent, or can guide the search for an isomorphism.

10.6.1 Invariants

Recall from Section 9.3 that we named the invariants that are easy to compute *arithmetic* invariants, and those that are hard to compute *geometric* invariants. The arithmetic invariants for an integral form

$\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{Z})$, such as the determinant, gcd, parity and the genus, provide an efficiently computable fingerprint $\mathrm{ari}(\mathbf{Q})$. The fingerprint is essentially only useful to answer the ΔLIP problem in the negative. When instantiating ΔLIP for cryptosystems, we should therefore make sure that these fingerprints match.

The geometric invariants, such as $|\mathrm{Min}(\mathbf{Q})|$ or more generally (part of) the Theta-series, appears to involve finding or even enumerating short vectors; in particular they are plausibly hard to compute.

10.6.2 Algorithms for distinguish-LIP and hardness conjecture

In Section 10.7, we will use ΔLIP with quadratic forms that have different minimal distances $\lambda_1(\mathbf{Q}_0) < \lambda_1(\mathbf{Q}_1)$. However we will be careful to ensure that their arithmetic invariant match $\mathrm{ari}(\mathbf{Q}_0) = \mathrm{ari}(\mathbf{Q}_1)$ to not make the problem trivial.

Approximate-SVP oracle. An f-approx-SVP oracle applied to a form $\mathbf{Q}$ finds a short vector of length at most $f \cdot \lambda_1(\mathbf{Q})$. So ΔLIP is no harder than f-approx-SVP for $f = \lambda_1(\mathbf{Q}_1)/\lambda_1(\mathbf{Q}_0) > 1$ in any of those lattices.

Unusual-SVP via lattice reduction. However even when the gap between $\lambda_1(\mathbf{Q}_0)$ and $\lambda_1(\mathbf{Q}_1)$ is small, the minimal vectors may individually still be unusually short, which make them significantly easier to find than in a random lattice. This is usually formalized via the f-unique-SVP problem, but many instances of interest do not have such a gap between λ_1 and λ_2. In fact the lattice $\mathbb{Z}^n$, the the lattices of Barnes-Wall and Barnes-Sloane, all have $\lambda_1 = \lambda_2 = \cdots = \lambda_n$. But practical and heuristic studies have showed that uniqueness is not that relevant to lattice attacks [AD21]. We therefore introduce yet another lattice problem, called *unusual-SVP* to discuss such instances. A formal complexity reduction between unusual-SVP and unique-SVP matching or approaching the heuristic state of the art appears to be a valuable research objective, but is beyond the scope of the present article.

We define f-unusual-SVP: find a minimal vector under the promise that $\lambda_1(\mathbf{Q}) \leq \mathrm{gh}(\mathbf{Q})/f$, where the Gaussian Heuristic $\mathrm{gh}(\mathbf{Q})$ is a

heuristic estimate for $\lambda_1(\mathbf{Q})$ given by:

$$\mathrm{gh}(\mathbf{Q}) := \det(\mathbf{Q})^{1/2n} \cdot \frac{1}{\sqrt{\pi}} \cdot \Gamma(1+n/2)^{1/n} \approx \det(\mathbf{Q})^{1/2n} \cdot \sqrt{\frac{n}{2\pi e}}.$$

State of the art lattice reduction techniques find these unusually short vector more easily than longer vectors with length around $\mathrm{gh}(\mathbf{Q})$, where (heuristically) the hardness is directly driven by the ratio $f = \mathrm{gh}(\mathbf{Q})/\lambda_1(\mathbf{Q})$ [AD21]. E.g., for $f = O(n^\alpha)$ with $\alpha \geq 0$, we can heuristically solve f-unusual-SVP by running BKZ-β with blocksize $\beta = \frac{n}{2\alpha+1} + o(n)$. Given a form $\mathbf{Q}' \in [\mathbf{Q}_0] \cup [\mathbf{Q}_1]$, we parametrize the lattice reduction algorithm to find a unusual short vector with length $\min\{\lambda_1(\mathbf{Q}_0), \lambda_1(\mathbf{Q}_1)\}$, then depending on success we learn that either $\mathbf{Q}' \in [\mathbf{Q}_0]$ or $\mathbf{Q}' \in [\mathbf{Q}_1]$.

Conclusion. To conclude, let us also note that any of the above attack can also be run over the dual. To state a hardness conjecture capturing these attacks we define the primal-dual gap to the Gaussian Heuristic as:

$$\mathrm{gap}(\mathbf{Q}) = \max\left\{\frac{\mathrm{gh}(\mathbf{Q})}{\lambda_1(\mathbf{Q})}, \frac{\mathrm{gh}(\mathbf{Q}^{-1})}{\lambda_1(\mathbf{Q}^{-1})}\right\}.$$

Note that this quantity might be slightly lower than 1 (but no lower than 1/2 by Minkowski's bound): there might exist excellent lattice packings beating the Gaussian Heuristic. We will be assuming[3] $\mathrm{gap}(\mathbf{Q}_i) \geq 1$, which implies that $\lambda_1(\mathbf{Q}_i)/\lambda_1(\mathbf{Q}_{1-i}) \leq \mathrm{gap}(\mathbf{Q}_i)$ for $i = 0, 1$, therefore also capturing the approximate-SVP approach.

In all the attacks above, one first searches for vectors no larger than $f \cdot \lambda_1(\mathbf{Q}_i)$ w.r.t. $\mathbf{Q}_i$ for $f = \mathrm{gap}(\mathbf{Q}_i)$, hence the following conjecture.

Conjecture 195 (Hardness of ΔLIP (Strong)). *For any two classes of quadratic forms $[\mathbf{Q}_0], [\mathbf{Q}_1]$ of dimension n, with $\mathrm{ari}([\mathbf{Q}_0]) = \mathrm{ari}([\mathbf{Q}_1])$, and $1 \leq \mathrm{gap}([\mathbf{Q}_i]) \leq f$, the best attack against wc-$\Delta\mathrm{LIP}^{\mathbf{Q}_0,\mathbf{Q}_1}$ requires solving f-approx-SVP in the worst-case from either $[\mathbf{Q}_0]$ or $[\mathbf{Q}_1]$.*

[3] That is, we do not make a hardness conjecture for such exceptionally dense lattice packings. Such a regime has never been considered in practical cryptanalysis and would deserve specific attention.

This conjecture is meant to offer a comparison point with existing lattice-based cryptography in terms of the approximating factor. Beyond contradicting this assumption, we also invite cryptanalysis effort toward concrete comparison of f-approx-SVP on those instances to SIS and LWE with the same approximation factor f.

If one only wishes to argue exponential security in n of the schemes proposed in this paper, a sufficient conjecture is the following.

Conjecture 196 (Hardness of ΔLIP (Mild)). *For any two classes of quadratic forms* $[\mathbf{Q}_0], [\mathbf{Q}_1]$ *of dimension* n*, with* $\mathrm{ari}([\mathbf{Q}_0]) = \mathrm{ari}([\mathbf{Q}_1])$*, and* $\mathrm{gap}([\mathbf{Q}_i]) \leq \mathrm{poly}(n)$*,* wc-$\Delta\mathrm{LIP}^{\mathbf{Q}_0,\mathbf{Q}_1}$ *is* $2^{\Theta(n)}$*-hard.*

Note that the conjectures above are very strong and 'best-case' over the choice of the isomorphism classes. That is, even though we may only want to use ΔLIP for specific choices of isomorphism classes, we gladly invite cryptanalysis effort on ΔLIP for any choice of isomorphism classes.

We would also like to motivate any reductions from more standard lattice assumptions to LIP, though given current knowledge on LIP, this may be hard to attain. A more reasonable goal might be to generalize the search-to-decision reduction of Szydlo [Szy03], which is currently limited to solving sLIP for the trivial lattice $\mathbb{Z}^n$ given a decisional LIP oracle for a few special lattices.

10.6.3 Algorithms for search-LIP and challenges

While the mentioned arithmetic and geometric invariants allow to semi-decide LIP, the search version requires more effort. Recall from Section 9.5 that it typically proceeds by enumerating sets of short vectors for both quadratic forms, and then to backtrack search for isometries between these sets. The hardest instances appear to be those with many minimal vectors, and in particular those with $\lambda_1(\mathbf{Q}) = \ldots = \lambda_n(\mathbf{Q})$.

Given the current state of the art, it seems difficult to give a precise conjecture for the hardness of search-LIP, as it may depend on the minimal distance, the kissing number, and the size and structure of the automorphism group. Given that all known approaches require finding at least one short vector, we can conjecture exponential hardness for

	Source	Dim.	Kissing Number
Λ_{24}	[NS11; CS13]	24	$196560 \approx 2^{17.6}$
MW_{44}	[NS11]	44	$2708112 \approx 2^{21.4}$
$\mathrm{P}_{48\mathrm{n}}$	[NS11]	48	$52416000 \approx 2^{25.6}$
Ne_{64}	[NS11]	64	$138458880 \approx 2^{27.0}$
Γ_{72}	[NS11]	72	$6218175600 \approx 2^{32.5}$
MW_{128}	[NS11]	128	$218044170240 \approx 2^{37.7}$
BW_n	[CS13, Sec 6.5]	$n = 2^m$	$2^{m^2/2+O(m)}$
V_n	[Vlă19]	n	$\geq 2^{0.0338n-o(n)}$

Table 10.2: Some challenge lattices for search-LIP

lattices with polynomial $\mathrm{gap}([\mathbf{Q}])$, bearing in mind that some instances may be much harder, with a complexity up to $n^{O(n)}$.

Conjecture 197 (Hardness of sLIP). *For any class* $[\mathbf{Q}]$ *of quadratic forms of dimension* n *such that* $\mathrm{gap}([\mathbf{Q}]) \leq \mathrm{poly}(n)$, wc-sLIP$^{\mathbf{Q}}$ *is* $2^{\Theta(n)}$*-hard.*

To motivate further study of the hardness of LIP, we provide a list of challenge lattices for LIP in Table 10.2, selected from known lattices with large kissing numbers. More remarkable lattices may be found in the online catalogue of Nebe and Sloane, in particular the section on the kissing number [NS11].

For these large kissing number challenges it might be more enlightening to separate the search for short vectors and the isomorphism reconstruction. We therefore invite the cryptanalist to even assume that sampling uniformly a random shortest vector comes at unit cost. The search for all shortest vectors in these specific lattices may also be harder in practice than for random lattices. Indeed, for these challenges, the kissing number is significantly larger than the $\approx (4/3)^{n/2}$ many short vectors provided by heuristic sieving algorithms [NV08].

10.7 Instantiating from remarkable lattices

To instantiate our KEM and signature scheme, we do not only need a lattice with efficient decoding or sampling; we also need a second lattice with a specific property to instantiate the ΔLIP problem and argue security. This section deals with how the ΔLIP pair is constructed from a single remarkable lattice, while keeping the gaps small. Note that this is just one generic way of doing this; for specific lattices there might exist better instantiations.

10.7.1 Key Encapsulation Mechanism

To instantiate our KEM we need two quadratic forms: a form $\mathbf{S}$ along with an efficient decoder that can decode up to some distance $\rho < \lambda_1(\mathbf{S})/2$, and a form $\mathbf{Q}$ with a dense rank $k = \Theta(n)$ sublattice $\mathbf{D}\cdot\mathbb{Z}^k \subset \mathbb{Z}^n$ such that $\eta_{\frac{1}{2}}(\mathbf{D}^t\mathbf{Q}\mathbf{D}) \leq \rho/(2\sqrt{n})$. For simplicity of notation we move to the lattice point of view.

We assume to have an n-dimensional lattice $\mathcal{L}$ for which $\mathrm{gap}(\mathcal{L}) \leq f = f(n)$ is bounded, and for which we can decode up to $\rho = \Theta(1/f)\cdot\mathrm{gh}(\mathcal{L}) < \lambda_1(\mathcal{L})/2$. I.e., a lattice for which the primal, dual and decoding gap are bounded by f. We consider a general construction leading to a $2n$-dimensional primary lattice $\mathcal{L}_\mathbf{S}$ and secondary lattice $\mathcal{L}_\mathbf{Q}$ with primal-dual gaps bounded by $O(f^3)$ and such that $\mathcal{L}_\mathbf{Q}$ has a dense enough sublattice to instantiate our KEM.

By Lemma 174 and due to the bounded gap of $\mathcal{L}$, we have

$$\eta_{\frac{1}{2}}(\mathcal{L}) \leq \eta_{2^{-n}}(\mathcal{L}) \leq \frac{\sqrt{n}}{\lambda_1(\mathcal{L}^*)} \leq \frac{\sqrt{n}\cdot f}{\mathrm{gh}(\mathcal{L}^*)} = \Theta(f\cdot\det(\mathcal{L})^{1/n}).$$

Now let $g = \Theta(f^2)$ be a positive integer and consider the lattices:

$$\mathcal{L}_\mathbf{S} := g\cdot\mathcal{L} \oplus (g+1)\cdot\mathcal{L}, \text{ and } \mathcal{L}_\mathbf{Q} := \mathcal{L} \oplus g(g+1)\mathcal{L}.$$

By construction the first lattice $\mathcal{L}_\mathbf{S}$ is geometrically similar to (two orthogonal copies of) the original lattice. In particular we can still decode $\mathcal{L}_\mathbf{S}$ up to radius $\rho' := g\cdot\rho = \Theta(g/f)\cdot\mathrm{gh}(\mathcal{L})$. Furthermore, the second lattice $\mathcal{L}_\mathbf{Q}$ contains by construction a dense sublattice $\mathcal{L} \subset \mathcal{L}_\mathbf{Q}$, where the parameter g allows to tune the precise (relative) density.

Invariants match. Both lattices have determinant $g^n(g+1)^n \det(\mathcal{L})^2$. Due to the coprimality of g and $g+1$ we still have $\gcd(\mathcal{L}_{\mathbf{S}}) = \gcd(\mathcal{L}_{\mathbf{Q}}) = \gcd(\mathcal{L})$, and similarly for the parity. It remains to check rational equivalence and p-adic equivalence for all primes p. Let $\mathbf{R}$ denote a quadratic form representing $\mathcal{L}$. Up to integral equivalence, we have:

$$\mathbf{S} := \begin{pmatrix} g^2\mathbf{R} & 0 \\ 0 & (g+1)^2\mathbf{R} \end{pmatrix} \qquad \mathbf{Q} := \begin{pmatrix} \mathbf{R} & 0 \\ 0 & g^2(g+1)^2\mathbf{R} \end{pmatrix}.$$

Let $\mathbf{I}_n$ be the $n \times n$ identity matrix and consider the transformations:

$$\mathbf{U}_1 := \begin{pmatrix} g^{-1}\mathbf{I}_n & 0 \\ 0 & g\mathbf{I}_n \end{pmatrix} \qquad \mathbf{U}_2 := \begin{pmatrix} 0 & (g+1)\mathbf{I}_n \\ (g+1)^{-1}\mathbf{I}_n & 0 \end{pmatrix}.$$

Then $\mathbf{Q} = \mathbf{U}_1^t \mathbf{S} \mathbf{U}_1$ over $\mathbb{Q}$: this implies $[\mathbf{S}]_{\mathbb{Q}} = [\mathbf{Q}]_{\mathbb{Q}}$. For any prime p we have that $\gcd(g, p) = 1$ or $\gcd(g+1, p) = 1$. So g or $(g+1)$ is invertible over the p-adic integers $\mathbb{Z}_p$, and thus $\mathbf{U}_1 \in \mathcal{GL}_d(\mathbb{Z}_p)$ exists and $\mathbf{Q} = \mathbf{U}_1^t \mathbf{S} \mathbf{U}_1$ over $\mathbb{Z}_p$ or $\mathbf{U}_2 \in \mathcal{GL}_d(\mathbb{Z}_p)$ exists and $\mathbf{Q} = \mathbf{U}_2^t \mathbf{S} \mathbf{U}_2$ over $\mathbb{Z}_p$. In any case, we have established $[\mathbf{S}]_{\mathbb{Z}_p} = [\mathbf{Q}]_{\mathbb{Z}_p}$, which concludes the comparison of arithmetic invariants: $\mathrm{ari}(\mathbf{S}) = \mathrm{ari}(\mathbf{Q})$.

Dense sublattice. We now check the requirements for Theorem 192, namely that $\eta_{\frac{1}{2}}(\mathcal{L}) \leq \rho'/(2\sqrt{2n})$, where $\rho' = \Theta(g/f) \cdot \mathrm{gh}(\mathcal{L})$ is the decoding radius of $\mathcal{L}_{\mathbf{S}}$. Given that $\eta_{\frac{1}{2}}(\mathcal{L}) \leq \Theta(f \cdot \mathrm{gh}(\mathcal{L})/\sqrt{n})$, it is sufficient if

$$\Theta(f \cdot \mathrm{gh}(\mathcal{L})/\sqrt{n}) \leq \rho'/(2\sqrt{2n}) = \Theta(g/f) \cdot \mathrm{gh}(\mathcal{L})/\sqrt{n},$$

and thus we can conclude that some $g = \Theta(f^2)$ indeed suffices.

Following the conclusions from the cryptanalysis in Section 10.6.2 and more specifically Conjecture 195, we take a look at the primal-dual gap for $\mathcal{L}_{\mathbf{S}}$ and $\mathcal{L}_{\mathbf{Q}}$. We have that $\mathrm{gap}(\mathcal{L}_{\mathbf{S}}) = \Theta(\mathrm{gap}(\mathcal{L})) \leq O(f)$, and $\mathrm{gap}(\mathcal{L}_{\mathbf{Q}}) = \Theta(g \cdot \mathrm{gap}(\mathcal{L})) \leq O(f^3)$.

More specifically, following the same computation above but for a primal gap of f, dual gap of f^*, and a decoding gap of $f' \geq 2f$ we would have $g = \Theta(f^* \cdot f')$ and obtain a final primal-dual gap of $O(\max(f, f^*) \cdot f^* \cdot f')$.

10.7.2 Signature scheme

Our signature scheme can be instantiated with any lattice for which we can sample efficiently at small Gaussian widths, following a similar ΔLIP pair as above.

Namely, we assume to have a lattice $\mathcal{L}$ with $\text{gap}(\mathcal{L}) \leq f$ and such that we can sample a discrete Gaussian over $\mathcal{L}$ efficiently with parameter $\rho/\sqrt{n} = \Theta(\eta_{2^{-\Theta(n)}}(\mathcal{L}))$ close to the smoothing bound. Similarly to the KEM we set $\mathcal{L}_{\mathbf{S}} := g \cdot \mathcal{L} \oplus (g+1) \cdot \mathcal{L}$, and $\mathcal{L}_{\mathbf{Q}^{-1}} = \mathcal{L} \oplus g(g+1) \cdot \mathcal{L}$ for some integer $g \geq 1$. In particular, as in the KEM, we do have $\text{ari}(\mathbf{S}) = \text{ari}(\mathbf{Q}^{-1})$ with $\mathbf{Q}^{-1}$ instead of $\mathbf{Q}$.

Then for the dual we have $\mathcal{L}_{\mathbf{Q}} = \mathcal{L}^* \oplus \frac{1}{g(g+1)}\mathcal{L}^*$, with $\frac{1}{g(g+1)}\mathcal{L}^*$ as a dense sublattice. The constraint of Theorem 194 boils down to the inequality $\Theta(g \cdot f \cdot \det(\mathcal{L})^{1/n}) \leq \Theta(g^2 \det(\mathcal{L})^{1/n})$, and thus some $g = \Theta(f)$ suffices. The final primal-dual gap of $\mathcal{L}_{\mathbf{S}}$ and $\mathcal{L}_{\mathbf{Q}^{-1}}$ is then bounded by $O(f^2)$.

The simplest lattice for which we have very efficient samplers is of course the integer lattice $\mathbb{Z}^n$, leading to a gap of $O(n)$ via the above construction. Instantiating our scheme with this lattice would lead to an interesting signature scheme where there is no need to compute any Cholesky decomposition, even for signing, and that could be fully implemented with efficient integer arithmetic.

We refer to our last open question (Section 10.1.3) regarding lattices with a tighter Gaussian sampler, in order to obtain a signature scheme with a better underlying approximation factor.

10.7.3 Getting down to $O(f)$

The general constructions presented turn a well-decodable or well-sampleable lattice $\mathcal{L}$ with gaps f into a primary and secondary lattice with gap $O(f^3)$ and $O(f^2)$ to instantiate our KEM and signature scheme respectively. We suggest here that these losses from $O(f)$ to $O(f^3)$ and $O(f^2)$ respectively might be an artifact of the security proof and our generic instantiation construction.

Suppose we can generate a random lattice $\mathcal{L}_{\mathbf{Q}}$ such that $\text{ari}(\mathcal{L}_{\mathbf{Q}}) = \text{ari}(\mathcal{L})$; without the arithmetic constraint we would have with overwhelming probability that $\text{gap}(\mathcal{L}_{\mathbf{Q}}) = O(1)$ (but even $O(f)$ would suffice). Let's assume that the constraint does not affect this gap. Then similar to the scheme of McEliece, by adding the extra security

assumption that it is hard to decode in $\mathcal{L}_{\mathbf{Q}}$ (or hard to sample for the signature scheme), we could remove the lossyness argument from the security proof and directly instantiate our schemes with the pair $(\mathcal{L}, \mathcal{L}_{\mathbf{Q}})$, leading to a gap of $O(f)$.

10.7.4 Remarkable lattices

In Table 10.1 we show a list of some remarkable lattices that have efficient decoding algorithms together with their primal, dual and decoding gap. These are interesting candidates for instantiating our KEM, especially those that reach poly-logarithmic decoding gaps. Unfortunately, for all candidates at least one gap exceeds the poly-logarithmic regime, and thus we do not yet obtain a KEM that is secure up to poly-logarithmic approximation factors. Note, however, that the best decoders in the list are very recent, and that there is no (obvious) geometric obstacle for the existence of an efficiently decodable lattice reaching poly-logarithmic primal, dual and decoding gaps.

For the signature scheme we have yet to find a lattice for which one can do discrete Gaussian sampling significantly better than the generic randomized Babai approach. Potentially the list decoding [GP12] algorithm for the Barnes-Wall lattice can be turned into a sampler. From a practical perspective, the lattice $\mathbb{Z}^n$ is extremely interesting, as it allows to sample very efficiently (and in parallel), with similar or even better parameters than those based on LWE and NTRU trapdoor lattices.

10.8 HAWK: fast, compact and simple

The end goal of LIP based cryptography is to build lattice-based schemes that significantly improve upon the state-of-the-art based on LWE, SIS and NTRU assumptions. Until then, we show here that LIP is already competitive with the state-of-the-art using a lattice as simple as $\mathbb{Z}^d$.

We shortly present and summarize the signature scheme Hawk [DPPW22], as a practical instantiation of the signature scheme from Section 10.5. Hawk is a hash-then-sign signature scheme similar to the to be standardized scheme Falcon. Hawk significantly improves over Falcon in its simplicity, and lack of floating point operations

	[Pre+20] Falcon 512	This work Hawk 512	Gain $\left(\frac{\text{Falcon}}{\text{Hawk}}\right)$
AVX2 **KeyGen**	7.95 ms	4.25 ms	×1.87
Reference **KeyGen**	19.32 ms	13.14 ms	×1.47
AVX2 **Sign**	193 µs	50 µs	× 3.9
Reference **Sign**	2449 µs	168 µs	×14.6
AVX2 **Verify**	50 µs	19 µs	×2.63
Reference **Verify**	53 µs	178 µs	×0.30
Secret key (bytes)	1281	1153	×1.11
Public key (bytes)	897	1006 ± 6	×0.89
Signature (bytes)	652 ± 3	542 ± 4	×1.21

Table 10.3: Performance of Falcon and Hawk for $n = 512, 1024$ on an Intel® Core™ i5-4590 @3.30GHz processor with TurboBoost disabled. The **Sign** timings correspond to batch usage.

in signing and verification. This is all due to the simplicity of the underlying lattice $\mathbb{Z}^d$ compared to the more complicated NTRU lattice of Falcon.

Discrete sampling in (cosets of) $\mathbb{Z}^d$ is almost trivial and can be done coordinate wise. The Gaussian parameter s, and thus the signatures, can also be a relative factor 1.17 smaller than that of Falcon due to the orthogonality of $\mathbb{Z}^d$. To make the sampling even more practically efficient and easy to implement in constant time we restrict the target to the cosets $\{0, \frac{1}{2}\}^d + \mathbb{Z}^d$, such that we can use two precomputed tables for the discrete Gaussian: one for $\mathbb{Z}$ and one for $\frac{1}{2} + \mathbb{Z}$. Note that any short coset representative $\mathbf{t}$ also gives a short vector $2\mathbf{t} \in \mathbb{Z}^d$. With the appropriate parameters the concrete cryptanalysis shows that these vectors can be leaked safely.

To limit the size of the keys and signatures, and the arithmetical operations, we instantiate $\mathbb{Z}^d$ as a rank 2 module lattice. The number ring $R = \mathbb{Z}[X]/(X^n + 1)$ for $n = d/2$ is naturally orthogonal under the trace norm and can thus be identified with $\mathbb{Z}^n$. Then we simply take $R^2 = R \oplus R$ as a rank 2 R-module which can be identified with

$\mathbb{Z}^d$. A basis $\mathbf{B} \in R^{2\times 2}$ of R^2 consists of only 4 ring elements. Instead of quadratic forms we have to work with hermitian forms $\mathbf{Q} = \mathbf{B}^*\mathbf{B}$, where $\mathbf{B}^*$ denotes the adjoint transpose of $\mathbf{B}$. The inner product and norm w.r.t. $\mathbf{Q}$ can be defined in an analogues way from the original trace inner product.

What remains is to discuss a sampler for the hermitian forms $\mathbf{Q}$ corresponding to R^2 serving as a public key. Given that $\mathbf{I}_2$ is a basis of R^2 the basis transformations and the bases themself coincide. I.e., we only have to consider how to sample bases $\mathbf{B} \in \mathcal{SL}_2(R)$. Surprisingly the natural way to do this is similar to the key generation of FALCON. First we sample a vector $(f, g)^\top \in R^2$ each element according to a discrete Gaussian, which we then have to complete with another vector $(F, G)^\top \in \mathbb{R}^2$ to a full basis $\mathbf{B}$ with determinant $f \cdot G - g \cdot F = 1$. The FALCON key generation does exactly the same, except that there the final basis must have determinant q, and FALCON's efficient algorithms for this can simply be adapted to suit the needs of HAWK. They key generation is thus as follows: sample $(f, g) \in R^2$, extend to a basis $\mathbf{B}$ of R^2, and return $(\mathbf{sk} = \mathbf{B}, \mathbf{pk} = \mathbf{B}^*\mathbf{B})$. A similar worst-case to average-case reduction as in Section 10.2 can be shown for the resulting distribution.

The key and signature sizes can be further reduced. For example the public key $\mathbf{Q}$ is hermitian: the symmetry, the self-adjointness of the diagonal elements, and the relation $\det(\mathbf{Q}) = 1$, allow to throw away (and later reconstruct) a lot of information. Similarly we can throw away the first half of the signature, and reconstruct the other half simply by size-reduction with the hermitian form. The resulting scheme is very competitive with FALCON, see Table 10.3. The key and signature sizes for NIST security level 1 are about the same, while HAWK is about 1.5× to 4× faster with **KeyGen**, **Sign** and **Verify** than FALCON. But HAWK really shines on low-end hardware without floating-point support. Due to the simplicity and floating-point free sampling over $\mathbb{Z}^n$ the reference **Sign** implementation is more than 14× faster than FALCON. Solving a big disadvantage of the FALCON scheme.

Bibliography

[ABD16] Martin Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions. In: Springer, 2016, pages 153–178.

[AD21] Martin Albrecht and Léo Ducas. Lattice Attacks on NTRU and LWE: A History of Refinements. In: *Computational Cryptography: Algorithmic Aspects of Cryptology*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021, pages 15–40.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum Key Exchange–A New Hope. In: 2016, pages 327–343.

[AEN19] Yoshinori Aono, Thomas Espitau, and Phong Q. Nguyen. Random Lattices: Theory And Practice. Available at https://espitau.github.io/bin/random_lattice.pdf. 2019.

[AFG13] Martin R. Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In: Springer, 2013, pages 293–310.

[AGPS20] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In: Springer, 2020, pages 583–613.

[AGVW17] Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pages 297–322.

[Ajt99] Miklós Ajtai. Generating Hard Instances of the Short Basis Problem. In: *ICALP*. 1999, pages 1–9.

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

[AL22] Martin R. Albrecht and Jianwei Li. Predicting BKZ Z-Shapes on q-ary Lattices. Cryptology ePrint Archive, Report 2022/843. 2022.

[Alb+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugere, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. In: *Designs, Codes and Cryptography* 74.2 (2015), pages 325–354.

[Alb+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pages 717–746.

[ALL19] Nicolas Aragon, Julien Lavauzelle, and Matthieu Lequesne. Decoding Challenge. Available at `http://decodingchallenge.org`. 2019.

[AN17] Yoshinori Aono and Phong Q. Nguyen. Random sampling revisited: lattice enumeration with discrete pruning. In: *Eurocrypt*. 2017, pages 65–102.

[ANS18] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In: *Asiacrypt*. 2018, pages 405–434.

[AP11] Joël Alwen and Chris Peikert. Generating Shorter Bases for Hard Random Lattices. In: *Theory of Computing Systems* 48.3 (Apr. 2011). Preliminary version in STACS 2009, pages 535–553.

[AR05] Dorit Aharonov and Oded Regev. Lattice problems in NP ∩ coNP. In: *J. ACM* 52.5 (2005). Preliminary version in FOCS 2004, pages 749–765.

[AUV19] Divesh Aggarwal, Bogdan Ursu, and Serge Vaudenay. Faster sieving algorithm for approximate SVP with constant approximation factors. Cryptology ePrint Archive, Report 2019/1028. 2019.

[AWHT16] Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In: Springer, 2016, pages 789–819.

[Bab16] László Babai. Graph isomorphism in quasipolynomial time. In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 2016, pages 684–697.

[Bab19] László Babai. Canonical form for graphs in quasipolynomial time: preliminary report. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 2019, pages 1237–1246.

[Bab86] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. In: *Combinatorica* 6.1 (1986). Preliminary version in STACS 1985, pages 1–13.

[Ban93] W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. In: *Mathematische Annalen* 296.4 (1993), pages 625–636.

[Bar+11] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In: *Annual Cryptology Conference*. Springer. 2011, pages 1–20.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In: *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2016, pages 10–24.

[Ber+20] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. *NTRU Prime*. Technical report. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020.

[BGJ15] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. 2015.

[Bia+17] Jean-François Biasse, Thomas Espitau, Pierre-Alain Fouque, Alexandre Gélin, and Paul Kirchner. Computing generator in cyclotomic integer rings. In: *Eurocrypt*. Springer. 2017, pages 60–88.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2 n/20: How 1+ 1= 0 improves information set decoding. In: Springer, 2012, pages 520–536.

[BKL80] László Babai, Paul Klingsberg, and Eugene M. Luks. Canonical labeling for vertex colored graphs. In: *To appear* (1980).

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In: *Journal of the ACM (JACM)* 50.4 (2003), pages 506–519. ISSN: 0004-5411.

[BL16] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In: *International Conference on Cryptology in Africa*. Springer. 2016, pages 3–23.

[BL83] László Babai and Eugene M. Luks. Canonical labeling of graphs. In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing*. 1983, pages 171–183.

[Bla+20] Pierre Blanchard, Nicholas J. Higham, Florent Lopez, Theo Mary, and Srikara Pranesh. Mixed Precision Block Fused Multiply-Add: Error Analysis and Application to GPU Tensor Cores. In: *SIAM Journal on Scientific Computing* 42.3 (2020), pages C124–C141.

[BLLN13] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In: Springer, 2013, pages 45–64.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. In: *LMS Journal of Computation and Mathematics* 19.A (2016), pages 146–162.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In: Springer, 2018, pages 25–46.

[BM21] Tamar Lichter Blanks and Stephen D. Miller. Generating cryptographically-strong random lattice bases and recognizing rotations of $\mathbb{Z}^n$. In: *CoRR* (2021).

[BMPS20] Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. LESS is more: Code-based signatures without syndromes. In: *International Conference on Cryptology in Africa*. Springer. 2020, pages 45–65.

[BN09] Johannes Blömer and Stefanie Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. In: *Theoretical Computer Science* 410.18 (2009), pages 1648–1665. ISSN: 0304-3975.

[BNP17] Joppe W. Bos, Michael Naehrig, and Joop Van De Pol. Sieving for shortest vectors in ideal lattices: a practical perspective. In: *International Journal of Applied Cryptography* 3.4 (2017), pages 313–329.

[Bog01] Michael I. Boguslavsky. Radon transforms and packings. In: *Discrete applied mathematics* 111.1-2 (2001), pages 3–22.

[BP22] Huck Bennett and Chris Peikert. Hardness of the (Approximate) Shortest Vector Problem: A Simple Proof via Reed-Solomon Codes. arXiv preprint arXiv:2202.07736. 2022.

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

[BSW18] Shi Bai, Damien Stehlé, and Weiqiang Wen. Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In: Springer, 2018, pages 369–404.

[CDPR16] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In: *Eurocrypt*. Springer. 2016, pages 559–585.

[CG05] Michael Coglianese and Bok-Min Goi. MaTRU: A new NTRU-based cryptosystem. In: *International Conference on Cryptology in India*. Springer. 2005, pages 232–243.

[Cha02] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 2002, pages 380–388.

[Che+20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. *NTRU*. Technical report. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low-level encoding of zero. In: *LMS Journal of Computation and Mathematics* 19.A (2016), pages 255–266.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In: *Asiacrypt*. 2011, pages 1–20.

[COT16] Alain Couvreur, Ayoub Otmani, and Jean-Pierre Tillich. Polynomial time attack on wild McEliece over quadratic extensions. In: *IEEE Transactions on Information Theory* 63.1 (2016), pages 404–427. ISSN: 0018-9448.

[CR88] Benny Chor and Ronald L. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. In: *IEEE Transactions on Information Theory* 34.5 (1988), pages 901–909.

[CS13] John Horton Conway and Neil James Alexander Sloane. Sphere packings, lattices and groups. Volume 290. Springer Science & Business Media, 2013.

[CS97] Don Coppersmith and Adi Shamir. Lattice Attacks on NTRU. In: *Eurocrypt*. 1997, pages 52–61.

[Dam02] Ivan Damgård. On Σ-protocols. In: *Lecture Notes, University of Aarhus, Department for Computer Science* (2002).

[DB15] Daniel Dadush and Nicolas Bonifas. Short paths on the Voronoi graph and closest vector problem with preprocessing. In: *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2015, pages 295–314.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: attacks and concrete security estimation. In: *Annual International Cryptology Conference*. Springer. 2020, pages 329–358.

[DDW22] Thomas Debris-Alazard, Leo Ducas, and Wessel van Woerden. An Algorithmic Reduction Theory for Binary Codes: LLL and more. In: *IEEE Transactions on Information Theory* (2022), pages 1–1.

[DE+04] Persi Diaconis, Paul Erdös, et al. On the distribution of the greatest common divisor. In: *A festschrift for Herman Rubin*. Institute of Mathematical Statistics, 2004, pages 56–61.

[DG22] Léo Ducas and Shane Gibbons. Hull Attacks on the Lattice Isomorphism Problem. To appear. 2022.

[DHVW20] Mathieu Dutour Sikirić, Anna Haensch, John Voight, and Wessel van Woerden. A canonical form for positive definite matrices. In: *ANTS XIV* 4.1 (2020), pages 179–195.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. In: *Numerische Mathematik* 1.1 (1959), pages 269–271.

[DLW19] Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger. Finding closest lattice vectors using approximate Voronoi cells. In: *PQCrypto*. 2019.

[DLW20a] Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger. Sieve, Enumerate, Slice, and Lift. In: Springer, 2020, pages 301–320.

[DLW20b] Léo Ducas, Thijs Laarhoven, and Wessel van Woerden. The randomized slicer for CVPP: sharper, faster, smaller, batchier. In: *IACR International Conference on Public-Key Cryptography*. Springer. 2020, pages 3–36.

[DM13] Daniel Dadush and Daniele Micciancio. Algorithms for the densest sub-lattice problem. In: *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2013, pages 1103–1122.

[DMW22] Mathieu Dutour Sikirić, Alexander Magazinov, and Wessel van Woerden. The complete classification of six-dimensional C-types. In: *To appear.* (2022).

[DP19] Léo Ducas and Cécile Pierrot. Polynomial time bounded distance decoding near minkowski's bound in discrete logarithm lattices. In: *Designs, Codes and Cryptography* 87.8 (2019), pages 1737–1748.

[DPPW22] Léo Ducas, Eamonn W. Postlethwaite, Ludo N. Pulles, and Wessel van Woerden. Hawk: Module LIP makes Lattice Signatures Fast, Compact and Simple. In: *Asiacrypt, 28th Annual International Conference* (2022).

[DST19] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. In: Springer, 2019, pages 21–51.

[DSW21] Léo Ducas, Marc Stevens, and Wessel van Woerden. Advanced Lattice Sieving on GPUs, with Tensor Cores. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2021, pages 249–279.

[Duc18] Léo Ducas. Shortest vector from lattice sieving: a few dimensions for free. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pages 125–145.

[Duc22] Léo Ducas. Estimating the Hidden Overheads in the BDGL Lattice Sieving Algorithm. In: *International Conference on Post-Quantum Cryptography*. Springer. 2022, pages 480–497.

[Dum91] Ilya Dumer. On minimum distance decoding of linear codes. In: 1991, pages 50–52.

[Dut22] Mathieu Dutour Sikirić. Polytopes, lattices and quadratic forms programs. Available at https://github.com/MathieuDutSik/polyhedral_common. 2022.

[DW18] Léo Ducas and Wessel van Woerden. The closest vector problem in tensored root lattices of type A and in their duals. In: *Designs, Codes and Cryptography* 86.1 (2018), pages 137–150.

[DW21] Léo Ducas and Wessel van Woerden. NTRU Fatigue: How Stretched is Overstretched? In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2021, pages 3–32.

[DW22] Léo Ducas and Wessel van Woerden. On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pages 643–673.

[EHKS14] Kirsten Eisenträger, Sean Hallgren, Alexei Kitaev, and Fang Song. A quantum algorithm for computing the unit group of an arbitrary degree number field. In: *STOC*. ACM. 2014, pages 293–302.

[EV22] Friedrich Eisenbrand and Moritz Venzin. Approximate CVPp in time 20.802 n. In: *Journal of Computer and System Sciences* 124 (2022), pages 129–139. ISSN: 0022-0000.

[Fit+14] Robert Fitzpatrick, Christian Bischof, Johannes Buchmann, Özgür Dagdelen, Florian Göpfert, Artur Mariano, and Bo-Yin Yang. Tuning GaussSieve for speed. In: *International Conference on Cryptology and Information Security in Latin America*. Springer. 2014, pages 288–305.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In: Springer, 1999, pages 537–554.

[FP85] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. In: *Mathematics of computation* 44.170 (1985), pages 463–471.

[FSS83] Martin Fürer, Walter Schnyder, and Ernst Specker. Normal forms for trivalent graphs and graphs of bounded valence. In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing*. 1983, pages 161–170.

[FSW14] Felix Fontein, Michael Schneider, and Urs Wagner. PotLLL: a polynomial time version of LLL with deep insertions. In: *Designs, codes and cryptography* 73.2 (2014), pages 355–368. ISSN: 1573-7586.

[Gab85] Ernest Mukhamedovich Gabidulin. Theory of codes with maximum rank distance. In: *Problemy peredachi informatsii* 21.1 (1985), pages 3–16. ISSN: 0555-2923.

[Gab95] Ernst M. Gabidulin. Public-key cryptosystems based on linear codes. Citeseer, 1995.

[Gen+19] Nicholas Genise, Craig Gentry, Shai Halevi, Baiyu Li, and Daniele Micciancio. Homomorphic encryption for finite automata. In: Springer, 2019, pages 473–502.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate Multilinear Maps from Ideal Lattices. In: *Eurocrypt*. 2013, pages 1–17.

[GHKN06] Nicolas Gama, Nick Howgrave-Graham, Henrik Koy, and Phong Q. Nguyen. Rankin's constant and blockwise lattice reduction. In: *Annual International Cryptology Conference*. Springer. 2006, pages 112–130.

[GMSS99] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and Jean-Pierre Seifert. Approximating Shortest Lattice Vectors is not Harder than Approximating Closest Lattice Vectors. In: *Inf. Process. Lett.* 71.2 (1999), pages 55–61.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. In: *Journal of the ACM (JACM)* 38.3 (1991), pages 690–728.

[GN08a] Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within mordell's inequality. In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM. 2008, pages 207–216.

[GN08b] Nicolas Gama and Phong Q. Nguyen. Predicting Lattice Reduction. In: *Eurocrypt*. 2008, pages 31–51.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice Enumeration Using Extreme Pruning. In: *Eurocrypt*. 2010, pages 257–278.

[GP12] Elena Grigorescu and Chris Peikert. List Decoding Barnes-Wall Lattices. In: *IEEE Conference on Computational Complexity*. 2012, pages 316–325.

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

[Gri60] James H. Griesmer. A bound for error-correcting codes. In: *IBM Journal of Research and Development* 4.5 (1960), pages 532–542. ISSN: 0018-8646.

[GS02] Craig Gentry and Michael Szydlo. Cryptanalysis of the Revised NTRU Signature Scheme. In: *Eurocrypt*. 2002, pages 299–320.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. In: *SIAM Journal on Computing* 28.4 (1999), pages 1364–1396.

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in Euclidean norm. In: *IACR International Workshop on Public Key Cryptography*. Springer. 2017, pages 16–40.

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time–memory trade-offs for tuple lattice sieving. In: *IACR International Workshop on Public Key Cryptography*. Springer. 2018, pages 407–436.

[HM19] Nicholas J. Higham and Theo Mary. A new approach to probabilistic rounding error analysis. In: *SIAM Journal on Scientific Computing* 41.5 (2019), A2815–A2835.

[HP10] W. Cary Huffman and Vera Pless. Fundamentals of error-correcting codes. Cambridge university press, 2010.

[HPS11a] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In: *International Conference on Coding and Cryptology*. Springer, 2011, pages 159–190.

[HPS11b] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In: Springer, 2011, pages 447–464.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In: *ANTS*. 1998, pages 267–288.

[HR14] Ishay Haviv and Oded Regev. On the lattice isomorphism problem. In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pages 391–404.

[IKMT14] Tsukasa Ishiguro, Shinsaku Kiyomoto, Yutaka Miyake, and Tsuyoshi Takagi. Parallel Gauss sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In: *International Workshop on Public Key Cryptography*. Springer, 2014, pages 411–428.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In: 1998, pages 604–613.

[JK07] Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In: *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2007, pages 135–149.

[Kan83] Ravi Kannan. Improved Algorithms for Integer Programming and Related Lattice Problems. In: *STOC*. 1983, pages 193–206.

[KB79] Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. In: *siam Journal on Computing* 8.4 (1979), pages 499–507.

[Ker+22] Andrew Kerr, Haicheng Wu, Manish Gupta, Dustyn Blasig, Pradeep Ramini, Duane Merrill, Aniket Shivam, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Matt Nicely. *CUTLASS*. Version 2.9. Available at https://github.com/NVIDIA/cutlass. Apr. 2022.

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In: Springer, 2017, pages 3–26.

[KL20] Jonathan Katz and Yehuda Lindell. Introduction to modern cryptography. CRC press, 2020.

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

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In: *Annual Cryptology Conference*. Springer. 2015, pages 3–22.

[Laa16] Thijs Laarhoven. Sieving for closest lattice vectors (with preprocessing). In: *SAC*. 2016, pages 523–542.

[Laa19] Thijs Laarhoven. Approximate Voronoi cells for lattices, revisited. In: *MathCrypt*. 2019.

[Lap21] Oleksandra Lapiha. Comparing Lattice Families for Bounded Distance Decoding near Minkowski's Bound. Cryptology ePrint Archive, Report 2021/1052. 2021.

[LB88] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In: Springer, 1988, pages 275–280.

[Len91] Hendrik W Lenstra. On the Chor-Rivest knapsack cryptosystem. In: *Journal of Cryptology* 3.3 (1991), pages 149–155.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra Jr., and László Lovász. Factoring polynomials with rational coefficients. In: *Mathematische Annalen* 261.4 (Dec. 1982), pages 515–534.

[LLXY20] Zhe Li, San Ling, Chaoping Xing, and Sze Ling Yeo. On the bounded distance decoding problem for lattices constructed and their cryptographic applications. In: *IEEE Transactions on Information Theory* 66.4 (2020), pages 2588–2598.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem. In: *Crypto*. 2009, pages 577–594.

[LM18] Thijs Laarhoven and Artur Mariano. Progressive lattice sieving. In: *International Conference on Post-Quantum Cryptography*. Springer. 2018, pages 292–311.

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

[LN20] Jianwei Li and Phong Q. Nguyen. A complete analysis of the BKZ lattice reduction algorithm. Cryptology ePrint Archive, Report 2020/1237. 2020.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors Over Rings. In: *Journal of the ACM* 60.6 (Nov. 2013). Preliminary version in Eurocrypt 2010, 43:1–43:35.

[LS14] Hendrik W. Lenstra and Alice Silverberg. Revisiting the gentry-szydlo algorithm. In: *Annual Cryptology Conference*. Springer. 2014, pages 280–296.

[LW20] Changmin Lee and Alexandre Wallet. Lattice analysis on MiNTRU problem. Cryptology ePrint Archive, Report 2020/230. 2020.

[MBL15] Artur Mariano, Christian Bischof, and Thijs Laarhoven. Parallel (probable) lock-free hash sieve: A practical sieving algorithm for the SVP. In: *2015 44th International Conference on Parallel Processing*. IEEE, 2015, pages 590–599.

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. In: *Coding Thv* 4244 (1978), pages 114–116.

[McK07] Brendan D. McKay. Nauty user's guide (version 2.4). In: *Computer Science Dept., Australian National University* (2007), pages 225–239.

[McK81] Brendan D. McKay. Practical graph isomorphism. In: (1981).

[MDB14] Artur Mariano, Özgür Dagdelen, and Christian Bischof. A comprehensive empirical comparison of parallel ListSieve and GaussSieve. In: *European Conference on Parallel Processing*. Springer, 2014, pages 48–59.

[MG02] Daniele Micciancio and Shafi Goldwasser. Complexity of Lattice Problems: a cryptographic perspective. Volume 671. The Kluwer International Series in Engineering and Computer Science. Boston, Massachusetts: Kluwer Academic Publishers, 2002.

[Mic08] Daniele Micciancio. Efficient reductions among lattice problems. In: *SODA*. 2008, pages 84–93.

[Min97] Hermann Minkowski. Allgemeine Lehrsatze uber die konvexen Polyeder. In: *Nachr. Ges. Wiss. Gottingen, Math.-Phys. KL* (1897), pages 198–219.

[MLB17] Artur Mariano, Thijs Laarhoven, and Christian Bischof. A parallel variant of LDSieve for the SVP on lattices. In: *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE. 2017, pages 23–30.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Springer, 2011, pages 107–124.

[MN08] Daniele Micciancio and Antonio Nicolosi. Efficient bounded distance decoders for Barnes-Wall lattices. In: *2008 IEEE International Symposium on Information Theory*. IEEE. 2008, pages 2484–2488.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In: Springer, 2015, pages 203–228.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller. In: *Eurocrypt*. 2012, pages 700–718.

[MP14] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. In: *Journal of symbolic computation* 60 (2014), pages 94–112.

[MP21] Ethan Mook and Chris Peikert. Lattice (List) Decoding Near Minkowski's Inequality. In: *IEEE Transactions on Information Theory* (2021). ISSN: 0018-9448.

[MR07] Daniele Micciancio and Oded Regev. Worst-Case to Average-Case Reductions Based on Gaussian Measures. In: *SIAM J. Comput.* 37.1 (Apr. 2007), pages 267–302. ISSN: 0097-5397.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based Cryptography. In: *Post Quantum Cryptography*. Springer, Feb. 2009, pages 147–191.

[MS01] Alexander May and Joseph H. Silverman. Dimension reduction methods for convolution modular lattices. In: *International Cryptography and Lattices Conference*. Springer. 2001, pages 110–125.

[MS11] Benjamin Milde and Michael Schneider. A parallel implementation of GaussSieve for the shortest vector problem in lattices. In: *International Conference on Parallel Computing Technologies*. Springer, 2011, pages 452–458.

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. The theory of error correcting codes. Volume 16. Elsevier, 1977.

[MTB14] Artur Mariano, Shahar Timnat, and Christian Bischof. Lock-free GaussSieve for linear speedups in parallel high performance SVP calculation. In: *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*. IEEE, 2014, pages 278–285.

[Mul54] David E. Muller. Application of Boolean algebra to switching circuit design and to error detection. In: *Transactions of the IRE professional group on electronic computers* 3 (1954), pages 6–12. ISSN: 2168-1740.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster Exponential Time Algorithms for the Shortest Vector Problem. In: *SODA*. 2010, pages 1468–1480.

[MV13] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In: *SIAM Journal on Computing* 42.3 (2013), pages 1364–1391.

[MW01] Daniele Micciancio and Bogdan Warinschi. A linear space algorithm for computing the Hermite normal form. In: *ISSAC*. 2001, pages 231–236.

[MW16] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, pages 820–849.

[NBGS08] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. In: *Queue* 6.2 (2008), pages 40–53.

[NS11] Gabriele Nebe and Neil Sloane. Table of the Highest Kissing Numbers Presently Known. Available at http://www.math.rwth-aachen.de/~Gabriele.Nebe/LATTICES/kiss.html. 2011.

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve Algorithms for the Shortest Vector Problem Are Practical. In: *Journal of Mathematical Cryptology* (2008).

[NVF20] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. CUDA, release: 10.2.89. Available at https://developer.nvidia.com/cuda-toolkit. 2020.

[NVI] NVIDIA. cuBLAS: Basic Linear Algebra on NVIDIA GPUs. Available at https://developer.nvidia.com/cublas.

[Odl90] Andrew M. Odlyzko. The rise and fall of knapsack cryptosystems. In: *Cryptology and Computational Number Theory*. Edited by C. Pomerance. Volume 42. Proceedings of Symposia in Applied Mathematics. 1990, pages 75–88.

[OS09] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In: Springer, 2009, pages 95–145.

[OTU00] Tatsuaki Okamoto, Keisuke Tanaka, and Shigenori Uchiyama. Quantum public-key cryptosystems. In: *Annual international cryptology conference*. Springer. 2000, pages 147–165.

[Pei10] Chris Peikert. An Efficient and Parallel Gaussian Sampler for Lattices. In: *Crypto*. 2010, pages 80–97.

[Poh87] Michael Pohst. A modification of the LLL reduction algorithm. In: *Journal of Symbolic Computation* 4.1 (1987), pages 123–127. ISSN: 0747-7171.

[PP02] Athanasios Papoulis and S Unnikrishna Pillai. Probability, random variables, and stochastic processes. Tata McGraw-Hill Education, 2002.

[PP85] Wilhelm Plesken and Michael Pohst. Constructing integral lattices with prescribed minimum. I. In: *mathematics of computation* 45.171 (1985), pages 209–221.

[PR06] Chris Peikert and Alon Rosen. Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices. In: *TCC*. 2006, pages 145–166.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. In: *IRE Transactions on Information Theory* 8.5 (1962), pages 5–9. ISSN: 0096-1000.

[Pre+20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *FALCON*. Technical report. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020.

[PS09] Xavier Pujol and Damien Stehlé. Solving the Shortest Lattice Vector Problem in Time 22.465 n. 2009.

[PS97] Wilhelm Plesken and Bernd Souvignier. Computing isometries of lattices. In: *Journal of Symbolic Computation* 24.3-4 (1997), pages 327–334.

[PT08] Gábor Pataki and Mustafa Tural. On sublattice determinants in reduced bases. arXiv preprint arXiv:0804.4014. 2008.

[PV21] Eamonn W. Postlethwaite and Fernando Virdia. On the success probability of solving unique SVP via BKZ. In: Springer, 2021, pages 68–98.

[PW11] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In: *SIAM Journal on Computing* 40.6 (2011), pages 1803–1844.

[Rab79] Michael O. Rabin. *Digitalized signatures and public-key functions as intractable as factorization.* Technical report. 1979.

[Ree53] Irving S. Reed. *A class of multiple-error-correcting codes and the decoding scheme.* Technical report. 1953.

[Reg04] Oded Regev. Quantum Computation and Lattice Problems. In: *SIAM J. Comput.* 33.3 (2004). Preliminary version in FOCS 2002, pages 738–760.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In: *J. ACM* 56.6 (2009). Preliminary version in STOC 2005, pages 1–40.

[RV22] Thomas Rothvoss and Moritz Venzin. Approximate in Time-Now in Any Norm! In: Springer, 2022, pages 440–453.

[Sch03] Claus-Peter Schnorr. Lattice Reduction by Random Sampling and Birthday Methods. In: *STACS*. 2003, pages 145–156.

[Sch87] Claus-Peter Schnorr. A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. In: *Theor. Comput. Sci.* 53 (1987), pages 201–224.

[Sch94] Claus-Peter Schnorr. Block reduced lattice bases and successive minima. In: *Combinatorics, Probability and Computing* 3.4 (1994), pages 507–522. ISSN: 1469-2163.

[SCM01] Patrick Solé, Chris Charnes, and Bruno Martin. A lattice-based McEliece scheme for encryption and signature. In: *Electronic Notes in Discrete Mathematics* 6 (2001), pages 402–411.

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

[Sen00] Nicolas Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. In: *IEEE Transactions on Information Theory* 46.4 (2000), pages 1193–1203.

[SFS09] Naftali Sommer, Meir Feder, and Ofir Shalvi. Finding the closest lattice point by iterative slicing. In: *SIAM Journal on Discrete Mathematics* 23.2 (2009), pages 715–731.

[SG10] Michael Schneider and Nicolas Gama. Darmstadt SVP Challenges. Accessed: 16-02-2022. 2010.

[Sho94] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In: *FOCS*. 1994, pages 124–134.

[Sie45] Carl Ludwig Siegel. A mean value theorem in geometry of numbers. In: *Annals of Mathematics* (1945), pages 340–347. ISSN: 0003-486X.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient Public Key Encryption Based on Ideal Lattices. In: *Asiacrypt*. 2009, pages 617–635.

[SSV07] Mathieu Sikirić, Achill Schürmann, and Frank Vallentin. Classification of eight-dimensional perfect forms. In: *Electronic Research Announcements of the American Mathematical Society* 13.3 (2007), pages 21–32.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In: Springer, 1988, pages 106–113.

[Szy03] Michael Szydlo. Hypercubic lattice reduction and analysis of GGH and NTRU signatures. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2003, pages 433–448.

[tea21a] The FPLLL development team. FPLLL, a lattice reduction library, Version: 5.4.1. Available at https://github.com/fplll/fplll. 2021.

[tea21b] The FPLLL development team. fpylll, a Python wraper for the fplll lattice reduction library, Version: 0.5.6. Available at https://github.com/fplll/fpylll. 2021.

[TV95] Michael A. Tsfasman and Serge G. Vladut. Geometric approach to higher weights. In: *IEEE Transactions on Information Theory* 41.6 (1995), pages 1564–1588.

[Var97] Alexander Vardy. The intractability of computing the minimum distance of a code. In: *IEEE Transactions on Information Theory* 43.6 (1997), pages 1757–1766. ISSN: 0018-9448.

[Vlă19] Serge Vlăduţ. Lattices with exponentially large kissing numbers. In: *Moscow Journal of Combinatorics and Number Theory* 8.2 (2019), pages 163–177.

[Vor08] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs. In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1908.134 (1908), pages 198–287.

[Vor09] Georges Voronoi. Nouvelles applications des paramètres continus à théorie des formes quadratiques. Deuxième Mémoire. Recherches sur les paralléloèdres primitifs. In: *Journal für die reine und angewandte Mathematik* 1909.136 (1909), pages 67–182.

[Wal21] Michael Walter. The Convergence of Slide-type Reductions. In: Springer, 2021, pages 45–67.

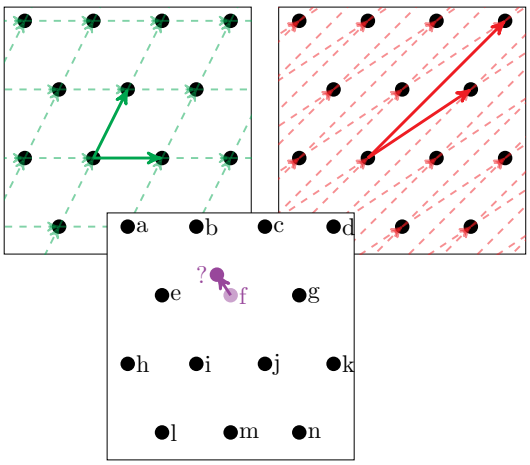
[Wei91] Victor K. Wei. Generalized Hamming weights for linear codes. In: *IEEE Transactions on information theory* 37.5 (1991), pages 1412–1418.

[Woe18] Wessel P.J. van Woerden. Perfect quadratic forms: an upper bound and challenges in enumeration. Master's thesis. Leiden University, 2018.

[Woe20] Wessel van Woerden. An upper bound on the number of perfect quadratic forms. In: *Advances in Mathematics* 365 (2020), page 107031.

[YD17] Yang Yu and Léo Ducas. Second order statistical behavior of LLL and BKZ. In: Springer, 2017, pages 3–22.

[YKYC17] Shang-Yi Yang, Po-Chun Kuo, Bo-Yin Yang, and Chen-Mou Cheng. Gauss sieve algorithm on GPUs. In: *Cryptographers' Track at the RSA Conference*. Springer. 2017, pages 39–57.



Samenvatting

Dit proefschrift gaat hoofdzakelijk over roosters. Niet in dimensie twee of drie waar we ons nog een beeld van kunnen scheppen, maar in honderden, soms wel duizenden dimensies. Deze hoge dimensies brengen een hoop computationele problemen met zich mee. Vragen die makkelijk zijn in twee dimensies, kunnen erg lastig zijn in hogere dimensies. En deze moeilijke problemen geven aanleiding tot cryptografie.

Cryptografie heeft als doel om communicatie veilig te maken in het geval de verbinding afgeluisterd of beïnvloed wordt. Verrassend genoeg kan dit zonder dat de communicerende partijen vooraf een geheime codetaal afgesproken hebben. Dit vereist problemen die in het algemeen lastig zijn, maar die toch opgelost kunnen worden als bepaalde geheime sleutelinformatie bekend is. Als alleen de ontvanger bekend is met die geheime sleutel, dan kan alleen die het bericht lezen, en niemand anders. Denk aan een kluis met een hangslot. Iedereen kan een bericht in de kluis stoppen en het slot dichtdrukken, maar alleen de eigenaar van de sleutel kan de kluis (moeiteloos) ontgrendelen.

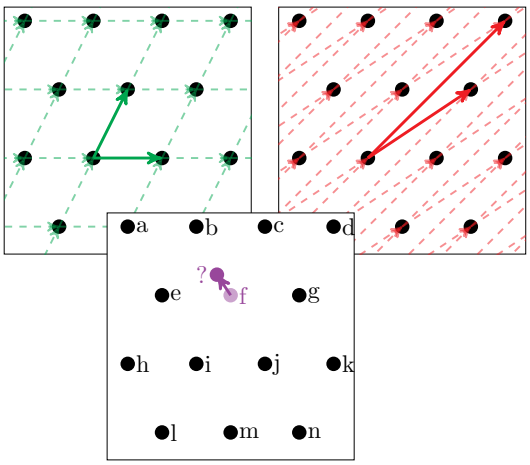
Roosterproblemen zijn uitermate geschikt voor zo'n versleuteling, met als extra voordeel dat ze zelfs niet door kwantumcomputers efficiënt opgelost lijken te kunnen worden. Het idee is als volgt. Een rooster kan, ondanks de hoge dimensie, omschreven worden door een paar pijlen (zie afbeelding). Een omschrijving met *lange pijlen* (rood) vormt hierbij de kluis met hangslot en een omschrijving met *korte pijlen* (groen) vormt hierbij de sleutel. Het vinden van zo'n korte omschrijving, gegeven een lange omschrijving, heet *reduceren* en dit is een moeilijk computationeel probleem. Als we een roosterpunt zien als een letter, dan kan deze versleuteld worden door het roosterpunt

iets te verplaatsen naar buiten het rooster. Het terugvinden van de oorspronkelijke locatie, en dus het ontcijferen van het bericht, heet *decoderen*. Decoderen is in het algemeen moeilijk, behalve met een korte omschrijving, de sleutel.

Deel II en III gaan over cryptanalyse met de centrale vraag: hoe moeilijk is het precies om het versleutelde bericht te lezen? We kunnen de sleutel achterhalen (reduceren), of het slot met geweld open breken (decoderen). In Hoofdstuk 4 laten we zien hoe we met moderne algoritmes en grafische kaarten in staat zijn om een rooster in dimensie 180 te reduceren, 25 dimensies meer dan het oude record. In Hoofdstuk 5 analyseren we een algoritme, dat na wat voorwerk, veel punten kan decoderen in hetzelfde rooster. In Hoofdstuk 6 bespreken we dat roosters die gebruikt worden in de cryptografie vaak wat vreemd gevormd zijn, bijvoorbeeld doordat ze roosterpunten bevatten die te dicht bij elkaar liggen. Deze afwijkingen leiden ertoe dat het reduceren of decoderen vaak aanzienlijk makkelijker is dan je zou verwachten, gegeven de hoge dimensie. We behandelen in Hoofdstuk 7 zogenoemde NTRU roosters, die al in de cryptografie gebruikt worden sinds 1996. Deze roosters hebben in sommige gevallen een nog grotere afwijking dan verwacht en wij voorspellen voor het eerst precies wat hiervan de invloed is op hun veiligheid.

Helaas zijn deze afwijkingen inherent aanwezig in alle roosters die momenteel gebruikt worden. Speciale roosters, die minder grote afwijkingen hebben, maar nog wel decodeerbaar zijn, bestaan wel en zouden in theorie veiliger zijn. Het probleem is dat deze speciale roosters voor iedereen bekend zijn, dus ook hun korte omschrijving, waardoor ze momenteel niet bruikbaar zijn. In Deel IV laten we zien dat de korte omschrijving alsnog verstopt kan worden, simpelweg door het rooster te roteren, waarbij nu de rotatie de sleutel vormt. In Hoofdstuk 9 onderbouwen we waarom het vinden van de rotatie lastig is. In Hoofdstuk 10 laten we zien hoe dit in theorie gebruikt kan worden voor het versleutelen of het digitaal ondertekenen van berichten en we geven een praktische demonstratie in de vorm van een zeer efficiënte digitale handtekening genaamd HAWK.

In het losstaande Hoofdstuk 8 laten we zien dat bekende begrippen en algoritmes voor roosters, ook vertaalbaar zijn naar lineaire codes, een soort eindige analogie van roosters. We dragen onder andere het beroemde LLL-reductiealgoritme over naar lineaire codes.



Summary

This thesis is primarily about lattices. Not in dimension two or three which we can still visualize, but in hundreds, sometimes thousands of dimensions. These high dimensions bring with them a lot of computational problems. Questions that are easy in two dimensions, can be very hard in higher dimensions. And these hard problems give rise to cryptography.

Cryptography aims to make communications secure in case the connection is overheard or influenced. Surprisingly enough, this can be done without the communicating parties agreeing in advance on a secret code language. This requires problems which are hard in general, yet can be solved if certain secret key information is known. If only the recipient is familiar with that secret key, then only that person can read the message, and no one else. Think of a safe with a padlock. Anyone can put a message in the safe and push the lock shut, but only the owner of the key can unlock the safe (effortlessly).

Lattice problems are ideally suited for such an encryption, with the added advantage that they do not appear to be efficiently solvable even by quantum computers. The idea is as follows. A lattice, despite its high dimension, can be described by a few arrows (see figure). A description with long arrows (red) hereby forms the safe with padlock and a description with short arrows (green) hereby forms the key. Finding such a short description, given a long description, is called reduction and is hard computational problem. If we think of a lattice point as a letter, it can be encrypted by moving the lattice point slightly outside the lattice. Recovering the original location, and thus

decrypting the message, is called decoding. Decoding is generally hard except with a short description, the key.

Part II and III deal with cryptanalysis asking the central question: exactly how hard is it to read the encrypted message? We can find out the key (reduce), or break open the lock by force (decrypt). In Chapter 4, we show how, using modern algorithms and graphic cards, we are able to reduce a lattice in dimension 180, 25 dimensions more than the old record. In Chapter 5, we analyse an algorithm, which after some preliminary work, can decode many points in the same lattice. In Chapter 6 we discuss that lattices used in cryptography are often somewhat oddly shaped, for example by containing lattice points that are too close together. These deviations make reduction or decoding often considerably easier than you might expect, given the high dimension. We deal in Chapter 7 with so-called NTRU lattices, which have been used in cryptography since 1996. These lattices have in some cases an even larger deviation than expected, and we predict for the first time exactly how this affects their security.

Unfortunately, these deviations are inherently present in all lattices currently in use. Remarkable lattices, which have smaller deviations but are still decodable, actually do exist and would theoretically be safer. The problem is that these remarkable lattices are known to everyone, including their short description, making them currently unusable. In Part IV, we show that the short description can still be hidden, simply by rotating the lattice, where now the rotation is the key. In Chapter 9, we substantiate why finding the rotation is hard. In Chapter 10, we show how this can theoretically be used to encrypt or digitally sign messages, and give a practical demonstration in the form of a highly efficient digital signature called HAWK.

In the stand-alone Chapter 8, we show that well-known notions and algorithms for lattices, are also translatable into linear codes, a kind of finite analogue to lattices. Among others, we transfer the famous LLL reduction algorithm to linear codes.

Acknowledgments

I would like to thank my advisors, Prof.dr. Léo Ducas and Prof.dr. Ronald Cramer for their supervision.

Léo, my promotor and main advisor, has always shown an unwavering support and enthusiasm for my research and has happily shared his knowledge and insights with me. He always has a unique, often geometric, and intuitive view on lattices and related topics, and I am grateful that he has taught me these ways. After my Bachelor, Master and now PhD thesis under his advice, I can only conclude that we made a great team.

Ronald, my second promotor and group leader, is gratefully acknowledged for his wisdom and great stories. I am grateful for the opportunities he gave me, both directly, but also indirectly, by bringing together such a great group of people.

I am also very grateful to the members of the Doctorate Committee for reading my thesis and for providing feedback.

Additionally, I would like to thank my colleagues from the Cryptology Group at CWI, and other colleagues at CWI, Leiden University, and abroad, for providing me with a friendly, motivating and always curious environment.

To my family and friends, thank you all for your unwavering love, interest and support, for all the light and deep conversations and lessons that shaped me, for allowing me to unwind my mind, and for making me feel at home regardless of the location.

Special thanks go to the many friends and colleagues that helped with proofreading parts of this thesis, and my sisters Marjolein and Sanne for their help with the cover design.

Curriculum Vitae

Wessel van Woerden was born in Koudekerk aan den Rijn, Rijnwoude, the Netherlands, on March 8, 1995. He also grew up there, and obtained his high school diploma from Groene Hart Lyceum in Alphen aan den Rijn in 2013. That year he also graduated from the Pre-University College of Universiteit Leiden. After this he continued his studies at Universiteit Leiden. In 2016, he obtained his bachelor degrees *summa cum laude* in both Mathematics and Computer Science. In 2018 he obtained his master's degree *summa cum laude* in Mathematics. His interest in lattices started with his bachelor thesis "The closest vector problem in cyclotomic lattices" and continued with his master thesis "Perfect quadratic forms: an upper bound and challenges in enumeration", both written under the supervision of Prof.dr. Léo Ducas.

In 2018, Wessel obtained a PhD position at the Universiteit Leiden under supervision of Prof.dr. Léo Ducas and Prof.dr. Ronald Cramer, to do research in the Cryptology Group at Centrum Wiskunde & Informatica (CWI) in Amsterdam. Here he combined his interest in lattices and algorithms with the topic of cryptology. His main focus was on cryptanalysis of lattice-based schemes, ranging from asymptotic to concrete hardness estimates in theory, and record computations in practice. These cryptanalytic results, and a remaining interest in lattice packings and isomorphisms from his master thesis, also motivated constructive work, eventually leading in a joint effort to the signature scheme HAWK.

In 2022, he started as a post-doc in the Number Theory group at Institut de Mathématiques de Bordeaux.

List of Publications

This thesis is based on the following published papers.

[DLW20] **The randomized slicer for CVPP: sharper, faster, smaller, batchier**
Léo Ducas, Thijs Laarhoven, and Wessel van Woerden,
PKC, 23rd Annual International Conference, 2020.

[SHVW20] **A canonical form for positive definite matrices**
Mathieu Dutour Sikirić, Anna Haensch, John Voight, and Wessel van Woerden,
ANTS XIV, 2020.

[DSW21] **Advanced Lattice Sieving on GPUs, with Tensor Cores**
Léo Ducas, Marc Stevens, and Wessel van Woerden,
Eurocrypt, 40th Annual International Conference, 2021.

[DW21] **NTRU Fatigue: How Stretched is Overstretched?**
Léo Ducas and Wessel van Woerden,
Asiacrypt, 27th Annual International Conference, 2021.

[DW22] **On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography**
Léo Ducas and Wessel van Woerden,
Eurocrypt, 41st Annual International Conference, 2022.

[DDW22] **An Algorithmic Reduction Theory for Binary Codes: LLL and more**
Thomas Debris-Alazard, Leo Ducas, and Wessel van Woerden,
IEEE Transactions on Information Theory, 2022.

The following published paper is briefly summarised in this thesis.

[DPPvW22] **Hawk: Module LIP makes Lattice Signatures Fast, Compact and Simple**
Léo Ducas, Eamonn W. Postlethwaite, Ludo N. Pulles, and Wessel van Woerden,
Asiacrypt, 28th Annual International Conference, 2022.

The author has additionally published the following papers, which are not included in the thesis.

[DW18] **The closest vector problem in tensored root lattices of type A and in their duals**
Léo Ducas and Wessel van Woerden,
Designs, Codes and Cryptography, 2018.

[Woe20] **An upper bound on the number of perfect quadratic forms**
Wessel van Woerden,
Advances in Mathematics, 2020.