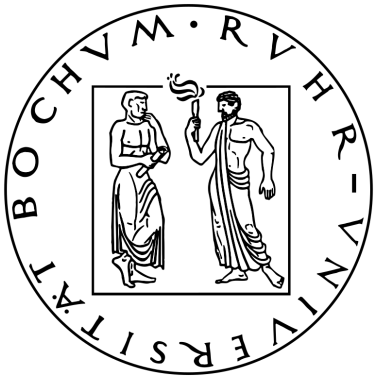


Complexity of the Learning with Errors Problem and Memory-Efficient Lattice Sieving

Dissertation

zur Erlangung des Doktorgrades
der Naturwissenschaften
an der Fakultät für Mathematik
der Ruhr-Universität Bochum

vorgelegt von
Elena Kirshanova

unter der Betreuung von
Prof. Dr. Alexander May

Bochum
November 2016

First reviewer: Prof. Dr. Alexander May

Second reviewer: Prof. Dr. Gregor Leander

Date of the oral examination: 2.12.2016

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Ich erkläre weiterhin, dass ich alles gedanklich, inhaltlich oder wörtlich von anderen (z.B. aus Büchern, Zeitschriften, Zeitungen, Lexika, Internet usw.) Übernommene als solches kenntlich gemacht, d.h. die jeweilige Herkunft im Text oder in den Anmerkungen belegt habe. Dies gilt gegebenenfalls auch für Tabellen, Skizzen, Zeichnungen, bildliche Darstellungen usw.

Ort, Datum	Unterschrift

Acknowledgments

First and foremost, my deepest thanks go to my advisor Alexander May. I feel privileged to have his guidances and support over the last three years. I would like to thank him for the immense amount of time he spent explaining me what cryptography is actually about and for his endless patience when I was slow on the uptake. I was extremely fortunate to have such an amazing advisor.

Much of this Thesis is joint work with Gottfried Herold, to whom I am much indebted for his vital contribution to my understanding of math and crypto. Not only is he a person able to elegantly solve math problems when I got stuck, but also a good friend. I am grateful to my other co-author, Friedrich Wiemer, for settling out my countless questions about programming.

I thank the whole crypto group at RUB for the invaluable support and encouragement especially during the last and toughest months of my PhD. Being a part of such a group is a privilege. A special thanks go to my office mate, Felix Heuer, not only for proof-reading my entire Thesis within one day, but also for making our office NA 5/75 a place full of joy and fun, not a place full of PCB. Marion Reinhardt-Kalender, whose help and assistance made my stay in Germany most comfortable and untroubled, deserves a special *Vielen Dank*.

During my last year, I had a great honour to collaborate with the crypto group in ENS Lyon lead by Damien Stehlé. I have learned a great deal during my visit there and I am looking forward to working together.

I would not have even considered doing research in crypto, if I had not been introduced to the subject during my studies at I.Kant Baltic Federal University in Kaliningrad. I would like to thank S. Aleschnikov, A. Zaytzev, and all the others members of the Faculty of Mathematics at BFU for their inspiring lectures.

Most importantly, I would like to thank my family and especially my mother, for her constant support and her firm belief in me.

Contents

Notations

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$	.....	sets of natural, integer, rational, real numbers
$\mathbb{Z}_q = \begin{cases} [-q/2 \dots q/2), \ q \text{ even} \\ [-\frac{q-1}{2} \dots \frac{q-1}{2}], \ q \text{ odd} \end{cases}$	.....	ring of integers mod q
$\mathbb{Z}^n, \mathbb{Z}_q^n, \mathbb{Q}^n, \mathbb{R}^n$	.....	vector-spaces of dim. n
$\vec{x}$	.....	column vector
$\vec{x}^{\mathsf{t}}$	.....	row vector
$\|\vec{x}\|$	.....	Euclidean length of vector $\vec{x}$
$\|\vec{x}\|_\infty$	.....	ℓ_∞-norm of $\vec{x}$: $\max_i \|x_i\|$
$\vec{1}, \vec{0}$	.....	all-ones, all-zeros vectors
A, B, etc.	.....	matrices (composed from vectors column-wise)
$A_{i,j}$	.....	(i^{th} row, j^{th} column) element of matrix A
$A[1 \dots i]$	.....	the sub-matrix of A formed by first i rows, i columns
$[\vec{x}]_\ell^u$	.....	vector $\vec{x}$ projected on coordinates $[\ell, \dots, u]$
$\mathrm{Im}(A)$	.....	the image of matrix A
$\ker(A)$	.....	the kernel of matrix A
$\mathbb{1}_n$	.....	$n \times n$ identity matrix
$wt(\vec{x})$	.....	Hamming weight of vector $\vec{x} \in \{0,1\}^n$
$\mathcal{B}(\vec{0}, r)$	.....	ball of radius r centred at $\vec{0}$
S^n	.....	n-sphere of radius 1, i.e. $\mathsf{S}^n = \{\vec{x} \in \mathbb{R}^{n+1} \mid \|\vec{x}\| = 1\}$
$\log x$	.....	$\log_2 x$

We use the Landau notations $\mathcal{O}(\cdot), \widetilde{\mathcal{O}}(\cdot), \Theta(\cdot), \Omega(\cdot), \omega(\cdot), o(\cdot)$. We write $\widetilde{\mathcal{O}}_k(\cdot)$ when we want to stress that the asymptotic result holds for fixed k.

Chapter 1

Introduction

Today, private communication is protected by cryptographic systems that rely on the hardness of specific number-theoretic problems. The link between the two disciplines, cryptography and number theory, first established in the revolutionizing work of Diffie and Hellman [DH76] and later reinforced by the work of Rivest, Shamir, and Adleman on RSA [RSA78], has led to extremely efficient ways to exchange information privately. The effect of these discoveries on our everyday life is enormous as we can hardly imagine ourselves being unable to buy goods, make reservations or handle other types of financial operations over the Internet.

The elegance of number-theoretic constructions has been shadowed by arguably the most famous quantum algorithm – the period-finding algorithm by Shor [Sho97], first appeared in 1996. Provided a large-scale quantum computer is built, all the cryptographically relevant number-theoretic problems, like integer factorization or the discrete logarithm problem, can be efficiently solved by Shor's algorithm. The existence of such a quantum computer may look unrealistic today as there are several serious obstacles on the way to build a quantum device that could be of any threat to our modern cryptosystems; yet many concerns have been raised on the security of the deployed systems. These worries are also backed up by the progress in classical methods for factoring large numbers and solving the discrete logarithm problem in multiplicative groups of finite fields. For example, the general number field sieve algorithm, the most efficient algorithm known for factoring large integers, allows to factor N in time $L_N(1/3, 1.902)$– a function[1] truly sub-exponential in the bit-length of N.

Hard problems on a lattice in $\mathbb{R}^n$ offer an attractive alternative to the aforementioned number-theoretic problems and serve as a foundation to what is now known as *lattice-based cryptography*.

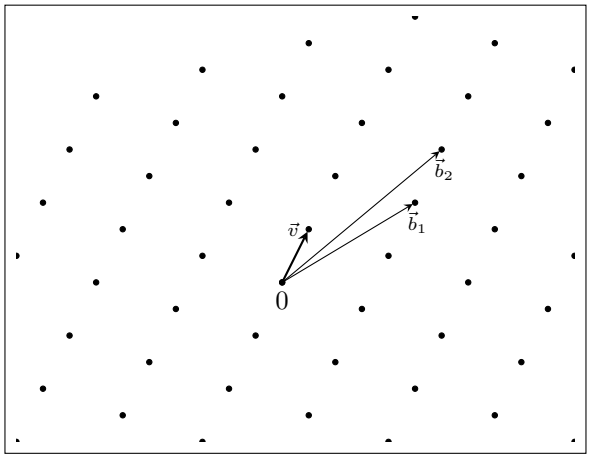


Fig. 1.1: 2-dimensional lattice with a basis $\{\vec{b}_1, \vec{b}_2\}$. $\vec{v}$ is one of the shortest vectors of this lattice.

[1] where L_N is defined as $L_N(\alpha, c) = \exp(c + o(1)(\log N)^\alpha (\log\log N)^{1-\alpha}$

A lattice is a set of points in $\mathbb{R}^n$ where each point is an integer linear combination of n-linearly independent vectors $\vec{b}_1, \ldots, \vec{b}_n \in \mathbb{R}^n$ known as a lattice-basis. An example of a 2-dimensional lattice with a basis is shown in Figure 1.1.

Lattices have attracted the attention of mathematicians since the late 18^{th} century. Early works of Gauss and Lagrange aimed at finding short lattice bases in $\mathbb{R}^2$ have evolved into a whole range of algorithms known as lattice-basis reduction algorithms. The publication of *Geometrie der Zahlen* by Hermann Minkowski at the beginning of the previous century marks the birth of the Geometry of Numbers – a branch of number theory that studies convex bodies and lattice points contained in these bodies.

In more recent time, lattices have become an active research topic in various computer science areas like integer programming [Len83], complexity theory [GG98, Kho03], and many others. Cryptography finds itself among this list as well. Interestingly, lattices stepped into cryptography as a destructive tool: Coppersmith's method to find small roots of low-degree polynomials [Cop01], an algorithm of Lagarias-Odlyzko for the low-density subset-sum problem [LO85], rounding techniques for the hidden number problem [BV97] – all these methods form an incomplete list of cryptanalytic tools based on lattices.

It was in 1996 when Ajtai shows in his celebrated paper [Ajt96] how to *construct* cryptographic primitives from lattices. Ajtai's breakthrough is acknowledged as the starting point of the lattice-based cryptography that has been flourishing over the last 20 years.

Another significant milestone in the lattice-based cryptography era is the work of Regev *On Lattices, Learning with Errors, Random Linear Codes, and Cryptography* [Reg05]. While Ajtai's discovery allows to construct primitives that emerge from one-way functions such as collision-resistant hash functions and signatures, the question of building a public key cryptosystem remained open. Regev was the first to present a public key cryptosystem and relate its hardness to the problem of finding short vectors in a lattice known as the Shortest Vector Problem (SVP) (see Figure 1.1).

The conjectured security against quantum attacks is one among several other attractive features of lattice-based constructions. Not only does Ajtai show how to build a primitive, he also proves that this primitive is hard to break unless *all* instances of a certain lattice-problem are easy to solve. This connection is known as the *worst-case* hardness. What it means is that a random instance of a problem (which, in a cryptographic setting, translates into a randomly chosen key-pair of a user) is as hard as the worst-case instance of this problem. Such a remarkable feature is not shared by number-theoretic problems like RSA: for instance, some large numbers are easier to factor than others.

More importantly from a practical side, lattice-based cryptosystems are very efficient and highly parallelizable. Typical computations involve linear operations on matrices modulo a small integer. Primitives based on special *ideal* lattices reduce the communication overhead thus making the constructions truly competitive with their number-theoretic counterparts.

Despite all these nice features, lattice-based cryptography is not yet widely deployed[2]. Leaving aside costs of setting up a new algorithm into a communication channel, we still have theoretical questions to be answered before we could let lattice-based cryptography drive the real-world private communications.

These questions are primarily of cryptanalytic nature. How hard are the lattice-problems underlying a cryptosystem? In particular, what are the best algorithms for the Shortest Vector problem? In this thesis, we address these questions.

The dissertation consists of two parts. The first part analyzes the complexity of the problem that underlies all known lattice-based cryptosystems. The second part is devoted to the Shortest Vector Problem. We now detail on each part.

[2]In July 2016, Google has announced [Blo16] that they incorporated a lattice-based public key-exchange scheme *The New Hope* [ADPS15] for experimental purposes.

Part I. Learning with Errors as Bounded Distance Decoding

The cryptosystem of Regev presented in [Reg05] does not *directly* rely on a lattice-problem. It relies instead on the so-called *Learning with Errors* problem. In [Reg05], Learning with Errors (LWE), an average-case hard problem, is proved to be at least as hard as a certain problem on lattices. This result enables us to relate the security of a cryptosystem to a hard lattice-based problem *via* LWE.

Regev shows that LWE is at least as hard as certain *approximation* problems on a lattice: instead of asking for a shortest vector, we require to output a vector no longer than a predefined bound.

Due to the fact that the approximation factors for these problems are polynomial in the lattice dimension, denoted as n, known NP-hardness results for lattice problems [Kho03, Mic98, HR07] do not apply to LWE: the best approximation factor for which SVP is known to be NP-hard is sub-polynomial $2^{(\log n)^{1-\varepsilon}}$.

The LWE problem can be stated as a problem of decoding random linear codes in $\mathbb{Z}_q^n$ for some modulus q. The error-vector in a code that arises from LWE is of a special form: its entries are chosen from a discrete Gaussian distribution with 0-mean and a known standard deviation. This standard deviation guarantees a unique solution for the decoding, which allows us to attack LWE using algorithms for *bounded* distance decoding.

Note that this decoding problem is parametrized by dimension n, modulus q, and the standard deviation of the error-distribution. So it is reasonable to expect that all these parameters affect the hardness of LWE. We call the triple (n, q, standard deviation) an LWE parameter-set.

The first part of the thesis is devoted to the analysis of the decoding problem that arises from LWE. We separate asymptotical and practical studies into two sections. In the asymptotical part, we contribute the following results:

- In Sect. 3.1, we study the asymptotical behavior of all known *lattice-based* decoding algorithms when applied to LWE. We give precise constants in the leading-order exponents as functions of LWE parameters. The algorithms are considered under various settings: polynomial/exponential memory complexity and limited/unlimited access to the LWE samples (i.e., phrased in the language of codes, in the LWE problem, one can control the length of a codeword).

- To unify the analysis, we identify features that a decoding algorithm should have in order to be "reasonable" (a precise definition of *reasonable* is given) and describe a decoding algorithm, which we call *Generalized Pruning* that shares these features. Asymptotical analysis of this Generalized Pruning algorithm enables us to conclude on the asymtotics of other decoding algorithms.

- Interestingly, our analysis shows that all the decoding algorithms achieve *the same* constant in the leading-order exponent – a conclusion that was not drawn by previous results.

The reader interested only in the results of our asymptotical analysis and not in the proofs, should be referred to Sect. 3.1.6 and, in particular, to Fig. 3.3 where we compare *all* known algorithms for LWE (not only lattice-based decoding). Using the figure, one can easily deduce which algorithm is the best for a given LWE parameter-set.

The results of this section are presented in the joint work with G. Herold and A. May [HKM].

Practical hardness of Learning with Errors is the topic of Sect. 3.2. Our goal is to determine which LWE parameters are feasible to solve on a modern computer. We consider the two-phase lattice-based decoding algorithm – the most relevant algorithm for LWE in practice.

The results are based on our parallelized implementation of the *second* phase (also referred to as *enumeration* phase hereafter) of the decoding algorithm, which is a tree-traversal algorithm. Our results are the following:

- the enumeration step of the lattice-based LWE decoding can be almost perfectly parallelized which allows for a significant speed-up of the decoding attack in practice.
- We run our parallelized algorithm on various LWE parameter-sets and present the results in Table 3.5. This is the first time the concrete running times of lattice-based attacks for non-toy LWE parameter-sets are presented.
- We show how certain deviations from standard LWE parameters (like binary instead of Gaussian error) make practical attacks significantly faster (see Sect. 3.2.3).

These results stem from the joint work with A. May and F. Wiemer published in [KMW16].

Part II. k-List algorithms for SVP

The Shortest Vector Problem is the main computational problem associated with a lattice. There are four main families of algorithms for this problem. We summarize them in Table 1.1.

Algorithm	**Running time**	**Memory complexity**
Deterministic algorithms:		
Enumeration [Kan87, HS07]	$n^{(1/2e)n+o(n)}$	poly(n)
Voronoi Cell [MV10]	$2^{2n+o(n)}$	$2^{n+o(n)}$
Probabilistic algorithms:		
Gaussian Sampling [ADRS15]	$2^{n+o(n)}$	$2^{n+o(n)}$
Sieving [AKS01]		
– Provable [PS09]	$2^{2.465n+o(n)}$	$2^{1.325n+o(n)}$
– Heuristic:		
– 2-sieve [BDGL16]	$2^{0.292n+o(n)}$	$2^{0.208n+o(n)}$
– 3-sieve [BLS16]	$2^{0.4812n+o(n)}$	$2^{0.1887n+o(n)}$

Tab. 1.1: Algorithms for SVP on an n-dimensional lattice

All single-exponential algorithms share one major drawback: exponential memory-requirement. This fact precludes single-exponential SVP algorithms from being practical: currently, the best performance is shown by memory-friendly enumeration algorithms with super-exponential complexities.

The main result of the second part of the thesis is a faster heuristic sieving algorithm for SVP with $2^{0.1877n+o(n)}$ memory and improved running time $2^{0.396n+o(n)}$. These results were obtained by phrasing the shortest vector problem as a special case of an approximate k-List problem in Euclidean norm and applying an algorithm for the latter problem to SVP. The resulting algorithm is called the k-sieve. The view on sieving algorithms for SVP via k-Lists allows us to

- prove the conjectured in [BLS16] memory complexity of the algorithm,
- improve the running time of the triple-sieve (and, more generally, k-tuple sieve) algorithm from [BLS16],
- obtain closed formulas for running time of k-sieve algorithms for any fixed $k \geq 3$.

In Sect. 4.1.5, we also show the results of practical experiments with our algorithm that confirm the speed-up.

This part of the thesis is joint work with G. Herold and is currently in submission.

We proceed in this chapter with investigating the hardness of the approximate SVP, denoted appSVP_γ, on a so-called q-ary lattice. The approximation factor γ we are interested in is polynomial in the lattice dimension. This type of lattice emerges from the LWE problem, and the question of finding a poly(n)-approximate solution to the shortest vector problem is at the heart of algorithms considered in the first part of the thesis. We present two combinatorial algorithms for appSVP_γ and analyze their complexity in Sect. 4.2. The first algorithm is a reformulation of a known combinatorial algorithm for LWE, the second is its improved version.

During her PhD, the author of this thesis has also published a paper on the construction of lattice-based proxy re-encryption [Kir14], which is not described here.

Chapter 2

Preliminaries

2.1 Lattices

In this chapter, we present basic definitions and algorithms associated to lattices. We give only the necessary definitions and facts that concern lattices and the Learning with Errors problem. There is a rich variety of surveys on lattices in cryptography and on LWE in particular. To name a few, lecture notes of Regev [Reg09] and a recent survey by Peikert [Pei16] offer a comprehensive overview on lattice-based cryptography.

A (full-rank) lattice $\mathcal{L} \subset \mathbb{R}^n$ is the set of all integer linear combinations of n linearly independent vectors $B = (\vec{b}_1, \ldots, \vec{b}_n)$. These vectors form a *basis* of the lattice. A basis is not unique: for any unimodular $U \in \mathbb{Z}^{n \times n}$, BU is another basis. We write $\mathcal{L}(B)$ when we want to stress that the lattice is represented by a basis B. The *fundamental region* of $\mathcal{L}(B)$ is $\mathcal{P}_{1/2}(B) = \{\sum_{i=1}^{n} c_i \vec{b}_i \colon c_i \in [-\frac{1}{2}, \frac{1}{2})\}$. Its volume, known as the volume of $\mathcal{L}(B)$, equals to $|\det(B)|$ and is independent of the choice of basis. We denote this value $\det \mathcal{L}$. We let $\mathrm{Span}(\mathcal{L}(B))$ to be the set of *all* (not only integer) linear combinations of $(\vec{b}_1, \ldots, \vec{b}_n)$.

The *Gram-Schmidt orthogonalization* (GSO) $B^* = (\vec{b}_1^*, \ldots, \vec{b}_k^*)$ is obtained iteratively by setting $\vec{b}_1^* = \vec{b}_1$, and $\vec{b}_i^*$ as the orthogonal projection of $\vec{b}_i$ on $(\vec{b}_1, \ldots, \vec{b}_{i-1})^{\perp}$ for $i = 2, \ldots, k$. This orthogonalization process can be described via matrix-decomposition $B = B^* \mu^{\mathsf{t}}$, where μ is a lower-triangular matrix with $\mu_{i,j} = \langle \vec{b}_i, \vec{b}_j^* \rangle / \|\vec{b}_j^*\|^2$ for $i \geq j$.

The *minimum distance* (or the *first successive minimum*) of lattice $\mathcal{L}$ is the length of its shortest non-zero vector: $\lambda_1(\mathcal{L}) = \min_{\vec{v} \in \mathcal{L} \setminus \{\vec{0}\}} \vec{v}$. Minkowski's inequality states that $\lambda_1 \leq \sqrt{n} \cdot (\det \mathcal{L})^{1/n}$. It is tight up to a constant and we usually treat it as equality to approximate the length of the shortest vector. The i^{th} *successive minima* $\lambda_i(\mathcal{L})$ is the smallest r s.t. $\mathcal{B}(\vec{0}, r)$ contains $\geq i$ linearly independent vectors of $\mathcal{L}$. λ_i's are independent of the choice of basis.

In Chap. 3 and Sect. 4.2, we deal with the so-called q-ary lattices:

$$\mathcal{L}_q(B) = \Big\{ \vec{y} \in \mathbb{Z}^n \colon \vec{y} = \sum_{i=1}^{n} z_i \cdot \vec{b}_i = B\vec{z} \mod q \colon z_i \in \mathbb{Z} \Big\}.$$

Such a lattice forms an image of B, $\mathrm{Im}(B)$. Later, we shall be dealing with a q-ary lattice $\mathcal{L}_q \subset \mathbb{Z}^m$ formed by $B \in \mathbb{Z}_q^{m \times n}$ for $m > n$. This is a lattice of rank m: the first n basis-vectors are columns of B and the remaining $m - n$ vectors are the q-vectors of the form $(0, \ldots, q, \ldots, 0)$.

The kernel of B forms another (the so-called scaled *dual* to $\mathcal{L}_q(B)$) q-ary lattice:

$$\mathcal{L}_q^{\perp}(B) = \{\vec{x} \in \mathbb{Z}^n \colon B\vec{x} = 0 \mod q\}.$$

In general, the *dual* of $\mathcal{L}$ is defined as $\mathcal{L}^* = \{\vec{y} \in \mathrm{Span}(\mathcal{L}) \colon \langle \vec{x}, \vec{y} \rangle \in \mathbb{Z} \quad \forall \vec{x} \in \mathcal{L}\}$.

Hard problems on lattices. There are several fundamental problems related to lattices. The most cryptographically relevant are the following five.

The *Closest Vector Problem* (CVP) asks to find a lattice point $\vec{v} \in \mathcal{L}$ closest to a given (target) point $\vec{t} \in \mathbb{R}^n$. We write $(\mathcal{L}(B), \vec{t})$ for a CVP instance on the lattice $\mathcal{L}(B)$.

In the promise variant of CVP, the *Bounded Distance Decoding* (BDD) problem, we know in addition that $\|\vec{t} - \vec{v}\| < R$ where $R \ll \lambda_1(\mathcal{L})$. In this case, the solution $\vec{v}$ is unique.

In the *Shortest Vector Problem* (SVP), we are asked to find $\vec{v} \in \mathcal{L}$ s.t. $\|\vec{v}\| = \lambda_1(\mathcal{L})$.

We can relax the above and ask for a vector $\vec{v}$ s.t. $\|\vec{v}\| \leq \gamma\lambda_1(\mathcal{L})$. This problem is called the approximate Shortest Vector Problem (appSVP_γ). In general, the approximation factor γ can be a function of n. In Chap. 4 we present an algorithm that solves appSVP_γ for constant γ.

A promise variant of SVP is a so-called *unique* SVP problem: we are promised that the first successive minimum λ_1 is γ times shorter than the second minimum λ_2. The quantity $\frac{\lambda_2}{\lambda_1} = \gamma$ is know as the lattice *gap*. We write uSVP_γ for short. For both appSVP_γ and uSVP_γ, the larger γ is, the easier the problem. We refer to [LM09] for reductions between the BDD, uSVP_γ, and appSVP_γ problems.

Lattice basis reduction is an algorithm that on a lattice basis as input returns another basis for this lattice that consists of shorter and more mutually orthogonal vectors (mutual orthogonality translates into the slow decay of the length of the Gram-Schmidt vectors). There are several notions of reducedness of a basis ranging from fast but weak (in terms of quality of the output) LLL reduction due to A. Lenstra, H. Lenstra, and L. Lovász [LLL82] to strong but computationally inefficient Hermite-Korkine-Zolotarev reduction. The basis reduction we are mostly interested in is called the BKZ reduction (short for Block-Korkine-Zolotarev, [Sch87]). Together with a lattice-basis, it receives as input parameter β that determines the length of the output basis-vectors. The larger β is, the shorter the output basis-vectors will be. More formally, BKZ run on an n-dimensional lattice $\mathcal{L}$, produces a basis with the first (i.e. the shortest) vector satisfying

$$\|\vec{b}_1\| \leq 2\beta^{\frac{n}{2\beta}} \cdot (\det \mathcal{L})^{\frac{1}{n}}. \tag{2.1}$$

BKZ works by calling an SVP-solver on a sub-lattice of dimension β. In [HPS11] it was shown that after $\mathrm{poly}(n)$ number of SVP-calls, the guarantee defined in Eq. (2.1) is achieved. Hence, if the running time of an SVP algorithm for dimension β is $T_{\mathsf{SVP}}(\beta)$, the running time of BKZ is $T_{\mathtt{BKZ}}(\beta) = \mathrm{poly}(n) \cdot T_{\mathsf{SVP}}(\beta)$. Currently, the best algorithms for SVP are at least exponential in the dimension: the algorithm due to [ADRS15] *provably* solves SVP in $2^{n+o(n)}$ time, while *heuristically* we have a slightly better constant in the exponent due to [BDGL16], namely $2^{0.292n+o(n)}$. All these single-exponential algorithms require $2^{\mathcal{O}(n)}$ memory. In the memory-efficient SVP-solver of Kannan [Kan83], the running time increases to $2^{\mathcal{O}(n \log n)}$ with only $\mathrm{poly}(n)$ space complexity.

As already mentioned above, a weaker form of lattice basis reduction that runs in polynomial time, is realized by the LLL algorithm, where we have the guarantee that the shortest returned vector satisfies $\|\vec{b}_1\| \leq \gamma^{\frac{n-1}{2}} (\det \mathcal{L})^{1/n}$ for $\gamma > 4/3$.

Geometric Series Assumption (GSA) proposed by Schnorr in [Sch03], gives an estimate on the relative length of the Gram-Schmidt vectors $\vec{b}^*$ of a basis output by β-BKZ. The assumption says that the sequence $\|\vec{b}_i^*\|$ decays geometrically in i, namely $\frac{\|\vec{b}_i^*\|}{\|\vec{b}_{i+1}^*\|} \approx \beta^{1/\beta}$. It is equivalent to

$$\|\vec{b}_i^*\| \approx \|\vec{b}_1\| \cdot \beta^{-\frac{i}{\beta}}. \tag{2.2}$$

We treat the above Eq. (2.2) as equality. From the fact that the product of all Gram-Schmidt vectors is equal to the lattice determinant, combining Eq. (2.1) and Eq. (2.2) yields

$$\|\vec{b}_1\| = (\beta)^{\frac{n}{2\beta}} \cdot (\det \mathcal{L})^{\frac{1}{n}}. \tag{2.3}$$

Discrete Gaussian distribution. For a vector $\vec{v}$ and any $s > 0$, define $\varrho_s(\vec{v}) = \exp(-\pi\|\vec{v}\|^2/s^2)$ as a Gaussian function with *parameter* (or *width*) s. To turn this function into a probability density function over a (countable) set $\mathcal{A} \subset \mathbb{R}^n$, define the normalization factor as $\varrho_s(\mathcal{A}) = \sum_{\vec{v}\in\mathcal{A}} \varrho_s(\vec{v})$. When $\mathcal{A}$ is taken as a lattice $\mathcal{L}$, the *discrete Gaussian probability distribution* with parameter s over $\mathcal{L}$ is defined with the probability density function

$$D_{\mathcal{L},s}(\vec{v}) = \frac{\varrho_s(\vec{v})}{\varrho_s(\mathcal{L})} = \frac{\exp(-\pi\|\vec{v}\|^2/s^2)}{\sum_{\vec{v}\in\mathcal{L}} \varrho_s(\vec{v})}. \tag{2.4}$$

The parameter s is the scaled standard deviation: for $s \to \infty$, the standard deviation is $s/\sqrt{2\pi} + o(s)$. A way to sample a discrete Gaussian for a given lattice can be found in [GPV08]. We will be mainly concerned with the discrete Gaussian defined over the lattice $\mathbb{Z}_q^n$.

It is known that for integer lattices $\mathcal{L}$ (i.e. $\mathcal{L} \subset \mathbb{Z}^n$), a sufficiently wide discrete Gaussian distribution 'blurs' the discrete structure of $\mathcal{L}$, such that the distribution becomes very close to a continuous Gaussian [LP11], [MP12]. Hence, for large enough s, we can approximate a discrete Gaussian by a continuous one. We make use of the tail-bounds for the Gaussian distribution. For fixed $s > 0$ and $y \to \infty$:

$$1 - \frac{1}{s}\int_{-y}^{y} \exp\Big(-\frac{\pi x^2}{s^2}\Big)\mathrm{d}x = e^{-\Theta\left(\frac{y^2}{s^2}\right)} \qquad 1 - \frac{\sum_{x=-y}^{y} \exp(-\frac{\pi x^2}{s^2})}{\sum_{x=-\infty}^{\infty} \exp(-\frac{\pi x^2}{s^2})} = e^{-\Theta\left(\frac{y^2}{s^2}\right)} \tag{2.5}$$

2.2 Learning with Errors

The Learning with Errors problem (LWE) was introduced by Regev in [Reg05]. The LWE problem is parametrized by an integer n, modulus $q = q(n)$ (not necessarily prime) and an error distribution $\chi_\alpha : \mathbb{Z}_q \to \mathbb{R}^+$ with $\alpha < 1$. α is known as the 'error-rate'. Usually χ_α is taken as a discrete Gaussian distribution over $\mathbb{Z}_q$ of width $s = \alpha q$.

Definition 1 (LWE distribution)**.** *For an integer $q = q(n)$, an error distribution χ_α, and a* secret *$\vec{s} \in \mathbb{Z}_q^n$, the LWE distribution $\mathcal{A}_{\vec{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is defined by (1) choosing $\vec{a} \in \mathbb{Z}_q^n$ uniformly at random, (2) sampling $e \leftarrow \chi_\alpha$, and outputting a pair $(\vec{a}, \langle\vec{a},\vec{s}\rangle + e \bmod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. We call this pair an LWE-sample.*

Note that $\vec{s}$ and χ are fixed for $\mathcal{A}_{\vec{s},\chi}$. We use m to denote the number of LWE-samples. There are two problems related to the LWE distribution:

Definition 2 (Search-LWE)**.** *An algorithm solves the* search-LWE *problem if given m independent LWE samples from $\mathcal{A}_{\vec{s},\chi}$ $(\vec{a}_i, \langle\vec{a},\vec{s}\rangle + e_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, for $1 \le i \le m$, it outputs $\vec{s}$ with high probability.*

Definition 3 (Decisional-LWE)**.** *An algorithm solves the* decisional-LWE *problem if given m independent samples from $\mathbb{Z}_q^n \times \mathbb{Z}_q$, it distinguishes with a non-negligible advantage whether these samples were chosen from $\mathcal{A}_{\vec{s},\chi}$ or from a uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*

There is a Search-to-Decision reduction [MM11, Reg05] running is poly(q) time that shows that the above problems are equivalent. The reduction remains efficient for an exponential composite q with poly(n)-bounded divisors [MP12].

The number of LWE-samples m is set large enough so that the secret $\vec{s}$ is uniquely defined with high probability. Further, instead of asking for $\vec{s}$, we can ask for the error-vector $\vec{e}$, as one uniquely determines the other.

In [Reg05], Regev shows that the LWE problem is at least as hard as certain worst-case lattice-problems on a lattice of dimension n. The reduction is quantum: during the proof, an LWE oracle is

used to create a *reversible* transformation that is applied to a quantum state. A de-quantized version of the reduction was first shown by Peikert in [Pei09] for q exponential in n and later improved by Brakerski et al. in [BLP+13] for any modulus. This result shows a reduction from appSVP_γ on $\sqrt{n}$-dimensional lattice to n-dimensional LWE. Removing this square-root loss in the classical reduction remains a major open problem in the complexity of LWE. Both classical and quantum reductions require $\alpha q > \sqrt{2}n$.

The main hardness parameter of LWE is the dimension n and the modulus q.[1] The noise-rate α does not affect asymptotical hardness of LWE as long as it is of order $1/\mathrm{poly}(n)$, i.e. $\alpha q = n^\gamma$ for some small constant $\gamma > 0$. But when $q = 2^{\mathrm{poly}(n)}$ and $\alpha = 2^{-n^a}$ for some $a \in (0,1)$, LWE can be solved in sub-exponential time $2^{\widetilde{\mathcal{O}}(n^{1-a})}$. In this work, we consider the most popular choices of q and α: $q = \mathrm{poly}(n)$, where the degree of the polynomial is a small constant, and $\alpha = o(1)$ (more specifically, in [Reg05], the parameters considered are $q = n^2, \alpha = 1/n$ or $\alpha = 1/(\sqrt{n}\log n)$). The number of samples, m does not affect the complexity of the problem and can be chosen arbitrarily large.

For our asymptotical analysis of LWE, we relate the parameters as

$$\boxed{q = n^{\mathsf{c}_q} \quad \text{and} \quad \alpha = \frac{1}{n^{\mathsf{c}_\alpha}},} \tag{2.6}$$

where $\mathsf{c}_q, \mathsf{c}_\alpha > 0$ are constants and $\mathsf{c}_q > \mathsf{c}_\alpha$.

LWE as a BDD instance. Having m LWE samples, we compose (column-wise) a matrix $A \in \mathbb{Z}_q^{n\times m}$ out of the first components $\vec{a}_i$ and a vector $\vec{t}$ out of the second components $\langle \vec{a}, \vec{s}\rangle + e_i$. We obtain an LWE instance in a matrix form:

$$(A, \vec{t}^{\,\mathsf{t}} = \vec{s}^{\,\mathsf{t}} A + \vec{e}^{\,\mathsf{t}} \mod q) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^m,$$

where $\vec{e} \to \chi_\alpha^m$. If χ_α is Gaussian with parameter αq, we have $\|\vec{e}\| = \Theta(\sqrt{m}\alpha q)$ with high probability.

It is easy to see that the search-LWE problem is an average-case BDD problem for the m-dimensional q-ary lattice $\mathcal{L}_q(A^\mathsf{t}) = \{A^\mathsf{t}\vec{x} \mod q \colon \vec{x} \in \mathbb{Z}_q^n\}$. A basis for this lattice over $\mathbb{Z}^m$ is given by the columns of the matrix:

$$B = \begin{pmatrix} \mathbb{1}_n & 0 \\ A' & q\mathbb{1}_{m-n} \end{pmatrix}, \tag{2.7}$$

where $\begin{pmatrix} \mathbb{1}_n \\ A' \end{pmatrix}$ is a column-reduced echelon form of A^t (see Chap. 2.3 in [Coh93]).

Assuming A is full-rank (which is the case with high probability), it easily follows that the determinant of $\mathcal{L}_q(A^\mathsf{t})$ is $\det(\mathcal{L}(B)) = q^{m-n}$. Further, from Minkowski's inequality, we approximate the length of the shortest vector in this lattice as $\lambda_1(\mathcal{L}(B)) \approx \sqrt{m}q^{1-n/m}$. Then the LWE problem is a BDD instance given by $(\mathcal{L}(B), \vec{t})$ with a promise that $\|\vec{t} - A^\mathsf{t}\vec{s} \mod q\| = \|\vec{e}\|$. For typical choices of α and q, the length of this error-vector, $\Theta(\sqrt{m}\alpha q)$, is much smaller than $\lambda_1(\mathcal{L}(B))$.

[1] Indeed, Brakerski et al. in [BLP+13] show that LWE preserves its hardness as soon as the quantity $n \log q$ remains the same. In other words, LWE with parameters (n, q, α) is equivalent to LWE with parameters $(1, n^q, \beta)$ where β is not significantly larger than α.

Chapter 3

Learning with Errors as BDD

In his seminal paper [Reg05], Regev shows that the Learning with Errors problem (see Def. 1) is at least as hard as certain hard lattice problems. It is thus important to understand how hard these lattices problems really are. This chapter addresses this question.

In more detail, we treat the Learning with Errors Problem as a Bounded Distance Decoding (BDD) Problem on the lattice $\mathcal{L}_q(A^\mathsf{t})$: given $(A, \vec{t} = A^\mathsf{t}\vec{s} + \vec{e} \bmod q) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^m$, we want to find the nearest to $\vec{t}$ lattice point $A^\mathsf{t}\vec{s} \in \mathcal{L}_q(A^\mathsf{t})$. In other words, we want to solve a decoding problem for a code generated by A^t with messages from $\mathbb{Z}_q^n$ and codewords from $\mathbb{Z}_q^m$. The solution $A^\mathsf{t}\vec{s}$ is unique since $\|\vec{e}\| \ll \lambda_1(\mathcal{L}_q(A^\mathsf{t}))$. The error-vector $\vec{e}$ is sampled from the discrete Gaussian distribution with parameter αq over the integer lattice $\mathbb{Z}_q^m$.

We start with the *asymptotical* hardness of lattice-based decoding attacks on LWE. In Sect. 3.1, we analyze the complexity of the problem under the following algorithms: Babai's `NearestPlane` Algorithm [Bab85], its extension due to Lindner and Peikert [LP11], and the Generalized Pruning Algorithm `GenPruning` – a unification of the decoding strategies which allows us to analyze the existing lattice-based algorithms for LWE/BDD. The so-called Pruning algorithms of [GNR10, LN13] appear as special cases of `GenPruning`.

The main LWE parameter that determines the complexity is the dimension of the secret – n. All the aforementioned decoding algorithms are either super-exponential: $2^{\mathsf{c}n\log n + o(n\log n)}$, or singe-exponential $2^{\mathsf{c}'n+o(n)}$, where c, c' are constants that depend on the other two LWE parameters: q, α. The goal of Sect. 3.1 is to determine these constants c, c'. Note that from the modulus-dimension trade-off by Brakerski et al. [BLP+13], stating that LWE preserves its hardness as long as the value $n \log q$ stays the same, we can already explain why for all algorithms, the leading order constants c, c' have a multiple of $\mathsf{c}_q = \log q / \log n$.

The results of our analysis are summarized in Table 3.1 in Sect. 3.1.6 where we list the running times of *all* known attacks on LWE together with constants in the exponents. These constants are made explicit as functions of the LWE parameters q, α. The table is translated into Fig. 3.3 to give a clear 'winner' among all the known attacks for a concrete choice of LWE parameters.

The second part is devoted to *practical* hardness of LWE. In Sect. 3.2, we present real running times of the Linear-Length Pruning attack on LWE – a pruning strategy that appears to perform best for the Learning with Errors problem.

The results of the first section are mainly based on [HKM]. The cryptanalysis of real LWE instances presented in Sect. 3.2 is published in [KMW16].

3.1 Asymptotical Hardness of LWE

For a decoding algorithm `ALG`, we will be interested in the quantity $\rho(\texttt{ALG}) = \frac{T(\texttt{ALG})}{P_{\text{succ}}(\texttt{ALG})}$, the trade-off between the running time and the success probability of `ALG` (in other words, we are interested in the

expected time to decode successfully). For an LWE instance $(A, \vec{t} = A^{\mathsf{t}}\vec{s} + \vec{e} \mod q)$, the decoding is successful if the returned lattice vector is indeed $A^{\mathsf{t}}\vec{s}$ or, equivalently, the returned error is $\vec{e}$. In the decoding algorithms we actually search for $\vec{e}$. It is easy to verify whether a given $\vec{e}$ is correct or not, as the correct one is much shorter than an error-vector that leads to a wrong solution.

The algorithm we analyze here is a two-phase BDD decoding algorithm: first, we preprocess a basis for $\mathcal{L}_q(A^{\mathsf{t}})$ (given in Eq. (2.7)) using β-BKZ reduction to obtain a shorter basis. Denote this basis as B. We do not explain here how the reduction works. All we need for the analysis is the running time of the reduction and the quality of its output. We use Eqs. (2.3) and (2.2) as guarantees on the quality of B. Under these guarantees, during the second phase, we form a search space to enumerate candidates for the error $\vec{e}$ within this search space. The shortest candidate $\vec{e}'$ is output as the solution. We now explain how the actual decoding, i.e. enumeration, works.

Enumeration is done via orthogonal projection of the target vector $\vec{t}$ onto a (close) translate of the lattice $\mathcal{L}(B)$: $i\vec{b}_m + \mathrm{Span}(\vec{b}_1, \ldots, \vec{b}_{m-1})$ for some appropriately chosen $i \in \mathbb{Z}$. A projected vector that now belongs to an $m-1$-dimensional $\mathrm{Span}(\vec{b}_1, \ldots, \vec{b}_{m-1})$ together with the lattice $\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{m-1})$ forms a new BDD instance. We can run this procedure recursively. After m such recursive projections, we end up with a lattice-vector (the last projection onto a zero-dimensional space is just choosing a close point) and hope it is the closest to the original $\vec{t}$.

The way we choose close translates on each recursive step defines the search space of the enumeration: restricting the search to the fundamental parallelepiped of the Gram-Schmidt basis of the lattice $\mathcal{P}_{1/2}(B^*)$ gives Babai's `NearestPlane` algorithm [Bab85]. Enlarging it by stretching $\mathcal{P}_{1/2}(B^*)$ to a parallelepiped $\mathcal{P}_{1/2}(B^* \cdot D)$ for some diagonal matrix D results in the Linder-Peikert `NearestPlanes` algorithm [LP11]. Considering error-vectors $\vec{e}$ that lie within some ball of radius R gives rise to the Spherical Pruning [SE94]. The so-called Linear-Length pruning of [GNR10] forms a search space of cylinder intersections (i.e. a candidate error-vector $\vec{e}$ is enumerated if its coordinates satisfy the system of inequalities $\{e_1^2 \le R_1, e_1^2 + e_2^2 \le R_2, \ldots, \|\vec{e}\|^2 \le R_n\}$ for some input-sequence $R_1, \ldots, R_n$).

Certainly, one is free to choose a search space of any shape and hope it will bring an improvement to the enumeration phase. By 'improvement' we mean a better running-time/success probability trade-off ρ. We notice that the aforementioned pruning strategies share some common 'rules', which allows us to analyze them at one shot. To do that we define what we call a *reasonable* pruning strategy that describes these 'rules', give the algorithm `GenPruning` that follows this strategy, and analyze its complexity. It turns out that the Spherical Pruning, Linear-Length Pruning and other (called 'Extreme' in [GNR10]) pruning strategies are all reasonable, so it is sufficient to consider our generalization to conclude on their asymptotics.

To achieve our original goal – to determine the complexity of LWE as a BDD problem – we show in Sects. 3.1.1 – 3.1.3 the running time/success probability trade-off $\rho(\texttt{Enum}) = \frac{T(\texttt{Enum})}{P_{\mathrm{succ}}(\texttt{Enum})}$, where we consider $\texttt{Enum} \in \{\text{Babai's } \texttt{NearestPlane}, \text{Lindner-Peikert's } \texttt{NearestPlanes}, \texttt{GenPruning}\}$. Since `Enum` is only the second step of the whole algorithm, the actual trade-off on the BDD attack is $\rho(\mathsf{BDD}) = \frac{T(\texttt{BKZ})+T(\texttt{Enum})}{P_{\mathrm{succ}}(\texttt{Enum})}$. We take care about it in Sect. 3.1.4.

3.1.1 Babai's `NearestPlane` Algorithm

Suppose we are given a lattice-basis $B = (\vec{b}_1, \ldots, \vec{b}_m) \in \mathbb{Z}^{m \times m}$ and a target point $\vec{t} \in \mathbb{Q}^m$.[1] We search for a lattice-vector $\vec{v} \in \mathcal{L}(B)$ that is close to $\vec{t}$. Babai's algorithm works as follows. We view the m-dimensional lattice $\mathcal{L}(B)$ as the $m-1$-dimensional lattice $\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{m-1})$ translated via shifts $i\vec{b}_m$:

$$\mathcal{L}(B) = \bigcup_{i \in \mathbb{Z}} i\vec{b}_m + \mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{m-1}).$$

[1] While for LWE the target and the lattice agree in the dimension, this is not required for the algorithm to work. If they do not agree, we project $\vec{t}$ onto $\mathrm{Span}(\mathcal{L}(B))$ and work with the projection as the input target.

Fixing i, we receive a translate $i\vec{b}_m + \mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{m-1}) \subset U_i^{(m-1)}$ that is contained in the $m-1$-dimensional hyperplane $U_i^{(m-1)} = \left\{\vec{y} \in \mathbb{R}^m : \left\langle \vec{y}\ ,\ \frac{\vec{b}_m^*}{\|\vec{b}_m^*\|^2} \right\rangle = i\right\}$ (cf. Fig. 3.1a). Babai's algorithm chooses a hyperplane $U_i^{(m-1)}$ that is closest to $\vec{t}$ (see line 5 in Alg. 1) with the corresponding translate $\vec{x}^{(m)} = i\vec{b}_m$ (line 6), and then projects $\vec{t} = \vec{t}^{(m)}$ orthogonally onto $U_i^{(m-1)}$ to obtain $\vec{t}^{(m-1)}$ (line 8).

Now we have a new target $\vec{t}^{(m-1)}$ and a (shifted) sub-lattice $\vec{x}^{(m)} + \mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{m-1})$, so we repeat the process by choosing $U_j^{(m-2)} = \left\{\vec{y} \in \mathbb{R}^m : \left\langle \vec{y}\ ,\ \frac{\vec{b}_{m-1}^*}{\|\vec{b}_{m-1}^*\|^2} \right\rangle = j + \left\langle \vec{x}^{(m)}\ ,\ \frac{\vec{b}_{m-1}^*}{\|\vec{b}_{m-1}^*\|^2} \right\rangle \text{ and } \left\langle \vec{y}\ ,\ \frac{\vec{b}_m^*}{\|\vec{b}_m^*\|^2} \right\rangle = i\right\}$ closest to $\vec{t}^{(m-1)}$ (note that $U_j^{(m-2)} \subset U_i^{(m-1)}$). The shifts $\vec{x}^{(k)}$ accumulate the output vector $\vec{v}$ coordinate-wise w.r.t. the basis B starting from $\vec{b}_m$. In the algorithm described below, we also keep track of the error incurred by projections. The output error vector $\vec{e}'$ is constructed coordinate-wise w.r.t. the Gram-Schmidt basis B^* starting from $\vec{b}_m^*$.

Algorithm 1 Babai's `NearestPlane` $(B, \vec{x}, \vec{t})$

Input: $B = (\vec{b}_1, \ldots, \vec{b}_m) \in \mathbb{Z}^{m \times m}, \vec{x} \in \mathbb{Q}^m, \vec{t} \in \vec{x} + \mathrm{Span}(B), \vec{e}' \in \mathbb{Q}^m$ ▷ $\vec{e}' = \vec{x} = 0, k = m$ in the initial call
Output: $\vec{v} \in \mathcal{L}(B)$ close to $\vec{t}$ and $\vec{e}' = \vec{t} - \vec{v}$ corresponding error vector

1: $\vec{x}^{(k)} \leftarrow \vec{x}, \vec{t}^{(k)} \leftarrow \vec{t}, \vec{e}'^{(k)} \leftarrow \vec{e}'$.
2: Let $B^* \leftarrow \mathrm{GSO}(B)$.
3: **if** $k = 0$ **then return** $(\vec{x}, \vec{e}')$
4: Compute $c_1^{(k)} \leftarrow \left\langle \vec{t}^{(k)}\ ,\ \frac{\vec{b}_k^*}{\|\vec{b}_k^*\|^2} \right\rangle$
5: Choose $i^{(k)} \in \mathbb{Z}$ s.t. $c_2^{(k)} = \left\langle \vec{x}^{(k)}\ ,\ \frac{\vec{b}_k^*}{\|\vec{b}_k^*\|^2} \right\rangle + i^{(k)}$ closest to $c_1^{(k)}$
6: $\vec{x}^{(k-1)} \leftarrow \vec{x}^{(k)} + i^{(k)}\vec{b}_k$ ▷ $U_i^{(k-1)} = \vec{x}^{(k-1)} + \mathcal{L}(B^{(k-1)})$ is the nearest plane
7: $\vec{e}'^{(k-1)} \leftarrow \vec{e}'^{(k)} + (c_1^{(k)} - c_2^{(k)})\vec{b}_k^*$
8: $\vec{t}^{(k-1)} = \vec{t}^{(k)} - (c_1^{(k)} - c_2^{(k)})\vec{b}_k^*$ ▷ Project onto $U_i^{(k-1)}$
9: **return** `NearestPlanes`$((\vec{b}_1, \ldots, \vec{b}_{k-1}), \vec{x}^{(k-1)}, \vec{t}^{(k-1)}, \vec{e}'^{(k-1)})$

Analysis. It is easy to verify that Babai's Algorithm runs in time polynomial in m. In the context of LWE, the algorithm succeeds (i.e. the output vector $\vec{e}'$ is the LWE error $\vec{e}$) if $\vec{e}$ lies in the interior of $\mathcal{P}_{1/2}(B^*)$. In other words, if we write $\vec{e} = \sum_k e_k \frac{\vec{b}_k^*}{\|\vec{b}_k^*\|}$ w.r.t. the normalized Gram-Schmidt basis, we have $\vec{e} = \vec{e}'$ if $|e_k| < \frac{1}{2}\|\vec{b}_k^*\|$. If there exist an index k s.t. $|e_k| > \frac{1}{2}\|\vec{b}_k^*\|$, the algorithm fails. In case, $|e_k| = \frac{1}{2}\|\vec{b}_k^*\|$, there will be two equally close translates and we choose one arbitrarily. This case does not affect the asymptotics.

For the analysis, we approximate the discrete Gaussian $\vec{e}$ by a continuous one, i.e., the e_k's are assumed to be independent Gaussians with parameter αq. Note that expressing $\vec{e}$ in terms of the normalized Gram-Schmidt basis (instead of the standard $\mathbb{Z}$-basis) does not change the distribution of e_k as the former basis is just a rotation of the later one and the continuous Gaussian distribution is rotation-invariant.

Recall that Babai's algorithm receives as input a β-BKZ reduced basis B. Under the Geometric Series Assumption (Eq. (2.2)), the sequence $\|\vec{b}_1^*\|, \ldots, \|\vec{b}_m^*\|$ decays geometrically. Combining this with the guarantee on $\|\vec{b}_1\|$ (Eq. (2.3)), we can say whether (1) $\|\vec{b}_m^*\| > \alpha q$ (and the success probability of the algorithm is constant), or (2) $\|\vec{b}_1^*\| = \alpha q$ (the success probability is super-exponentially low). In the intermediate case (relevant for our LWE setting), $\|\vec{b}_m^*\| \ll \alpha q \ll \|\vec{b}_1^*\|$, all the steps k for which $\|\vec{b}_k^*\| \ll \alpha q$ contribute to a super-exponentially small success probability, while the steps starting from $\|\vec{b}_j^*\| \approx \alpha q$ do not change the success probability much. The next lemma formalizes these arguments.

Lemma 4. *Let the sequence $\|\vec{b}_1^*\|, \ldots, \|\vec{b}_m^*\|$ be geometrically decreasing with $\|\vec{b}_k^*\|/\|\vec{b}_{k+1}^*\| = \beta^{1/\beta} > 1$. Let $e_1, \ldots, e_m$ be independent continuous Gaussians with the density function $\varrho(x) = \frac{1}{s}\exp(-\frac{\pi x^2}{s^2})$. Denote $p_k := \Pr[|e_k| < \|\vec{b}_k^*\|]$.*

1. *If $\|\vec{b}_m^*\| > s(\log m)^{1/2+\varepsilon}$ for fixed constant $\varepsilon > 0$, then $\prod_k p_k = 1 - o(1)$.*

2. If $\|\vec{b}_1^*\| = s$*, then* $\prod_k p_k = 2^{-\mathcal{O}(m)} \cdot 2^m \beta^{-\frac{1}{2}\frac{m(m+1)}{\beta}}$.

Proof. To see the first statement, we use the Gaussian tails-bounds Eq. (2.5). For $\|\vec{b}_k\| > s(\log m)^{1/2+\varepsilon}$, $1-p_k$ is super-polynomially small, namely, $1-p_k = e^{-\Theta((\log m)^{1+2\varepsilon})}$. The result follows from the union bound and the fact that $e^{-t} \sim 1-t$ for $t \to 0$.

For the second statement, informally, we approximate the area under the Bell-shaped curve on the interval $[-\|\vec{b}_i^*\|, \|\vec{b}_k^*\|]$ with a parallelepiped. More precisely,

$$p_k = \frac{1}{2\varrho_s(\mathbb{Z})} \int_{-\|\vec{b}_k^*\|}^{\|\vec{b}_k^*\|} \exp\Big(-\frac{\pi x^2}{s^2}\Big)\mathrm{d}x = \Theta(1)\frac{2\|\vec{b}_k^*\|}{s}.$$

Then

$$\prod_k p_k = 2^{-\mathcal{O}(n)} \cdot 2^n \frac{\prod_k \|\vec{b}_k^*\|}{s^n} = 2^{-\mathcal{O}(m)} \cdot 2^n \frac{\|\vec{b}_1^*\|^m}{s^m} \prod_{i=2}^{m} \beta^{-i/\beta} = 2^{-\mathcal{O}(m)} \cdot 2^m \beta^{-\frac{1}{2}\frac{m(m+1)}{\beta}}.$$

□

For LWE, we have $s = \alpha q$. Recall that we relate the LWE parameters (n, q, α) as $q = \mathcal{O}(n^{\mathsf{c}_q})$, $\alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha})$ for positive constants $\mathsf{c}_q, \mathsf{c}_\alpha, \mathsf{c}_q > \mathsf{c}_\alpha$, from where it easily follows that the width $s = \alpha q$ is $\mathcal{O}(n^{\mathsf{c}_q - \mathsf{c}_\alpha})$. With the above lemma, we show the the success probability of Babai's algorithm depends on whether $\|\vec{b}_m^*\|$ is larger or smaller than $s = \alpha q$.

Theorem 5 (Analysis of the `NearestPlane` Algorithm 1)**.** *Given a* $\beta = \Theta(n)$*-BKZ reduced basis that arises from* $m = \Theta(n)$ *LWE-samples with parameters* $(n, q = \mathcal{O}(n^{\mathsf{c}_q}), \alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha}))$ *for positive constants* $\mathsf{c}_q > \mathsf{c}_\alpha$*, Babai's* `NearestPlane` *Algorithm 1 solves the Search-LWE problem in time* $\mathrm{poly}(m)$ *with success probability*

$$P_{succ}(\texttt{NearestPlane}) = \begin{cases} 2^{-\frac{1}{2}\left(\frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q\right)^2 (1+o(1))\cdot\beta\log\beta}, & \text{if } \frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q > 0 \\ 1 - o(1), & \text{if } \frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q < 0, \end{cases}$$

assuming the Geometric Series Assumption holds and the LWE error follows a continuous Gaussian distribution.

Proof. From Eqs. (2.2), (2.3), we have $\|\vec{b}_i^*\| = \beta^{\frac{m}{2\beta} - \frac{i}{\beta}} \cdot q^{1-\frac{n}{m}}$. We want to compute a 'critical' level k^* s.t. $\|\vec{b}_{k^*}^*\| = \alpha q$. It is easy to verify that $k^* = \beta\big(\frac{m}{2\beta} - \frac{n}{m}\mathsf{c}_q + \mathsf{c}_\alpha\big)$, from where it follows

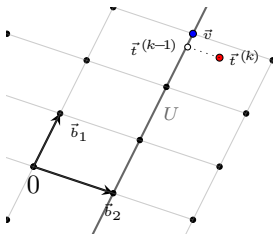
$$m - k^* = \beta\Big(\frac{m}{2\beta} + \frac{n}{m}\mathsf{c}_q - \mathsf{c}_\alpha\Big). \tag{3.1}$$

In case $\frac{m}{2\beta} + \frac{n}{m}\mathsf{c}_q - \mathsf{c}_\alpha < 0$, we have $m < k^*$ and $\|\vec{b}_m^*\| > \alpha q \cdot \mathrm{poly}(n)$. So all the Gram-Schmidt vectors are large enough to guarantee a super-polynomially small (in m) error probability on each level. From Lemma 4, first statement, the success probability of Algorithm 1 is then $1 - o(1)$.

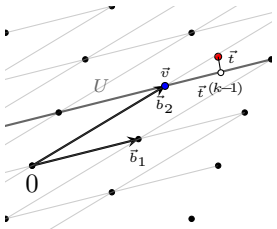
In case $\frac{m}{2\beta} + \frac{n}{m}\mathsf{c}_q - \mathsf{c}_\alpha > 0$, we use the second statement of Lemma 4 having $\|\vec{b}_{k^*}^*\| = \alpha q$ instead of $\|\vec{b}_1^*\|$ and obtain

$$P_{\mathrm{succ}}(\texttt{NearestPlane}) = 2^{-\mathcal{O}(m-k^*)} \cdot 2^{m-k^*} \beta^{-\frac{1}{2}\frac{(m-k^*)^2}{\beta}} = 2^{-\frac{1}{2}\left(\frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q + o(1)\right)^2 \cdot \beta\log\beta}.$$

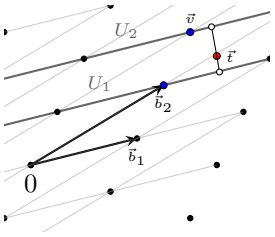
□



(a) Babai's `NearestPlane` Algorithm on a 'good' basis. The target point $t^{(k)}$ (red) is projected onto the closest hyperplane $U = \vec{b}_2 + \mathrm{Span}(\vec{b}_1)$. The recursive call for this 1-dimensional U projects $\vec{t}^{(k-1)}$ onto the closest zero-dimensional subspace, i.e. lattice point $\vec{v}$ (blue).



(b) Babai's `NearestPlane` Algorithm on a 'bad' basis for the same lattice. Now the target point $\vec{t}^{(k)}$ (red) is projected onto the hyperplane $U = \vec{b}_2 + \mathrm{Span}(\vec{b}_1)$ but for different $\vec{b}_2, \vec{b}_1$, so the hyperplane has changed. The returned vector $\vec{v}$ is not the closest vector.



(c) Lindner-Peikert `NearestPlanes` Algorithm on the same 'bad' basis. We set $\vec{d} = (1, 2)$ and project the target $\vec{t}$ onto two hyperplanes $U_1 = \vec{b}_2 + \mathrm{Span}(\vec{b}_1), U_2 = 2\vec{b}_2 + \mathrm{Span}(\vec{b}_1)$. The output points are marked blue. The closets point $\vec{v}$ is found among them.

Fig. 3.1: `NearestPlane(s)` Algorithms

3.1.2 Lindner-Peikert `NearestPlanes` Algorithm

For Eq. (3.1) to be smaller than 0, which translates to a constant success probability for the Babai's LWE decoding, the `BKZ` parameter β must be set almost as large as the lattice dimension m. For example, for parameters $q = \mathcal{O}(n^2), \alpha = \mathcal{O}(1/n^{3/2})$, Eq. (3.1) is smaller than 0 when $\beta > \frac{1}{2}\frac{m}{\mathsf{c}_\alpha - \frac{n}{m}\mathsf{c}_q}$. Setting $m = 2\frac{\mathsf{c}_q}{\mathsf{c}_\alpha}n$ (as this choice minimizes β), yields $\beta > 2\frac{\mathsf{c}_q}{\mathsf{c}_\alpha^2}n \approx \frac{16}{9}n$. For such large β, the `BKZ` reduction is not efficient. Choosing smaller β and, hence, decreasing the running time of `BKZ` reduction, leads to a super-exponentially small success probability of the decoding (case 1 in Thm. 5).

To amplify the success probability of Babai's algorithm (at the expense of its running time), Lindner and Peikert [LP11] proposed an extended variant of the `NearestPlane` algorithm. Instead of choosing only one closest hyperplane U_i (line 5 in Alg. 1), we choose several, say d, close hyperplanes (line 5 in Alg. 2) and project the target vector onto them. This results in d new targets which are, in turn, projected onto d' several close hyperplanes (see Fig. 3.1c). For example, Fig. 3.2a represents the case $m = 3, \vec{d} = (3, 2, 1)$. Babai's `NearestPlane` algorithm corresponds to $\vec{d} = \vec{1}$.

Geometrically this idea amounts to stretching the search region $V_{\text{Babai}} = \mathcal{P}_{1/2}(B^*)$ to $V_{\text{LP}} = \mathcal{P}_{1/2}(B^* \cdot D)$ for a diagonal matrix D having $(d_1, \ldots, d_m)$ on the main diagonal. In the end, we have $\prod_i d_i$ candidate error-vectors, out of which the shortest is chosen.

The formal description of this algorithm, which we call the `NearestPlanes` algorithm, is given in Alg. 2. In addition to a lattice and a target vector, the algorithm receives a vector $\vec{d} = (d_1, \ldots, d_m)$. Below we explain how to choose this vector.

Analysis. Like in the analysis of Babai's algorithm, we approximate the discrete Gaussian error $\vec{e}$ sampled with parameter αq by a continuous one. Our goal is to determine a choice of $\vec{d}$ that guarantees a constant success probability and from such a choice, deduce the running time of the `NearestPlanes` algorithm.

From [LP11], the success probability of the algorithm when applied to m LWE-samples is

$$P_{\text{succ}}(\texttt{NearestPlanes}) = \Pr[\vec{e} \in \mathcal{P}_{1/2}(B^*D)] = \prod_{i=1}^{m} \Pr\left[|\langle \vec{e}, \vec{b}_i^* \rangle| < \frac{d_i \|\vec{b}_i^*\|^2}{2}\right] = \prod_{i=1}^{m} \mathrm{erf}\left(\frac{d_i \|\vec{b}_i^*\|^2}{2\alpha q}\right), \quad (3.2)$$

where $\mathrm{erf} = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\mathrm{d}t$. Tail-bounds on the Gaussian distribution (Eq. (2.5)) suggest that if $\min_{1 \le i \le m} \frac{d_i \|\vec{b}_i^*\|}{\alpha q} = o(1)$, then $P_{\text{succ}}(\texttt{NearestPlanes}) = o(1)$. However, if $\min_{1 \le i \le m} \frac{d_i \|\vec{b}_i^*\|}{\alpha q} = \omega(\sqrt{\log m})$,

Algorithm 2 Lindner-Peikert $\texttt{NearestPlanes}_{\underline{\texttt{s}}}$ $(B, \vec{x}, \vec{t}, \vec{d})$

Input: $B = (\vec{b}_1, \ldots, \vec{b}_m) \in \mathbb{Z}^{m\times m}, \vec{x} \in \mathbb{Q}^m, \vec{t} \in \vec{x} + \mathrm{Span}(B), \vec{e}' \in \mathbb{Q}^m$ ▷ $\vec{e}' = \vec{x} = 0$ in the initial call
Output: A set of pair $(\vec{v}, \vec{e}')$ where $\vec{v} \in \mathcal{L}(B)$ and $\vec{e}' = \vec{v} - \vec{t}$

$\vec{x}^{(k)} \leftarrow \vec{x}, \vec{t}^{(k)} \leftarrow \vec{t}, \vec{e}'^{(k)} \leftarrow \vec{e}'$.
Let $B^* \leftarrow \mathrm{GSO}(B)$.
if $k = 0$ **then return** $\{(\vec{x}, \vec{e}')\}$
Compute $c_1^{(k)} \leftarrow \left\langle \vec{t}^{(k)}, \frac{\vec{b}_k^*}{\|\vec{b}_k^*\|^2} \right\rangle$
Compute $c_j^{(k)} = \left\langle \vec{x}^{(k)}, \frac{\vec{b}_k^*}{\|\vec{b}_k^*\|^2} \right\rangle + i_j^{(k)}$ for $i_j^{(k)} \in \mathbb{Z}, 1 \le j \le d_k$ s.t. $c_j^{(k)}$ are closest to $c_1^{(k)}$
for each $(i_j^{(k)}, c_j^{(k)})$ **do**
    $\vec{x}_j^{(k-1)} \leftarrow \vec{x}^{(k)} + i_j^{(k)}\vec{b}_k$ ▷ $U_j^{(k-1)} = \vec{x}_j^{(k-1)} + \mathcal{L}(B^{(k-1)})$ are the d_k nearest planes
    $\vec{e}_j'^{(k-1)} \leftarrow \vec{e}'^{(k)} + (c_1^{(k)} - c_j^{(k)})\vec{b}_k^*$
    $\vec{t}_j^{(k-1)} = \vec{t}^{(k)} - (c_1^{(k)} - c_j^{(k)})\vec{b}_k^*$ ▷ Project onto $U_j^{(k-1)}$
    return $\bigcup_j \texttt{NearestPlanes}_{\underline{\texttt{s}}}((\vec{b}_1, \ldots, \vec{b}_{k-1}), \vec{x}_j^{(k-1)}, \vec{t}_j^{(k-1)}, \vec{e}_j'^{(k-1)}, \vec{d})$

then $P_{\text{succ}}(\texttt{NearestPlanes}_{\underline{\texttt{s}}}) = 1 - o(1)$. Setting

$$d_i = \left\lceil \frac{\alpha q \cdot (\log m)^c}{\|\vec{b}_i^*\|} \right\rceil \tag{3.3}$$

for some constant $c > 1/2$, the decoding succeeds with probability almost 1. In case $\|\vec{b}_1^*\| \gg \alpha q$ (which is the case for LWE), we set the first d_i's equal to 1 and start increasing the sequence once $\|\vec{b}_i^*\|$'s become equal or smaller than αq.

The recursive $\texttt{NearestPlanes}_{\underline{\texttt{s}}}$ Algorithm 2 has a tree-structure (see Fig. 3.2a) with the root corresponding to the initial call and every node is created by projecting the target onto one out of d_i hyperplanes. As the result, each node on level k has d_{m-k+1} children. The superscripts for $\vec{x}^{(k)}, \vec{e}'^{(k)}, \vec{t}^{(k)}$ denote the level, the root is on level m, the leaves are on level 0. Vectors $\vec{x}_j^{(k)}, \vec{e}_j'^{(k)}, \vec{t}_j^{(k)}$ are the data associated to one node on level k, giving (1) a partial solution (w.r.t. the basis B), (2) a partial error-vector (w.r.t. the basis B^*), and (3) a new target.

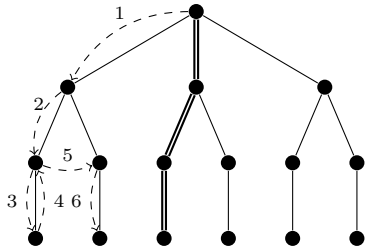
Let N_k be the number of nodes at level k and N be the total number of nodes. At the root we have $N_m = 1$. Down the tree, we have $N_k = \prod_{i=k+1}^m d_i$ and $N = \sum_{k=0}^m N_k$. The work done on a node is clearly polynomial in m, so the total complexity of Alg. 2 is $N \cdot \mathrm{poly}(m)$. The following theorem gives the value for N when the d_i's are set as in Eq. (3.3).

Theorem 6 (Analysis of the $\texttt{NearestPlanes}_{\underline{\texttt{s}}}$ Algorithm 2)**.** *Given a $\beta = \Theta(n)$-BKZ reduced basis that arises from $m = \Theta(n)$ LWE-samples with parameters $(n, q = \mathcal{O}(n^{\mathsf{c}_q}), \alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha}))$ for positive constants $\mathsf{c}_q > \mathsf{c}_\alpha$, the $\texttt{NearestPlanes}_{\underline{\texttt{s}}}$ Algorithm 2 with the $\vec{d}$ set as in Eq. (3.3), solves the Search-LWE problem with success probability $1 - o(1)$ in time*

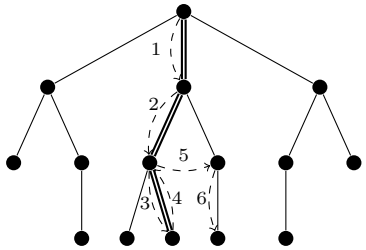
$$T(\texttt{NearestPlane}) = \mathrm{poly}(m) \cdot N = \begin{cases} 2^{\frac{1}{2}\left(\frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q\right)^2 (1+o(1))\cdot\beta\log\beta}, & \text{if } \frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q > 0 \\ \mathrm{poly}(m) & \text{if } \frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q < 0, \end{cases}$$

and $\mathrm{poly}(m)$ memory (using depth-first search) assuming the Geometric Series Assumption holds and the LWE error follows a continuous Gaussian distribution.

Proof. As in Thm. 5, under GSA, the inequality $\frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q < 0$ translates into $\|\vec{b}_m^*\| > \alpha q \cdot \mathrm{poly}(n)$. It immediately gives $d_i = \lceil \frac{\alpha q \cdot (\log m)^c}{\alpha q \cdot \mathrm{poly}(n)} \rceil = 1$ (for some constant $c > 1/2$). This is exactly Babai's $\texttt{NearestPlane}$ algorithm which has $\mathrm{poly}(m)$ running time.



(a) Enumeration tree of the Lindner-Peikert algorithm for 3-dimensional lattice with $\vec{d} = (3, 2, 1)$.



(b) Enumeration tree of the Generalized Pruning algorithm realized via some bounding function $\mathfrak{B}$. As opposed to the left figure, the number of children varies for nodes on the same level.

Fig. 3.2: Enumeration tree of the Lindner-Peikert Algorithm (left) and the Generalized Pruning (right). The roots correspond to the initial call, the leaves contain the candidate error-vectors and the corresponding lattice vectors. The double-line represents the path (i.e. the choices of hyperplanes) that would have been chosen by the Babai's `NearestPlane` Alg. 1. The dashed curved arrows show the order of tree-traversals. On the left, the left-most child is visited first (usual depth-first tree-traversal). On the right, the 'best' (w.r.t. the error-length) child is chosen first (the *best-first* traversal).

In case $\frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q > 0$, we have to increase some d_i's to guarantee the desired success probability. Again as in Thm. 5, there exist a critical level k^* s.t. $\|\vec{b}^*_{k^*}\| > \alpha q$ and k^* is maximal. This level is determined in Eq. (3.1) and we have $d_{k^*+1} > 1$, i.e. we increase the d_i's from this level. Since $N < (m+1)N_0$ (i.e. the total number of nodes is essentially determined by the number of leaves), the running time of Alg. 2 is given by (up to poly(m) factors) $N_0 = \prod_{i=1}^m d_i$. We have

$$N_0 = \prod_{i=1}^{m} d_i = \prod_{i=1}^{m} \left\lceil \frac{\alpha q \cdot (\log m)^c}{\|\vec{b}^*_i\|} \right\rceil < (1 + (\log m)^c)^m \cdot \prod_{i=1}^{k^*} \left\lceil \frac{\alpha q}{\|\vec{b}^*_i\|} \right\rceil \prod_{i=k^*+1}^{m} \left\lceil \frac{\alpha q}{\|\vec{b}^*_i\|} \right\rceil$$
$$\leq (1 + \log m^c)^m 2^{m-k^*} \cdot \prod_{i=k^*}^{m} \left\lceil \frac{\alpha q}{\|\vec{b}^*_i\|} \right\rceil = (1 + \log m^c)^m 2^{m-k^*} \cdot 2^{\left(\frac{(m-k^*)^2}{2\beta^2} + o(1)\right)\beta \log \beta},$$

where the last product $\prod_{i=k^*+1}^{m} \left\lceil \frac{\alpha q}{\|\vec{b}^*_i\|)} \right\rceil$ was already computed in Lemma 4 and Thm. 5. The factor $(1 + \log m^c)^m 2^{m-k^*} = 2^{\mathcal{O}(m \log \log m)}$ contributes to the $o(1)$-term in the theorem statement. □

The following corollary follows immediately from Thms. 5 and 6.

Corollary 7. *Given a $\beta = \Theta(n)$-BKZ reduced basis that arises from $m = \Theta(n)$ LWE-samples with parameters $(n, q = \mathcal{O}(n^{\mathsf{c}_q}), \alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha}))$ for positive constants $\mathsf{c}_q > \mathsf{c}_\alpha$, the decoding algorithms `Enum` $\in \{$Babai's `NearestPlane`, Lindner-Peikert's `NearestPlanes` with $\vec{d}$ set as in Eq. (3.3)$\}$ attain the running time/success probability trade-off*

$$\rho(\texttt{Enum}) = \frac{T(\texttt{Enum})}{P_{succ}(\texttt{Enum})} = \begin{cases} 2^{\frac{1}{2}\left(\frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q\right)^2 (1+o(1))\cdot\beta \log \beta}, & \text{if } \frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q > 0 \\ \text{poly}(m) & \text{if } \frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q < 0, \end{cases}$$

assuming the Geometric Series Assumption holds and the LWE error follows a continuous Gaussian distribution.

3.1.3 Generalized Pruning Algorithm

Spherical and Linear-Length Pruning. Note that the error-vector $\vec{e}' = \sum_i e'_i \frac{\vec{b}^*_i}{\|\vec{b}^*_i\|}$ is not explicitly bounded during the Lindner-Peikert's enumeration. It is done implicitly via restricting its individual coordinates e'_i. Each coordinate e'_i is obtained by going one level down the enumeration tree: on level k we compute $\vec{e}'^{(k)} = \sum_{i=k+1}^{m} e'_i \frac{\vec{b}^*_i}{\|\vec{b}^*_i\|}$ by adding $e'_{k+1} \frac{\vec{b}^*_i}{\|\vec{b}^*_i\|}$ to $\vec{e}'^{(k+1)}$ for an appropriately chosen coordinate

e'_{k+1} (see line 8 in Alg. 2). By the end of the enumeration, $\vec{e}'^{(k)}$ builds a final output $\vec{e}'$ and hence, this final $\vec{e}'$ will have length greater than $\|\vec{e}'^{(k)}\|$. A moment's thought reveals that it is reasonable to make the number of children for a node dependent on the length of the error $\|\vec{e}'^{(k)}\|$ associated to this node: the smaller this length is, the more children we want this node to have as it is more likely to lead to the correct solution, while if $\|\vec{e}'^{(k)}\| \gg \sqrt{m}\alpha q$ (i.e. the accumulated error-length exceeds the expected length of the LWE error), we might as well choose no children at all and stop the recursion for this node. This is why we use the term 'pruning' as we prune the enumeration tree at some unpromising nodes.

For example, the *Spherical Pruning* ([SE94]) chooses the number of the children for a k^{th}-level node with $\|\vec{e}'^{(k)}\|^2 = \sum_{i=k+1}^{m} e_i'^2$ as $\Theta\left(\frac{1}{\|\vec{b}_k^*\|^2}(m(\alpha q)^2 - \sum_{i=k+1}^{m} e_i'^2)\right)$. This choice exactly captures our intuition: the shorter the accumulated error, the more children a node is allowed to have. We divide by $\|\vec{b}_k^*\|^2$ to get the actual number of the hyperplanes we recurse on (i.e. the number of children). It is easy to see that a pruning strategy enumerates all possible error-vectors that lie within the ball $\mathcal{B}(\vec{t}, \alpha q)$.

Another pruning strategy, the *Linear-Length Pruning* ([GNR10]), reduces the search space of the Spherical Pruning by allowing a k^{th}-level error-vector to have norm $\|\vec{e}'^{(k)}\|^2 \leq \Theta((m-k+1)(\alpha q)^2)$. This upper bound is the expected length of an $(m-k+1)$-dimensional vector with Gaussian entries of width αq. It results, as in case of the Spherical Pruning, in the output error-vectors having norm $\Theta(m\sqrt{\alpha q})$, but it is more restrictive on the intermediate levels.

There are several other pruning strategies considered in [GNR10], all aiming at reducing the search space by pruning the enumeration tree more aggressively, thus reducing the running time but sacrificing success probability (and hence, they are called *Extreme Pruning* with the most 'extreme' case being Babai's `NearestPlane`).

Generalized Pruning. All the enumeration strategies considered here can be described via a family of bounding functions $\mathfrak{B}^{(k)} : \mathbb{Q}_{\geq 0}^{m-k} \to \mathbb{Q}_{\geq 0}$, $1 \leq k \leq m$. In general, this function may not even be efficiently computable in practice. For example, Aono in [Aon14] suggests a way to find an optimal $\mathfrak{B}^{(k)}$ given the Gram-Schmidt basis B^* via solving an m-dimensional optimization problem. But for our `GenPruning` algorithm (Alg. 3) and its analysis we assume an efficiently computable family $\mathfrak{B}^{(k)}$ which is given as an additional input.

The algorithm computes the maximal allowed distance (denoted $D_{\max}$) to the next hyperplanes (line 6 of Alg. 3) and chooses only the hyperplanes for which $e_k'^2$ satisfies $e_k'^2 < \mathfrak{B}^{(k)}(e_m'^2, \ldots, e_{k+1}'^2) = D_{\max}^2$. The search region of `GenPruning` is

$$V_{\texttt{GP}} = \left\{\vec{e}' = \sum_k e'_k \frac{\vec{b}_k^*}{\|\vec{b}_k^*\|} : e_k^2 \leq \mathfrak{B}^{(k)}(e_m'^2, \ldots, e_{k+1}'^2)\ \forall k\right\}.$$

The algorithm successfully solves the LWE problem if the LWE error-vector $\vec{e}$ is contained in the search region $V_{\texttt{GP}}$. It is easy to see that Generalized Pruning captures all the enumeration strategies discussed here. Namely,

- $\mathfrak{B}^{(k)}(e_m'^2, \ldots, e_{k+1}'^2) = \left(\frac{\|\vec{b}_k^*\|}{2}\right)^2$: Babai's `NearestPlane`
- $\mathfrak{B}^{(k)}(e_m'^2, \ldots, e_{k+1}'^2) = \left(\frac{d_k\|\vec{b}_k^*\|}{2}\right)^2$: Lindner-Peikert's `NearestPlanes`
- $\mathfrak{B}^{(k)}(e_m'^2, \ldots, e_{k+1}'^2) = \Theta(m(\alpha q)^2) - \sum_{i=k+1}^{m} e_i'^2$: Spherical Pruning
- $\mathfrak{B}^{(k)}(e_m'^2, \ldots, e_{k+1}'^2) = \Theta((m-k)(\alpha q)^2) - \sum_{i=k+1}^{m} e_i'^2$: Linear Pruning
- $\mathfrak{B}^{(k)}(e_m'^2, \ldots, e_{k+1}'^2) = R_k^2 - \sum_{i=k+1}^{m} e_i'^2$: pruned strategy with some level-dependent bounds R_k.

Algorithm 3 Generalized Pruning Algorithm $\texttt{GenPruning}(B, \vec{x}, \vec{t}, \vec{e}', \mathfrak{B}^{(k)})$

Input: $B = (\vec{b}_1, \ldots, \vec{b}_m) \in \mathbb{Z}^{m \times m}, \vec{x} \in \mathbb{Q}^m, \vec{t} \in \vec{x} + \mathrm{Span}(B), \vec{e}' \in \mathbb{Q}^m, \mathfrak{B}^{(k)}$ ($\vec{e}' = \vec{x} = 0$ in the initial call)

Output: A set of pairs $(\vec{v}, \vec{e}')$ with $\vec{v} \in \vec{x} + \mathcal{L}(B)$ and $\vec{e}' = \vec{t} - \vec{v}$ corresponding error vector

$\vec{x}^{(k)} \leftarrow \vec{x}, \vec{t}^{(k)} \leftarrow \vec{t}, \vec{e}'^{(k)} \leftarrow \vec{e}'$

Let $B^* \leftarrow \mathrm{GSO}(B)$

if $k = 0$ **then return** $\{(\vec{x}, \vec{e}')\}$

Compute $c_1^{(k)} \leftarrow \langle \vec{t}^{(k)}, \frac{\vec{b}_k^*}{\|\vec{b}_k^*\|^2} \rangle$

Let $e'_i = \langle \vec{e}'^{(k)}, \frac{\vec{b}_i^*}{\|\vec{b}_i^*\|} \rangle$ for $k < i \leq m$ ▷ Coefficients of $\vec{e}'$

Let $D^2_{\max} = \mathfrak{B}^{(k)}(e'^2_m, \ldots, e'^2_{k+1})$ ▷ bound on distance of next hyperplanes

Compute $c_j^{(k)} = \langle \vec{x}^{(k)}, \frac{\vec{b}_k^*}{\|\vec{b}_k^*\|^2} \rangle + i_j^{(k)}$ for all $i_j^{(k)} \in \mathbb{Z}$ s.t. $\left| c_1^{(k)} - c_j^{(k)} \right|^2 \cdot \|\vec{b}_k^*\|^2 \leq D^2_{\max}$

for each $(i_j^{(k)}, c_j^{(k)})$ **do**

  $\vec{x}_j^{(k-1)} \leftarrow \vec{x}^{(k)} + i_j^{(k)} \vec{b}_k$ ▷ $U_j^{(k-1)} = \vec{x}_j^{(k-1)} + \mathcal{L}(B^{(k-1)})$ are the nearby planes

  $\vec{e}_j'^{(k-1)} \leftarrow \vec{e}'^{(k)} + (c_1^{(k)} - c_j^{(k)}) \vec{b}_k^*$

  $\vec{t}_j^{(k-1)} = \vec{t}^{(k)} - (c_1^{(k)} - c_j^{(k)}) \vec{b}_k^*$ ▷ Project onto $U_j^{(k-1)}$

  return $\bigcup_j \texttt{GenPruning}((\vec{b}_1, \ldots, \vec{b}_{k-1}), \vec{x}_j^{(k-1)}, \vec{t}_j^{(k-1)}, \vec{e}_j'^{(k-1)}, \mathfrak{B}^{(k)})$

All the extreme pruning approaches of [GNR10] as well any numerically optimized strategy are covered by the last choice of $\mathfrak{B}^{(k)}$.

For the analysis, we extend $\mathfrak{B}^{(k)}$ to real-valued arguments so that we can apply the algorithm to a continuous Gaussian.

Analysis. As before, we are interested in the ratio $\rho(\texttt{GP}) = \frac{T(\texttt{GP})}{P_{\mathrm{succ}}(\texttt{GP})}$ (GP is used as a shorthand for Generalized Pruning). A family of bounding functions $\mathfrak{B}^{(k)}$ defines the search region on level k as

$$V_{\texttt{GP}}(k) = \Big\{ \vec{e}' = \sum_{i=k+1}^{m} e'_i \frac{\vec{b}_i^*}{\|\vec{b}_i^*\|} : e_j^2 \leq \mathfrak{B}^{(j)}(e'^2_m, \ldots, e'^2_{j+1}) \; k < j \leq m \Big\}. \tag{3.4}$$

Let N_k be the number of nodes at level k and p_k be the probability that the correct solution is retained on level k (i.e. there exists a node with $\vec{e}^{(k)}$ that can be extended to the correct LWE error $\vec{e}$ by traversing the tree down to the last level). We define a *reasonable pruning* via requiring a set of conditions to be met by $\mathfrak{B}^{(k)}, N_k, p_k$. We assume that the error we seek for, follows a continuous Gaussian with parameter αq and that $\|\vec{b}_m^*\| < \alpha q < \|\vec{b}_1^*\|$. The latter is satisfied by the choice of β.

Definition 8 (Reasonable Pruning)**.** *Let k^* be the maximal level s.t. $\|\vec{b}_{k^*}^*\| > \alpha q$. The Generalized Pruning algorithm with the associated $\mathfrak{B}^{(k)}$ is* reasonable *if the following conditions are satisfied*

1. $\mathfrak{B}^{(k)}(e'^2_m, \ldots, e'^2_{k+1}) + \sum_{i=k+1}^{m} e'^2_i = \Theta(m(\alpha q)^2)$
2. $P_{succ}(\mathit{GP}) \geq 2^{-\mathcal{O}(m)} \cdot p_{k^*}$
3. $N_k \leq 2^{\mathcal{O}(m)} \cdot N_{k^*}$ *for* $k \leq k^*$
4. $\frac{N_{k-1}}{N_k} = \Omega(1)$ *for* $k \geq k^*$
5. $V_{\mathit{GP}}(k^*)$ *is convex.*

Informally, the conditions in Def. 8 have the following meaning: Condition 1 implies that we prune the nodes with the accumulated error larger than the expected length of the LWE error. Conditions 2 and 3 mean that as soon as the correct error survived until the 'critical' level k^*, we can find $\vec{e}$ with

high probability (Condition 2) at essentially no additional cost (Condition 3). For example, from k^* down, we can start the Babai's `NearestPlane` algorithm. Condition 4 ensures that for levels above k^* we choose *at least* a constant number of hyperplanes (line 7 in Alg. 3) to recurse on. Note that on these upper levels we have $\|\vec{b}_k^*\| < \alpha q$ for $k > k^*$, so we must have $\Omega(1)$ hyperplanes at distance at most αq.

We elaborate on the convexity condition a bit more. Let us take a closer look at the search region $V_{\mathtt{GP}}$. From the way the error-vectors $\vec{e}'^{(k)}$ are constructed during the algorithm, the number of nodes at level k, denoted N_k, is the number of points in $V_{\mathtt{GP}}(k) - \vec{e}$ that belong to the lattice $\mathcal{L}(\vec{b}_m^*, \ldots, \vec{b}_{k+1}^*)$, i.e. to the orthogonal projection of $\mathcal{L}(B)$ onto $\mathrm{Span}(\vec{b}_1, \ldots, \vec{b}_k)$ (the shift by $\vec{e}$ does not change N_k asymptotically but is needed for the proof below). The Gaussian Heuristic suggests that

$$N_k \approx \frac{\mathrm{vol}\, V_{\mathtt{GP}}(k)}{\prod_{i=k+1}^{m} \|\vec{b}_i^*\|}.$$

We could use this approximation in our theorem below, but since it is enough in our setting to upper-bound N_k up to a factor of $2^{\mathcal{O}(m)}$, we can prove the above equation by relying on a variant of Minkowski's Convex Body Theorem. Roughly, this theorem tells us that if an m-dimensional 0-symmetric convex point-set ($V_{\mathtt{GP}}(k)$ for us) has volume larger than the volume of a lattice, then it contains a non-zero point of this lattice.

The generalization of this result is due to Rado [Rad46]: instead of a 0-symmetric convex set, he considers a non-negative integrable function $f(\vec{x})$ and connects the quantity $\sum_{\vec{v}\in\mathcal{L}} f(\vec{v})$ with the value $\mathcal{V}(f) = \int_{-\infty<x_i<\infty} f(\vec{x})\mathrm{d}\vec{x}$. The convexity condition of the set is replaced by the quasi-concavity condition for f: for a linear map Λ from an m-dimensional vector space into itself, the function f must satisfy $f(\Lambda\vec{x} - \Lambda\vec{y}) \geq \min\{f(\vec{x}), f(\vec{y})\}$. Later in Thm. 10, we consider Λ acting like $\Lambda\vec{x} = \frac{1}{2}\vec{x}$. Minkowski's Convex Body Theorem is a special case of the Thm. 9 for $f(\vec{x})$ being a characteristic function of a convex symmetric set V in which case $\mathcal{V}(f) = \mathrm{vol}\, V$.

Theorem 9 (Rado's generalization of Minkowski's Convex Body Theorem)**.** *Let $f(\vec{x})$ be a non-negative integrable function with the property $f(\Lambda\vec{x} - \Lambda\vec{y}) \geq \min\{f(\vec{x}), f(\vec{y})\}$ for all $\vec{x}, \vec{y} \in \mathbb{R}^m$ and a linear map $\Lambda : \mathbb{R}^m \to \mathbb{R}^m$. Then*

$$f(\vec{0}) + \tfrac{1}{2}\sum_{\vec{0}\neq\vec{v}\in\mathcal{L}} f(\vec{v}) \geq \frac{|\det\Lambda|}{\det(\mathcal{L})}\mathcal{V}(f), \tag{3.5}$$

for every lattice $\mathcal{L}$, where

$$\mathcal{V}(f) = \int\limits_{\substack{-\infty<x_i<\infty \\ 1\leq i\leq m}} f(\vec{x})\mathrm{d}\vec{x},$$

and $\det(\Lambda)$ is the determinant of the $m \times m$ matrix that defines the transformation Λ.

Following the proof of the theorem (see the book of Cassels [Cas97, Chap. III] for a comprehensive proof) and setting $\Lambda\vec{x} = \frac{1}{2}\vec{x}$ (so $|\det\Lambda| = 2^{-m}$), we observe that up to $2^{\mathcal{O}(m)}$ the inequality given in Eq. (3.5) is an equality, so in our proof we rely on the fact that

$$f(\vec{0}) + \tfrac{1}{2}\sum_{\vec{0}\neq\vec{v}\in\mathcal{L}} f(\vec{v}) = \frac{2^{\pm\mathcal{O}(m)}}{\det(\mathcal{L})}\mathcal{V}(f). \tag{3.6}$$

Now we have the tools to present the main theorem of this section *without* relying on the Gaussian Heuristic. We apply Eq. (3.6) to the function $f(\vec{x}) = \frac{1}{(\alpha q)^{k^*}}\int_{V_{\mathtt{GP}}(k^*)} \exp\left(\frac{-\pi\|\vec{y}\,\|^2}{(\alpha q)^2}\right)\mathrm{d}\vec{y}$ and to the lattice $\mathcal{L}(\vec{b}_m^*, \ldots, \vec{b}_{k^*+1}^*)$, where k^* is the 'critical' level as in Def. 8. $f(\vec{x})$ is the convolution of the Gaussian density function with the characteristic function of our convex search region $V_{\mathtt{GP}}(k^*)$ and hence, it satisfies the quasiconcavity condition of Thm. 9.

Theorem 10 (Analysis of the `GenPruning` Algorithm 3)**.** *Given a $\beta = \Theta(n)$-BKZ reduced basis that arises from $m = \Theta(n)$ LWE-samples with parameters $(n, q = \mathcal{O}(n^{\mathsf{c}_q}), \alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha}))$ for positive constants $\mathsf{c}_q > \mathsf{c}_\alpha$ s.t. $\frac{m}{2\beta} + \frac{n}{m}\mathsf{c}_q - \mathsf{c}_\alpha > 0$, any reasonable (as in Def. 8) pruning algorithm GP has an expected (over the choice of $\vec{e}$) running time/success probability trade-off*

$$\rho(\mathit{GP}) = \frac{\mathbb{E}[T(\mathit{GP})]}{P_{succ}(\mathit{GP})} = 2^{\frac{1}{2}\left(\frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q\right)^2 (1+o(1))\cdot\beta\log\beta},$$

with $\mathrm{poly}(m)$ *memory (using depth-first search) assuming the Geometric Series Assumption holds and the LWE error follows a continuous Gaussian distribution.*

Proof. The amount of work done on one node of the enumeration tree of Alg. 3 is clearly $\mathrm{poly}(m)$, so the total running time is $T(\mathtt{GP}) = \mathrm{poly}(m) \cdot \sum N_k$. For the 'critical' level k^* (i.e. maximal level with $\|\vec{b}^*_{k^*}\| > \alpha q$), we have $N_i < 2^{\mathcal{O}(m)}$ for all i (for $i < k^*$ due to Condition 3 in Def. 8, for $i > k^*$ by the definition of N_k). So for a reasonable pruning strategy, it holds that

$$T(\mathtt{GP}) = 2^{\mathcal{O}(m)} N_{k^*}.$$

As discussed above, N_{k^*} is (up to $2^{\mathcal{O}(m)}$) the number of lattice vectors $\vec{v} \in \mathcal{L}(\vec{b}^*_m, \ldots, \vec{b}^*_{k^*+1})$ that lie in the convex region $V_{\mathtt{GP}} - \vec{e}$. Taking the expectation over the choice of $\vec{e}$,

$$\mathop{\mathbb{E}}_{\vec{e}}[N_{k^*}] = \sum_{\vec{v}\in\mathcal{L}(\vec{b}^*_m,\ldots,\vec{b}^*_{k^*+1})} f(x) \qquad \text{where} \qquad f(\vec{x}) = \tfrac{1}{(\alpha q)^{k^*}} \int\limits_{V_{\mathtt{GP}}(k^*)} \exp\left(\frac{-\pi\|\vec{y}\,\|^2}{(\alpha q)^2}\right) \mathrm{d}\vec{y}.$$

This expectation is essentially the left-hand side of Eq. (3.6). Note that $f(x)$ is quasiconcave (as it is a convolution of two log-concave functions: the Gaussian density function and the characteristic function of the convex region $V_{\mathtt{GP}}$). For $\mathcal{L} = \mathcal{L}(\vec{b}^*_m, \ldots, \vec{b}^*_{k^*+1})$, we obtain

$$\begin{aligned}\mathop{\mathbb{E}}_{\vec{e}}[N_{k^*}] &= \frac{2^{\pm\mathcal{O}(m)}}{\det(\mathcal{L})}\int f(\vec{x})\mathrm{d}\vec{x} = \frac{2^{\pm\mathcal{O}(m)}}{\det(\mathcal{L})}\int\limits_{V_{\mathtt{GP}}(k^*)}\int\limits_{\substack{-\infty<y_i<\infty\\ 1\le i\le m}} \frac{1}{(\alpha q)^{k^*}}\exp\left(\frac{-\pi\|\vec{y}\,\|^2}{(\alpha q)^2}\right)\mathrm{d}\vec{y}\,\mathrm{d}\vec{x} = \\ &= \frac{2^{\pm\mathcal{O}(m)}\operatorname{vol} V_{\mathtt{GP}}(k^*)}{\det(\mathcal{L})}.\end{aligned}$$

If the pruning is reasonable, we have $P_{\mathrm{succ}}(\mathtt{GP}) = 2^{-\mathcal{O}(m)} \cdot p_{k^*}$, where p_{k^*} is the probability that there exist a vector in V_{k^*} s.t. its $m - k^*$ coordinates extend to the LWE error-vector, yielding

$$\begin{aligned}\frac{\mathbb{E}[T(\mathtt{GP})]}{P_{\mathrm{succ}}(\mathtt{GP})} &= \frac{2^{\pm\mathcal{O}(m)}\operatorname{vol} V_{\mathtt{GP}}(k^*)}{\det(\mathcal{L})\cdot\frac{1}{(\alpha q)^{m-k^*}}\int\limits_{\vec{x}\in V_{\mathtt{GP}}(k^*)}\exp(-\frac{\pi\|\vec{x}\|^2}{(\alpha q)^2})\mathrm{d}\vec{x}} = \frac{2^{\pm\mathcal{O}(m)}\cdot(\alpha q)^{m-k^*}\operatorname{vol} V_{\mathtt{GP}}(k^*)}{\prod_{i=m-k^*}^m\|\vec{b}^*_i\|\cdot\int\limits_{\vec{x}\in V_{\mathtt{GP}}(k^*)}\exp(-\frac{\pi\|\vec{x}\|^2}{(\alpha q)^2})} = \\ &= \frac{2^{\pm\mathcal{O}(m)}\cdot(\alpha q)^{m-k^*}\int_{\vec{x}\in V_{\mathtt{GP}}(k^*)}1\mathrm{d}\vec{x}}{\prod_{i=m-k^*}^m\|\vec{b}^*_i\|\cdot\int_{\vec{x}\in V_{\mathtt{GP}}(k^*)}1\mathrm{d}\vec{x}} = 2^{\frac{1}{2}\left(\frac{m}{2\beta}-\mathsf{c}_\alpha+\frac{n}{m}\mathsf{c}_q\right)^2(1+o(1))\cdot\beta\log\beta},\end{aligned}$$

where for the third equality we used Condition 1 in Def. 8 stating that $e^{-\mathcal{O}(m)} \le \exp(-\frac{\pi\|\vec{x}\|^2}{(\alpha q)^2}) \le 1$ for any $\vec{x} \in V_{\mathtt{GP}}(k^*)$. □

We remark that for a pruning strategy that has $P_{\mathrm{succ}} \le p_{k^*}$ and $T \ge \mathrm{poly}(m)N_{k^*}$, the result of Thm. 10 gives a lower bound on the trade-off ρ for such a strategy.

Finally, note that the Lindner-Peikert `NearestPlanes` algorithm is a corner-case of a reasonable pruning: in the corners of the parallelepiped-shaped search region $V_{\mathtt{NearestPlanes}}$ Condition 1 is not

satisfied. This is why we analyzed it separately. Now we can move on to the discussion on the total complexity of the BDD attack on LWE where we balance the running time of the reduction and enumeration phases.

3.1.4 Total complexity of LWE decoding

The BDD attack on LWE is a two-phase algorithm: the enumeration (phase 2) in performed on a β-BKZ reduced basis (phase 1). So far we have been discussing the second phase – enumeration – ignoring the complexity of the reduction. By now we have the quantity $\rho(\texttt{Enum}) = \frac{T(\texttt{Enum})}{P_{\mathrm{succ}}(\texttt{Enum})}$, and we can finally turn our attention to the total complexity of the attack taking into account the reduction: $\rho(\mathsf{BDD}) = \frac{T(\texttt{BKZ})+T(\texttt{Enum})}{P_{\mathrm{succ}}(\texttt{Enum})}$.

$T(\texttt{BKZ})$ is determined by the input parameter β or, more precisely, by the running time of an SVP-solver on a lattice of dimension β (the dimension of the original lattice – m for LWE– only affects a polynomial factor in $T(\texttt{BKZ})$). We have already mentioned two ways to instantiate an SVP-solver in Chap. 2.1:

- $T(\texttt{BKZ}) = 2^{c_{\texttt{BKZ}}\cdot\beta\log\beta+o(\beta\log\beta)}$ with poly(β) space complexity due to Kannan [Kan83] for $c_{\texttt{BKZ}} = \Theta(1)$ (Hanrot and Stehlé in [HS07] estimate $c_{\texttt{BKZ}} = \frac{1}{2e}$),
- $T(\texttt{BKZ}) = 2^{c_{\texttt{BKZ}}\cdot\beta+o(\beta)}$ with $2^{\mathcal{O}(\beta)}$ memory due to Micciancio-Voulgaris [MV10] and Aggarwal et al. [ADRS15] (for the former, $c_{\texttt{BKZ}} = 2$, and for the latter $c_{\texttt{BKZ}} = 1$; the space complexity is $2^{\beta+o(\beta)}$ for both algorithms).

From now on, we ignore the $o(\cdot)$-terms in the exponents. The above two cases result into two statements for the complexity of the BDD attack. We start with Kannan's SVP. We make a distinction between enumerations that achieve a constant success probability (Lindner-Peikert NearestPlanes, Spherical or Linear-Length Pruning) and those that have an arbitrarily small success probability (Babai's NearestPlane, Extreme Pruning). As in the previous sections, we assume the LWE error follows a continuous Gaussian distribution with parameter αq.

Theorem 11 (Super-exponential BDD-attack on LWE)**.** *The LWE problem with parameters $(n, q = \mathcal{O}(n^{\mathsf{c}_q}),\ \alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha}))$ where $\mathsf{c}_q, \mathsf{c}_\alpha = \Theta(1)$, can be solved via BDD with (1) a $\beta = \Theta(n)$-basis reduction running in time $2^{c_{\texttt{BKZ}}\beta\log\beta}$ and (2) any enumeration algorithm $\texttt{Enum} \in \{\texttt{GenPruning}, \texttt{NearestPlanes}\}$ using the optimal choice of $m = \left(\frac{2\mathsf{c}_q}{\sqrt{2c_{\texttt{BKZ}}}+\mathsf{c}_\alpha} + o(1)\right)\cdot n$ samples in time*

$$T(\mathsf{BDD}) = 2^{\left(c_{\texttt{BKZ}}\cdot\frac{2\mathsf{c}_q}{(\sqrt{2c_{\texttt{BKZ}}}+\mathsf{c}_\alpha)^2}\right)\cdot n\log n},$$

if $P_{succ}(\texttt{Enum}) = 1 - o(1)$. For arbitrary $P_{succ}(\texttt{Enum})$, the above quantity is a lower bound for $\rho(\mathsf{BDD})$.

Proof. Assume we run an enumeration algorithm with $P_{\mathrm{succ}}(\texttt{Enum}) = 1 - o(1)$. Investing more time in the reduction (i.e. improving the quality of the output basis) results in a faster enumeration, so the total running time $T(\mathsf{BDD}) = T(\texttt{BKZ}) + T(\texttt{Enum})$ is minimized if the two phases are balanced. Thms. 6 resp. 10 give $T(\texttt{NearestPlanes})$ resp. $T(\texttt{GenPruning})$. On the logarithmic scale, the balancing condition $T(\texttt{BKZ}) = T(\texttt{Enum})$ is equivalent to (omitting the $o(1)$-terms)

$$\frac{1}{2}\left(\frac{m}{2\beta} + \frac{n}{m}\mathsf{c}_q - \mathsf{c}_\alpha\right)^2 \beta\log\beta = c_{\texttt{BKZ}}\beta\log\beta.$$

This equation yields $\beta = \frac{1}{2}\frac{m}{\sqrt{2c_{\texttt{BKZ}}}-(n/m)\mathsf{c}_q+\mathsf{c}_\alpha}$. This expression attains its minimum at $m = \frac{2\mathsf{c}_q}{\sqrt{2c_{\texttt{BKZ}}}+\mathsf{c}_\alpha}$, from where the first theorem statement easily follows. Note that for a constant success probability of enumeration, $\rho(\mathsf{BDD}) = \Theta(T(\mathsf{BDD}))$.

For arbitrary $P_{\text{succ}}(\texttt{Enum})$, we have $\rho(\mathsf{BDD}) = \frac{T(\texttt{BKZ})}{P_{\text{succ}}(\texttt{Enum})} + \rho(\texttt{Enum}) \geq T(\texttt{BKZ}) + \rho(\texttt{Enum})$, and we have just computed $T(\texttt{BKZ}) + \rho(\texttt{Enum})$. □

Now we consider the case when a lattice-basis reduction has a *single* exponential complexity $2^{c_{\text{BKZ}} \cdot \beta}$. Recall Thm. 10 shows that for $\texttt{Enum} = \texttt{GenPruning}$, $\rho(\texttt{Enum}) = 2^{\Theta(\beta \log \beta)}$. Cor. 7 states the same trade-off for the Babai's `NearestPlane` and Lindner-Piekert `NearestPlanes` algorithms.

So asymptotically, it is optimal to reduce the input basis to a point where the cost of enumeration switches from super-exponential to polynomial since in this case, $T(\mathsf{BDD})$ is dominated by the reduction and stays singe-exponential. In the next theorem, we find a value of β for which the reduction phase will produce a basis required for a poly(m)-time enumeration that will produce the correct output with success probability almost 1. This enumeration may be either the Babai's `NearestPlane`, Lindner-Peikert `NearestPlanes` or any (reasonable) pruning strategy with poly(m) number of nodes.

Theorem 12 (Single-exponential BDD-attack on LWE)**.** *The LWE problem with parameters (n, $q = \mathcal{O}(n^{\mathsf{c}_q})$, $\alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha})$) where $\mathsf{c}_q, \mathsf{c}_\alpha = \Theta(1)$ can be solved via BDD with (1) a $\beta = \left(\frac{2\mathsf{c}_q}{\mathsf{c}_\alpha^2} + o(1)\right) \cdot n$-basis reduction running in single-exponential time $2^{c_{\text{BKZ}}\beta}$ and (2) poly(m)-time enumeration algorithm using the optimal choice $m = \left(\frac{2\mathsf{c}_q}{\mathsf{c}_\alpha} + o(1)\right) \cdot n$ of samples in time*

$$T(\mathit{BDD}) = 2^{\left(c_{\text{BKZ}} \cdot \frac{2\mathsf{c}_q}{\mathsf{c}_\alpha^2} + o(1)\right) \cdot n}$$

with $P_{\mathit{succ}}(\mathit{BDD}) = 1 - o(1)$.

Proof. To guarantee a constant success probability and polynomial running time for enumeration, we need to ensure that the last Gram-Schmidt vector of the basis returned by β-reduction satisfies $\|\vec{b}_m^*\| > \alpha q$. This is equivalent to (cf. Cor. 7) $\frac{m}{2\beta} - \mathsf{c}_\alpha + \frac{n}{m}\mathsf{c}_q < 0$, so β must be set as

$$\beta > \frac{1}{2} \frac{m}{\mathsf{c}_\alpha - \mathsf{c}_q \cdot n/m}.$$

This value is minimized for $m = \frac{2\mathsf{c}_q}{\mathsf{c}_\alpha}$ yielding $\beta > \frac{2\mathsf{c}_q}{\mathsf{c}_\alpha^2} \cdot n$. The theorem statement follows directly from plugging this value in the running time of the reduction. □

This theorem concludes the discussion on the two-phase BDD attack on LWE. In the following section, we briefly discuss some other lattice-based methods to solve LWE.

3.1.5 Other lattice-based algorithms for LWE

To provide a complete picture on lattice-based attacks, we briefly describe two other algorithms one can use to solve LWE. First is the so-called Kannan's homogenization technique [Kan87], which allows us to convert a CVP (resp. BDD) instance to an SVP (resp. uSVP_γ) instance in a higher-dimensional lattice (the approximation parameter γ in uSVP_γ depends on the promise given in the original BDD instance). This approach is know as the *embedding* technique. We show the running time of this attack when applied to LWE in Thm. 13. An analogous result was presented in [APS15] but our choice of m is different.

The second approach we consider, the so-called *dual* attack, tackles the *decisional*-LWE (see Def. 3) by solving appSVP_γ (for an appropriate choice of γ) in the lattice dual to the LWE lattice.

Embedding. Assume a CVP instance $(\mathcal{L}(B), \vec{t})$ has a solution $\vec{v} \in \mathcal{L}(B)$. Consider a higher-dimensional lattice $\mathcal{L}_{\text{Embed}}(B, \vec{t})$ generated by the columns of the matrix

$$B' = \begin{pmatrix} B & \vec{t} \\ 0 & \tau \end{pmatrix}, \tag{3.7}$$

where τ, known as the *embedding factor*, is chosen such that the shortest vector in $\mathcal{L}_{\text{Embed}}(B, \vec{t})$ is of the form $(\vec{v}, e)$ for $e \neq 0$. In other words, solving SVP on $\mathcal{L}_{\text{Embed}}(B, \vec{t})$ leads to a solution of the original CVP problem. Note that τ should not be too small (in the worst-case CVP instance, τ is $\frac{1}{2}\lambda_1(\mathcal{L}(B))$), otherwise the last column of the matrix B' may be used too often and the returned shortest vector may be the closest to a multiple of $\vec{t}$. On the other hand, τ should not be too large, otherwise the returned shortest vector will lie in Span(B) giving no information on the solution to CVP.

The embedding technique becomes more powerful if we know some information on the CVP instance we are given. In the BDD problem, for instance, we have a promise that the target vector is much closer to the solution-vector $\vec{v}$ than to any other lattice-vector. In this case, Kannan's technique leads us to the uSVP_γ problem. In case of LWE, we know that $\|\vec{t} - \vec{v}\| = \|\vec{e}\| = \Theta(\alpha q\sqrt{m})$, and, further, we know $\lambda_1(B)$. It allows us to estimate λ_1 and λ_2 in $\mathcal{L}_{\text{Embed}}(B, \vec{t})$ as $\lambda_1^2 = \|\vec{e}\|^2 + \tau^2$, $\lambda_2 = \lambda_1(B)$, and hence, we know the gap of the lattice $\mathcal{L}_{\text{Embed}}$. This gives us a bound on parameter $\gamma = \frac{\lambda_2}{\lambda_1}$ in the uSVP_γ problem we are solving.

We solve the uSVP_γ problem via lattice-basis reduction. From Eq. (2.3), we know that an m-dimensional β-BKZ reduced lattice, gives a $\beta^{\frac{m}{2\beta}}$ approximation to a shortest vector of the lattice. Hence, as soon as the first (the shortest) vector of the reduced basis satisfies $\|\vec{b}_1\| < \lambda_2(\mathcal{L}_{\text{Embed}}(B, \vec{t}))$, this vector is the shortest in $\mathcal{L}_{\text{Embed}}(B, \vec{t})$ and, consequently, is the solution to uSVP_γ. All that remains is to determine the BKZ parameter β for which the first vector meets the requirement.

In the theorem below, we consider the two possible running times of BKZ, super-exponential (resp. single-exponential) as $f(\beta) = \beta \log \beta + o(\beta \log \beta)$ (resp. $f(\beta) = \beta + o(\beta)$). We omit the $o(\cdot)$-terms. The space complexity of the attack is polynomial in the first case and exponential in β in the second.

Theorem 13 (Complexity of the Embedding Attack on LWE)**.** *The LWE problem with parameters* $(n, q = \mathcal{O}(n^{\mathsf{c}_q}), \alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha}))$ *where* $\mathsf{c}_q, \mathsf{c}_\alpha = \Theta(1)$*, can be solved via* embedding *using a* β*-BKZ reduction with* $T(\textit{BKZ}) = 2^{c_{\textit{BKZ}} \cdot f(\beta)}$*, where either* $f(\beta) = \beta$*, or* $f(\beta) = \beta \log \beta$ *using the optimal choice of* $m = \left(\frac{2\mathsf{c}_q}{\mathsf{c}_\alpha} + o(1)\right) \cdot n$ *of LWE samples in time*

$$T(\textit{Embed}) = 2^{\left(c_{\textit{BKZ}} \cdot \frac{2\mathsf{c}_q}{\mathsf{c}_\alpha^2} + o(1)\right) f(n)}.$$

Proof. Let B be a matrix that arises from m LWE samples and let $\mathcal{L}_{\text{Embed}}(B, \vec{t})$ be the corresponding $m+1$-dimensional lattice generated by the matrix defined in Eq. (3.7). Let us estimate the first two successive minima λ_1, λ_2 for the embedded lattice. Setting the embedding factor $\tau = \Theta(\|\vec{e}\|)$, we obtain

$$\lambda_1^2(\mathcal{L}_{\text{Embed}}(B, \vec{t})) = \|\vec{e}\|^2 + \tau^2 = \Theta((\alpha q)^2 m).$$

For a q-ary LWE lattice, we have $\lambda_1(\mathcal{L}(B)) = \min\{q, \sqrt{m} q^{1-n/m}\}$. For our choice of m, Minkowski's bound is always smaller, leading to

$$\lambda_2(\mathcal{L}_{\text{Embed}}(B, \vec{t})) = \lambda_1(\mathcal{L}(B)) \leq \sqrt{m} q^{1-n/m}.$$

The value of β, for which the β-reduced basis achieves an approximation of $\frac{\lambda_2(\mathcal{L}_{\text{Embed}}(B,\vec{t}))}{\lambda_1(\mathcal{L}_{\text{Embed}}(B,\vec{t}))}$, is given by

$$\beta^{m/(2\beta)} = \frac{\lambda_2(\mathcal{L}_{\text{Embed}}(B,\vec{t}))}{\lambda_1(\mathcal{L}_{\text{Embed}}(B,\vec{t}))} = \Theta\left(\frac{q^{1-n/m}}{\alpha q}\right),$$

assuming Minkowski's bound holds with equality. This is equivalent to $\beta = \left(\frac{2\mathsf{c}_q}{\mathsf{c}_\alpha^2} + o(1)\right) \cdot n$. The minimum value for β is attained at $m = \left(\frac{2\mathsf{c}_q}{\mathsf{c}_\alpha} + o(1)\right) \cdot n$. □

There is no surprise that the complexity of the embedding attack is exactly the same as the complexity of the two-phase BDD attack with the single-exponential reduction followed by a polynomial-time enumeration (cf. Thm. 12). Both methods are, in fact, equivalent: performing a Babai-type (i.e. polynomial) enumeration on a reduced basis can be interpreted as embedding the target into the reduced basis and then size-reducing it (i.e. running LLL). After such a procedure, the first vector reveals the closest vector.

Dual attack, originally considered in [MR09] and further discussed in [KF15], solves the *decisional*-LWE.

Given an LWE instance $(A, \vec{t} = A^{\mathsf{t}}\vec{s} + \vec{e} \bmod q) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^m$, instead of working with the lattice $\mathcal{L}_q(A^{\mathsf{t}})$ (i.e. the image of A^{t}) as we did so far, we make use of another q-ary m-dimensional lattice – the kernel of A:

$$\mathcal{L}_q^\perp(A) = \{\vec{x} \in \mathbb{Z}^m \colon A\vec{x} = \vec{0} \bmod q\}. \tag{3.8}$$

This lattice is the scaled dual to $\mathcal{L}_q(A^{\mathsf{t}})$ with determinant $\det \mathcal{L}_q^\perp(A) = q^n$ (the duality follows from the fact that $\mathcal{L}_q^\perp(A) = q(\mathcal{L}(A^{\mathsf{t}}))^*$). A basis for this lattice can be easily formed from a (non-zero) matrix X that satisfies $AX = 0 \mod q$.

Assume we have found a short non-zero vector $\vec{v} \in \mathcal{L}_q^\perp(A)$. Computing $w = \langle \vec{v}, \vec{t}\rangle \bmod q = \vec{v}^{\,t}(A^{\mathsf{t}}\vec{s} + \vec{e}) \bmod q = \langle \vec{v}, \vec{e}\rangle \bmod q$, we can distinguish whether $\vec{e}$ is uniform or Gaussian. If $\vec{e}$ is uniform, w is also uniform, while if $\vec{e}$ is Gaussian, $w = \sum_i v_i e_i$ is Gaussian with parameter $\alpha q \cdot \|\vec{v}\|$ (again, we assume the LWE error follows a continuous Gaussian distribution). In the second case, the statistical distance between w mod q and a uniform random variable mod q or, a *bias* of a continuous Gaussian with parameter $\alpha q \cdot \|\vec{v}\|$, is $\delta = 2^{-\mathcal{O}(\alpha^2\|\vec{v}\|^2)}$. (To see this, we take a Fourier transform over $\mathbb{Z}_q$ of the Gaussian density function with standard deviation $\alpha q\|\vec{v}\|$ and evaluate at $1/q$). There exists an efficient distinguisher that has the advantage δ in deciding whether w is uniform or Gaussian. For instance, [[DTV15], Lemma 10] shows that for Gaussian w with standard deviation s over $\mathbb{Z}_q$, $\mathbb{E}[\cos(\frac{2\pi w}{q})] \geq \frac{q}{\pi}\sin\left(\frac{\pi}{q}\right) e^{-2\pi^2 s^2/q^2}$ (cf. line 4 in Alg. 4), while for a uniform $w \in \mathbb{Z}_q$, $\mathbb{E}[\cos(\frac{2\pi w}{q})] = 0$.

To keep the bias $\delta = 2^{-\mathcal{O}(\alpha^2\|\vec{v}\|^2)}$ sub-exponential, we must have $\|\vec{v}\| = \mathcal{O}(n^{1/2+\mathsf{c}_\alpha-\varepsilon})$ for any $\varepsilon > 0$. The following lemma estimates the value for β s.t. a β-BKZ reduction (now on $\mathcal{L}_q^\perp(A)$) outputs $\vec{v}$ of desired length.

Lemma 14 (Decisional LWE under the Dual attack)**.** *The* decisional-*LWE problem with parameters* (n, $q = \mathcal{O}(n^{\mathsf{c}_q})$, $\alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha})$) *where* $\mathsf{c}_q, \mathsf{c}_\alpha = \Theta(1)$*, can be solved via running a* β-BKZ *reduction on the* dual *lattice defined as in Eq. (3.8) with* $T(\textit{BKZ}) = 2^{c_{\textit{BKZ}}\cdot f(\beta)}$*, where either* $f(\beta) = \beta$*, or* $f(\beta) = \beta\log\beta$*, using the optimal choice of* $m = \left(\frac{2\mathsf{c}_q}{1/2+\mathsf{c}_\alpha} + o(1)\right)\cdot n$ *of samples in time*

$$T(\textit{Dual}) = 2^{\left(c_{\textit{BKZ}}\cdot\frac{2\mathsf{c}_q}{(1/2+\mathsf{c}_\alpha)^2}+o(1)\right)f(n)}.$$

with $P_{\textit{succ}}(\textit{Dual}) = 2^{-\mathcal{O}(n^{1-\varepsilon})}$ *for* $\varepsilon > 0$.

Algorithm 4 Dual attack on decisional-LWE $\texttt{Dual}(A, \vec{t}, \varepsilon)$

Input: $A \in \mathbb{Z}^{m\times k}, \vec{t} \in \mathbb{Z}^m$ where either (1) $\vec{t} = A^{\mathsf{t}}\vec{s} + \vec{e} \bmod q$ or (2) $\vec{t}$ is uniform from $\mathbb{Z}_q^m$
Output: "Yes" if $\vec{t} = A^{\mathsf{t}}\vec{s} + \vec{e} \bmod q$, "No" otherwise.

1: Compute X s.t. $AX = 0 \bmod q$ via Gaussian elimination
2: $B \leftarrow \beta$-BKZ $(\mathcal{L}_q(X))$ with $\beta = \left(\frac{2\mathsf{c}_q}{(1/2+\mathsf{c}_\alpha)^2} + o(1)\right)\cdot n$ ▷ Although we do not know c_α, we can approximate it by trying several $\mathsf{c}_\alpha \in [0, \mathsf{c}_q]$ in a binary-search manner
3: **if** $\exists \vec{v} \in \mathcal{L}(B)$ s.t. $\|\vec{v}\| = \mathcal{O}(n^{1/2+\mathsf{c}_\alpha-\varepsilon})$ **then**
4: **if** $\cos\left(\frac{2\pi\langle \vec{v}, \vec{t}\,\rangle}{q}\right) > \frac{q}{\pi}\sin\left(\frac{\pi}{q}\right) e^{-2\pi^2\cdot n^{1-\varepsilon}}$ **then**
5: **return** "Yes"
6: **else**
7: **return** "No"

Proof. From Eq. (2.3), the shortest vector of a β-BKZ reduced basis of $\mathcal{L}_q^\perp(A)$ satisfies $\|\vec{v}\| = \mathcal{O}\left(\beta^{\frac{m}{2\beta}} q^{\frac{n}{m}}\right)$. Since we want the bias $\delta = 2^{-\mathcal{O}(\alpha^2\|\vec{v}\|^2)}$ remain sub-exponential, the length of $\vec{v}$ should additionally satisfy $\|\vec{v}\| = \mathcal{O}(n^{1/2+\mathsf{c}_\alpha-\varepsilon})$. Letting $\varepsilon \to 0$, we choose $\beta = \left(\frac{2\mathsf{c}_q}{(1/2+\mathsf{c}_\alpha)^2} + o(1)\right)\cdot n$ and $m = \left(\frac{2\mathsf{c}_q}{1/2+\mathsf{c}_\alpha} + o(1)\right)\cdot n$. The theorem follows after substituting this β into the running time of BKZ. □

One of the remarkable properties of LWE is the equivalence between the *decisional* and *search* versions of the problem. While the direction decisional-LWE $\leq$ search-LWE is trivial, the reverse is not that immediate. But it was proved to be true in several papers starting with the original result of Regev [Reg05] for prime $q = \text{poly}(n)$ and later extended to exponentially large composite moduli [MP12].

Now assume we want to use this equivalence to turn the result of Lemma 14 into an algorithm for the search-LWE. Note that the search-to-decision reduction requires a decisional-LWE oracle that returns the correct answer (i.e. given a pair $(\vec{a}, t)$, it decides whether it is uniform or follows an LWE distribution) with success probability $1 - o(1)$. The advantage of the distinguisher from Alg. 4 is only sub-exponential: $\delta = 2^{-\mathcal{O}(n^{1-\varepsilon})}$. In order to boost the advantage to $1 - o(1)$, we have to repeat the algorithm $\text{poly}(n)\delta^{-2}$ times on independent LWE samples.[2]

In case we have sub-exponentially many LWE samples, the asymptotical complexity stated in Lemma 4 also holds for the search-LWE, as the additional sub-exponential term that comes from the search-to-decisional reduction is suppressed by the leading-order single/super-exponential term.

In a more natural scenario, when the number of LWE samples is limited to only $\text{poly}(n)$, we can resort to the so-called amplification technique aimed at creating exponentially many 'fresh looking' LWE samples out of $\text{poly}(n)$-many samples. This amplification – originally considered for the combinatorial BKW attack on LWE [ACF+15] – can also be applied to the Dual attack to generate new samples.

We now briefly describe how to generate many new samples and refer the reader for the complete proof to [HKM]. Given an LWE instance $(A, \vec{t}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^m$ with $m = \text{poly}(n)$, we sample a discrete Gaussian $\vec{x} \in \mathbb{Z}_q^m$ with parameter $\eta = \Omega(1)$ and output $(A\vec{x} \bmod q, \langle \vec{t}, \vec{x}\rangle \bmod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. This tuple serves as a new LWE sample.

The main challenge in the amplification process is to show that (1) the new $\vec{a}' = A\vec{x} \bmod q$ is uniformly distributed over $\mathbb{Z}_q^n$ and independent from the original samples, and (2) the new error e' in $\langle \vec{t}, \vec{x}\rangle \bmod q$ is independent from a' conditioned on the original samples. For a wide enough Gaussian $\vec{x}$ (i.e. taking η to be a large constant in case $m = \Theta(n \log n)$, or even $\eta = \Theta(n^{c_\eta})$ for small constant c_η in case we have only $m = \Theta(n)$ original samples), the amplification by $\vec{x}$ was shown to satisfy the

[2] More precisely, if we have found $m = \text{poly}(n)\delta^{-1}$ short enough vectors $\vec{v}_i \in \mathcal{L}_q^\perp(A_i)$, in Alg. 4 we rather compute $\frac{1}{m}\sum_i \cos\left(\frac{2\pi\langle \vec{v}_i, \vec{t}_i\rangle}{q}\right)$ and check if the result is large enough. The correctness follows from Chernoff bounds. Note that for uniform $\vec{t}_i$'s, the expected value of this sum is 0.

above conditions, [HKM].

Note that the width of the error in this new amplified LWE samples gets increased from αq to $\sqrt{m}\alpha q$, thus adding $1/2$ to c_α (in case $\eta = \Theta(1)$). Combining amplification with the result obtained for the Dual attack on decisional-LWE in Lemma 14, yields the following theorem.

Theorem 15 (Search-LWE under the Dual attack)**.** *The* search-*LWE problem with parameters (n, $q = \mathcal{O}(n^{\mathsf{c}_q})$, $\alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha})$) where $\mathsf{c}_q, \mathsf{c}_\alpha = \Theta(1)$ and m LWE samples, can be solved via running β-BKZ reduction on the* dual *lattice defined in Eq. (3.8) with $T(\mathit{BKZ}) = 2^{c_{\mathtt{BKZ}} \cdot f(\beta)}$, where either $f(\beta) = \beta$, or $f(\beta) = \beta \log \beta$, in time*

$$T(\mathit{Dual}) = \begin{cases} 2^{\left(c_{\mathtt{BKZ}} \cdot \frac{2\mathsf{c}_q}{(1/2+\mathsf{c}_\alpha)^2} + o(1)\right) \cdot f(n)} & \text{if } m = 2^{\mathcal{O}(n)} \\ 2^{\left(c_{\mathtt{BKZ}} \cdot \frac{2\mathsf{c}_q}{\mathsf{c}_\alpha^2} + o(1)\right) \cdot f(n)} & \text{if } m = \Omega(n \log n). \end{cases}$$

with $P_{succ}(\mathit{Dual}) = 1 - o(1)$.

The memory complexity of the attack is determined by the β-BKZ reduction. Note that we do not have to store all the $2^{\mathcal{O}(n)}$ LWE samples required by the decision-to-search reduction, but rather access (or amplify) them once needed.

3.1.6 Summary of the results

In this section we summarize all the above results into Table 3.1 and Fig. 3.3. To make the overview complete, we include the results on asymptotic analysis of BKW algorithm [ACF+15, KF15, HKM] and the linearization attack by Arora-Ge [AG11].

The most important part of the table is the middle-column showing the constants in the exponents of the algorithms' runtimes. These constants are functions of LWE parameters $\mathsf{c}_q, \mathsf{c}_\alpha = \Theta(1)$ and the constant $c_{\mathtt{BKZ}}$ – the exponent of the running time of SVP solver called within BKZ.

The upper part of the table states the complexities of attacks when only poly(n) memory is available. Only lattice-based attacks are applicable in this setting. In this part of the table, running times of all the algorithms are of order $2^{\Theta(n \log n)}$. The reader should not be confused with the exponential number of samples and polynomial memory for the Dual attack: as explained in Sect. 3.1.5, we do not have to store all the samples at once. Since for Kannan's enumeration we have the term $\sqrt{2c_{\mathtt{BKZ}}} = \sqrt{1/e} \approx 0.42$ in the denominator, BKZ +Enum is faster than Dual or Embed algorithms when the number of samples is limited. The Dual attack, however, is asymptotically faster with $2^{\Theta(n)}$ samples: the additive term $1/2$ is slightly larger than 0.42.

In the lower part of the table, exponential space-complexity is allowed resulting in single-exponential running times for the attacks. In this case, *all* lattice-based attacks (BKZ+Enum, Dual, Embed) have the *same* constants provided we can access only poly(n) LWE samples. Exponential memory complexity comes from the lattice-basis reduction. The Dual attack, unlike other *lattice-based* algorithms, can profit from exponential number of samples: the constant gets improved by the additive term of $1/2$. Since the Dual attack needs exponentially many samples, this $1/2$ term vanishes when we have a limited number of samples, in which case we have to run the amplification process. Recall that as the result of the amplification, a 'fresh' LWE sample of the form $(\vec{a}', t')$ is produced, where $(\vec{a}' = A\vec{x} \bmod q, t' = \langle \vec{t}, \vec{x} \rangle \bmod q)$ for some constant-width Gaussian $\vec{x} \in \mathbb{Z}_q^m$. It is easy to verify that in case of $\Theta(n \log n)$ samples, the noise-rate in t' increases exactly by $1/2$, thus making the whole attack slightly worse.

In case the number of samples is only linear in n, i.e. $m = \mathsf{c}_m n$ for some constant c_m, the amplification works only for certain range of parameters $\mathsf{c}_m, \mathsf{c}_q, \mathsf{c}_\alpha$. This is due to the fact that we amplify the samples with a Gaussian $\vec{x}$ of a *non-constant* width of order $\Theta(n^a)$ for some $a = \Theta(1)$. In case $\mathsf{c}_\alpha < 1/2 + \mathsf{c}_q/\mathsf{c}_m$, or in other words, the noise in the original samples is too large relative to the

modulus q, the standard deviation of the amplified samples will become larger than q. We refer the reader to [HKM, Lemma 10] for the details.

The same situation happens for the combinatorial BKW algorithms, since we can again use amplification to produce enough samples for the algorithm to work. As opposed to the Dual attack, BKW works only with exponential memory at disposal as it has to store many samples at once. Further, note that the denominators of the exponents for lattice-based attacks and for BKW are the same except they are squared for the former. As a consequence, as soon as c_α (or $\mathsf{c}_\alpha + 1/2$ for $2^{\Theta(n)}$ many samples) is greater than 1, lattice-based techniques will outperform BKW. Essentially it means that lattice-based attacks are better for low-noise rates.

Yet this is not always the case if we compare lattice-based attacks with the recent improvement for the BKW algorithm by Kirchner-Fouque [KF15] and Guo et al. [GJS15], which we name BKW2. Its constant $\left(1/\mathsf{c}_q + 2\ln(\frac{\mathsf{c}_q}{\mathsf{c}_q - \mathsf{c}_\alpha})\right)$ (in case of $2^{\Theta(n)}$ many samples) is always smaller than the constant $\frac{\mathsf{c}_q}{\mathsf{c}_\alpha + 1/2}$ of BKW, but not necessarily smaller than the constant for the lattice-based attacks. It hugely depends on the value $c_{\mathtt{BKZ}}$ that obviously impacts their complexity. How exactly these all constants compare with each other is illustrated in Fig. 3.3.

In the upper figure we compare single-exponential attacks with *polynomial* number of LWE samples. If we order the algorithms relative to the value of their constant for the parameters $(\mathsf{c}_q, \mathsf{c}_\alpha)$, differently coloured areas correspond to different orderings. Since the constants for lattice-based attacks heavily rely on the value $c_{\mathtt{BKZ}}$, we distinguish the two cases: (1) $c_{\mathtt{BKZ}} = 1$ (Aggarwal et al. [ADRS15] provable instantiation of an SVP solver), and (2) $c_{\mathtt{BKZ}} = 0.292$ (heuristic SVP algorithm from [BDGL16]). The subscript in the name of lattice-based attack indicates which case is chosen.

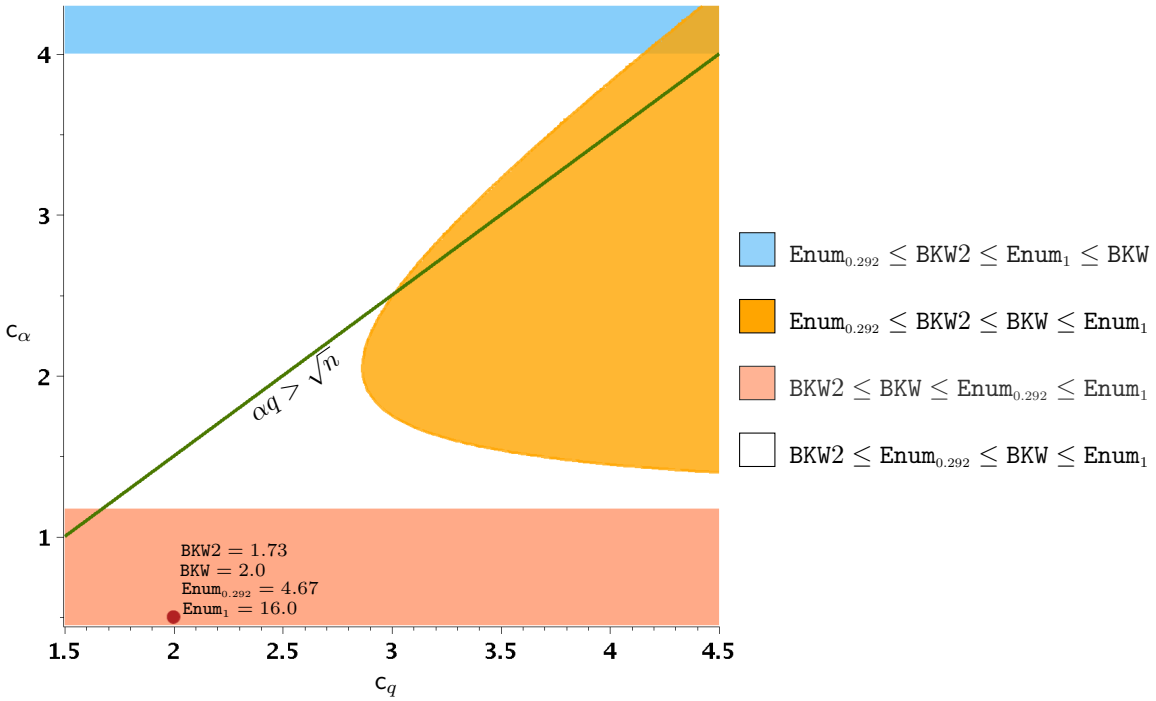
For example, the orange area marks the range of LWE parameters $(\mathsf{c}_q, \mathsf{c}_\alpha)$ where the BKZ + Enum attack with $c_{\mathtt{BKZ}} = 0.292$ performs better than BKW2. For small values of c_α, however, BKW algorithms outperform lattice-based techniques. This is due to the fact that combinatorial attacks are more robust to large-noise instances (the smaller c_α, the larger Gaussian width $\alpha q = n^{\mathsf{c}_q - \mathsf{c}_\alpha}$), while lattice-based algorithms seem to perform significantly worse for an increased error-rate. We note that in case of poly(n) samples, by Enum we mean all the lattice-based attacks as they have the same constant.

We present the same figure for the case of *exponential* number of samples. Here we take only the Dual algorithm to compare with BKW. Notice that the horizontal areas (blue at the top and pink at the bottom) get shifted by $1/2$ comparatively to the upper figure – the gain we have in the exponents for BKW and Dual algorithms once we have exponentially many samples.

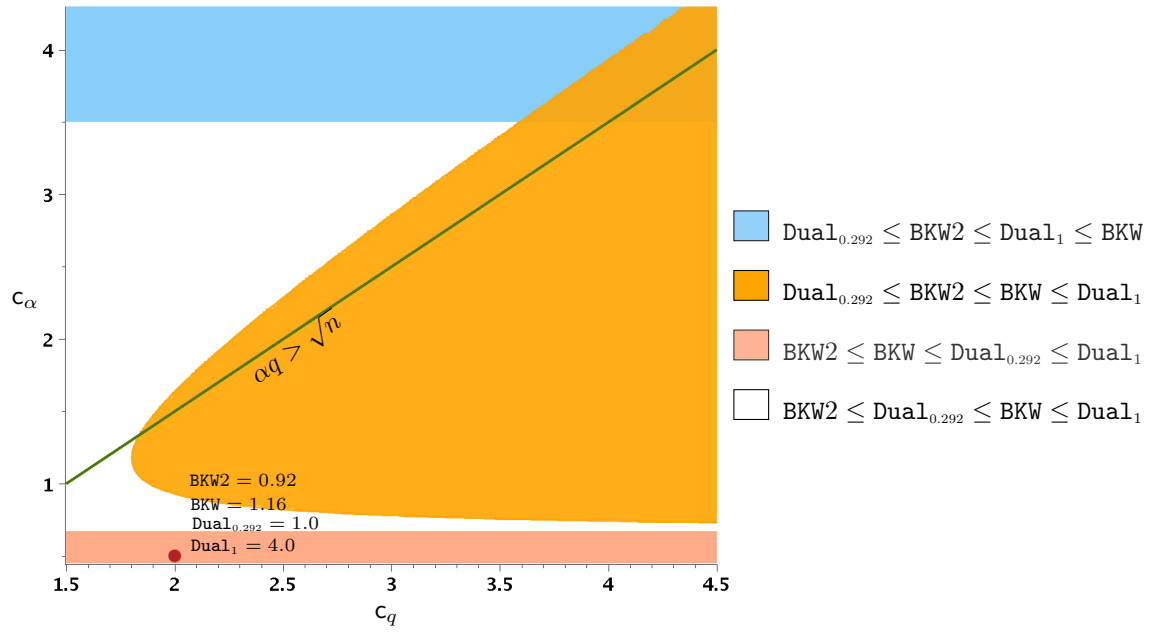
In both figures, the area below the green line denotes the values of $\mathsf{c}_q, \mathsf{c}_\alpha$ for which hardness reductions (both classical and quantum) hold. As an example, for parameters $(\mathsf{c}_q = 2, \mathsf{c}_\alpha = 0.5)$ – the values chosen for the cryptosystem described in [Reg05] – we make the constants explicit.

$\rho(\texttt{ALG}) = \frac{T(\texttt{ALG})}{P_{\text{succ}}(\texttt{ALG})}$, $M = \text{Space}$		
polynomial memory	$M = \text{poly}(n), T(\texttt{BKZ}) = 2^{c_{\texttt{BKZ}} n \log n}$	
	$\log(\rho)/(n \log n)$	# Samples
LATTICE-BASED ALGORITHMS		
BKZ + Enum, where Enum is: – Babai (Sect. 3.1.1) – Lindner-Peikert (Sect. 3.1.2) – GenPruning (Sect. 3.1.3)	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{(\sqrt{2c_{\texttt{BKZ}}} + \mathsf{c}_\alpha)^2}$	$\Theta(n)$
Dual (Sect. 3.1.5, Thm. 15)	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{\mathsf{c}_\alpha^2}$	$\Theta(n \log n)$
	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{(\mathsf{c}_\alpha + 1/2)^2}$	$2^{\Theta(n)}$
Embedding (Sect. 3.1.5, Thm. 13)	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{\mathsf{c}_\alpha^2}$	$\Theta(n)$
exponential memory	$M = 2^{\Theta(n)}, T(\texttt{BKZ}) = 2^{c_{\texttt{BKZ}} n}$	
	$\log(\rho)/n$	# Samples
BKZ + Enum, where Enum is: – Babai (Sect. 3.1.1) – Lindner-Peikert (Sect. 3.1.2) – GenPruning (Sect. 3.1.3)	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{\mathsf{c}_\alpha^2}$	$\Theta(n)$
Dual (Sect. 3.1.5, Thm. 15), [HKM]	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{(\mathsf{c}_\alpha + 1/2)^2}$	$2^{\Theta(n)}$
	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{\mathsf{c}_\alpha^2}$	$\Theta(n \log n)$
	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{(\mathsf{c}_\alpha - \mathsf{c}_q/\mathsf{c}_m)^2}$	$(\mathsf{c}_m + o(1))n$
Embedding (Sect. 3.1.5, Thm. 13)	$\frac{2c_{\texttt{BKZ}} \cdot \mathsf{c}_q}{\mathsf{c}_\alpha^2}$	$\Theta(n)$
COMBINATORIAL ALGORITHMS		
BKW ([ACF+15])	$\frac{1}{2}\frac{\mathsf{c}_q}{\mathsf{c}_\alpha + 1/2}$	$2^{\Theta(n)}$
BKW (Thm. 8 in [HKM])	$\frac{1}{2}\frac{\mathsf{c}_q}{\mathsf{c}_\alpha}$	$\Theta(n \log n)$
	$\frac{1}{2}\frac{\mathsf{c}_q}{\mathsf{c}_\alpha - \mathsf{c}_q/\mathsf{c}_m}$	$(\mathsf{c}_m + o(1))n$
BKW2 ([GJS15, KF15])	$\left(1/\mathsf{c}_q + 2\ln(\frac{\mathsf{c}_q}{\mathsf{c}_q - \mathsf{c}_\alpha})\right)^{-1}$	$2^{\Theta(n)}$
BKW2 (Thm. 8 in [HKM])	$\left(2\ln(\frac{\mathsf{c}_q}{\mathsf{c}_q - \mathsf{c}_\alpha})\right)^{-1}$	$\Theta(n \log n)$
	$\left(2\ln(\frac{\mathsf{c}_q - \mathsf{c}_q/(\mathsf{c}_m - 1)}{\mathsf{c}_q - \mathsf{c}_\alpha})\right)^{-1}$	$(\mathsf{c}_m + o(1))n$
Arora-Ge ([APS15, AG11]), $\mathsf{c}_q - \mathsf{c}_\alpha < 1/2$	$\omega \cdot (1 - 2\mathsf{c}_\alpha) \cdot n^{2\mathsf{c}_\alpha} \log^2(n)$	$\mathcal{O}(2^{n^{\mathsf{c}_\alpha}} \log^2 n)$
Arora-Ge ([APS15, AG11]), $\mathsf{c}_q - \mathsf{c}_\alpha \geq 1/2$	$\omega \cdot (2\mathsf{c}_\alpha - 1) \cdot n \log(n)$	$\mathcal{O}(2^{n \log n})$

Tab. 3.1: Asymptotic comparison of algorithms for LWE. We denote $q = \mathcal{O}(n^{\mathsf{c}_q})$, $\alpha = \mathcal{O}(1/n^{\mathsf{c}_\alpha})$ and $c_{\texttt{BKZ}}$ is the constant hidden in the run-time exponent of lattice-basis reduction. In order to run the BKW (resp. BKW 2) algorithm when only *linear* number of samples is available, the LWE parameters must satisfy $\mathsf{c}_\alpha > 1/2 + \mathsf{c}_q/\mathsf{c}_m$ (resp. $\mathsf{c}_\alpha > 1/2 + \mathsf{c}_q/(\mathsf{c}_m - 1)$) and $\mathsf{c}_q > \mathsf{c}_q/\mathsf{c}_m - 1/2$ (resp. $\mathsf{c}_q > \mathsf{c}_q/(\mathsf{c}_m - 1) - 1/2$). Such constraints come from the amplification. For Arora-Ge, $2 \leq \omega < 3$ is the linear algebra constant.



(a) Comparison of *single*-exponential attacks with *polynomial* number of samples relative to the (c_q, c_α) parameters. The orange area illustrates the parameter-ranges where a BKZ-reduction instantiated with heuristic SVP-solver (i.e. $c_{\text{BKZ}} = 0.292$) followed by an enumeration algorithm Enum beats the BKW2 algorithm from [GJS15, KF15]. For large values of c_α depicted blue (i.e. small noise-rate), the enumeration algorithms are better than the combinatorial BKW. On the other hand, BKW performs better for higher-noise rates (pink area). The red dot denotes LWE parameters considered in [Reg05]: $c_q = 2, c_\alpha = 0.5$. The area below the green line corresponds to the parameters (c_q, c_α) for which the hardness reductions hold.



(b) Same as above but for *exponential* number of LWE samples.

Fig. 3.3: Asymptotical behaviour of various algorithms for LWE: lattice-based Enumeration or Dual attacks vs. combinatorial BKW.

3.2 Practical Hardness of LWE

In this section we leave asymptotics and draw our attention to the practical hardness of the Learning with Errors problem.

The results of the previous section tell us that the two-phase BKZ + Enum approach to solve LWE in poly(n)-memory regime performs better than Dual or Embedding attacks when only $\Theta(n)$ samples are provided. Moreover, it is only slightly worse than the Dual algorithm when the latter can access exponentially many samples. So in the most realistic scenario – poly(n) memory, limited number of samples – BKZ reduction followed by enumeration is the right strategy.

We already saw in Thm. 11 that in the two-phase algorithm BKZ+Enum, both steps have complexities of order $2^{\Theta(n \log n)}$ since the BKZ parameter β optimizes the attack when $\beta = \Theta(n)$, where we made the constant for β explicit.

These arguments are well-suited to conclude on the asymptotics. On the practical side, however, the BKZ algorithm is notoriously hard to implement and until very recently[3], the only available implementation of lattice-basis reduction was provided in Shoup's NTL library [Sho] and most of the complexity benchmarks ([APS15, MR09, NR06]) were obtained by running this implementation. During the execution, the BKZ algorithm in NTL calls Fincke-Pohst enumeration [FP83] as an SVP solver. The running time of this enumeration procedure is of order $2^{\mathcal{O}(\beta^2)}$, thus resulting in much worse complexity for the reduction than theory suggests.

So it is reasonable to try to shift the workload of our BDD attack from the reduction to the enumeration phase. This approach is even more advantageous once we notice that the enumeration algorithm – a tree-traversal routine – is amenable to efficient parallelization.

In this section, we present the real running times of the BKZ +Enum attack on LWE with our parallelized implementation of the enumeration step.[4] The experiments are carried out in combination with the BKZ algorithm from the NTL library. We note that, to perform the attack, one can use any other BKZ implementation to preprocess a basis and then run our enumeration algorithm.

From our experimental results we draw two main conclusions: (1) the BDD enumeration algorithms described in Sect. 3.1 can be almost perfectly parallelized by splitting the enumeration tree into subtress and traversing the sub-trees in parallel; (2) the combinatorial BKW-type attacks ([GJS15, KF15]) are *not* better in practice than the lattice-based attacks even for parameters favorable for the former (e.g. small or even binary secret). We emphasize on *practical* superiority of the lattice-based methods over combinatorial despite the fact that asymptotics might present a different picture (cf. Table 3.1, Fig. 3.3).

The roadmap of this subsection is as follows. First, we describe a single-threaded tree-traversal enumeration algorithm. Next, we show how to distribute the traversal of sub-trees among several threads to execute it in parallel. We discuss certain tweaks of the BDD attack one can apply to variants of LWE. At the end, we present complexities of real-time attacks on concrete LWE instances (see Table 3.5).

3.2.1 Single threaded implementation

Here we give an alternative representation of the pruning algorithm GenPruning (Alg. 3) suitable for efficient implementation. Recall that a BDD enumeration algorithm for LWE with parameters (n, α, q) receives as input a β-reduced lattice-basis $B \in \mathbb{Z}_q^{m \times m}$ and a target $\vec{t} \in \mathbb{Z}_q^m$ with a promise that $\vec{t}$ is only $\Theta(\alpha q \sqrt{m})$ away from a vector $\vec{v} \in \mathcal{L}_q(B)$ we search for. In addition, the algorithm is provided

[3] On the 22.09.16, Albrecht et al. announced [DT16] the release of the BKZ 2.0 algorithm, which asymptotically meets the desired $2^{\mathcal{O}(n \log n)}$ complexity.

[4] The code is available on-line: https://github.com/pfasante/cvp-enum

with a description of a bounding function $\mathfrak{B}$ which is used to prune the enumeration tree (see examples of $\mathfrak{B}$ in Sect. 3.1.3).

Algorithm `GenPruningDepthFirst` (Alg. 5) is a *depth-first* description of Alg. 3 from the previous section. It constructs an enumeration tree where a k-level node stores (1) a target vector $\vec{t}^{(k)}$, (2) a coefficient-vector $\vec{c}$ of a candidate-solution $\vec{x}^{(k)} = \sum_{i=k+1}^{m} c_i^{(k)} \vec{b}_i$ ($\vec{c}$ is constructed starting with its m^{th} coordinate c_m down to c_1), and (3) an accumulated *error-length* $e^{(k)} = \sum_{i=k+1}^{m} e_i'^2 \|\vec{b}_i^*\|^2$, where $\vec{e}^{(k)} = \sum_{i=k+1}^{m} e_i' \vec{b}_i^*$ is the error accumulated by a node on level k. On the root we have $k = m, e^{(m)} = 0, t^{(m)} = \vec{t}$. The leaves ($k = 0$) give candidate-solutions $\vec{x} = \sum_{i=1}^{m} c_i \vec{b}_i$ with error-length $e^{(0)} = \|\vec{t} - \vec{x}\|$. Different paths have different coefficient-vectors $\vec{c}$. Depth-first traversing is memory-efficient (as opposed to the recursive version given in Alg. 3) since we consider only one path $\vec{c}$ at a time and decide whether the corresponding error is smaller than the previously found or not.

Note that instead of keeping the coordinates of a partial error-vector as in Alg. 3, we store only its length. We do so by observing that for bounding functions $\mathfrak{B}$ of our interest (like Length Pruning), we only need the error-*length* but not its individual coordinates to evaluate $\mathfrak{B}$. So for the algorithm `GenPruningDepthFirst` we simplify the definition of a bounding function and consider only functions $\mathfrak{B} : \mathbb{Q}_{\geq 0} \to \mathbb{Q}_{\geq 0}$ that take a squared error-length as input and output the remaining allowed length. From the value $\mathfrak{B}(e^{(k)})$, we compute the number of children for a node with the (squared) error-length $e^{(k)}$ (line 7), and all the relevant information for its left-most child (lines 11–13). From this left-most child we go down-left again. Once a leaf is reached, we compare its error-length $e^{(0)}$ with the error minLen of the best (i.e. the shortest) solution found so far. In case $e^{(0)}$ is smaller than minLen, a new candidate-solution is constructed from the coefficient vector $\vec{c}$ of the current path (line 17). At the end, the returned solution has the minimal error-length among all the solutions considered by the algorithm.

The algorithm described above traverses the enumeration tree in (depth-first) *left-most* child manner (on line 11, we start with $c_{\min}$ that represents the left-most child). This 'classical' traversal is depicted in Fig. 3.2a. In the actual implementation, instead of choosing the left-most child and traversing its sub-tree, we first visit the child that gives the shortest error (i.e. the one that would have been chosen by Babai's algorithm). Then the sub-tree of this most promising 'middle' child is traversed. See Fig. 3.2b for this tree-traversing strategy.

Further, once we reach the 'critical' level k^* determined by the maximal k s.t. $\|\vec{b}_k^*\| > \mathsf{c}\alpha q$ (for some input constant c), we consider only one child for all levels below k^*. This additional pruning conforms to the Condition 3 of reasonable pruning (see Def. 8): once the Gram-Schmidt vectors are long enough and the solution has 'survived' until this level (i.e. there exist a path $\vec{c}$ that contains the coefficients of the solution), we can run the efficient (one child-only) Babai's algorithm.

Obviously, it makes sense to make the enumeration tree 'bushier' on the levels where the $\vec{b}_k^*$'s are relatively short. This is controlled by the function $\mathfrak{B}$. In our implementation, it is the linear length pruning function with an additional parameter that controls how wide the tree is allowed to be.

3.2.2 Parallel implementation

In Alg. 5, sub-tree traversals for two different nodes on the same level are independent, so we can parallelize the algorithm. Let #NThreads be the number of threads (processors) at our disposal. Our goal is to determine the upper-most level k that has at least as many nodes $\#\mathrm{N}(k)$ as #NThreads. Then we can traverse the $\#\mathrm{N}(k)$ sub-trees in parallel by calling Alg. 5 on each thread.

We start traversing the enumeration tree in a *breadth-first* manner using a queue. In a breadth-first traversal, once all the nodes of level k are visited, the queue contains all their children (i.e. all the nodes of level $k+1$), thus their number $\#\mathrm{N}(k+1)$ can be computed from the size of the queue. Once a level k with $\#\mathrm{N}(k) \geq \mathsf{c} \cdot$ #NThreads for some constant $\mathsf{c} \geq 1$ is found, we stop the breadth-first traversal and start Alg. 5 for each of the $\#\mathrm{N}(k)$ sub-trees separately on each thread. The benefit of having $\mathsf{c} > 1$ is

Algorithm 5 `GenPruningDepthFirst`$(B, \vec{t}, \mathfrak{B}^{(k)})$

Input: $B = (\vec{b}_1, \ldots, \vec{b}_m) \in \mathbb{Z}^{m \times m}, \vec{t} \in \mathbb{Z}^m$, a family of bounding functions $\mathfrak{B}^{(k)} : \mathbb{Q} \to \mathbb{Q}$
Output: $\vec{x} \in \mathcal{L}(B)$ close to $\vec{t}$ and $e = \|\vec{e}\| = \|\vec{t} - \vec{v}\|$

1: $\vec{t}^{(m)} \leftarrow \vec{t}, e^{(m)} \leftarrow 0, k \leftarrow m$.
2: Let $\widetilde{B} \leftarrow \mathrm{GSO}(B)$
3: **if** $m = 0$ **then return** $(\vec{t}^{(m)}, e)$
4: $(\vec{t}^{(0)}, \mathrm{minLen}) \leftarrow$ `NearestPlane`$(B, \vec{t})$
5: **while** (true) **do**
6: **if** $(k > 0)$ **then**
7: $Int \leftarrow \sqrt{\mathfrak{B}^{(k)}(e^{(k)})}/\|\vec{b}_k^*\|$ ▷ Number of children
8: $c^* \leftarrow \langle \vec{t}^{(k)}, \vec{b}_k^* \rangle / |\vec{b}_k^*|^2$
9: $c_{\min} \leftarrow \lceil c^* - \frac{1}{2} Int \rceil$ ▷ Left-most child
10: $c_{\max} \leftarrow \lfloor c^* + \frac{1}{2} Int \rfloor$ ▷ Right-most child
11: $\vec{c}_k \leftarrow c_{\min}$
12: $\vec{t}^{(k-1)} \leftarrow \vec{t}^{(k)} - \vec{c}_k \vec{b}_k$ ▷ Project onto $U^{(k)} = \vec{c}_k \vec{b}_k^* + \mathrm{Span}(\vec{b}_1, \ldots, \vec{b}_{k-1})$
13: $e^{(k-1)} \leftarrow e^{(k)} + (\vec{c}_k - c^*)^2 \|\vec{b}_k^*\|^2$ ▷ Compute the squared error-length
14: $k \leftarrow k - 1$ ▷ Go down the tree
15: **else** ▷ On a leaf
16: **if** $(e^{(k)} < \mathrm{minLen})$ **then**
17: $\vec{x} \leftarrow \sum_{i=1}^{k} c^{(i)} \vec{b}_i$ ▷ Current best solution
18: **repeat** ▷ Traverse up
19: **if** $(k = 0$ AND $\vec{c}_k > c_{\max})$ **then** ▷ On the root, no right siblings
20: **return** $(\vec{x}, \mathrm{minLen})$
21: $k \leftarrow k + 1$
22: **until** $(\vec{c}_k \geq c_{\max})$
23: $\vec{c}_k \leftarrow c^{(k)} + 1$ ▷ Traverse to the right sibling
24: $\vec{t}^{(k-1)} \leftarrow \vec{t}^{(k)} - \lceil \vec{c}_k \rfloor \vec{b}_k$
25: $e^{(k-1)} \leftarrow e^{(k)} + (\vec{c}_k - \lceil \vec{c}_k \rfloor)^2 \|\vec{b}_k^*\|^2$
26: **return** $(\vec{t}^{(0)}, e^{(0)})$

that whenever one of the threads finishes quickly, it can be assigned to traverse another sub-tree. This strategy compensates for imbalanced sizes of sub-trees.

This breadth-first traversal is described in Alg. 6. At the root we have $\#\mathrm{N}(m) = 1$. The associated data to each node are the target $\vec{t}^{(m-1)}$, the error-length $e^{(m-1)}$, and the partial solution $\vec{x}^{(m-1)}$. We store them in queues Q_t, Q_e, Q_x. Traversing the tree down is realized via dequeuing the first element from a queue (line 9) and enqueuing all its children into the queue. When Alg. 6 terminates, we spawn a thread that receives as input a target $\vec{t}^{(k)}$ from Q_t, an accumulated so far error-length $e^{(k)} \in Q_e$, a partial solution $\vec{x}^{(k-1)} \in Q_x$, GSO-lengths $(\|\vec{b}_{k-1}^*\|, \ldots, \|\vec{b}_1^*\|)$, and bounding functions $\mathfrak{B}^{(i)}$, $1 \leq i \leq k-1$. Since the number of possible threads is usually a small constant (30-40 on the cluster we are using), there is no blow-up in memory usage in the breadth-first traversal.

Note that for a family of bounding functions $\mathfrak{B}^{(k)}$ that allows to compute the number of children per node without actually traversing the tree, e.g. the Lindner-Peikert bounding strategy, it is easier to find the level where we start parallelization. In case of Lindner-Peikert, $\#\mathrm{N}(k) = \prod_{i=m}^{m-k} d_i$ and hence, we simply compute the largest level k where $\#\mathrm{N}(k) \geq \mathsf{c} \cdot \#\mathrm{NThreads}$.

In the implemented algorithm we slightly modify the above breadth-first traversal: before starting threads with #NThreads elements from the queue, we sort the queues Q_t, Q_e, Q_x w.r.t. the elements from Q_e s.t. the paths with shorter error-length are scheduled first. This might be implemented via priority queues or changing the container type to list and sorting the resulting list. This might speed-up the enumeration if we additionally abort the tree-traversal once we have a leaf with the error of length

$\mathsf{c} \cdot \sqrt{m}\alpha q$ for some input constant c. With this, we exploit the fact that the correct error-vector is much shorter than any other error-vector considered by the algorithm.

Algorithm 6 Traverse Breadth-First $(B, \vec{t}, \mathfrak{B}^{(k)}, \mathsf{c})$

Input: $B = (\vec{b}_1, \ldots, \vec{b}_m) \in \mathbb{Z}^{m \times m}, \vec{t} \in \mathbb{Z}^m$, a family of bounding functions $\mathfrak{B}^{(k)}$, #NThreads $\in \mathbb{Z}$, $\mathsf{c} \in \mathbb{Z}$

Output: An array $(\vec{t}^{(k)})_i$ of size $\#\mathrm{N}(k)$, where $\#\mathrm{N}(k) \geq \mathsf{c} \cdot \#\mathrm{NThreads}$, an array of associated error-length $(e^{(k)})_i$, an array of associated partial solutions $(\vec{x}^{(k)})_i$, $1 \leq i \leq \#\mathrm{N}(k)$.

Initialize queues Q_t, Q_e, Q_x
Q_t.Enqueue($\vec{t}$), Q_e.Enqueue(0), Q_x.Enqueue($\vec{0}$)
Let $\widetilde{B} \leftarrow \mathrm{GSO}(B)$
$\#\mathrm{N}(m) \leftarrow 1$
$k \leftarrow m - 1$
while $(\#\mathrm{N}(k+1) < \mathsf{c} \cdot \#\mathrm{NThreads})$ **do**
    $\#\mathrm{N}(k) \leftarrow 0$
    for $j = 1 \ldots \#\mathrm{N}(k+1)$ **do**
        $\vec{t} \leftarrow Q_t$.Dequeue(), $e \leftarrow Q_e$.Dequeue(), $\vec{x} \leftarrow Q_x$.Dequeue()
        $\#\mathrm{N}(k) \leftarrow \#\mathrm{N}(k) + \lceil \sqrt{B^{(m)}(e)}/\|\vec{b}_m^*\| \rceil$
        $c^* \leftarrow \langle \vec{t}, \vec{b}_m^* \rangle / \|\vec{b}_m^*\|^2$
        for $i = 0 \ldots \lceil \sqrt{\mathfrak{B}^{(m)}(e)}/\|\vec{b}_m^*\| \rceil - 1$ **do**
            Q_t.Enqueue($\vec{t} - \lceil c^* \pm i \rfloor \vec{b}_k$)
            Q_e.Enqueue($e + (c^* - \lceil c^* \pm i \rfloor)^2 \|\vec{b}_k^*\|^2$)
            Q_x.Enqueue($\vec{x} + \lceil c^* \pm i \rfloor \vec{b}_k$)
    $k \leftarrow k - 1$
return (Q_t, Q_e, Q_s)

3.2.3 Attacks on Variants of LWE

In [BLP+13], the classical hardness of LWE is proved via a reduction to the so-called binary-LWE, where the secret vector $\vec{s}$ is chosen from $\{0,1\}^n$. While this version of LWE is shown to be at least as hard as standard LWE in [BLP+13], the reduction loses a factor of $\log q$ in the dimension. Further, Kirchner and Fouque show in [KF15] that for binary LWE, a version of the BKW algorithm achieves slightly sub-exponential running time of order $2^{\mathcal{O}(\frac{n}{\log \log n})}$. The BDD attack can also profit from the fact that the secret is smaller than the error: Bai and Galbraith describe in [BG14] how to tweak a BDD instance for a faster attack. Our experiments confirm (see Table 3.2) that indeed for binary-LWE we can tackle considerably higher dimensions than for standard LWE. Further, the dimensions we attack are larger than those solved in [KF15] yet using much lower sample-complexity.

While binary-LWE remains to be hard and only requires to slightly increase the lattice-dimension in order to achieve the same security-level as standard LWE, other modifications to LWE might be fatal for cryptographic applications. For instance, we show that the cryptosystem based on the hardness of 'binary' matrix LWE proposed in [Gal], can be broken in several hours for relatively large dimensions using the BDD attack. We note that we attack Regev's *cryptosystem* ([Reg05]) instantiated with a binary matrix. The binary matrix LWE problem itself remains an interesting cryptanalytic target.

Binary secret LWE. To speed-up the binary-secret LWE attack, Bai and Galbraith transform a BDD instance $(\mathcal{L}_q(A^\mathsf{t}), \vec{b})$ with error $\vec{e}$ into a BDD instance

$$\left(\mathcal{L}_q^\perp(\mathbb{1}_m \mid A^\mathsf{t}), (\vec{b}, \vec{0})\right) \tag{3.9}$$

with the unbalanced error $(\vec{e}, \vec{s})$. The instance is correctly defined since

$$(\mathbb{1}_m \mid A^{\mathsf{t}}) \left[\begin{pmatrix} \vec{e} \\ \vec{s} \end{pmatrix} - \begin{pmatrix} \vec{t} \\ \vec{0} \end{pmatrix} \right] = 0 \bmod q.$$

The lattice $\mathcal{L}_q^{\perp}(\mathbb{1}_m \mid A^{\mathsf{t}}) \in \mathbb{Z}^{m+n}$ is generated by the columns of $A^{\perp}$ given by

$$A^{\perp} = \left(\begin{array}{c|c} -A^{\mathsf{t}} & q\mathbb{1}_{n+m} \\ \mathsf{c}\mathbb{1}_n & \end{array} \right),$$

where c is some input parameter that re-balances the error by increasing the determinant of the lattice. Larger determinant and hence, larger $\lambda_1(A^{\perp})$, will speed-up the BDD attack.

We run the BDD enumeration algorithm described in Alg. 5 in both single- and multi-threaded variants. The results are presented in the table below. Notice that in contrast to the BKW attack of Kirchner and Fouque [KF15], we choose as few samples as possible (while keeping a unique solution) to aid the reduction step. Concretely, we used only $m = 150$ LWE samples as opposed to 2^{28} samples required in the BKW attack. We also observe an almost perfect speed-up during a 10-threaded run on an instance of dimension 140.

LWE-parameters				BKZ-reduction		Length Pruning	
n	q	α	m	β	T	#NThreads	T
120	16411	0.001	150	10	2.3h	1	2h
130	16411	0.001	150	15	6.6h	1	1h
140	16411	0.001	170	15	12h	1	16.3h
140	16411	0.001	170	15	12h	10	1.7h

Tab. 3.2: Running times of the BDD-decoding attack on binary secret LWE

Binary matrix. To implement an LWE-based encryption on lightweight devices, [Gal] proposed not to store the whole random matrix $A \in \mathbb{Z}_q^{n \times m}$, but to generate the entries of a *binary* $A \in \mathbb{Z}_2^{n \times m}$ via some Pseudorandom Number Generator. Galbraith's ciphertexts are of the form

$$(C_1, C_2) = (A\vec{u}, \langle \vec{u}, \vec{b} \rangle + m\lceil q/2 \rceil \bmod q)$$

for a message $m \in \{0, 1\}$, some uniformly random $\vec{u} \in \{0, 1\}^m$ and a modulus $q \in \mathbb{Z}$. The task is to recover $\vec{u}$ given $(A, A\vec{u})$.

Let us describe a simple lattice attack on the instance $(A, A\vec{u})$. Notice that $C_1 = A\vec{u}$ holds over $\mathbb{Z}$ and, hence, over $\mathbb{Z}_q$ for large enough modulus q since we expect to have $A\vec{u} \approx m/4 < q$. First, we find any solution $\vec{w}$ for $A\vec{w} = C_1 \mod q$. Note that

$$(\vec{w} - \vec{u}) \in \ker(A).$$

So we have a BDD instance $(\mathcal{L}_q^{\perp}(A), \vec{w})$, with $\vec{u}$ as the error-vector of expected length $m/2$ and a lattice

with $\det(\mathcal{L}_q^\perp(A)) = q^n$. Since we can freely choose q to be as large as we want, we can guarantee that $\lambda_1(\mathcal{L}_q^\perp(A)) \gg m/2$. Such an instance can be solved by first running β-BKZ for some small constant β and then Babai's CVP algorithm.

As a challenge, Galbraith proposes a parameter-set $(n = 256, m = 400)$ and estimates that computing $\vec{u}$ from $A\vec{u}$ should take around one day. We solve this instance using NTL's BKZ implementation with $\beta = 4$ and $q = 500009$ in 4.5 hours.

LWE-parameters				BKZ-reduction	Babai's CVP
n	q	m	β	T	T
256	500009	400	4	4.5h	2min
280	500009	440	4	6.5h	3min

Tab. 3.3: Running times of the BDD-decoding attack on cryptosystem based on binary matrix LWE

3.2.4 Details on Implementation

We implemented our BDD enumeration step choosing Linear Length Pruning as a bounding strategy. All programs are written in C++ and we used C++11 STL for implementing the threading. Our tests were performed on the Ruhr University's Crypto Crunching Cluster (C3) [CCC] which consists of one master node to schedule jobs and four computing nodes. Each computing node has four AMD Bulldozer Opteron 6276 CPUs, and thus 64 cores, running at 2.3 GHz and 256 GByte of RAM. The results of our experiments are presented in Table 3.5. Let us take a closer look at this table.

All instances are split into tree categories depending on the noise-rate: the left-most have $\alpha = 0.001$, middle $\alpha = 0.002$, right-most $\alpha = 0.005$. The instances for the first two cases were generated by ourselves with modulus $q = 4093$, while for the last case, we attack the instances offered by the LWE-Challenge [LWE]. For $n = 40, 45, 50$, the moduli are $q = 1601, 2027, 2053$ respectively.

For the LWE instances with small error-rate, we took $m = 2n$ samples. For $\alpha = 0.002$, to aid the enumeration step, we slightly increase the number of samples to $m \approx 2.2n$. Notice, that for a larger m, the determinant of the LWE-lattice, $\det(\mathcal{L}_q(A^\mathsf{t})) = q^{1-n/m}$, increases. This leads to a larger $\lambda_1(\mathcal{L}_q(A^\mathsf{t}))$, making the error, from the enumeration point of view, closer to the target. Larger m explains why the running-time for BKZ with the same block-size β increases for the large noise-rates. For $\alpha = 0.005$, we use even more samples $m \approx 2.3n$. Note that theory suggests an increased m as well: in Thm. 11, the optimal choice of m was proved to be $m = \left(\frac{2\mathsf{c}_q}{\sqrt{2\mathsf{c}_{\text{BKZ}}+\mathsf{c}_\alpha}} + o(1)\right) \cdot n$. Recall that larger noise-rate corresponds to smaller c_α.

For the pruning strategy $\mathfrak{B}$, we choose the *Linear-Length* pruning function [GNR10]. This means that our tree-traversal Algorithm 5 receives on input an m-dimensional array R consisting of level-bounds $R_{m-k} = \mathsf{c}_f((m-k)(\alpha q)^2)$, where k goes from m down to 0. These bounds determine the allowed accumulated error-length per level. c_f is an additional input-constant. The larger c_f we input, the bushier the enumeration tree is and, hence, the more expensive the algorithm is.

Since we know the length of the Gram-Schmidt basis-vectors $\vec{b}^*$, we can determine the maximal level k (called 'critical' from our asymptotical analysis in Sect. 3.1), for which $\|\vec{b}_k^*\| > \mathsf{c}\alpha q$ for a constant c which we set $\mathsf{c} = 2$. From this level down, we run Babai's Algorithm 1.

From the experiments, we draw the following conclusions.

Enumeration can be perfectly parallelized. Indeed, the way we schedule the jobs for the parallel tree-traversal in Algorithm 6 allows for the speed-up equal to the number of available threads.

Recall that in Algorithm 6, we create much more sub-trees (i.e. jobs) than the number of threads #NThreads, and store the roots of these sub-trees in a queue. An additional input parameter c determines the size of this queue. In our tests, we set this parameter large enough to guarantee that the number of jobs in the queue is of order $5000 - 6000$ (for #NThreads $= 10$, it corresponds to $\mathsf{c} = 500 - 600$). For the dimensions we tackle, these numbers are larger enough to guarantee that there will many equally big sub-trees and all the threads will be evenly occupied. It is reasonable to predict that for higher dimensions and/or more threads at hand, one should choose queues of larger size.

An almost perfect speed-up was achieved for all dimensions where we run the parallelized enumeration: for $\alpha = 0.001$ and dimensions $n = 70, 80, 90$, our multi-threaded implementation allows to choose relatively small β's for the reduction and hence, to balance out the running times for the reduction and enumeration. Based on the experiments for these dimensions, for some other instances only the parallelized version was run. For example, for $n = 75$, both reduction and enumeration on 10 threads were finished in about an hour. On instances with larger noise-rates, the tests were mostly run in the multi-threaded regime as enumeration becomes significantly slower.

Binary error is significantly easier for enumeration. We also performed some tests on instances with a binary noise (this version of LWE also admits a hardness reduction, [MP13], but for a restricted number of samples $m = \mathcal{O}(n)$). For such a small error-rate, enumeration is fast, so in order to balance the attack, we choose a *smaller* m (but still large enough to guarantee the unique solution). This speeds up the reduction but slows down the enumeration. Again, we mitigate this slow-down with parallelization. In Table 3.4 below, we choose $n = 130$ as an example. Note that for this dimension, the attack runs in approximately the same time as for $n = 75$ in the Gaussian-error case with small noise-rate.

LWE-parameters			BKZ-reduction		Length Pruning	
n	q	m	β	T	#NThreads	T
130	4093	190	18	1.6e4	1	4.8e4
130	4093	190	18	1.6e4	10	6.1e3

Tab. 3.4: Running times of the BDD-decoding attack on binary LWE

An increase in the error-rate causes a substantial slow-down for the attack. Indeed, in case of the large noise-rate of $\alpha = 0.005$, the attack performs significantly worse than for a smaller rates. In order to obtain long enough Gram-Schmidt vectors for successful decoding, we have to (1) increase m, and (2) increase β. Both factors result in slower BKZ weakening the whole attack.

We would like to mention that a couple of months after the publication of [KMW16], the LWE Challenge was announced in [LWE]. Currently, the attack that tackles the hardest parameter-sets is the parallelized two-phase decoding.

n \ α	0.001				0.002				0.005			
	β	T(BKZ)	#NThreads	T(Enum)	β	T(BKZ)	#NThreads	T(Enum)	β	T(BKZ)	#NThreads	T(Enum)
40	2	1.1e2	1	5.0e1	10	2.0e2	1	2.2e2	16	9.1e2	10	3.02e2
45	3	1.2e2	1	5.1e1	12	2.3e2	1	4.5e2	19	3.2e3	10	3.5e3
50	3	1.25e2	1	7.3e1	15	6.7e2	5	7.6e2	21	1.6e4	10	2.3e4
55	5	1.5e2	1	1.4e2	17	1.7e3	10	2.1e3				
60	10	1.8e2	1	3.1e2	18	2.4e3	10	4.0e3				
65	12	2.1e2	5	5.3e2	22	9.1e3	10	7.8e3				
70	15	2.5e2	1	5.04e4	25	1.3e4	10	2.4e4				
			10	5.4e3			20	1.4e4				
75	18	3.1e3	10	3.4e3								
80	25	1.5e4	1	4.68e4								
			10	5.4e3								
85	23	1.55e5	10	2.1e5								
90	22	4.07e4	1	1.28e5								
			10	1.3e4								

Tab. 3.5: Running times (in seconds) of the enumeration attack (Alg. 5) with *Linear-Length* Pruning on LWE with parameters $(n, \alpha, q = 4093)$ for $\alpha = 0.001, 0.002$, $(n = 40, q = 1601), (n = 45, q = 2027, n = 50), q = 2503)$ for $\alpha = 0.005$.

Chapter 4

k-List Algorithms

In its most general form, the k-List problem is defined as follows:

Definition 16 (k-List Problem)**.** *Given k lists $L_1, \ldots, L_k$ of elements from a set X, the task is to find k-tuples $(x_1, \ldots, x_k) \in L_1 \times \ldots \times L_k$ that satisfy some condition C. Such a tuple $(x_1, \ldots, x_k)$ is called a solution to the k-List problem.*

Typically, the elements of the lists are iid. uniformly chosen from X, and the size of the lists L_i is exponential in the bit-length of a list-element. The number of output solutions depends on a concrete instantiation of the k-List problem: in some cases (as in Sect. 4.1), we require to output almost all solutions, while sometimes only one or constant number of solutions is enough.

In the examples of the k-List problem we consider here, a set X from where the list-elements are taken, is equipped with a metric. For instance, in case $X = \{0,1\}^n$ it is the Hamming weight (i.e., distance from $\vec{0}$) $wt(\cdot)$, and in case X is a subspace of Euclidean space, there is the ℓ_2-norm defined on X. Condition C that must be satisfied by the output will be naturally related to this metric. For example, when $L_i \subset \mathbb{R}^n$, we can ask for tuples whose sum, $\vec{x}_1 + \ldots + \vec{x}_k$, is short. We give more examples below.

Clearly, algorithms for k-List problems require at least $\sum_i |L_i|$ memory, and the running time is at least $\max\{\sum_i |L_i|, \#\text{output solutions}\}$. We also consider cases when lists are equal.

There is a plethora of cryptographic tasks that can be phrased as a k-List problem. Probably the most popular is the collision-search problem for a hash-function $f : \{0,1\}^* \to \{0,1\}^n$. The birthday paradox states that if we have 2 lists each of size $2^{n/2}$ with elements of the form $(x, f(x))$ in the first list and $(x', f(x'))$ in the second, then search for a pair with s.t. $f(x) = f(x')$ gives a collision for f with constant success probability.

Wagner extended this idea to what he calls the Generalized Birthday problem [Wag02]: given k lists $L_i \subset \{0,1\}^n$, a tuple $(\vec{x}_1, \ldots, \vec{x}_k) \in L_1 \times \ldots \times L_k$ is a solution if $\vec{x}_1 \oplus \ldots \oplus \vec{x} = \vec{t}$ for some input $\vec{t} \in \{0,1\}^n$. He proposed an algorithm running in time $\widetilde{\Theta}(2^{\sqrt{n}})$ that uses $\widetilde{\Theta}(2^{\sqrt{n}})$ lists of size $\widetilde{\Theta}(2^{\sqrt{n}})$. The Generalized Birthday problem has found its applications in breaking hash-functions, forging signature schemes [Wag02], and attacking stream-ciphers [NS15].

Another famous k-List algorithm is due to Blum, Kalai, and Wasserman [BKW03]. We write `BKW` when we refer to this algorithm. It solves the *Learning Parity with Noise* problem – a binary counterpart of LWE – a very well-known problem in machine learning. Different to LWE, the vectors $\vec{a}_i$ and the secret $\vec{s}$ are now from $\{0,1\}^n$, and the noise e_i follows the Bernoulli distribution with parameter $\tau \in [0, 1/2)$. In contrast to Wagner's algorithm, where the list-sizes and their number k can be optimally chosen, the number of lists in the `BKW` algorithm is limited to $n^{1-\varepsilon}$. A tuple $(\vec{x}_1, \ldots, \vec{x}_k)$ is a solution if $wt(x_1 \oplus \ldots \oplus x_k) = 1$.

The `BKW` algorithm received even more attention after Albrecht et al. in [ACF+15] analyzed it for LWE. In this case, the list elements are formed from $\vec{a}_i \in \mathbb{Z}_q^n$ – the first components of LWE samples. A tuple we seek for must satisfy $\vec{x}_1 + \ldots + \vec{x}_k = (0, \ldots 0, 1, 0, \ldots, 0) \bmod q$. Recently, Kircher-Fouque

[KF15] and Guo et al. [GJS15] independently realized that the condition can be slightly relaxed: asking for a tuple with a short sum $\vec{x}_1 + \ldots + \vec{x}_k$ leads to an improved algorithm for LWE.

Kuperberg's quantum algorithm for the Dihedral Hidden Subgroup problem [Kup05] is yet another example of a k-List algorithm. It operates with lists $L_i \subset \mathbb{Z}_N$ and searches for a tuple $(x_1, \ldots, x_k)$ that sums up to $N/2$. This is a quantum analog of Wagner's algorithm for the Generalized Birthday problem, but it operates with relative phases, x_i's, of quantum superpositions. Optimal list sizes and k are chosen exactly in the same way as in Wagner's algorithm.

If we turn our attention to k-List problems over Euclidean spaces, we land in the realm of algorithms for the Shortest Vector Problem. Namely, for $X \subset \mathcal{L}$, and $k = 2$, the task of finding pairs $(\vec{x}_1, \vec{x}_2) \in L_1 \times L_2$ s.t. $\|\vec{x}_1 + \vec{x}_2\| < \min\{\|\vec{x}_1\|, \|\vec{x}_2\|\}$ is at heart of so-called *sieving* algorithms for SVP [AKS01]. List-elements are vectors of a lattice. The lists are *exponential* in the lattice-dimension. An important requirement here is that the number of solutions has to be asymptotically equal to the size of input lists, i.e. exponential. Large memory complexity precludes sieving algorithms from being practically competitive with other algorithms for SVP.

A progress towards memory-efficient SVP-sieving was recently achieved by Bai, Laarhoven, and Stehlé (BLS, for short). In [BLS16], they generalize sieving algorithms for k larger than 2. Intuitively, the larger k is, the shorter the input lists can be to guarantee the same number of solutions, since instead of $|L_1| \cdot |L_2|$ tuples, we have $|L_1| \cdot \ldots \cdot |L_k|$ tuples in total. On the other hand, larger k results in increased running time. In Sect. 4.1 we present a k-List algorithm that achieves a better running time than the one presented in [BLS16].

We should stress that in all these examples the expected number of solutions is very large. In other words, k-List problems are *high density* problems. The algorithms exploit this fact by dropping many solutions and focusing only on solutions with some distinguished property.

A common strategy to solve a k-List problem is to identify such a distinguished property (or a search criterion) for a solution-tuple that would help to find a solution. The size of input lists is then chosen such that the number of solutions that satisfy this property is large enough.

For example, Wagner's algorithm [Wag02] outputs a solution $(\vec{x}_1, \ldots, \vec{x}_n)$ if $[(\vec{x}_i \oplus \vec{x}_{i+1})]_1^\ell = \vec{0}$ for all odd $i < n$, where for a vector $\vec{x}$ we denote $[x]_i^j$ as its projection on coordinates $(i, \ldots j)$ for $i \leq j$. The value ℓ in Wanger's algorithm is optimally chosen as $\ell = \sqrt{n}$. Such a pair-wise constraint allows to search for a pair of vectors $(\vec{x}_i, \vec{x}_{i+1}) \in L_i \times L_{i+1}$ independently from another pair $(\vec{x}_j, \vec{x}_{j+1}) \times (L_j, L_{j+1})$. Each of these pairs is then combined into a vector producing two new lists with elements having 0's on the last ℓ coordinates. The same constraint is then put on the next ℓ coordinates of vectors from the new lists. Of course, with this approach we lose many solutions and we account for that by appropriately setting the input-lists' sizes. Currently, the searching criteria of Wagner's algorithm is the best for the Generalized k-List problem over $\{0,1\}^n$.

We describe our searching criteria in Sect. 4.1.1 for SVP. It is similar to Wagner's criteria in a sense that it puts a *pairwise* constraint on a solution-tuple. On the other hand, different from Wagner's algorithm, our constraint makes a 'global' effect on all the lists: choosing $\vec{x} \in L_1$ affects not only L_2, but all L_i's. It turns out that our constraint does not only speed up the search, but is also met by a large fraction of all the solutions. Thus, our searching constraint does not incur an growth of the input lists. This is particularly beneficial for SVP-sieving where memory is a big concern.

Finally, in Sect. 4.2, we present another k-List algorithm but now for the *approximate* SVP problem on q-ary lattices of dimension $2n$, where the approximation factor is poly(n). We present two algorithms: the first is very similar to the `BKW` algorithm presented in [GJS15, KF15]. The second puts a more rigid constraint on the solution-tuple, which nevertheless, does not increase the input lists and yet results in a faster k-List algorithm.

Table 4.1 summarizes known k-List algorithms.

$L_i \subset \{0,1\}^n$			
Algorithm	k	$\lvert L_i\rvert$	T
BKW for LPN:			
$wt(\vec{x}_1 \oplus \ldots \oplus \vec{x}_k) = 1$ ([BKW03]), or			
$wt(\vec{x}_1 \oplus \ldots \oplus \vec{x}_k)$ - small ([GJL14])			
• $2^{\mathcal{O}(\frac{n}{\log n})}$ samples	$n^{1-\varepsilon}$	$2^{\mathcal{O}(\frac{n}{\log n})}$	$2^{\mathcal{O}(\frac{n}{\log n})}$
• poly(n) samples [Lyu05b]	$n^{1-\varepsilon}$	poly(n)	$2^{\mathcal{O}(\frac{n}{\log\log n})}$
Wagner's k-tree algorithm [Wag02] $\vec{x}_1 \oplus \ldots \oplus \vec{x}_k = \vec{t}$ for some input $\vec{t}$	k	$\widetilde{\mathcal{O}}(k2^{\frac{n}{\log k+1}})$	$\widetilde{\mathcal{O}}(k2^{\frac{n}{\log k+1}})$
Extended k-tree algorithm [MS09] $\vec{x}_1 \oplus \ldots \oplus \vec{x}_k = \vec{t}$ for some input $\vec{t}$	k	m	$2^{(\log k+\frac{n-2^p\log m}{\log k-p})}$, where p is the smallest integer s.t. $n \le (\log k - p + 1)2^p \log m$
$L_i \subset \mathbb{Z}_q$			
Dense Subset-Sum [Lyu05a] $\sum_{i\in I} x_i = t \bmod q$, $q = 2^{n^\varepsilon}$	$\frac{1}{2}n^{1-\varepsilon}$	$2^{\mathcal{O}(\frac{n^\varepsilon}{\log n})}$	$2^{\mathcal{O}(\frac{n^\varepsilon}{\log n})}$
Kuperberg's algorithm [Kup05] $\sum_{i\in I} x_i = q/2$, $q = 2^n$	$2^{\mathcal{O}(\sqrt{n})}$	$2^{\mathcal{O}(\sqrt{n})}$	$2^{\mathcal{O}(\sqrt{n})}$
$L_i \subset \mathbb{Z}_q^n$			
BKW for LWE with parameters (n, α, q):			
• $2^{\Theta(n)}$ LWE samples			
– $\lVert\vec{x}_1 + \ldots + \vec{x}_k\rVert = 1$ [ACF+15]	$\Theta(n)$	$2^{\frac{1}{2}\frac{c_q}{c_\alpha-1/2}n+o(n)}$	$2^{\frac{1}{2}\frac{c_q}{c_\alpha-1/2}n+o(n)}$
– $\lVert\vec{x}_1 + \ldots + \vec{x}_k\rVert$ - small [GJS15, KF15]	$\Theta(n)$	$2^{\frac{c_q}{1+2\ln(c_q/(c_q+c_\alpha))}n+o(n)}$	$2^{\frac{c_q}{1+2\ln(c_q/(c_q+c_\alpha))}n+o(n)}$
• $\Theta(n\log n)$ LWE samples			
– $\lVert\vec{x}_1 + \ldots + \vec{x}_k\rVert = 1$	$\Theta(n)$	$\Theta(n\log n)$	$2^{\frac{1}{2}\frac{c_q}{c_\alpha}n+o(n)}$
– $\lVert\vec{x}_1 + \ldots + \vec{x}_k\rVert$ - small [HKM]	$\Theta(n)$	$\Theta(n\log n)$	$2^{\frac{1}{\ln(c_q/(c_q+c_\alpha))}n+o(n)}$
$L_i \subset \mathcal{L}$			
Sieving algorithms for SVP			
• $\lVert\vec{x}_1 \pm \vec{x}_2\rVert < \max\{\lVert\vec{x}_1\rVert, \lVert\vec{x}_2\rVert\}$ [BDGL16]	2	$2^{0.208n+o(n)}$	$2^{0.292n+o(n)}$
• $\lVert\vec{x}_1 \pm \vec{x}_2 \pm \ldots \pm \vec{x}_k\rVert < \max_i\{\lVert\vec{x}_i\rVert\}$			
– BLS Algorithm [BLS16]	$\Theta(1)$	$\widetilde{\mathcal{O}}\Big(\big(\frac{k}{k\frac{k-1}{k+1}}\big)^{\frac{n}{2}}\Big)$	see Eq. (4.7)
– Our Algorithm 7 (Sect. 4.1.2)	$\Theta(1)$		see Eq. (4.9)
$L_i \subset \mathcal{L}_q^\perp(A) \subset \mathbb{Z}_q^{2n}$			
Combinatorial algorithms for appSVP_γ $\lVert\sum_i \vec{x}_i\rVert < n^{\mathsf{c}_\gamma}\lambda_1(\mathcal{L}_q^\perp(A))$, $\mathsf{c}_\gamma = \Theta(1)$			
• Algorithm 9 (Sect. 4.2.1)	$\Theta(n)$	See Eq. (4.17)	
• Algorithm 9 (Sect. 4.2.1)	$\Theta(n)$	See Eq. (4.19)	

Tab. 4.1: k-List algorithms. In the left-most column alongside with an algorithm, we show the condition that should be met by a solution tuple $(x_1, \ldots, x_k)$. For LWE, we set $q = n^{c_q}, \alpha = \frac{1}{n^{c_\alpha}}$.

4.1 Approximate k-List in Euclidean norm

The problem we are going solve in this section is the following special case of the k-List problem. This problem is at the heart of sieving algorithms for the Shortest Vector Problem.

Definition 17 (Approximate k-List problem in l_2-norm)**.** *Let $0 < t < \sqrt{k}$. Given k lists $L_1, \ldots, L_k$ of equal exponential size whose elements are iid. uniformly chosen vectors from the n-sphere S^n, the task is to output a $1 - o(1)$-fraction of k-tuples $\vec{x}_1 \in L_1, \ldots, \vec{x}_k \in L_k$ s.t. $\|\vec{x}_1 + \ldots + \vec{x}_k\|^2 \leq t^2$. Such a tuple $(\vec{x}_1, \ldots, \vec{x}_k)$ is called a solution to the approximate k-List problem.*

We analyze the case where t and k are constant and the input lists are of size $2^{\mathsf{c}n}$ for some constant c. The restriction $t < \sqrt{k}$ is set to get a meaningful problem. Due to the fact that for large n, random vectors $\vec{x}_i$'s from S^n are almost orthogonal with high probability (cf. Thm. 22), a $1 - o(1)$-fraction of tuples $(\vec{x}_1, \ldots, \vec{x}_k) \in L_1 \times \ldots \times L_k$ satisfy $\|\vec{x}_1 + \ldots + \vec{x}_k\|^2 \approx k$. So the problem is non-trivial when either $t < \sqrt{k}$, or $t > \sqrt{k}$. We concentrate on the former. With a simple modification our results apply to the latter case as well. Moreover, our algorithm works for lists of different sizes, but it would unnecessarily complicate the analysis, so we stick to lists of equal size. Additionally, equally-sized lists is a relevant scenario for the SVP-sieving algorithms.

In the applications to sieving (see Sect. 4.1.4), we have $t = 1$ and look for solutions with the property $\|\vec{x}_1 \pm \ldots \pm \vec{x}_2\| \leq 1$. Since there are $2^k = \mathcal{O}(1)$ possible choices for signs, we can consider each choice separately increasing the running time of the algorithm by a constant factor.

4.1.1 Configurations

All the k-List algorithms make use of the fact that there are many solution-tuples and we are allowed to output a fraction of all the solutions. The main challenge in designing an efficient k-List algorithm is to identify a criterion s.t. (1) a solution that matches the criterion is easy to find, and (2) enough solutions satisfy this criterion. The first property leads to a faster algorithm, the second is specified by the problem. In the case of the approximate k-List problem, we want to output almost all solutions.

To define the criterion we use in our algorithm, recall the definition of a Gram matrix.

Definition 18 (Gram Matrix)**.** *For vectors $\vec{x}_1, \ldots, \vec{x}_k$ from $\mathbb{R}^n$, the* Gram matrix $C \in \mathbb{R}^{k \times k}$ *is a positive semidefinite matrix whose entries are pairwise inner products: $C_{i,j} = \langle x_i, x_j \rangle$.*

Note that the Gram matrix is invariant under simultaneous rotations and reflections of all $\vec{x}_i$'s. This property also holds for a k-tuple from S^n that forms a solution to the approximate k-List problem as both, rotation and reflection, preserve distance. Hence, we are interested in solutions up to such symmetry. We set our searching criteria to be a specific Gram matrix of vectors $\vec{x}_1, \ldots, \vec{x}_k$ which we call a *configuration*.

Definition 19 (Configuration)**.** *The* configuration $C = \mathit{Conf}(\vec{x}_1, \ldots, \vec{x}_k)$ *for $\vec{x}_1, \ldots, \vec{x}_k \in \mathsf{S}^n$ is the Gram matrix $C_{i,j} = \langle x_i, x_j \rangle$.*

The configuration gives all the necessary information on the geometry of the tuple, and in particular

$$\|\sum_i \vec{x}_i\|^2 = \sum_i \|\vec{x}_i\|^2 + \sum_{i \neq j} \langle \vec{x}_i, \vec{x}_j \rangle = k + 2 \sum_{i<j} \langle \vec{x}_i, \vec{x}_j \rangle. \tag{4.1}$$

Let us define the space of all possible configurations for $\vec{x}_i \in \mathsf{S}^n$ together with the space of those

configurations that give a tuple with the property $\|\sum_i \vec{x}_i\|^2 \le t$:

$$\mathscr{C} = \{C \in \mathbb{R}^{k\times k} \mid C \text{ symmetric positive semi-definite}, C_{i,i} = 1\},$$
$$\mathscr{C}_{\le t} = \{C \in \mathscr{C} \mid \sum_{i,j} C_{i,j} \le t^2\}.$$

For fixed k, we think of the set $\mathscr{C}$ as a finite set which we can efficiently enumerate. Observing that a tuple $(\vec{x}_1, \ldots, \vec{x}_k)$ is a solution to the approximate k-List problem iff $\mathrm{Conf}(\vec{x}_1, \ldots, \vec{x}_k) \in \mathscr{C}_{\le t}$, immediately gives us an algorithm: we enumerate over all configurations in $\mathscr{C}_{\le t}$ and solve the *k-List configuration problem* defined as follows.

Definition 20 (Configuration Problem)**.** *Given k exponentially sized lists $L_1, \ldots, L_k$ of vectors from S^n, a target configuration $C \in \mathscr{C}$, and $\varepsilon > 0$, the task is to output* all *k-tuples $\vec{x}_1 \in L_1, \ldots, \vec{x}_k \in L_k$ s.t. $|\langle \vec{x}_i, \vec{x}_j\rangle - C_{i,j}| \le \varepsilon$ for all i, j. Such a tuple is called a solution to the Configuration problem.*

Remark 21. *To simplify the analysis, we assume we can compute with real numbers. Our algorithm and the analysis remain true when we use sufficiently precise approximations. Since the inner products $\langle \vec{x}_i, \vec{x}_j\rangle$ take real values, asking for the exact equality to C does not bring any solution. We, therefore, introduce some small $\varepsilon > 0$. For two configurations C, C', we write $C \approx_\varepsilon C'$ when $|C_{i,j} - C'_{i,j}| \le \varepsilon$.*

The crucial property of a solution $(\vec{x}_1, \ldots, \vec{x}_k)$ to the Configuration problem is the fact that it can be *locally* verified as we only have to look at *pairs* $\vec{x}_i, \vec{x}_j$. Note that a solution to the approximate k-List problem as given in Def. 17 does not share this 'locality' feature.

Now we have an algorithm to solve the k-List problem: (1) enumerate all the configurations $C \in \mathscr{C}_{\le t}$, and (2) solve the Configuration problem for each C. Below we show how to bypass the first step. It turns out that we do not have to enumerate all the configurations to output a $1 - o(1)$-fraction of the solutions to the k-List problem. There exist one particular configuration, later denoted as $C_{\mathrm{B,t}}$, which is attained by most of the solutions. This allows us to solve the Configuration problem only for $C_{\mathrm{B,t}}$ to obtain enough solution-tuples for the k-List problem. To give $C_{\mathrm{B,t}}$ explicitly, we study the Wishart distribution [Wis28], which is a matrix generalization of the chi-squared distribution.

Wishart distribution. Consider k vectors $\vec{x}_1, \ldots, \vec{x}_k \in \mathbb{R}^{n+1}$ sampled independently from $(n+1)-$dimensional spherical Gaussian (mean 0 and standard deviation 1). Set $S_{i,j} = \langle \vec{x}_i, \vec{x}_j\rangle \in \mathbb{R}^{k\times k}$. For integer k, n with $n + 1 > k - 1$, a random $k \times k$ symmetric matrix S has a Wishart distribution with the probability density function

$$\rho_{\mathrm{Wishart}}(S) = \frac{e^{-\frac{1}{2}\cdot \mathrm{Tr}S} \cdot \det(S)^{\frac{n-k}{2}}}{2^{\frac{(n+1)k}{2}} \pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma\left(\frac{n+1-i}{2}\right)} \mathrm{d}S, \tag{4.2}$$

where $\mathrm{d}S = \prod_{i\le j} \mathrm{d}S_{i,j}$, $\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}\mathrm{d}x$ is the Gamma-function, and $\mathrm{Tr}(S)$ is the trace-function (i.e., the sum of the main-diagonal elements of S). A derivation of these density can be found in [Eat07].

Note that matrix S is a Gram matrix of vectors *not* from S^n. To get the density function for distribution on our configuration space $\mathscr{C}$ where vectors are sampled uniformly from the n-sphere, we have to normalize S and change the reference density $\mathrm{d}S$ appropriately.

Theorem 22. *Let $\vec{x}_1, \ldots, \vec{x}_k \in \mathsf{S}^n$ be independent uniformly distributed on the n-sphere, $n > k$. Then the configuration $C = Conf(\vec{x}_1, \ldots, \vec{x}_k)$ follows a distribution $\rho_{\mathscr{C}}$ on $\mathscr{C}$ with the probability density function*

$$\rho_{\mathscr{C}}(C) = W_{n,k} \cdot \det(C)^{\frac{1}{2}(n-k)} \mathrm{d}\mathscr{C} = \widetilde{\mathcal{O}}_k\Big(\det(C)^{\frac{n}{2}}\Big)\mathrm{d}\mathscr{C},$$

where $\mathrm{d}\mathscr{C} = \mathrm{d}C_{1,2}\cdots\mathrm{d}C_{k-1,k}$ *and* $W_{n,k} = \pi^{-\frac{k(k-1)}{4}}\prod_{i=0}^{k-1}\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n+1-i}{2}\right)} = \widetilde{\mathcal{O}}_k\left(n^{\frac{(k-1)k}{4}}\right)$ *is a normalization constant that only depends on* n *and* k.

Proof. Let $C_{i,j} = \frac{S_{i,j}}{\sqrt{S_{i,i}S_{j,j}}}$, where S follows the Wishart distribution with pdf given in Eq. (4.2). This normalization defines the map Φ from $\mathbb{R}^{\frac{k(k+1)}{2}}$ to itself that represents the following change of variables:

$$\begin{aligned}&\Phi(S_{1,1}, S_{2,2}, \ldots, S_{k,k}, S_{1,2}, \ldots, S_{1,k}, S_{2,3}, \ldots, S_{k-1,k}) = \\ &(S_{1,1}, S_{2,2}, \ldots, S_{k,k}, \frac{S_{1,2}}{\sqrt{S_{1,1}S_{2,2}}}, \ldots, \frac{S_{1,k}}{\sqrt{S_{1,1}S_{k,k}}}, \frac{S_{2,3}}{\sqrt{S_{2,2}S_{3,3}}}, \ldots, \frac{S_{k-1,k}}{\sqrt{S_{k-1,k-1}S_{k,k}}}) = \\ &(S_{1,1}, S_{2,2}, \ldots, S_{k,k}, C_{1,2}, \ldots, C_{1,k}, C_{2,3}, \ldots, C_{k-1,k}).\end{aligned}$$

Note that we keep the diagonal elements $S_{i,i}$ to make the transformation Φ injective, as we want to apply the substitution rule $\prod_{i=1}^{k}\mathrm{d}S_{i,i}\prod_{i<j}C_{i,j} = |\det(D\Phi)|\cdot\prod_{i\le j}\mathrm{d}A_{i,j}$, where $D\Phi$ is the Jacobian of Φ. This Jacobian is a triangular matrix with the main diagonal

$$(1, \ldots, 1, \frac{1}{\sqrt{S_{1,1}S_{2,2}}}, \ldots, \frac{1}{\sqrt{S_{1,1}S_{k,k}}}, \frac{1}{\sqrt{S_{2,2}S_{3,3}}}, \ldots, \frac{1}{\sqrt{S_{k-1,k-1}S_{k,k}}}),$$

so its determinant is $|\det(D\Phi)| = \prod_i \frac{1}{\sqrt{S_{i,i}}^{k-1}}$. Further, we can express S as $S = TCT$, where T is a diagonal matrix with $(\sqrt{S_{1,1}}, \ldots, \sqrt{S_{k,k}})$ on the main diagonal. Therefore, $\det(S) = \det(C)\cdot\prod_i S_{i,i}$. Applying the transformation to the Wishart density function, one readily obtains a pdf. in $(S_{1,1}, \ldots, S_{k,k}, C_{1,2}, \ldots, C_{k-1,k})$-variables

$$\rho_{\text{Wishart}} = \frac{e^{-\frac{1}{2}\sum_i S_{i,i}}\det(C)^{\frac{n-k}{2}}\prod_i S_{i,i}^{\frac{n-k}{2}}}{2^{\frac{(n+1)k}{2}}\pi^{\frac{k(k-1)}{4}}\prod_{i=0}^{k-1}\Gamma(\frac{n-i+1}{2})}\prod_i\sqrt{S_{i,i}}^{k-1}\prod_i\mathrm{d}S_{i,i}\prod_{i<j}\mathrm{d}C_{i,j}. \tag{4.3}$$

As we want to relate the distribution to $C_{i,j}$ only, we integrate over $\prod_i \mathrm{d}S_{i,i}$ to obtain the desired $\rho_{\mathscr{C}}$. From Eq. (4.3), it follows that $\rho_{\mathscr{C}}$ is of the form $\rho_{\text{Wishart}} = W_{n,k}\det(C)^{\frac{n-k}{2}}\mathrm{d}\mathscr{C}$ for some factor $W_{n,k}$ that depends only on k and n. We write $W_{n,k}$ as $W_{n,k} = \frac{1}{t}e^{-\frac{1}{2}\sum_i S_{i,i}}\prod_i S_{i,i}^{\frac{n-k}{2}}$, where t is the denominator in Eq. (4.3), and compute

$$\begin{aligned}W_{n,k} &= \frac{1}{t}\int_{A_{1,1}}\cdots\int_{A_{k,k}} e^{-\frac{1}{2}\sum_i S_{i,i}}\prod_i S_{i,i}^{\frac{n-k}{2}}\prod_i\sqrt{S_{i,i}}^{k-1}\prod_i\mathrm{d}S_{i,i} \\ &= \frac{1}{t}\int_{A_{1,1}}\cdots\int_{A_{k,k}} e^{-\frac{1}{2}\sum_i S_{i,i}}\prod_i S_{i,i}^{\frac{n-1}{2}}\prod_i\mathrm{d}S_{i,i} \\ &= \frac{1}{t}\Big(\int_{S_{1,1}=0}^{+\infty} S_{1,1}^{\frac{n-1}{2}}e^{-\frac{1}{2}S_{1,1}}\mathrm{d}S_{1,1}\Big)^k \\ &= \frac{2^{\frac{k(n+1)}{2}}}{t}\Big(\int_{S_{1,1}=0}^{+\infty}\Big(\frac{S_{1,1}}{2}\Big)^{\frac{n+1}{2}-1}e^{-\frac{1}{2}S_{1,1}}\mathrm{d}\frac{S_{1,1}}{2}\Big)^k \\ &= \frac{2^{\frac{k(n+1)}{2}}}{t}\Big(\int_{x=0}^{+\infty}x^{\frac{n+1}{2}-1}e^{-x}\mathrm{d}x\Big)^k \\ &= \frac{2^{\frac{k(n+1)}{2}}\cdot\Gamma\left(\frac{n+1}{2}\right)^k}{t} = \frac{\Gamma\left(\frac{n+1}{2}\right)^k}{\pi^{\frac{k(k-1)}{4}}\prod_{i=0}^{k-1}\Gamma(\frac{n-i+1}{2})}.\end{aligned}$$

From Stirling's formula, $\Gamma(n) \sim \left(\frac{n}{e}\right)^n$, we have for any fixed z and $n \to \infty$, $\frac{\Gamma(n+z)}{\Gamma(n)} = \mathcal{O}_z(n^z)$. Finally, we have

$$W_{n,k} = \mathcal{O}_k\Big(n^{\sum_{i=0}^{k-1} \frac{i}{2}}\Big) = \mathcal{O}_k\Big(n^{\frac{k(k-1)}{4}}\Big).$$

□

Since we now know the distribution on the space $\mathscr{C}$, we want to find a configuration $C \in \mathscr{C}$ with the largest mass. It turns out that this is a configuration with the highest amount of symmetry, i.e. when all off-diagonal entries are equal. We prove this statement in Thm. 25 below. We call such configurations *balanced* and denote them C_{B}.

Definition 23 (Balanced Configuration)**.** *A configuration $C_B \in \mathscr{C}$ is balanced if $C_{i,j} = C_{i',j'}$ for all $i \neq i', j \neq j'$. A balanced configuration that additionally belongs to $\mathscr{C}_{\leq t}$ for some target length $t > 0$ is denoted $C_{B,t}$.*

Before showing that $\rho_{\mathscr{C}}$ indeed attains its maximum at C_{B}, we compute the determinant of C_{B}.

Lemma 24. *Let*

$$C = \begin{pmatrix} 1 & a & a & \dots & a \\ a & 1 & a & \dots & a \\ a & a & 1 & \dots & a \\ \vdots & & & \ddots & \vdots \\ a & a & a & \dots & 1 \end{pmatrix} \in \mathbb{R}^{k \times k}.$$

Then $\det(C) = (1-a)^{k-1}(1 + (k-1)a)$.

Proof. We write $C = (1-a) \cdot \mathbb{1}_k + a \cdot \vec{1} \cdot \vec{1}^t$. Sylvester's Determinant Identity states that $\det(\mathbb{1}_m + AB) = \det(\mathbb{1}_n + BA)$ for any A and B of dimensions $m \times n$ resp. $n \times m$. It follows that

$$\begin{aligned} \det(C) &= \det\big((1-a)(\mathbb{1}_k + \tfrac{a}{a-1}\vec{1} \cdot \vec{1}^t)\big) = (1-a)^k \det\big(\mathbb{1}_1 + \tfrac{a}{a-1}\vec{1}^t \cdot \vec{1}\big) \\ &= (1-a)^k\big(1 + \tfrac{a}{1-a}k\big) = (1-a)^{k-1}(1 + (k-1)a). \end{aligned}$$

□

The space of all configurations $\mathscr{C}$ is compact[1] and, therefore, integrating over it will asymptotically pick the maximum value. So the probability that a uniform random tuple $(\vec{x}_1, \dots, \vec{x}_k)$ forms a "good" configuration from $\mathscr{C}_{\leq t}$, and consequently, gives a solution to the Configuration problem, is

$$\int_{\mathscr{C}_{\leq t}} \rho_{\mathscr{C}} = \widetilde{\mathcal{O}}\big(\max_{C \in \mathscr{C}_{\leq t}} \det(C)^{n/2}\big).$$

The next theorem determines this maximum.

Theorem 25. *Let $0 < t < \sqrt{k}$ be a target length and $\mathscr{C}_{\leq t} \subset \mathscr{C}$ be the subset of configurations with target length at most t. Then* $\det(C)$ *attains its unique maximum over $\mathscr{C}_{\leq t}$ at the balanced configuration $C_{B,t}$ with $C_{i,j} = \frac{t^2-k}{k^2-k}$ for all $i \neq j$ with maximal value*

$$\det(C_{B,t}) = \frac{t^2}{k}\left(\frac{k^2 - t^2}{k^2 - k}\right)^{k-1}.$$

In particular, for $t = 1$, $C_{i,j} = -\frac{1}{k}$ and $\det(C_{B,1}) = \frac{(k+1)^{k-1}}{k^k}$.

[1] Closeness immediately follows from the fact that $\mathscr{C} \subset \mathbb{R}^{k^2}$; further, all entries of an element from $\mathscr{C}$ are bounded, in particular, $C_{i,j} \leq 1$.

Proof. For $k = 2$, the statement immediately follows from Eq. (4.1). So assume $k \geq 3$.

Consider configurations C with $\mathrm{Tr}(C) = k$ and $\sum_{i,j} C_{i,j} \leq t^2$. This is clearly weaker than the condition $C_{i,i} = 1$ and later we show that the stronger condition is met at the maximum.

From the fact that C is a Gram matrix, and hence, it is positive semi-definite, it follows that its eigenvectors $\vec{v}_1, \ldots, \vec{v}_k$ form an orthonormal basis and its eigenvalues $0 \leq \lambda_1 \leq \ldots \leq \lambda_k$ are positive.

We have $\mathrm{Tr}(C) = \sum_i \lambda_i$ and $\det(C) = \prod_i(\lambda_i)$. We want to find λ_i's that maximize the determinant. In particular, we show that $\vec{1}$ is an eigenvector for maximal $\det(C)$. To see this, write $\sum_{i,j} C_{i,j} = \vec{1}C\vec{1}^{\mathsf{t}}$, so for the smallest eigenvalue λ_1 it holds

$$t^2 \geq \vec{1}C\vec{1}^{\mathsf{t}} \geq \lambda_1 \|\vec{1}\|^2 = k\lambda_1. \tag{4.4}$$

We have $\lambda_1 \leq \frac{t^2}{k} < 1$. The Arithmetic Mean-Geometric Mean inequality stating that for non-negative real x_i's, $\frac{\sum_i x_i}{n} \geq \sqrt[n]{\prod_i x_i}$, gives for $\det(C) = \lambda_1 \prod_{i=2}^k \lambda_i$ and $\sum_{i=2}^k \lambda_i = k - \lambda_1$:

$$\det(C) \leq \lambda_1 \left(\frac{k-\lambda_1}{k-1}\right)^{k-1}.$$

As the derivative of the right-hand side w.r.t. λ_1, $\frac{k(1-\lambda_1)}{k-1}\left(\frac{k-\lambda_1}{k-1}\right)^{k-2} > 0$, is everywhere positive, we bound $\det(C)$ by plugging in the maximal $\lambda_1 = \frac{t^2}{k}$:

$$\det(C) \leq \lambda_1 \left(\frac{k-\lambda_1}{k-1}\right)^{k-1} \leq \frac{t^2}{k}\left(\frac{k-\frac{t^2}{k}}{k-1}\right)^{k-1} = \frac{t^2}{k}\left(\frac{k^2-t^2}{k^2-k}\right)^{k-1}. \tag{4.5}$$

The first inequality in Eq. (4.5) becomes an equality iff $\lambda_2 = \ldots = \lambda_k$ and the second when $\lambda_1 = \frac{t^2}{k}$. In this case, from $\mathrm{Tr}(C) = k$, we have $\lambda_2 = \frac{k^2-t^2}{k(k-1)}$ and $\vec{1}$ is an eigenvector of C with eigenvalue λ_1.

Now we show that $C_{i,i} = 1$ for maximal $\det(C)$ and $C_{i,j}$ for $i \neq j$ as in the theorem statement. Consider the eigendecomposition $C = V\Lambda V^{-1} = V\Lambda V^{\mathsf{t}}$, where V is the orthonormal matrix with $\vec{v}_i$'s as columns, Λ is the diagonal matrix with λ_i's on the main diagonal. Equivalently, we can write

$$C = \sum_i \lambda_i \vec{v}_i \vec{v}_i^{\mathsf{t}} = (\lambda_1 - \lambda_2)\vec{v}_1\vec{v}_1^{\mathsf{t}} + \lambda_2 \sum_{i=1}^k \vec{v}_i\vec{v}_i^{\mathsf{t}} = \frac{\lambda_1 - \lambda_2}{k}\vec{1}\vec{1}^{\mathsf{t}} + \lambda_2 \mathbb{1}_k.$$

Therefore, all the diagonal entries of C are equal to $\frac{\lambda_1-\lambda_2}{k} + \lambda_2$ and all the off-diagonal entries are equal to $\frac{\lambda_1-\lambda_2}{k}$. The theorem follows once we substitute the eigenvalues $\lambda_1 = \frac{t^2}{k}$, $\lambda_2 = \frac{k^2-t^2}{k(k-1)}$ for maximal $\det(C)$. □

In case we search for configurations with the target length $t > \sqrt{k}$, the proof should consider the largest eigenvalue λ_k instead of the smallest λ_1.

Also from the above proof it follows that looking at 'only-1' linear combination of $\vec{x}_i$'s is optimal. If instead we look for $\|\sum_i a_i\vec{x}_i\| \leq t$ for some $\vec{a} \neq \vec{1}$, the set of 'good' configurations $\mathscr{C}_{\leq t}$ would be $\{C \in \mathscr{C} \mid \|\vec{a}^{\mathsf{t}}C\vec{a}\| \leq t\}$ and in the proof above the eigenvector $\vec{1}$ would be replaced by $\vec{a}$. Since $\vec{1}$ has the minimal norm, the configurations from $\mathscr{C}_{\leq t}$ we are looking at are optimal.

In case $t = 1$, the balanced configuration $(\vec{x}_1, \ldots, \vec{x}_k)$ forms a regular $k+1$-dimensional simplex with center in the origin (the condition on pair-wise inner products $\langle \vec{x}_i, \vec{x}_j \rangle = -\frac{1}{k}$ matches with the central angle for the regular simplex). The missing $k+1^{\text{st}}$ point of the simplex is $-\sum_i \vec{x}_i$, i.e. the negative of the sum (see Fig. 4.1).

From our concentration result, it follows that a random k-tuple from S^n is a solution to the Configuration problem with probability $\widetilde{\mathcal{O}}(\det(C_{\mathrm{B,t}})^{n/2})$. Since we know the size of input list L_i, we can compute the expected number of solutions.

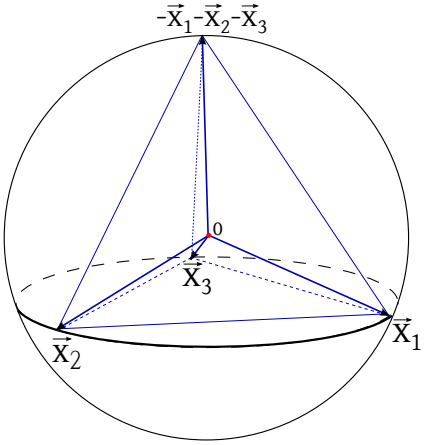


Fig. 4.1: A regular tetrahedron (3−simplex) represents a balanced configuration for $k = 3$.

Corollary 26. *Let k, t be fixed. Then the expected number of solutions to the Configuration problem with input lists of size $|L|$ is*

$$\mathbb{E}[\#\textit{solutions}] = \widetilde{\mathcal{O}}\left(|L|^k\left(\frac{t^2}{k}\left(\frac{k^2-t^2}{k^2-k}\right)^{k-1}\right)^{\frac{n}{2}}\right). \tag{4.6}$$

Proof. The total number of k-tuples is $|L|^k$. From Thms. 22 and 25, the probability that a random k-tuple forms a configuration from $\mathscr{C}_{\leq t}$ is $\widetilde{\mathcal{O}}(\det(C_{\mathrm{B,t}})^{n/2})$. Thm. 25 states the value for this determinant. □

As a consequence, we can easily compute the size of input lists for a desired output list's size. In algorithms for SVP, $t = 1$ and the output list is required to have the same size as input lists. The following corollary proves the conjecture stated in [BLS16].

Corollary 27. *Let k be fixed and $t = 1$. In the Configuration problem, for the input lists each of size $|L|$, the output list is expected to be of size $|L|$ if $|L| = \widetilde{\mathcal{O}}\left(\left(\frac{k}{k\frac{k}{k+1}}\right)^{\frac{n}{2}}\right)$.*

Proof. The statement immediately follows from setting the expression in Eq. (4.6) equal to $|L|$ for $t = 1$. □

Finally, we argue that solving the Configuration Problem gives a $1 - o(1)$ fraction of solutions for the k-List problem. This follows from Thm. 25. Essentially it states that for any fixed $\varepsilon > 0$, the probability that a randomly chosen solution to the approximate k-List problem forms a configuration ε-close to $C_{\mathrm{B,t}}$, converges exponentially to 1 as $n \to \infty$. Therefore, solving the k-List Configuration problem for $C_{\mathrm{B,1}}$ and restricting to only those solutions whose sum is at most t, gives a $1-o(1)$ fraction of solutions for the approximate k-List problem. These arguments justify the following corollary.

Corollary 28. *Let k, t be fixed. Then the approximate k-List problem with target length t can be solved in the same time as the k-List configuration problem with target configuration $C_{B,t}$ for any fixed $\varepsilon > 0$.*

4.1.2 Algorithm

Algorithm 7 k-List for the Configuration Problem

Input: $L_1, \ldots, L_k$ – lists of vectors from S^n. $\mathrm{Conf}_{i,j} = \langle \vec{x}_i, \vec{x}_j \rangle \in \mathbb{R}^{k \times k}$ – Gram matrix. $\varepsilon > 0$.
Output: L_{out} – list of k-tuples $\vec{x}_1 \in L_1, \ldots, \vec{x}_k \in L_k$, s.t. $|\langle \vec{x}_i, \vec{x}_j \rangle - \mathrm{Conf}_{ij}| \leq \varepsilon$, for all i, j.

$L_{\text{out}} \leftarrow \{\}$
for all $\vec{x}_1 \in L_1$ **do**
  for all $j = 2 \ldots k$ **do**
    $L_j^{(1)} \leftarrow \textsc{Filter}(\vec{x}_1, L_j, \mathrm{Conf}_{1,j}, \varepsilon)$
  for all $\vec{x}_2 \in L_2^{(1)}$ **do**
    for all $j = 3 \ldots k$ **do**
      $L_j^{(2)} \leftarrow \textsc{Filter}(\vec{x}_2, L_j^{(1)}, \mathrm{Conf}_{2,j}, \varepsilon)$
    $\ddots$
      for all $\vec{x}_k \in L_k^{(k-1)}$ **do**
        $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\vec{x}_1, \ldots \vec{x}_k)\}$
return L_{out}

function $\textsc{Filter}(\vec{x}, L, c, \varepsilon)$
  $L' \leftarrow \{\}$
  for all $\vec{x}' \in L$ **do**
    if $|\langle \vec{x}, \vec{x}' \rangle - c| \leq \varepsilon$ **then**
      $L' \leftarrow L' \cup \{\vec{x}'\}$
  return L'

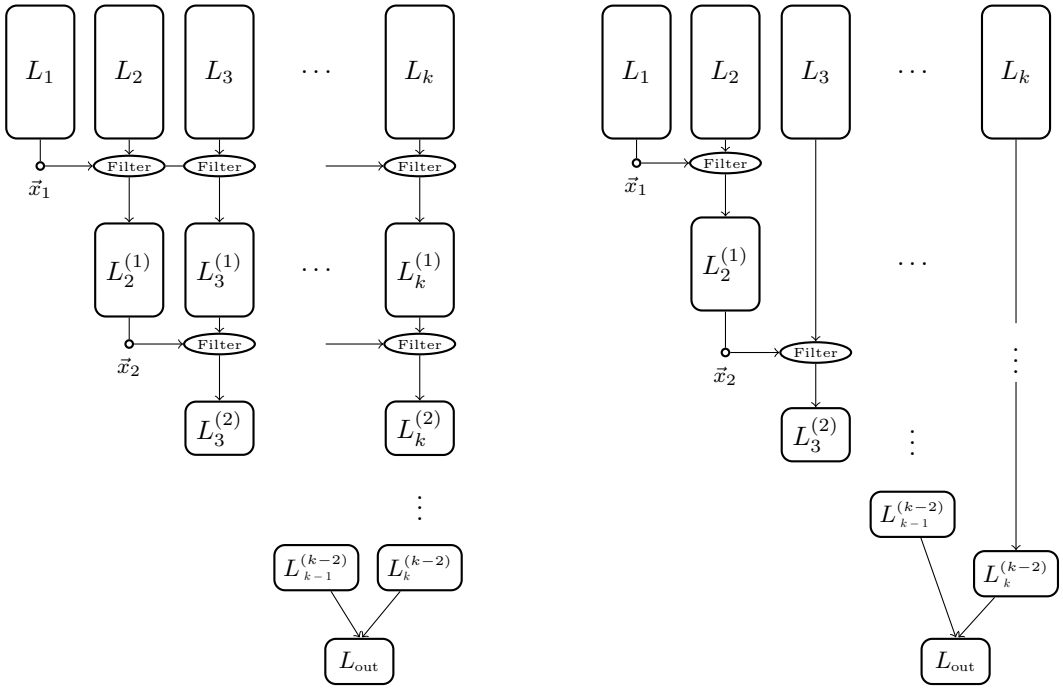
Now we are ready to describe our algorithm for the Configuration problem given in Def. 20.

On input the algorithm receives k lists $L_1, \ldots, L_k$, a target configuration Conf in the form of a Gram matrix $\mathrm{Conf}_{i,j} = \langle \vec{x}_i, \vec{x}_j \rangle \in \mathbb{R}^{k \times k}$ and a small $\varepsilon > 0$. The algorithm proceeds as follows: it picks an $\vec{x}_1 \in L_1$ and filters all the remaining lists with respect to the values $\langle \vec{x}_1, \vec{x}_i \rangle$ for all $2 \leq i \leq k$. More precisely, $\vec{x}_i \in L_i$ 'survives' the filter if $|\langle \vec{x}_1, \vec{x}_i \rangle - \mathrm{Conf}_{1,i}| \leq \varepsilon$. We put such an $\vec{x}_i$ into $L_i^{(1)}$ (the superscript indicates how many filters were applied to the original list L_i). On this step, all the k-tuples of the form $(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_k) \in \{\vec{x}_1\} \times L_2^{(1)} \times \ldots \times L_k^{(1)}$ with the first vector $\vec{x}_1$ fixed partially match the target configuration. Most importantly, the lists $L_i^{(1)}$ become much shorter than the original ones.

Next, we take $\vec{x}_2 \in L_2^{(1)}$ and create smaller lists $L_i^{(2)}$ from $L_i^{(1)}$ by filtering out all the $\vec{x}_i \in L_i^{(1)}$ that do not satisfy $|\langle \vec{x}_2, \vec{x}_i \rangle - \mathrm{Conf}_{2,i}| \leq \varepsilon$ for all $3 \leq i \leq k$. A tuple of the form $(\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots, \vec{x}_k) \in \{\vec{x}_1\} \times \{\vec{x}_2\} \times L_3^{(2)} \times \ldots \times L_k^{(2)}$ satisfies the target configuration $\mathrm{Conf}_{i,j}$ for $i = 1, 2$. Now we have the first two vectors fixed.

We proceed with this list-filtering strategy until we have all $\vec{x}_i$ for $1 \leq i \leq k$ fixed. We output all the survived k-tuples. Note that our algorithm becomes the trivial brute-force search algorithm once we are down to 2 lists $L_{k-1}^{(k-2)}, L_k^{(k-2)}$. As soon as we have fixed $\vec{x}_1, \ldots, \vec{x}_{k-2}$ and created $L_{k-1}^{(k-2)}, L_k^{(k-2)}$, we iterate over $L_{k-1}^{(k-2)}$ and check scalar products with every element from $L_k^{(k-2)}$. Our algorithm is detailed in Alg. 7.

In Fig. 4.2a, we stress the difference between our algorithm (left) and the algorithm for the Configuration problem presented in [BLS16] (right). While not being stated in terms of configurations, the BLS algorithm actually does search for tuples that form the balanced configuration but differently: for a fixed $\vec{x}_1$, it only filters the next list L_2 and the remaining $L_3, \ldots, L_k$ are left unchanged. Once $\vec{x}_2 \in L_2^{(1)}$ is chosen next, $L_3^{(2)}$ is obtained by applying filtering to the input L_3, while our Alg. 7 filters a smaller $L_3^{(1)}$. Certainly, in our approach we can miss some solutions that would be found by the BLS algorithm, but the results of Sect. 4.1.1 show that this is a tiny fraction of solutions which vanishes in the asymptotics. To see the effect of this fact in practice, we refer the reader to Sect. 4.1.5.



(a) Pictorial representation of Alg. 7. At level i, a filter receives as input $\vec{x}_i \in L_i^{(i-1)}$ and a vector $\vec{x}_j$ from $L_j^{(i-1)}$ (for the input lists, $L = L^{(0)}$). $\vec{x}_j$ passes through the filter if $|\langle \vec{x}_i, \vec{x}_j \rangle - \text{Conf}_{i,j}| \leq \varepsilon$, in which case it is added to $L_j^{(i)}$. All the vectors from $L_j^{(i-1)}$ for all $j \leq i+1$ are processed in this manner. The configuration Conf and $\varepsilon > 0$ are global parameters.

(b) The k-List algorithm given in [BLS16]. The main difference is that a filter receives as inputs $\vec{x}_i$ and a vector $\vec{x}_j \in L_j$, as opposed to $\vec{x}_j \in L_j^{(i-1)}$. Technically, in [BLS16], $\vec{x}_i$ survives the filter if $|\langle \vec{x}_i, \vec{x}_1 + \ldots + \vec{x}_{i-1} \rangle| \geq c_i$ for some predefined c_i. Due to our concentration results, this description is equivalent to the one given in [BLS16] in the sense that the returned solutions are (up to a sub-exponential fraction) the same.

Fig. 4.2: k-List Algorithms for the Configuration Problem (Def. 20). Left: Our Alg. 7. Right: The algorithm from [BLS16]

4.1.3 Analysis

In this section we analyze the complexity of Alg. 7 for the Configuration problem. First, we should mention that the memory complexity is completely determined by the input list-sizes $|L_i|$ (remember that we restrict to constant k) and the application of k filters does not change the asymtotics. In practice, all intermediate lists $L_i^{(j)}$ can be implemented by storing pointers to the elements of the original lists.

In the following, we compute the expected sizes of filtered lists $L_i^{(j)}$ and establish the expected running time of Alg. 7. Since our algorithm has an exponential running time 2^{cn} for some $c = \Theta(1)$, we are interested in determining c. We ignore polynomial factors, e.g. we do not take into account time spent for computing inner products.

Theorem 29. *Let k be fixed. Alg. 7 given as input k lists $L_1, \ldots, L_k \subset \mathsf{S}^n$ of the same size $|L|$, a target balanced configuration $C_{B,t} \in \mathbb{R}^{k\times k}$, a target length $0 < t < \sqrt{k}$, and $\varepsilon > 0$, outputs the list L_{out} of solutions to the Configuration problem. The expected running time of Alg. 7 is*

$$T = \widetilde{\mathcal{O}}\Big(|L| \cdot \max_{1\le i\le k-1} |L|^i \cdot \frac{(k^2-t^2)^i}{(k^2-k)^{i+1}} \cdot \Big(\frac{(k^2-k+(i-1)(t^2-k))^2}{k^2-k+(i-2)(t^2-k)}\Big)^{\frac{n}{2}}\Big). \tag{4.7}$$

In particular, for $t = 1$ and $|L_{\text{out}}| = |L|$ it holds that

$$T = \widetilde{\mathcal{O}}\Big(\Big(\frac{k^{\frac{1}{k-1}}}{k+1} \cdot \max_{1\le i\le k-1} k^{\frac{i}{k-1}} \cdot \frac{(k-i+1)^2}{k-i+2}\Big)^{\frac{n}{2}}\Big). \tag{4.8}$$

Remark 30. *In the proof below we also show that the expected running time of the BLS k-List algorithm from [BLS16] for $t = 1, |L_{\text{out}}| = |L|$ is*

$$T_{BLS} = \widetilde{\mathcal{O}}\Big(\Big(\frac{k^{\frac{k}{k-1}}}{(k+1)^2} \cdot \max_{1\le i\le k-1}\big(k^{\frac{i}{k-1}} \cdot (k-i+1)\big)\Big)^{\frac{n}{2}}\Big). \tag{4.9}$$

Proof of Thm. 29. The correctness of the algorithm is straightforward: let us associate the lists $L^{(i)}$ with a level i, where i indicates the number of filtering steps applied to L (we identify the input lists with the 0^{th} level: $L_i = L_i^{(0)}$). So for executing the filtering for the i^{th} time, we choose an $\vec{x}_i \in L_i^{(i-1)}$ that satisfies the condition $|\langle \vec{x}_i, \vec{x}_{i-1}\rangle - C_{i,i-1}| \le \varepsilon$ (for a fixed $\vec{x}_{i-1}$) and append to a previously obtained $(i-1)$-tuple $(\vec{x}_1, \ldots, \vec{x}_{i-1})$. Thus, on the last level, we put into L_{out} a k-tuple $(\vec{x}_1, \ldots, \vec{x}_k)$ that is a solution to the Configuration problem. To make the subscripts less confusing, we set $C = C_{\text{B,t}}$ throughout the proof.

Let us first estimate the size of the list $L_i^{(i-1)}$ output by the filtering process applied to the list $L_i^{(i-2)}$ for $i > 1$ (i.e. the left-most lists in Fig. 4.2a). Recall that all $\vec{x}_i \in L_i^{(i-1)}$ satisfy $|\langle \vec{x}_i, \vec{x}_j\rangle - C_{i,j}| \le \varepsilon$, $1 \le j \le i-1$. Then the *total* number of i-tuples $(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_i) \in L_1 \times L_2^{(1)} \times \ldots \times L_i^{(i-1)}$ considered by the algorithm is determined by the probability that in a random i-tuple, all pairs $(\vec{x}_j, \vec{x}_{j'}), 1 \le j, j' \le i$ satisfy the inner product constraints given by $C_{j,j'}$. This probability is given in Thm. 22 and, since the input lists are of the same size $|L|$, we have[2]

$$|L_1| \cdot |L_2^{(1)}| \cdot \ldots |L_i^{(i-1)}| = |L|^i \cdot \det(C[1\ldots i])^{\frac{n}{2}}, \tag{4.10}$$

where $\det(C[1\ldots i])$ denotes the i-th principal minor of C. Using Eq. (4.10) for two consecutive values

[2]Throughout this proof, the equations that involve list-sizes $|L|$ and running time T are assumed to have $\widetilde{\mathcal{O}}(\cdot)$ on the right-hand side. We omit it for clarity.

of i and dividing one equation by the other, we obtain

$$|L_{i+1}^{(i)}| = |L| \cdot \Big(\frac{\det(C[1\dots i+1]}{\det(C[1\dots i])}\Big)^{\frac{n}{2}}. \tag{4.11}$$

Note that these expected list sizes can be smaller than 1. This should be thought of as the inverse probability that the list is not empty. Since our target is a balanced configuration $C_{\mathrm{B,t}}$, the entries of the input Gram matrix are specified by Thm. 25 and, hence, we compute the determinants in the above quotient by applying Lemma 24 with $a = \frac{t^k-k}{k^2-k}$. Again, from the shape of the Gram matrix $C_{\mathrm{B,t}}$ and the equally-sized input lists, it follows that the filtered list on each level are of the same size: $|L_{i+1}^{(i)}| = |L_{i+2}^{(i)}| = \ldots = |L_k^{(i)}|$. Therefore, for all filtering levels $0 \le j \le k-1$ and for all $j+1 \le i \le k$,

$$\big|L_i^{(j)}\big| = |L| \cdot \Big(\frac{k^2-t^2}{k^2-k} \cdot \frac{k^2-k+j(t^2-k)}{k^2-k+(j-1)(t^2-k)}\Big)^{\frac{n}{2}}. \tag{4.12}$$

Now let us discuss the complexity of the algorithm. Clearly, the running time of Alg. 7 is (up to sub-exponential factors in n)

$$T = |L_1^{(0)}| \cdot (|L_2^{(0)}| + |L_2^{(1)}| \cdot (|L_3^{(1)}| + |L_3^{(2)}| \cdot (\ldots \cdot (|L_k^{(k-2)}| + |L_k^{(k-1)}|))).$$

Multiplying out and observing that $|L_k^{(k-2)}| > |L_k^{(k-1)}|$ (so creating a filtered list takes longer than enumerating over it), we may ignore the very last term and deduce that the total running time is (up to sub-exponential factors) given by

$$T = |L| \cdot \max_{1\le i\le k-1} |L^{(i-1)}| \cdot \prod_{j=1}^{i-1} |L^{(j)}|, \tag{4.13}$$

where $|L^{(j)}|$ is the size of any filtered list on level j (so we omit the subscripts). Consider the value $i_{\max}$ of i, where the maximum is attained in the above formula. The meaning of $i_{\max}$ is that the total cost over all loops to create the lists $L_j^{(i_{\max})}$ dominates the running time. At this level, the lists $L_j^{(i_{\max})}$ become small enough such that iterating over them (i.e. creating $L_j^{(i_{\max}+1)}$) does not contribute to the asymptotics. Plugging in Eqns. (4.10) and (4.11) into Eq. (4.13), we obtain

$$T = |L| \cdot \max_{1\le i\le k-1} |L|^i \Big(\frac{(\det C[1\dots i])^2}{\det C[1\dots(i-1)]}\Big)^{\frac{n}{2}}. \tag{4.14}$$

From Lemma 24, $\det C[1\dots i] = \Big(1+(i-1)\frac{t^2-k}{k^2-k}\Big)\Big(\frac{k^2-t^2}{k^2-k}\Big)^{i-1}$, giving us the desired expression for the running time.

For the case $t = 1$ and $|L_{\mathrm{out}}| = |L|$, the result of Cor. 27 on the size of the input lists $|L|$ yields a compact formula for the filtered lists:

$$\big|L_i^{(j)}\big| = \Big(k^{\frac{1}{k-1}} \cdot \frac{k-j}{k-j+1}\Big)^{\frac{n}{2}}. \tag{4.15}$$

Plugging this into either Eq. (4.13) or Eq. (4.14), the running time stated in Eq. (4.8) easily follows.

It remains to show the complexity of the BLS algorithm [BLS16], claimed in Rmk. 30. The algorithm is illustrated in Fig. 4.2b. We change the presentation of the algorithm to our configuration setting: in the original description, a vector $\vec{x}_i$ survives the filter if it satisfies $|\langle \vec{x}_i, \vec{x}_1 + \ldots + \vec{x}_{i-1}\rangle| \ge c_i$ for a predefined c_i (a sequence $(c_1,\ldots,c_{k-1}) \in \mathbb{R}^{k-1}$ is given as input to the BLS algorithm). Our concentration result (Thm. 22) also applies here and the condition $|\langle \vec{x}_i, \vec{x}_1 + \ldots + \vec{x}_{i-1}\rangle| \ge c_i$ is equivalent to a pairwise constraint on the scalar product $\langle \vec{x}_i, \vec{x}_j\rangle$ up to losing an exponentially small fraction of solutions. The optimal sequence of c_i's corresponds to the configuration $C_{\mathrm{B,t}}$ derived in Thm. 25.

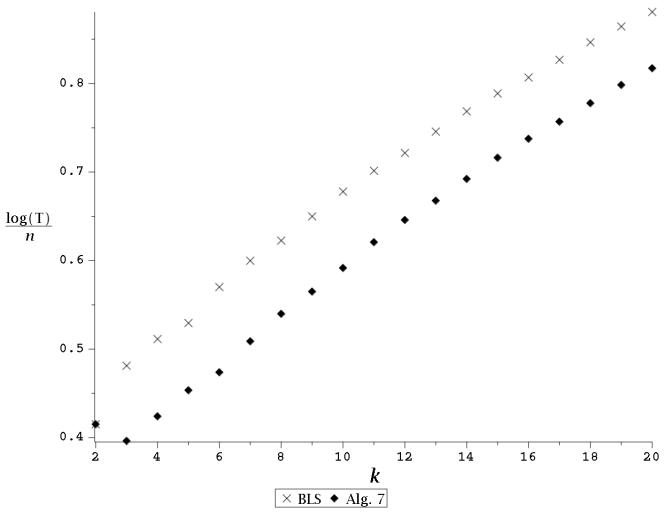


Fig. 4.3: Running time exponents scaled by $1/n$ for the target length $t = 1$. For $k = 2$, both algorithms are the Nguyen-Vidick sieve [NV08] with $\log(T)/n = 0.415$ (naive brute-force over two lists). For $k = 3$, Algorithm 7 achieves $\log(T)/n = 0.3962$, the BLS algorithm has $\log(T)/n = 0.4812$.

Indeed, Table 1 in [BLS16] matches $C_{\mathrm{B,t}}$ (case $t = 1$) exactly. So we may rephrase their filtering where instead of shrinking the list L_i by taking inner products with the sum $\vec{x}_1 + \ldots + \vec{x}_{i-1}$, we filter L_i by considering $\langle \vec{x}_i, \vec{x}_j \rangle$ for $1 \leq j \leq i-1$.

It follows that the filtered lists $L^{(i)}$ on level i are of the same size (in leading order) for both our and BLS algorithms. In particular, Eq. (4.11) holds for the expected list-sizes of the BLS algorithm.

The crucial difference lies in the construction of these lists. To construct the list $L_i^{(i-1)}$ in BLS, the filtering procedure is applied not to $L_i^{(i-2)}$ but to a (larger) input-list L_i. Hence, the running time of the BLS algorithm ignoring sub-exponential factors, is (cf. Eq. (4.13))

$$T_{BLS} = |L_1| \cdot (|L_2| + |L_2^{(1)}| \cdot (|L_3| + |L_3^{(2)}| \cdot (\ldots \cdot (|L_k| + |L_k^{(k-1)}|))) = |L|^2 \cdot \max_{1 \leq i \leq k-1} \cdot \prod_{j=1}^{i-1} |L^{(j)}|.$$

The result follows after substituting Eq. (4.15) into the above product. □

Both runtime expressions, Eq. (4.7) for Alg. 7 and Eq. (4.9) for the BLS algorithm, can be easily evaluated given k and $|L|$, see Fig. 4.3. The input list-sizes $|L|$ are chosen to guarantee $|L_{\mathrm{out}}| = |L|$ on expectation.

Our algorithm can be further improved by applying Locality-Sensitive Hashing techniques similar to [BDGL16] to shorten the lists prior to filtering. Unfortunately, the gain is very modest: for $k = 3, t = 1$, we can get the running time down from $2^{0.3962n+o(n)}$ to $2^{0.3717n+o(n)}$. The details on this extension are presented in [HK].

We remark that it seems quite challenging to analyze the k-List algorithms for a *non-fixed* k. Our approach heavily relies on the fact that k is a fixed constant allowing to suppress all the pre-factors in both run-times and list sizes in the $\widetilde{\mathcal{O}}_k(\cdot)$ notation. Indeed, taking a closer look at the suppressed pre-factors, we immediately notice that they depend at least *exponentially* on k (see, for example, the expression for $W_{n,k}$ in Thm. 22). Being able to let $k \to \infty$ would, however, greatly contribute to our understanding of complexity of SVP as it would enable us to compare sieving techniques with enumeration.

Further, we do not know what is an optimal choice of ε given k and $|L|$. In Sect. 4.1.5, we present our experimental results for Alg. 7, where we just try several ε's.

4.1.4 Approximate Shortest Vector Problem

In this section we expound the connection between the approximate k-List problem in Euclidean norm (Def. 17) and the approximate Shortest Vector Problem, appSVP_γ, for a constant approximation factor γ. Recall the definition of the latter problem.

On input, we are given a full-rank lattice $\mathcal{L}(B)$ described by a matrix $B \in \mathbb{R}^{n\times n}$ (with polynomially-sized entries) and some constant $\gamma > 1$. The task is to output a non-zero lattice vector $\vec{x} \in \mathcal{L}(B)$ s.t. $\|\vec{x}\| \leq \gamma\lambda_1(B)$. $\vec{x}$ is a solution to the approximate shortest vector problem. Since the solution is not unique, we are fine with any vector that satisfies the length condition.

The family of so-called sieving (or AKS) algorithms, described in the pioneering work of Ajtai, Kumar, and Sivakumar [AKS01], offers the best known to-date heuristic algorithm for appSVP_γ. The fact that this algorithm achieves a single-exponential running time and memory complexity was already stated in the original paper [AKS01], but a more precise analysis of the constant in the exponent has a long history. The result of Nguyen and Vidick in [NV08], stating the running time of order $2^{5.9n+o(n)}$, was later improved by Pujol and Stehlé to $2^{2.465n+o(n)}$ running time and $2^{1.42n+o(n)}$ space [PS09]. Under an assumption on the distribution of lattice-vectors under sieving, we are able to *heuristically* solve appSVP_γ in $2^{0.415n+o(n)}$ time and $2^{0.208n+o(n)}$ space. Finally, the currently best known running time of $2^{0.292n+o(n)}$ in [BDGL16] comes from a line of works based on the techniques from Locality-Sensitive Hashing. This is to be compared with the fastest *provable* appSVP_γ solver by Aggarwal et al. [ADRS15]. Based on the so-called discrete Gaussian sampling, this algorithm achieves $2^{n+o(n)}$-time and space complexity.

Practically, however, sieving algorithms are less attractive than Kannan's enumeration with running time of order $2^{\mathcal{O}(n\log n)}$. This fact is attributed to exponential memory requirement of sieving (and also to the advances in pruning techniques for enumeration). Recently, Bai, Laarhoven, Stehlé aiming at reducing memory, presented a variant of sieving algorithm with space complexity of $2^{0.1887n+o(n)}$ – an exponential improvement over the previous $2^{0.208n+o(n)}$-space sieving algorithm. Yet the gain comes at cost of increased running time: $2^{0.4812n+o(n)}$ as opposed to $2^{0.415n+o(n)}$ (for non-LSH sieving). To understand the BLS algorithm and how our improved k-List solver gives a faster sieving algorithm, we briefly explain how the AKS algorithm works.

The Nguyen-Vidick sieve. Sieving algorithms have two flavours: the Nguyen-Vidick sieve [NV08] and the Gauss sieve [MV10]. Both make $\mathrm{poly}(n)$ number of calls to the approximate 2-List solver. The Nguyen-Vidick sieve starts by sampling lattice-vectors $\vec{x} \in \mathcal{L}(B) \cap \mathcal{B}(\vec{0}, 2^{\mathcal{O}(n)} \cdot \lambda_1(B))$. This can be done using, for example, Klein's sampling procedure [Kle00] that outputs a lattice-vector of length not greater than $2^{\mathcal{O}(n)} \cdot \lambda_1(B)$. In the 2-List Nguyen-Vidick sieve, we sample many such lattice-vectors, put them in a list L, and search for *pairs* $\vec{x}_1 \times \vec{x}_2 \in L \times L$ s.t. $\|\vec{x}_1 \pm \vec{x}_2\| \leq (1-\varepsilon)\max\{\|\vec{x}_1\|, \|\vec{x}_2\|\}$ for some small $\varepsilon > 0$. The sum is put into L_{out}. The size of L is chosen in a way to guarantee $|L| \approx |L_{\text{out}}|$. The search for pairs is repeated over the list L_{out} once it is large enough.

The size of L determines the space complexity of the algorithm. A natural way to shorten the size of the input list L is, instead of looking for pairs, look for triples, or, more general, k-tuples that form a short sum. Indeed, it easily follows from Cor. 27 that the larger k is, the fewer vectors we should sample for the starting list L in order to expect $|L_{\text{out}}| = |L|$.

So the Nguyen-Vidick can be generalized to the search for k-tuples $\vec{x}_1, \ldots, \vec{x}_k \in L \times \ldots \times L$ s.t. $\|\vec{x}_1 + \ldots + \vec{x}_k\| \leq (1-\varepsilon)\max_{1\leq i\leq k}\{\|\vec{x}_i\|\}$. Now the sum $\vec{x}_1 + \ldots + \vec{x}_k$ is put into L_{out} and the search for k-tuples is repeated over L_{out}. Note that since with each new iteration we obtain vectors that are shorter by a constant factor $(1-\varepsilon)$, starting with $2^{\mathcal{O}(n)}$ approximation to the shortest vector (a property guaranteed by Klein's sampler applied to an LLL-reduced basis), we need only $\mathrm{poly}(n)$ iterations to find the desired $\vec{x} \in \mathcal{L}(B)$.

Naturally, we can apply our Alg. 7 to k copies of the list L to implement the search for short sums.

We do so by making a commonly used assumption: we assume the sampled lattice-vectors we put into the list lie uniformly on a spherical shell (on a very thin shell, essentially a sphere). The heuristic here is that it does not affect the behaviour of the algorithm. Intuitively, the discreteness of a lattice should not be 'visible' to the algorithm (at least not in the search for the approximate shortest vector; as soon as we see the discreteness, the vectors are already short enough). We refer to [NV08] for a more exhaustive discussion on this heuristic.

The advantage in using our Alg. 7 instead of the BLS k-List search within an appSVP_γ algorithm is straightforward: the search for a 'good' k-tuple is the routine that determines the complexity of the algorithm. So any improved algorithm for the approximate k-List problem immediately leads to a better appSVP_γ algorithm.

Gauss sieve. More interestingly, our improved k-List algorithm for $k \geq 3$ can as well be used within the Gauss sieve, which is known to perform faster in practice than the Nguyen-Vidick sieve. Let us briefly recall the Gauss sieve algorithm.

An iteration of the original 2-Gauss sieve as described in [MV10], searches for pairs $(\vec{p}, \vec{v})$ s.t. $\|\vec{p}+\vec{v}\| < \max\{\|\vec{p}\|, \|\vec{v}\|\}$, where $\vec{p} \in \mathcal{L}(B)$ is *fixed*, $\vec{v} \in L \subset \mathcal{L}(B)$, and $\vec{p} \neq \vec{v}$. Once such a pair is found and $\|\vec{p}\| > \|\vec{v}\|$, we reduce $\vec{p}$ by setting $\vec{p}' \leftarrow \vec{p}+\vec{v}$ and proceed with the search over $(\vec{p}', \vec{v})$, otherwise if $\|\vec{p}\| < \|\vec{v}\|$, we delete $\vec{v} \in L$ and store the sum $\vec{p}+\vec{v}$ as $\vec{p}$-input point for the next iteration. Once no pair is found, we add $\vec{p}'$ to L. On the next iteration, the search is repeated with another $\vec{p}$ which is obtained either by reducing some previously deleted $\vec{v} \in L$, or by sampling from $\mathcal{L}(B)$. The idea is to keep only those vectors in L that *cannot* form a pair with a shorter sum. Bai, Laarhoven, and Stehlé in [BLS16], generalize it to the k-Gauss sieve by keeping only those vectors in L that do not form a shorter k-sum. In the language of configuration search, we look for configurations $(\vec{p}, \vec{v}_1, \ldots, \vec{v}_{k-1}) \in \vec{p} \times L \times \ldots \times L$ where the first point is fixed, so we apply our Alg. 7 on $k-1$ (identical) lists.

Pseudo-code for 3-Gauss sieve is given in Alg. 8 below. We assume the approximation $\gamma\lambda_1(B)$ is given as input. The main procedure (first lines 1-11) is exactly the same as in the original algorithm of Micciancio-Voulgaris. The difference is in the main subroutine TripleReduce() that implements the approximate 3-List search with the first vector in a triple being $\vec{p}$. The list L is always kept sorted so that at the end of the procedure the shortest vector in the list is $L[1]$. The algorithm can be easily generalized to the larger k, but we decided to present $k = 3$ case as the most practically relevant. The experimental results on 3-Gauss sieve are given in the next section.

Algorithm 8 3-Gauss sieve

Input: $B \in \mathbb{R}^{n \times n}$ - an LLL-reduced lattice basis, $\gamma\lambda_1(B)$ - the desired approximation factor, $\varepsilon > 0$
Output: $\vec{x} \in \mathcal{L}(B)$ s.t. $\|\vec{x}\| \leq \gamma\lambda_1(B)$

$L \leftarrow \{\}$ ▷ Sorted list of triple-reduced vectors
$S \leftarrow \{\}$ ▷ Stack of vectors
while $(L[1] > \gamma\lambda_1(B))$ **do**
  if S is not empty **then**
    $\vec{p} \leftarrow S$.pop()
  else
    $\vec{p} \leftarrow \text{KleinSample}(B)$ ▷ Sample a vector from $\mathcal{L}(B)$
  $\vec{p}' \leftarrow \textsc{TripleReduce}(\vec{p}, L, s)$
  if $\vec{p}' \neq \vec{0}$ **then**
    $L \leftarrow L \cup \{\vec{p}'\}$
return $L[1]$

function $\textsc{TripleReduce}(\vec{p}, L, S)$
  while ($\vec{p}$ cannot be reduced) **do** ▷ Try to reduced $\vec{p}$ first
    $L' \leftarrow \textsc{Filter}(\vec{p}, L, \varepsilon)$
    for all $\vec{v}_1, \vec{v}_2 \in L' \times L'$ **do**
      if $\|\vec{p} \pm \vec{v}_1 \pm \vec{v}_2\| < \|\vec{p}\|$ **then**
        $\vec{p} \leftarrow \vec{v}_1 \pm \vec{v}_2$ ▷ the sign should satisfy the If-condition
  $L' \leftarrow \textsc{Filter}(\vec{p}, L)$ ▷ with a new reduced $\vec{p}$
  for all $\vec{v}_1, \vec{v}_2 \in L' \times L'$ **do**
    if $\|\vec{p} \pm \vec{v}_1 \pm \vec{v}_2\| < \max\{\|\vec{v}_1\|, \|\vec{v}_2\|\}$ **then**
      $\max\{\|\vec{v}_1\|, \|\vec{v}_2\|\} \leftarrow \vec{p} \pm \vec{v}_1 \pm \vec{v}_2$
  return $\vec{p}$

function $\textsc{Filter}(\vec{p}, L, \varepsilon)$ ▷ Filter w.r.t. balanced configuration $C_{\mathrm{B,t}}$
  $L' \leftarrow \{\}$
  for all $\vec{v} \in L$ **do**
    if $\left|\frac{\langle \vec{v}, \vec{p} \rangle}{\|\vec{v}\|\|\vec{p}\|}\right| \geq \frac{1}{3} - \varepsilon$ **then**
      $L' \leftarrow L' \cup \{\vec{v}\}$
  return L'

4.1.5 Experimental results

We implement the 3-Gauss sieve Algorithm 8 in collaboration with S. Bai [Bai16]. The implementation is based on the program developed by Bai, Laarhoven, and Stehlé in [BLS16]. The experiments are run on the Ruhr University C3 cluster [CCC]. The results are presented in Table 4.2.

Lattice bases are generated by the SVP challenge generator [SVP]. It produces a lattice generated by the columns of the matrix

$$B = \begin{pmatrix} p & x_1 & \dots & x_{n-1} \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix},$$

where p is a large prime, and $x_i < p$ for all i. Lattices of this type are random in the sense of Goldstein and Mayer [GM06].

For all the dimensions except $n = 80$, the bases are preprocessed with BKZ reduction of block-size 20. For $n = 80$, the block-size is 30. For our input lattices, we do not know their minimum λ_1. The algorithm terminates when it finds many linearly dependent triples $(\vec{p}, \vec{v}_1, \vec{v}_2)$. It means that at some point TripleReduce() starts outputting $\vec{0}$. We set a counter for such an event and terminate the algorithm once this counter goes over a pre-defined threshold. The intuition behind this idea is straightforward: at some point the list L will contain very short basis-vectors and the remaining list-vectors will be their linear combinations. Trying to reduced the latter will ultimately produce the zero-vector. The same termination condition was already used in [MLB15], where the authors experimentally determine a threshold of such 'zero-sum' triples.

Up to $n = 64$, the experiments are repeated 5 times (i.e. on 5 random lattices), for the dimensions less than 80, 3 times. For the running times and the list-sizes presented in the table below, the average is taken. For $n = 80$, the experiment was performed once.

Our tests confirm a noticeable speed-up of the 3-Gauss sieve when our Configuration Search Algorithm 7 is used. Moreover, as the analysis suggests (see Fig. 4.3), our algorithm outperforms the naive 2-Gauss sieve while using much less memory.

Another interesting aspect of the algorithm is the list-sizes when compared with BLS. Despite the fact that asymptotically the size of the list $|L|$ is the same for our and for the BLS algorithms, in practice our algorithm requires a longer list (cf. the right numbers in each column). This is due to the fact that we filter out a larger fraction of solutions. Also notice that increasing ε – the approximation to the target configuration – we achieve an additional speed-up. This becomes obvious once we look at the Filter() procedure: allowing for a smaller inner-product throws away less vectors, which in turn results in a shorter list L. For the range of dimensions we consider, the optimum is attained at $\varepsilon = 0.3$.

	2-sieve	BLS 3-sieve	Alg. 8, 3-sieve			
			$\varepsilon = 0.0$	$\varepsilon = 0.015$	$\varepsilon = 0.3$	$\varepsilon = 0.4$
n	T, $\lvert L\rvert$	T, $\lvert L\rvert$	T, $\lvert L\rvert$	T, $\lvert L\rvert$	T, $\lvert L\rvert$	T, $\lvert L\rvert$
60	1.38e3, 13257	1.02e4, 4936	1.32e3, 7763	1.26e3, 7386	1.26e3, 6751	**1.08e3, 6296**
62	2.88e3, 19193	1.62e4, 6239	2.8e3, 10356	3.1e3, 9386	**1.8e3, 8583**	2.2e3, 8436
64	8.64e3, 24178	5.5e4, 8369	5.7e3, 13573	3.6e3, 12369	**3.36e3, 11142**	4.0e4, 10934
66	1.75e4, 31707	9.66e4, 10853	1.5e4, 17810	1.38e4, 16039	**9.1e3, 14822**	1.2e4, 14428
68	3.95e4, 43160	2.3e5, 14270	2.34e4, 24135	2.0e4, 21327	**1.68e4, 19640**	1.86e4, 18355
70	6.4e4, 58083	6.2e5, 19484	6.21e4, 32168	3.48e5, 26954	**3.3e4, 25307**	3.42e4, 24420
72	2.67e5, 77984	1.2e6, 25034	7.6e4, 40671	7.2e4, 37091	**6.16e4, 34063**	6.35e4, 34032
74	3.45e5, 106654	—	2.28e5, 54198	2.08e5, 47951	**2.02e5, 43661**	2.03e5, 40882
76	4.67e5, 142397	—	3.58e5, 71431	2.92e5, 64620	**2.42e5, 56587**	2.53e5, 54848
78	9.3e5, 188905	—	—	—	**4.6e5, 74610**	4.8e5, 70494
80	—	—	—	—	**9.47e5, 98169**	9.9e5, 98094

Tab. 4.2: Experimental results for k-tuple Gauss sieve. The running times T are given in seconds, $|L|$ is the maximal size of the list L. ε is the approximation parameter for the subroutine FILTER() of Alg. 8. The best running-time per dimension is type-set bold.

4.2 Approximate SVP on a q-ary lattice

In this section we present a combinatorial algorithm that solves the approximate shortest vector problem, appSVP_γ, on a q-ary lattice. As opposed to the algorithm from the previous section where the approximation factor γ was a constant, here γ is polynomial in the lattice-dimension, i.e. we look for a vector $\vec{v}$ from $\mathcal{L}_q \subset \mathbb{Z}_q^n$ s.t. $\|\vec{v}\| \leq \text{poly}(n)\lambda_1(\mathcal{L}_q)$.

An algorithm for appSVP_γ with a polynomial approximation factor gives a way to solve the so-called Short Integer Solution Problem (SIS) – the problem introduced by Ajtai in [Ajt96] that serves as the foundation for a variety of cryptographic primitives. Given a matrix $A \in \mathbb{Z}_q^{n \times m}$ composed column-wise from uniformly chosen $\vec{a}_i \in \mathbb{Z}_q^n$, SIS asks to find a short $\vec{v} \in \mathbb{Z}_q^m$ s.t. $A\vec{v} = 0 \mod q$. The length condition is specified by an input parameter β, i.e. the output $\vec{v}$ must satisfy $\|\vec{v}\| \leq \beta$, where $\beta = \text{poly}(n)$ and the degree of the polynomial depends on the modulus q. Note that we are not interested in the trivial solution $\vec{v} = (q, 0, \ldots, 0)$. Also notice that if we ask for $\vec{v} \in \{0,1\}^n$, SIS becomes the vectorial Subset Sum Problem.

To see the connection between SIS and the approximate SVP, let us consider an m-dimensional q-ary lattice $\mathcal{L}_q^\perp(A) = \{\vec{x} \in \mathbb{Z}^m \colon A\vec{x} = 0 \mod q\}$. A solution to appSVP_γ on $\mathcal{L}_q^\perp(A)$ is a vector $\vec{v} \in \mathcal{L}_q^\perp(A)$ of length $\|\vec{v}\| \leq \gamma\lambda_1(\mathcal{L}_q^\perp(A))$ and therefore, it is a solution to the SIS problem when γ is appropriately chosen. From Minkowski's bound we know that $\lambda_1(\mathcal{L}_q^\perp(A) \leq \sqrt{m}q^{n/m}$. Hence, if we set $q = \mathcal{O}(n^{\mathsf{c}_q})$ (as we did in Chap. 3 for the analysis of LWE), $\gamma = \mathcal{O}(n^{\mathsf{c}_\gamma})$ for constants $\mathsf{c}_q > 1, \mathsf{c}_\gamma$, and take $m = \Theta(n)$, a solution for appSVP_γ will be a vector of length $\|\vec{v}\| \leq n^{\mathsf{c}_\gamma + \mathsf{c}_q/2 + 1/2}$. Values of c_γ stem from the connection of SIS to the worst-case lattice-problems. Since Ajtai's proof, the constant has been improved from the original $\mathsf{c}_\gamma = 8 + o(1)$ [Ajt96] down to $\mathsf{c}_\gamma = 2.5 + o(1)$ in [Mic05] and, finally, to $\mathsf{c}_\gamma = 1 + o(1)$ in [MR04]. In the language of SIS, c_γ is known as Ajtai's connection factor.

We have a natural restriction on c_γ that comes from the fact that we want to avoid trivial solutions of length q, namely $\mathsf{c}_\gamma < \mathsf{c}_q/2 - 1/2$.

Notice that we already mentioned appSVP_γ in Sect. 3.1.5 when we discussed the so-called `Dual` attack on LWE. The name of the attack comes from the fact that the two problems, LWE and SIS, are 'dual' to each other. What it means is that the LWE problem – the decoding problem on $\mathcal{L}(A^\mathsf{t})$ – can be solved using a SIS oracle for A (or, equivalently, an oracle for appSVP_γ on $\mathcal{L}_q^\perp(A)$) as we have already seen in Alg. 4, Chap. 3. The two lattices, $\mathcal{L}(A^\mathsf{t})$ and $\mathcal{L}_q^\perp(A)$, are dual to each other up to scaling by q: $\mathcal{L}_q(A^\mathsf{t})^* = \frac{1}{q}\mathcal{L}_q^\perp(A)$, $\mathcal{L}_q^\perp(A)^* = \frac{1}{q}\mathcal{L}_q(A^\mathsf{t})$. We refer the reader to [Mic10] for more interesting outcomes of this duality.

In the following sections we present two combinatorial algorithms for appSVP_γ on $\mathcal{L}_q^\perp(A)$ in case $\gamma = \text{poly}(n)$. The second algorithm has a better constant in the running time exponent. Next, we compare our algorithms with the `BKZ` reduction run on $\mathcal{L}_q^\perp(A)$ when the block-size β is chosen s.t. the first vector of the reduced basis is a solution to appSVP_γ. We conclude that our improved algorithm outperforms `BKZ` for some values of $\mathsf{c}_q, \mathsf{c}_\gamma$ even when `BKZ` is instantiated with the best known heuristic SVP oracle.

4.2.1 An algorithm for appSVP_γ on a q-ary lattice

In this section, we present a combinatorial algorithm that on input $A \in \mathbb{Z}_q^{n \times 2n}$ and $\mathsf{c}_\gamma < \mathsf{c}_q/2 - 1/2$, outputs a vector $\vec{v} \in \mathcal{L}_q^\perp(A) \subset \mathbb{Z}_q^{2n}$ s.t. $\|\vec{v}\| \leq n^{\mathsf{c}_\gamma + \mathsf{c}_q/2 + 1/2}$. Notice that we set the lattice dimension m as $m = 2n$. This choice simplifies the exposition and is necessary for the improved algorithm described in Sect. 4.2.3. Case $m = \mathsf{c}_m n$ for $\mathsf{c}_m = \Theta(1)$ is considered in Rmk. 32.

The algorithm is actually a combinatorial `BKW`-type algorithm for LWE due to Kirchner-Fouque [KF15] and Guo et al. [GJS15] adapted to the appSVP_γ problem. We now give its high-level overview.

The idea is to split the dimension of the lattice, $2n$, into k blocks $d_1, \ldots, d_k$, i.e. $\sum_{i=1}^k d_i = 2n$. We also choose k positive values $R_1, \ldots, R_k$ where $R_i < q/2$ for all i. The algorithm proceeds in k steps.

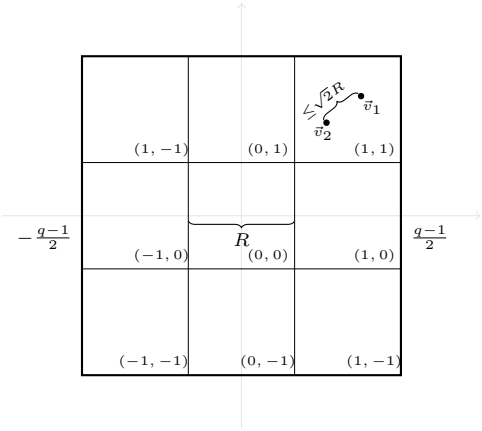


Fig. 4.4: Bucketing on a 2-dimensional q-ary lattice. Each small cube of length R gets its two-dimensional label. The vectors $\vec{v}_1$, $\vec{v}_2$ appear in the same bucket $(1, 1)$ and, hence, the ℓ_2-norm of their difference is bounded by $\sqrt{2}R$.

On Step 1, we search for pairs of lattice-vectors $(\vec{v}_1, \vec{v}_2)$ s.t.

$$\left\lfloor \frac{[\vec{v}_1]_1^{d_1}}{R_1} \right\rceil = \left\lfloor \frac{[\vec{v}_2]_1^{d_1}}{R_1} \right\rceil^3.$$

In other words, we split our q-ary cube $[-\frac{q-1}{2}, \frac{q-1}{2}]^{d_1}$ (assume q is odd, the algorithm can be easily adapted for an even q) into many smaller cubes $[-R_1, R_1]^{d_1}$ and search for pairs $(\vec{v}_1, \vec{v}_2)$ that on their first d_1 coordinates lie in the same cube. In our algorithm we set $R_1 = n^{o(1)} \ll q$, and we can adjust our choice for R_1 s.t. the small cubes split the large cube evenly. See Fig. 4.4 for a 2-dimensional example.

Once two such vectors are found, we subtract one from the other and put the result into list L_1. Important is that we can bound the ℓ_∞-norm of elements in L_1: on average, $\|[\vec{v}_1 - \vec{v}_2]_1^{\mathrm{d}_1}\|_\infty \leq \sqrt{2}R_1$. The output of Step 1 are many vectors with *bounded* ℓ_∞-norm on their first d_1 coordinates. On Step 2, we use vectors from L_1 to search for pairs that lie in the same $[-R_2, R_2]^{\mathrm{d}_2}$ cube on their $d_1 + 1, \ldots, d_2$ coordinates analogously to Step 1. The output of Step 2 is a list L_2 with vectors bounded in ℓ_∞-norm on coordinates $1, \ldots, d_1$ and $d_1 + 1, \ldots, d_2$.

Repeating this procedure for all k steps, we end up with lattice-vectors for which we can bound their ℓ_∞-norm on all the $2n$ coordinates and hence, their Euclidean norm. From the upper-bound on the length of the output, we find an optimal on k. We defer the discussion on how to set R_i's and k to the next section.

There is one simple trick which greatly improves the running time of the algorithm. We can write our input matrix $A \in \mathbb{Z}_q^{n \times 2n}$ as $A = [A_1|A_2]$ where $A_i \in \mathbb{Z}_q^{n \times n}$. With high probability, we have that A_1 is invertible mod q allowing us to write $A = [\mathbb{1}_n|A']$ where $A' = A_1^{-1}A_2 \bmod q$. Essentially this procedure brings a q-ary code generated by A to a systematic form. It is easy to verify that a basis for $\mathcal{L}_q^\perp(A)$ is of the form (cf. with the basis B for $\mathcal{L}_q(A^\mathsf{t})$ given in Eq. (2.7)):

$$D = \begin{pmatrix} -A' & q\mathbb{1}_n \\ \\ \mathbb{1}_n & 0 \end{pmatrix}. \tag{4.16}$$

[3]Recall the notation $[\vec{x}]_i^j = x_i \ldots x_j$ for $i \leq j$.

Algorithm 9 appSVP_γ on a q-ary lattice

Input: D – a basis for the lattice $\mathcal{L}_q^\perp(A) \subset \mathbb{Z}_q^{2n}$ defined as in Eq. (4.16), $\gamma = n^{\mathsf{c}_\gamma}$ – the approximation factor, $\mathsf{c}_\gamma > 0$

Output: L_k – list of vectors from $\mathcal{L}_q^\perp(A)$ with vectors of norm $\|\vec{v}\| \leq n^{\mathsf{c}_\gamma + \mathsf{c}_q/2 + 1/2}$;

Set the sieving bounds R_i as $R_1 = n^{o(1)}$ and $R_i = \sqrt{2}^{i-1} R_1$ for $i \geq 2$.
Set the lengths of blocks d_i as in Eq. (4.18) and the boundaries of each block $(l_{i-1}, \ldots, l_i)$ s.t. $l_i - l_{i-1} = d_i$ and $l_k = 1, l_0 = n$.
repeat ▷ Create the list L_0
    Choose $\vec{x} \in \mathbb{Z}_q^{2n}$ s.t. $\|[\vec{x}]_{n+1}^{2n}\|_\infty \leq R_1$
    $L_0 \leftarrow L_0 \cup \{D\vec{x} \bmod q\}$
until L_0 is large enough
$T \leftarrow \emptyset$ ▷ Initialize an array T indexed by buckets
for all $i = 1 \ldots k$ **do**
    for all $\vec{v} \in L_{i-1}$ **do**
        $b \leftarrow \left\lfloor \frac{[\vec{v}]_{l_i}^{l_{i-1}}}{R_i} \right\rceil$ ▷ Find the bucket for $\vec{v}[l_i, \ldots, l_{i-1}]$
        if $T[b] = \emptyset$ **then**
            $T[b] \leftarrow \vec{v}$
        else
            $L_i \leftarrow L_i \cup \{T[b] - \vec{v}\}$
            $T[b] \leftarrow \emptyset$
return L_k

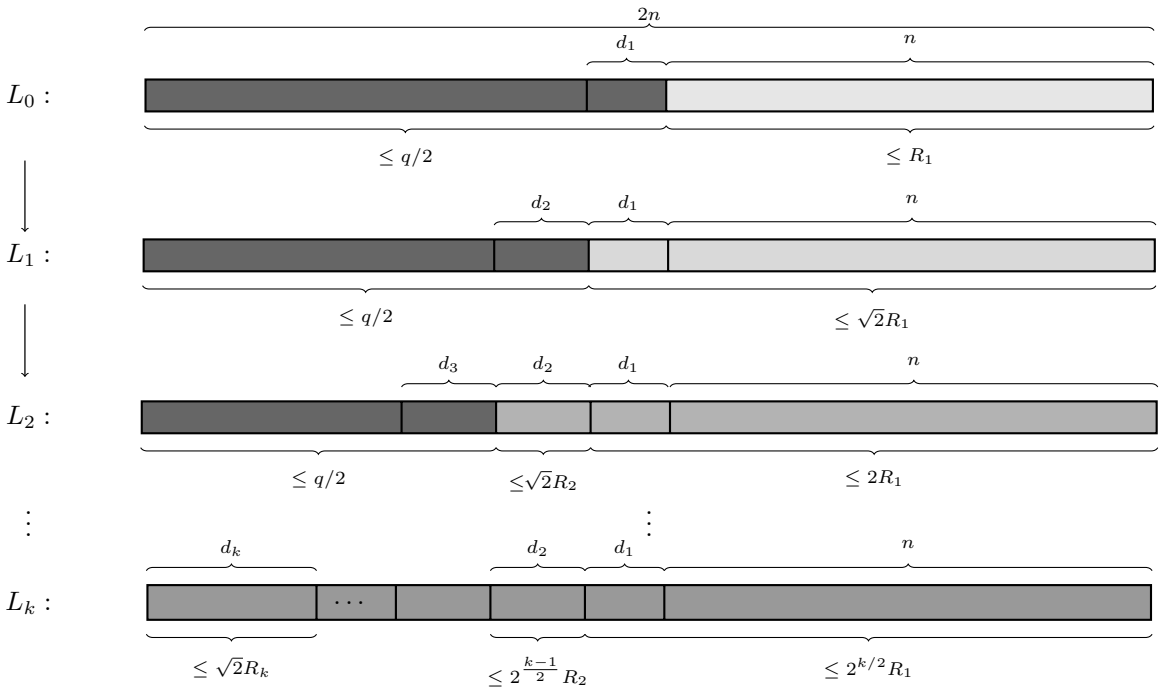


Fig. 4.5: Visualization of Alg. 9. Each horizontal rectangle represents a form of a vector from the input-list L_{i-1} on step i for $i = 1, 2, 3$ (counting from top to bottom) and a vector from the output list L_k (the lower-most rectangle). The vectors are of dimension $2n$. Labels of the upper brackets denote the length of the blocks, while labels of the lower brackets denote the ℓ_∞-norm of the corresponding block. The darker the shading for a block is, the larger its ℓ_∞-norm. Note that the algorithm chooses the bounds R_i s.t. the ℓ_∞-norm on the previous (right-hand side) blocks is the same as on the currently considered block (i.e. $R_i = \sqrt{2}R_{i-1}$). With such a choice, the contribution to the expected norm of vectors from the final list L_k is equal from each block.

Now we use this basis to generate the lattice-vectors to perform the initial search. We will choose $\vec{x} = (x_1, \ldots, x_n, x_{n+1}, x_{2n}) \in \mathbb{Z}_q^{2n}$ and produce vectors

$$D\vec{x} \bmod q = (y_1, \ldots, y_n, x_{n+1}, \ldots, x_{2n})^{\mathrm{t}}.$$

This way we can already bound the ℓ_∞-norm of the vectors on the right-most n coordinates by choosing $(x_{n+1}, \ldots, x_n)$, say, less than R_1 (as if we would have already bucketed the right-hand side). We put vectors of this form in our starting list L_0. Elements from this list allow us to perform our 'cube-bucketing' of the remaining left n-coordinates only as opposed to $2n$.

Our appSVP_γ algorithm can be easily formulated as an algorithm for a k-List problem in the sense of Def. 16: given 2^k copies of L_0, find k-tuples $(\vec{v}_1, \ldots, \vec{v}_k) \in L_0 \times \ldots \times L_0$, s.t. $\|\vec{v}_1 \pm \ldots \pm \vec{v}_k\| \leq n^{\mathsf{c}_\gamma + \mathsf{c}_q/2 + 1/2}$. On Step 1, the algorithm groups 2^k lists L_0 into 2^{k-1} pairs of lists and from each list-pair (L_0, L_0) searches for $(\vec{v}_i, \vec{v}_{i+1})$ that appear in the same 'bucket' on their first d_1 coordinates. Once found, $(\vec{v}_i - \vec{v}_{i+1})$ is put into L_1. At the end, we have 2^{k-1} copies of the list L_1. The algorithm terminates with one copy of L_k that contains a vector with bounded ℓ_∞-norm. By setting k appropriately, we can guarantee that the Euclidean length of this vector is bounded as desired.

Below we describe the algorithm in pseudo-code and in Fig. 4.5. The array T in pseudo-code serves as a look-up table: on step i, it is indexed by d_i-dimensional vectors (buckets) b and whenever we find a vector $\vec{v}$ s.t. $\left\lfloor \frac{[\vec{v}_1]_1^{d_1}}{R_1} \right\rceil = b$, we look up whether $T[b]$ is empty or not. In the latter case, the collision is found and a new vector is added to L_i.

4.2.2 Analysis

It is reasonable to assume that the above algorithm for appSVP_γ on a q-ary lattice, as all known BKW-type algorithms for LWE, has both running time and memory complexity of the form $2^{(\mathsf{c}+o(1))n}$ for some constant c. The goal of this section is to determine c as a function of input parameters: c_γ, where $\gamma = \mathcal{O}(n^{\mathsf{c}_\gamma})$, and c_q, where $q = \mathcal{O}(n^{\mathsf{c}_q})$. We consider average-case instances, and our analysis will show the *expected* running time and memory.

Let us elaborate more on the running-time/memory trade-off achieved by the algorithm. Recall that on step i, one entry of table T represents one bucket which is one of the small cubes $[-R_i/2, R_i/2]^{\ell_i}$ inside a large cube $[-\frac{q-1}{2}, \frac{q-1}{2}]^{\ell_i}$ (see Fig. 4.4). We expect to have all entries of T be filled after we have bucketed (#buckets)-many lattice vectors from L_{i-1} (here we use the fact that elements from L_{i-1}, subjected on block ℓ_i, are uniformly distributed in $[-\frac{q-1}{2}, \frac{q-1}{2}]^{\ell_i}$). Hence, after bucketing $(2 \cdot$#buckets$)$-many lattice vectors from L_{i-1}, we expect to put (#buckets)-many lattice vectors into L_i. Overall, the lists get shorter by at least a factor of 2 per level. After k levels, we expect $|L_k| = \Theta(2^{-k}|L_0|)$.

The analysis below reveals $k = \Theta(\log n)$, and hence, the output list L_k is expected to be only poly(n)-times shorter than the initial list L_0. We shall see in the proof that the number of buckets on each level will be exponential in n, hence, to find even one collision on step i, we need exponentially many lattice vectors in the list L_{i-1}. So we ignore poly(n)-factors coming from the fact that we lose approximately half of the list on each step. Also, the amount of computations we perform per bucket is only $\mathcal{O}(n)$ as we add up two n-dimensional vectors. Thus, both the expected running time and memory complexity are equal (up to poly(n)-factors) to the number of buckets. Exactly the same arguments apply to BKW algorithms for LWE.

Intuitively, it would be beneficial to take the number of steps k large as it leads to shorter block-lengths ℓ_i's which, in turn, speeds up the collision-finding. However, we have to set k as low as $k = \Theta(\log n)$ where the constant in Θ-notation will ultimately depend on c_γ and c_q. This bound comes from the fact that each time we perform addition of two vectors that happen to lie in the same bucket, the ℓ_∞-norm of the resulting vector on *already considered* blocks $\ell_1, \ldots, \ell_{i-1}$ increases (on average) by a factor of $\sqrt{2}$. So shortening the ℓ_∞-norm from q down to $\sqrt{2}R_i$ on a block enlarges the ℓ_∞-norm

of the vector by $\sqrt{2}$ on the right-hand side blocks. This growth is depicted in Fig. 4.5. At the end, on the coordinate block $[d_{i-1},\ldots,d_i]$ of length ℓ_i, we have $\|[\vec{v}]_{d_i}^{d_{i-1}}\|_\infty \le 2^{\frac{k-i+1}{2}} R_i$ for $\vec{v}\in L_k$. Hence our choice for k is restricted by the upper bound of the Euclidean length of $\vec{v}$ we should output. This situation should be compared with the error-growth in the BKW algorithm for LWE that puts a bound on k of the same order.

In the proof below we show how to set the block-lengths ℓ_i's, the ℓ_∞-norm bounds R_i's, and the number of steps k.

Theorem 31. *Algorithm 9 on input (1) a lattice-basis $D\in\mathbb{Z}_q^{2n\times 2n}$ as in Eq. (4.16) for the lattice $\mathcal{L}_q^\perp(A)$ with $q=\mathcal{O}(n^{\mathsf{c}_q})$ and (2) an approximation factor $\gamma=\mathcal{O}(n^{\mathsf{c}_\gamma})$, outputs a vector $\vec{v}\in\mathcal{L}_q^\perp(A)$ of length $\|\vec{v}\|\le n^{\mathsf{c}_\gamma+\mathsf{c}_q/2+1/2}$ in expected time $T(\mathsf{appSVP}_\gamma)=2^{(\mathsf{c}+o(1))n}$, where*

$$\mathsf{c}=\frac{1}{2\ln\left(\frac{\mathsf{c}_q}{\mathsf{c}_q/2-\mathsf{c}_\gamma}\right)}. \tag{4.17}$$

Proof. The expected running time to fill up all the buckets on step i and, thus, to create the list L_i is determined by the number of buckets or, equivalently, the number of ℓ_i-dimensional cubes $[-R_i/2,R_i/2]^{\ell_i}$ that 'fit' inside the large cube $[-\frac{q-1}{2},\frac{q-1}{2}]$. This number is given by the fraction of the two volumes:

$$\mathbb{E}[\#\text{buckets on level } i]=\frac{\mathrm{vol}([-\frac{q-1}{2},\frac{q-1}{2}]^{d_i})}{\mathrm{vol}([-R_i/2,R_i/2]^{d_i})}=\Theta\Big(\Big(\frac{q}{R_i}\Big)^{d_i}\Big).$$

This is (up to poly(n)-factors) the expected running time of the inner for-loop (line 9). As we show below, the number of steps k will be of size $k=\Theta(\log n)$ and hence, the outer for-loop on line 8 in Alg. 9 contributes to the running time only by a poly(n)-factor.

Thus, asymptotically, we have $\left(\frac{q}{R_i}\right)^{d_i}=2^{\mathsf{c}n}$, from where it follows that

$$d_i=\frac{\mathsf{c}n}{\log q-\log R_i}. \tag{4.18}$$

Additionally, we have $\sum_{i=1}^k d_i=n$. We shall conclude on c from these two equations.

But before doing that let us get an upper-bound for k. The expected length of vector $\vec{v}$ in the list L_k is upper-bounded as $\|\vec{v}\|\le\sqrt{2R_k^2d_k+4R_{k-1}^2d_{k-1}+\ldots+2^{k-1}R_2^2d_2+2^kR_1^2d_1+2^kR_1^2n}$. It is easy to verify (see also Fig. 4.5) that if we set $R_{i+1}=\sqrt{2}R_i$, the first k summands in our bound on $\|\vec{v}\|$ contribute to the total sum equally. Finally, we obtain $\|\vec{v}\|\le\sqrt{22^kR_1^2n}$. This bound should be less than $n^{\mathsf{c}_\gamma+\mathsf{c}_q/2+1/2}$. If we set the first bound R_1 as small as $R=n^{o(1)}$, the inequality $\sqrt{22^kR_1^2n}\le n^{\mathsf{c}_\gamma+\mathsf{c}_q/2+1/2}$ leads to $k\le 2(\mathsf{c}_\gamma+\mathsf{c}_q/2+o(1))\log n$. We take the upper bound as the value for k.

Since we set $R_i=\sqrt{2}^{i-1}R_1$, we now can compute the sum $\sum_{i=1}^k d_i$ as

$$\sum_{i=1}^{k}\frac{\mathsf{c}n}{\log\frac{q}{R_1}-\frac{1}{2}(i-1)}\le\int_{i=0}^{k-1}\frac{\mathsf{c}n\mathrm{d}i}{\log(\frac{q}{R_1})-\frac{1}{2}i}=-2\mathsf{c}n\Big(\ln(\log\tfrac{q}{R_1}-\tfrac{1}{2}i)\Big|_0^{k-1}\Big)=2\mathsf{c}n\ln\Big(\frac{\log\frac{q}{R_1}}{\log\frac{q}{R_1}-\frac{1}{2}(k-1)}\Big).$$

The error that comes from approximating the sum by the integral contributes to the $o(1)$-term in the exponent. From the facts that all d_i's sum up to n, $k=2(\mathsf{c}_\gamma+\mathsf{c}_q/2)\log n$, and $R_1=n^{o(1)}$, we obtain

$$\mathsf{c}=\frac{1}{2\ln\left(\frac{\mathsf{c}_q}{\mathsf{c}_q/2-\mathsf{c}_\gamma}\right)}+o(1).$$

□

Remark 32. *From the proof above, it is easy to deduce that for* $m = \mathsf{c}_m \cdot n$, $\mathsf{c}_m = \Theta(1)$

$$\mathsf{c} = \frac{\mathsf{c}_m - 1}{2\ln\left(\frac{\mathsf{c}_q}{\mathsf{c}_q(1-1/\mathsf{c}_m)-\mathsf{c}_\gamma}\right)}.$$

We conclude the discussion on the algorithm with a couple of remarks.

- **Connection to BKZ algorithms.** An important property of our q-ary lattice $\mathcal{L}_q^\perp(A)$ we are exploiting in this algorithm is the *orthogonal* sub-lattice $q\mathbb{1}_n \subset \mathcal{L}_q^\perp(A)$. We can view each bucketing step as projection of vectors onto the q-ary vectors $\{(q,0,\ldots,0),\ldots(0,0,\ldots,q)\}$ first, then performing the summation, and finally 'lifting' the result. Since the sub-lattice is orthogonal, the lifting is simply a coordinate-wise mod$q$ operation. This allows us not to worry about the ℓ_∞-norm on the left-most coordinates where the bucketing was not yet performed as we implicitly assume the mod q operation. The algorithm can be thought of as a special kind of BKZ-reduction, where, instead of projecting on short and non-orthogonal vectors, we project on long but orthogonal vectors. Also, as opposed to a BKZ algorithm where the block-size is fixed to β, our d_i's differ. Further, our blocks do not intersect similar to the BKZ slide-reduction.

- **Connection to BKW algorithms**. As we already mentioned, the presented algorithm is a reformulation of the BKW algorithms of [KF15, GJS15] to appSVP_γ. The fact that we can 'save' half of the dimension, n, by switching to the systematic form of the generator matrix A is no surprise: solving LWE directly with BKW (not via lattices) does not have this additional n at all.

4.2.3 An improved algorithm for appSVP_γ on a q-ary lattice

In this section we present an improved algorithm for appSVP_γ on a q-ary lattice. The improvement is in the constant c. We introduce a new constant ε in the algorithm and c will depend on it. Namely, when $0 < \varepsilon < 1/4$, the algorithm achieves a smaller value for c, and when $\varepsilon = 0$, it is Algorithm 9 from the previous section.

The gain comes from the following two changes. First, we do not only perform bucketing on 'new' coordinates on the left part (i.e. on coordinates with ℓ_∞-norm less than $q/2$), but also on already considered coordinates on the right part, i.e. on blocks in-between the n^{th} and the $2n^{\text{th}}$ coordinates.

Second, and more importantly, we reduce the growth of the block-bounds R_i. Recall that in the previous algorithm, we had $R_{i+1} = \sqrt{2}R_i$, where the $\sqrt{2}$ comes from the average ℓ_∞-norm of the sum of two vectors whose ℓ_∞-norm is R_i. This choice balanced the ℓ_∞-norm on the block d_i with the ℓ_∞-norm on all the previous blocks $d_{i-1},\ldots,d_1$. Now, we set $R_{i+1} = 2^{1/2-\varepsilon}R_i$ for some $\varepsilon > 0$. The fact that we set $\varepsilon > 0$ is justified by our additional bucketing on the right part which requires another choice of R_i's to balance the norms between the blocks. When compared with the previous algorithm, the expected length of a vector from L_i when we perform this new bucketing is shorter, and since the final length is fixed to $n^{\mathsf{c}_\gamma+\mathsf{c}_q/2+1/2}$, we can choose larger (by a constant factor) k. From the analysis of the previous algorithm, it follows that any constant improvement for k results in smaller c (see the proof of Thm. 31).

Let us describe the algorithm in more detail. The initial list L_0 is created in the same way as in Algorithm 9: we sample a vector $\vec{x} \in \mathbb{Z}_q^{2n}$ with the right-most n coordinates bounded by R_1. The remaining n left-most coordinates are divided into k blocks of length d_i. Note that we can 'mirror' this partition to the right-most n coordinates w.r.t. the 'middle' n^{th} coordinate. Let us denote the bounds of the i^{th} block of length d_i as $[l_i,\ldots,l_{i-1}]$ for the left part (i.e. $[l_{i-1},\ldots,l_i] \in [1,\ldots,n]$) and $[r_{i-1},\ldots,r_i]$ for the right part (i.e. $[r_{i-1},\ldots,r_i] \in [n+1,\ldots,2n]$, see Fig. 4.6).

Algorithm 10 appSVP_γ on a q-ary lattice

Input: D – a basis for the lattice $\mathcal{L}_q^\perp(A) \subset \mathbb{Z}_q^{2n}$ defined as in Eq. (4.16), $\gamma = n^{\mathsf{c}_\gamma}$ – the approximation factor, $\mathsf{c}_\gamma = \Theta(1)$
Output: L_k – list of vectors from $\mathcal{L}_q^\perp(A)$ with vectors of norm $\|\vec{v}\| \leq n^{\mathsf{c}_\gamma + \mathsf{c}_q/2+1/2}$;

Set the sieving bounds R_i as $R_1 = n^{o(1)}$ and $R_i = 2^{(1/2-\varepsilon)(i-1)} R_1$ for $i \geq 2$.
Set the lengths of blocks d_i as in Eq. (4.20) together with the corresponding boundaries of the left-hand side blocks $(l_i, \ldots, l_{i-1})$ and of the right-hand side blocks $(r_{i-1}, \ldots, r_i)$ s.t. $l_i - l_{i-1} = r_{i-1} - r_i = d_i$, and $l_k = 2n, l_0 = r_0 = n, r_k = 2n$.
repeat ▷ Create the list L_0
    Choose $\vec{x} \in \mathbb{Z}_q^{2n}$ s.t. $\|[\vec{x}]_{n+1}^{2n}\|_\infty \leq R_1$
    $L_0 \leftarrow L_0 \cup \{D\vec{x} \bmod q\}$
until L_0 is large enough
$T \leftarrow \emptyset$ ▷ Initialize table T indexed by buckets
for all $i = 1 \ldots k$ **do**
    for all $\vec{v} \in L_{i-1}$ **do**
        $b \leftarrow \left\lfloor \frac{[\vec{v}]_{l_i}^{l_{i-1}} [\vec{v}]_{r_{i-1}}^{r_i}}{R_i} \right\rceil$ ▷ Find the bucket for $\vec{v}[l_i, \ldots, l_{i-1}, r_{i-1}, \ldots r_i]$
        if $T[b] = \emptyset$ **then**
            $T[b] \leftarrow \vec{v}$
        else
            $L_i \leftarrow L_i \cup \{T[b] - \vec{v}\}$
            $T[b] \leftarrow \emptyset$
return L_k

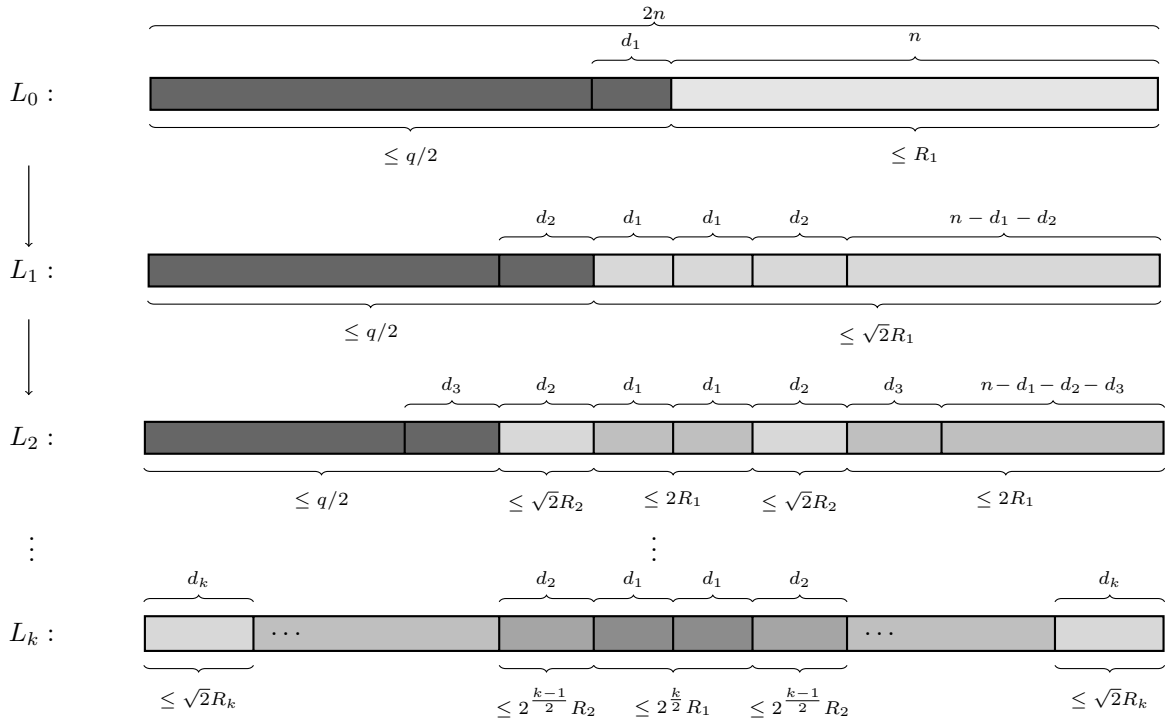


Fig. 4.6: Pictorial representation of Alg. 10. Due to the fact that our bounds satisfy $R_{i+1} < \sqrt{2}R_i$, the ℓ_∞-norm is not evenly distributed over the length.

Now, on step i, the two vectors, $\vec{v}_1$ and $\vec{v}_2$ from L_{i-1}, land in the same bucket if

$$\left\lfloor \frac{[\vec{v}_1]_{l_i}^{l_{i-1}}[\vec{v}_1]_{r_{i-1}}^{r_i}}{R_i} \right\rceil = \left\lfloor \frac{[\vec{v}_2]_{l_i}^{l_{i-1}}[\vec{v}_2]_{r_{i-1}}^{r_i}}{R_i} \right\rceil.$$

This additional bucketing on the $[r_{i-1}, \ldots, r_i]$-coordinates makes the difference $\vec{v}_1 - \vec{v}_2 \in L_i$ shorter.

The complete algorithm is presented in pseudo-code in Alg. 10. Parts where the new algorithm differs from the one given in the previous section are highlighted blue.

4.2.4 Analysis

The analysis of the improved algorithm proceeds exactly like the analysis of Algorithm 9 presented in Thm. 31. The only difficulty comes from estimating the length of the vectors in the output list L_k and, hence, concluding on k. The proof below is mostly dedicated to this matter.

Theorem 33. *Algorithm 9 on input (1) a lattice-basis $D \in \mathbb{Z}_q^{2n\times 2n}$ as in Eq. (4.16) for the lattice $\mathcal{L}_q^\perp(A)$ with $q = \mathcal{O}(n^{\mathsf{c}_q})$, (2) an approximation factor $\gamma = \mathcal{O}(n^{\mathsf{c}_\gamma})$, and (3) $0 < \varepsilon < 1/4$, outputs a vector $\vec{v} \in \mathcal{L}_q^\perp(A)$ of length $\|\vec{v}\| \le n^{\mathsf{c}_\gamma + \mathsf{c}_q/2 + 1/2}$ in expected time $T(\mathsf{appSVP}_\gamma) = 2^{(\mathsf{c}+o(1))n}$, where*

$$\mathsf{c} = \frac{1/2 - 2\varepsilon}{\ln\left(\frac{\mathsf{c}_q}{\mathsf{c}_q\cdot(1/2-2\varepsilon\ln(2)) - \mathsf{c}_\gamma\cdot 2(1/2-2\varepsilon)}\right)}. \tag{4.19}$$

Proof. Analogously to Algorithm 9, the expected number of lattice-vectors needed to fill-up all the buckets on step i is now given by

$$\mathbb{E}[\#\text{buckets on level } i] = \frac{\mathrm{vol}([-\frac{q-1}{2}, \frac{q-1}{2}]^{d_i})}{\mathrm{vol}([-R_i/2, R_i/2]^{d_i})} \cdot \frac{\mathrm{vol}([-\frac{\sqrt{2}^{i-1}R_1}{2}, \frac{\sqrt{2}^{i-1}R_1}{2}]^{d_i})}{\mathrm{vol}([-R_i/2, R_i/2]^{d_i})} = \Theta\Big(\Big(\frac{q}{R_i}\cdot\frac{\sqrt{2}^{i-1}R_1}{R_i}\Big)^{d_i}\Big),$$

where the second multiple, the fraction of the volumes of two cubes, comes from the additional bucketing on the right-hand side coordinate-blocks. Setting $R_i = x^{i-1}R_1$ for some $x = 2^{1/2-\varepsilon}$, the expected number of buckets on level i, or, equivalently, the running time of the algorithm, is (up to a poly(n)-factor)

$$\Big(\Big(\frac{\sqrt{2}}{x^2}\Big)^{i-1}\frac{q}{R_1}\Big)^{d_i} = 2^{n\mathsf{c}}.$$

The above formula yields for d_i

$$d_i = \frac{n\mathsf{c}}{\log(q/R_1) - (i-1)\log(x^2/\sqrt{2})}. \tag{4.20}$$

For the rest of the proof, assume $\varepsilon < 1/4$ and, hence, $\log(x^2/\sqrt{2}) > 0$. As in the proof of Thm. 31, from the above expression for d_i and the fact that $\sum_{i=1}^k d_i = n$, we determine c once we know k.

The expected length of a vector from L_k is upper-bounded as follows

$$\|\vec{v}\| \le \sqrt{2d_k(\sqrt{2}R_k)^2 + \ldots + 2d_1(\sqrt{2}^k R_1)^2} = 2R_1\sqrt{\sum_{i=1}^k d_i(\sqrt{2}^{k-i}x^{i-1})^2} = R_1\sqrt{2^{k+1}}\sqrt{\sum_{i=1}^k d_i 2^{-2\varepsilon(i-1)}}.$$

Using the expression for d_i given in Eq. (4.20) and the fact that $x = 2^{1/2-\varepsilon}$, the argument in the $\sqrt{(\cdot)}$ from above is

$$\sum_{i=1}^k d_i 2^{-2\varepsilon(i-1)} = n\mathsf{c}\sum_{i=0}^{k-1}\frac{1}{2^{2\varepsilon i}(\log q/R_1 - i(1/2-2\varepsilon))}.$$

Upper-bounding the sum by the integral, we notice that an integral of the form $\int \frac{\mathrm{d}x}{2^{ax}(b-xc)}$ for positive a, b, c is equal to $\frac{2^{-ab/c}}{c}\mathrm{E}_1(a\ln(2)x - \frac{ab\ln(2)}{c})$, where E_1 is the exponential integral $\mathrm{E}_1(z) = \int_1^\infty \frac{e^{-tz}}{t}\mathrm{d}t$ (we refer the reader to [Leb63] for properties of this integral). We know that the sum we are currently computing is of order $\Theta(n^\alpha)$ for some constant α and we are only interested in determining this α (not the precise constants and lower-order terms) as α will appear in the constant for k. In our case, α is actually negative, so the bound on the length of an element in L_k will eventually be (up to constants) $2^k\sqrt{n^{1+\alpha}}$ (here, as in the previous algorithm, we set $R_1 = n^{o(1)}$ and we ignore $o(1)$-terms).

Substituting our values for a, b, c, we have (up to multiplicative constants) (1) $2^{-ab/c} = n^{-\frac{2\varepsilon \mathsf{c}_q}{1/2-2\varepsilon}}$, and (2) $\mathrm{E}_1(a\ln(2)x - \frac{ab\ln(2)}{c}) = n^{-\frac{2\ln(2)\varepsilon \mathsf{c}_q}{1/2-2\varepsilon}}$. The length of the output vector is required to be bounded by $n^{\mathsf{c}_\gamma + \mathsf{c}_q/2+1/2}$, from where we have (ignoring $o(1)$-terms)

$$k \leq 2\Big(\frac{(1+\ln(2))\varepsilon \mathsf{c}_q}{1/2-2\varepsilon} + \frac{\mathsf{c}_q}{2} + \mathsf{c}_\gamma\Big)\log n.$$

Note that for $\varepsilon = 0$, we receive the value for k as in Alg. 9. As soon as $\varepsilon < 1/4$, the above choice for k guarantees that $k < \frac{\mathsf{c}_q}{1/2-\varepsilon}$ – the upper-bound for k coming trivially from $d_k > 0$ (otherwise, the denominator in Eq. (4.20) is negative).

We compute the sum $\sum_{i=1}^k d_i$ as we did in the proof of Thm. 31, and obtain

$$\mathsf{c} = \frac{1/2-2\varepsilon}{\ln\Big(\frac{\mathsf{c}_q}{\mathsf{c}_q\cdot(1/2-2\varepsilon\ln(2))-\mathsf{c}_\gamma\cdot 2(1/2-2\varepsilon)}\Big)} + o(1).$$

In case $\varepsilon = 0$, the algorithm is exactly our first Algorithm 9 with $\mathsf{c} = \frac{1}{2\ln\left(\frac{\mathsf{c}_q}{\mathsf{c}_q/2-\mathsf{c}_\gamma}\right)} + o(1)$. □

One could further optimize for ε, but the resulting expression neither simplifies the expression for c, nor does it provide more insights into the algorithm's complexity. In the next section, we compare the two algorithms, Alg. 9 and Alg. 10 (setting $\varepsilon = 1/5$ for the latter) with the BKZ algorithm for appSVP_γ.

4.2.5 Comparison with BKZ

In this section we compare the constants in the exponents of the running time of algorithms for appSVP_γ on the $2n$-dimensional lattice $\mathcal{L}_q^\perp(A)$. The length of the target vector is $\|\vec{v}\| \leq n^{\mathsf{c}_\gamma}\det(\mathcal{L}_q^\perp(A))^{1/(2n)}$. The algorithms in consideration are the BKZ lattice-reduction algorithm with a block-size $\beta = \Theta(n)$ and our two algorithms Alg. 9 and Alg. 10. For the BKZ algorithm, we have the following simple lemma.

Lemma 34. *Given on input (1) a $2n$-dimensional q-ary lattice $\mathcal{L}_q^\perp(A)$ with a basis as in Eq. (4.16), and (2) an approximation factor $\gamma = \mathcal{O}(n^{\mathsf{c}_\gamma})$, the BKZ basis-reduction algorithm instantiated with an SVP oracle that runs in time $\mathcal{O}(2^{(c_{BKZ}+o(1))n})$, outputs a vector of desired length in time*

$$T(BKZ) = 2^{\Big(\frac{c_{BKZ}}{\mathsf{c}_\gamma+1/2}+o(1)\Big)n}.$$

Proof. The result follows immediately from Eq. (2.1): solving for β the inequality $\beta^{\frac{2n}{2\beta}}\cdot\det(\mathcal{L}_q^\perp(A))^{\frac{1}{2n}} \leq n^{\mathsf{c}_\gamma+1/2}\cdot\det(\mathcal{L}_q^\perp(A))^{1/(2n)}$, yields $\beta = \Big(\frac{c_{\mathtt{BKZ}}}{\mathsf{c}_\gamma+1/2} + o(1)\Big)n$. □

One should not be surprised that the constant for BKZ does not depend on c_q: the size of the modulus only affects polynomial pre-factors in the complexity of the algorithm [HPS11].

As discussed in Chap. 2 and Chap. 3, we can instantiate an SVP oracle using provable algorithms with $c_{\mathtt{BKZ}} = 1$, or using algorithms that rely on some heuristics with $c_{\mathtt{BKZ}} = 0.292$. These together with

our results on combinatorial algorithms for appSVP_γ given in Thms. 31 and 33, allow us to compare all four algorithms and deduce which one performs best for given $\mathsf{c}_q, \mathsf{c}_\gamma$.

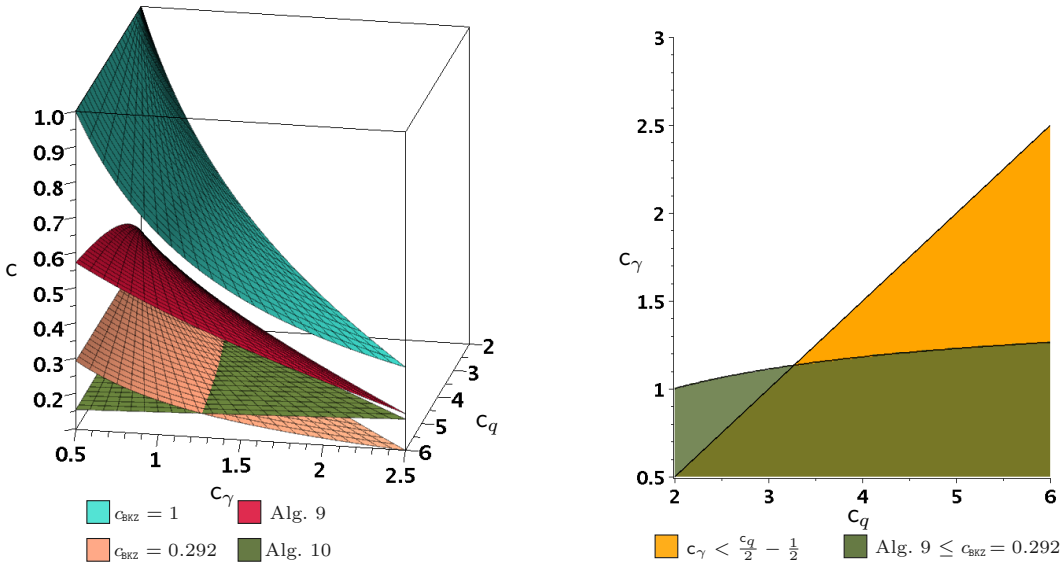


Fig. 4.7: Comparison of two families algorithms for appSVP_γ: those that are based on BKZ reduction and combinatorial ones. The considered parameter range is $\mathsf{c}_q = [2, \ldots, 6], \mathsf{c}_\gamma = [0.5, \ldots \mathsf{c}_q/2 - 1/2]$.

Combinatorial Dual attack on LWE. Recall the Dual attack on LWE with parameters $n, q = \mathcal{O}(n^{\mathsf{c}_q})$, $\alpha = \mathcal{O}(n^{-\mathsf{c}_\alpha})$ (see Alg. 4 in Sect. 3.1.5). As the main subroutine, it runs an appSVP_γ solver on the lattice $\mathcal{L}_q^\perp(A)$. The approximation factor γ depends on the number of LWE samples provided. Instead of running a β-BKZ reduction to solve appSVP_γ as we do in Sect. 3.1.5, we can run a combinatorial algorithm for the search of a short vector in $\mathcal{L}_q^\perp(A)$. In case, we have exponentially-many samples, we run our Alg. 9 with $\mathsf{c}_\gamma = \mathsf{c}_\alpha - \mathsf{c}_q/2$ in which case the running time of the (combinatorial) Dual attack on LWE is $2^{(\mathsf{c}+o(1))n}$ where $\mathsf{c} = \left(2\ln\left(\frac{\mathsf{c}_q}{\mathsf{c}_q - \mathsf{c}_\alpha}\right)\right)^{-1}$. In case $m = \Theta(n \log n)$, we resort to the sample amplification which decreases c_α by $1/2$. This results in a smaller approximation factor leading to a worse running time constant $\mathsf{c} = \left(2\ln\left(\frac{\mathsf{c}_q}{\mathsf{c}_q - \mathsf{c}_\alpha + 1/2}\right)\right)^{-1}$.

Open Problems

There are several open problems that emerge from the results presented here. These are interesting directions that can be taken for future work.

Open problems from Part I

- Throughout the whole asymptotical analysis of the Learning with Errors problem, we assumed certain *polynomial* dependencies between parameters (n, q, α), While this is the most relevant case for cryptography, it would be interesting to see how the complexity changes if we choose, for example, a sub-exponential $q = 2^{\sqrt{n}}$. Note that a similar regime has been recently studied for problem a closely related to LWE, the NTRU problem [ABD16]. The NTRU problem in case $q = 2^{\sqrt{n}}$ becomes significantly easier. For LWE the answer will certainly depend on the third parameter, α.

- On a practical side, the following question is relevant: how much can we lower q while preserving the value $q\alpha$ (i.e. the width of the LWE error)? Small q would reduce the bandwidth of LWE-based protocols. While a small modulus is unlikely to be favourable for lattice-based attacks, combinatorial attacks (even naive brute-force for very small q) can gain a reasonable speed-up and may become practical (in terms of the memory-complexity) once the modulus is set to be very small.

Open problems from Part II

- We have already mentioned a couple of open questions that emerged from our memory efficient k-List SVP algorithm: how to analyze our Algorithm 7 for a non-fixed k? How our algorithm will improve if we allow to use *more* memory, e.g. for building hash-tables like it is done in previous sieving algorithms for SVP [Laa15]? From the practical side, parallelizing our Algorithm 8 seems to be an important but a very non-trivial task.

- Naturally, we would want to transfer our k-List algorithm for the Euclidean spaces to domains with other metrics, e.g., consider binary vectors and Hamming distance. Decoding random linear codes over $\mathbb{F}_2^n$ would be an application for such an algorithm.

- Our combinatorial algorithms for appSVP_γ open up several questions as well. Recall that in Algorithm 10, we used a rather specific way to partition the coordinates and bucket them accordingly (see Fig. 4.6). We do not know whether this choice is optimal. We chose to perform the 'additional' bucketing on blocks of length $\mathcal{O}(n/\log n)$ as our analysis showed that bucketing on blocks of length $\mathcal{O}(n)$ does not seem to improve the algorithm.

  Finally, the same technique as to our Algorithm 10 should bring an improvement for BKW algorithms on LWE as these are the same k-List problem. Our way to shorten the length during the bucketing can be converted to a method that decreases the error-growth in the BKW algorithm. The impact of such algorithms on LWE is left open.

Bibliography

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 153–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, 74(2):325–354, 2015.

[ADPS15] Erdem Alkim, Léo Ducas, Thomas Pppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092, 2015. http://eprint.iacr.org/2015/1092.

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: Extended abstract. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 733–742, Portland, OR, USA, June 14–17, 2015. ACM Press.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415, Zurich, Switzerland, July 4–8, 2011. Springer, Heidelberg, Germany.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108, Philadephia, PA, USA, May 22–24, 1996. ACM Press.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *33rd ACM STOC*, pages 601–610, Crete, Greece, July 6–8, 2001. ACM Press.

[Aon14] Yoshinori Aono. A faster method for computing Gama-Nguyen-Regev's extreme pruning coefficients. *CoRR*, abs/1406.0342, 2014.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Cryptology ePrint Archive, Report 2015/046*, 2015.

[Bab85] László Babai. On Lovász' lattice reduction and the nearest lattice point problem (shortened version). In *Proceedings of the 2Nd Symposium of Theoretical Aspects of Computer Science*, STACS '85, pages 13–20, London, UK, 1985. Springer-Verlag.

[Bai16] Shi Bai. Personal communication, 07 2016.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24, Arlington, VA, USA, January 10–12, 2016. ACM-SIAM.

[BG14] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *ACISP 14*, volume 8544 of *LNCS*, pages 322–337, Wollongong, NSW, Australia, July 7–9, 2014. Springer, Heidelberg, Germany.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, July 2003.

[Blo16] Google Security Blog. Experimenting with post-quantum cryptography, October 26, 2016. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *LMS Journal of Computation and Mathematics*, 19(A):146–162, 08 2016.

[BV97] Dan Boneh and Ramarathnam Venkatesan. Rounding in lattices and its cryptographic applications. In Michael E. Saks, editor, *8th SODA*, pages 675–681, New Orleans, LA, USA, January 5–7, 1997. ACM-SIAM.

[Cas97] John William Scott Cassels. *An introduction to the geometry of numbers*. Classics in mathematics. Springer, Berlin, Heidelberg, Paris, 1997. Originally published as vol. 99 of : Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen.

[CCC] Crypto Crunching Cluster. Ruhr-Universität Bochum. https://www.hgi.rub.de/hgi/Netzwerk/c3/.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[Cop01] Don Coppersmith. *Finding Small Solutions to Small Degree Polynomials*, pages 20–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 1976.

[DT16] The FPLLL Development Team. FPLLL, a lattice reduction library. Available at https://github.com/fplll/fplll, 2016.

[DTV15] Alexandre Duc, Florian Tramèr, and Serge Vaudenay. Better algorithms for LWE and LWR. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 173–202, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[Eat07] Morris L. Eaton. *Chapter 8: The Wishart Distribution*, volume Volume 53 of *Lecture Notes–Monograph Series*, pages 302–333. Institute of Mathematical Statistics, Beachwood, Ohio, USA, 2007.

[FP83] U. Fincke and Michael Pohst. A procedure for determining algebraic integers of given norm. In J. A. van Hulzen, editor, *Computer Algebra, EUROCAL 1983 Proceedings*, volume 162 of *Lecture Notes in Computer Science*, pages 194–202. Springer, 1983.

[Gal] Steven D. Galbraith. Space-efficient variants of cryptosystems based on learning with errors.

[GG98] Oded Goldreich and Shafi Goldwasser. On the limits of non-approximability of lattice problems. In *30th ACM STOC*, pages 1–9, Dallas, TX, USA, May 23–26, 1998. ACM Press.

[GJL14] Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 1–20, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[GJS15] Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-BKW: Solving LWE using lattice codes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 23–42, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[GM06] Daniel Goldstein and Andrew Mayer. On the equidistribution of hecke points. *Forum Mathematicum*, 15(3):165–189, 01 2006.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 257–278, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

[HK] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In submission.

[HKM] Gottfried Herold, Elena Kirshanova, and Alexander May. On the asymptotic complexity of solving LWE. *Designes, Codes and Cryptography, To appear.*

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 447–464, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[HR07] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 469–477, San Diego, CA, USA, June 11–13, 2007. ACM Press.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 170–186, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 193–206, New York, NY, USA, 1983. ACM.

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

[KF15] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 43–62, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[Kho03] Subhash Khot. Hardness of approximating the shortest vector problem in high Lp norms. In *44th FOCS*, pages 290–297, Cambridge, MA, USA, October 11–14, 2003. IEEE Computer Society Press.

[Kir14] Elena Kirshanova. Proxy re-encryption from lattices. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 77–94, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In David B. Shmoys, editor, *11th SODA*, pages 937–941, San Francisco, CA, USA, January 9–11, 2000. ACM-SIAM.

[KMW16] Elena Kirshanova, Alexander May, and Friedrich Wiemer. Parallel implementation of bdd enumeration for lwe. In *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, pages 580–591. Springer International Publishing, 2016.

[Kup05] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM J. Comput.*, 35(1):170–188, 2005.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 3–22, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[Leb63] Nikolai Lebedev. *Special functions and their applications*. Gosizdat, Moscow, 1963. (in Russian).

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

[LLL82] Arjen K. Lenstra, Hedrik W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 577–594, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[LN13] Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 293–309, San Francisco, CA, USA, February 25 – March 1, 2013. Springer, Heidelberg, Germany.

[LO85] Jeffrey C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, January 1985.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339, San Francisco, CA, USA, February 14–18, 2011. Springer, Heidelberg, Germany.

[LWE] LWE challenge generator. http://latticechallenge.org/svp-challenge.

[Lyu05a] Vadim Lyubashevsky. On random high density subset sums. *Electronic Colloquium on Computational Complexity (ECCC)*, (007), 2005.

[Lyu05b] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Proceedings of the 8th International Workshop on Approximation, Randomization and Combinatorial Optimization Problems, and Proceedings of the 9th International Conference on Randamization and Computation: Algorithms and Techniques*, APPROX'05/RANDOM'05, pages 378–389, Berlin, Heidelberg, 2005. Springer-Verlag.

[Mic98] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *39th FOCS*, pages 92–98, Palo Alto, CA, USA, November 8–11, 1998. IEEE Computer Society Press.

[Mic05] Daniele Micciancio. Almost perfect lattices, the covering radius problem, and applications to ajtai's connection factor. *SIAM J. Comput.*, 34(1):118–169, January 2005.

[Mic10] Daniele Micciancio. Duality in lattice cryptography. In Public Key Cryptography. Invited talk, 2010.

[MLB15] Artur Mariano, Thijs Laarhoven, and Christian Bischof. Parallel (probable) lock-free hash sieve: A practical sieving algorithm for the svp. In *44th International Conference on Parallel Processing (ICPP)*, pages 590–599, September 2015.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 465–484, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 21–39, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[MR09] Daniele Micciancio and Oded Regev. *Lattice-based Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[MS09] Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. In Claire Mathieu, editor, *20th SODA*, pages 586–595, New York, NY, USA, January 4–6, 2009. ACM-SIAM.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In Leonard J. Schulman, editor, *42nd ACM STOC*, pages 351–358, Cambridge, MA, USA, June 5–8, 2010. ACM Press.

[NR06] Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 271–288, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[NS15] Ivica Nikolic and Yu Sasaki. Refinements of the k-tree algorithm for the generalized birthday problem. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 683–703, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

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

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[Pei16] Chris Peikert. A decade of lattice cryptography, 2016. Monograph.

[PS09] Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. Cryptology ePrint Archive, Report 2009/605, 2009. http://eprint.iacr.org/2009/605.

[Rad46] R. Rado. A theorem on the geometry of numbers. *Journal of the London Mathematical Society*, s1-21(1):34–47, 1946.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[Reg09] Oded Regev. Lecture notes: Lattices in computer science, 2009. http://www.cims.nyu.edu/~regev/teaching/lattices_fall_2009/index.html.

[RSA78] Ron L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2-3):201–224, August 1987.

[Sch03] Claus-Peter Schnorr. Lattice reduction by random sampling and birthday methods, 2003.

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, 1994.

[Sho] Victor Shoup. Number Theory Library 5.5.2 for C++. http://www.shoup.net/ntl/.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[SVP] SVP challenge generator. http://latticechallenge.org/svp-challenge.

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[Wis28] John Wishart. The generalized product moment distribution in samples from a normal multivariate population. *Biometrika*, 20A(1-2):32–52, 1928.

Elena Kirshanova

Contact Information	Universitätstraße 150, NA 5/75 44801 Bochum, Germany	+49 (0)234 32 23259 elena.kirshanova@rub.de

Research Interests

Lattice-based cryptography, cryptanalysis, algorithms

Education

I. Kant Baltic Federal University, Kaliningrad, Russia

Dipl. Math., Jan. 2013

- Topic: *Lattice-based cryptography*
- Advisor: Dr. Sergey Aleshnikov

Research Experience

PhD Candidate — May 2013 to present

Ruhr University Bochum
Faculty of Mathematics,
Chair of Cryptology and IT-Security

- Topic: *Complexity of the Learning with Errors Problem and Memory-Efficient Lattice Sieving*
- Supervisor: Prof. Dr. Alexander May

Submitted Publications

1. G. Herold, E. Kirshanova, A. May. On the Asymptotic Complexity of Solving LWE, 2016. Submitted to *Designs, Codes and Cryptography*
2. G. Herold, E. Kirshanova. Improved Algorithms for the Approximate k-List Problem in Euclidean norm. 2016. Submitted to *PKC*

Paper in preparation

1. E. Kirshanova, D. Stehlé, W. Wen. Learning With Errors and the Generalized Hidden Shift Problem. 2016

Conference Publications

1. E. Kirshanova. Proxy re-encryption from lattices. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 7794, Buenos Aires, Argentina, March, pages 26–28, 2014. Springer, Heidelberg, Germany.
2. E. Kirshanova, A. May, and F. Wiemer. Parallel implementation of BDD enumeration for LWE. In *Applied Cryptography and Network Security: 14th International Conference*, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings, pages 580–591. Springer International Publishing, 2016.

Teaching Experience

Teaching Assistant

Quantum Algorithms — Winter term 2013-14
Lecturer: Prof. Dr. A. May
Ruhr University Bochum

Cryptanalysis I-II — 2014-15
Lecturer: Prof. Dr. A. May
Ruhr University Bochum

Quantum Random Walks (seminar) — Winter term 2016-17
Ruhr University Bochum

Awards	Euler Travel Grant (visit at the University of Leipzig)	Feb. 2012
Presentations	• Workshop on Public Key Cryptography, Buenos Aires, Argentina	March 2014
	• Kryptotag, Berlin, Germany	June 2014
	• CrossFire, Bochum, Germany	July 2014
	• ACNS Conference, Surrey, UK	June 2016
	• Workshop on Mathematical Structures in Cryptography, Leiden	August 2016
	• HNI Symposium, Padeborn, Germany	Sep 2016
Languages	• English (fluent)	
	• German (professional proficiency)	
	• Russian (native)	

References

Alexander May alex.may@rub.de
Professor at the University of Bochum
Faculty of Mathematics
Chair of Cryptology and IT-Security

Sergey Aleshnikov sergey.aleshnikov@gmail.com
Head of the Chair Mathematical Methods in Cryptography
Faculty of Mathematics
I.Kant Baltic Federal University