EINDHOVEN UNIVERSITY OF TECHNOLOGY

Lattice cryptanalysis

Citation for published version (APA):
Doulgerakis, E. (2022). *Lattice cryptanalysis: Theoretical and practical aspects*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Eindhoven University of Technology.

Document status and date:
Published: 09/09/2022

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 16. Nov. 2023

Lattice cryptanalysis

Theoretical and practical aspects

Emmanouil Doulgerakis

Copyright © Emmanouil Doulgerakis
E-mail: emmanouhld@gmail.com

First edition August 2022

This research was supported by the Netherlands Organisation for Scientific Research (NWO) grant 628.001.028 (“Faster and Stronger Onion Routing”).

Printed by: Gildeprint – www.gildeprint.nl

A catalogue record is available from the Eindhoven University of Technology Library.

ISBN: 978-90-386-555-0

The cover (front and back) presents an analogy of how cryptography is usually viewed by non-expert people. In most cases the “end product” receives the most attention: the security of communications. The mathematical foundations of cryptography tend to be ignored, like a back cover is usually ignored compared to the front cover.

The picture on the front cover is used under a Standard License from the webpage www.istockphoto.com. The graph on the back cover was a result of this thesis. Further information about it can be found in Chapter 3.

Lattice cryptanalysis

Theoretical and practical aspects

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven,
op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie
aangewezen door het College voor Promoties, in het openbaar te verdedigen
op vrijdag 9 september 2022 om 16:00 uur

door

Emmanouil Doulgerakis

geboren te Rethymnon, Griekenland

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotie-commissie is als volgt:

voorzitter:	prof. dr. E.R. van den Heuvel
1e promotor:	dr. B.M.M. de Weger
2e promotor:	prof. dr. T. Lange
leden:	prof. dr. M. Albrecht (Royal Holloway)
	prof. dr. D.J. Bernstein
	dr. D. Dadush (Centrum Wiskunde & Informatica)
	prof. dr. A. May (Ruhr-Universität Bochum)
	dr. ir. L.A.M. Schoenmakers

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Acknowledgments

Like many PhD candidates before me, I would like to extend a word about the journey called PhD and the people who helped me throughout it.

The first people whom I would like to thank are my supervisors, as without them this journey would not have even started. Despite the fact that I was coming from a relatively remote place, you interviewed me and decided to give me the opportunity to join your group. Since then you have kept supporting and being patient with me even though I had a rough start with my project. In addition, you gave me the freedom to pursue my own ideas and become independent. A special thanks goes to my daily supervisor Benne. Benne, you had always been there for me, listening to my crazy, semi-crazy and sometimes non-crazy ideas. There was always something to learn from you. Even when we went for hiking at the mountains of Zhangjiajie, you would still have a nice lesson to teach me. Additionally, I would like to thank you for your endless patience while helping me to improve my writing style. I would also like to thank my second supervisor, Tanja. Even though we only worked more closely towards the end of my PhD, I always knew that I could ask you for anything I would need help with. Furthermore I must mention that I greatly enjoyed your chocolate cakes, which you used to bring on special occasions. Concluding, I could say that I had the privilege to be supervised by great scientists who are also great humans. I will be grateful to you for the rest of my life for this opportunity.

Next I would like to thank my committee Martin Albrecht, Daniel J. Bernstein, Daniel Dadush, Alexander May and Berry Schoenmakers for taking the time to read this thesis and performing all the duties that follow by joining my doctorate committee.

Being a member of the coding theory and cryptology group I had the opportunity to travel around the world to conferences and workshops. In many cases, much farther than I had ever imagined I would travel, or I would dare to travel. This is an additional reason why I should thank my supervisors. While attending all these conferences and workshops I had the opportunity to broaden and deepen my knowledge on the field of cryptology. Apart from scientific knowledge, all these journeys provided me the opportunity to grow more mature through the interaction with great scientists as well as with a variety of different cultures. Such experiences make a person rethink of his place in the world, both as a scientist and a human being. Questions like, which projects match my interests, which is my place in the scientific community, or which is the future I desire for myself, become eminent. Cultural questions also tend to arise. Some of these are: why do we behave differently, how are cultures formed, do we notice the similarities we share, do we respect each other enough. At this point I will stop because as one of my supervisors

says, I tend to getting too philosophical.

My staying in the coding theory and cryptology group would not have been as fun if it had not been for some great colleagues. Thank you all, for the nice lunch breaks and coffee breaks. In alphabetical order I would like to thank Alberto, Alessandro, Alessandro, Andy, Anita, Bor, Boris, Chloe, Christine, Daan, Daniel, Davide, Dominik, Florian, Francisco, Frank, Gustavo, Harm, Huub, Iggy, Jake, Kai-Chun, Laura, Leon, Lorenz, Mahdi, Milan, Mina, Niek, Niels, Pavlo, Putranto, Simon, Sowmya, Stefan, Taras, Thijs and Tomer. For some of them I would like to express my gratitude separately.

To my permanent officemate Gustavo, I want to say that it was a great pleasure to share the office with you. We could always understand each other when the weather was too depressing or the food at the canteen was not... satisfying. We would also have some nice conversations when we did not feel like working, with topics ranging from poisonous animals of the Amazon to ways of cooking meat in Brazil vs Greece. Furthermore, I am very glad that during our studies I had the chance to introduce you to Greek and especially Cretan cuisine (which you loved). Summarising, you offered me one of the most unexpected and happy surprises during my PhD, you showed me that even nations from two opposite sides of the globe may share similar cultures.

I would also like to thank Alessandro for being a fellow Mediterranean guy, who would also understand me and offer me a positive thought even in difficult times. Next I would like to thank Thijs for many reasons. The most important of them is that he greatly inspired me as a researcher since his return in Eindhoven. He introduced me to sieving, Voronoi cells and showed me how to prepare nice graphics for talks. Together we spent a lot of time discussing about lattices and various other topics. Also we had the chance to spend some nice time together during conferences around the world. Last but not least, I would like to thank our secretary Anita for always happily helping when bureaucratic issues would arise.

Moving to family and friends from Crete, I would like to thank my cousin Antonis, whose help and hospitality were of critical importance for my survival during my first days in the Netherlands. Antonis, thank you for your generosity and your optimistic spirit! Also a special thanks goes to my dear friends and “koumparoi” in Crete, Dimitris and Maria. Thank you both for always making my vacation in Crete much more fun and enjoyable. Our trips around Crete were definitely amongst the highlights of my summer vacation. I wish that in the future we will be able to spend more time together.

Moving towards the end of this acknowledgments, I would like to thank my family for their love and support during all these years. Even though modern technology allows calls and video calls, managing the distance was not always easy. Thank you for supporting me even when I would not have enough time for us to talk or I would not be in a good mood.

I want to conclude these acknowledgments with the person who has supported me the most during my PhD journey. Anthi, it would simply have been impossible without your love and support. I am not able to express in only just a paragraph the magnitude of your support during the past five years. Together we faced many challenges while pursuing our PhDs. Despite these difficulties, you have always been there for me. The least to say is that you help me become the best of me. Thank you for everything!

Emmanouil Doulgerakis

Paderborn, June 2022

Contents

Introduction

Perhaps one of the most important moments in human history was when we first managed to establish a formal means of communication, a language. This was a great breakthrough as it allowed knowledge not only to be shared among people in a standardised way, but also to be transferred through time. However, it soon became clear that some information was supposed to remain secret and not to be shared indiscriminately. This became apparent in the field of military communications. For instance, assume that a message-bearer in ancient Greece carries a scroll including a message from an Athenian general to a Spartan. It would be very unfortunate if that scroll fell in Persian possession. If the message-bearer was bribed or captivated by the Persians that would enable them to just read the scroll and become aware of the message. A new idea was needed, one that would ensure the secrecy of the message without any assumptions on the trustworthiness of the message-bearer. This is when cryptography was born.

Even though nowadays cryptography is not only used to ensure secrecy, this was actually its only purpose for a very long time. Cryptographers of the past would devise various methods (cryptosystems) in order to make sure that a message could be concealed from everybody except from its righteous recipient. These methods combined with a secret key would help a sender to transform his message to a seemingly unreadable state (encrypt) which would be handed to the recipient. Then he and only he was supposed to be able to reverse the used process and recover the message (decrypt). At this point two natural questions arise.

Firstly, why would the recipient be the only one able to reverse the process? The answer is that even though the cryptosystem used could be publicly known,[1] he would possess some extra knowledge that no third-party had, namely the secret key. Therefore, a sender and a recipient may have to agree in advance on a secret key, which of course is a big challenge on its own.[2] A second question which could come to mind is: could somebody reveal the message without knowing the key, or even recover the key itself by just observing encrypted messages? This question lies at the core of one of the longest lasting battles in science: *Cryptographers vs Cryptanalysts*.

Cryptographers have used various techniques in order to design cryptosystems. Each such attempt was founded on the assumption that breaking the cryptosystem (i.e. recov-

[1]In some cases even the cryptosystem was assumed to be secret, but such an assumption proved to be unrealistic due to espionage.

[2]The key-exchange problem was only solved in 1976 by Diffie and Hellman in their seminal work "New directions in cryptography" [DH76].

ering the secret key or decrypting without it) would result in actually solving a “hard” problem. In the dawn of cryptography an example of what was considered “hard” was finding a rotation of the alphabet. In fact, the Caesar cipher [Sin00] was doing exactly that. It attempted to secure messages by substituting each letter in the message by the letter in a predetermined number of places forward in the alphabet. Although this was considered secure at that time, people realised that it only requires to try all 26 possible shifts of the alphabet in order to find the “correct” one. Thus, a problem, initially considered as “hard”, proved to not actually be “hard”. Nevertheless, cryptographers did not give up and struck back. An example from the Renaissance is the Vigenère cipher [Sin00]. In order to deal with the small keyspace of the Caesar cipher, the Vigenère cipher introduced the idea of using multiple rotations of the alphabet. Admittedly, this increases the difficulty of the underlying problem, but cryptanalysts found a way to break it as well. Therefore, once more an initially considered “hard” problem proved to not actually be “hard”. Moving forward to the 20th century, cryptographers utilised electro-mechanical devices in order to build even more complex cryptosystems. Probably the most well-known of them is the ENIGMA [Sin00], used by the Germans in World War II.[3] The “rule” under which ENIGMA chose how to substitute each letter was too complicated for any pen-and-paper method to threaten it. That is why cryptanalysts employed the power of machines in order to break it.

This brings us to the modern era, and especially after 1976, when public-key cryptography was born. This breakthrough in cryptography introduced a new trend in the design of cryptographic schemes. Cryptographers who design public-key cryptographic schemes nowadays, utilise number-theoretical problems instead of puzzles or complex machine designs in order to ensure their security. One of the most well-known cryptographic schemes in this category is RSA [RSA78]. The security of this scheme relies on the difficulty of factoring integers which are the product of two (big) primes. But once again the question arises, given a specific mathematical problem, can we decide if it is “hard” or not?

Usually a definite answer to this question is only possible when the answer is negative. An affirmative answer has proven to be more challenging and has only been reached in some very special cases. However, mathematical problems are not discarded if a definite answer cannot be given. Instead, we build confidence in the hardness of a problem through many years of research, which we call cryptanalysis. This process involves the extensive use of human ingenuity in order to achieve two goals:

1. Deepen our understanding of a problem by developing extra theory.
2. Build algorithms which solve the mathematical problem and analyse their performance.

As we already described in the above paragraphs whenever cryptographers deployed more complicated cryptosystems, cryptanalysts responded by more advanced cryptanalytic techniques. The same is true in our times as well. However, this time the threat seems to not be a theoretical breakthrough in our understanding of the utilised mathematical problems, but rather a technological advancement. The race for building the first large scale quantum computer is already on. Once such a device is built, there are serious

[3] For the reader who would be interested in ENIGMA and the history around it, I recommend visiting `https://www.cryptomuseum.com/crypto/enigma/index.htm`. Also, for those who would not like diving into the technical cryptographical details of this machine I suggest the excellent movie “The Imitation Game”.

reasons to believe that most of the modern, widely used cryptography will be broken. In particular, Shor [Sho94] described a quantum algorithm which can factor a product of two primes much faster than any classical algorithm. The same algorithm affects not only RSA but all widely used public-key cryptography.

Even though this might sound scary, cryptographers always struck back throughout history, and our times are no exception. The need for post-quantum cryptography has led some brave cryptographers of our times to investigate new mathematical foundations of cryptography which are believed to be secure even in the presence of quantum computers. One such option is lattice-based cryptography. However, as was previously mentioned, in practice the only way to ensure that a cryptographic scheme is secure is through extensive cryptanalysis.

This dissertation is devoted to the study of lattice problems, with the aim to contribute to the process of "extensive cryptanalysis" for lattice-based cryptography. The dissertation consists of two main parts. The first part, (Chapters 1, 2, 3) focuses on the theory of lattices and lattice problems, while the second part (Chapters 4, 5, 6) focuses more on practice (attacks).

Part 1: Lattice theory

- **Chapter 1** describes the basic notions used in the study of lattices and introduces some lattice problems of great cryptographic interest. It also mentions a few classical results and some commonly used heuristic assumptions.

- **Chapter 2** discusses the Closest Vector Problem with Preprocessing (CVPP). In particular, it describes how the *Voronoi cell* of a lattice can be used in order to solve CVPP and how the use of approximate Voronoi cells can result in improved complexities.

- **Chapter 3** introduces two notions: the *irreducible vectors* of a lattice and a *complete system of irreducible vectors*. As a first step, some main properties of these sets of lattice vectors are given. Furthermore, it is discussed how these sets relate to the set of *Voronoi relevant vectors*, sieving algorithms and the study of lattice problems.

Part 2: Lattice attacks

- **Chapter 4** describes some hybrid lattice algorithms for the Shortest Vector Problem (SVP). The hybrids are built by combining the approximate Voronoi cell techniques mentioned in Chapter 2 with lattice enumeration and Babai lifting.

- **Chapter 5** examines some combinatorial aspects of guessing a pattern of zeros in a secret lattice vector. Namely, it is examined in more detail why random patterns seem to be more likely to occur than the consecutive zeros pattern and how we could possibly optimise our choice of a pattern.

- **Chapter 6** investigates the potential benefits of using a "wider" set of lattices modelling the Ring-LWE problem. The idea is mainly examined within an "overstretched" case analysis and briefly within the framework of lattice reduction attacks.

Finally, Chapter 7 discusses open problems emerging from this dissertation.

Chapter 1

A brief introduction to lattices

In this chapter we will mention some basic notions, problems and results regarding lattices. In this way we aim at both easing the exposition of ideas in the following chapters, as well as motivating a non-expert reader to read further ahead. The interested reader can refer to [MG02] for further details on the topic.

1.1 — Basic notions and main problems

1.1.1 – Basic notions. The first definition in this thesis could be none other than that of a lattice.

Definition 1.1. *Let $\mathbb{R}^d$ be the d-dimensional Euclidean space and $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbb{R}^d$ a set of linearly independent vectors. The lattice $\mathcal{L}$ induced by $\mathbf{B}$ is the set of all integral combinations of the vectors in $\mathbf{B}$:*

$$\mathcal{L} = \mathcal{L}(\mathbf{B}) := \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\}. \tag{1.1}$$

An equivalent definition using algebraic terms is that a lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^d$. By discrete we imply that there exists a positive real $\lambda > 0$ such that around each lattice point there is a ball with radius λ that has no other lattice point in it. An example of a lattice is shown in Figure 1.1. The integer d is called the dimension of the lattice and n the rank of the lattice. If $n = d$ we call $\mathcal{L}$ a full rank lattice. The set $\mathbf{B}$ is called a basis of the lattice. We may also interpret $\mathbf{B}$ as a matrix with columns $\mathbf{b}_i$. A lattice has several bases. If $\mathbf{B}_1, \mathbf{B}_2$ are two bases of a lattice then there will exist an integral unimodular matrix $\mathbf{U}$ such that $\mathbf{B}_2 = \mathbf{B}_1\mathbf{U}$. This leads us to one of the most commonly used invariants of a lattice, its volume.

Definition 1.2 (Volume)**.** *Let $\mathcal{L}$ be a lattice in $\mathbb{R}^d$ and $\mathbf{B}$ a basis of it. We define $\mathrm{Vol}(\mathcal{L}) := \det(\mathbf{B}^\mathsf{T}\mathbf{B})^{1/2}$ to be the volume of the lattice.*

If $\mathcal{L}$ is a full rank lattice the definition reduces to $\mathrm{Vol}(\mathcal{L}) := |\det(\mathbf{B})|$. A crucial element in this definition is that the volume of the lattice is independent of the basis. A geometrical interpretation of the lattice volume is that it corresponds to the volume of the region $\{\mathbf{B}x : x \in [0,1]^n\}$ (see Figure 1.1).

As a lattice is a discrete structure, we can also define another invariant, the length of a shortest non-zero vector. Throughout this thesis we will consider problems in the Euclidean norm, unless stated otherwise.

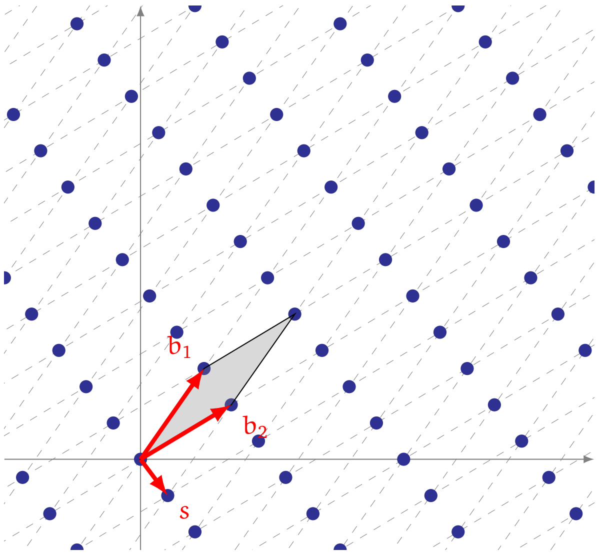


Figure 1.1: A lattice in $\mathbb{R}^2$.

Definition 1.3 (First minimum). *Let $\mathcal{L}$ be a lattice. We define $\lambda_1(\mathcal{L}) := \min_{\mathbf{v} \in \mathcal{L}\setminus\{\mathbf{0}\}} \|\mathbf{v}\|$. We call $\lambda_1(\mathcal{L})$ the first successive minimum of $\mathcal{L}$.*

It can be the case that many lattice vectors reach the first successive minimum.

Definition 1.4 (First shell). *Let $\mathcal{L}$ be a lattice. We define*

$$S_1(\mathcal{L}) := \{\mathbf{v} \in \mathcal{L} \mid \|\mathbf{v}\| = \lambda_1(\mathcal{L})\}. \tag{1.2}$$

We call $S_1(\mathcal{L})$ the first shell of $\mathcal{L}$.

Like the first successive minimum we can further define the i-th successive minimum of $\mathcal{L}$ for $i \leqslant n$.

Definition 1.5 (Successive minima). *Let $\mathcal{L}$ be a lattice of rank n. For $i = 1, 2, \ldots, n$ we define the i-th successive minimum of $\mathcal{L}$ to be*

$$\lambda_i(\mathcal{L}) := \min\{\max\{\|\mathbf{x}_1\|, \ldots, \|\mathbf{x}_i\|\} \mid \mathbf{x}_1, \ldots, \mathbf{x}_i \in \mathcal{L} \textit{ are linearly independent}\}. \tag{1.3}$$

The first successive minimum indicates how close a lattice vector can be to the origin. However, we can also consider the case where it is required to find a lattice vector not closest to $\mathbf{0}$ but to a given vector $\mathbf{t}$ in $\mathbb{R}^d$.

Definition 1.6 (Covering radius). *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^d$. We define*

$$\mu(\mathcal{L}) \coloneqq \max_{\mathbf{t}\in\mathbb{R}^d} \min_{\mathbf{v}\in\mathcal{L}} \|\mathbf{t}-\mathbf{v}\| \tag{1.4}$$

and $\mu(\mathcal{L})$ is called the covering radius of the lattice.

For the sake of simplicity the aforementioned definition of the covering radius is given for a full rank lattice. However it can be easily adapted to the case of a non-full rank lattice. This can be done by replacing $\mathbb{R}^d$ with the linear span of a basis of the lattice. Therefore in both cases the covering radius will indicate how far from the lattice a point of its linear span can be.

Finally, we will give the definition of the dual of a lattice. Even though this notion might seem complicated or counter-intuitive at first glance, it has been used a lot in the literature.

Definition 1.7 (Dual lattice). *The dual of a lattice $\mathcal{L}$ is the set of all vectors $\mathbf{x}$ in the linear span of $\mathcal{L}$ such that $\langle \mathbf{x}, \mathbf{v}\rangle \in \mathbb{Z}$ for all $\mathbf{v} \in \mathcal{L}$. We denote the dual lattice of $\mathcal{L}$ by $\mathcal{L}^*$.*

A result which helps in making the definition of the dual lattice more explicit is the following. If $\mathbf{B}$ is a basis of a lattice $\mathcal{L}$ then $\mathbf{B}(\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}$ is a basis for $\mathcal{L}^*$.

Having in place enough definitions regarding a lattice, we are ready to define some of the most common lattice problems in the literature.

1.1.2 – Lattice problems. We will start our exposition of lattice problems with those regarding finding short lattice vectors.

Definition 1.8 (Shortest Vector Problem, SVP). *Given a lattice $\mathcal{L}$, find a non-zero lattice vector $\mathbf{s} \in \mathcal{L}$ satisfying $\|\mathbf{s}\| = \lambda_1(\mathcal{L})$.*

In some cases, an exact solution to SVP might be "too hard" to find, or even not necessary and thus we may resort to an approximate version of the problem.

Definition 1.9 (Approximate Shortest Vector Problem, SVP$_\gamma$). *Given a lattice $\mathcal{L}$ and an approximation factor $\gamma \geqslant 1$, find a non-zero vector $\mathbf{s} \in \mathcal{L}$ such that $\|\mathbf{s}\| \leqslant \gamma \cdot \lambda_1(\mathcal{L})$.*

The SVP and SVP$_\gamma$ ask for a lattice vector achieving the first successive minimum $\lambda_1(\mathcal{L})$ or one approximating it respectively. However, if a "short" lattice vector is given, it is required to know the value of $\lambda_1(\mathcal{L})$ in order to verify if it constitutes a solution to the problem. In order to circumvent the computation of $\lambda_1(\mathcal{L})$ we can ask for short lattice vectors with regard to a lattice invariant which is easily computable, its volume.

Definition 1.10 (Hermite Shortest Vector Problem, HSVP$_\gamma$). *Given a lattice $\mathcal{L}$ of rank n and an approximation factor $\gamma \geqslant 1$, find a non-zero vector $\mathbf{s} \in \mathcal{L}$ such that* $\|\mathbf{s}\| \leqslant \gamma \cdot \mathrm{Vol}(\mathcal{L})^{1/n}$.

Apart from stating lattice problems where only one vector is required as output, problems asking for a set of lattice vectors can be of interest. One such problem is the following.

Definition 1.11 (Shortest Independent Vector Problem, SIVP). *Given a lattice $\mathcal{L}$ of rank n, find n linearly independent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathcal{L}$ such that* $\max_{1\leqslant i\leqslant n} \|\mathbf{v}_i\| \leqslant \lambda_n(\mathcal{L})$.

Following the same path as in the previous subsection, we will move from problems asking for lattice vectors close to the origin, to lattice problems asking for lattice vectors close to a random vector in $\mathbf{t} \in \mathbb{R}^d$. Again in order to ease the exposition we will give definitions for full rank lattices. Of course these definitions can be extended to the case of non-full rank lattices.

Definition 1.12 (Closest Vector Problem, CVP). *Given a full rank lattice $\mathcal{L}$ in R^d and a target vector $\mathbf{t} \in \mathbb{R}^d$, find a lattice vector $\mathbf{s} \in \mathcal{L}$ satisfying $\|\mathbf{t}-\mathbf{s}\| = \min_{\mathbf{v}\in\mathcal{L}} \|\mathbf{v}-\mathbf{t}\|$.*

Like in the case of SVP an approximate solution to the problem may suffice, therefore an approximate version of the problem is given.

Definition 1.13 (Approximate Closest Vector Problem, CVP_γ). *Given a full rank lattice $\mathcal{L}$ in R^d, a target vector $\mathbf{t} \in \mathbb{R}^d$, and an approximation factor $\gamma \geqslant 1$, find a vector $\mathbf{s} \in \mathcal{L}$ with $\|\mathbf{s}-\mathbf{t}\| \leqslant \gamma \cdot \min_{\mathbf{v}\in\mathcal{L}} \|\mathbf{v}-\mathbf{t}\|$.*

An important difference between SVP and CVP is that the second depends on a "target vector" $\mathbf{t} \in \mathbb{R}^d$. Hence, a preprocessing version of the problem can be defined. In the preprocessing variant of CVP (CVPP), one is allowed to preprocess the lattice $\mathcal{L}$, and use the preprocessed data to solve a CVP instance for $\mathbf{t}$. This problem naturally comes up in contexts where either $\mathcal{L}$ is known long before $\mathbf{t}$ is known, or if a large number of CVP instances on the same lattice are to be solved.

Definition 1.14 (Closest Vector Problem with Preprocessing, CVPP). *Let $\mathcal{L}$ be a full rank lattice in R^d and Π be a preprocessing function, computing the preprocessing data $\Pi(\mathcal{L})$. Given a target vector $\mathbf{t} \in \mathbb{R}^d$ find a lattice vector $\mathbf{s} \in \mathcal{L}$ satisfying $|\mathbf{t}-\mathbf{s}| = \min_{\mathbf{v}\in\mathcal{L}} |\mathbf{v}-\mathbf{t}|$ using the preprocessing data $\Pi(\mathcal{L})$.*

We will conclude this section with a remark on the theoretical vs the practical use of the definitions we gave in this section. In all cases we considered the "general" case where a lattice $\mathcal{L}$ was a subset of $\mathbb{R}^d$. However when computational problems are processed in practice it is easier to work with rational or even integer arithmetic instead of floating point. That is why it is common when lattices are used in practical applications to assume that $\mathcal{L} \subseteq \mathbb{Z}^d$ instead of $\mathcal{L} \subseteq \mathbb{R}^d$.[1]

1.2 — Classical results and heuristic assumptions

In this section we will mention a couple of classical results/assumptions which are used in the study of lattices. Again, we will start with results regarding short vectors of the lattice. A natural question to ask would be if there is an upper bound to $\lambda_1(\mathcal{L})$. Hermite gave an upper bound on $\lambda_1(\mathcal{L})$ depending on the volume of $\mathcal{L}$.

Theorem 1.15 (Hermite). *For each $d \in \mathbb{N}$ there exists a constant γ_d such that for any full rank lattice in $\mathbb{R}^d$ it holds that*

$$\lambda_1(\mathcal{L}) \leqslant {\gamma_d}^{1/2}\mathrm{Vol}(\mathcal{L})^{1/d}. \tag{1.5}$$

[1]A lattice $\mathcal{L}$ with $\mathcal{L} \subseteq \mathbb{Q}^d$ can be associated to one $\mathcal{L}' \subseteq \mathbb{Z}^d$ by proper scaling which again allows to work with integers.

We notice that Theorem 1.15 justifies the definition of problem 1.10 as it provides a guarantee that for $\gamma \geqslant {\gamma_d}^{1/2}$ there always exists a solution to HSVP$_\gamma$.

Definition 1.16. *The supremum of $\lambda_1(\mathcal{L})^2/\mathrm{Vol}(\mathcal{L})^{2/d}$ over all rank-d lattices $\mathcal{L}$ is denoted by γ_d and called Hermite's constant for dimension d.*

Asymptotic bounds on γ_d show that it actually is linear in d (see [Ngu09]). However, the exact value of γ_d is only known for dimensions $1 \leqslant d \leqslant 8$ and $d = 24$.

Even though Theorem 1.15 is a nice result for bounding $\lambda_1(\mathcal{L})$, a similar result cannot occur for rest of the successive minima. The classical counterexample is the lattice generated by

$$B = \begin{pmatrix} \epsilon & 0 \\ 0 & 1/\epsilon \end{pmatrix} \text{ with } \epsilon > 0. \tag{1.6}$$

As it becomes clear for this example the volume of the lattice is equal to 1 but for small ϵ we get $\lambda_2(\mathcal{L}) = 1/\epsilon$ which can become arbitrarily large. However, by Minkowski we know the following theorem which bounds the norms of the successive minima in a "collective way".

Theorem 1.17 (Minkwoski)**.** *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^d$. Then*

$$\prod_{i=1}^{d} \lambda_i(\mathcal{L}) \leqslant {\gamma_d}^{d/2}\,\mathrm{Vol}(\mathcal{L}). \tag{1.7}$$

When studying lattices in practice, apart from exact results, some so called heuristic assumptions are used. These assumptions will probably not be true for all lattices. However they will closely approximate some property for "most" lattices. In this way a specific property of a lattice, or a lattice algorithm, can be more easily studied/analysed in what is commonly referred as the "average case". Such an assumption commonly used in the literature is the Gaussian heuristic.

Assumption 1.18 (Gaussian heuristic)**.** *Given a full-rank lattice $\mathcal{L}$ and a region $\mathcal{A} \subset \mathbb{R}^n$, the (expected) number of lattice points in $\mathcal{A}$, denoted $|\mathcal{A} \cap \mathcal{L}|$, satisfies:*

$$|\mathcal{A} \cap \mathcal{L}| \approx \frac{\mathrm{Vol}(A)}{\mathrm{Vol}(\mathcal{L})}. \tag{1.8}$$

Remark 1.19. *As it was mentioned earlier, heuristic assumptions can be precise for some families of lattices but imprecise for some others. We refer to [BL21] for a detailed analysis formalising when the Gaussian heuristic is accurate and investigating for which families of lattices.*

Using volume arguments, the Gaussian heuristic predicts that $\lambda_1(\mathcal{L}) = \mathrm{gh}(\mathcal{L})$ where $\mathrm{gh}(\mathcal{L}) := \sqrt{d/(2\pi e)} \cdot \mathrm{Vol}(\mathcal{L})^{1/d} \cdot (1 + o(1))$. For average-case, random CVP(P) target instances $\mathbf{t} \in \mathbb{R}^d$, this further means that we expect the distance to the closest lattice point to be roughly $\lambda_1(\mathcal{L})$: any smaller ball around $\mathbf{t}$ of radius $(1-\varepsilon)\lambda_1(\mathcal{L})$ is expected to be empty, and a bigger ball of radius $(1+\varepsilon)\lambda_1(\mathcal{L})$ will likely contain up to $(1+\varepsilon)^{d+o(d)}$ (exponentially many) lattice points for a random lattice $\mathcal{L}$.

As it was mentioned earlier, each lattice has infinitely many bases. Usually we are interested in a "good" basis of the lattice, which roughly speaking means that the basis vectors are short and close to orthogonal. In order to compute such nice bases of a lattice we deploy lattice basis reduction algorithms. The most famous and widely used are the LLL algorithm [LLL82] and a generalisation of it, the BKZ algorithm [Sch87]. In particular, the BKZ algorithm requires an "SVP oracle" solving SVP in some dimension β, that is why we adopt the notation of a BKZ-β reduced basis. In addition, a basis which is BKZ-β reduced for some "big" β, will be called a strongly reduced basis.

For a basis $\mathbf{B} = \{\mathbf{b}_0, \dots, \mathbf{b}_{d-1}\}$ and $i \in \{0, 1, \dots, d-1\}$ we define π_i as the projection onto the orthogonal complement of $\{\mathbf{b}_0, \dots, \mathbf{b}_{i-1}\}$. The Gram-Schmidt vectors $\mathbf{b}_0^*, \dots, \mathbf{b}_{d-1}^*$ are defined as $\mathbf{b}_i^* \coloneqq \pi_i(\mathbf{b}_i)$. The sequence $(\|\mathbf{b}_i^*\|)_{i=0}^{d-1}$ is called the *profile* of a basis. The profile of a lattice basis is related to the volume of the lattice by the following formula: $\mathrm{Vol}(\mathcal{L}) = \prod_{i=0}^{d-1} \|\mathbf{b}_i^*\|$. A way to measure the quality of a basis is via the behaviour of its profile. In particular, we examine if it decreases too fast. A commonly used heuristic assumption on the profile of a BKZ-reduced basis is the Geometric Series Assumption (GSA).

Assumption 1.20 (Geometric Series Assumption). *Let $\mathbf{B}$ be a BKZ-β reduced basis of a full rank d-dimensional lattice. Then the Gram-Schmidt vectors $\mathbf{b}_i^*$ satisfy*

$$\|\mathbf{b}_i^*\| = \alpha_\beta^{(d-1-2i)/2} \mathrm{Vol}(\mathcal{L})^{1/d}$$

where $\alpha_\beta \approx (\beta/2\pi e)^{1/(\beta-1)}$.

Remark 1.21. *The value of α_β given in Assumption 1.20 is derived by an asymptotic estimate. For small values of β (say $\beta < 50$) it will not give a good approximation of a basis' profile.*

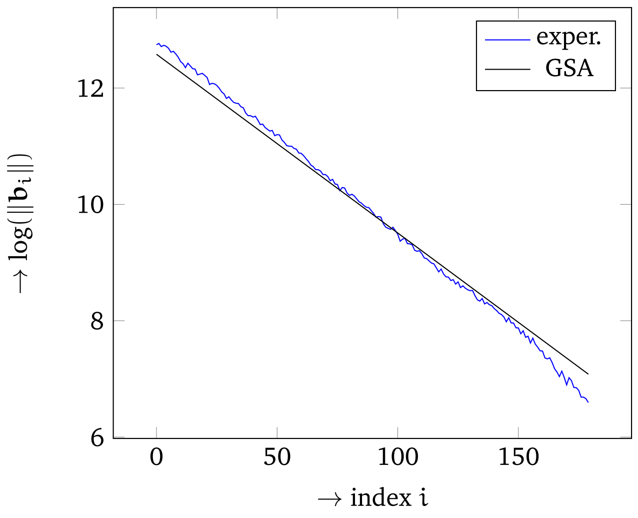


Figure 1.2: A comparison of a BKZ-60 reduced basis' profile to the prediction of the GSA. For this particular example we reduced a basis of a 180-dimensional lattice with BKZ using block size 60. The picture is drawn in logarithmic scale.

1.3 — Structured lattices

So far we have considered a basis of a lattice as its representation. If $\mathcal{L}$ is a full rank lattice in $\mathbb{Z}^d$ then in order to store (or transmit) it we would need to handle d^2 integers. If $d = 1000$ then it would take a million (10^6) integers in order to represent the lattice. Even though in general this is not a problem with modern technology, it may not be very "convenient" once we start considering lattices within cryptographic applications.

One reason why it may not be so convenient, is that nowadays there is a wide variety of devices using cryptographic schemes. Many of these devices may have a restricted memory capacity (e.g. embedded devices), unlike a laptop for example. Therefore, for a full-scale deployment of a lattice-based scheme it would be preferred to make it fit in as many frameworks as possible. A second reason is more of a matter of comparison to the currently used cryptography. The cryptographic schemes used at the moment store keys of at most a few KB. If a (new) lattice-based scheme requires a few MB for storing a key, this may be considered as loss in the performance of the used cryptography. Therefore, the question arises: could it be possible to represent a lattice in a more compact way?

The answer to this question is positive for some lattices and we will mention two examples from the literature. Such examples emerge from the use of algebraic objects, in this case, rings. The underlying structure of a ring allows a more compact representation of a lattice, reducing the memory requirements.

The first such example we will mention is NTRU [HPS98].

Definition 1.22 (The NTRU Problem). *Let n be a prime, q a positive integer and let $\mathbf{f}, \mathbf{g} \in \mathbb{Z}_q[X]$ be polynomials of degree n with small coefficients sampled from some distribution χ under the condition that $\mathbf{f}$ is invertible in $R_q := \mathbb{Z}_q[X]/\langle X^n - 1\rangle$. The pair $(\mathbf{f}, \mathbf{g})$ forms the secret key and the public key is defined as $\mathbf{h} := \mathbf{g}\mathbf{f}^{-1}$ in R_q. The NTRU problem is to recover any rotation $(X^i\mathbf{f}, X^i\mathbf{g})$ of the secret key from $\mathbf{h}$.*

Usually, the polynomials $\mathbf{f}, \mathbf{g}$ are chosen to have coefficients in $\{0, \pm 1\}$ with each value occurring about $n/3$ times. In the definition of the NTRU problem, no lattices were involved. Nevertheless, we can reduce the NTRU problem to a lattice problem by defining the NTRU lattice. In order to do so, it is useful to consider the multiplication matrix of an element in R_q. For $\mathbf{a} \in R_q$ we set $\mathbf{M}(\mathbf{a})$ to be the multiplication matrix of $\mathbf{a}$, i.e. the j-th column is formed by the coefficients of $X^j\mathbf{a}$.

Definition 1.23. *Let $(n, q, \mathbf{f}, \mathbf{g}, \mathbf{h})$ be an NTRU instance. We define the NTRU lattice as*

$$\mathcal{L}^{\mathbf{h},q} := \begin{pmatrix} q\mathbf{I}_n & \mathbf{M}(\mathbf{h}) \\ \mathbf{0} & \mathbf{I}_n \end{pmatrix} \cdot \mathbb{Z}^{2n}. \tag{1.9}$$

If we identify $\mathbf{f}, \mathbf{g}$ with their coefficient vectors, then $(\mathbf{g}|\mathbf{f})^T$ is an exceptionally short vector in the lattice $\mathcal{L}^{\mathbf{h},q}$ as:

$$\begin{pmatrix} q\mathbf{I}_n & \mathbf{M}(\mathbf{h}) \\ \mathbf{0} & \mathbf{I}_n \end{pmatrix} \begin{pmatrix} \mathbf{k} \\ \mathbf{f} \end{pmatrix} = \begin{pmatrix} \mathbf{g} \\ \mathbf{f} \end{pmatrix} \quad \text{for some } \mathbf{k} \in \mathbb{Z}^n.$$

Hence, the NTRU problem is reduced to finding a short vector in a lattice.

Remark 1.24. *The ring R_q could be defined for other choices of a modulus polynomial apart from $X^n - 1$. Other options might actually be even more secure (e.g. see [BCLv19]).*

There are two special properties of the NTRU lattice which are worth mentioning. Initially, the NTRU lattice is a q-ary lattice.

Definition 1.25 (q-ary lattice)**.** *A lattice $\mathcal{L}$ of dimension d is called q-ary if for some $q > 0$ we have*

$$q\mathbb{Z}^d \subseteq \mathcal{L} \subseteq \mathbb{Z}^d.$$

Also the NTRU lattice is a cyclic lattice.

Definition 1.26. *A lattice $\mathcal{L}$ is called cyclic if for every lattice vector $\mathbf{v} \in \mathcal{L}$, all the vectors obtained by cyclically rotating the coordinates of $\mathbf{v}$ also belong to $\mathcal{L}$.*

A second example taking advantage of the algebraic structure of rings is the Ring Learning With Errors problem (Ring-LWE). Let q be a prime and $f(X) \in \mathbb{Z}[X]$ a polynomial of degree n which is also irreducible in $\mathbb{Z}[X]$. We set $R = \mathbb{Z}[X]/\langle f(X)\rangle$ and $R_q = \mathbb{Z}_q[X]/\langle f(X)\rangle$. In order to ease the exposition we will consider the specific case $f(X) = X^n + 1$ with $n = 2^l$, known as the $2n$-th cyclotomic polynomial.

Definition 1.27 (The Ring-LWE Problem)**.** *Let $\mathbf{a}, \mathbf{e} \in R$ with "small" coefficients, sampled according to some distributions χ_1 and χ_2.*[2] *Let $\mathbf{G}$ be a uniformly random element of R_q and $\mathbf{A} \in R_q$ such that $\mathbf{aG} + \mathbf{e} = \mathbf{A}$ in R_q. Given, $\mathbf{G}, \mathbf{A}$ find $\mathbf{a}$.*

In order to express Ring-LWE in terms of a lattice problem we define the lattice generated by the columns of the following matrix,

$$\mathbf{B} = \begin{pmatrix} q\mathbf{I}_n & \mathbf{A} & \mathbf{M}(-\mathbf{G}) \\ \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_n \end{pmatrix}. \tag{1.10}$$

If we consider $\mathbf{a}, \mathbf{e}$ as vectors, then $(\mathbf{e}|1|\mathbf{a})$ is an exceptionally short vector in the lattice $\mathcal{L}(\mathbf{B})$. Hence, the Ring-LWE problem is reduced to a lattice problem.

Remark 1.28. *The matrix $\mathbf{B}$ given above generates the most commonly used lattice for modelling Ring-LWE as a lattice problem. However, it is not the only one that can be used, more options are possible. This is a topic examined in Chapter 6.*

We note that using the aforementioned lattice for modelling Ring-LWE does not allow the formation of a cyclic structure in the lattice.

Concluding we can claim that both for NTRU and Ring-LWE one or two ring elements were used respectively in order to encode a lattice problem. These ring elements take $O(n)$ space while the corresponding lattices are of size $O(n^2)$. Therefore the representation of the underlying lattices via the ring elements is smaller by a factor of $O(n)$ compared to any other random lattice in the same dimension. Hence, an improvement is achieved.

[2]Here "small" usually refers to some set like $\{0, \pm 1\}$.

Further background. As it was mentioned at the beginning of this chapter, its purpose is to provide a brief introduction to the subject of lattices. This implies that later chapters will present background specific to the scientific publications they cover. In particular, even though in this chapter we briefly discussed about lattice basis reduction, we did not mention anything about algorithms solving problems like SVP and CVP. This will be done in the following chapters. We refer to Section 2.2 for a discussion on lattice sieving and to Section 4.2.2 for an brief introduction to lattice enumeration. Finally we refer to Section 2.4 for an introduction to Voronoi cells.

Chapter 2

Finding closest lattice vectors using approximate Voronoi cells

This chapter presents parts of the paper *Finding closest lattice vectors using approximate Voronoi cells* [DLdW19] authored jointly with Thijs Laarhoven and Benne de Weger, which was published at PQCrypto 2019. As this chapter merely *presents* some parts of the paper, proofs of results will not be included. We refer to [DLdW19] for the proofs.

2.1 — Introduction

Given a basis of a lattice $\mathcal{L}$, the shortest vector problem (SVP) asks to find a shortest non-zero vector in $\mathcal{L}$ under the Euclidean norm, i.e., a non-zero lattice vector $\mathbf{s}$ of norm $\|\mathbf{s}\| = \lambda_1(\mathcal{L})$. Given a basis of a lattice and a target vector $\mathbf{t} \in \mathbb{R}^d$, the closest vector problem (CVP) asks to find a lattice vector $\mathbf{s} \in \mathcal{L}$ closest to $\mathbf{t}$. The preprocessing variant of CVP (CVPP) permits to preprocess the lattice $\mathcal{L}$ such that, when later given a target vector $\mathbf{t}$, one can quickly find a closest lattice vector to $\mathbf{t}$.

SVP and CVP are fundamental in the study of lattice-based cryptography, as the security of many schemes is directly related to their hardness. Various other hard lattice problems, such as Learning With Errors (LWE), are closely related to SVP and CVP; see e.g. [Ste16] for reductions among lattice problems. These reductions show that understanding the hardness of SVP and CVP is crucial for accurately estimating the security of lattice-based cryptographic schemes.

Worst-case SVP/CVP analyses. Although SVP and CVP are both central in the study of lattice-based cryptography, algorithms for SVP have received somewhat more attention, including a benchmarking website to compare different methods [svp19]. Various SVP algorithms have been studied which can solve CVP as well, such as the polynomial-space, superexponential-time lattice enumeration (see Section 4.2.2) studied in e.g. [Kan83, FP85, GNR10, MW15, AN17]. More recently, methods have been proposed which solve SVP/CVP in only single exponential time, but which also require exponential-sized memory [AKS01, MV10a, ADRS15]. By constructing the Voronoi cell of the lattice, Micciancio–Voulgaris [MV10a] showed that SVP and CVP(P) can provably be solved in time $2^{2d+o(d)}$, and Bonifas–Dadush [BD15] reduced the complexity for CVPP to only $2^{d+o(d)}$. In high dimensions the best provable complexities for SVP and CVP are currently due to discrete

Gaussian sampling [ADRS15, ADSD15], solving both problems in $2^{d+o(d)}$ time and space in the worst case on arbitrary lattices.

Average-case SVP/CVP algorithms. When considering and comparing these methods in practice on random lattices, we get a completely different picture. Currently the fastest heuristic methods for SVP and CVP in high dimensions are based on lattice sieving (see Section 2.2). After a long series of theoretical works on constructing efficient heuristic sieving algorithms like e.g. [NV08, MV10b, Laa15, BDGL16] as well as applied papers studying how to further speed up these algorithms in practice e.g. [LM18, Duc18], the best heuristic time complexity for solving SVP (and CVP [Laa16b]) currently stands at $2^{0.292d+o(d)}$ [BDGL16], using $2^{0.208d+o(d)}$ memory. The highest records in the SVP challenge [svp19] were recently obtained using a BKZ-sieving hybrid [ADH+19]. These recent improvements have resulted in a shift in security estimates for lattice-based cryptography, from estimating the hardness of SVP/CVP using the best enumeration methods, to estimating this hardness based on state-of-the-art sieving results (e.g. [ADPS16]) .

Hybrid algorithms and batch-CVP. In moderate dimensions, enumeration-based methods dominated for a long time, and the cross-over point with single-exponential time algorithms like sieving seemed to be far out of reach [MW15]. Moreover, the exponential memory of e.g. lattice sieving will ultimately also significantly slow down these algorithms due to the large number of random memory accesses [BCLV17]. Some previous work focused on obtaining a tradeoff between enumeration and sieving, using less memory for sieving [BLS16, HK17, HKL18].

Another well-known direction for a hybrid between memory-intensive methods and enumeration is to use a fast CVP(P) algorithm as a subroutine within enumeration (see Chapter 4). As described in e.g. [GNR10, MW15], at any given level in the enumeration tree, one is attempting to solve a CVP instance in a lower-rank sublattice, where the target vector is determined by the path from the root to the current node in the tree. Each node at this level in the tree corresponds to a CVP instance in the same sublattice, but with a different target. If we can preprocess this low-dimensional sublattice such that the amortized time complexity of solving a batch of CVP-instances in this sublattice is small, then this may speed up processing the bottom part of the enumeration tree.

A first step in this direction was taken in [Laa16b], where it was shown that with a sufficient amount of preprocessing and space, one can achieve better amortized time complexities for batch-CVP than when solving just one instance. The large memory requirement (at least $2^{d/2+o(d)}$ memory is required to improve upon direct CVP approaches) as well as the large number of CVP instances required to get a lower amortized complexity made this approach impractical to date.

In this chapter we study CVPP in terms of approximate Voronoi cells, and observe better time and space complexities using randomized slicing, which is similar in spirit to using randomized bases in lattice enumeration [GNR10].

2.1.1 – Notation. We write vectors in boldface (e.g. $\mathbf{x}$), and we denote their coordinates with non-boldface subscripts (e.g. x_i). Throughout the chapter we primarily consider problems in the Euclidean norm, hence unless stated otherwise, $\|\mathbf{x}\| = \|\mathbf{x}\|_2 := (\sum_i x_i^2)^{1/2}$ denotes the Euclidean norm of the vector $\mathbf{x}$. We write $\mathcal{S}^{d-1}$ for the Euclidean unit sphere in $\mathbb{R}^d$, i.e. the set of vectors $\mathbf{x} \in \mathbb{R}^d$ with $\|\mathbf{x}\| = 1$. We denote balls in high-dimensional space by $\mathcal{B}(\mathbf{x}, r) := \{\mathbf{y} \in \mathbb{R}^d : \|\mathbf{y} - \mathbf{x}\| \leqslant r\}$, and we write $\mathcal{H}(\mathbf{x}) := \{\mathbf{v} \in \mathbb{R}^d : \|\mathbf{v}\| \leqslant \|\mathbf{v} - \mathbf{x}\|\}$ for half-spaces whose boundaries are the hyperplanes orthogonal to $\mathbf{x}$ and passing through $\frac{1}{2}\mathbf{x}$. For regions $\mathcal{R} \subset \mathbb{R}^d$, we denote their volume by $\mathrm{Vol}(\mathcal{R})$.

Given a parameter $s > 0$, we define $\rho_s(\mathbf{v}) := \exp(-\pi\|\mathbf{v}\|^2/s^2)$, and given a lattice $\mathcal{L}$ we define $\rho_s(\mathcal{L}) := \sum_{\mathbf{v} \in \mathcal{L}} \rho_s(\mathbf{v})$. We define a discrete probability distribution on this lattice $\mathcal{L}$ by setting $\Pr(\mathbf{X} = \mathbf{x}) := \rho_s(\mathbf{x})/\rho_s(\mathcal{L})$ for $\mathbf{x} \in \mathcal{L}$ and $\Pr(\mathbf{X} = \mathbf{x}) := 0$ if $\mathbf{x} \notin \mathcal{L}$. We denote this distribution as the discrete Gaussian distribution on $\mathcal{L}$ with parameter s, and we write $\mathbf{X} \sim D_{\mathcal{L},s}$ to denote that the random variable $\mathbf{X}$ follows this distribution. For cosets of a lattice $\mathbf{t} + \mathcal{L}$, we analogously define $D_{\mathbf{t}+\mathcal{L},s}$ with relative density $\rho_{s,\mathbf{t}}(\mathbf{v}) := \exp(-\pi\|\mathbf{v} - \mathbf{t}\|^2/s^2)$.

2.2 — Preliminaries

2.2.1 – Heuristic assumptions. As discussed in the introduction, worst-case analyses of algorithms for e.g. SVP and CVP(P) are far off from the best average-case performance we can achieve in practice by just testing these algorithms on random lattices. For purposes in cryptography, where it is in a sense better to be safe than sorry, it therefore makes sense to try to analyze algorithms under “mild” assumptions that allow us to obtain tighter estimates on their performance on average-case lattices. Even if we can no longer formally prove these complexity bounds hold in the worst-case—indeed, these complexities may not even be accurate for e.g. exotic, dense lattices like the Leech lattice [CS98]—such estimates may give us a better idea of the actual performance of the best algorithms on random lattices appearing in cryptanalysis.

A commonly made heuristic assumption for analyzing lattice algorithms is the Gaussian heuristic 1.18. A consequence of this assumption is that for random lattices of high dimension d, the length of the shortest vector can be approximated as:

$$\lambda_1(\mathcal{L}) \approx \sqrt{\frac{d}{2\pi e}} \cdot \det(\mathcal{L})^{1/d} \cdot (1 + o(1)). \tag{2.1}$$

For average-case, random CVP(P) target instances $\mathbf{t} \in \mathbb{R}^d$, this further means that we expect the distance to the closest lattice point to be roughly $\lambda_1(\mathcal{L})$ as was already mentioned in Section 1.2.

When working with lattice vectors $\mathbf{v} \in \mathcal{L}$, even if for some algorithm we know the distribution of the “input vectors” over the lattice, after modifying these vectors we quickly lose track of the actual distribution these vectors now follow. A common assumption here is then to simply assume that if at some point in the execution of the algorithm, we are left with a vector $\mathbf{v}' \in \mathcal{L}$ of norm $\|\mathbf{v}'\|$, then this vector follows a uniform distribution over the sphere of radius $\|\mathbf{v}'\|$ around the origin. Clearly this assumption is incorrect and ignores the discrete nature of the lattice (which may play a bigger role as the radius gets smaller), but unless this inaccuracy is exploited and abused in the analysis, this often gives us a better grip on e.g. the probability that two vectors $\mathbf{v}, \mathbf{w}$ appearing in a lattice algorithm can be combined to form a shorter vector $\mathbf{v} \pm \mathbf{w}$.

Finally, observe that although these heuristic assumptions may not be provably accurate for all lattices, extensive experimentation with these algorithms on random lattices has supported these claims for average-case lattices.

2.2.2 – Lattice sieving algorithms. Heuristic lattice sieving algorithms for solving SVP are based on the following two principles: (1) if $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{L}$, then their sum/difference $\boldsymbol{v} \pm \boldsymbol{w}$ is also a lattice vector; and (2) if we have a sufficiently long list L of lattice vectors, then we expect there to be pairs $\boldsymbol{v}, \boldsymbol{w} \in L$ with $\|\boldsymbol{v} \pm \boldsymbol{w}\| < \|\boldsymbol{v}\|, \|\boldsymbol{w}\|$. This intuitively describes the approach: we first generate a sufficiently long list of lattice vectors, and then keep combining pairs of vectors in our list to form shorter and shorter lattice vectors until we (hopefully) find a shortest lattice vector in our list.

To make sure the algorithm makes progress in finding shorter lattice vectors, L needs to contain exponentially many lattice vectors; for vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{L}$ of similar norm, the vector $\boldsymbol{v} - \boldsymbol{w}$ is shorter than $\boldsymbol{v}, \boldsymbol{w}$ iff the angle between $\boldsymbol{v}, \boldsymbol{w}$ is smaller than $\pi/3$, which for random vectors $\boldsymbol{v}, \boldsymbol{w}$ of similar norm would occur with probability $(3/4)^{d/2+o(d)}$. Under the aforementioned heuristic assumption, that when normalized, vectors in L follow the same distribution as vectors sampled uniformly at random from the unit sphere, this then also models the probability that two vectors in our list can reduce one another.

The expected space complexity of heuristic sieving algorithms follows from the previous observation: if we sample $(4/3)^{d/2+o(d)}$ vectors uniformly at random from the unit sphere, then we expect a significant number of pairs of vectors to have angle less than $\pi/3$, leading to many short difference vectors. Therefore, if we start by sampling a list L of $(4/3)^{d/2+o(d)}$ rather long lattice vectors, and iteratively consider combinations of vectors in L to find shorter vectors (and replace the longer vector with the shorter combination), we expect to keep making progress. Combining all pairs of vectors in a list of size $(4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}$ naively takes time $(4/3)^{d+o(d)} \approx 2^{0.415d+o(d)}$.

The Nguyen–Vidick sieve. The heuristic sieve of Nguyen and Vidick [NV08] starts by sampling a list L of $(4/3)^{d/2+o(d)}$ reasonably long lattice vectors, sampled from a discrete Gaussian $D_{\mathcal{L},s}$ with the standard deviation s chosen such that (1) we can efficiently sample from this distribution, and (2) the returned vectors are at most of norm $2^{O(d)}\lambda_1(\mathcal{L})$. Then we use a *sieve* to map L, with some maximum norm $R := \max_{\boldsymbol{v} \in L} \|\boldsymbol{v}\|$, to a new list L', with maximum norm at most $R' := \gamma R$ for a geometric factor $0 \ll \gamma < 1$ close to 1. By repeatedly applying this sieve operation, after $\text{poly}(d)$ iterations we expect to find a long list of lattice vectors of norm at most $\gamma^{\text{poly}(d)} R = O(\lambda_1(\mathcal{L}))$, which then (with high probability) contains a shortest vector in the lattice.

Algorithm 2.1 describes a variant of Nguyen–Vidick's original sieve, to map L to L' in $|L|^2$ time (ignoring costs polynomial in d). The presented algorithm is a more intuitive version of the original sieve; see [Laa15, Appendix B] for details on this equivalence. Without any further modifications to this algorithm, the heuristic complexities for solving SVP with this method are as follows [NV08, Section 4].

Lemma 2.1 (Complexities of the Nguyen–Vidick sieve). *Heuristically, the Nguyen–Vidick sieve solves SVP in space S and time T, with*

$$S = (4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}, \qquad T = (4/3)^{d+o(d)} \approx 2^{0.415d+o(d)}. \tag{2.2}$$

By applying more sophisticated techniques for indexing the list L and searching for pairs of vectors that can be combined to form shorter vectors, the time complexity can be further reduced to $(3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$ [BDGL16]. The following result (a restatement of [BDGL16, Corollary 8]) shows that this can be done without increasing the space complexity.

Algorithm 2.1 The Nguyen–Vidick sieve for finding shortest vectors [NV08]

```
Require: An LLL-reduced basis $B$ of a lattice $\mathcal{L}(B)$
Ensure: The algorithm finds a shortest lattice vector
 1: Initialize empty lists $L, L'$ and set $\gamma \leftarrow 1 - 1/d$
 2: Sample $(4/3)^{d/2+o(d)}$ lattice vectors and add them to $L$
 3: Set $R \leftarrow \max_{w \in L} \|w\|$
 4: repeat
 5:     for each $w_1, w_2 \in L$ do            ▷ NNS techniques can be used to speed this up
 6:         if $\|w_1 - w_2\| < \gamma R$ then
 7:             Add $w_1 - w_2$ to the list $L'$
 8:         end if
 9:     end for
10:     Replace $L \leftarrow L'$, set $L' \leftarrow \emptyset$, and recompute $R \leftarrow \max_{w \in L} \|w\|$
11: until $L$ contains a shortest lattice vector
12: return $\operatorname{argmin}_{0 \neq v \in L} \|v\|$
```

Lemma 2.2 (Complexities of the optimized Nguyen–Vidick sieve)**.** *The Nguyen–Vidick sieve with the spherical locality-sensitive filters of Becker–Ducas–Gama–Laarhoven heuristically solves SVP in space* S *and time* T*, with*

$$S = (4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}, \qquad T = (3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}. \tag{2.3}$$

Micciancio and Voulgaris' GaussSieve. Micciancio and Voulgaris used a slightly different approach in their GaussSieve algorithm [MV10b]. This algorithm reduces the memory footprint by immediately *reducing* all pairs of lattice vectors that can be combined to form shorter lattice vectors. The algorithm uses a single list L, which is continuously kept in a state where for all $w_1, w_2 \in L$, $\|w_1 \pm w_2\| \geqslant \|w_1\|, \|w_2\|$. Each time a new vector $v \in \mathcal{L}$ is sampled, its norm is reduced with vectors in L by adding/subtracting vectors $w \in L$ which would lead to a shorter vector, and vectors in the list are also reduced with the vector v. After v can no longer be reduced with L, v is finally added to the list, guaranteeing that the pairwise reduction property is maintained. Modified list vectors are added to a stack to be reconsidered later. Algorithm 2.2 describes this procedure in pseudocode.

By immediately reducing all pairs of vectors, the GaussSieve achieves significantly better practical time and space complexities than the Nguyen–Vidick sieve. At the same time however, Nguyen and Vidick's (heuristic) proof technique does not apply to the GaussSieve, and there is no proven theoretical bound on the time complexity of the GaussSieve, even using heuristic assumptions. However, it is commonly believed that the Nguyen–Vidick sieve and the GaussSieve have the same heuristic asymptotic space and time complexities, i.e. using $2^{0.208d+o(d)}$ space and $2^{0.415d+o(d)}$ time without any further modifications.

2.2.3 – Nearest neighbor algorithms. Related to nearest neighbor searching, we recall the following problem definitions. These are all problems where preprocessing is essential, and the most general statements of these problems are given below.

Algorithm 2.2 The GaussSieve algorithm for finding shortest vectors [MV10b]

```
Require: A basis $B$ of a lattice $\mathcal{L}(B)$
Ensure: The algorithm outputs a shortest non-zero lattice vector
 1: Initialize an empty list $L$ and an empty stack $S$
 2: repeat
 3:     Get a vector $v$ from the stack (or sample a new one if $S = \emptyset$)
 4:     for each $w \in L$ do
 5:         if $\|v - w\| < \|v\|$ then
 6:             Replace $v \leftarrow v - w$
 7:         end if
 8:     end for
 9:     for each $w \in L$ do
10:         if $\|w - v\| < \|w\|$ then
11:             Replace $w \leftarrow w - v$
12:             Move $w$ from the list $L$ to the stack $S$ (unless $w = \mathbf{0}$)
13:         end if
14:     end for
15:     if $v \neq \mathbf{0}$ then
16:         Add $v$ to the list $L$
17:     end if
18: until $L$ contains a shortest lattice vector
19: return $\operatorname{argmin}_{\mathbf{0} \neq v \in L} \|v\|$
```

Definition 2.3 (Nearest Neighbor Searching – NNS). *Given a data set $L \subset \mathbb{R}^d$, preprocess this data in such a way that, when given a target vector $\mathbf{t} \in \mathbb{R}^d$ later, one can quickly find a vector $\mathbf{s} \in L$ such that $\|\mathbf{s} - \mathbf{t}\| = \min_{v \in L} \|v - \mathbf{t}\|$.*

Definition 2.4 (Approximate Nearest Neighbor Searching – NNS$_c$). *Let $L \subset \mathbb{R}^d$ and given an approximation factor $c \geqslant 1$, preprocess the data in such a way that when given a target vector $\mathbf{t} \in \mathbb{R}^d$ later, one can quickly find a vector $\mathbf{s} \in L$ such that $\|\mathbf{s}-\mathbf{t}\| \leqslant c \cdot \min_{v \in L} \|v-\mathbf{t}\|$.*

NNS is essentially equivalent to CVPP, except that (1) the data set in nearest neighbor searching is not assumed to be structured, and (2) the data set is assumed to be of finite cardinality $n < \infty$. Naive brute force algorithms for nearest neighbor searching take $O(n)$ time and $O(n)$ space without any preprocessing costs, and the literature commonly focuses on sublinear time algorithms, running in time $O(n^\rho)$ for $\rho < 1$, commonly with superlinear space and preprocessing costs.

A celebrated technique for finding near neighbors in high dimensions is locality-sensitive hashing (LSH). Here the idea is to construct many random partitions of the space, and index the data set L in hash tables with buckets corresponding to the regions induced by these partitions. Preprocessing then consists of constructing these hash tables, and a query $\mathbf{t}$ is answered by doing a lookup in each of the hash tables, and searching for a(n approximate) nearest neighbor in the hash buckets corresponding to $\mathbf{t}$. For a data set of size $|L| = n$, this commonly leads to a sublinear time complexity $O(n^\rho)$ $(\rho < 1)$ as long as an approximate solution suffices, or if the majority of data points lie significantly further from the target than the nearest point in the data set. LSH has also been used to speed up lattice sieving (e.g. [Laa15]).

Similar to locality-sensitive hash functions, locality-sensitive filters (LSF) divide the space into regions, with the added relaxation that these regions do not have to form a proper partition; regions may overlap, and part of the space may not even be covered by any region. This leads to improved results when n is exponential in d [BDGL16].

Below we restate the main result of [Laa16c] for our applications, where n is assumed to be exponential in d. The specific problem considered here is: given a data set L of points sampled uniformly at random from the unit sphere $\mathcal{S}^{d-1}$, and a random query $\mathbf{t} \in \mathcal{S}^{d-1}$, return a vector $\boldsymbol{w} \in L$ such that the angle between $\boldsymbol{w}$ and $\mathbf{t}$ is at most $\theta \in (0, \frac{\pi}{2})$. The following result further assumes that the list L contains exactly $n = (1/\sin\theta)^{d+o(d)}$ vectors. The following is a restatement of [Laa16c, Corollary 1].

Lemma 2.5 (Nearest neighbor costs for spherical data sets). *Let $\theta \in (0, \frac{1}{2}\pi)$, and let $u \in [\cos\theta, 1/\cos\theta]$. Let $L \subset \mathcal{S}^{d-1}$ be a list of $n = (1/\sin\theta)^{d+o(d)}$ vectors sampled uniformly at random from $\mathcal{S}^{d-1}$. Then, using spherical LSF with parameters $\alpha_q = u\cos\theta$ and $\alpha_u = \cos\theta$, one can preprocess L in time $n^{1+\rho_u+o(1)}$, using $n^{1+\rho_u+o(1)}$ space, and with high probability answer a random query $\mathbf{t} \in \mathcal{S}^{d-1}$ correctly in time $n^{\rho_q+o(1)}$, where:*

$$n^{\rho_q} = \left(\frac{\sin^2\theta\,(u\cos\theta + 1)}{u\cos\theta - \cos 2\theta}\right)^{d/2}, \quad n^{\rho_u} = \left(\frac{\sin^2\theta}{1 - \cot^2\theta\,(u^2 - 2u\cos\theta + 1)}\right)^{d/2}. \tag{2.4}$$

In the above lemma, the parameter $u \in [\cos\theta, 1/\cos\theta]$ controls the trade-off between the preprocessing/space complexities on the one hand, and the query time complexity on the other. The two extreme cases correspond to near-linear space and preprocessing with a slightly sublinear query time complexity (for $u = \cos\theta$), and very high space/preprocessing complexities with almost instant query responses (for $u = 1/\cos\theta$). The case $u = 1$ corresponds to $\rho_q = \rho_u$.

2.3 — The CVPP cost model

For analyzing the performance of CVPP algorithms, we split these methods into two phases: the preprocessing phase (whose input is only the lattice $\mathcal{L}$), and the query phase (where also the target vector $\mathbf{t}$ is known). We keep track of four costs of CVPP algorithms.

- **Preprocessing phase**: Preprocess the lattice $\mathcal{L}$ (without the target $\mathbf{t}$);
  - S_1: The memory used during the preprocessing phase;
  - T_1: The time used during the preprocessing phase;
- **Query phase**: Process the query $\mathbf{t}$ and output a vector $\mathbf{s} \in \mathcal{L}$ near $\mathbf{t}$;
  - S_2: The memory used during the query phase;
  - T_2: The time used during the query phase.

Intuitively the main goal of CVPP algorithms is to reduce the complexities of the query phase (S_2, T_2) compared to a non-preprocessed CVP algorithm. That way, a sufficiently large batch of CVP instances on the same lattice can be solved faster than with direct CVP approaches. However, in any practical application we need to perform the preprocessing at least once, and therefore CVPP algorithms with enormous preprocessing costs may be useless even if the query complexities are great. Also note that as we are interested in reducing the query complexity compared to solving CVP, and this usually comes at the cost of a higher preprocessing cost, we generally have $T_2 \leqslant T \leqslant T_1$, where T is the corresponding asymptotic time complexity for CVP.

Polynomial vs. exponential advice. Note that in the literature on CVPP, a common assumption is that the output of the preprocessing stage has size *polynomial* in the lattice dimension d [Mic01]. This is partly because with unlimited preprocessing power (time and space), heuristically the "post-processing" stage of CVPP can easily be made polynomial time. As an example, one could cover a sufficiently large ball around the origin with tiny cubes, and precompute/store the centers of these cubes, together with solutions to CVP with these centers as target vectors. Given a target vector for CVPP, one could then size-reduce with an LLL-reduced basis $\mathbf{B}$, identify the cube the vector is in, and assuming the net of cubes is sufficiently fine-grained, a solution to CVP for the center of this cube is then likely a solution to CVP for the target vector as well.

Since the costs of the preprocessing stage are usually disregarded when assessing the performance of a CVPP method, this would make CVPP (and CVPP_κ, BDDP_δ) altogether trivial. Throughout we are interested in the practicality of the "total package" of the CVPP algorithm, including the preprocessing. Taking these costs into account, the problem is no longer trivial even when allowing exponential-sized advice from the preprocessing stage. We explicitly do not make the assumption that the output of the preprocessing stage is of polynomial size.

2.4 — Voronoi cells

In this section we recall some definitions and results regarding the Voronoi cell of a lattice from [VB96, AEVZ02, SFS09, MV10a]. First, we give a formal definition of Voronoi cells below, which are essentially the enclosing regions of points closer to the origin than to any other lattice point.

Definition 2.6 (Voronoi cell of a lattice)**.** *The Voronoi cell of a lattice $\mathcal{L}$ is defined as the region $\mathcal{V} \subset \mathbb{R}^d$ such that $\mathbf{v} \in \mathcal{V}$ iff $\|\mathbf{v}\| \leqslant \|\mathbf{v} - \mathbf{x}\|$ for all $\mathbf{x} \in \mathcal{L}$. In other words:*

$$\mathcal{V} := \bigcap_{\mathbf{r} \in \mathcal{L}} \mathcal{H}(\mathbf{r}). \tag{2.5}$$

An important property of Voronoi cells, which immediately follows from the definition, is that the closest vector to a target vector $\mathbf{t} \in \mathbb{R}^d$ in a lattice $\mathcal{L}$ is the vector $\mathbf{s} \in \mathcal{L}$ if and only if $\mathbf{t} \in \mathbf{s} + \mathcal{V}$. In particular, the latter condition is equivalent to $\mathbf{t} - \mathbf{s} \in \mathcal{V}$, which indicates that if we can find a point $\mathbf{t}' \in (\mathbf{t} + \mathcal{L}) \cap \mathcal{V}$ (i.e. $\mathbf{t}' = \mathbf{t} - \mathbf{s}$), then we have found a solution to CVP for $\mathbf{t}$ as $\mathbf{s} = \mathbf{t} - \mathbf{t}'$.

In (2.5) above, we see that the Voronoi cell can be described in terms of an infinite number of half-spaces generated by the vectors in the lattice $\mathcal{L}$. In reality, the Voronoi cell of a lattice is a convex polytope with only a bounded number of facets, and its facets are closely related to what are commonly known as the *relevant vectors*, defined below.

Definition 2.7 (Relevant vectors)**.** *Given a lattice $\mathcal{L}$, a vector $\mathbf{r} \in \mathcal{L}$ is a* relevant vector *of the lattice $\mathcal{L}$ if and only if $\mathcal{V} \cap (\mathbf{r} + \mathcal{V})$ is an $(d-1)$-dimensional facet of $\mathcal{V}$. We denote the set of all relevant vectors by $\mathcal{R} \subseteq \mathcal{L}$.*

The relevant vectors of the lattice shape the boundary of $\mathcal{V}$, and the set $\mathcal{R}$ can be seen as a more compact way of representing/storing the Voronoi cell of a lattice, as $\mathcal{V}$ can be equivalently described by the following equation:

$$\mathcal{V} = \bigcap_{\mathbf{r} \in \mathcal{R}} \mathcal{H}(\mathbf{r}). \tag{2.6}$$

In other words, the Voronoi cell is also equal to the intersection of half-spaces generated only by the relevant vectors $\mathbf{r} \in \mathcal{R}$. The set $\mathcal{R}$ is by definition the minimal set $S \subseteq \mathcal{L}$ with the property that $\mathcal{V} = \bigcap_{\mathbf{r} \in S} \mathcal{H}(\mathbf{r})$ – other vectors do not contribute to the shape of the Voronoi cell, and removing any vector from $\mathcal{R}$ would result in a different, larger enclosed region.

To efficiently describe and store the Voronoi cell of a lattice, it is important to know that $\mathcal{R}$ is finite and cannot be too large. Fortunately the size of $\mathcal{R}$ can be bounded (in the worst-case) as follows; see e.g. [MV10a, Corollary 2.5] for a proof.

Lemma 2.8 (Number of relevant vectors). *For arbitrary lattices $\mathcal{L}$, the set $\mathcal{R}$ of relevant vectors satisfies $|\mathcal{R}| \leqslant 2^{d+1}$.*

As a result, a description of the Voronoi cell of a lattice can be stored in $2^{d+o(d)}$ memory, by storing all the relevant vectors. An example of a Voronoi cell for a two-dimensional lattice, as well as the related relevant vectors, is given in Section 2.6 in Figure 2.1a. On the negative side, note that a storage requirement of the order 2^d means it is infeasible to store the exact Voronoi cell of lattices in dimensions higher than e.g. 80. This in contrast to heuristic sieving algorithms, whose space requirement of the order $2^{0.21d}$ means these algorithms can still be used in higher dimensions as well.

2.5 — Voronoi cell algorithms

As stated above, a crucial property of Voronoi cells, highlighting their relevance for closest point searching, is that $\mathbf{s} \in \mathcal{L}$ is the closest point to a target vector $\mathbf{t} \in \mathbb{R}^d$ if and only if $\mathbf{t} - \mathbf{s} \in \mathcal{V}$. Since $\mathbf{t} - \mathbf{s} \in \mathbf{t} + \mathcal{L}$, a common approach of Voronoi cell algorithms for finding closest points to target vectors $\mathbf{t}$ is to start with $\mathbf{t}$ and gradually move along the coset $\mathbf{t} + \mathcal{L}$ towards the origin (by adding/subtracting lattice vectors to our current vector in the coset $\mathbf{t} + \mathcal{L}$). When no shorter vector in $\mathbf{t} + \mathcal{L}$ exists than our current guess $\mathbf{t}' \in \mathbf{t} + \mathcal{L}$, we know that $\mathbf{t}' \in \mathcal{V}$ and therefore $\mathbf{s} = \mathbf{t} - \mathbf{t}'$ is the closest lattice point to $\mathbf{t}$.

Building upon work of Sommer, Feder and Shalvi [SFS09], Micciancio and Voulgaris [MV10a] described algorithms for constructing the Voronoi cell of a lattice (or equivalently, the set of $2^{d+o(d)}$ relevant vectors of the lattice), and with proven time complexities at most $2^{2d+o(d)}$ this allowed them to solve SVP, CVP and CVPP. Bonifas and Dadush [BD15] later showed how to improve the time complexity for CVPP to only $2^{d+o(d)}$, by bounding the number of iterations in the post-processing stage to $\mathrm{poly}(d)$ rather than $2^{d+o(d)}$.

An important technique for finding closest vectors, using the list of relevant vectors to peform the aforementioned procedure of finding a point $\mathbf{t}' \in \mathbf{t} + \mathcal{L}$, is the iterative slicer of Sommer–Feder–Shalvi [SFS09] given in Algorithm 2.3. Given a target vector $\mathbf{t}$ and the Voronoi cell of the lattice as input, within a finite number of steps [SFS09, Theorem 1] this algorithm terminates and finds the closest vector to any target $\mathbf{t} \in \mathbb{R}$. Micciancio–Voulgaris later showed that by selecting relevant vectors for reduction in a specific order, the number of iterations can be bounded by $2^{d+o(d)}$ [MV10a, Lemma 3.2].

Algorithm 2.3 The iterative slicer for finding closest vectors [SFS09]

```
Require: The relevant vectors $\mathcal{R} \subset \mathcal{L}$ and a target $\mathbf{t} \in \mathbb{R}^d$
Ensure: The algorithm outputs a closest lattice vector $\mathbf{s} \in \mathcal{L}$ to $\mathbf{t}$
1: Initialize $\mathbf{t}' \leftarrow \mathbf{t}$
2: for each $\mathbf{r} \in \mathcal{R}$ do
3:     if $\|\mathbf{t}' - \mathbf{r}\| < \|\mathbf{t}'\|$ then
4:         Replace $\mathbf{t}' \leftarrow \mathbf{t}' - \mathbf{r}$ and restart the for-loop
5:     end if
6: end for
7: return $\mathbf{s} = \mathbf{t} - \mathbf{t}'$
```

By similar techniques as in heuristic lattice sieving (or as in [BD15]), one can bound the number of iterations of this slicer until termination. Given a target $\mathbf{t}$, one can use e.g. Babai rounding on an LLL-reduced basis of the lattice to get an initial guess $\mathbf{t}' \in \mathbf{t} + \mathcal{L}$ satisfying $\|\mathbf{t}'\| \leqslant 2^{O(d)} \min_{\mathbf{s} \in \mathbf{t}+\mathcal{L}} \|\mathbf{s}\|$. Then, by only performing reductions whenever $\|\mathbf{t}' \pm \mathbf{r}\| < \gamma \|\mathbf{t}'\|$ for some geometric factor $\gamma = 1 - 1/d^k$ for certain $k > 1$, one can ensure that the number of iterations is polynomially bounded by $\log_{1/\gamma} \|\mathbf{t}'\| = O(d^{1+k})$. At the same time, due to this geometric factor γ, after the algorithm terminates we might only have $\mathbf{t}' \in (1/\gamma)\mathcal{V}$ instead of $\mathbf{t}' \in \mathcal{V}$. Since $\mathrm{Vol}(\mathcal{V}/\gamma) = \mathrm{Vol}(\mathcal{V})/\gamma^d$, we therefore expect this algorithm to succeed with probability proportional to $\gamma^d = 1 - O(d^{1-k}) = 1 - o(1)$ over the randomness of $\mathbf{t}$. As k increases, the (polynomial) number of iterations increases, while heuristically the success probability becomes overwhelming.

Bonifas and Dadush [BD15] described a different method to bound the number of iterations to $\mathrm{poly}(d)$, by carefully choosing which relevant vectors to use for reduction in each step. Although there is no formal proof that other approaches allow for solving exact CVP as well, in the remainder of this chapter we will assume (heuristically) that the number of iterations of this slicer (until termination) is only $\mathrm{poly}(d)$, for random lattices and average-case target vectors.

2.6 — Approximate Voronoi cells

In this section we revisit the preprocessing approach to CVP of [Laa16b], as well as the trend of speeding up these algorithms using nearest neighbor searching. These results can be viewed as a first step towards a practical, heuristic alternative to the Voronoi cell algorithm of Micciancio–Voulgaris [MW15], where instead of constructing the exact Voronoi cell, the preprocessing computes an approximation of it, requiring less time and space to compute and store.

First, our preprocessing step consists of computing a list L of most lattice vectors below a given norm[1]. This preprocessing can be done using e.g. enumeration or sieving. The preprocessed data can best be understood as representing an *approximate* Voronoi

[1]Heuristically, finding a large fraction of all lattice vectors below a given norm will suffice – one does not necessarily need to run a deterministic preprocessing algorithm to ensure all short lattice vectors are found.

cell of the lattice, where the size of L determines how good the approximation is (see Figure 2.1 for an example). Using this approximate Voronoi cell, a CVP instance can then be solved by applying the iterative slicing procedure of [SFS09], with nearest neighbor optimizations to reduce the search costs.

Below we start with a formal definition of approximate Voronoi cells, where as before we write $\mathcal{H}(\mathbf{x})$ for half-spaces whose boundaries are orthogonal to $\mathbf{x}$ and pass through $\frac{1}{2}\mathbf{x}$.

Definition 2.9 (Approximate Voronoi cells). *For a lattice $\mathcal{L}$ and a list $L \subseteq \mathcal{L}$, the approximate Voronoi cell generated by L is defined as:*

$$\mathcal{V}_L := \bigcap_{\mathbf{r} \in L} \mathcal{H}(\mathbf{r}). \tag{2.7}$$

Note that $\mathcal{V} \subseteq \mathcal{V}_L$ for any $L \subseteq \mathcal{L}$, and $\mathcal{V} = \mathcal{V}_L$ if and only if $\mathcal{R} \subseteq L$ [SFS09, Lemma 5]. Similarly, $\mathcal{R}$ is the smallest set $L \subseteq \mathcal{L}$ with the property that $\mathcal{V}_L = \mathcal{V}$. To quantify the 'quality' of an approximate Voronoi cell $\mathcal{V}_L$ (or a list L), recall that the volume of the exact Voronoi cell $\mathcal{V}$ is equal to the volume of the lattice: $\mathrm{Vol}(\mathcal{V}) = \det(\mathcal{L})$. If $\mathcal{R}$ is not contained in L, then $\mathcal{V}_L$ will have a strictly larger volume, and the following quantity can therefore serve as a guideline as to how well an approximate Voronoi cell $\mathcal{V}_L$ approximates $\mathcal{V}$.

Definition 2.10 (Approximation factor). *Given a lattice $\mathcal{L}$ and a list $L \subseteq \mathcal{L}$, we define the approximation factor A_L for the cell $\mathcal{V}_L$ generated by L as:*

$$A_L := \frac{\mathrm{Vol}(\mathcal{V}_L)}{\mathrm{Vol}(\mathcal{V})}. \tag{2.8}$$

Clearly $A_L \geqslant 1$ with equality iff $\mathcal{R} \subseteq L$. If L is very small, A_L may be infinite, but as long as L contains a basis of the lattice one has $A_L < \infty$ [VB96]. For arbitrary lists L, L' with $L \subseteq L' \subseteq \mathcal{L}$ we have $A_{L'} \leqslant A_L$, i.e. if we add vectors to L to form L', the approximation factor either stays the same or decreases.

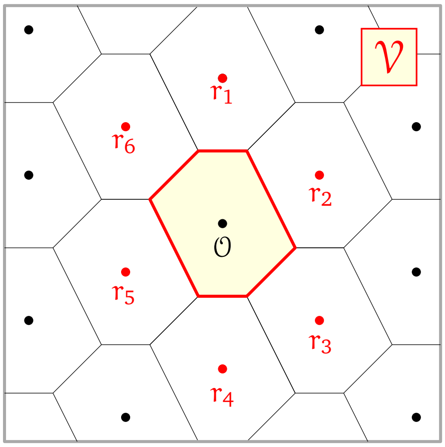
Compared to [Laa16b], the main improvement of this approach lies in generalizing how similar the approximate Voronoi cell $\mathcal{V}_L$ (generated by the list L) needs to be to the exact Voronoi cell of the lattice, $\mathcal{V}$. We distinguish two cases below. As sketched in Figure 2.1, a worse approximation leads to a larger approximate Voronoi cell, so $\mathrm{vol}(\mathcal{V}_L) \geqslant \mathrm{vol}(\mathcal{V})$ with equality iff $\mathcal{V} = \mathcal{V}_L$.

2.6.1 – Good approximations. The main result of [Laa16b] for solving CVPP can be summarized in terms of approximate Voronoi cells by the following lemma, stating how big L must heuristically be to obtain approximation factors close to 1.

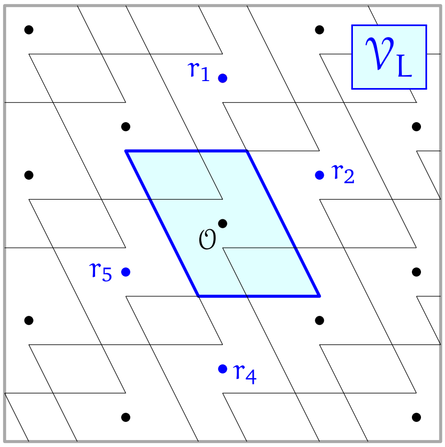
Lemma 2.11 (Good approximations). *Let L consist of the $\alpha^{d+o(d)}$ shortest vectors of a lattice $\mathcal{L}$, with $\alpha \geqslant \sqrt{2} + o(1)$. Then heuristically,*

$$A_L = 1 + o(1). \tag{2.9}$$

In other words, if L contains the $2^{d/2+o(d)}$ shortest lattice vectors of a random lattice $\mathcal{L}$, we expect $\mathcal{V}_L$ to approximate the exact Voronoi cell $\mathcal{V}$ very well. This in contrast with the best proven worst-case bounds, which suggest that up to $2^{d+o(d)}$ vectors are needed to accurately represent the Voronoi cell of a lattice.



(a) A tiling of $\mathbb{R}^2$ with exact Voronoi cells $\mathcal{V}$ of a lattice $\mathcal{L}$ (red/black points), generated by the set $\mathcal{R} = \{r_1, \dots, r_6\}$ of all *relevant vectors* of $\mathcal{L}$. Here $\mathrm{vol}(\mathcal{V}) = \det(\mathcal{L})$.



(b) An overlapping tiling of $\mathbb{R}^2$ with approximate Voronoi cells $\mathcal{V}_L$ of the same lattice $\mathcal{L}$, generated by a subset of the relevant vectors, $L = \{r_1, r_2, r_4, r_5\} \subset \mathcal{R}$.

Figure 2.1: Exact and approximate Voronoi cells of the same two-dimensional lattice $\mathcal{L}$.
For the **exact** Voronoi cell $\mathcal{V}$ (Figure 2.1a), the cells around the lattice points form a tiling of $\mathbb{R}^2$, covering each point in space exactly once. Given that a point $\mathbf{t}$ lies in the Voronoi cell around $\mathbf{s} \in \mathcal{L}$, we know that $\mathbf{s}$ is the closest lattice point to $\mathbf{t}$.
For the **approximate** Voronoi cell $\mathcal{V}_L$ (Figure 2.1b), the cells around the lattice points overlap, and cover a non-empty fraction of the space by multiple cells. Given that a vector $\mathbf{t}$ lies in an approximate Voronoi cell around a lattice point $\mathbf{s}$, we further do not have the definite guarantee that $\mathbf{s}$ is the closest lattice point to $\mathbf{t}$.

Algorithm 2.4 The heuristic slicer for finding closest vectors [Laa16b]

Require: A list $L \subset \mathcal{L}$ of the $2^{d/2+o(d)}$ shortest vectors of $\mathcal{L}$, and a target $\mathbf{t} \in \mathbb{R}^d$
Ensure: The algorithm outputs a closest lattice vector $\mathbf{s} \in \mathcal{L}$ to $\mathbf{t}$

```
1: Initialize $\mathbf{t}' \leftarrow \mathbf{t}$
2: for each $\mathbf{r} \in L$ do                                ▷ NNS speedups can be used here
3:     if $\|\mathbf{t}' - \mathbf{r}\| < \|\mathbf{t}'\|$ then
4:         Replace $\mathbf{t}' \leftarrow \mathbf{t}' - \mathbf{r}$ and restart the for-loop
5:     end if
6: end for
7: return $\mathbf{s} \leftarrow \mathbf{t} - \mathbf{t}'$
```

Heuristically, Lemma 2.11 implies that if we use Voronoi cell algorithms for CVP(P), using only the $2^{d/2+o(d)}$ shortest lattice vectors as our approximate list of relevant vectors (instead of all $2^{d+o(d)}$ relevant vectors), the algorithm will still succeed with high probability in returning the actual closest vector to random target vectors. The resulting heuristic slicer, which serves as the algorithm for the query phase of CVPP in [Laa16b], is given in Algorithm 2.4.

Assuming the list L contains the $2^{d/2+o(d)}$ shortest vectors of $\mathcal{L}$, it returns the closest vector with high probability. Naively, this algorithm has query time and space complexities of $2^{d/2+o(d)}$, but with nearest neighbor search techniques the search for relevant vectors that can reduce $\mathbf{t}'$ can be sped up significantly.

2.6.2 – Arbitrary approximations. To obtain improved results compared to [Laa16b], we will use the following generalization of Lemma 2.11, providing a heuristic upper bound on the approximation factor A_L for lists L containing fewer than $2^{d/2+o(d)}$ lattice vectors. [2]

Lemma 2.12 (Arbitrary approximations). *Let L consist of the $\alpha^{d+o(d)}$ shortest vectors of a lattice $\mathcal{L}$, with $\alpha \in (1.03396, \sqrt{2})$. Then heuristically,*

$$A_L \leqslant \left(\frac{16\alpha^4\left(\alpha^2-1\right)}{-9\alpha^8+64\alpha^6-104\alpha^4+64\alpha^2-16}\right)^{d/2+o(d)}. \tag{2.10}$$

Above, $1.03396\ldots$ is a root of the polynomial in the denominator. Note that as $\alpha \to \sqrt{2}$, the ratio approaches 1, and Lemma 2.12 therefore is a proper generalization of Lemma 2.11. For $\alpha \downarrow 1.03396\ldots$, the ratio tends to ∞, suggesting that for very small lists our analysis does not provide a proper, meaningful upper bound on the approximation factor – there is no reason to believe that $A_L = \infty$ for exponentially large preprocessed lists.

For random target vectors, we heuristically expect the probability of success of finding a closest vector to this target with a preprocessed list L to be approximately $p_L = 1/A_L$ – assuming a reduced vector returned by the slicer lies uniformly in $\mathcal{V}_L$, the probability that it also lies in $\mathcal{V}$ is proportional to $\mathrm{Vol}(\mathcal{V})/\mathrm{Vol}(\mathcal{V}_L)$. With the above result in mind, we can thus generalize the previous heuristic slicer to construct a CVPP algorithm which works with even smaller lists L, but has an exponentially small success probability in the dimension d. This is somewhat unsatisfactory, as an algorithm succeeding with exponentially small success probability is unlikely to be useful in any application. However, similar to e.g. extreme pruning in enumeration [GNR10], as long as the gain in the time (and space) complexity is more than the loss in the success probability, such an algorithm may well turn out to be useful if we can somehow randomize our reduction algorithm.

2.7 — Randomized slicing

To instantiate Lemma 2.12 with an actual CVPP algorithm succeeding with high probability, ideally we need to be able to rerandomize problem instances such that, if an algorithm succeeds with small probability p in time T, we can repeat the algorithm approximately $1/p$ times to obtain an algorithm succeeding with constant probability in T/p time. The (heuristic) iterative slicer mostly works deterministically[3], so if an initialized data structure and problem instance fail, running the same slicing algorithm on the same target would likely result in failure again.

To rerandomize problem instances, we will use the following procedure: instead of starting with a single vector $\mathbf{t}' \leftarrow \mathbf{t}$ and attempting to reduce it to a vector $\mathbf{t}'' \in \mathcal{V}$, we use several vectors of the form $\mathbf{t}' \sim D_{\mathbf{t}+\mathcal{L},s}$ sampled from a discrete Gaussian over the

[2]The heuristic proof/derivation of this result from [DLdW19] is not tight, as it was later shown by Ducas Laarhoven and van Woerden in [DLvW20].

[3]Observe that there is some room for randomization within the slicing algorithm itself: if an intermediate vector $\mathbf{t}'$ can be reduced with two vectors $\mathbf{v}_1, \mathbf{v}_2 \in L$ to form a shorter vector, one could randomly choose either option for potentially different outcomes of the slicer.

coset $\mathbf{t} + \mathcal{L}$ with a well-chosen parameter s. This parameter s needs to be chosen small enough so that sampling can be done in polynomial time in the lattice dimension d, and large enough so that the sampled vectors are not too long, and so that the reductions of $\mathbf{t}'$ to short vectors $\mathbf{t}''$ do not take too long. To analyze our CVPP method using this sampling procedure, we propose the following heuristic assumption, essentially stating that this rerandomization procedure works perfectly.

Heuristic 2.13 (Randomized slicing). *For* $L \subset \mathcal{L}$ *and large* s,

$$\Pr_{\mathbf{t}' \sim D_{\mathbf{t}+\mathcal{L},s}} \left[\mathrm{Slice}_L(\mathbf{t}') \in \mathcal{V} \right] \approx \frac{1}{\mathcal{A}_L}. \tag{2.11}$$

For intuition behind this statement, recall that with a preprocessed list L, a vector $\mathbf{t}'$ will ultimately be reduced by the slicer to a vector $\mathbf{t}'' = \mathrm{Slice}(\mathbf{t}') \in \mathcal{V}_L$. If $\mathbf{t}''$ now also lies in $\mathcal{V}$, which we may heuristically model as a sphere of radius $\lambda_1(\mathcal{L})$, then $\mathbf{0}$ is the closest lattice vector to $\mathbf{t}'' \in \mathbf{t} + \mathcal{L}$, and so $\mathbf{s} = \mathbf{t} - \mathbf{t}''$ is indeed the closest lattice vector to $\mathbf{t}$. However, since $\mathcal{V}_L$ is potentially much larger than $\mathcal{V}$, this may only occur with small probability, and heuristically we essentially assume that $\Pr_{\mathbf{t}'' \leftarrow \mathrm{Slice}(\mathbf{t}')}[\mathbf{t}'' \in \mathcal{V} \mid \mathbf{t}'' \in \mathcal{V}_L] = 1/\mathcal{A}_L$.

Note that the probability on the left hand side of (2.11) is for arbitrary, fixed target vectors $\mathbf{t}$, and the randomness is purely over the Gaussian sampling of $\mathbf{t}' \sim D_{\mathbf{t}+\mathcal{L},s}$ – we heuristically assume/expect that we can effectively apply this rerandomization procedure to *any* target vector, rather than e.g. always failing for certain targets and succeeding for other targets. Experiments presented in [DLdW19] indeed suggest that this assumption is justified.

Assuming the above heuristic assumption holds, an algorithm for CVPP follows, by repeating the slicing algorithm on randomly sampled vectors $\mathbf{t}' \sim D_{\mathbf{t}+\mathcal{L},s}$. The randomized heuristic slicer that we obtain is given in Algorithm 2.5.

Algorithm 2.5 The randomized heuristic slicer for finding closest vectors

```
Require: A list $L \subset \mathcal{L}$ and a target $\mathbf{t} \in \mathbb{R}^d$
Ensure: The algorithm outputs a closest lattice vector $\mathbf{s} \in \mathcal{L}$ to $\mathbf{t}$
 1: $\mathbf{s} \leftarrow \mathbf{0}$                                ▷ Initial guess $\mathbf{s}$ for closest vector to $\mathbf{t}$
 2: repeat
 3:     Sample $\mathbf{t}' \sim D_{\mathbf{t}+\mathcal{L},s}$         ▷ Randomly shift $\mathbf{t}$ by a vector $\mathbf{v} \in \mathcal{L}$
 4:     for each $\mathbf{r} \in L$ do
 5:         if $\|\mathbf{t}' - \mathbf{r}\| < \|\mathbf{t}'\|$ then          ▷ New shorter vector $\mathbf{t}' \in \mathbf{t} + \mathcal{L}$
 6:             Replace $\mathbf{t}' \leftarrow \mathbf{t}' - \mathbf{r}$ and restart the for-loop
 7:         end if
 8:     end for
 9:     if $\|\mathbf{t}'\| < \|\mathbf{t} - \mathbf{s}\|$ then
10:         $\mathbf{s} \leftarrow \mathbf{t} - \mathbf{t}'$                      ▷ New lattice vector $\mathbf{s}$ closer to $\mathbf{t}$
11:     end if
12: until $\mathbf{s}$ is a closest lattice vector to $\mathbf{t}$
13: return $\mathbf{s}$
```

For randomized slicing, the costs of the algorithm are mostly the same in terms of α as for a single run of the slicer, except that the (expected) time complexity T_2 for the query phase is multiplied by a factor $1/p$, to account for the expected number of trials necessary to find a closest vector. On the positive side, this means that we do not need to fix $\alpha = \sqrt{2}$ in advance, and can obtain significantly better space complexities, as well as better space–time trade-offs.

2.8 — Preprocessing costs

For the preprocessing phase, we need to generate a list of the $\alpha^{d+o(d)}$ lattice vectors of norm at most $\alpha \cdot \lambda_1(\mathcal{L})$, and store it in a nearest neighbor data structure to allow for fast searching. This preprocessing step can be done using different methods. In moderate dimensions, the fastest way may be to use lattice enumeration but in high dimensions (as well asymptotically) heuristic lattice sieving methods will lead to the best complexities. As we are mostly interested in obtaining the best asymptotic complexities here, let us consider the preprocessing costs for a sieving-based preprocessing stage.

Recall that with standard heuristic sieving methods, we reduce pairs of lattice vectors if their angle is at most $\theta = \frac{\pi}{3}$, resulting in a list of size $(\sin\theta)^{-d+o(d)}$. To generate a list of the $\alpha^{d+o(d)}$ shortest lattice vectors of a lattice $\mathcal{L}$ with the GaussSieve, rather than the $(4/3)^{d/2+o(d)}$ lattice vectors one would normally get, we relax the reduction step in sieving: we reduce a list vector $\boldsymbol{v}$ with another list vector $\boldsymbol{w}$ if and only if their pairwise angle is less than $\theta = \arcsin(1/\alpha)$, which for vectors $\boldsymbol{v}, \boldsymbol{w}$ of similar norm corresponds to the following condition:

$$\|\boldsymbol{v} - \boldsymbol{w}\|^2 < 2(1 - \cos\theta) \cdot \|\boldsymbol{v}\|^2 = \left(2 - \frac{2}{\alpha}\sqrt{\alpha^2 - 1}\right) \cdot \|\boldsymbol{v}\|^2. \tag{2.12}$$

This leads to the modified GaussSieve described in Algorithm 2.6. Note that for $\alpha = \sqrt{2}$, the reduction criterion becomes $\|\boldsymbol{v} - \boldsymbol{w}\| < \sqrt{2 - \sqrt{2}} \cdot \|\boldsymbol{v}\|$.

Intuitively, Algorithm 2.6 could be interpreted as a relaxed version of standard heuristic sieving approaches. In standard sieving, pairwise reductions are *always* performed, even if they lead to minor progress in reducing the norms of the vectors. This turns out to lead to the smallest list sizes. By only reducing vectors when significant progress is made in reducing their norms, reductions occur less frequently, leading to larger list sizes before real progress is made. By not always doing reductions in Algorithm 2.6, it takes longer to complete this preprocessing step, but a longer list of lattice vectors is returned. Moreover, by only searching for vectors with very small angles, the speed-ups obtained from applying nearest neighbor techniques become bigger as well.

Note that in Algorithm 2.6, as well as other lattice sieving algorithms in Section 2.2, the stopping criterion is stated as continuing until the list contains a shortest vector. In the literature on lattice sieving, many different stopping criterions have been considered, often involving a bound on the number of “collisions” to the all-zero vector. An alternative stopping criterion here might also be to continue until, say, at least 90% of the expected number of lattice vectors below a certain norm $\alpha \cdot \lambda_1(\mathcal{L})$ have been encountered by the sieve. In reality, the precise termination condition is somewhat irrelevant – at some point during the run, the list will be a very good quality and contain most short vectors, and continuing a bit longer only means that slightly more time is spent on the preprocessing phase, and slightly more of the shortest lattice vectors will be in the preprocessed list for the query phase.

Algorithm 2.6 The GaussSieve-based preprocessing phase for solving CVPP

```
Require: A basis $B$ of a lattice $\mathcal{L}(B)$, a parameter $\alpha \geqslant \sqrt{4/3}$
Ensure: The output list $L \subset \mathcal{L}$ contains $\alpha^{d+o(d)}$ vectors of norm at most $\alpha \cdot \lambda_1(\mathcal{L})$
 1: Initialize an empty list $L$ and an empty stack $S$
 2: repeat
 3:     Get a vector $\boldsymbol{v}$ from the stack (or sample a new one if $S = \emptyset$)
 4:     for each $\boldsymbol{w} \in L$ do                          ▷ NNS can be used to speed this up
 5:         if $\|\boldsymbol{v} - \boldsymbol{w}\|^2 < (2 - \frac{2}{\alpha}\sqrt{\alpha^2 - 1}) \cdot \|\boldsymbol{v}\|^2$ then
 6:             Replace $\boldsymbol{v} \leftarrow \boldsymbol{v} - \boldsymbol{w}$
 7:         end if
 8:     end for
 9:     for each $\boldsymbol{w} \in L$ do
10:         if $\|\boldsymbol{w} - \boldsymbol{v}\|^2 < (2 - \frac{2}{\alpha}\sqrt{\alpha^2 - 1}) \cdot \|\boldsymbol{w}\|^2$ then
11:             Replace $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{v}$
12:             Move $\boldsymbol{w}$ from the list $L$ to the stack $S$ (unless $\boldsymbol{w} = \boldsymbol{0}$)
13:         end if
14:     end for
15:     if $\boldsymbol{v} \neq \boldsymbol{0}$ then
16:         Add $\boldsymbol{v}$ to the list $L$
17:     end if
18: until $\boldsymbol{v}$ is a shortest vector
19: return $L$
```

The following lemma summarizes the preprocessing costs obtained by using the optimized nearest neighbor techniques from Lemma 2.5.

Lemma 2.14 (Preprocessing complexities). *Let $\alpha \in (1, \infty)$ and suppose that $u \in (\sqrt{\frac{\alpha^2-1}{\alpha^2}}, \sqrt{\frac{\alpha^2}{\alpha^2-1}})$. With Algorithm 2.6, we can heuristically generate a list of the $\alpha^{d+o(d)}$ shortest vectors in a lattice $\mathcal{L}$ with the following space and time complexities S_1 and T_1.*

$$S_1 = \max\left\{S_2, \left(\frac{4}{3}\right)^{d/2+o(d)}\right\}, \tag{2.13}$$

$$T_1 = \max\left\{S_2, \left(\frac{3}{2}\right)^{d/2+o(d)}\right\}, \tag{2.14}$$

$$S_2 = \left(\frac{\alpha}{\alpha - (\alpha^2 - 1)(\alpha u^2 - 2u\sqrt{\alpha^2 - 1} + \alpha)}\right)^{d/2+o(d)}. \tag{2.15}$$

Moreover, at the end of the preprocessing step, we have a data structure of size S_2 which can answer CVP queries in time T_2 as follows:

$$T_2 = \left(\frac{\alpha + u\sqrt{\alpha^2 - 1}}{-\alpha^3 + \alpha^2 u\sqrt{\alpha^2 - 1} + 2\alpha}\right)^{d/2+o(d)}. \tag{2.16}$$

Observe that reductions between vectors only make sense if vectors get shorter; if $\boldsymbol{v}$ and $\boldsymbol{w}$ have similar norms, then one cannot reduce $\boldsymbol{v}$ with $\boldsymbol{w}$ if their pairwise angle is larger than $\frac{\pi}{3}$. To generate a list with $\alpha^{d+o(d)}$ short vectors with $\alpha < \sqrt{4/3}$, one can just run the algorithm for $\alpha = \sqrt{4/3}$ (corresponding to the regular GaussSieve), and afterwards discard lattice vectors which are too long. Alternatively, if one is interested in minimizing the memory complexity, for small values α one could consider using tuple lattice sieving approaches discussed in [BLS16, HK17, HKL18]. We restrict our attention to sieving using pairwise reductions, and we leave a complexity analysis based on tuple reductions to future work.

Note also that S_1 and T_1 in Lemma 2.14 are lower bounded by the costs for solving SVP, which based on the current best space and time complexities for (pairwise) sieving are $(4/3)^{d/2+o(d)}$ and $(3/2)^{d/2+o(d)}$ respectively [BDGL16]. Using tuple sieving [BLS16, HK17, HKL18], it is possible to eliminate this lower bound on S_1, at the cost of worse preprocessing time complexities – however this further generalization of the complexities goes beyond the scope of this chapter.

2.9 — CVPP complexities

With the previous results and techniques in place, we are now ready to state a summarising result. For the exact closest vector problem with preprocessing, the improved complexities over [Laa16b] mainly come from the aforementioned randomizations. To illustrate this with a simple example, suppose we run an optimized LDSieve [BDGL16], ultimately resulting in a list of $(4/3)^{d/2+o(d)}$ of the shortest vectors in the lattice, indexed in a nearest neighbor data structure of size $(3/2)^{d/2+o(d)}$. Asymptotically, using this list as our approximate Voronoi cell, the iterative slicer succeeds with probability $p = (13/16)^{d/2+o(d)}$ (by (2.10) and (2.11) for $\alpha = \sqrt{4/3}$), while processing a query with this data structure takes time $(9/8)^{d/2+o(d)}$ (by (2.16) for $\alpha = \sqrt{4/3}$, $u = 1$). By repeating a query $1/p$ times with rerandomizations of the same CVP instance, we obtain the following heuristic complexities for CVPP.

Proposition 2.15 (Standard sieve preprocessing)**.** *Using the output of the LDSieve [BDGL16] as the preprocessed list and encompassing data structure, we can heuristically solve CVPP with the following query space and time complexities:*

$$S = (3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}, \qquad T = (18/13)^{d/2+o(d)} \approx 2^{0.235d+o(d)}.$$

If we use a more general analysis of the approximate Voronoi cell approach, varying over both the nearest neighbor parameters and the size of the preprocessed list, we can obtain even better query complexities. For a memory complexity of $(3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$, we can achieve a query time complexity of approximately $2^{0.220d+o(d)}$ by using a shorter list of lattice vectors, and a more memory-intensive parameter setting for the nearest neighbor data structure. The following main result summarizes all the asymptotic time–space trade-offs we can obtain for heuristically solving CVPP in the average case.

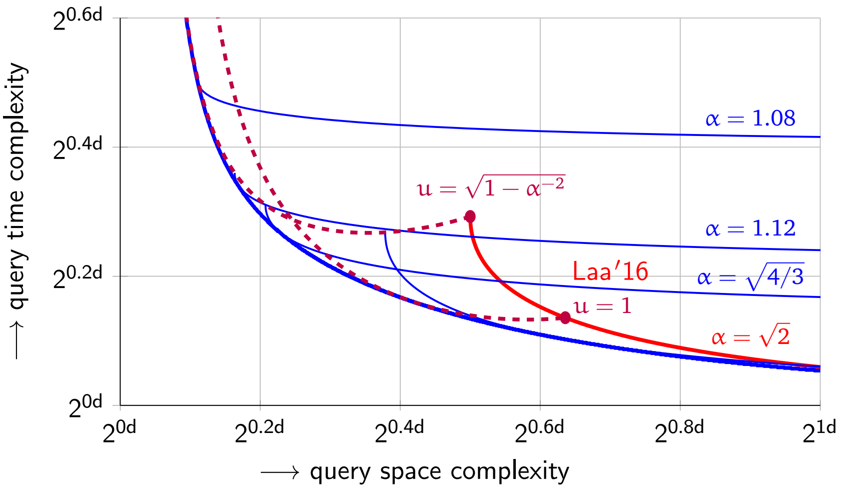


Figure 2.2: Complexities for randomized slicing. The light blue curves correspond to different values α and different success probabilities p_α. The right red curve corresponds to $\alpha = \sqrt{2}$ and $p_\alpha \approx 1$, i.e., not using randomized slicing as in Section 2.7. Dashed purple curves correspond to fixing the nearest neighbor parameter u and varying α. No single curve lies below all others, and the minimum over all curves is depicted by the bottom blue curve.

Theorem 2.16 (Optimized CVPP complexities). *Let $\alpha \in (1.03396, \sqrt{2})$ and $u \in (\sqrt{\frac{\alpha^2-1}{\alpha^2}}, \sqrt{\frac{\alpha^2}{\alpha^2-1}})$. With approximate Voronoi cells we can heuristically solve CVPP with preprocessing space and time S_1 and T_1, and query space and time S_2 and T_2, where:*

$$S_1 = \max\left\{S_2, \left(\frac{4}{3}\right)^{d/2+o(d)}\right\}, \qquad T_1 = \max\left\{S_2, \left(\frac{3}{2}\right)^{d/2+o(d)}\right\}, \tag{2.17}$$

$$S_2 = \left(\frac{\alpha}{\alpha - (\alpha^2-1)(\alpha u^2 - 2u\sqrt{\alpha^2-1}+\alpha)}\right)^{d/2+o(d)}, \tag{2.18}$$

$$T_2 \leqslant \left(\frac{16\alpha^4\left(\alpha^2-1\right)}{-9\alpha^8+64\alpha^6-104\alpha^4+64\alpha^2-16} \cdot \frac{\alpha + u\sqrt{\alpha^2-1}}{-\alpha^3+\alpha^2 u\sqrt{\alpha^2-1}+2\alpha}\right)^{d/2+o(d)}. \tag{2.19}$$

The best query complexities (S_2, T_2) together form the thick blue curve in Figure 2.2.

For $\alpha = \sqrt{2}$, Theorem 2.16 leads to the exact same complexities as without rerandomizations as in [Laa16b] depicted by the red curve in Figure 2.2. The time–space curve $\mathcal{C}_\alpha$ corresponding to $\alpha = \sqrt{4/3}$, as well as a few other values α, are shown in Figure 2.2. By taking the minimum over all these curves $\{\mathcal{C}_\alpha\}_{\alpha \in (1.03396, \sqrt{2})}$, where curves are defined by varying the parameter $u \in (\sqrt{(\alpha^2-1)/\alpha^2}, \sqrt{\alpha^2/(\alpha^2-1)})$, we obtain the thick blue curve in Figure 2.2. There seems to be no simple expression for this curve; for a particular choice of the space complexity, the best query time complexity T_2 can be found by considering all different α, and for each α computing the value u such that the space complexity is as desired, and taking the minimum over all these values. Note that due to the condition $\alpha > 1.03396$, the curve terminates on the left side at a minimum space complexity of $1.03396^{d+o(d)} \approx 2^{0.0482d+o(d)}$; with this method we cannot obtain a space complexity $S_2 = 2^{o(d)}$ for exact CVPP.

2.9.1 – Concrete complexities. Although Theorem 2.16 and Figure 2.2 illustrate how well we expect these methods to scale in high dimensions d, it should be stressed that Theorem 2.16 is a purely asymptotic result, with potentially large order terms hidden by the $o(d)$ in the exponents for the time and space complexities. To obtain security estimates for real-world applications, and to assess how fast this algorithm actually solves problems appearing in the cryptanalysis of lattice-based cryptosystems, it therefore remains necessary to perform extensive experiments, and to cautiously try to extrapolate from these results what the real attack costs might be for high dimensions d, necessary to attack actual instantiations of cryptosystems. In [DLdW19] some preliminary experiments were performed to test the practicality of this approach, but further work is still necessary to assess the impact of these results on the concrete hardness of CVPP. We briefly mention some of the experiments from [DLdW19] in the next section.

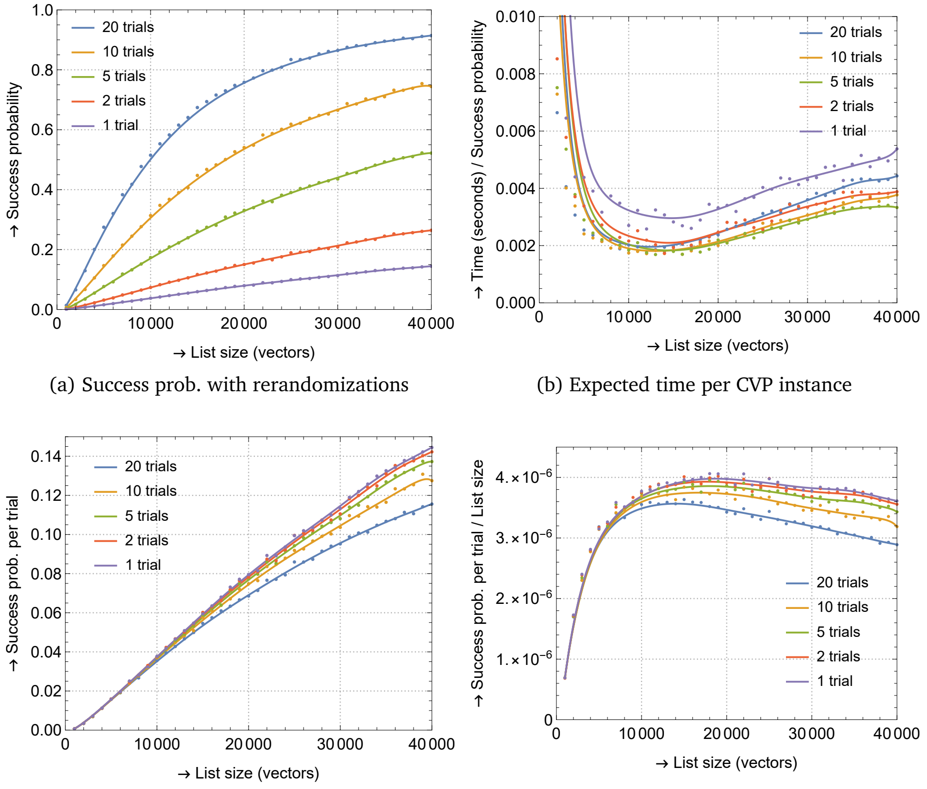
2.10 — Experimental results

Besides the theoretical results mentioned in Section 2.9, with improved heuristic time and space complexities compared to [Laa16b], [DLdW19] also implemented a (sieving-based) CVPP solver using approximate Voronoi cells. For the preprocessing it used a slight modification of a lattice sieve, returning more vectors than a standard sieve, allowing to vary the list size in the experiments. The implementations served two purposes: validating the additional heuristic assumption made, and to see how well the algorithm performs in practice.

To obtain the aforementioned improved asymptotic complexities for solving CVPP, required a new heuristic assumption, stating that if the iterative slicer succeeds with some probability p on a CVP instance $\mathbf{t}$, then we can repeat $1/p$ times with perturbations $\mathbf{t}' \sim D_{\mathbf{t}+\mathcal{L},s}$ to achieve a high success probability for the same target $\mathbf{t}$. To verify this assumption, the method was implemented and tested on lattices in dimension 50 with a range of randomly chosen targets to see whether, if the probability of success is small, repeating m times will increase the success rate by a factor m. Figure 2.3 shows performance metrics for various numbers of trials/repetitions and for varying list sizes. In particular, Figure 2.3a illustrates the increased success probability as the number of repetitions increases, and Figure 2.3c shows that the normalized success probability per trial[4] seems independent of the number of repetitions. Therefore, the “expected time” metric as illustrated in Figure 2.3b appears to be independent of the number of trials.

Unlike the success probabilities, the time complexity might vary a lot depending on the underlying nearest neighbor data structure. The experiments in Figure 2.3 was used the hyperplane LSH [Cha02] as it was easy to implement and it is also used in the HashSieve [Laa15]. To put the complexities of Figure 2.3b into perspective, we compare the normalized time complexities for CVPP with the complexities of sieving for SVP, which by [Laa16b] are comparable to the costs for CVP. First, we note that the HashSieve algorithm solves SVP in approximately 4 seconds on the same machine. This means that in dimension 50, the expected time complexity for CVPP with the HashSieve (roughly 2

[4]As the success prob. q for m trials scales as $q = 1 - (1-p)^m$ if each trial independently has success prob. p, the success prob. per trial is computed as $p = 1 - (1-q)^{1/m}$.



(a) Success prob. with rerandomizations

(b) Expected time per CVP instance

(c) Average success prob. per trial

(d) Success prob. per trial, per vector

Figure 2.3: Experimental results for solving CVPP with randomized slicing in dimension 50. Each data point corresponds to 10 000 random target vectors for those parameters.

milliseconds) is approximately 2000 times smaller than the time for solving SVP. To explain this gap, observe that the list size for solving SVP is approximately 4000, and so the HashSieve algorithm needs to perform in the order of 4000 reductions of newly sampled vectors with a list of size 4000. For solving CVPP, we only need to reduce 1 target vector, with a slightly larger list of 10 000 to 15 000 vectors. So we save a factor 4000 on the number of reductions, but the searches are more expensive, leading to a speed-up of less than a factor 4000.

For solving SVP or CVP, the HashSieve [Laa15] reports time complexities in dimension d of $2^{0.45d-19}$ seconds, corresponding to 11 seconds in dimension 50, i.e. a factor 3 slower than here. This is based on doing $n \approx 2^{0.21d}$ reductions of vectors with the list. If doing only one of these searches takes a factor $2^{0.21d}$ less time, and we take into account that for SVP the time complexity is now a factor 3 less than in [Laa15], then we obtain an estimated complexity for CVPP in dimension d of $2^{0.24d-19}/3$, which for $d = 50$ corresponds to approximately 2.6 milliseconds. A rough extrapolation would then lead to a time complexity in dimension 100 of only 11 seconds. This however seems to be rather optimistic – preliminary experiments in dimensions 60 and 70 suggest that the overhead of using a lot of memory may be rather high, as the list size is usually even larger than for standard sieving.

2.11 — Another few dimensions for free

Ducas [Duc18] showed that in practice, one can effectively use the additional vectors found by lattice sieving to solve a few extra dimensions of SVP "for free". More precisely, by running a lattice sieve in a base dimension d, one can solve SVP in dimension $d' = d + \Theta(d/\log d)$ at little additional cost. This is done by taking all vectors returned by a d-dimensional lattice sieve, and running Babai's nearest plane algorithm [Bab86] on all these vectors in the d'-dimensional lattice to find short vectors in the full lattice. If d' is close enough to d, one of these vectors will then be "rounded" to a shortest vector of the full lattice.

On a high level, Ducas' approach can be viewed as a sieving/enumeration hybrid, where the *top* part of enumeration is replaced with sieving, and the bottom part is done regularly as in enumeration, which is essentially equivalent to doing Babai rounding [Bab86]. The approach of using a CVPP-solver inside enumeration is in a sense dual to Ducas' idea, as here the *bottom* part of the enumeration tree is replaced with a (sieving-like) CVPP routine. Since the CVPP complexities of Section 2.9 are strictly better than the best SVP/CVP complexities, we can also gain up to $\Theta(d/\log d)$ dimensions for free as follows:

1. First, we run the CVPP preprocessing on a d-dimensional sublattice of the full lattice of dimension $d' = d + \Theta(d/\log d)$. This may for instance take time $2^{0.292d+o(d)}$ and space $2^{0.208d+o(d)}$ when using the fastest sieve of [BDGL16].
2. Then, we initialize an enumeration tree in the full lattice, and we process the top $k = \varepsilon \cdot d/\log d$ levels as usual in enumeration. This will result in $2^{\Theta(k \log k)} = 2^{\Theta(d)}$ target vectors at level k, and this requires a similar time complexity of $2^{\Theta(d)}$ to generate all these target vectors.
3. Finally, we take the batch of $2^{\Theta(d)}$ target vectors at level k in the enumeration tree, and we solve CVP for each of them with our approximate Voronoi cell, with query time $2^{0.220d+o(d)}$ each.

By setting $k = \varepsilon \cdot d/\log d$ as above with small, constant $\varepsilon > 0$, the costs for solving SVP or CVP in dimension d' are asymptotically dominated by the costs of the preprocessing step, which is as costly as solving SVP or CVP in dimension d. So similar to [Duc18], asymptotically we also get $\Theta(d/\log d)$ dimensions "for free". However, unlike for Ducas' idea, in practice the dimensions are likely not quite as free here, as there is more overhead for doing the CVPP-version of sieving than for Ducas' additional batch of Babai nearest plane calls. This approach is analysed in detail in Chapter 4.

Chapter 3

The irreducible vectors of a lattice

This chapter is for all practical purposes identical to the paper *The irreducible vectors of a lattice: Some theory and applications* [DLdW21] authored jointly with Thijs Laarhoven and Benne de Weger.

3.1 — Introduction

The need for quantum-resistant cryptography has led to rapid developments in the area of lattice-based cryptography, mainly spurred by the NIST PQ-Crypto competition. Large scale deployment of lattice-based cryptosystems in the near future becomes realistic. This continues to make the deeper understanding of lattice problems an urgent research topic.

In 2010 Micciancio and Voulgaris, based also on previous work [AEVZ02], described deterministic $\tilde{O}(2^{2n})$–time and $\tilde{O}(2^{n})$–space algorithms to solve some of the most important lattice problems (such as SVP, SIVP and CVP) [MV10a] in dimension n. This result mainly relies on an algorithm to compute the set of relevant vectors of (the Voronoi cell of) a lattice. Even though this is a very interesting result, the constants in the exponents of time and space complexities of the Micciancio–Voulgaris algorithm make it impractical, even for moderate dimensions.

The set of relevant vectors was first introduced in 1908 by Voronoi [Vor08]. It provides a useful representation of the Voronoi cell of a lattice. Even though the set of relevant vectors seems to hold the key for solving many lattice problems, its expected size makes it impractical. This becomes even more clear when that size is compared to the (time and) space complexity of algorithms used in practice for solving lattice problems such as [ADH+19, GNR10, CN11].

In this chapter, we introduce a different set of lattice vectors, which appears to serve as a bridge between the provable results relying on the set of relevant vectors and heuristic sieving algorithms [NV08, MV10b, BLS16].

Notions of irreducibility are considered to be fundamental in many areas. Often irreducibility is defined with respect to multiplication. Since a lattice is an additive object, we will however use an additive notion of irreducibility. Clearly the notion of *lattice basis* could be seen as such a construct, but it has been observed to be a too weak notion to provide, on its own, interesting results for lattice problems. Our new notion of *irreducible vectors* provides us with a set of lattice vectors, larger than a basis but smaller than the set of relevant vectors, and possessing interesting properties. To the best of our knowledge this definition is new in the area of lattices.

3.1.1 – Contributions. In this chapter we define a notion of irreducibility for a lattice vector. As a first result we show that every irreducible vector of a lattice belongs to the lattice's set of relevant vectors. Hence, the set of irreducible vectors which we denote by $\mathrm{Irr}(\mathcal{L})$ is finite. Additionally, it is shown that the set of irreducible vectors generates the lattice and also contains vectors achieving all the successive minima of the lattice. Finally, the set of irreducible vectors of the root lattices A_n, D_n and their duals A_n^*, D_n^* is examined as they prove to be interesting extreme cases.

As it turns out, the set $\mathrm{Irr}(\mathcal{L})$ can be as big as the set of relevant vectors. In order to get a set of cardinality provably smaller than 2^n, a complete system of irreducible vectors is defined, which is denoted by $\mathrm{P}(\mathcal{L})$. This set inherits the aforementioned properties of the set $\mathrm{Irr}(\mathcal{L})$ and also it is proved that $|\mathrm{P}(\mathcal{L})| < 2^{0.402n}$ where n is the rank of the lattice. Heuristically it is expected that $\mathrm{P}(\mathcal{L})$ will have a cardinality of $2^{0.21n}$. From a computational point of view, it is shown that slightly modified versions of already existing sieving algorithms asymptotically output such a set (modulo sign). This statement is further supported by experimental results. Finally, we discuss the applicability of $\mathrm{P}(\mathcal{L})$ in showing that sieving algorithms like the ones described in [MV10b, BLS16] can be used for tackling SVP, SIVP and computing the kissing number of a lattice. Additionally we discuss the applicability of $\mathrm{P}(\mathcal{L})$ as preprocessing data in a CVPP algorithm which we call "the tuple slicer". The tuple slicer can provide a time–memory trade-off without the use of rerandomisations.

3.1.2 – Notation. We adopt the already introduced notation from Chapter 1 for basic notions regarding lattices and lattice problems. However we point out that in this chapter the dimension of a lattice will be denoted by n instead of d. Let $\mathcal{B}(x, r) \coloneqq \{y \in \mathbb{R}^n \mid \|y - x\| \leqslant r\}$ denote the closed n-dimensional ball with center x and radius r. Finally we have the *kissing number* τ_n, defined as the maximum number of equal n-dimensional spheres that can be made to touch another central sphere of the same size without intersecting.

3.2 — Previous Work

In this section we recall some known results on the set of relevant vectors. This is done for a matter of completeness but also in order to indicate what kind of results we would like to obtain for the set of irreducible vectors which we will define later.

For $\boldsymbol{v} \in \mathcal{L}$ we define $\mathrm{H}(\boldsymbol{v}) \coloneqq \{x \in \mathbb{R}^n \mid \|x\| \leqslant \|x - \boldsymbol{v}\|\}$, to relate the Voronoi cell of a lattice to its relevant vectors.

Proposition 3.1. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. The set of relevant vectors $\mathrm{R}(\mathcal{L})$ is the minimal set $L \subset \mathcal{L}$ such that*

$$\mathcal{V}(\mathcal{L}) = \bigcap_{\boldsymbol{v} \in L} \mathrm{H}(\boldsymbol{v}). \tag{3.1}$$

In order to get a more practical description of the relevant vectors the following theorem is used.

Theorem 3.2 (Voronoi [Vor08])**.** *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$ and $\boldsymbol{v} \in \mathcal{L} \setminus \{\mathbf{0}\}$. Then $\boldsymbol{v} \in \mathrm{R}(\mathcal{L})$ if and only if $\mathbf{0}$ and $\boldsymbol{v}$ are the only closest vectors of $\mathcal{L}$ to $\frac{1}{2}\boldsymbol{v}$.*

This implies that

$$R(\mathcal{L}) = \{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \|\tfrac{1}{2}\mathbf{v} - \mathbf{x}\| > \|\tfrac{1}{2}\mathbf{v}\| \ \ \forall \mathbf{x} \in \mathcal{L} \setminus \{\mathbf{0}, \mathbf{v}\}\} \tag{3.2}$$

$$= \{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \langle \mathbf{v}, \mathbf{x} \rangle < \|\mathbf{x}\|^2 \ \ \forall \mathbf{x} \in \mathcal{L} \setminus \{\mathbf{0}, \mathbf{v}\}\} \tag{3.3}$$

Remark 3.3. *It holds that $\mathbf{0} \notin R(\mathcal{L})$. Also note that if $\mathbf{v} \in R(\mathcal{L})$ then $-\mathbf{v} \in R(\mathcal{L})$.*

Remark 3.4. *The condition $\langle \mathbf{v}, \mathbf{x} \rangle < \|\mathbf{x}\|^2$ needs to be checked only for $\mathbf{x} \in \mathcal{L} \setminus \{\mathbf{0}\}$ such that $\|\mathbf{x}\| < \|\mathbf{v}\|$, because otherwise it is trivially true.*

For checking if a vector is relevant, the following lemma is useful.

Lemma 3.5 ([MV10a]). *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$, and $\mathbf{v} \in \mathcal{L}$. If $\mathbf{v} \notin R(\mathcal{L})$ then there exists $\mathbf{r} \in R(\mathcal{L})$ such that $\langle \mathbf{v}, \mathbf{r} \rangle \geqslant \|\mathbf{r}\|^2$.*

Also a lower bound for the set $R(\mathcal{L})$ can be obtained by the following trivial lemma.

Lemma 3.6. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. It holds that $S_1(\mathcal{L}) \subseteq R(\mathcal{L})$.*

Equality in the above lemma holds for a very special type of lattices called root lattices (see [CS98, Chapter 4]).

Theorem 3.7 ([RS96]). *$S_1(\mathcal{L}) = R(\mathcal{L})$ iff $\mathcal{L}$ is a root lattice.*

The following theorem by Minkowski gives an upper bound on the size of $R(\mathcal{L})$.

Theorem 3.8 (Minkowski [Min11]). *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. It holds that $|R(\mathcal{L})| \leqslant 2(2^n - 1)$.*

Apart from an upper bound we can also obtain a lower bound on $|R(\mathcal{L})|$.

Proposition 3.9. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. It holds that $\mathrm{Vol}(\mathcal{L}) = \mathrm{Vol}(\mathcal{V}(\mathcal{L}))$.*

Remark 3.10. *For the lattice $\mathbb{Z}^n$ is true that $|R(\mathbb{Z}^n)| = 2n$ (see [CS98]). As the set $R(\mathcal{L})$ needs to have n linearly independent vectors in order the volume of $\mathcal{V}(\mathcal{L})$ to be finite then $2n \leqslant |R(\mathcal{L})|$ is a tight lower bound.*

Other properties and results. Finally a property of the set $R(\mathcal{L})$ is that it generates $\mathcal{L}$. The proof of this is rather simple. The Voronoi cell of $\mathcal{L}$ implies a tiling of $\mathbb{R}^n$. Thus, every vector in $\mathbb{R}^n$ can be reduced to a vector in $\mathcal{V}(\mathcal{L})$ through reductions by elements of $R(\mathcal{L})$. As $\mathbf{0}$ is the only lattice vector in $\mathcal{V}(\mathcal{L})$ it follows that all lattice vectors are reduced to $\mathbf{0}$. Therefore, $R(\mathcal{L})$ spans the entire lattice.

So far we have mentioned a number of properties and definitions on the relevant vectors of a lattice. Computing them is however a different matter. The following result is the current state of the art on this.

Theorem 3.11 (Micciancio–Voulgaris [MV10a]). *There exists a deterministic $\tilde{O}(2^{2n})$–time and $\tilde{O}(2^n)$–space algorithm which, given an n-rank lattice $\mathcal{L}$ with basis $\mathbf{B}$, outputs the set of relevant vectors.*

3.3 — Irreducibility of lattice vectors

3.3.1 – The set of irreducible vectors. Inspired by number theoretic notions of (multiplicative) irreducibility, we introduce a similar concept for lattice (additively structured).

Definition 3.12 (Irreducibility). *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$ and $\boldsymbol{v} \in \mathcal{L} \setminus \{\mathbf{0}\}$. The vector $\boldsymbol{v}$ is called k-irreducible iff $\nexists\, \boldsymbol{v}_1, \dots, \boldsymbol{v}_k \in \mathcal{L}$ such that $\|\boldsymbol{v}_i\| < \|\boldsymbol{v}\|$ and $\boldsymbol{v}_1 + \dots + \boldsymbol{v}_k = \boldsymbol{v}$. For the special case $k = 2$, $\boldsymbol{v}$ will be just called irreducible.*

Remark 3.13. *The definition of k-irreducible vectors implies that if a vector is k-irreducible then it is also $(k-1)$-irreducible. This observation allows the construction of a chain of subsets based on the notion of irreducibility.*

In this chapter we are going to focus on the properties of 2-irreducibility. Further research on the notion of k-irreducibility for $k > 2$ is left for future research.

Definition 3.14 (Irreducible vectors). *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. We define*

$$\operatorname{Irr}(\mathcal{L}) := \{\boldsymbol{v} \in \mathcal{L} \mid \boldsymbol{v} \text{ is irreducible}\}. \tag{3.4}$$

Remark 3.15. *It holds that $\mathbf{0} \notin \operatorname{Irr}(\mathcal{L})$. Also, if $\boldsymbol{v} \in \operatorname{Irr}(\mathcal{L})$ then $-\boldsymbol{v} \in \operatorname{Irr}(\mathcal{L})$.*

The above properties hold for the set of relevant vectors as well and this is not a coincidence as we will see. First we show that this set is not empty, and indeed that it also contains vectors achieving the first successive minimum.

Lemma 3.16. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. It holds that $S_1(\mathcal{L}) \subseteq \operatorname{Irr}(\mathcal{L})$.*

Proof. Let $\boldsymbol{v} \in S_1(\mathcal{L})$. Then clearly $\boldsymbol{v} \neq \mathbf{0}$. Assume that $\boldsymbol{v} \notin \operatorname{Irr}(\mathcal{L})$, so there exist $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathcal{L}$ such that $\|\boldsymbol{v}_i\| < \|\boldsymbol{v}\|$ and $\boldsymbol{v}_1 + \boldsymbol{v}_2 = \boldsymbol{v}$. As $\boldsymbol{v} \in S_1(\mathcal{L})$ this implies that $\|\boldsymbol{v}_i\| < \lambda_1(\mathcal{L})$ and thus $\|\boldsymbol{v}_i\| = 0$. Hence, we get $\boldsymbol{v}_1 = \boldsymbol{v}_2 = \mathbf{0}$, which contradicts $\boldsymbol{v} \neq \mathbf{0}$. □

Remark 3.17. *It can be easily checked that Lemma 3.16 would still hold under the notion of k-irreducibility for $k > 2$. Therefore we can conclude that k-irreducibility is not leading to a trivially empty set of vectors for $k > 2$. One may expect that it will also include a lattice basis.*

We show that something similar occurs for the rest of the successive minima as well.

Definition 3.18. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$ and $1 \leqslant i \leqslant n$. We define $\mathcal{L}_\lambda$ to be the sublattice spanned by all the vectors in $\mathcal{L}$ with norm strictly less than λ.*

Proposition 3.19. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$, and $\boldsymbol{v} \in \mathcal{L}$ satisfying $\|\boldsymbol{v}\| = \lambda_i := \lambda_i(\mathcal{L})$ for some $1 \leqslant i \leqslant n$. If $\boldsymbol{v} \notin \mathcal{L}_{\lambda_i}$ then $\boldsymbol{v}$ is irreducible.*

Proof. It has been already proven in Lemma 3.16 that this is true for $i = 1$ so we can consider $i \geqslant 2$. Assume $\boldsymbol{v} \in \mathcal{L}$ such that $\|\boldsymbol{v}\| = \lambda_i(\mathcal{L})$ for some $2 \leqslant i \leqslant n$, $\boldsymbol{v} \notin \mathcal{L}_{\lambda_i}$ and $\boldsymbol{v}$ is not irreducible. Then there exist $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathcal{L}$ such that $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2$ and $\|\boldsymbol{v}_j\| < \|\boldsymbol{v}\|$ for $j = 1, 2$. Clearly $\boldsymbol{v}_j \neq 0$. As $\|\boldsymbol{v}_j\| < \|\boldsymbol{v}\| = \lambda_i(\mathcal{L})$ this implies that $\boldsymbol{v}_j \in \mathcal{L}_{\lambda_i}$ for $j = 1, 2$. This further implies that $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2 \in \mathcal{L}_{\lambda_i}$, contradiction. □

Remark 3.20. *Proposition 3.19 points out that a lattice vector achieving a successive minimum is not necessarily irreducible. An enlightening example of such an occasion is the following. Consider the lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ generated by the matrix*

$$\mathbf{B} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 10 \end{pmatrix}. \tag{3.5}$$

Then $\lambda_1(\mathcal{L}) = 3$, $\lambda_2(\mathcal{L}) = 4$ and $\lambda_3(\mathcal{L}) = 10$. The vector $\mathbf{v} = (6, 8, 0)$ is such that $\|\mathbf{v}\| = \lambda_3(\mathcal{L})$ but $\mathbf{v}$ is not irreducible as it can be written as a sum of shorter vectors i.e. $\mathbf{v} = (6, 0, 0) + (0, 8, 0)$. The reason why $\mathbf{v}$ fails to be irreducible is that it belongs to the sublattice $\mathcal{L}_{\lambda_3}$.

Corollary 3.21. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. For every $i = 1, \dots, n$ there exists a vector $\mathbf{v} \in \operatorname{Irr}(\mathcal{L})$ such that $\|\mathbf{v}\| = \lambda_i(\mathcal{L})$.*

Proof. By Proposition 3.19 it suffices to show that for every $i = 1, \dots, n$ there exists a vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| = \lambda_i(\mathcal{L})$ and $\mathbf{v} \notin \mathcal{L}_{\lambda_i}$. Assume that for every vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| = \lambda_i(\mathcal{L})$ for some fixed $2 \leqslant i \leqslant n$ it holds $\mathbf{v} \in \mathcal{L}_{\lambda_i}$. For convenience we define $\lambda_0(\mathcal{L}) = 0$. Let k be $\min_{1 \leqslant j \leqslant i} j$ such that $\lambda_j(\mathcal{L}) = \lambda_i(\mathcal{L})$ and therefore $\lambda_{k-1}(\mathcal{L}) < \lambda_k(\mathcal{L}) = \lambda_i(\mathcal{L})$. Then $\mathcal{L}_{\lambda_i}$ has rank $k-1$ as $\lambda_{k-1}(\mathcal{L}) < \lambda_k(\mathcal{L})$. If $k - 1 = 0$ then we are done as this would imply $\mathbf{v} = \mathbf{0}$. If $k - 1 \geqslant 1$ then $\mathbf{v}$ belongs to the sublattice containing all the shorter vectors than it, $\mathcal{L}_{\lambda_i}$ and this sublattice is of rank $k - 1$. Thus any choice of $k - 1$ vectors such that $\max\{\|\mathbf{v}_1\|, \dots, \|\mathbf{v}_{k-1}\|, \|\mathbf{v}\|\} = \|\mathbf{v}\|$ will result in a linearly dependent set. Hence it cannot be that $\lambda_k(\mathcal{L}) = \|\mathbf{v}\|$, contradiction. □

Apart from vectors reaching the successive minima, it can be shown that the set $\operatorname{Irr}(\mathcal{L})$ contains a generating set of the lattice as well.

Proposition 3.22 (Irreducible vectors generate). *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. There exists a generating set $\mathbf{G}$ of $\mathcal{L}$ such that $\mathbf{G} \subseteq \operatorname{Irr}(\mathcal{L})$.*

Proof. We will prove that the set $\operatorname{Irr}(\mathcal{L})$ spans the lattice and therefore it includes a generating set. Let $\mathbf{v} \in \mathcal{L}$. If $\nexists \mathbf{v}_1, \mathbf{v}_2 \in \mathcal{L}$ with $\|\mathbf{v}_i\| < \|\mathbf{v}\|$ such that $\mathbf{v}_1 + \mathbf{v}_2 = \mathbf{v}$ then $\mathbf{v} \in \operatorname{Irr}(\mathcal{L})$. If there exist such $\mathbf{v}_i$ then write $\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2$. If the $\mathbf{v}_i \in \operatorname{Irr}(\mathcal{L})$ then we are done. If not then further reduce the vectors $\mathbf{v}_i$ such that they are written as a sum of two strictly shorter vectors. As in each step the length of the vectors strictly reduces and there is a finite number of lattice points in $\mathcal{B}(\mathbf{0}, \|\mathbf{v}\|)$, after a finite number of steps we will reach a state where $\mathbf{v} = \sum \mathbf{p}_i$ and $\mathbf{p}_i \in \operatorname{Irr}(\mathcal{L})$. This concludes the proof. □

Given the result of Proposition 3.22 the following conjecture can be formulated.

Conjecture 3.23. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. The set $\operatorname{Irr}(\mathcal{L})$ contains a basis of $\mathcal{L}$.*

Our next goal is to derive some more explicit descriptions of the set $\operatorname{Irr}(\mathcal{L})$.

Lemma 3.24. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. It holds that*

$$\begin{aligned} \operatorname{Irr}(\mathcal{L}) &= \{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \forall \mathbf{x} \in \mathcal{L} \text{ with } \|\mathbf{x}\| < \|\mathbf{v}\| \text{ it holds } \|\mathbf{v} - \mathbf{x}\| \geqslant \|\mathbf{v}\|\} & (3.6) \\ &= \{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \forall \mathbf{x} \in \mathcal{L} \text{ with } \|\mathbf{x}\| < \|\mathbf{v}\| \text{ it holds } 2\langle \mathbf{v}, \mathbf{x} \rangle \leqslant \|\mathbf{x}\|^2\}. & (3.7) \end{aligned}$$

Proof. Let $A = \{v \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \forall x \in \mathcal{L}$ with $\|x\| < \|v\|$ it holds $\|v - x\| \geqslant \|v\|\}$. Let $p \in \operatorname{Irr}(\mathcal{L})$ and $v \in \mathcal{L}$ with $\|v\| < \|p\|$. Then as $p \in \operatorname{Irr}(\mathcal{L})$ we get $\|p - v\| \geqslant \|p\|$ because otherwise p would have a decomposition into two shorter vectors, thus $p \in A$. This gives $\operatorname{Irr}(\mathcal{L}) \subseteq A$. Next, let $v \in A$, and write $v = v_1 + v_2$ for some $v_1, v_2 \in \mathcal{L}$. If $\|v_1\| < \|v\|$ then as $v \in A$ we get $\|v - v_1\| \geqslant \|v\|$ and hence we do not get a decomposition of v in two shorter vectors. If $\|v_1\| \geqslant \|v\|$ this is trivially true. Thus $v \in \operatorname{Irr}(\mathcal{L})$. This implies equality (3.6) and equality (3.7) is an immediate consequence. This concludes the proof. ☐

Even though this lemma is rather straightforward it implies an interesting result for the set $\operatorname{Irr}(\mathcal{L})$.

Proposition 3.25. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. Every irreducible vector of $\mathcal{L}$ is also a relevant vector of $\mathcal{L}$, hence* $\operatorname{Irr}(\mathcal{L}) \subseteq R(\mathcal{L})$.

Proof. As we already saw by Theorem 3.2, we can write the set $R(\mathcal{L})$ as $R(\mathcal{L}) = \{v \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \langle v, x\rangle < \|x\|^2 \;\; \forall x \in \mathcal{L} \setminus \{\mathbf{0}, v\}\}$ and we can further improve that description to

$$R(\mathcal{L}) = \{v \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \forall x \in \mathcal{L} \setminus \{\mathbf{0}\} \text{ with } \|x\| < \|v\| \text{ it holds } \langle v, x\rangle < \|x\|^2\}.$$

For the set of irreducible vectors we got from Lemma 3.24 that

$$\operatorname{Irr}(\mathcal{L}) = \{v \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \forall x \in \mathcal{L} \text{ with } \|x\| < \|v\| \text{ it holds } 2\langle v, x\rangle \leqslant \|x\|^2\}.$$

Thus by carefully checking these two descriptions for the sets $R(\mathcal{L})$ and $\operatorname{Irr}(\mathcal{L})$ it suffices to prove that if $v \in \mathcal{L} \setminus \{\mathbf{0}\}$ and $x \in \mathcal{L} \setminus \{\mathbf{0}\}$ with $\|x\| < \|v\|$ then $2\langle v, x\rangle \leqslant \|x\|^2 \Rightarrow \langle v, x\rangle < \|x\|^2$.
If $\langle v, x\rangle \leqslant 0$ this is trivially true as $x \neq \mathbf{0}$. Also if $\langle v, x\rangle > 0$ then $\langle v, x\rangle < 2\langle v, x\rangle$ and the result follows. ☐

Remark 3.26. *Combining the result of Lemma 3.16 and Proposition 3.25 we get that* $S_1(\mathcal{L}) \subseteq \operatorname{Irr}(\mathcal{L}) \subseteq R(\mathcal{L})$. *Therefore* $\operatorname{Irr}(\mathcal{L})$ *is finite.*

We already saw that for the case of root lattices it holds $S_1(\mathcal{L}) = R(\mathcal{L})$. This implies that for the root lattices it also holds that $S_1(\mathcal{L}) = \operatorname{Irr}(\mathcal{L}) = R(\mathcal{L})$. Thus, the sets $S_1(\mathcal{L})$ and $R(\mathcal{L})$ are tight inclusions of $\operatorname{Irr}(\mathcal{L})$.

We expect that in general though it will hold $S_1(\mathcal{L}) \subsetneq \operatorname{Irr}(\mathcal{L}) \subsetneq R(\mathcal{L})$. A question that might be of interest is when and if $S_1(\mathcal{L}) = \operatorname{Irr}(\mathcal{L}) \subsetneq R(\mathcal{L})$ or $S_1(\mathcal{L}) \subsetneq \operatorname{Irr}(\mathcal{L}) = R(\mathcal{L})$ are possible.

We believe that lattices satisfying either of these properties will be very special and highly symmetric. The reason why we believe this, is that some already well known very special families of lattices satisfy these properties. Namely, in Section 3.6 we will prove the following two theorems.

Theorem 3.27. *Let $n \in \mathbb{N}$ with $n \geqslant 5$. Then for the lattice D_n^* it holds that* $S_1(D_n^*) \subsetneq \operatorname{Irr}(D_n^*) = R(D_n^*)$. *Furthermore* $|\operatorname{Irr}(D_n^*)| = 2^n + 2n$.

Theorem 3.28. *Let $n \in \mathbb{N}$ with $n \geqslant 3$. Then for the lattice A_n^* it holds that* $S_1(A_n^*) = \operatorname{Irr}(A_n^*) \subsetneq R(A_n^*)$. *Furthermore* $|\operatorname{Irr}(A_n^*)| = 2(n+1)$.

Additionally the famous Leech lattice Λ_{24} [CS98, p. 131] satisfies the property $S_1(\Lambda_{24}) = \operatorname{Irr}(\Lambda_{24}) \subsetneq R(\Lambda_{24})$. We will actually be able to prove in the next subsection that for every lattice that reaches the kissing number τ_n it holds that $S_1(\mathcal{L}) = \operatorname{Irr}(\mathcal{L})$.

3.3.2 – A complete system of irreducible vectors. The special family of lattices D_n^* indicates that the set $\mathrm{Irr}(\mathcal{L})$ can become as big as $R(\mathcal{L})$ and actually grow as much in size as 2^n. However, our goal is to obtain a subset of $\mathrm{Irr}(\mathcal{L})$ which is closely related to it but also provably smaller than 2^n.

Definition 3.29. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. We define an equivalence relation in $\mathrm{Irr}(\mathcal{L})$ in the following way.*

$$\text{Let } \boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathrm{Irr}(\mathcal{L}) \quad \text{then} \quad \boldsymbol{v}_1 \sim \boldsymbol{v}_2 \quad \text{iff} \quad \|\boldsymbol{v}_1\| = \|\boldsymbol{v}_2\|. \tag{3.8}$$

From each equivalence class we will consider a representative set instead of just one element. We choose it in the following way and we will explain afterwards why.

Definition 3.30. *For each equivalence class $S = \{\boldsymbol{v}_1, \dots, \boldsymbol{v}_m\}$ of $\mathrm{Irr}(\mathcal{L})$ according to (3.8) we choose a subset $\tilde{S} \subseteq S$ such that the following two conditions hold:*

(i) If $\boldsymbol{v} \in \tilde{S}$ then also $-\boldsymbol{v} \in \tilde{S}$.

(ii) $\tilde{S}$ is a maximal subset of S such that for every pair of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \tilde{S}$ with $\boldsymbol{v}_2 \neq -\boldsymbol{v}_1$ it holds that $\|\boldsymbol{v}_1 + \boldsymbol{v}_2\| \geqslant \|\boldsymbol{v}_1\|$.

The main motivation is that the new set of vectors which will be built under these rules will include irreducible vectors whose pairwise angle is "big" as we will prove later. However, there are several details of this definition which should be clarified. First of all, from the definition it follows that for an equivalence class we consider at least two representatives, which is not usually done. The reasons for this are the following.

Initially, for the subset of $\mathrm{Irr}(\mathcal{L})$ which we are trying to define, we would like it to inherit the property of $\mathrm{Irr}(\mathcal{L})$ that if $\boldsymbol{v}$ belongs to it then also $-\boldsymbol{v}$ belongs to it. A second, more important reason is that choosing only one representative per equivalence class could lead to a set that does not even span the lattice (for example in the case of root lattices, or whenever $S_1(\mathcal{L}) = \mathrm{Irr}(\mathcal{L})$).

The second condition of the definition implies that for every element $\boldsymbol{v}$ of a class S which is not included in $\tilde{S}$ there exists a vector $\tilde{\boldsymbol{v}} \in \mathcal{L}$ such that $\|\boldsymbol{v} - \tilde{\boldsymbol{v}}\| < \|\boldsymbol{v}\|$. From this point of view the remaining elements of a class S which are not included in $\tilde{S}$ can be generated by the elements of $\tilde{S}$ plus some strictly shorter vector. In order to ensure that this holds we take $\tilde{S}$ to be maximal. Also by taking $\tilde{S}$ to be maximal we make sure that the set $\tilde{S}$ contains as much information about the class as possible.

Remark 3.31. *Choosing a representative set $\tilde{S}$ of a class S can be translated into a graph problem. We define a graph where the set of vertices is the equivalence class, and there exists an edge between two vertices iff the difference of the corresponding vectors is strictly shorter than both of them. Then choosing a set of representatives translates to finding a maximal subset of vertices that are not adjacent, while keeping the symmetry about $\mathbf{0}$. In terms of graph theory this can be phrased as finding a special independent set of the graph. This idea is further analysed in Section 3.7.*

Definition 3.32. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. We define a set $P \subseteq \mathrm{Irr}(\mathcal{L})$ to be a complete system of irreducible vectors of $\mathcal{L}$ if it is of the form:*

$$P = \bigcup_{S \in \mathrm{Irr}(\mathcal{L})/\sim} \tilde{S}. \tag{3.9}$$

Remark 3.33. *Below we denote by* $\mathrm{P}(\mathcal{L})$ *any one of the complete systems of irreducible vectors of* $\mathcal{L}$*. It is clear that there always exists such a set* $\mathrm{P}(\mathcal{L})$ *and it is not necessarily unique. In fact, even the size of* $\mathrm{P}(\mathcal{L})$ *can vary.*

Remark 3.34. *By the fact that for each class* S *of* $\mathrm{Irr}(\mathcal{L})/\sim$ *we have* $\tilde{\mathrm{S}} \subseteq \mathrm{S}$ *we get that* $\mathrm{P}(\mathcal{L}) \subseteq \mathrm{Irr}(\mathcal{L})$*. Also the class of* $\mathrm{Irr}(\mathcal{L})/\sim$ *containing all the shortest vectors i.e.* $\mathrm{S}_1(\mathcal{L})$ *will be entirely included in* $\mathrm{P}(\mathcal{L})$ *as any pairwise sum of vectors (for non-trivial pairs) in this class will be longer or equally long by definition. Thus we can conclude that*

$$\mathrm{S}_1(\mathcal{L}) \subseteq \mathrm{P}(\mathcal{L}) \subseteq \mathrm{Irr}(\mathcal{L}) \subseteq \mathrm{R}(\mathcal{L}).$$

We will also give an example in order to illustrate this definition.

Example 3.35. *Let* $\mathcal{L} = \mathcal{L}(\mathbf{B})$ *be the lattice generated by the columns of the matrix*

$$\mathbf{B} = \begin{pmatrix} 0 & 0 & 0 & -1 & 3 \\ -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 & 1 \end{pmatrix}.$$

We find the sets $\mathrm{S}_1(\mathcal{L}), \mathrm{P}(\mathcal{L}), \mathrm{Irr}(\mathcal{L}), \mathrm{R}(\mathcal{L})$.

In fact $\mathbf{B}$ *is an LLL-reduced basis [LLL82] of the lattice. By means of enumeration one could verify that* $\mathrm{S}_1(\mathcal{L}) = \{\pm(0,-1,0,1,0)\}$*. By running an algorithm that computes the set of relevant vectors like [VB96] in SAGE [TSD19] we get*

$$\begin{aligned} \mathrm{R}(\mathcal{L}) = \{ & \pm(0,-1,0,1,0), \\ & \pm(0,-1,0,-1,1), \\ & \pm(0,0,2,0,0), \\ & \pm(-1,0,-1,1,2), \pm(-1,0,1,1,2), \pm(-1,1,-1,0,2), \pm(-1,1,1,0,2), \\ & \pm(-1,1,-1,2,1), \pm(-1,1,1,2,1), \pm(-1,2,-1,1,1), \pm(-1,2,1,1,1), \\ & \pm(3,1,0,0,1), \pm(3,0,0,1,1), \\ & \pm(2,1,-1,1,3), \pm(2,1,1,1,3), \\ & \pm(2,2,-1,2,2), \pm(2,2,1,2,2)\}. \end{aligned}$$

(each line has vectors of equal norm). The next step is to find the set of irreducible vectors $\mathrm{Irr}(\mathcal{L})$*. We consider the subset of* $\mathrm{R}(\mathcal{L})$ *containing relevant vectors which cannot be written as a sum of two strictly shorter vectors (by cross-checking with the set of relevant vectors). It turns out that this set just contains all the vectors achieving the successive minima thus it must be that this is* $\mathrm{Irr}(\mathcal{L})$.

$$\begin{aligned} \mathrm{Irr}(\mathcal{L}) = \{ & \pm(0,-1,0,1,0), \\ & \pm(0,-1,0,-1,1), \\ & \pm(0,0,2,0,0), \\ & \pm(-1,0,-1,1,2), \pm(-1,0,1,1,2), \pm(-1,1,-1,0,2), \pm(-1,1,1,0,2), \\ & \pm(3,1,0,0,1), \pm(3,0,0,1,1)\} \end{aligned}$$

The set $\mathrm{Irr}(\mathcal{L})$ *contains* 5 *equivalence classes according to the equivalence relation* (3.8). *We denote them by* C_i *for* $i = 1, \ldots, 5$. *As we can see for the first three of them, computing a set of representatives* $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$ *is trivial as in these cases it will be* $\tilde{C}_1 = C_1$, $\tilde{C}_2 = C_2$ *and* $\tilde{C}_3 = C_3$. *The cases of* C_4 *and* C_5 *are more interesting. We start by examining* C_5 *as it contains fewer vectors. We set* $\mathbf{v}_1 = (3, 1, 0, 0, 1)$ *and* $\mathbf{v}_2 = (3, 0, 0, 1, 1)$. *Next we draw the corresponding graph with vertices the* $\pm\mathbf{v}_1, \pm\mathbf{v}_2$ *and edges if the pairwise differences are strictly shorter than* $\|\mathbf{v}_1\|$.

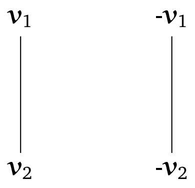


Figure 3.1: The graph of the class C_5

The graph in Figure 3.1 *shows that we can take either* $\tilde{C}_5 = \{\pm\mathbf{v}_1\}$ *or* $\tilde{C}_5 = \{\pm\mathbf{v}_2\}$. *We are now going to do the same for the class* C_4. *We set* $\mathbf{v}_1 = (-1, 0, -1, 1, 2)$,
$\mathbf{v}_2 = (-1, 0, 1, 1, 2), \mathbf{v}_3 = (-1, 1, -1, 0, 2), \mathbf{v}_4 = (-1, 1, 1, 0, 2)$.

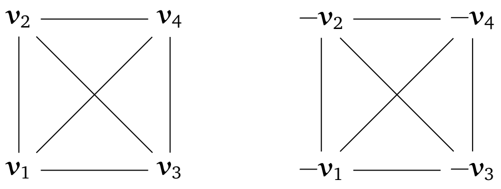


Figure 3.2: The graph of the class C_4

The graph in Figure 3.2 *shows that we can take* $\tilde{C}_4 = \{\pm\mathbf{v}_i\}$ *for any* $i = 1, 2, 3, 4$. *Therefore one choice for the set* $P(\mathcal{L})$ *is the following.*

$$\begin{aligned} P(\mathcal{L}) = \{ & \pm(0, -1, 0, 1, 0), \\ & \pm(0, -1, 0, -1, 1), \\ & \pm(0, 0, 2, 0, 0), \\ & \pm(-1, 0, -1, 1, 2), \\ & \pm(3, 1, 0, 0, 1)\} \end{aligned}$$

Remark 3.36. *The above example should not mislead the reader to the assumption that the corresponding graph of each equivalence class will always have at least two connected components. It can happen that the graph of a class is connected. One such example can be derived from the family of lattices examined in Theorem* 3.65.

One property of the set $\mathrm{Irr}(\mathcal{L})$ was that it includes a generating set of $\mathcal{L}$. We can show that $P(\mathcal{L})$ inherits that property.

Proposition 3.37 (Complete system generates)**.** *Let* $\mathcal{L}$ *be a full rank lattice in* $\mathbb{R}^n$. *Then for every* $P(\mathcal{L}) \subseteq \mathrm{Irr}(\mathcal{L})$ *there exists a generating set* $\mathbf{G}$ *of* $\mathcal{L}$ *such that* $\mathbf{G} \subseteq P(\mathcal{L})$.

Proof. As in the proof of Proposition 3.22 for the set $\mathrm{Irr}(\mathcal{L})$ we will prove that the set $P(\mathcal{L})$ spans the lattice and therefore it includes a generating set. However, in this case the proof is more technical. Let $P(\mathcal{L})$ be a complete system of irreducible vectors of $\mathcal{L}$ as defined in (3.9). We have already shown that $\mathrm{Irr}(\mathcal{L})$ is finite as $\mathrm{Irr}(\mathcal{L}) \subseteq R(\mathcal{L})$ and thus we can define $t := |\mathrm{Irr}(\mathcal{L})/{\sim}|$. We further set C_i for $i = 1, \dots, t$ to be the equivalence classes in $\mathrm{Irr}(\mathcal{L})/{\sim}$. Hence, the set $P(\mathcal{L})$ can be written as $P(\mathcal{L}) = \cup_{i=1}^{t} \tilde{C}_i$. Each equivalence class C_i contains all irreducible vectors of a specific length μ_i, and we can assume that we have ordered the C_i according to increasing μ_i. We define the following sequence of subsets of $\mathrm{Irr}(\mathcal{L})$:

$$A_i := \left(\bigcup_{j=1}^{i-1} C_i\right) \cup \left(\bigcup_{j=i}^{t} \tilde{C}_i\right) \quad \text{for} \quad i = 1, \dots, t+1.$$

As for every i it holds $\tilde{C}_i \subseteq C_i$ then it follows that $A_i(\mathcal{L}) \subseteq A_{i+1}(\mathcal{L})$ and thus

$$P(\mathcal{L}) = A_1 \subseteq A_2 \subseteq \cdots \subseteq A_t \subseteq A_{t+1} = \mathrm{Irr}(\mathcal{L}).$$

We will prove by induction that each term of this sequence of sets spans the lattice $\mathcal{L}$.
Base case $i = t+1$: The set $\mathrm{Irr}(\mathcal{L}) = A_{t+1}$ spans the lattice as it was already shown in Proposition 3.22.
Induction hypothesis: Assume that it holds for some $i = k$, i.e. A_k spans the lattice for some $k \in \{2, \dots, t+1\}$.
Induction step: Prove that A_{k-1} spans the lattice. By the definition of the sets A_i we can conclude that $A_{k-1} = A_k \setminus (C_{k-1} \setminus \tilde{C}_{k-1})$. By the induction hypothesis it suffices to show that the vectors in $C_{k-1} \setminus \tilde{C}_{k-1}$ can be generated by the vectors in A_{k-1}. Let $\boldsymbol{v} \in C_{k-1} \setminus \tilde{C}_{k-1}$. As $\boldsymbol{v} \in C_{k-1}$ but $\boldsymbol{v} \notin \tilde{C}_{k-1}$ this implies that there exists a $\tilde{\boldsymbol{v}} \in \tilde{C}_{k-1}$ such that $\|\boldsymbol{v} + \tilde{\boldsymbol{v}}\| < \|\boldsymbol{v}\|$. This holds because $\tilde{C}_{k-1}$ is maximal by definition. We set $\boldsymbol{w} = \boldsymbol{v} + \tilde{\boldsymbol{v}}$. Furthermore as $\|\boldsymbol{w}\| < \|\boldsymbol{v}\|$ then $\boldsymbol{w}$ is either irreducible or can be written as a sum of irreducible vectors shorter than $\|\boldsymbol{v}\|$. We use the ordering of the C_i. Thus by its definition the set A_{k-1} contains all the vectors in $\mathrm{Irr}(\mathcal{L})$ which are shorter than $\|\boldsymbol{v}\|$. Hence as, $\|\boldsymbol{w}\| < \|\boldsymbol{v}\|$ this implies that $\boldsymbol{w}$ can be generated by the vectors in A_{k-1}. So, concluding we wrote $\boldsymbol{v}$ as $\boldsymbol{v} = \boldsymbol{w} - \tilde{\boldsymbol{v}}$ where both $\boldsymbol{w}$ and $\tilde{\boldsymbol{v}}$ belong to A_{k-1}. This concludes the proof. □

For $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathcal{L}$ we denote by $\vartheta(\boldsymbol{v}_1, \boldsymbol{v}_2)$ the angle formed by $\boldsymbol{v}_1, \boldsymbol{v}_2$.

Proposition 3.38. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$, and $\mathbf{p}_1, \mathbf{p}_2 \in P(\mathcal{L})$ such that $\mathbf{p}_1 \neq \pm\mathbf{p}_2$. Then it holds that*
(i) $\min\{\|\mathbf{p}_1 \pm \mathbf{p}_2\|\} \geqslant \max\{\|\mathbf{p}_1\|, \|\mathbf{p}_2\|\}$ *and*
(ii) $|\cos\vartheta(\mathbf{p}_1, \mathbf{p}_2)| \leqslant \frac{1}{2}$.

Proof. (Part i) By Lemma 3.24 we have that

$$\mathrm{Irr}(\mathcal{L}) = \{\boldsymbol{v} \in \mathcal{L} \setminus \{\mathbf{0}\} \mid \forall \mathbf{x} \in \mathcal{L} \text{ with } \|\mathbf{x}\| < \|\boldsymbol{v}\| \text{ it holds } \|\boldsymbol{v} - \mathbf{x}\| \geqslant \|\boldsymbol{v}\|\}.$$

Let $\mathbf{p}_1, \mathbf{p}_2 \in P(\mathcal{L})$ such that $\mathbf{p}_1 \neq \pm\mathbf{p}_2$. Without loss of generality we assume that $\|\mathbf{p}_2\| \leqslant \|\mathbf{p}_1\|$. Initially we will prove that $\|\mathbf{p}_1 + \mathbf{p}_2\| \geqslant \max\{\|\mathbf{p}_1\|, \|\mathbf{p}_2\|\}$.
Case 1: If $\|\mathbf{p}_2\| < \|\mathbf{p}_1\|$. Then $\mathbf{p}_1, \mathbf{p}_2 \in \mathrm{Irr}(\mathcal{L})$ and they are not in the same class. Using

the description of the Lemma 3.24 with $\mathbf{v} = \mathbf{p}_1 \in \mathrm{Irr}(\mathcal{L})$ and $\mathbf{x} = -\mathbf{p}_2$ we get $\|\mathbf{p}_1+\mathbf{p}_2\| \geqslant \|\mathbf{p}_1\|$. But as $\|\mathbf{p}_2\| < \|\mathbf{p}_1\|$ we can conclude that $\|\mathbf{p}_1 + \mathbf{p}_2\| \geqslant \max\{\|\mathbf{p}_1\|, \|\mathbf{p}_2\|\}$.
Case 2: If $\|\mathbf{p}_2\| = \|\mathbf{p}_1\|$. Then $\mathbf{p}_1, \mathbf{p}_2 \in \mathrm{Irr}(\mathcal{L})$ and they are in the same class. Let $S \in \mathrm{Irr}(\mathcal{L})/\sim$ such that $\mathbf{p}_1, \mathbf{p}_2 \in S$. Then as $\mathbf{p}_1, \mathbf{p}_2 \in P(\mathcal{L})$ we get that $\mathbf{p}_1, \mathbf{p}_2$ belong to the same $\tilde{S}$. Thus, by the definition of $\tilde{S}$ we can again conclude that $\|\mathbf{p}_1 + \mathbf{p}_2\| \geqslant \max\{\|\mathbf{p}_1\|, \|\mathbf{p}_2\|\}$.
The result follows from the fact that for every $\mathbf{v} \in P(\mathcal{L})$ also $-\mathbf{v} \in P(\mathcal{L})$.
(Part ii) Let $\mathbf{p}_1, \mathbf{p}_2 \in P(\mathcal{L})$ such that $\mathbf{p}_1 \neq \pm\mathbf{p}_2$. Without loss of generality we assume that $\|\mathbf{p}_2\| \leqslant \|\mathbf{p}_1\|$. By part (i) we get that $\|\mathbf{p}_1 \pm \mathbf{p}_2\| \geqslant \|\mathbf{p}_1\|$. This in turn implies that $2|\langle \mathbf{p}_1, \mathbf{p}_2\rangle| \leqslant \|\mathbf{p}_2\|^2$. Hence,

$$|\cos\vartheta(\mathbf{p}_1, \mathbf{p}_2)| = \frac{|\langle \mathbf{p}_1, \mathbf{p}_2\rangle|}{\|\mathbf{p}_1\|\|\mathbf{p}_2\|} \leqslant \frac{|\langle \mathbf{p}_1, \mathbf{p}_2\rangle|}{\|\mathbf{p}_2\|^2} \leqslant \frac{1}{2}.$$

□

We will use the following theorem in order to bound $|P(\mathcal{L})|$.

Theorem 3.39 ([KL78]). *Let $A(n, \phi_0)$ be the maximal size of any set C of points in $\mathbb{R}^n$ such that the angle between any two distinct vectors $\mathbf{v}_i, \mathbf{v}_j \in C$ (denoted $\phi_{\mathbf{v}_i,\mathbf{v}_j}$) is at least ϕ_0. If $0 < \phi_0 < 63°$, then for all sufficiently large n, $A(n, \phi_0) = 2^{cn}$ for some*

$$c \leqslant -\frac{1}{2}\log_2(1 - \cos(\phi_0)) - 0.099. \tag{3.10}$$

Proposition 3.40. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. It holds that $|P(\mathcal{L})| < 2^{0.402n}$.*

Proof. By using Theorem 3.39 with $\phi_0 = \frac{\pi}{3}$ (which can be deduced from Proposition 3.38) we get that $|P(\mathcal{L})| = 2^{cn}$ with $c \leqslant -\frac{1}{2}\log_2(1 - \cos(\frac{\pi}{3})) - 0.099$. Evaluating the right hand side of this inequality implies the result. □

Proposition 3.38 states the same condition that is also satisfied by the output of the GaussSieve algorithm described in [MV10b]. As in the paper describing the GaussSieve algorithm [MV10b] the size of $P(\mathcal{L})$ can actually be bounded by the kissing number τ_n. Following the same arguments as in [MV10b] we can argue that in practice we expect $P(\mathcal{L}) \approx 2^{0.21n}$ which is a factor 2 smaller in the exponent than the provable bound $|P(\mathcal{L})| < 2^{0.402n}$.

A result that might be of interest in the search for lattices reaching the kissing number is the following.

Theorem 3.41. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. If the lattice $\mathcal{L}$ is such that it reaches the kissing number τ_n then $S_1(\mathcal{L}) = \mathrm{Irr}(\mathcal{L})$.*

Proof. As the lattice $\mathcal{L}$ reaches the kissing number τ_n, that implies $|S_1(\mathcal{L})| = \tau_n$. By Proposition 3.38 we can conclude that the angle between any two vectors in $P(\mathcal{L})$ is at least $\pi/3$. This is also the minimal possible angle between the centers of two equal n-dimensional spheres which touch another central sphere of the same size without intersecting. Hence $|P(\mathcal{L})| \leqslant \tau_n$. Combining this with $S_1(\mathcal{L}) \subseteq P(\mathcal{L})$ and $|S_1(\mathcal{L})| = \tau_n$ implies that $P(\mathcal{L}) = S_1(\mathcal{L})$. As the set $P(\mathcal{L})$ was build from classes of $\mathrm{Irr}(\mathcal{L})$ and we showed that it actually contains only vectors of norm $\lambda_1(\mathcal{L})$ that means that there is only one class in $\mathrm{Irr}(\mathcal{L})/\sim$, namely the class of $S_1(\mathcal{L})$. But in this class there is no pair of vectors that adds to a shorter one, thus the whole class is included in $P(\mathcal{L})$. That implies that $\mathrm{Irr}(\mathcal{L}) = P(\mathcal{L}) = S_1(\mathcal{L})$. □

Remark 3.42. *A similar result for the set* $R(\mathcal{L})$ *is not possible. For example for the root lattice* E_8 *reaching the kissing number in dimension* 8 *it holds* $S_1(E_8) = R(E_8)$ *but for the Leech lattice* Λ_{24} *it holds that* $S_1(\Lambda_{24}) \subsetneq R(\Lambda_{24})$ *(see [CS98]).*

3.4 — Computation of the set $P(\mathcal{L})$

In the previous sections we investigated some properties of the set $P(\mathcal{L})$ and its relation to the set $R(\mathcal{L})$. Ultimately we aim at using this set instead of $R(\mathcal{L})$ due to its provably smaller cardinality. However, in order to actually benefit from this replacement an algorithm that computes $P(\mathcal{L})$ without using the set $R(\mathcal{L})$ is needed. The goal of this section is to examine ways of computing the set $P(\mathcal{L})$.

3.4.1 – The “brute force” approach. If the set $\mathrm{Irr}(\mathcal{L})$ is given then the set $P(\mathcal{L})$ can be computed by means of a graph-based technique already described in Remark 3.31 and further analysed in Section 3.7. Thus, it suffices to describe an algorithm which computes the set $\mathrm{Irr}(\mathcal{L})$. The naive approach is to use the fact that $\mathrm{Irr}(\mathcal{L}) \subseteq R(\mathcal{L})$. Hence, as a first step one can run the algorithm described in [MV10a] in order to get the set $R(\mathcal{L})$. Then having a superset of $\mathrm{Irr}(\mathcal{L})$ it suffices to remove all the reducible vectors from it. This can be done by iterating through $R(\mathcal{L})$ and checking for each $\mathbf{r} \in R(\mathcal{L})$ if there exists a $\mathbf{v} \in R(\mathcal{L})$ such that $\|\mathbf{v}\| < \|\mathbf{r}\|$ and $\|\mathbf{r}-\mathbf{v}\| < \|\mathbf{r}\|$. If $\mathbf{r} \in \mathrm{Irr}(\mathcal{L})$ then by definition there will not exist a vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| < \|\mathbf{r}\|$ and $\|\mathbf{r}-\mathbf{v}\| < \|\mathbf{r}\|$ and thus the algorithm will not discard any of the irreducible vectors. If the vector $\mathbf{r}$ is reducible then we need the following heuristic assumption.

Assumption 3.43. *Let* $\mathcal{L}$ *be a full rank lattice in* $\mathbb{R}^n$ *with* $\mathrm{Irr}(\mathcal{L}) \neq R(\mathcal{L})$. *If* $\mathbf{r} \in R(\mathcal{L}) \setminus \mathrm{Irr}(\mathcal{L})$ *then* $\exists \mathbf{v} \in R(\mathcal{L})$ *such that* $\|\mathbf{v}\| < \|\mathbf{r}\|$ *and* $\|\mathbf{r}-\mathbf{v}\| < \|\mathbf{r}\|$.

Heuristic assumption 3.43 can be considered as the analogue of Lemma 3.5 for the set $\mathrm{Irr}(\mathcal{L})$. Lemma 3.5 guaranteed that for every non-relevant vector there would exist a relevant vector acting as a “witness” of “non-relevancy”. Heuristic assumption 3.43 speculates that for every reducible relevant vector there exists a relevant vector acting as a “witness” of reducibility. This claim can be further supported by the heuristic expectation for the set $R(\mathcal{L})$ to include most of the “short” lattice vectors. Some experimental support can be derived for low dimensional lattices from Table 3.1 and Figure 3.3a.

Remark 3.44. *Under Heuristic assumption 3.43 and the result of [MV10a] on computing the set* $R(\mathcal{L})$ *we can conclude that computing the set* $\mathrm{Irr}(\mathcal{L})$ *by “brute-force” can take up to* $\tilde{O}(2^{2n})$*–time and* $\tilde{O}(2^n)$*–space. This complexity can serve as an upper bound on the computation of the set* $\mathrm{Irr}(\mathcal{L})$. *Combining this observation with the discussion in Section 3.7 can give an upper bound in the complexity of computing* $P(\mathcal{L})$. *Namely, for lattices which are not extremely structured (i.e.* $\max_{S \in \mathrm{Irr}(\mathcal{L})/\sim} |S| = \mathrm{poly}(n)$ *) we can conclude that computing* $P(\mathcal{L})$ *from* $\mathrm{Irr}(\mathcal{L})$ *can take* $O(\mathrm{poly}(n)|\mathrm{Irr}(\mathcal{L})|)$ *time. Therefore the computation of* $\mathrm{Irr}(\mathcal{L})$ *dominates the time complexity, leading to an overall upper bound for* $P(\mathcal{L})$ *of* $\tilde{O}(2^{2n})$*–time.*

However the approach in the next section could offer a better performance.

Algorithm 3.1 The GaussSieve algorithm as described in [MV10b]

Require: A basis $\mathbf{B}$ of a lattice $\mathcal{L}(\mathbf{B})$ and a $c > 0$.
Ensure: A list $L \subset \mathcal{L}$ s.t. $\min\{\|\mathbf{v}_1 \pm \mathbf{v}_2\|\} \geqslant \max\{\|\mathbf{v}_1\|, \|\mathbf{v}_2\|\}$ for all $\mathbf{v}_1, \mathbf{v}_2 \in L$.

```
function GaussSieve($\mathbf{B}, c$)
    $L \leftarrow \{\mathbf{0}\}$, $S \leftarrow \{\}$, $K \leftarrow 0$
    while $K < c$ do
        if $S$ is not empty then
            $\mathbf{v}_{\text{new}} \leftarrow$ S.pop()
        else
            $\mathbf{v}_{\text{new}} \leftarrow$ SampleGaussian($\mathbf{B}$)
        end if
        $\mathbf{v}_{\text{new}} \leftarrow$ GaussReduce($\mathbf{v}_{\text{new}}, L, S$)
        if $\mathbf{v}_{\text{new}} = \mathbf{0}$ then
            $K \leftarrow K + 1$
        else
            $L \leftarrow L \cup \{\mathbf{v}_{\text{new}}\}$
        end if
    end while
end function
```

```
function GaussReduce($\mathbf{p}, L, S$)
    while $\exists \mathbf{v}_i \in L : \|\mathbf{v}_i\| \leqslant \|\mathbf{p}\|$
            $\wedge \|\mathbf{p} - \mathbf{v}_i\| \leqslant \|\mathbf{p}\|$ do
        $\mathbf{p} \leftarrow \mathbf{p} - \mathbf{v}_i$
    end while
    while $\exists \mathbf{v}_i \in L : \|\mathbf{v}_i\| > \|\mathbf{p}\|$
            $\wedge \|\mathbf{v}_i - \mathbf{p}\| \leqslant \|\mathbf{v}_i\|$ do
        $L \leftarrow L \setminus \{\mathbf{v}_i\}$
        S.push($\mathbf{v}_i - \mathbf{p}$)
    end while
    return $\mathbf{p}$
end function
```

3.4.2 – Using the GaussSieve/MinkowskiSieve **algorithms.** As it was already mentioned in Section 3.3.2, it is expected that the output of the GaussSieve algorithm [MV10b] will be closely related to a set $P(\mathcal{L})$. This conjecture was motivated by the fact that both sets, $P(\mathcal{L})$ and the output of the GaussSieve, possess the property $\min\{\|\mathbf{v}_1 \pm \mathbf{v}_2\|\} \geqslant \max\{\|\mathbf{v}_1\|, \|\mathbf{v}_2\|\}$ for any pair of $\mathbf{v}_1 \neq \pm\mathbf{v}_2$ in the set. At this point it should be clarified that for our purposes we will consider a slightly modified version of the GaussSieve algorithm which will be described here.

Algorithm 3.2 The modified GaussReduce function

```
1:  function PrimeGaussReduce($\mathbf{p}, L, S$)[1]
2:      while $\exists \mathbf{v}_i \in L : \|\mathbf{v}_i\| \leqslant \|\mathbf{p}\| \wedge \|\mathbf{p} \pm \mathbf{v}_i\| < \|\mathbf{p}\|$ do
3:          if $\|\mathbf{p} + \mathbf{v}_i\| < \|\mathbf{p}\|$ then
4:              $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{v}_i$
5:          else
6:              $\mathbf{p} \leftarrow \mathbf{p} - \mathbf{v}_i$
7:          end if
8:      end while
9:      while $\exists \mathbf{v}_i \in L : \|\mathbf{v}_i\| > \|\mathbf{p}\| \wedge \|\mathbf{v}_i \pm \mathbf{p}\| < \|\mathbf{v}_i\|$ do
10:         $L \leftarrow L \setminus \{\mathbf{v}_i\}$
11:         S.push($\mathbf{v}_i \pm \mathbf{p}$)
12:     end while
13:     return $\mathbf{p}$
14: end function
```

For our purposes we will use the GaussSieve algorithm 3.1 but with the modified version of the GaussReduce function 3.2. In this way the following conditions are met.

(i) Any irreducible vector which has been added to the GaussSieve list L will never be removed from it.

(ii) Any irreducible vector encountered by the algorithm will be added to L provided that it can extend its class representative set already in L.

Lemma 3.45. *The* GaussSieve *algorithm 3.1 equipped with the function PrimeGaussReduce (Algorithm 3.2) satisfies both properties (i) and (ii).*

Proof. (Property *i*) The only way for a vector $\mathbf{v}_i \in L$ to be removed from the list L is by entering the **while** loop in line 9 of the PrimeGaussReduce function. Let $\mathbf{v}_i \in L$ and also $\mathbf{v}_i \in \mathrm{Irr}(\mathcal{L})$. In order for the algorithm to remove $\mathbf{v}_i$ from L it should encounter another vector $\mathbf{p}$ such that $\|\mathbf{v}_i\| > \|\mathbf{p}\|$ and $\|\mathbf{v}_i - \mathbf{p}\| < \|\mathbf{v}_i\|$ or $\|\mathbf{v}_i\| > \|-\mathbf{p}\|$ and $\|\mathbf{v}_i + \mathbf{p}\| < \|\mathbf{v}_i\|$ which contradicts the irreducibility of $\mathbf{v}$.
(Property *ii*) Assume that the function PrimeGaussReduce is called and in some iteration of the **while** loop in line 2, $\mathbf{p}$ becomes such that $\mathbf{p} \in \mathrm{Irr}(\mathcal{L})$. In order for $\mathbf{p}$ to not be added in L this would mean that $\mathbf{p}$ could be further modified by the **while** loop in line 2. Thus the algorithm should encounter another vector $\mathbf{v}_i \in L$ such that $\|\mathbf{v}_i\| \leqslant \|\mathbf{p}\|$ and $\|\mathbf{p} \pm \mathbf{v}_i\| < \|\mathbf{p}\|$. The case where $\|\mathbf{v}_i\| < \|\mathbf{p}\|$ and $\|\mathbf{p} \pm \mathbf{v}_i\| < \|\mathbf{p}\|$ violates the irreducibility of $\mathbf{p}$ and thus can be disregarded. This leaves only one possible case, namely $\|\mathbf{v}_i\| = \|\mathbf{p}\|$ and $\|\mathbf{p} \pm \mathbf{v}_i\| < \|\mathbf{p}\|$. This condition implies that $\mathbf{v}_i$ and $\mathbf{p}$ belong to the same equivalence class and they are adjacent. Therefore this pair of vectors cannot belong to any set $P(\mathcal{L})$ of $\mathcal{L}$. Hence $\mathbf{p}$ should not be included in L anyway and the algorithm correctly further reduces it. □

Remark 3.46. *If the PrimeGaussReduce function in line 9 was the same as in the original GaussReduce, then the algorithm could encounter an instance where it would enter the loop with* $\|\mathbf{v}_i\| > \|\mathbf{p}\|$, $\|\mathbf{v}_i - \mathbf{p}\| = \|\mathbf{v}_i\|$ *and* $\mathbf{v}_i, \mathbf{v}_i - \mathbf{p} \in \mathrm{Irr}(\mathcal{L})$. *This could be possible if an equivalence class in* $\mathrm{Irr}(\mathcal{L})$ *was not trivial. In this case the algorithm would remove the vector* $\mathbf{v}_i$ *from the list and add its equivalent* $\mathbf{v}_i - \mathbf{p}$ *to S. As a result for these non-trivial classes the algorithm could behave in a bad way by repetitively removing and adding representatives of the same class.*

Remark 3.47. *If the PrimeGaussReduce function in line 2 was the same as in the original GaussReduce, then the algorithm could encounter an instance where it would enter the loop with* $\|\mathbf{v}_i\| \leqslant \|\mathbf{p}\|$, $\|\mathbf{p} - \mathbf{v}_i\| = \|\mathbf{p}\|$ *and* $\mathbf{p}, \mathbf{p} - \mathbf{v}_i \in \mathrm{Irr}(\mathcal{L})$. *Thus,* $\mathbf{p}$ *and* $\mathbf{p} - \mathbf{v}_i$ *are equivalent. In case* $\|\mathbf{v}_i\| < \|\mathbf{p}\|$ *then* $\mathbf{p}$ *and* $\mathbf{p} - \mathbf{v}_i$ *are also adjacent in the class graph and therefore in this case the algorithm would cycle through the adjacent vectors of* $\mathbf{p}$. *Therefore there is no need to perform a reduction in this case. In case* $\|\mathbf{v}_i\| = \|\mathbf{p}\|$ *then all three* $\mathbf{p}, \mathbf{v}_i, \mathbf{p} - \mathbf{v}_i$ *are equivalent but not adjacent. Hence in this case the algorithm does not make any progress by replacing* $\mathbf{p}$ *by* $\mathbf{p} - \mathbf{v}_i$. *Thus, there is no need to perform a reduction in this case as well.* [2]

[1] This version is the one used in [The19a] as well.

[2] However, as $\mathbf{p} - \mathbf{v}_i$ is not adjacent to both $\mathbf{p}$ and $\mathbf{v}_i$ an option could be to move $\mathbf{p} - \mathbf{v}_i$ to the stack S for further consideration later.

We consider the GaussSieve algorithm 3.1 equipped with the PrimeGaussReduce function (Algorithm 3.2). We denote by GaussSieve$(\mathcal{L})$ a list of vectors L created by this algorithm and possessing the property that L cannot be further modified by the algorithm. In order to relate the sets GaussSieve$(\mathcal{L})$ and $P(\mathcal{L})$ we give the following definition.

Definition 3.48. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. Given a $P(\mathcal{L}) \subseteq \mathrm{Irr}(\mathcal{L})$ we define $P^+(\mathcal{L})$ and $P^-(\mathcal{L})$ to be a partition of $P(\mathcal{L})$ according to sign.*

In other words, we take for $P^+(\mathcal{L})$ some subset of $P(\mathcal{L})$ such that of each pair $\pm\mathbf{v} \in P^+(\mathcal{L})$ exactly one is in $P^+(\mathcal{L})$. Of course, there are many choices for $P^+(\mathcal{L})$ and $P^-(\mathcal{L})$, any one will do.

Even though the output of GaussSieve converges to a set which is maximal in $\mathcal{L}$ under the property $\min\{\|\mathbf{v}_1 \pm \mathbf{v}_2\|\} \geqslant \max\{\|\mathbf{v}_1\|, \|\mathbf{v}_2\|\}$, the same is not true in general for the set $P^+(\mathcal{L})$ as shown by experiments (Table 3.1). In particular, we can conclude by Lemma 3.45 that if we allow this modified version of the GaussSieve to run long enough i.e. it samples “enough” vectors, then the output will converge to a set GaussSieve$(\mathcal{L})$, which will contain a $P^+(\mathcal{L})$.

Hence we cannot claim that the output of GaussSieve converges to a set $P^+(\mathcal{L})$ but only to a superset of it. The fact that a $P^+(\mathcal{L})$ is not maximal in $\mathcal{L}$ under the property $\min\{\|\mathbf{v}_1 \pm \mathbf{v}_2\|\} \geqslant \max\{\|\mathbf{v}_1\|, \|\mathbf{v}_2\|\}$ implies the existence of vectors which are not irreducible but they also cannot be reduced by any of the vectors in $P(\mathcal{L})$.

The definition of the set $P_2(\mathcal{L})$ will help us in bounding the output of the GaussSieve algorithm. Also, the definition of the sets $P_k(\mathcal{L})$ for $k > 2$ will help us in bounding the output of modified versions of “higher” sieving algorithms like the Triple and Quadruple MinkowskiSieve, described in [BLS16].

Definition 3.49. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. Given a $P(\mathcal{L}) \subseteq \mathrm{Irr}(\mathcal{L})$ we define*

$$P_2(\mathcal{L}) := \{\mathbf{v} \in \mathcal{L} \mid \nexists \mathbf{p} \in P(\mathcal{L}) \text{ with } \|\mathbf{p}\| < \|\mathbf{v}\| \text{ and } \|\mathbf{v} - \mathbf{p}\| < \|\mathbf{v}\|\}.$$

A first remark on this definition is that as $P(\mathcal{L}) \subseteq \mathcal{L}$ also $P(\mathcal{L}) \subseteq P_2(\mathcal{L})$. The output of the (modified) GaussSieve converges to a set GaussSieve$(\mathcal{L})$ including a set $P^+(\mathcal{L})$. Therefore, every $\mathbf{v} \in$ GaussSieve$(\mathcal{L})$ cannot be reduced by any $\mathbf{p} \in P^+(\mathcal{L})$ and as GaussSieve$(\mathcal{L}) \subseteq \mathcal{L}$ we can conclude that GaussSieve$(\mathcal{L})$ can be bounded as follows:

$$P^+(\mathcal{L}) \subseteq \mathrm{GaussSieve}(\mathcal{L}) \subseteq P_2(\mathcal{L}) \tag{3.11}$$

Under this set inequality GaussSieve$(\mathcal{L})$ can be viewed in the following way. A set GaussSieve$(\mathcal{L})$ can be considered as the closure of a $P^+(\mathcal{L})$ in $P_2(\mathcal{L})$ under the property of Gauss-reduction. In more detail GaussSieve$(\mathcal{L})$ can be viewed as the minimal (according to included vector norms) subset of a $P_2(\mathcal{L})$ including $P^+(\mathcal{L})$ and being a maximal subset of $P_2(\mathcal{L})$ with the property of Gauss-reduction (i.e. $\min\{\|\mathbf{v}_1 \pm \mathbf{v}_2\|\} \geqslant \max\{\|\mathbf{v}_1\|, \|\mathbf{v}_2\|\}$).

Remark 3.50. *It is unclear if the set $P_2(\mathcal{L})$ is a finite or an infinite set. Although we would like to point out one remark that we made in the case that $P_2(\mathcal{L})$ was infinite. If the set $P_2(\mathcal{L})$ contained arbitrarily long lattice vectors, then these vectors would get arbitrarily close to being orthogonal to the entire set $P(\mathcal{L})$.*

Definition 3.51. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$ and $k \in \mathbb{N}$ with $k \geqslant 2$. Given a $P(\mathcal{L}) \subseteq \mathrm{Irr}(\mathcal{L})$ we define*

$$P_{k+1}(\mathcal{L}) \coloneqq \{\boldsymbol{v} \in P_k(\mathcal{L}) \,|\, \nexists \mathbf{p} \in P^{(k)}(\mathcal{L}) \text{ with } \|\mathbf{p}\| < \|\boldsymbol{v}\| \text{ and } \|\boldsymbol{v} - \mathbf{p}\| < \|\boldsymbol{v}\|\}$$

where $P^{(k)}(\mathcal{L})$ is defined as

$$\bigcup_{i=1}^{\lfloor k/2 \rfloor} \{\boldsymbol{v}_1 + \boldsymbol{v}_2 \,|\, \boldsymbol{v}_1 \in P^{(i)}(\mathcal{L}),\, \boldsymbol{v}_2 \in P^{(k-i)}(\mathcal{L}) \text{ and } \|\boldsymbol{v}_j\| < \|\boldsymbol{v}_1 + \boldsymbol{v}_2\|,$$
$$\|\boldsymbol{v}_l\| \leqslant \|\boldsymbol{v}_1 + \boldsymbol{v}_2\| \text{ where } (j, l) \in (1, 2), (2, 1)\}$$

and $P^{(1)}(\mathcal{L}) \coloneqq P(\mathcal{L})$.

Lemma 3.52. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$ and $P(\mathcal{L})$ be a subset of $\mathrm{Irr}(\mathcal{L})$. Then for the sequence $P_k(\mathcal{L})$ given in definition 3.51 it holds that*

(i) $P_{k+1}(\mathcal{L}) \subseteq P_k(\mathcal{L})$ *for every* $k \geqslant 2$.

(ii) $\lim_{k\to\infty} P_k(\mathcal{L}) = \mathrm{Irr}(\mathcal{L})$.

So, in one line:

$$P_2(\mathcal{L}) \supseteq \ldots \supseteq P_k(\mathcal{L}) \supseteq P_{k+1}(\mathcal{L}) \supseteq \ldots \supseteq \mathrm{Irr}(\mathcal{L}).$$

Proof. First of all, as we chose a random but fixed $P(\mathcal{L}) \subseteq \mathrm{Irr}(\mathcal{L})$ the sets $P_k(\mathcal{L})$ are well-defined. Part *(i)* of the lemma is an immediate consequence of the $P_k(\mathcal{L})$ definition. Initially we show that $\mathrm{Irr}(\mathcal{L}) \subseteq P_k(\mathcal{L})$ for every $k \geqslant 2$. This follows directly by the definition of $P_k(\mathcal{L})$ and the fact that $P^{(k)}(\mathcal{L}) \subseteq \mathcal{L}$. By the (recursive) definition of $P_k(\mathcal{L})$ it follows that it includes all vectors $\boldsymbol{v} \in \mathcal{L}$ such that they can not be reduced by any shorter vector in $\cup_{i=1}^{k-1} P^{(i)}(\mathcal{L})$. Thus for part *(iii)* of the lemma it suffices to show that $\lim_{k\to\infty} \cup_{i=1}^{k} P^{(i)}(\mathcal{L}) = \mathcal{L}$. As $P^{(i)}(\mathcal{L}) \subseteq \mathcal{L}$ for every $i \geqslant 1$ it follows that $\lim_{k\to\infty} \cup_{i=1}^{k} P^{(i)}(\mathcal{L}) \subseteq \mathcal{L}$. It is only left proving the converse inequality. Let $\boldsymbol{v} \in \mathcal{L}$, it suffices to show that $\exists k \geqslant 1$ such that $\boldsymbol{v} \in P^{(k)}(\mathcal{L})$.

A vector $\boldsymbol{v} \in \mathcal{L}$ can be repeatedly reduced as in the proof of Proposition 3.22 until it is written as a sum $\boldsymbol{v} = \sum_{i=1}^{l} \mathbf{p}_i$ of shorter vectors $\mathbf{p}_i \in \mathrm{Irr}(\mathcal{L})$ for some $l \geqslant 1$. This decomposition satisfies the recursive condition implied by the definition of the $P^{(k)}(\mathcal{L})$. If all the vectors $\mathbf{p}_i \in \mathrm{Irr}(\mathcal{L})$ actually belong to $P(\mathcal{L})$ then $\boldsymbol{v} \in P^{(l)}(\mathcal{L})$ and we are done. If there exists some $\mathbf{p}_i \in \mathrm{Irr}(\mathcal{L}) \setminus P(\mathcal{L})$ then $\mathbf{p}_i = \tilde{\mathbf{p}}_i + \mathbf{p}'_i$ where $\tilde{\mathbf{p}}_i \in P(\mathcal{L})$ and $\|\mathbf{p}'_i\| < \|\mathbf{p}_i\|$, $\|\mathbf{p}_i\| = \|\tilde{\mathbf{p}}_i\|$ by the definition of $P(\mathcal{L})$. Thus, $\mathbf{p}'_i$ can be further get decomposed into shorter vectors (like $\boldsymbol{v}$) and as $\|\mathbf{p}'_i\| < \|\mathbf{p}_i\|$ progress was made which implies that this decomposition will finish after finitely many steps as there is a finite number of lattice points in $\mathcal{B}(\mathbf{0}, \|\boldsymbol{v}\|)$. Therefore $\boldsymbol{v}$ can be repeatedly reduced until it is written as a sum of vectors in $P(\mathcal{L})$, concluding the proof. $\square$

We are now going to describe the "higher" sieving algorithms which we will consider. We have already mentioned the Triple and the Quadruple MinkowskiSieve described in [BLS16]. The difference between the GaussSieve algorithm and these higher ones lies in the reduction function. Hence, if we equip Algorithm 3.1 with function PrimeMinkowskiReduce (Algorithm 3.3), we get the modified MinkowskiSieve which we are interested in.

Algorithm 3.3 The modified MinkowskiReduce function

```
 1: function PRIMEMINKOWSKIREDUCE($p, L, S, k$)
 2:     loop = true
 3:     while loop do
 4:         loop = false
 5:         if $k > 2$ then
 6:             PRIMEMINKOWSKIREDUCE($p, L, S, k-1$)
 7:         end if
 8:         for all $\{v_1, \dots, v_{k-1}\} \subset L$ s.t. $\|v_i\| \leqslant \|p\|$ do
 9:             for all $w \in \left\{\sum_{i=1}^{k-1} (-1)^{a_i} v_i \mid a_i \in \{0,1\}\right\}$ do
10:                 if $\|w\| \leqslant \|p\|$ and $\|p - w\| < \|p\|$ then
11:                     $p \leftarrow p - w$
12:                     loop = true
13:                     goto next
14:                 end if
15:             end for
16:         end for
17:         next:
18:     end while
19:     for all $\{v_1, \dots, v_{k-1}\} \subset L$ with $\|v_i\| \leqslant \|v_{i+1}\|$ and s.t. $\|v_{k-1}\| > \|p\|$ do
20:         for all $w \in \left\{(-1)^{a_0} p + \sum_{i=1}^{k-2} (-1)^{a_i} v_i \mid a_i \in \{0,1\}\right\}$ do
21:             if $\|w\| < \|v_{k-1}\|$ and $\|v_{k-1} - w\| < \|v_{k-1}\|$ then
22:                 $L \leftarrow L \setminus \{v_{k-1}\}$
23:                 S.push($v_{k-1} - w$)
24:             end if
25:         end for
26:     end for
27:     return p
28: end function
```

The modification compared to the description in [BLS16] appears in lines 10 and 21 of Algorithm 3.3, where the extra conditions $\|w\| \leqslant \|p\|$ and $\|w\| < \|v_{k-1}\|$ respectively are added. By adding these conditions it is guaranteed to get an output list which will satisfy properties (i) and (ii) like in Lemma 3.45 for the GaussSieve. Hence, based on these properties it can be concluded that the output list of vectors will again contain a set $P^+(\mathcal{L})$. In order to ease our exposition we set the following notation.

Let $k \in \mathbb{N}$ with $k \geqslant 2$. We consider the k-MinkowskiSieve algorithm equipped with the function PrimeMinkowskiReduce (Algorithm 3.3). We denote by $\text{MinkowskiSieve}_k(\mathcal{L})$ a list of vectors L created by this algorithm and possessing the property that L cannot be further modified by the algorithm. Note that for $k = 2$ one has $\text{MinkowskiSieve}_2(\mathcal{L}) = \text{GaussSieve}(\mathcal{L})$.

Remark 3.53. *The output of the modified* k-MinkowskiSieve *algorithm will not be a list of vectors which will be* k*-Minkowski-reduced if* $k > 2$ *(for the Minkowski-reduced definition see [NS09]). If this was desired, then the lines 10 and 21 of Algorithm 3.3 should be modified* *in order to allow reductions by longer vectors as well. For a k-Minkowski-reduced list with $k > 4$ lines 9,10 and 20,21 of Algorithm 3.3 should also allow for the coefficients of the vectors $\boldsymbol{v}_i$, $\mathbf{p}$ and $\boldsymbol{v}_{k-1}$ to take more values than ± 1 (see for example [NS09, Theorem 2.2.2]).*

The "higher" sieving algorithms which we considered by making the generalisation from the GaussSieve towards the MinkowskiSieve will contribute towards an asymptotic computational argument. But first we state a heuristic assumption which we will use.

Assumption 3.54. *Consider the* k-MinkowskiSieve *algorithm equipped with the function PrimeMinkowskiReduce (Algorithm 3.3). Then the output of this algorithm will converge to a set* $\text{MinkowskiSieve}_k(\mathcal{L})$.

Remark 3.55. *Heuristic assumption 3.54 actually claims that the* k-MinkowskiSieve *does not diverge or enter an infinite loop. The experimental results in section 3.4.3 (see Figure 3.3) indicate that for $k \in \{2, 3, 4\}$ this seems to be a valid assumption. However, this is the only argument we have in favour of this assumption. We leave the investigation for concrete arguments supporting this heuristic assumption as an open problem for future research.*

Theorem 3.56. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. We consider the* k-MinkowskiSieve *algorithm equipped with the function PrimeMinkowskiReduce. Under Heuristic Assumption 3.54, as k increases the set* $\text{MinkowskiSieve}_k(\mathcal{L})$ *converges to a set* $P^+(\mathcal{L})$.

Proof. In order to simplify the proof and avoid ambiguities we make the following convention. Both sets $P^+(\mathcal{L})$ and $\text{MinkowskiSieve}_k(\mathcal{L})$ are defined/constructed in such a way that for a vector $\boldsymbol{v}$ only one of $\pm\boldsymbol{v}$ belongs to the set. This allows many possible choices for these sets. In order to avoid this kind of ambiguities we make the convention that a vector $\boldsymbol{v}$ is included in the aforementioned sets only if its first non-zero coordinate is positive.

Initially we will prove that for every $k \geqslant 2$ there exists a set $P^+(\mathcal{L})$ and a set $P_k(\mathcal{L})$ such that

$$P^+(\mathcal{L}) \subseteq \text{MinkowskiSieve}_k(\mathcal{L}) \subseteq P_k(\mathcal{L}). \tag{3.12}$$

Let $k \geqslant 2$ and $\text{MinkowskiSieve}_k(\mathcal{L})$ be the converging set of an execution of the k-MinkowskiSieve. As mentioned before, we can transfer Lemma 3.45 from the case of GaussSieve to the k-MinkowskiSieve algorithm described in this section. This implies that for every $\text{MinkowskiSieve}_k(\mathcal{L})$ there will exist a set $P^+(\mathcal{L})$ such that
$P^+(\mathcal{L}) \subseteq \text{MinkowskiSieve}_k(\mathcal{L})$. We fix this set $P^+(\mathcal{L})$.

Let $\boldsymbol{v} \in \text{MinkowskiSieve}_k(\mathcal{L})$ and $\mathbf{p}_1, \dots, \mathbf{p}_{k-1} \in P^+(\mathcal{L})$ with $\|\mathbf{p}_i\| < \|\boldsymbol{v}\|$. As the set $\text{MinkowskiSieve}_k(\mathcal{L})$ is k-reduced according to the notion implied by Algorithm 3.3 we can conclude that $\boldsymbol{v}$ cannot be reduced by any vector of the form $\sum_{i=1}^{l} (-1)^{a_i} \mathbf{p}_i$ for $1 \leqslant l \leqslant k-1$. As the vectors $\mathbf{p}_1, \dots, \mathbf{p}_{k-1}$ belong to the set $\text{MinkowskiSieve}_k(\mathcal{L})$ as well, they are $k-1$-reduced. This in turn implies that the vectors of the form $\sum_{i=1}^{l} (-1)^{a_i} \mathbf{p}_i$ belong to the set $P^{(l)}(\mathcal{L})$ for $1 \leqslant l \leqslant k-1$. This holds for any tuple of l vectors in $P^+(\mathcal{L})$. Hence, the set of vectors emerging from the union of all $\{\sum_{i=1}^{l} (-1)^{a_i} \mathbf{p}_i\}$ will be exactly $P^{(l)}(\mathcal{L})$. This implies that $\boldsymbol{v}$ cannot be reduced by any vector in $\cup_{i=1}^{k-1} P^{(i)}(\mathcal{L})$. This is equivalent to the condition a vector $\boldsymbol{v}$ has to satisfy according to definition 3.51 in order to belong to $P_k(\mathcal{L})$. Thus we can conclude that $\text{MinkowskiSieve}_k(\mathcal{L})$ is included in the

$P_k(\mathcal{L})$ implied by the set $P(\mathcal{L}) = P^+(\mathcal{L}) \cup (-P^+(\mathcal{L}))$. This concludes the first part of the proof.

For the second part of the proof we distinguish between the cases of $\mathrm{Irr}(\mathcal{L}) = P(\mathcal{L})$ and $\mathrm{Irr}(\mathcal{L}) \neq P(\mathcal{L})$.
If it holds that $\mathrm{Irr}(\mathcal{L}) = P(\mathcal{L})$ then apart from $P(\mathcal{L})$ being uniquely determined the same holds for the sets $P_k(\mathcal{L})$. Hence, for every $k \geqslant 2$ the boundary sets in (3.12) are uniquely determined. This enables a direct use of Lemma 3.52. As k increases the set $\mathrm{MinkowskiSieve}_k(\mathcal{L})$ will be contained in even smaller and smaller sets $P_k(\mathcal{L})$ which converge to $\mathrm{Irr}(\mathcal{L})$ according to (i) and (iii) of Lemma 3.52. Therefore for the limit case it could be stated that

$$P^+(\mathcal{L}) \subseteq \lim_{k\to\infty} \mathrm{MinkowskiSieve}_k(\mathcal{L}) \subseteq \mathrm{Irr}(\mathcal{L}). \tag{3.13}$$

But we assumed $\mathrm{Irr}(\mathcal{L}) = P(\mathcal{L})$ and thus we can conclude that

$$\lim_{k\to\infty} \mathrm{MinkowskiSieve}_k(\mathcal{L}) = P^+(\mathcal{L}).$$

In order to finish the proof we have to deal with the case $\mathrm{Irr}(\mathcal{L}) \neq P(\mathcal{L})$. In this case, the sets $P^+(\mathcal{L})$ and $P_k(\mathcal{L})$ used in inequality (3.12) are not uniquely determined and therefore Lemma 3.52 cannot be used directly. In Lemma 3.52 it was shown that given the sequence of $P_k(\mathcal{L})$ implied by any $P(\mathcal{L})$ then $\lim_{k\to\infty} P_k(\mathcal{L}) = \mathrm{Irr}(\mathcal{L})$. Hence any $P_k(\mathcal{L})$ belongs to a sequence converging to the same limit, $\mathrm{Irr}(\mathcal{L})$. Interchanging terms $(P_k(\mathcal{L}))$ among these sequences does not affect their limit. Therefore, we can again use inequality (3.12) and take limits leading to a result like (3.13). We have to be careful though. The right hand-side limit (i.e. $\mathrm{Irr}(\mathcal{L})$) is well-defined but the left one can cycle over all choices of $P^+(\mathcal{L})$. This is expected as the limit of the sequence $\mathrm{MinkowskiSieve}_k(\mathcal{L})$ as $k \to \infty$ is not unique but depends on the choice of representatives made for each non-trivial class of vectors. For convenience we assume that $\forall k > k_0$ for some k_0 this choice stabilises to some random but fixed choice. Thus, we have again reached inequality (3.13).

We examine the sets in inequality (3.13) according to the Gauss-reduced property. Let $k \geqslant 2$, the set $\mathrm{MinkowskiSieve}_k(\mathcal{L})$ is a set to which the output of the algorithm converges to and also possesses the Gauss-reduced property by construction. This holds for every $k \geqslant 2$ and thus transfers to the limit as well, as $k \to \infty$. The set $P^+(\mathcal{L})$ is not a maximal subset of $\mathcal{L}$ satisfying the Gauss-reduced property but due to its construction it is maximal in the set $\mathrm{Irr}(\mathcal{L})$. Hence, inequality (3.13) and maximality of $P^+(\mathcal{L})$ in $\mathrm{Irr}(\mathcal{L})$ imply the result. □

The conclusion in Theorem 3.56 is supported by the experimental results given in Table 3.1.

Remark 3.57. *Theorem 3.56 describes asymptotic behaviour of the modified* MinkowskiSieve *algorithm with the goal of providing a faster way of computing sets $P^+(\mathcal{L})$. Even though, asymptotically, the algorithm possesses the desired behaviour, this does not make it immediately a computational tool for $P^+(\mathcal{L})$. There are two obstacles towards that goal. The first one is, given a lattice $\mathcal{L}$ in dimension n, to find for which $k \geqslant 2$ to run* k-MinkowskiSieve. *This k should not be too high in order to be computationally efficient to run the algorithm. The second problem is finding for how long this* k-MinkowskiSieve *should run in order to approximate well enough a set* $\mathrm{MinkowskiSieve}_k(\mathcal{L})$.

3.4.3 – Experimental results. In this section we provide some experimental results which support our claims in the previous subsections. In particular, as a first step we computed the sets $R(\mathcal{L}), \mathrm{Irr}(\mathcal{L}), P(\mathcal{L})$ for 10 lattices in dimension 20 and afterwards we computed the output of the GaussSieve, the Triple and the Quadruple MinkowskiSieve. In order to generate 10 lattices in dimension 20 we used the Sage computer algebra system [TSD19]. In particular we used Sage's "Hard lattice generator" with the following choice of parameters,

sage.crypto.gen_lattice(type='random', n=1, m=20, q=10^42, seed=*seed*)

and 10 different values of *seed*. Initially, using the OpenMP parallel implementation build for the projects [FCMP19, CMF19] we computed the set of relevant vectors $R(\mathcal{L})$ for each lattice. On top of this code (which the authors were so kind to provide us) we implemented the method described in Section 3.4.1 and computed the set $\mathrm{Irr}(\mathcal{L})$. As for our experiments the lattices used were generated randomly, they did not possess any specific structure and hence $P(\mathcal{L}) = \mathrm{Irr}(\mathcal{L})$ for all of them. This part of the experiments was performed on a node of the Lisa cluster [SUR19] with a 16-core CPU (2.10GHz) and 96 GB of RAM. The computation of the sets $R(\mathcal{L})$ and $\mathrm{Irr}(\mathcal{L})$ using the aforementioned implementation and hardware took about 5.5 seconds per lattice.

Finally, by modifying the already existing sieve implementations in FPLLL [The19a] we computed the output of the GaussSieve, Triple and Quadruple MinkowskiSieve as described in Section 3.4.2 for the same 10 lattices. The modifications which we made to the already existing FPLLL implementations were:

- A vector is allowed to be reduced only by a shorter vector.
- The termination condition is changed to a fixed number of collisions: $5 \cdot 10^5$ for the GaussSieve and 10^5 for the Triple and Quadruple MinkowskiSieve. These numbers were chosen to ensure the created list by the algorithm remains unchanged for "many" iterations before the algorithm terminates. These choices seem to not be optimal according to our experimental data and could possibly be further reduced.

This part of the experiments was performed on a Lenovo X250 laptop with 4 Intel Core i3-5010U CPU (2.10GHz) and 8 GB of RAM. The output of these experiments is summarised in Table 3.1.

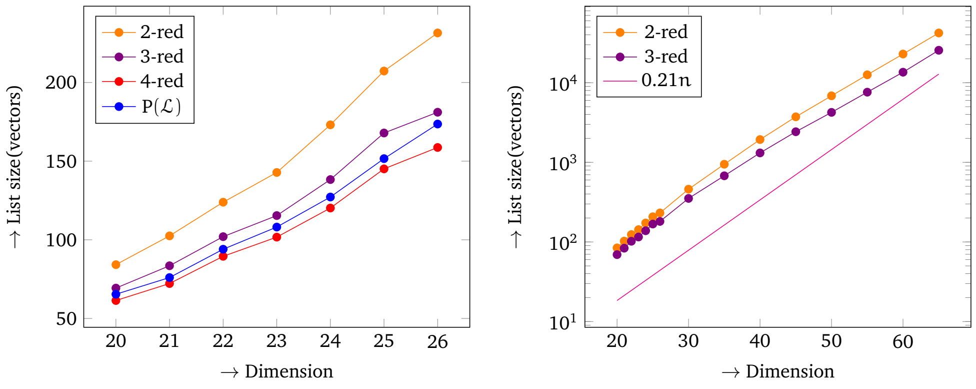
Table 3.1 motivates a number of remarks about the involved sets. Initially, the number of relevant vectors observed was indeed close to the expected number $2 \cdot (2^{20} - 1)$. Also, the sets $\mathrm{Irr}(\mathcal{L})$ and $P(\mathcal{L})$ were equal in all 10 cases, as we had assumed for random lattices without any underlying structure. The size of $P(\mathcal{L})$ (and $\mathrm{Irr}(\mathcal{L})$ in this case) was observed to be some orders of magnitude smaller than the size of $R(\mathcal{L})$ making it more appealing to use in practice.

The right part of Table 3.1 justifies our idea to try and correlate the output of sieving algorithms with the set of irreducible vectors. Even though we cannot display here the lists of vectors which we computed but rather only their sizes, we observed the following behaviour. The list of vectors output by the GaussSieve contained the set $P(\mathcal{L})$ in 8 out of the 10 cases and in the other two of them there was only 1 vector missing. This supports our claim the output of GaussSieve converges to a superset of $P(\mathcal{L})$. Also, as we moved to "higher" sieving algorithms like our modified version of the Triple and Quadruple MinkowskiSieve the output of the sieving algorithms approximated even closer the set $P(\mathcal{L})$. Actually, it is not a coincidence that the numbers in the columns "4-red" and

Table 3.1: The following tables describe the sizes of the lists involved in our experiments with 10 random lattices in dimension 20. The first columns indicate the seed used for the generation of the lattice. The table on the left gives the sizes of the corresponding sets $R(\mathcal{L})$, $\mathrm{Irr}(\mathcal{L})$ and $P(\mathcal{L})$ for each lattice. The factor 2 is due to the sign symmetry. The table on the right shows the sizes of the lists generated by the modified GaussSieve, Triple and Quadruple MinkowskiSieve.

Seed	$\lvert R(\mathcal{L})\rvert$	$\lvert \mathrm{Irr}(\mathcal{L})\rvert, \lvert P(\mathcal{L})\rvert$
314	$2 \cdot 1048361$	$2 \cdot 66$
417	$2 \cdot 1048388$	$2 \cdot 70$
849	$2 \cdot 1048389$	$2 \cdot 68$
422	$2 \cdot 1048349$	$2 \cdot 67$
168	$2 \cdot 1048371$	$2 \cdot 60$
84	$2 \cdot 1048363$	$2 \cdot 64$
105	$2 \cdot 1048375$	$2 \cdot 62$
273	$2 \cdot 1048360$	$2 \cdot 60$
390	$2 \cdot 1048376$	$2 \cdot 66$
656	$2 \cdot 1048372$	$2 \cdot 71$

Seed	2-red	3-red	4-red
314	86	77	66
417	95	80	70
849	98	85	68
422	93	74	67
168	88	69	60
84	92	75	64
105	88	74	62
273	83	68	60
390	89	76	66
656	95	79	71



(a) List sizes for dimensions 20–26.

(b) List sizes for dimensions 20–65.

Figure 3.3: Experimental results on the scaling of size of $P(\mathcal{L})$ according to the dimension of $\mathcal{L}$. Each point in the graphs corresponds to the average value taken amongst 10 lattices. The labels k-red are used to indicate the modified sieve algorithms described in this chapter and not the ones in the literature [MV10b, BLS16].

"$\lvert \mathrm{Irr}(\mathcal{L})\rvert, \lvert P(\mathcal{L})\rvert$" in Table 3.1 differ only by a factor of 2. The output of the Quadruple MinkowskiSieve in all 10 cases gave exactly a set $P^+(\mathcal{L})$ as for every vector $\boldsymbol{v}$ it stores only one of $\pm\boldsymbol{v}$.

Another question which could be investigated experimentally is how the expected size of $P(\mathcal{L})$ behaves as the dimension of $\mathcal{L}$ increases. In order to develop an intuition about this behaviour we performed a number of experiments in dimensions 20–65, the results of which are shown in Figure 3.3. Like in our experiments in dimension 20 we used the modified OpenMP parallel implementation from [FCMP19, CMF19] and the modified sieve implementations in FPLLL [The19a]. For each dimension we depict the average value amongst 10 lattices. However, as in this case we dealt with higher dimensions we reduced the number of collisions in the termination condition of the sieve algorithms to

• GaussSieve:	10,000 collisions
• Triple MinkowskiSieve:	2,500 collisions
• Quadruple MinkowskiSieve:	2,000 collisions.

Therefore the results in Figure 3.3 related to sieving algorithms should only be interpreted as approximations of the algorithm's converging set size. As we will discuss later, estimating the accuracy of this approximation is left for future research. Figure 3.3a illustrates the result of our experiments in dimensions 20-26. We believe that for these "smaller" dimensions the approximations are "more" accurate and that is why we show them separately. Another reason is that running the OpenMP Voronoi implementation beyond these dimensions has a substantial memory requirement (tens of GB).

Computing a least squares fit for the points in the blue curve (which indicates the correct expected values for $|\mathrm{P}(\mathcal{L})|$ under assumption 3.43) gives the formula $2^{0.237n+1.286}$ which closely matches the heuristic expectation for the size of $\mathrm{P}(\mathcal{L})$, namely $2^{0.21n}$. Furthermore Figure 3.3a reveals that the GaussSieve gives only a superset of $\mathrm{P}(\mathcal{L})$ even for small dimensions. The Triple and Quadruple MinkowskiSieve are much closer to the blue curve. The difference between the Triple and Quadruple MinkowskiSieve is that the one lies above the blue curve and the other below it. As we already observed in Table 3.1 the Triple MinkowskiSieve will probably remain above it. However the Quadruple MinkowskiSieve possess the potential to reach the "correct" curve asymptotically. Of course this could also be far from the truth for higher dimensions.

In order to put these curves more into perspective we created Figure 3.3b which shows the average output sizes of the GaussSieve and Triple MinkowskiSieve for dimensions 20–65. We did not draw the curve of the Quadruple MinkowskiSieve as it also turns out to be quite time costly for dimensions higher than 30. At this point we must emphasise that the used modified sieving algorithms take more time in order to terminate due to the modifications which aim not in solving SVP but computing close approximations of $\mathrm{P}(\mathcal{L})$. For instance the modified Triple MinkowskiSieve in dimension 65 took on average 3 days in order to terminate for each lattice. However this is only the average observed time. Actually one of the ten lattices used proved to be an "easier case", terminating in under 2 hours.

Even though these results provide some intuition on what kind of relation could be expected between the set of irreducible vectors and sieving algorithms, they also imply some questions.

A first question which would be interesting is examining the termination condition for the sieving algorithm. In our experiments we made a specific choice on the number of collisions but this was done by trial and error and could be possibly improved. In other words, we ask for a termination condition, which if it is satisfied by a sieving algorithm (as used in this section) it guarantees that the algorithm has reached a list of vectors which cannot be further modified by the algorithm.

A second question that arises is up to what level of sieving we should get in order to either get exactly a set $\mathrm{P}(\mathcal{L})$ or a "very good" approximation of it. In this case the Quadruple MinkowskiSieve was enough, but this might not be the case for higher dimensional lattices. Thus it would be interesting to know how this index increases relative to the dimension. So, given some termination condition, how close can a sieving algorithm approximate a set $P(\mathcal{L})$?

If these questions receive an answer it will help in making sieving algorithms a way to either compute exactly or approximately a set $P(\mathcal{L})$ of a lattice $\mathcal{L}$. This would be very interesting as it will provide a way to compute a set $P(\mathcal{L})$ (exactly or approximately) without having to compute the set $R(\mathcal{L})$ which is a very costly computation.

3.5 — Applications of $P(\mathcal{L})$

Even though the sets $\mathrm{Irr}(\mathcal{L})$ and $P(\mathcal{L})$ might be of interest in their own, examining their relation to already existing lattice problems and algorithms is a natural question that arises. We choose to focus on the set $P(\mathcal{L})$ as it seems to be the easier to compute/approximate with existing lattice algorithms.

3.5.1 – $P(\mathcal{L})$ in the study of shortest vector(s) problems. The results in Section 3.3 provide some interesting conclusions about the relation of the set $P(\mathcal{L})$ to well known lattice problems. A first observation in Section 3.3.2 was that $S_1(\mathcal{L})$ is included in $P(\mathcal{L})$. This leads to the following result.

Proposition 3.58. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. Computing a set $P(\mathcal{L})$ provides a solution to the SVP and the kissing number problem.*

The relation $S_1(\mathcal{L}) \subseteq P(\mathcal{L})$, implies that two classic lattice problems can be solved given a $P(\mathcal{L})$. Of course this holds for any superset of $P(\mathcal{L})$ as well. We combine this observation with the inclusion $P^+(\mathcal{L}) \subseteq \mathrm{MinkowskiSieve}_k(\mathcal{L})$ for $k \geqslant 2$ shown in the proof of Theorem 3.56. This provides some extra (heuristic) evidence that some sieving algorithms will indeed output a solution to SVP or the kissing number problem if they run long enough. This is no surprise as sieving algorithms were devised for solving SVP.

Examining the relation of SIVP to the set $P(\mathcal{L})$ is probably a more interesting question. By Corollary 3.21 we know that for every $i = 1, \dots, n$ there exists a vector $\boldsymbol{v} \in \mathrm{Irr}(\mathcal{L})$ such that $\|\boldsymbol{v}\| = \lambda_i(\mathcal{L})$. The following proposition completes this result.

Proposition 3.59. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$. Computing a set $P(\mathcal{L})$ provides a solution to the SIVP.*

Proof. Let $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n$ be a set of linearly independent vectors in $\mathcal{L}$ such that $\|\boldsymbol{v}_i\| = \lambda_i(\mathcal{L})$ for $i = 1, \dots, n$. We distinguish two cases.
<u>Case 1:</u> $\nexists i \geqslant 2$ such that $\lambda_1(\mathcal{L}) \leqslant \lambda_{i-1}(\mathcal{L}) < \lambda_i(\mathcal{L}) = \lambda_{i+1}(\mathcal{L})$. This implies that there exists a $k \geqslant 1$ such that

$$\lambda_1(\mathcal{L}) = \cdots = \lambda_k(\mathcal{L}) < \lambda_{k+1}(\mathcal{L}) < \cdots < \lambda_n(\mathcal{L}).$$

Then by $S_1(\mathcal{L}) \subseteq P(\mathcal{L})$ it follows that $\boldsymbol{v}_1, \dots, \boldsymbol{v}_k$ belong to $P(\mathcal{L})$. In addition, by Corollary 3.21 and the definition of $P(\mathcal{L})$ it follows that all the $\boldsymbol{v}_{k+1}, \dots, \boldsymbol{v}_n$ will be included in $P(\mathcal{L})$.
<u>Case 2:</u> $\exists i \geqslant 2$ such that $\lambda_1(\mathcal{L}) \leqslant \lambda_{i-1}(\mathcal{L}) < \lambda_i(\mathcal{L}) = \lambda_{i+1}(\mathcal{L})$. Let $i \geqslant 2$ such that the condition holds. We set $k = \max\{j > i \,|\lambda_i(\mathcal{L}) = \lambda_j(\mathcal{L})\}$. Hence,

$$\lambda_1(\mathcal{L}) \leqslant \lambda_{i-1}(\mathcal{L}) < \lambda_i(\mathcal{L}) = \lambda_{i+1}(\mathcal{L}) = \cdots = \lambda_k(\mathcal{L}).$$

We will show that $\mathbf{v}_i, \ldots, \mathbf{v}_k \in P(\mathcal{L})$. Let $j \in \{i, \ldots, k\}$ we set $\mathcal{L}_{\lambda_j}$ to be the sublattice of $\mathcal{L}$ spanned by all the vectors in $\mathcal{L}$ strictly shorter than λ_j. As $\lambda_i(\mathcal{L}) = \lambda_j(\mathcal{L})$ it follows that $\mathcal{L}_{\lambda_j} = \mathcal{L}_{\lambda_i}$ which has rank $i-1$. Assume that $\mathbf{v}_j \in \mathcal{L}_{\lambda_j}$. Then we would get that the set $\{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}, \mathbf{v}_j\}$ is a set of linearly dependent vectors. Contradiction. Thus $\mathbf{v}_j \notin \mathcal{L}_{\lambda_j}$ and by Proposition 3.19 we get that $\mathbf{v}_j \in \mathrm{Irr}(\mathcal{L})$. This holds for any $i \leqslant j \leqslant k$ and therefore we get that all $\mathbf{v}_j$ with $i \leqslant j \leqslant k$ belong to $\mathrm{Irr}(\mathcal{L})$. In order to show that they also do belong to a $P(\mathcal{L})$ it suffices to show that for every μ, ν such that $i \leqslant \mu < \nu \leqslant k$ it holds that $\|\mathbf{v}_\nu - \mathbf{v}_\mu\| \geqslant \lambda_i(\mathcal{L})$. Assume that there exist μ, ν such that $i \leqslant \mu < \nu \leqslant k$ and $\|\mathbf{v}_\nu - \mathbf{v}_\mu\| < \lambda_i(\mathcal{L})$. Then it follows that $\mathbf{v}_\nu - \mathbf{v}_\mu \in \mathcal{L}_{\lambda_i}$. The set of vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}, \mathbf{v}_\mu, \mathbf{v}_\nu\}$ is a linearly independent set and thus the same holds for $\{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}, \mathbf{v}_\nu - \mathbf{v}_\mu\}$. This implies a set of i linearly independent vectors in the lattice $\mathcal{L}_{\lambda_i}$ which is of rank $i-1$, contradiction.
Concluding, let $\mathbf{v}_l$ belong to the considered linearly independent set of vectors achieving the successive minima. If $\|\mathbf{v}_l\| = \|\mathbf{v}_{l+1}\|$ or $\|\mathbf{v}_l\| = \|\mathbf{v}_{l-1}\|$ then $\mathbf{v}_l \in P(\mathcal{L})$ by the proof in "case 2". If $\|\mathbf{v}_{l-1}\| < \|\mathbf{v}_l\| < \|\mathbf{v}_{l+1}\|$ then $\mathbf{v}_l \in P(\mathcal{L})$ by the same argument used in "case 1". □

Remark 3.60. *Obtaining a set of the shortest vector(s), given a set* $P(\mathcal{L})$*, amounts to scanning the entire set* $P(\mathcal{L})$ *a number of times. Thus, sorting* $P(\mathcal{L})$ *can be avoided.*

3.5.2 – Using $P(\mathcal{L})$ in CVPP algorithms. One main problem in lattice theory is the closest vector problem. A straightforward way of using the set $R(\mathcal{L})$ in order to solve CVPP was described in [SFS09]. In that work, an algorithm called the iterative slicer is given which takes as input the set $R(\mathcal{L})$ and a target vector $\mathbf{t}$ and outputs a closest lattice vector to $\mathbf{t}$ (Algorithm 3.4). The main idea behind this algorithm is to iteratively reduce the target vector $\mathbf{t}$ by the relevant vectors until the resulting vector $\mathbf{t}'$ is contained in the Voronoi cell $\mathcal{V}(\mathcal{L})$ of the lattice. Once this condition is satisfied it is known that $\mathbf{t} - \mathbf{t}'$ is a closest lattice point to $\mathbf{t}$. This algorithm is shown to terminate after a finite number of iterations.

Algorithm 3.4 The iterative slicer [SFS09]

```
Require: The set $R(\mathcal{L})$ and a target vector $\mathbf{t}$.
Ensure: A vector $\mathbf{s} \in \mathcal{L}$ closest to $\mathbf{t}$.
1: $\mathbf{t}' \leftarrow \mathbf{t}$
2: for every $\mathbf{r} \in R(\mathcal{L})$ do
3:     if $\|\mathbf{t}' - \mathbf{r}\| < \|\mathbf{t}'\|$ then
4:         $\mathbf{t}' \leftarrow \mathbf{t}' - \mathbf{r}$
5:         restart the for loop
6:     end if
7: end for
8: $\mathbf{s} = \mathbf{t} - \mathbf{t}'$
9: return $\mathbf{s}$
```

Inspired by the iterative slicer, in [MV10a] an algorithm is described to provably solve the CVPP in $\tilde{O}(2^{2n})$–time by using the set $R(\mathcal{L})$ as the preprocessing data. The difference between Algorithm 3.4 and the algorithm in [MV10a] is that the latter selects the relevant vectors in a specific order for reduction. This results in a $\tilde{O}(2^{2n})$–time and $\tilde{O}(2^n)$–space

Algorithm 3.5 The tuple slicer

```
Require: A set $P(\mathcal{L})$, a $C \in \mathbb{N}$ and a target vector $\mathbf{t}$.
Ensure: A vector $\mathbf{s} \in \mathcal{L}$ closest to $\mathbf{t}$.
 1: $\mathbf{t}' \leftarrow \mathbf{t}$
 2: for $l = 1$ to $C$ do
 3:     for all $\{v_1, \dots, v_l\} \subset P(\mathcal{L})$ do
 4:         $w \leftarrow \sum_{i=1}^{l} v_i$
 5:         if $\|\mathbf{t}' - w\| < \|\mathbf{t}'\|$ then
 6:             $\mathbf{t}' \leftarrow \mathbf{t}' - w$
 7:             restart the outer for loop
 8:         end if
 9:     end for
10: end for
11: $\mathbf{s} = \mathbf{t} - \mathbf{t}'$
12: return s
```

algorithm. This work was further improved in [BD15] by optimising the use of the preprocessing data.

However, using the set $R(\mathcal{L})$ in practice is not convenient due to its expected size of about $2^{n+1} - 2$ vectors. One way to reduce the memory requirements could be the use of a compact representation of $R(\mathcal{L})$ like the one described in [HRS19]. In such a scenario a superset of $R(\mathcal{L})$ would be generated on the fly by a CVPP algorithm which would only use a smaller set of vectors in order to generate $R(\mathcal{L})$.

Another way would be to use a subset of $R(\mathcal{L})$ instead of the entire set. Such an approach was introduced in [Laa16b]. In that work an approximate Voronoi cell is defined as the cell implied by a list of short lattice vectors which is potentially a subset of the set $R(\mathcal{L})$. That lead to a heuristic algorithm for CVPP using the approach of Micciancio–Voulgaris but with more practical time and space complexities.

We describe a CVPP algorithm (the tuple slicer, Algorithm 3.5) using the set $P(\mathcal{L})$, and we discuss its advantages and disadvantages against already existing approaches. We distinguish two cases.

If $C = 1$ in Algorithm 3.5 then it just uses a subset of $R(\mathcal{L})$. In this case the analysis of the algorithm just follows under the “approximate Voronoi cell” approach where a specific choice has been made on the used subset. The advantage in this case is that it is guaranteed that the used list of vectors is a subset of $R(\mathcal{L})$.

If $C > 1$ Algorithm 3.5 behaves similar to the tuple sieving approach in [BLS16]. A vector is reduced not only by a single vector but also by the sums of small tuples of vectors in the used list. Hence, a target vector $\mathbf{t}$ is reduced by a superset of $P(\mathcal{L})$. If this superset includes the set $R(\mathcal{L})$ then [SFS09, Lemma 5] guarantees the correctness of the algorithm. This depends on the value of C. We can prove that there always exists a value of C which guarantees the inclusion of $R(\mathcal{L})$ in the generated superset.

Remark 3.61. *In line 3 of Algorithm 3.5 it considers sets of vectors* $\{v_1, \dots, v_l\}$ *such that* $v_i \neq -v_j$ *but it could be that* $v_i = v_j$.

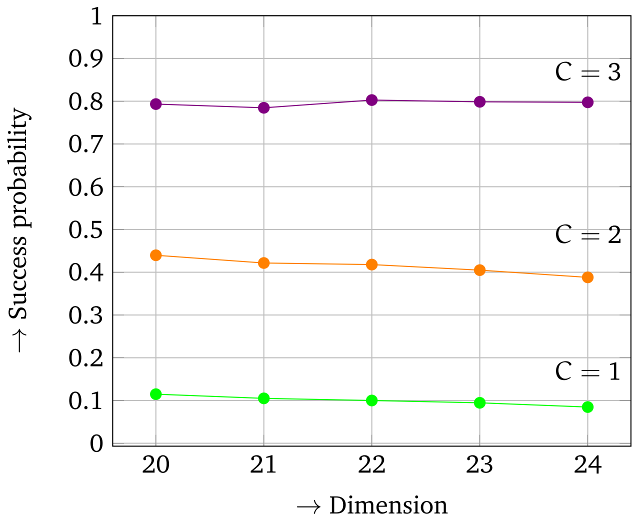


Figure 3.4: Preliminary experimental results on the success probability of Algorithm 3.5. The algorithm was tested on lattices of dimensions 20, 21, 22, 23, 24. For each dimension the algorithm was tested with input $C = 1, 2, 3$ against 10000 CVP instances. Each of the 10000 CVP blocks was formed by 10 smaller blocks of 1000 CVPs corresponding to 10 lattices. Each point in the graph corresponds to the ratio of correct answers out of the 10000 CVP instances.

Definition 3.62. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$ and k a positive integer. We define*

$$k\,P(\mathcal{L}) = \left\{ \sum_{i=1}^{j} \mathbf{p}_i \mid \mathbf{p}_i \in P(\mathcal{L}) \text{ and } j \in \{1, \dots, k\} \right\}.$$

Proposition 3.63. *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$ and $P(\mathcal{L})$ a complete system of irreducible vectors of it. Then there exists a positive integer $n_0 \in \mathbb{N}$ such that $R(\mathcal{L}) \subseteq n_0\,P(\mathcal{L})$.*

Proof. By Proposition 3.37 there exists a generating set $\mathbf{G} \subseteq P(\mathcal{L})$ with $|\mathbf{G}| = l \geqslant n$. Let $\mathbf{r} \in R(\mathcal{L})$, then there exists an $\mathbf{x} \in \mathbb{Z}^l$ such that $\mathbf{G}\mathbf{x} = \mathbf{r}$. With $\mathbf{x} = (x_1, \dots, x_l)$ set $m_\mathbf{r} = \|\mathbf{x}\|_1 = \sum_{i=1}^{l} |x_i|$. Then $\mathbf{r} \in m_\mathbf{r}\,P(\mathcal{L})$. Set $m = \max_{\mathbf{r} \in R(\mathcal{L})}\{m_\mathbf{r}\}$. As $R(\mathcal{L})$ is finite then m is finite and $\forall \mathbf{r} \in R(\mathcal{L})$ it holds $\mathbf{r} \in m\,P(\mathcal{L})$. $\square$

The used superset is computed on the fly. This allows for a time–memory trade-off. The algorithm loses on time complexity as it examines a larger list of vectors but it gains on the memory requirement as it stores a provably smaller subset of $R(\mathcal{L})$. In more detail the space complexity of the algorithm is proportional to $|P(\mathcal{L})|$ which can be bounded by $O(\tau_n)$. The time complexity will depend on the size of $P(\mathcal{L})$ but also on the parameter C. Following the analysis of [MV10a] we can argue that the time complexity of Algorithm 3.5 will be $O(|P(\mathcal{L})|^C \cdot 2^n \operatorname{poly}(n))$.

Remark 3.64. *From Theorem 3.68 it follows that if Algorithm 3.5 was to be applied to the lattice family A_n^*, it should consider a value of C as high as $(n+1)/2$ in order for $R(A_n^*)$ to be included in the used superset. Therefore, a provable upper bound on C alone will not lead to any good bound for the time complexity of Algorithm 3.5 in a provable setting.*

Considering Algorithm 3.5 in a heuristic setting seems to be a more appealing choice. In such a scenario the requirements of the algorithm can be relaxed in mainly two directions. The first one is using an approximation (a superset) of $P(\mathcal{L})$ instead of the set itself. Hence, the output of the MinkowskiSieve as described in Section 3.4.2 could serve as such a choice. Furthermore, choosing a specific approximation of $P(\mathcal{L})$ can allow fixing the value of the parameter C in the following way.

By a heuristic result of [Laa16b] we know that if a list L containing $2^{n/2+o(n)}$ lattice vectors of norm less than $\sqrt{2}\lambda_1(\mathcal{L})$ is used as input to the iterative slicer then the success probability of the algorithm is close to 1. Following this guideline, a value for the parameter C can be chosen in a way that guarantees that the set of all vectors used for reduction in Algorithm 3.5 contains a list of $2^{n/2+o(n)}$ shortest lattice vectors.

Further options can be examined if it is allowed for the used slicing algorithm to succeed with probability much smaller than 1. In such a case the results in [DLdW19, DLvW20] provide a way of relating the success probability to size of the used preprocessed list and hence in our case C.

We briefly experimented on the relation of the success probability of Algorithm 3.5 and the parameter C. The results can be found in Figure 3.4. From these results we get a first indication that the success probability of Algorithm 3.5 increases as the value of C increases. Unfortunately, extending these experiments to moderate dimensions was infeasible, as the exact computation of $P(\mathcal{L})$ would require hundreds or thousands of GB of RAM (using a "brute force" approach). Therefore, obtaining a specific guideline on how to choose a value for C remains an open question.

3.6 — Corner cases among $S_1(\mathcal{L})$, $\mathrm{Irr}(\mathcal{L})$ and $R(\mathcal{L})$

In Section 3.3.1 we posed two questions regarding the set $\mathrm{Irr}(\mathcal{L})$: if and when the corner cases $S_1(\mathcal{L}) \subsetneq \mathrm{Irr}(\mathcal{L}) = R(\mathcal{L})$ and $S_1(\mathcal{L}) = \mathrm{Irr}(\mathcal{L}) \subsetneq R(\mathcal{L})$ are possible. In this section we will give a partial answer to these questions by examining some already known families of special lattices, the duals of the root lattices D_n and A_n (see [CS98, Chapter 4]).

For $n \in \mathbb{N}$ with $n \geqslant 5$ we write[3] $\mathcal{L}_n = 2D_n^*$. Then a basis of $\mathcal{L}_n$ is the following (see [CS98, p. 120])

$$\mathbf{B}_n = \{2e_i \mid 1 \leqslant i \leqslant n-1\} \cup \{1^n\} \tag{3.14}$$

and 1^n represents the all-1 vector.

Theorem 3.65. *For every $n \in \mathbb{N}$ with $n \geqslant 5$*

$$S_1(\mathcal{L}_n) = \{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \quad \textit{and}$$
$$\mathrm{Irr}(\mathcal{L}_n) = R(\mathcal{L}_n) = \{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n.$$

Proof. By the definition of the lattice $\mathcal{L}_n$ it is clear that $\{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n \subset \mathcal{L}_n$. We will prove this theorem in three steps.
The first step is to show that $S_1(\mathcal{L}_n) = \{\pm 2e_i \mid 1 \leqslant i \leqslant n\}$.
The second step will be to show that $R(\mathcal{L}_n) \subseteq \{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n$.
Finally in the third step we will prove that $\{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n \subseteq \mathrm{Irr}(\mathcal{L}_n)$. These

[3]We choose to work with a scaling of D_n^* as in this way we get a lattice in $\mathbb{Z}^n$, which is easier to work with.

three steps imply the result as $\mathrm{Irr}(\mathcal{L}_n) \subseteq \mathrm{R}(\mathcal{L}_n)$.
The "defining property" of the lattice $\mathcal{L}_n$, that if $\boldsymbol{v} = (v_1, \dots, v_n) \in \mathcal{L}_n$ then $v_i \equiv v_j \pmod 2$ for all $1 \leqslant i, j \leqslant n$, will be used throughout the proof.
Step 1: Obtaining that $S_1(\mathcal{L}_n) = \{\pm 2e_i \mid 1 \leqslant i \leqslant n\}$ is trivial and is left as an exercise to the reader.
Step 2: Let $\boldsymbol{v} \notin \mathrm{R}(\mathcal{L}_n)$ and $\boldsymbol{v} \neq \mathbf{0}$. Then by Theorem 3.2 we know that there exists a vector $\boldsymbol{x} \in \mathcal{L}_n \setminus \{\mathbf{0}, \boldsymbol{v}\}$ such that $\langle \boldsymbol{v}, \boldsymbol{x} \rangle \geqslant \|x\|^2$. We will prove that for every vector $\boldsymbol{v} \in \mathcal{L}_n \setminus (\{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n \cup \{\mathbf{0}\})$ there exists a vector $\boldsymbol{x} \in \mathcal{L}_n \setminus \{\mathbf{0}, \boldsymbol{v}\}$ such that $\langle \boldsymbol{v}, \boldsymbol{x} \rangle \geqslant \|x\|^2$. This implies the desired property $\mathrm{R}(\mathcal{L}_n) \subseteq \{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n$.
Let $\boldsymbol{v} \in \mathcal{L}_n \setminus (\{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n \cup \{\mathbf{0}\})$, we distinguish two cases.
Case 1: Let $\boldsymbol{v}$ be such that $\boldsymbol{v} = (v_1, \dots, v_n)$ with $v_i \equiv 1 \pmod 2$ for all v_i. We already showed in step 1 of the proof that the shortest vectors with odd coordinates are the $\{\pm 1\}^n$. As $\boldsymbol{v}$ does not belong to this set, $|v_i| \geqslant 1$ for all v_i, and there exists at least one v_j such that $|v_j| \geqslant 3$. Consider the vector $\boldsymbol{x} = (\mathrm{sign}(v_1), \dots, \mathrm{sign}(v_n))$. This is a valid lattice vector as $\boldsymbol{x} \in \{\pm 1\}^n \subset \mathcal{L}_n$ and $\boldsymbol{x} \neq \boldsymbol{v}$ as $\boldsymbol{v} \in \mathcal{L}_n \setminus (\{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n)$. We check the inner product of $\boldsymbol{v}$ and $\boldsymbol{x}$.

$$\langle \boldsymbol{v}, \boldsymbol{x} \rangle = \sum_{i=1}^{n} \mathrm{sign}(v_i) v_i = \sum_{i=1}^{n} |v_i| \geqslant n + 2 > n = \|\boldsymbol{x}\|^2$$

This proves that $\boldsymbol{v} \notin \mathrm{R}(\mathcal{L}_n)$.
Case 2: Let $\boldsymbol{v}$ be such that $\boldsymbol{v} = (v_1, \dots, v_n)$ with $v_i \equiv 0 \pmod 2$ for all v_i. As $\boldsymbol{v}$ is a non-zero vector then it has at least one non-zero coordinate, let it be v_j. Also as v_j is even we can conclude that $|v_j| \geqslant 2$. We consider the vector $\boldsymbol{x} = 2\,\mathrm{sign}(v_j) e_j$. This is a valid lattice vector as $\boldsymbol{x} \in \{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \subset \mathcal{L}_n$ and $\boldsymbol{x} \neq \boldsymbol{v}$ as $\boldsymbol{v} \in \mathcal{L}_n \setminus (\{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n)$. We check the inner product of $\boldsymbol{v}$ and $\boldsymbol{x}$.

$$\langle \boldsymbol{v}, \boldsymbol{x} \rangle = \sum_{i=1}^{n} x_i v_i = 2\,\mathrm{sign}(v_j) v_j = 2|v_j| \geqslant 4 = \|\boldsymbol{x}\|^2$$

This proves that again $\boldsymbol{v} \notin \mathrm{R}(\mathcal{L}_n)$ concluding the proof of the second step.
Step 3: In this step we want to prove that $\{\pm 2e_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n \subseteq \mathrm{Irr}(\mathcal{L}_n)$. In step 1 we already showed that $S_1(\mathcal{L}_n) = \{\pm 2e_i \mid 1 \leqslant i \leqslant n\}$ and we know that $S_1(\mathcal{L}_n) \subseteq \mathrm{Irr}(\mathcal{L}_n)$ hence, we only have to show that $\{\pm 1\}^n \subseteq \mathrm{Irr}(\mathcal{L}_n)$. Assume that $\boldsymbol{v} \in \{\pm 1\}^n$ and $\boldsymbol{v} \notin \mathrm{Irr}(\mathcal{L}_n)$. Thus there are two strictly shorter vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ such that $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2$. In step 1 of the proof we showed that the vectors in $\{\pm 1\}^n$ are the shortest ones among those with odd coordinates. Therefore as $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are strictly shorter than $\boldsymbol{v}$ then it must be that they have even coordinates. This implies that $\boldsymbol{v}$ can be written as a sum of vectors with even coordinates. This is a contradiction, as a sum of even numbers is never odd. □

As a scaling of a lattice $\mathcal{L}$ has the same properties as $\mathcal{L}$ we get Theorem 3.27 already mentioned in Section 3.3.1.

Theorem 3.66. *Let $n \in \mathbb{N}$ with $n \geqslant 5$. Then for the lattice D_n^* it holds that $S_1(D_n^*) \subsetneq \mathrm{Irr}(D_n^*) = \mathrm{R}(D_n^*)$. Furthermore $|\mathrm{Irr}(D_n^*)| = 2^n + 2n$.*

This proves that $S_1(\mathcal{L}) \subsetneq \mathrm{Irr}(\mathcal{L}) = R(\mathcal{L})$ is possible for every dimension $n \geqslant 5$. In order to complete this result from this point of view we give another three lattices, one for each of the dimensions $n = 2, 3, 4$ that possess the same property.

For $n = 2, 3, 4$ we write $\mathcal{L}_2 = \mathcal{L}(\mathbf{B}_2), \mathcal{L}_3 = \mathcal{L}(\mathbf{B}_3), \mathcal{L}_4 = \mathcal{L}(\mathbf{B}_4)$ with $\mathbf{B}_2, \mathbf{B}_3, \mathbf{B}_4$ being

$$\mathbf{B}_2 = \begin{pmatrix} 3 & 1 \\ 0 & 1 \end{pmatrix} \quad \mathbf{B}_3 = \begin{pmatrix} 3 & 0 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{B}_4 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.15}$$

We leave it to the reader to verify our claim for these three lattices.
Our next goal is to derive a similar result for the case $S_1(\mathcal{L}) = \mathrm{Irr}(\mathcal{L}) \subsetneq R(\mathcal{L})$. In order to do so we will use a scaling of the lattices A_n^*.

For $n \in \mathbb{N}$ with $n \geqslant 3$, we write $\mathcal{M}_n = (n+1)A_n^*$. Then a basis of $\mathcal{M}_n$ is formed by the columns of $\mathbf{B}_n$ (see [CS98, p. 115]), where

$$\mathbf{B}_n = \begin{pmatrix} -n & 1 & 1 & \cdot & 1 \\ 1 & -n & 1 & \cdot & 1 \\ 1 & 1 & -n & \cdot & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \cdot & -n \\ 1 & 1 & 1 & \cdot & 1 \end{pmatrix} \tag{3.16}$$

is an $(n+1) \times n$ matrix.

Remark 3.67. *By the given basis $\mathbf{B}_n$ for $\mathcal{M}_n$ we can immediately observe that if $\mathbf{v} = (v_1, v_2, \ldots, v_{n+1}) \in \mathcal{M}_n$ then $v_i \equiv v_j \pmod{n+1}$. Additionally $\sum_{i=1}^{n+1} v_i = 0$.*

For the next theorem we will adopt the notation (x^n, y^m) in order to denote all vectors in $\mathbb{Z}^{n+m}$ containing n coordinates equal to x and m coordinates equal to y in some order.

Theorem 3.68. *For every $n \in \mathbb{N}$ with $n \geqslant 3$,*

$$S_1(\mathcal{M}_n) = \mathrm{Irr}(\mathcal{M}_n) = \{\pm(-n^1, 1^n)\} \quad \textit{and}$$
$$R(\mathcal{M}_n) = \left\{ \pm(\alpha^\beta, (-\beta)^\alpha) \mid \beta = n + 1 - \alpha\,,\ 1 \leqslant \alpha \leqslant \frac{n+1}{2} \right\}.$$

Proof. We set $A = \{\pm(\alpha^\beta, (-\beta)^\alpha) \mid \beta = n+1-\alpha\,,\ 1 \leqslant \alpha \leqslant (n+1)/2\}$. Verifying that $A \subseteq \mathcal{M}_n$ is left as an exercise to the reader. We will prove this theorem in four steps. The first step is to show that $S_1(\mathcal{M}_n) = \{\pm(-n^1, 1^n)\}$. The second step will be to show that $R(\mathcal{M}_n) \subseteq A$. The third step will be to show that $\mathrm{Irr}(\mathcal{M}_n) \subseteq \{\pm(-n^1, 1^n)\}$ and finally in the fourth step we will show that $A \subseteq R(\mathcal{M}_n)$.
Step 1: The vectors $\{\pm(-n^1, 1^n)\}$ have squared length $n^2 + n$ and hence we get $\lambda_1^2(\mathcal{L}) \leqslant n^2 + n$. This implies that a vector achieving $\lambda_1(\mathcal{L})$ cannot have a coordinate v_j such that $|v_j| \geqslant n+1$. Therefore a vector achieving $\lambda_1(\mathcal{L})$ belongs to A. The squared length of a vector in A is $\beta\alpha^2 + \alpha\beta^2 = (n+1)\alpha\beta$ which minimizes for $\alpha = 1$.
Step 2: Let $\mathbf{v} \in \mathcal{M}_n \setminus (A \cup \{\mathbf{0}\})$ and write it as $\mathbf{v} = (v_1, \ldots, v_{n+1})$. Then there will exist at least one coordinate of $\mathbf{v}$, let it be v_j, such that $|v_j| \geqslant n+1$. This can be proved by a

contradiction argument. Assume that there was no coordinate in $\boldsymbol{v}$ such that $|v_j| \geqslant n+1$ then it would hold that $|v_i| \leqslant n$ for all $1 \leqslant i \leqslant n$ and by the fact that $v_i \equiv v_j \pmod{n+1}$ we can conclude that there would be at most two possible values for $|v_i|$. But the set A contains all such vectors of the lattice, hence that would imply $\boldsymbol{v} \in A$, contradiction. We set $\boldsymbol{x}$ to be the vector having $\operatorname{sign}(v_j)n$ in the j-th position and $-\operatorname{sign}(v_j)$ in all other places. This is a valid lattice vector and $\boldsymbol{x} \neq \boldsymbol{v}$ as $\boldsymbol{v} \in \mathcal{M}_n \setminus A$. We check the inner product of $\boldsymbol{v}$ and $\boldsymbol{x}$:

$$\langle \boldsymbol{v}, \boldsymbol{x} \rangle = \sum_{i=1}^{n+1} v_i x_i = |v_j| n - \operatorname{sign}(v_j) \sum_{\substack{i=1 \\ i \neq j}}^{n+1} v_i = |v_j| n + |v_j| \geqslant (n+1)^2 > \|\boldsymbol{x}\|^2.$$

This proves that $\boldsymbol{v} \notin R(\mathcal{M}_n)$ concluding the proof of the second step.
Step 3: Let $\boldsymbol{v} \in \mathcal{M}_n \setminus (\{\pm(-n^1, 1^n)\} \cup \{\mathbf{0}\})$ and write it as $\boldsymbol{v} = (v_1, \dots, v_{n+1})$. Then we will show that $\boldsymbol{v}$ is reducible. By step 2 of the proof we know that $R(\mathcal{M}_n) \subseteq A$ and as we know that $\operatorname{Irr}(\mathcal{M}_n) \subseteq R(\mathcal{M}_n)$ we can restrict our choice to $\boldsymbol{v} \in A \setminus \{\pm(-n^1, 1^n)\}$. As $\boldsymbol{v} \in A$ we can write $\boldsymbol{v} = \pm(\alpha^\beta, (-\beta)^\alpha)$ with $\beta = n+1-\alpha$ for some $1 < \alpha \leqslant (n+1)/2$. By Lemma 3.24 it suffices to find a lattice vector $\boldsymbol{x}$ with $\|\boldsymbol{x}\| < \|\boldsymbol{v}\|$ and such that $2\langle \boldsymbol{v}, \boldsymbol{x} \rangle > \|\boldsymbol{x}\|^2$. Let $\gamma = \max\{|\alpha|, |\beta|\}$ and the j-th coordinate of $\boldsymbol{v}$ be such that $|v_j| = \gamma$. Consider $\boldsymbol{x}$ to be the vector with $\operatorname{sign}(v_j)n$ in the j-th position and $-\operatorname{sign}(v_j)$ in all other places. This is a valid lattice vector and $\|\boldsymbol{x}\| < \|\boldsymbol{v}\|$ as $\boldsymbol{x} \in S_1(\mathcal{M}_n)$ but $\boldsymbol{v} \notin S_1(\mathcal{M}_n)$. Then

$$\begin{aligned} 2\langle \boldsymbol{v}, \boldsymbol{x} \rangle &= 2 \sum_{i=1}^{n+1} v_i x_i = 2\left(|v_j| n - \operatorname{sign}(v_j) \sum_{\substack{i=1 \\ i \neq j}}^{n+1} v_i \right) = 2(\gamma n + \gamma) \\ &= 2(n+1)\gamma \geqslant (n+1)^2 > \|\boldsymbol{x}\|^2 \end{aligned}$$

as $\gamma \geqslant (n+1)/2$. This proves that $\boldsymbol{v} \notin \operatorname{Irr}(\mathcal{M}_n)$ and therefore $\operatorname{Irr}(\mathcal{M}_n) \subseteq \{\pm(-n^1, 1^n)\}$.
Step 4: By [CS92, Theorem 3] and the fact that the vectors $(-n^1, 1^n)$ form a strictly obtuse superbasis of $\mathcal{M}_n$ (see [CS92]) it follows that $A \subseteq R(\mathcal{M}_n)$ and finally $R(\mathcal{M}_n) = A$. $\square$

This implies Theorem 3.28 already mentioned in Section 3.3.1.

Theorem 3.69. *Let* $n \in \mathbb{N}$ *with* $n \geqslant 3$. *Then for the lattice* A_n^* *it holds that* $S_1(A_n^*) = \operatorname{Irr}(A_n^*) \subsetneq R(A_n^*)$. *Furthermore* $|\operatorname{Irr}(A_n^*)| = 2(n+1)$.

This proves that $S_1(\mathcal{L}) = \operatorname{Irr}(\mathcal{L}) \subsetneq R(\mathcal{L})$ is possible for every dimension $n \geqslant 3$. In order to complete the result from this point of view we give another lattice in dimension $n = 2$ that possess the same property: $\mathcal{M}_2 = \mathcal{L}(\mathbf{B}_2)$, where

$$\mathbf{B}_2 = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix} \tag{3.17}$$

We leave it to the reader to verify this.

3.7 — Some graph theoretical aspects

In Section 3.3.2 we introduced the notion of a complete system of irreducible vectors and we gave an example of how the set $P(\mathcal{L})$ can be computed. In that example the use of graph theoretical tools was demonstrated in order to compute the set $P(\mathcal{L})$ given the set $\mathrm{Irr}(\mathcal{L})$. A natural question that arises is how costly this step can be.

In order to answer this question a few graph theory definitions are necessary. Graphs will de denoted by $\Gamma = (V, E)$, where V is the set of vertices and E is the set of edges. If $e = \{u, v\} \in E$ then we say that u and v are adjacent.

Definition 3.70 (Independent set)**.** *Given a simple graph $\Gamma = (V, E)$ an independent set is a subset of vertices $U \subseteq V$, such that no two vertices in U are adjacent. An independent set is maximal if no vertex can be added without violating independence. An independent set of maximum cardinality is called maximum and its cardinality is denoted by $\alpha(\Gamma)$.*

Definition 3.71 (Class graph)**.** *Let $\mathcal{L}$ be a full rank lattice in $\mathbb{R}^n$ and $S \in \mathrm{Irr}(\mathcal{L})/\sim$. We define $\Gamma_{\mathcal{L}}(S)$ to be the graph where the set of vertices $V = S$ and there exists an edge between $\mathbf{v}_1, \mathbf{v}_2 \in V$ iff $\|\mathbf{v}_1 - \mathbf{v}_2\| < \|\mathbf{v}_1\|$.*

Computing $P(\mathcal{L})$ out of $\mathrm{Irr}(\mathcal{L})$ amounts to solving a maximal independence set instance in $\Gamma_{\mathcal{L}}(S)$ for every class $S \in \mathrm{Irr}(\mathcal{L})/\sim$. Therefore the complexity of this task highly depends on the size of the equivalence classes $S \in \mathrm{Irr}(\mathcal{L})/\sim$ and $|\mathrm{Irr}(\mathcal{L})/\sim|$. For average-case lattices the computational step from $\mathrm{Irr}(\mathcal{L})$ to $P(\mathcal{L})$ should almost always be trivial, i.e. $P(\mathcal{L}) = \mathrm{Irr}(\mathcal{L})$, as for all $S \in \mathrm{Irr}(\mathcal{L})/\sim$ it is expected that $|S| = 2$. In these cases the set $P(\mathcal{L})$ is uniquely determined.

However, in case the underlying lattice $\mathcal{L}$ possesses any kind of structure or symmetries it is expected that there will be equivalence classes $S \in \mathrm{Irr}(\mathcal{L})/\sim$ with $|S| > 2$. In these cases the computational task of finding a maximal independent set in the corresponding class graph is not trivial anymore. In such cases the first step is to construct the corresponding graph $\Gamma_{\mathcal{L}}(S)$, which will take time $O(|S|^2)$. Then, naively computing a maximal independent set (which should always include both $\pm\mathbf{v}$) will take time $O(|S|m)$ where m is the number of edges in $\Gamma_{\mathcal{L}}(S)$ but, there are better performing algorithms for this task [Lub86]. If we denote by h the maximum size of a class in $\mathrm{Irr}(\mathcal{L})/\sim$ then the time complexity of computing $P(\mathcal{L})$ out of $\mathrm{Irr}(\mathcal{L})$ will scale as $O(h^2|\mathrm{Irr}(\mathcal{L})|)$.

Thus if there does not exist a class S with $|S|$ exponential to the dimension n then computing $P(\mathcal{L})$ out of $\mathrm{Irr}(\mathcal{L})$ will take time $\tilde{O}(|\mathrm{Irr}(\mathcal{L})|)$. In practice, stumbling upon a lattice $\mathcal{L}$ possessing a class $S \in \mathrm{Irr}(\mathcal{L})/\sim$ where $|S|$ is exponential to the dimension n is highly unlikely as such a lattice would be extremely structured. For the sake of mathematical curiosity (and nice graph pictures) we briefly investigate such a case of lattices, namely the $\mathcal{L}_n$ for $n \geqslant 5$ defined in Section 3.6. For our exposition we will need the following definition.

Definition 3.72 (Cayley graph)**.** *Let G be a group and $T \subseteq G$ a generating set of G. The Cayley graph of G generated by T, denoted $\mathrm{Cay}(G, T)$ is the directed graph $\Gamma = (V, E)$ where $V = G$ and $E = \{(g, gs) \mid g \in G,\ s \in T\}$.*

If $T = T^{-1}$ (T is closed under inversion) then $\mathrm{Cay}(G, T)$ is an undirected graph.

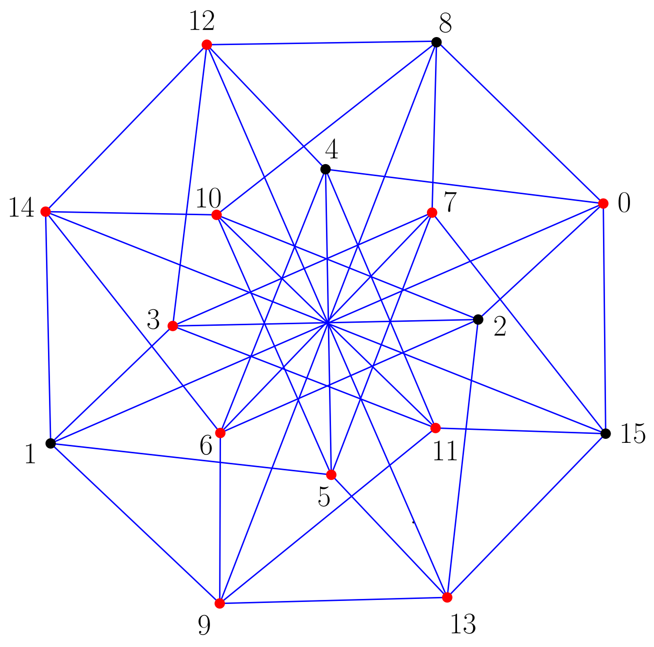


Figure 3.5: The uncoloured Cayley graph $\mathrm{Cay}(\mathbb{Z}_2^4, \varphi(T_1(\mathbb{Z}_2^5)))$ with generating set $\varphi(T_1(\mathbb{Z}_2^5)) = \{(0,0,0,1),(0,0,1,0),(0,1,0,0),(1,0,0,0),(1,1,1,1)\}$. For convenience, instead of labelling the vertices of the graph by the elements of $\mathbb{Z}_2^4$, we consider the elements of $\mathbb{Z}_2^4$ as binary representations and assign the corresponding integer (e.g. $(1,0,0,0)$ maps to 8). This graph can be used in order to compute a representative set $\tilde{S}_2$ for the class $S_2 = \{\pm 1\}^5$ in the $\mathcal{L}_5 = 2D_5^*$ lattice. One such set is implied by the black vertices of the graph. The graph possesses 40 maximal independent sets of cardinality 4 and 16 maximal independent sets of cardinality 5 (maximum).

In Section 3.6 we saw that $\mathrm{Irr}(\mathcal{L}_n) = \{\pm 2\boldsymbol{e}_i \mid 1 \leqslant i \leqslant n\} \cup \{\pm 1\}^n$. Hence $\mathrm{Irr}(\mathcal{L}_n)$ contains two equivalence classes of sizes $2n$ and 2^n respectively. The class $S_2 := \{\pm 1\}^n$ which we will study can be viewed as the group $G = \mathbb{Z}_2^n$. Two elements of S_2 are connected if their difference is shorter than $\sqrt{n}$, thus it is a sum of less than $n/4$ elements from the set $\{\pm 2\boldsymbol{e}_i \mid 1 \leqslant i \leqslant n\}$. In turn this implies that two elements of S_2 are connected in $\Gamma_{\mathcal{L}_n}(S_2)$ if they differ by a sign in less than $n/4$ of their coordinates. Thus we can now use the following observation.

$$\Gamma_{\mathcal{L}_n}(S_2) \cong \mathrm{Cay}(G,\, T_{\lceil n/4 \rceil}(G)), \tag{3.18}$$

where $G = \mathbb{Z}_2^n$ and $T_{\lceil n/4 \rceil}(G) := \{\boldsymbol{x} \in G \,|\, 1 \leqslant |\mathrm{supp}(\boldsymbol{x})| < \lceil n/4 \rceil\}$ and $\mathrm{supp}(\boldsymbol{x})$ denotes the support of $\boldsymbol{x}$. In our case $T_{\lceil n/4 \rceil}(G) = T^{-1}_{\lceil n/4 \rceil}(G)$ and thus $\mathrm{Cay}(G, T_{\lceil n/4 \rceil}(G))$ is an undirected graph.[4] We are interested in the maximal independent sets of the graph $\Gamma_{\mathcal{L}_n}(S_2)$, but not in all of them. It is additionally required that for every vector $\boldsymbol{v} \in \tilde{S}_2$ also $-\boldsymbol{v} \in \tilde{S}_2$. This could be phrased as we work "modulo sign". This property can be translated algebraically by working in the quotient group $H = Z_2^n / \langle (1,\dots,1) \rangle$ instead of

[4]Such type of Cayley graphs are of an interest in coding theory as independent sets of $\mathrm{Cay}(\mathbb{Z}_q^n, T_d(\mathbb{Z}_q^n))$ with $T_d(\mathbb{Z}_q^n) = \{x \in \mathbb{Z}_q^n \,|\, 1 \leqslant |\mathrm{supp}(\boldsymbol{x})| < d\}$ correspond to q-ary (n, d) codes.

$G = \mathbb{Z}_2^n$. Using the group isomorphism

$$\varphi : \mathbb{Z}_2^n / \langle (1, \dots, 1) \rangle \to \mathbb{Z}_2^{n-1}$$
$$(x_i)_{i=1}^n + \langle (1, \dots, 1) \rangle \mapsto (x_n + x_i)_{i=1}^{n-1}$$

we can transfer the problem to the graph $\Gamma_{sign} = \mathrm{Cay}(\mathbb{Z}_2^{n-1}, \varphi(T_{\lceil n/4 \rceil}(G)))$. Each independent set in Γ_{sign} implies an independent set in $\Gamma_{\mathcal{L}_n}(S_2)$ which possesses the extra property of the "sign symmetry". A first remark regarding the set of maximal independent sets of Γ_{sign} is that it is invariant under the group action of $\mathbb{Z}_2^{n-1}$. For example, if we consider the graph in Figure 3.5, all 16 maximal independent sets of cardinality 5 can be generated by acting with $\mathbb{Z}_2^4$ to the given independent set formed by the black vertices.

We briefly experimented with Γ_{sign} for the first few values $n = 5, 6, 7$ in order to get a first indication of how many maximal independent sets such a graph may have and how much their size can vary.

Table 3.2: Using SAGE [TSD19] we computed all possible sizes of a maximal independent set of $\mathrm{Cay}(\mathbb{Z}_2^{n-1}, \varphi(T_{\lceil n/4 \rceil}(\mathbb{Z}_2^n)))$ for $n = 5, 6, 7$ and the corresponding frequency of these sizes.

$n = 5$		
Cardinality	4	5
Frequency	40	16

$n = 6$				
Cardinality	6	8	11	16
Frequency	320	300	32	2

$n = 7$								
Cardinality	8	14	16	17	18	19	20	22
Frequency	240	1920	625548	203840	67200	13440	2800	64

As the number of maximal independent sets seems to grow super-exponentially in the dimension n we stopped at $n = 7$. Even though experimental results are useful in order to get intuition, theoretical results are those which give the final answer to a question. In our case there are some theoretical results, originating both from graph theory and coding theory which bound the sizes we experimented with.

- Let $\Gamma = (V, E)$ be a graph with $|V| = N$. In [MM65] it is proven that Γ can have up to $3^{N/3}$ maximal cliques in the worst case, a bound which is tight. Complementary this also proves that a graph Γ with N vertices can possess up to $3^{N/3}$ maximal independent sets in the worst case.
  The results in the same work also imply that the number of different sizes of maximal independent sets is upper bounded by $N - \log_2 N$ which is shown to be tight in the worst case.

- Let Γ be an m-regular graph with N vertices. In [Ros64] it is shown that $\alpha(\Gamma)$ can be upper bounded by $\min\{\lfloor N/2 \rfloor, N - m\}$. This bound is obtainable. In the same work, a lower bound for $\alpha(\Gamma)$ is given, depending on m and N. However this bound is not uniform but depends on number theoretic properties of N, m. In our case the appropriate lower bound for $\alpha(\Gamma)$ would be $\lceil N/(m+1) \rceil$.

- As the graph family in question, $\Gamma_{\mathcal{L}_n}(S_2)$ (and Γ_{sign}) is specific, better upper bounds can be obtained than the general ones given in [Ros64]. This is achieved with the use of coding theory [Hop18]. In more detail, $\alpha(\Gamma_{\mathcal{L}_n}(S_2)) = A_2(n, \lceil n/4 \rceil)$. This equality enables the use of already known upper bounds on $A_2(n, \lceil n/4 \rceil)$ from coding theory such as the Hamming bound [Ham50]. The lower bound implied by [Ros64] for $\alpha(\Gamma_{\mathcal{L}_n}(S_2))$ is equivalent to the Gilbert-Varshamov bound for $A_2(n, \lceil n/4 \rceil)$.

Chapter 4

Hybrid lattice algorithms for SVP via CVPP

This chapter is for all practical purposes identical to the paper *Sieve, Enumerate, Slice, and Lift: Hybrid Lattice Algorithms for SVP via CVPP* [DLdW20] authored jointly with Thijs Laarhoven and Benne de Weger, which was published at Africacrypt 2020.

4.1 — Introduction

In recent decades, lattice-based cryptography has emerged as a front-runner for building secure and efficient cryptographic primitives in the post-quantum age. For an accurate and reliable deployment of these schemes, it is essential to obtain a good understanding of the hardness of the underlying lattice problems, such as the shortest (SVP) and closest vector problems (CVP).

To date, research on lattice algorithms has resulted in two main flavors of algorithms: *enumeration* methods, requiring $2^{O(d \log d)}$ time and $d^{O(1)}$ space to solve hard lattice problems in dimension d [Kan83, FP85, GNR10, AN17]; and *sieving* methods, running in expected time and space $2^{O(d)}$ [AKS01, NV08, MV10a, ADRS15]. Just a few years ago, enumeration clearly dominated benchmarks for testing these algorithms in practice [GNR10, CN11, FK15, svp19], but recent improvements to sieving have allowed it to overtake enumeration in practice as well [MV10b, Laa15, BDGL16, Duc18, ADH+19]. Some attempts have also been made to combine the best of both worlds, a.o. resulting in the tuple sieving line of work [BLS16, HK17, HKL18]. A better comprehension of how to exploit the strengths and weaknesses of each method remains an interesting open problem.

A long-standing open problem from e.g. [GNR10, DLdW19] concerns the possibility of speeding up lattice enumeration with a batch-CVP solver: if an efficient algorithm exists that can solve a large number of CVP instances on the same lattice faster than solving each problem separately, then this algorithm can be used to solve the CVP instances appearing implicitly in the enumeration tree faster. For a long time no such efficient batch-CVP algorithms were known, until the recent line of work on approximate Voronoi cells and the randomized slicer [DLdW19, Laa19, DLvW20] showed that, at least in high dimensions, one can indeed solve large batches faster in practice than solving each problem separately. This raises the question whether these new results can be used to instantiate this conjectured hybrid algorithm and obtain better results, in theory and in practice.

4.1.1 – Contributions. In this chapter we study the feasibility of combining recent batch-CVP algorithms with lattice enumeration, and show that we heuristically obtain a $2^{\Theta(d/\log d)}$ speedup and memory reduction for solving SVP compared to the state-of-the-art lattice sieve. This improvement is proper, in the sense that this does not hide large order terms: we show that for solving SVP in dimension d, the costs are proportional to those of running a sieve in dimension $d - \Theta(d/\log d)$, making the leading constant explicit, and showing that the remaining overhead is negligible. The hybrid constructions we propose are independent of e.g. the underlying nearest neighbor data structure, and we expect that these and other heuristic improvements can be applied to the hybrid algorithms as well.

Obtaining $\Theta(d/\log d)$ dimensions *for free* may sound familiar, as Ducas [Duc18] showed that sieving in dimension $d-\Theta(d/\log d)$ implies solving SVP in dimension d. As the asymptotic improvement of Ducas is greater than ours, to improve upon his results we need to be able to combine both techniques. The feasibility of such a combined hybrid algorithm relies on Assumption 4.10, which Section 4.5 aims to verify with experiments. Combining both techniques, we asymptotically obtain $0.5305d/\log_2 d$ dimensions for free, compared to Ducas' $0.4150d/\log_2 d$.

4.1.2 – Notation. We adopt the already introduced notation from Chapter 1 for basic notions regarding lattices and lattice problems. We write $D_{\mathbf{t}+\mathcal{L},s}$ for the discrete Gaussian distribution on $\mathbf{t}+\mathcal{L}$ with probability mass function proportional to $\rho_{s,\mathbf{t}}(\mathbf{x}) = \exp(-\pi\|\mathbf{t}-\mathbf{x}\|^2/s^2)$. For $\mathbf{t} \in \mathbb{R}^d$ we define $d(\mathbf{t},\mathcal{L}) := \min_{\mathbf{v}\in\mathcal{L}} \|\mathbf{t}-\mathbf{v}\|$, where all norms are Euclidean norms.

4.2 — Preliminaries

4.2.1 – Heuristic Assumptions. For our asymptotic analyses we will rely on a number of common heuristic assumptions, which have often been used throughout the literature and we have introduced in Section 1.2. In particular, we will use the Gaussian heuristic 1.18 and the Geometric Series Assumption 1.20. Using volume arguments, the Gaussian heuristic predicts that $\lambda_1(\mathcal{L}) = \mathrm{gh}(\mathcal{L})$ where $\mathrm{gh}(\mathcal{L}) := \sqrt{d/(2\pi e)} \cdot \mathrm{Vol}(\mathcal{L})^{1/d} \cdot (1+o(1))$. For random targets $\mathbf{t} \in \mathbb{R}^d$, we further expect that $d(\mathbf{t},\mathcal{L}) = \mathrm{gh}(\mathcal{L})\cdot(1+o(1))$ with high probability.

In this chapter we will use a slightly different form of the Geometric Series Assumption 1.20 than the one given in Section 1.2. In fact we are interested in a formula relating $\|\mathbf{b}_i^*\|$ to $\|\mathbf{b}_1\|$ rather than $\mathrm{Vol}(\mathcal{L})^{1/d}$, where $\mathbf{b}_1$ denotes the first vector in the basis. We emphasise that the basis vectors will be denoted as $\mathbf{b}_1, \dots, \mathbf{b}_d$ instead of $\mathbf{b}_0, \dots, \mathbf{b}_{d-1}$ which was used in Section 1.2. Also, we will ignore the exact formula for α_β given in Section 1.2. We will use the Geometric Series Assumption in the following form.

Assumption 4.1. *Let $\mathbf{B}$ be a strongly reduced basis of a lattice $\mathcal{L}$. Then the Gram–Schmidt vectors $\mathbf{b}_i^*$ satisfy*

$$\|\mathbf{b}_i^*\| = q^{i-1}\|\mathbf{b}_1\|, \qquad q \in (0,1). \tag{4.1}$$

The GSA is used in analyzing enumeration and Babai lifting (Sections 4.2.2, 4.2.5).

Assumption 4.2 (Randomized slicer assumption [DLdW19]). *Let $s \gg 0$, and let $X_1, X_2, \dots \in \{0, 1\}$ denote the events that running the iterative slicer on $\mathbf{t}_i \sim D_{\mathbf{t}+\mathcal{L},s}$ returns the shortest vector $\mathbf{t}' \in \mathbf{t} + \mathcal{L}$ ($X_i = 1$) or not ($X_i = 0$). Then the random variables X_i are identically and independently distributed.*

This assumption is related to the randomized slicer, discussed in Section 4.2.4.

4.2.2 – Lattice Enumeration. For constructing hybrid algorithms for solving SVP, we will combine several existing techniques, the first of which is lattice enumeration. This method, first described in the 1980s [FP85, Kan83] and later significantly improved in practice [GNR10, MW15, AN17], can be seen as a brute-force approach to SVP: every lattice vector can be described as an integer linear combination of the basis vectors, and given some guarantees on the quality of the input basis, this results in bounds on the coefficients of the shortest vector in terms of this basis. The algorithm can be described as a depth-first tree search, requiring $d^{O(1)}$ memory and $2^{O(d \log d)}$ time. For further details, we refer the reader to e.g. [GNR10, HPS11, LvdPdW12].

For our purposes, what is important to know is that the complexity of (partial) enumeration is proportional to the number of nodes visited in the tree, and that the number of nodes at depth $k = o(d)$ for a strongly-reduced d-dimensional lattice basis is $2^{O(k \log d)}$. More precisely, we will need the following lemma which can be derived heuristically, based on estimates from [HS07].

Lemma 4.3 (Costs of enumeration [HS07]). *Let $\mathbf{B}$ be a strongly reduced basis of a lattice. Then the number of nodes E_k at depth $k = o(d)$, $k = d^{1-o(1)}$, satisfies:*

$$E_k = d^{k/2+o(k)}. \tag{4.2}$$

Enumerating all these nodes can be done in time T_{enum} and space S_{enum}, with:

$$T_{\text{enum}} = E_k \cdot d^{O(1)}, \qquad S_{\text{enum}} = d^{O(1)}. \tag{4.3}$$

Proof. As a starting point, we take the formula from [HS07, Section 6.2], which was derived using the Gaussian heuristic:

$$E_k = \frac{\pi^{k/2}}{\Gamma(k/2+1)} \cdot \frac{\|\mathbf{b}_1\|^k}{\prod_{i=d-k+1}^{d} \|\mathbf{b}_i^*\|}. \tag{4.4}$$

For the gamma function, we can use a very rough version of Stirling's approximation of the form $\Gamma(x) = (x/e)^{x+o(x)}$, which for the first term above gives an asymptotic scaling of $(2\pi e/k)^{k/2+o(k)} = k^{-k/2+o(k)}$. For the terms $\|\mathbf{b}_1\|$ and $\|\mathbf{b}_i^*\|$, we apply the geometric series assumption, which implies that $\|\mathbf{b}_i^*\| = q^{i-1}\|\mathbf{b}_1\|$ for some $q \in (0, 1)$. Using that $\sum_{i=d-k+1}^{d}(i-1) = k(2d-k-1)/2 = kd - o(kd)$ for $k = o(d)$, this reduces the above to:

$$E_k = k^{-k/2+o(k)} \cdot q^{-kd+o(kd)}. \tag{4.5}$$

Next, we note that for a sufficiently well-reduced basis $\mathbf{B}$, we have $\|\mathbf{b}_1\| = O(\lambda_1(\mathcal{L})) = O(\sqrt{d}) \cdot \mathrm{Vol}(\mathcal{L})^{1/d}$. From the GSA, we then get:

$$\mathrm{Vol}(\mathcal{L}) = \prod_{i=1}^{d} \|\mathbf{b}_i^*\| = q^{d(d+1)/2}\|\mathbf{b}_1\|^d = q^{d(d+1)/2} d^{-d/2+o(d)}\, \mathrm{Vol}(\mathcal{L}). \tag{4.6}$$

From this we can conclude that $q = d^{-1/d+o(1/d)}$ and $q^{-kd+o(kd)} = d^{k+o(k)}$. From the assumptions that $k = d^{1-o(1)}$ and $k = o(d)$ we then get:

$$E_k = d^{-k/2+o(k)} \cdot d^{k+o(k)} = d^{k/2+o(k)}. \tag{4.7}$$

As for the time and space complexities of enumeration, as has been noted several times before [FP85, GNR10, AN17] the time complexity is directly proportional to the size of the enumeration tree, while the space complexity is only polynomial in d. □

4.2.3 – Lattice Sieving. Another method for solving SVP, and which will be part of our hybrid algorithms, is lattice sieving. This method dates back to the 2000s [AKS01, NV08, MV10b] and has seen various recent improvements [Laa15, BDGL16, Duc18, HKL18, ADH+19] that allowed it to surpass enumeration in the SVP benchmarks [svp19]. This method only requires $2^{O(d)}$ time to solve SVP in dimension d (compared to $2^{O(d \log d)}$ for enumeration), but this comes at the cost of a memory requirement of $2^{O(d)}$. The algorithm starts out by generating a large number of lattice vectors as simple combinations of the basis vectors, and then proceeds by combining suitable pairs of vectors to form shorter lattice vectors. For additional details, see e.g. [BDGL16, HPS11, LvdPdW12, Laa16a].

In the context of this chapter we will make use of the following result from [BDGL16], which is the current state-of-the-art for (heuristic) lattice sieving in high dimensions d. The statement below is stronger than saying that sieving merely solves SVP, as lattice sieving commonly returns a list of all short lattice vectors within radius approximately $\sqrt{4/3} \cdot \lambda_1(\mathcal{L})$. This same assumption was used in [Duc18].

Lemma 4.4 (Costs of lattice sieving [BDGL16])**.** *Given a basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, the LDSieve heuristically returns a list* $L \subset \mathcal{L}$ *containing the* $(4/3)^{d/2+o(d)}$ *shortest lattice vectors, in time* T_{sieve} *and space* S_{sieve} *with:*

$$T_{\text{sieve}} = (3/2)^{d/2+o(d)}, \qquad S_{\text{sieve}} = (4/3)^{d/2+o(d)}. \tag{4.8}$$

With the LDSieve we can therefore solve SVP with the above complexities.

4.2.4 – The Randomized Slicer. The third ingredient for our hybrid algorithms is the randomized slicer for solving CVP(P). This algorithm, described in [DLdW19], is an extension of the iterative slicer [SFS09], and follows a procedure of reducing target $\mathbf{t}$ with a list $L \subset \mathcal{L}$ to find shorter vector $\mathbf{t}' \in \mathbf{t} + \mathcal{L}$. The goal is to find the shortest vector $\mathbf{t}^* \in \mathbf{t} + \mathcal{L}$ by repeatedly reducing $\mathbf{t}$ with L, since $\mathbf{t} - \mathbf{t}^*$ is the solution to CVP$(\mathcal{L}, \mathbf{t})$.

We will make use of two separate results from [DLvW20]. These results differ in whether one desires to solve only one or many CVP instances on the same lattice; as shown in [DLvW20], solving many CVP instances simultaneously allows for more efficient memory management, thus allowing to achieve a better overall time complexity for a given space bound. Here $\zeta = -\frac{1}{2}\log_2(1 - \frac{2(1-y)}{1+\sqrt{1-y}}) = 0.2639\ldots$ where $y = 0.7739\ldots$ is a root of $p(y) = 16y^4 - 80y^3 + 120y^2 - 64y + 9$.

Lemma 4.5 (Costs of the randomized slicer, single target [DLvW20])**.** *Given a list of the* $(4/3)^{d/2+o(d)}$ *shortest vectors of a lattice* $\mathcal{L}$ *and a target* $\mathbf{t} \in \mathbb{R}^d$*, the randomized slicer solves CVP for* $\mathbf{t}$ *in time* T_{slice} *and space* S_{slice}*, with:*

$$T_{\text{slice}} = 2^{\zeta d+o(d)}, \qquad S_{\text{slice}} = (4/3)^{d/2+o(d)}. \tag{4.9}$$

Lemma 4.6 (Costs of the randomized slicer, many targets [DLvW20]). *Given a list of the $(4/3)^{d/2+o(d)}$ shortest vectors of a lattice $\mathcal{L}$ and a batch of $n \geqslant (13/12)^{d/2+o(d)}$ target vectors $\mathbf{t}_1, \dots, \mathbf{t}_n \in \mathbb{R}^d$, the batched randomized slicer solves CVP for all targets $\mathbf{t}_i$ in total time T_{slice} and space S_{slice}, with:*

$$T_{\text{slice}} = n \cdot (18/13)^{d/2+o(d)}, \qquad S_{\text{slice}} = (4/3)^{d/2+o(d)}. \tag{4.10}$$

The amortized time complexity per instance equals $T_{\text{slice}}/n = (18/13)^{d/2+o(d)}$.

4.2.5 – Babai Lifting. Finally, we will revisit the extension to lattice sieving described in [Duc18], based on Babai's nearest plane algorithm [Bab86]. As observed by Ducas, lattice sieving returns much more information about a lattice than just the shortest vector, and this additional information can be used to obtain a few dimensions *for free* – to solve SVP in dimension d, it suffices to run sieving on a sublattice of dimension $d - \ell$ with $\ell = \Theta(d/\log d)$, and use the resulting list of vectors in this sublattice to find the shortest vector in the full lattice.

Lemma 4.7 (Costs of Babai lifting [Duc18]). *Let $\gamma > 1$, let $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_d\}$ be a sufficiently reduced basis of a lattice $\mathcal{L}$, and let $\mathcal{L}' \subset \mathcal{L}$ be the sublattice of $\mathcal{L}$ generated by $\mathbf{B}' = \{\mathbf{b}_1, \dots, \mathbf{b}_{d-\ell}\}$, where:*

$$\ell = \frac{2d \log_2 \gamma}{\log_2 d} \cdot (1 + o(1)). \tag{4.11}$$

Then, given a list L' of the $\gamma^{d+o(d)}$ shortest vectors of $\mathcal{L}'$, we can find a shortest vector of $\mathcal{L}$ through Babai lifting of L' in time T_{lift} and space S_{lift}, with

$$T_{\text{lift}} = \gamma^{d+o(d)}, \qquad S_{\text{lift}} = \gamma^{d+o(d)}. \tag{4.12}$$

For $\gamma = \sqrt{4/3}$ this results in $\ell = d \log_2(4/3)/\log_2 d$ dimensions for free.

4.3 — Sieve, Enumerate, Slice, and Lift

Suppose we have a basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_d\}$ of a lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, and we split it into two disjoint parts as follows, for some choice $0 \leqslant k \leqslant d$:

$$\mathbf{B} = \mathbf{B}_{\text{bot}} \cup \mathbf{B}_{\text{top}}, \quad \mathbf{B}_{\text{bot}} := \{\mathbf{b}_1, \dots, \mathbf{b}_{d-k}\}, \quad \mathbf{B}_{\text{top}} := \{\mathbf{b}_{d-k+1}, \dots, \mathbf{b}_d\}. \tag{4.13}$$

This defines a partition of the lattice $\mathcal{L} = \mathcal{L}_{\text{bot}} \oplus \mathcal{L}_{\text{top}}$ as a direct sum of the two sublattices $\mathcal{L}_{\text{bot}} := \mathcal{L}(\mathbf{B}_{\text{bot}})$ and $\mathcal{L}_{\text{top}} := \mathcal{L}(\mathbf{B}_{\text{top}})$. Let us further denote a solution $\mathbf{s} = \text{SVP}(\mathcal{L})$ as $\mathbf{s} = \mathbf{s}_{\text{bot}} + \mathbf{s}_{\text{top}}$ with $\mathbf{s}_{\text{bot}} \in \mathcal{L}_{\text{bot}}$ and $\mathbf{s}_{\text{top}} \in \mathcal{L}_{\text{top}}$. Finding $\mathbf{s}$ can commonly be described as solving a CVP instance on $\mathcal{L}_{\text{bot}}$:

$$\mathbf{s}_{\text{top}} \neq \mathbf{0} \quad \Longrightarrow \quad \mathbf{s} = \mathbf{s}_{\text{top}} - \text{CVP}(\mathcal{L}_{\text{bot}}, \mathbf{s}_{\text{top}}). \tag{4.14}$$

Note that the case $\mathbf{s}_{\text{top}} = \mathbf{0}$ is in a sense "easy", as then $\mathbf{s} = \text{SVP}(\mathcal{L}_{\text{bot}})$. The hardest problem instances occur when $\mathbf{s}_{\text{top}} \neq \mathbf{0}$, and this will be our main focus.

Lattice enumeration can be viewed as a procedure for solving SVP based on the above observations: first enumerate all target vectors $\mathbf{t} \in \mathcal{L}_{\text{top}}$ that have the potential to satisfy $\mathbf{t} = \mathbf{s}_{\text{top}}$, and then compute $\text{CVP}(\mathcal{L}_{\text{bot}}, \mathbf{t})$ for each of these targets through a continued enumeration procedure on the sublattice $\mathcal{L}_{\text{bot}}$, to see which of them produces the solution to SVP on the full lattice. Observe that lattice enumeration commonly solves each of these CVP instances separately, even though each problem instance can be viewed as a CVP instance on the *same* lattice $\mathcal{L}_{\text{bot}}$, but with a different target vector $\mathbf{t} \in \mathcal{L}_{\text{top}}$.

As previously outlined in e.g. [GNR10, DLdW19], a truly efficient CVPP algorithm would imply a way to speed up processing all these CVP instances in enumeration; one would first run a one-time preprocessing on the sublattice $\mathcal{L}_{\text{bot}}$, and then solve all the CVP instances at some level k using the preprocessed data as input for the CVP(P) oracle. The initial preprocessing step may be expensive, but these costs can be amortized over the many CVP instances that potentially have to be solved during the enumeration phase. At the time of [GNR10] no good heuristic CVPP algorithm was known, but with the results of [DLdW19, Laa19, DLvW20] we may now finally instantiate the above idea with the ingredients from Sections 4.2.2–4.2.4.

4.3.1 – Hybrid 1: Sieve, Enumerate–and–Slice. In the first hybrid, after the preprocessing (sieve) finishes, we compute closest vectors to targets $\mathbf{t} \in \mathcal{L}_{\text{top}}$ one vector at a time. This algorithm has two phases, where the second phase combines enumeration with the randomized slicer.

1. **Sieve**: First, run a lattice sieve on $\mathcal{L}_{\text{bot}}$ to generate a list $L \subset \mathcal{L}_{\text{bot}}$.
2. **Enumerate–and–slice**: Then, run a depth-first enumeration in $\mathcal{L}_{\text{top}}$, where for each leaf $\mathbf{t} \in \mathcal{L}_{\text{top}}$ we run the randomized slicer to find the closest vector $\text{CVP}(\mathbf{t}) \in \mathcal{L}_{\text{bot}}$. We keep track of the shortest difference vector $\mathbf{t} - \text{CVP}(\mathbf{t})$, and ultimately return the shortest one as a candidate solution for $\text{SVP}(\mathcal{L})$.[1]

To optimize the asymptotic time complexity of this algorithm, note that the cost of enumeration in $\mathcal{L}_{\text{top}}$ is $T_{\text{enum}} = 2^{O(k \log d)}$ while the costs of sieving and slicing in $\mathcal{L}_{\text{bot}}$ are $T_{\text{sieve}}, T_{\text{slice}} = 2^{O(d-k)}$. To balance these costs, and minimize the overall time complexity, we will therefore set k as follows:

$$k = \frac{\alpha \cdot d}{\log_2 d}, \qquad \text{with } \alpha > 0 \text{ constant.} \tag{4.15}$$

Using Lemmas 4.3–4.5, optimizing α to obtain the best overall asymptotic time complexity is a straightforward exercise, and we state the result below.

Heuristic result 4.8 (Sieve, enumerate–and–slice). *Let* $k = \alpha d / \log_2 d$ *with*

$$\alpha < \log_2(\tfrac{3}{2}) - 2\zeta = 0.0570\ldots \qquad (\zeta \textit{ as in Lemma 4.5}) \tag{4.16}$$

Let $T_1^{(d)}$ *and* $S_1^{(d)}$ *denote the overall time and space complexities of the sieve, enumerate–and–slice hybrid algorithm in dimension* d*. Then:*

$$T_1^{(d)} = T_{\text{sieve}}^{(d-k)} \cdot (1 + o(1)), \qquad S_1^{(d)} = S_{\text{sieve}}^{(d-k)} \cdot (1 + o(1)). \tag{4.17}$$

Proof. For the time complexities, recall that the costs of the individual parts of the algorithm, by Lemmas 4.3–4.5, are given by:

$$T_{\text{sieve}} = 2^{\frac{1}{2}\log_2(\frac{3}{2})d + o(d)}, \quad T_{\text{enum}} = 2^{\frac{\alpha}{2}d + o(d)}, \quad T_{\text{slice}} = 2^{(\frac{\alpha}{2} + \zeta)d + o(d)}. \tag{4.18}$$

[1] The case $\mathbf{s}_{\text{top}} = \mathbf{0}$ can be handled by checking if L contains an even shorter vector.

Clearly $T_{\text{enum}} = o(T_{\text{slice}})$ since $\zeta > 0$. Now, due to $\alpha < \alpha_0 = \log_2(\frac{3}{2}) - 2\zeta$ being strictly smaller than the point where $T_{\text{sieve}} \approx T_{\text{slice}}$, we have $T_{\text{slice}} = o(T_{\text{sieve}})$ as well, giving a total time complexity of $T = T_{\text{sieve}} \cdot (1 + o(1))$. Finally, looking closely, we note that the cost T_{sieve} actually corresponds to running a standard lattice sieve in dimension $d - k$, which can be done in time $T_{\text{sieve}}^{(d-k)}$ as claimed.

For the space complexities, we recall them from Lemmas 4.3–4.5 as follows:

$$S_{\text{sieve}} = (4/3)^{d/2+o(d)}, \quad S_{\text{enum}} = \text{poly}(d), \quad S_{\text{slice}} = (4/3)^{d/2+o(d)}. \tag{4.19}$$

Since $\alpha < \alpha_0$, the time complexity of the enumerate–and–slice procedure is strictly smaller than the cost of the preprocessing phase, and this will remain true even if we use a slightly smaller list as output from the preprocessing phase. So for sufficiently small $\varepsilon > 0$, we may therefore choose to use a list $L' \subset L$ for the enumerate–and–slice phase of size $|L'| = |L|^{1-\varepsilon}$, while still maintaining a time complexity $T_{\text{slice}} = o(T_{\text{sieve}})$. This guarantees that the overhead caused by the quasilinear-space nearest neighbor data structure, required in the third phase to achieve sublinear search costs, does not impose any overhead in the asymptotic space complexity; the memory required in the third phase will then be of size $S_{\text{slice}} = (S_{\text{sieve}}^{1-\varepsilon})^{1+o(1)} = o(S_{\text{sieve}})$. □

Letting $\alpha \to \log_2(\frac{3}{2}) - 2\zeta$ in the above result, we get $k \approx 0.0570d/\log_2 d$ with an asymptotic speedup of a factor $2^{0.0167d/\log_2 d}$ and a memory reduction of a factor $2^{0.0118d/\log_2 d}$ compared to running a sieve directly on $\mathcal{L}$. Note that the result does not hide subexponential or even polynomial hidden order terms; the time and space complexities are dominated by the preprocessing costs.

For the heuristic results in the following subsections 4.3.2-4.3.4, analogous derivations can be given to argue that both the time and space complexities are dominated by the initial sieving phase, as long as the parameters k (and ℓ) are below the point where the sieving and slicing (and lifting) become equally expensive. Further note that although the batched slicer has a cost of $(3/2)^{d/2+o(d)} + n \cdot (18/13)^{d/2+o(d)}$ for n targets due to the reinitializations of the costly nearest neighbor data structures [DLvW20], these costs can again be made to be $(3/2-\varepsilon)^{d/2+o(d)} + n^{1-\varepsilon} \cdot (18/13)^{d/2+o(d)}$ by slightly reducing the number of targets and the number of hash tables accordingly.

4.3.2 – Hybrid 2: Sieve, Enumerate, Slice. An alternative to the above approach is to separate the enumeration and slicing procedures into two disjoint parts, and run the hybrid algorithm in three phases. The benefit of this approach (cf. Section 4.2.4) is that the *batched* slicer can then be used to achieve better amortized complexities for CVPP.

1. **Sieve**: As before, run a lattice sieve on $\mathcal{L}_{\text{bot}}$, to generate a list $L \subset \mathcal{L}_{\text{bot}}$.
2. **Enumerate**: Then, enumerate all nodes $\mathbf{t} \in \mathcal{L}_{\text{top}}$ at depth k in the enumeration tree, and store them in a list of targets $T \subset \mathcal{L}_{\text{top}}$.
3. **Slice**: Finally, use the batched randomized slicer with the list L to solve CVP on $\mathcal{L}_{\text{bot}}$ for all targets $\mathbf{t} \in T$, and return the shortest vector $\mathbf{t} - \text{CVP}(\mathbf{t})$.

Asymptotically, the additional space required for storing the nodes from the enumeration phase will not play a large role, compared to the memory required for storing the output from the preprocessing phase. On the other hand, by using the improved batch-CVPP slicer of Lemma 4.6 we can use nearest neighbor searching more efficiently, without increasing the memory, leading to a bigger improvement over standard sieving than with the first hybrid algorithm.

Heuristic result 4.9 (Sieve, enumerate, slice)**.** *Let* $k = \alpha d / \log_2 d$ *with*

$$\alpha < \log_2(\tfrac{13}{12}) = 0.1154\ldots. \tag{4.20}$$

Let $T_2^{(d)}$ *and* $S_2^{(d)}$ *denote the overall time and space complexities of the batched sieve, enumerate, slice hybrid algorithm in dimension* d*. Then:*

$$T_2^{(d)} = T_{\text{sieve}}^{(d-k)} \cdot (1 + o(1)), \qquad S_2^{(d)} = S_{\text{sieve}}^{(d-k)} \cdot (1 + o(1)). \tag{4.21}$$

In the limit of $\alpha \to \log_2(\frac{13}{12})$ we get $k \approx 0.1154 d / \log_2 d$ dimensions *for free*, leading to an asymptotic speedup of a factor $2^{0.0338 d / \log_2 d + o(d/\log d)}$ and a memory reduction of a factor $2^{0.0240 d / \log_2 d + o(d/\log d)}$ over direct sieving on $\mathcal{L}$.

4.3.3 – Hybrid 3: Sieve, Enumerate–and–Slice, Lift. For the third and fourth hybrids, we observe that similar to lattice sieving, the slicer in the previous hybrid algorithms can actually produce much more information about the lattice than just the shortest lattice vector; for other targets $\mathbf{t} \neq \mathbf{s}_{\text{top}}$, as well as for "failed" outputs of the randomized slicer, the slicer will also return many short lattice vectors. This suggests that to get even more dimensions *for free*, we may be able to combine both hybrids with Babai lifting as outlined in Lemma 4.7.

Instead of splitting the lattice into two parts, we now split the input lattice basis into three parts $\mathbf{B} = \mathbf{B}_{\text{bot}} \cup \mathbf{B}_{\text{mid}} \cup \mathbf{B}_{\text{top}}$, where the three bases $\mathbf{B}_{\text{bot}} := \{\mathbf{b}_1, \ldots, \mathbf{b}_\ell\}$, $\mathbf{B}_{\text{mid}} := \{\mathbf{b}_{\ell+1}, \ldots, \mathbf{b}_{d-k}\}$, and $\mathbf{B}_{\text{top}} := \{\mathbf{b}_{d-k+1}, \ldots, \mathbf{b}_d\}$ generate lattices $\mathcal{L}_{\text{bot}}, \mathcal{L}_{\text{mid}}, \mathcal{L}_{\text{top}}$ of dimensions ℓ, $d - k - \ell$ and k respectively. For Hybrid 3 we essentially run Hybrid 1 on $\mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$, and use Babai lifting to deal with the additional ℓ dimensions of $\mathcal{L}_{\text{bot}}$. This leads to the following algorithm:

1. **Sieve**: Run a lattice sieve on $\mathcal{L}_{\text{mid}}$ to generate a list $L \subset \mathcal{L}_{\text{mid}}$.
2. **Enumerate–and–slice**: Enumerate all nodes $\mathbf{t} \in \mathcal{L}_{\text{top}}$, and repeatedly slice each of them with the list L to find close vectors $\mathbf{v} \in \mathcal{L}_{\text{mid}}$. For each pair $\mathbf{t}, \mathbf{v}$ add the vector $\mathbf{t} - \mathbf{v}$ to an output list $S \subset \mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$.
3. **Lift**: Finally, extend each vector $\mathbf{s}' \in S$ to a candidate solution $\mathbf{s} \in \mathcal{L}$ by running Babai's nearest plane algorithm. Return the shortest lifted vector.

As the slicer processes $E_k = d^{k/2+o(k)} = 2^{\alpha d/2 + o(d)}$ target vectors, and requires $\rho = (16/13)^{d/2+o(d)}$ rerandomizations per target for average-case CVP to succeed (see [DLdW19, DLvW20] for details), the slicer outputs $2^{(\alpha + \log_2(16/13)) \cdot d/2 + o(d)}$ lattice vectors, and ideally we might hope this list contains, similar to sieving [Duc18], (almost) all lattice vectors of norm at most $\gamma = 2^{(\alpha + \log_2(16/13))/2 + o(1)} \cdot \text{gh}(\mathcal{L})$.

Assumption 4.10 (Hybrid assumption[2])**.** *The list* S*, output by the slicer, contains the* $2^{(\alpha + \log_2(16/13)) \cdot d/2 + o(d)}$ *shortest lattice vectors of* $\mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$*.*

Assuming that the above heuristic is indeed valid, we derive the following result regarding the asymptotic time and space complexities of the described hybrid algorithm. In Section 4.5 we will revisit this assumption, to study its validity.

[2]After the paper on which this chapter is based on was published, Léo Ducas and Wessel van Woerden informed us that counterexamples to Assumption 4.10 can be found where S only contains at most an exponentially small fraction of the shortest vectors of $\mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$. As a result, our results relying on Assumption 4.10 should be seen as optimistic, best-case lower bounds on the true algorithm complexities.

Heuristic result 4.11 (Sieve, enumerate–and–slice, lift). *Let* $k = \alpha d/\log_2 d$ *and* $\ell = \beta d/\log_2 d$ *with*

$$\alpha < \log_2(\tfrac{3}{2}) - 2\zeta = 0.0570\ldots, \qquad \beta < \log_2(\tfrac{24}{13}) - 2\zeta = 0.3565\ldots. \tag{4.22}$$

Let $T_3^{(d)}$ *and* $S_3^{(d)}$ *denote the time and space complexities of the sieve, enumerate–and–slice, lift hybrid algorithm in dimension* d*. Then, under Assumption 4.10:*

$$T_3^{(d)} = T_{\text{sieve}}^{(d-k-\ell)} \cdot (1 + o(1)), \qquad S_3^{(d)} = S_{\text{sieve}}^{(d-k-\ell)} \cdot (1 + o(1)). \tag{4.23}$$

Observe that the number of dimensions we save compared to a full sieve here is $k+\ell \approx 0.4136d/\log_2 d$. Compared to the result of Ducas [Duc18] of $\ell \approx 0.4150d/\log_2 d$ this new hybrid is asymptotically slightly worse than a sieve–and–lift hybrid.

4.3.4 – Hybrid 4: Sieve, Enumerate, Slice, Lift. Finally, combining the second hybrid with lifting, as in the third hybrid algorithm above, results in the following optimized hybrid procedure:

1. **Sieve**: Run a lattice sieve on $\mathcal{L}_{\text{mid}}$ to generate a list $L \subset \mathcal{L}_{\text{mid}}$.
2. **Enumerate**: Enumerate all nodes $\mathbf{t} \in T \subset \mathcal{L}_{\text{top}}$ at depth k in $\mathcal{L}$.
3. **Slice**: Run the slicer, with the list L as input, to find close vectors in $\mathcal{L}_{\text{mid}}$ to the targets $\mathbf{t} \in T$. The result is a list $S \subset \mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$.
4. **Lift**: Finally, extend each vector $\mathbf{s}' \in S$ to a candidate solution $\mathbf{s} \in \mathcal{L}$ by running Babai's nearest plane algorithm. Return the shortest lifted vector.

Not only does splitting the enumeration and slicing guarantee that the batched version of the slicer gets better complexities; the smaller resulting value α also means that the number of vectors output by the slicer is larger, which leads to more dimensions for free from the lifting phase. In particular, with the batched slicer the number of vectors output by the slicer is proportional to $(4/3)^{d/2+o(d)}$, and we may get as many dimensions for free in the lifting phase as [Duc18].

Heuristic result 4.12 (Sieve, enumerate, slice, lift). *Let* $k = \alpha d/\log_2 d$ *and* $\ell = \beta d/\log_2 d$ *with*

$$\alpha < \log_2(\tfrac{13}{12}) = 0.1154\ldots, \qquad \beta < \log_2(\tfrac{4}{3}) = 0.4150\ldots. \tag{4.24}$$

Let $T_4^{(d)}$ *and* $S_4^{(d)}$ *denote the time and space complexities of the sieve, enumerate, slice, and lift hybrid algorithm in dimension* d*. Then, under Assumption 4.10:*

$$T_4^{(d)} = T_{\text{sieve}}^{(d-k-\ell)} \cdot (1 + o(1)), \qquad S_4^{(d)} = S_{\text{sieve}}^{(d-k-\ell)} \cdot (1 + o(1)). \tag{4.25}$$

We again stress that the above result relies on a *batched* version of the randomized slicer. With this batched hybrid algorithm with lifting, assuming the hybrid assumption holds, we can potentially get up to $k + \ell \approx 0.5305d/\log_2(d)$ dimensions *for free*, which would improve upon Ducas' $\ell \approx 0.4150d/\log_2(d)$ [Duc18].

An overview of the techniques used in the four hybrids, as well as the number of dimensions for free in each algorithm, is given in Table 4.1. Figure 4.1 presents graphical overviews of the hybrid algorithms described in this section where the horizontal axis depicts the basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d$ and the vertical axis corresponds to the time (with algorithms starting from the top and ending at the bottom).

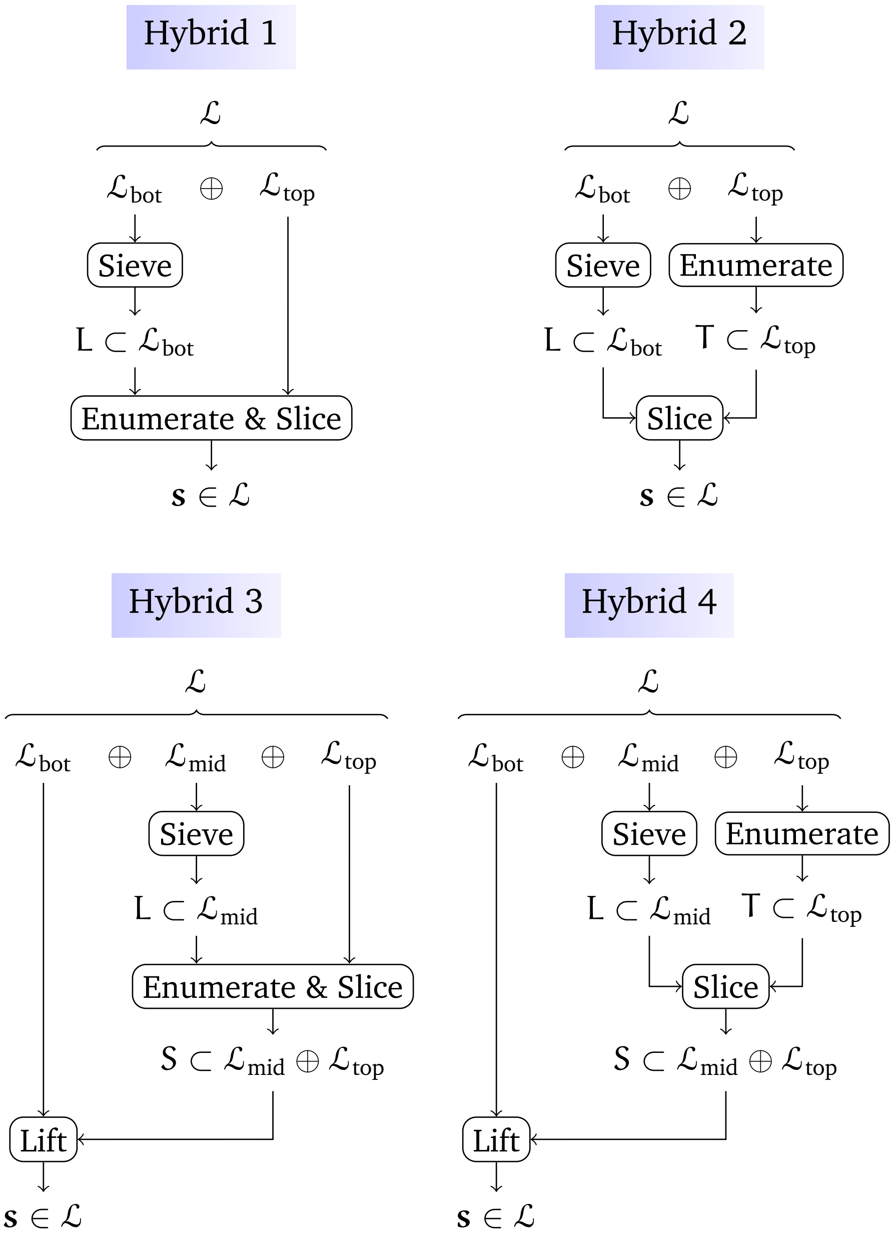

Hybrid 1
$\mathcal{L}$
$\mathcal{L}_{\text{bot}} \oplus \mathcal{L}_{\text{top}}$
Sieve
$L \subset \mathcal{L}_{\text{bot}}$
Enumerate & Slice
$\mathbf{s} \in \mathcal{L}$
Hybrid 2
$\mathcal{L}$
$\mathcal{L}_{\text{bot}} \oplus \mathcal{L}_{\text{top}}$
Sieve
Enumerate
$L \subset \mathcal{L}_{\text{bot}}$
$T \subset \mathcal{L}_{\text{top}}$
Slice
$\mathbf{s} \in \mathcal{L}$
Hybrid 3
$\mathcal{L}$
$\mathcal{L}_{\text{bot}} \oplus \mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$
Sieve
$L \subset \mathcal{L}_{\text{mid}}$
Enumerate & Slice
$S \subset \mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$
Lift
$\mathbf{s} \in \mathcal{L}$
Hybrid 4
$\mathcal{L}$
$\mathcal{L}_{\text{bot}} \oplus \mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$
Sieve
Enumerate
$L \subset \mathcal{L}_{\text{mid}}$
$T \subset \mathcal{L}_{\text{top}}$
Slice
$S \subset \mathcal{L}_{\text{mid}} \oplus \mathcal{L}_{\text{top}}$
Lift
$\mathbf{s} \in \mathcal{L}$

Figure 4.1: A high-level description of the hybrid algorithms presented in section 4.3. Hybrids 1 and 3 combine enumeration and slicing, performing the randomized slicing procedure for *only one* target vector at a time. Hybrids 3 and 4 use the Babai lifting technique from [Duc18]. The asymptotics of the slicer depend on whether targets are processed directly (left) or in batches (right). The lifting can be done directly as well, without affecting the performance of the algorithm.

Table 4.1: An overview of the techniques used in the hybrids, as well as the asymptotic number of dimensions *for free* for each part and in total (last column). In sufficiently high dimensions, under Assumption 4.10, Hybrid 4 outperforms all other algorithms, by saving up to $0.53 d / \log_2 d$ dimensions compared to sieving in the full lattice.

Algorithm	**Sieve**	**Enum./Slice**		**Lift**	**Dimensions for free**		
		(Single)	(Batch)		$(\frac{k}{d} \log_2 d)$	$(\frac{\ell}{d} \log_2 d)$	$(\frac{k+\ell}{d} \log_2 d)$
Full sieve [BDGL16]	✓				-	-	-
Hybrid 1	✓	✓			0.0570	-	0.0570
Hybrid 2	✓		✓		0.1154	-	0.1154
Hybrid 3	✓	✓		✓	0.0570	0.3566	0.4136
SubSieve [Duc18]	✓			✓	-	0.4150	0.4150
Hybrid 4	✓		✓	✓	0.1155	0.4150	**0.5305**

4.4 — Sieve, Enumerate, Slice, Repeat

For the fourth hybrid, under Assumption 4.10 the enumeration and batched slicer together take as input a list of all vectors of norm at most $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}')$ of a suitable sublattice $\mathcal{L}' \subset \mathcal{L}$, and output (almost) all lattice vectors of norm at most $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$ of $\mathcal{L}$. This suggests one might replace the initial sieving step on $\mathcal{L}_{\mathrm{mid}}$ by a sieve, enumerate, slice hybrid (Hybrid 2), by splitting $\mathcal{L}_{\mathrm{mid}} = \mathcal{L}_{\mathrm{mid}}^{(1)} \oplus \mathcal{L}_{\mathrm{mid}}^{(2)}$ with $\mathrm{rank}(\mathcal{L}_{\mathrm{mid}}^{(2)}) = \Theta(d/\log d)$; running a sieve on $\mathcal{L}_{\mathrm{mid}}^{(1)}$; enumerating $\mathcal{L}_{\mathrm{mid}}^{(2)}$; and then using the slicer to find a list of short vectors $L \subset \mathcal{L}_{\mathrm{mid}}$. Under Assumption 4.10, this substitution of the initial sieve by Hybrid 2 can be repeated many times to obtain $\Theta(d/\log d)$ dimensions for free several times.

As an alternative interpretation, rather than running enumeration on k levels directly, one additional level of nesting suggests we first run the lower $k/2$ levels of enumeration, lift the resulting target vectors to obtain short vectors in a lattice of rank $d - k/2$, and then run another $k/2$ levels of enumeration to find short vectors in the full lattice. Splitting up the enumeration this way decreases the overall enumeration costs and the number of targets for the slicing phases ($E_{k/2} + E_{k/2} \ll E_k$), but at the same time the list output by the first slicing phase might not be as good for the second slicing phase as what one would get from running a sieve directly; even if Assumption 4.10 is true, likely this still comes at a slight loss in the quality of the list, say in the first order terms.

We finally observe that the same idea of nesting does not seem to work for the sieve, lift hybrid of [Duc18]. Although one could define a “generalized” Babai lifting procedure, lifting targets to all nearby vectors in the higher-rank lattice, from a viewpoint of enumeration we are “missing” some branches in the tree due to L only containing some nodes in the tree at level $d - \ell$. Therefore, if the shortest vector in the lattice is actually in one of those missing branches, then a generalized lifting procedure will not succeed in finding this shortest vector.

Although we will briefly revisit the idea of nesting in the experiments in the next section, we leave a technical study of nesting for future work.

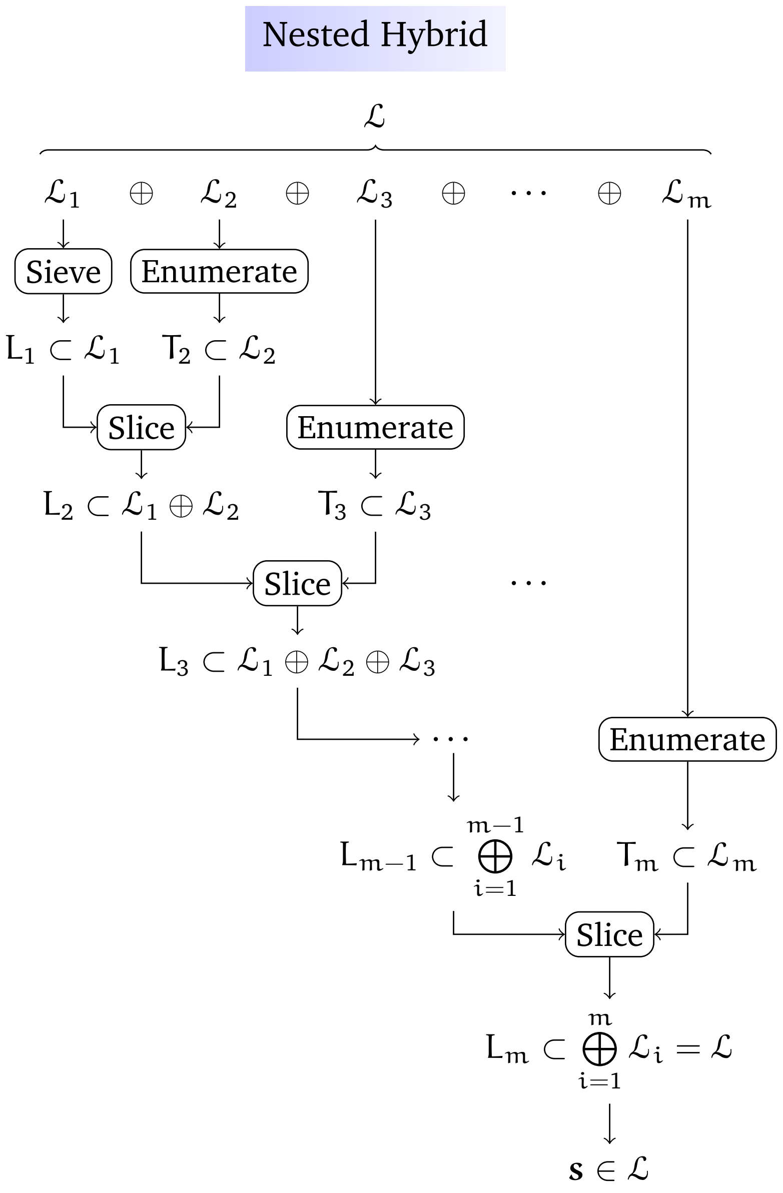

Nested Hybrid
$\mathcal{L}$
$\mathcal{L}_1 \oplus \mathcal{L}_2 \oplus \mathcal{L}_3 \oplus \cdots \oplus \mathcal{L}_m$
Sieve
Enumerate
$L_1 \subset \mathcal{L}_1$
$T_2 \subset \mathcal{L}_2$
Slice
Enumerate
$L_2 \subset \mathcal{L}_1 \oplus \mathcal{L}_2$
$T_3 \subset \mathcal{L}_3$
Slice
$\cdots$
$L_3 \subset \mathcal{L}_1 \oplus \mathcal{L}_2 \oplus \mathcal{L}_3$
$\cdots$
Enumerate
$L_{m-1} \subset \bigoplus_{i=1}^{m-1} \mathcal{L}_i$
$T_m \subset \mathcal{L}_m$
Slice
$L_m \subset \bigoplus_{i=1}^{m} \mathcal{L}_i = \mathcal{L}$
$\mathbf{s} \in \mathcal{L}$

Figure 4.2: A high-level description of the potential recursive hybrid algorithm, which starts on a lattice $\mathcal{L}_1$ of dimension $d - \Theta(d/\log d)$, and then repeatedly lifts the lists $L_i \subset \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_i$ to lists $L_{i+1} \subset \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_{i+1}$ by enumerating targets $T_{i+1} \subset \mathcal{L}_{i+1}$ and using the batched slicer with L_i as input to create L_{i+1}. Each lattice $\mathcal{L}_i$ for $i > 1$ has dimension $\Theta(d/\log d)$.

4.5 — Experimental Results

4.5.1 – Verifying Assumption 4.10. To attempt to validate (or disprove) the new heuristic assumption, we performed the following experiment. We used the 60-dimensional SVP challenge lattice with seed 0 [svp19], pre-reduced with BKZ-50 [Sch87], for which $\mathrm{gh}(\mathcal{L}) \approx 2001$ and $\lambda_1(\mathcal{L}) \approx 1943$. The black dashed line in Figure 4.3 shows the expected number of lattice points below a certain norm by the Gaussian heuristic (Assumption 1.18). The (barely visible) purple line intersecting this line for high norms shows the actual number of lattice vectors found by a "relaxed" sieve [Laa16b], showing the accuracy of the Gaussian heuristic for large balls.

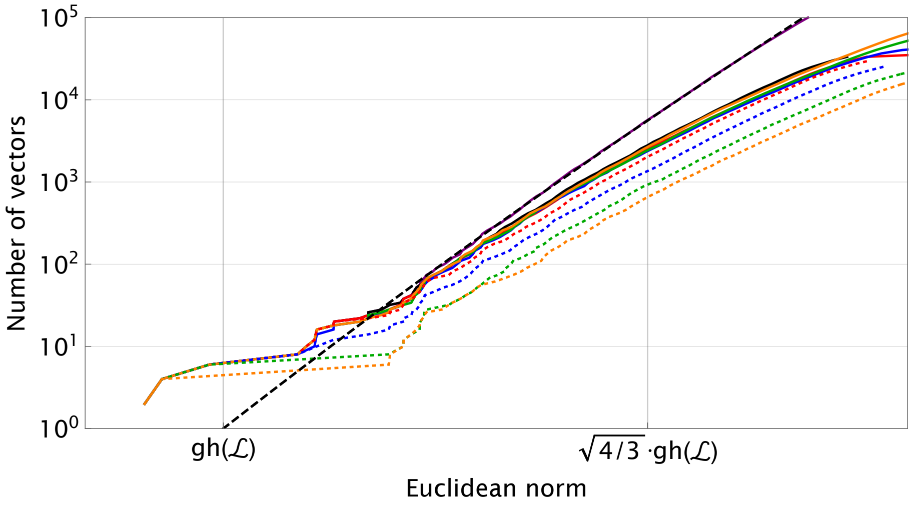


Figure 4.3: The number of vectors found through a sieve (black) and sieve, enumerate, slice hybrids for $k \in \{1, 2, 3, 4\}$ (orange, green, blue, red) in dimension 60. The dashed black line, and the purple line intersecting it for large norms, indicate the true number of lattice vectors below this norm. The dashed colored lines indicate the lists obtained from running sieving in sublattices of rank $d - k$.

To test Assumption 4.10, we then ran both a standard g6k lattice sieve to produce a list L_0 (black) [ADH+19]; and sieve, enumerate, slice hybrids for $k \in \{1, 2, 3, 4\}$ by (1) running g6k on the $(d - k)$-dimensional sublattice formed by $\mathbf{b}_1, \dots, \mathbf{b}_{d-k}$ to produce a list L_k, (2) running enumeration up to depth k in the full lattice to obtain targets T_k, (3) slicing each target $\mathbf{t} \in T_k$ up to $20 \cdot (16/13)^{(d-k)/2}$ times, to obtain a list S_k, and (4) plotting the sorted norms of both L_k (dashed) and $S_k \cup L_k$ (solid) in Figure 4.3. These results show that (i) as expected, the preprocessed lists L_k in rank $d - k$ become increasingly poor approximations of the sieved list L_0 as k increases, and (ii) the sliced lists $S_k \cup L_k$ together form very good approximations to the sieved list L_0. Note that, at norm $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$, all these lists are quite far off from the prediction by the Gaussian heuristic.

Table 4.2: Experimental results and estimates for the costs of the hybrid algorithms, in dimensions $d \in \{60, 65, 70, 75, 80\}$ and for parameter choices $k \in \{0, 1, 2, 3\}$ as well as the nested hybrid with two iterations of $k = 1$. Single-core timings are denoted in milliseconds (ms), seconds (s), minutes (m), hours (h), and days (d). List sizes $|L|$ and estimates on the required number of rerandomizations p_{iter}^{-1} are sometimes given in multiples of one thousand (k). The last column gives estimates for the total time complexities for these algorithms, by adding up the costs for BKZ, sieving, enumeration, and slicing. The case $k = 0$ corresponds to running a sieve on the full lattice directly.

Parameters		**BKZ**	**— Sieve —**		**— Enum —**			**— Slice —**		**Total**
d	k	$T_{\text{BKZ}}^{(d-10)}$	$\|L\|$	$T_{\text{sieve}}^{(d-k)}$	$\|T\|$	$T_{\text{enum}}^{(k)}$	$T_{\text{iter}}^{(d-k)}$	p_{iter}^{-1}	$T_{\text{slice}}^{(d-k)}$	$T_{\text{hyb}}^{(d)}$
60	0	4s	18k	19s	-	-	-	-	-	23s
	1	4s	16k	16s	5	0s	3.2ms	830	13s	33s
	2	4s	13k	12s	30	0s	2.7ms	530	43s	59s
	3	4s	12k	9s	155	0s	2.4ms	760	280s	293s
	1+1	4s	13k	12s	4	0s	3.0ms	500	6s	51s
			(16k)	(0s)	5	0s	3.2ms	1820	29s	
65	0	8s	37k	78s	-	-	-	-	-	1m
	1	8s	32k	57s	5	0s	6.8ms	12.5k	7m	8m
	2	8s	28k	44s	37	0s	6.6ms	2.9k	12m	13m
	3	8s	24k	36s	215	0s	5.6ms	2.9k	58m	59m
	1+1	8s	28k	44s	4	0s	6.6ms	1.1k	0.5m	6m
			(32k)	(0s)	5	0s	6.8ms	6.7k	4m	
70	0	1m	76k	5m	-	-	-	-	-	6m
	1	1m	65k	4m	6	0m	20ms	17k	35m	40m
	2	1m	57k	3m	46	0m	16ms	1k	12m	16m
	3	1m	49k	2m	293	0m	13ms	6k	381m	384m
	1+1	1m	57k	3m	5	0m	15ms	2k	2m	43m
			(65k)	(0m)	5	0m	18ms	25k	37m	
75	0	2m	155k	22m	-	-	-	-	-	0.4h
	1	2m	134k	16m	6	0m	40ms	25k	2h	2h
	2	2m	116k	11m	50	0m	48ms	20k	13h	14h
	3	2m	101k	8m	366	0m	30ms	12k	37h	37h
	1+1	2m	116k	11m	5	0m	35ms	4k	0.2h	>8h
			(134k)	(0m)	6	0m	41ms	>100k	>7h	
80	0	14m	320k	74m	-	-	-	-	-	1.5h
	1	14m	275k	58m	7	0m	95ms	>100k	>18h	>20h
	2	14m	240k	45m	64	0m	74ms	>50k	>66h	>67h
	3	14m	205k	36m	506	0m	66ms	>50k	>19d	>19d

4.5.2 – Assessing the Sieve, Enumerate–and–Slice Hybrid. To study the practical performance of these hybrid algorithms, we performed some preliminary experiments in dimensions 60–80, whose results are described in Table 4.2. We describe the setup of the experiments, and discuss the results as well as conclusions that can or cannot be drawn from these results.

BKZ. To start, we used the SVP challenge lattices [svp19] with seed 0 in dimensions $d \in \{60, 65, 70, 75, 80\}$. We preprocessed each basis with BKZ with block size $d - 10$. In case the shortest vector had a 0-coefficient for $\mathbf{b}_d$ when expressed in terms of $\mathbf{B}$, we would rerandomize the basis and run BKZ again, to guarantee that the preprocessed lists do not already contain the solution.

Sieve. Next, we used the g6k [The19b] framework to generate sieving lists in dimensions $d - k$, for $k = 0, 1, 2, 3$. We disabled the “dimensions for free” from g6k, to test the pure hybrids for their performance and limit the impact of other factors for now. The case $k = 0$ corresponds to sieving in the full lattice, and the timings in dimensions $d - k$

clearly decrease with k, as shown in Table 4.2. The resulting vectors were stored in an output file, and their sizes are also given in Table 4.2.

Enumerate. Then, we ran a full enumeration in the full lattice up to depth k, to generate the target vectors for the slicer. These were again stored in a separate file for later usage. Note that pruning would reduce the number of targets further, but (1) this would decrease the success probability of the overall algorithm, and (2) rerandomizing the lattice basis to get a high success probability would (naively) require running the costly sieving preprocessing step several times. We therefore restricted experiments to enumeration without pruning.

Slice. Finally, with the sieved list L and target vectors T as input, we identified the target $\mathbf{t} \in T$ corresponding to the shortest vector in the lattice, and for this target we ran the randomized slicer with 10^5 trials to estimate the success probability p_{iter} of the slicer in finding the shortest vector. Table 4.2 shows the inverse p_{iter}^{-1} as well as the average time for each trial, which together with $|T|$ can then be used to estimate the time for the slicing as $T_{\text{slice}} \approx |T| \cdot p_{\text{iter}}^{-1} \cdot T_{\text{iter}}$.

Nested hybrid. We also tested a simple nested hybrid from Section 4.4, with two successive (non-batched) enumerate–and–slice routines in dimension $k = 1$. In the first slicing phase, we chose the total number of iterations such that the size of the output list matches the size of a directly sieved list for $k = 1$. The rows $k = 1 + 1$ in Table 4.2 suggest this approach compares favorably to $k = 2$.

Conclusions. Although the results in Table 4.2 mainly suggest that these hybrid approaches may have a large overhead in practice, we stress that as d grows, the time complexity grows slower than a full sieve. Furthermore, for the slicer we did not use nearest neighbor techniques or batching to reduce the query times. Also, note that as $0.11d/\log_2 d < 2$ for $d < 128$ we do not expect to obtain many (additional) dimensions *for free* in dimensions $60 \leqslant d \leqslant 80$. The aforementioned reasons can provide some insight why the speedup was not observed in practice in our experiments.

Code in fplll. As part of this project, we implemented the iterative slicer in `fplll` [The19a], and we expect this code to be included in the library soon.

Chapter 5

Gambling at the Ring's casino: The zeros-guessing game

This chapter contains joint work with Benne de Weger which has not yet been published and was inspired by a talk of Daniel J. Bernstein and Tanja Lange.[1]

5.1 — Introduction

For a random lattice $\mathcal{L}$, short vectors do not seem to possess any special property apart from being short. In particular, for lattices without an exceptionally small determinant, no assumption can be made on how many small or even zero coordinates they may have. On the other hand, in lattice-based cryptography it is often the case that the secret key is a short vector with relatively many small or zero coordinates (e.g see [HPS98]). In many cases an attacker will be looking for the secret key as a short vector or part of a short vector in a lattice (e.g. see [BG14]). Due to its construction, this short vector will contain some unusually small coordinates. This extra knowledge could serve as an extra tool in the hands of the attacker.

For example, May and Silverman [MS01] discuss the option of guessing a number of coordinates, mainly coordinates of the short secret vector which will be zero. By guessing some of these coordinates, an attacker could decrease the rank of the used lattice, leading to a potentially faster attack. The efficiency of this method depends on the success probability of guessing a pattern correctly and the time complexity of the algorithm used in order to retrieve a short lattice vector. In this chapter we will focus on the "guessing of patterns".

The guessing technique is even more effective for lattices which possess a cyclic structure, meaning that all rotations of a lattice vector also belong to the lattice. Such type of lattices occur for example in the study of the NTRU problem [HPS98] or the Ring-LWE problem (see [BCLv19]), making this a very interesting case study. For the rest of this chapter we focus on such lattices with an underlying cyclic structure.

Let r be the number of zeros to be guessed, which we will take as fixed. We consider a pattern as a set of indices which presumably point to zero coordinates of a vector. For example, by referring to the pattern $\{1, 2, \ldots, r\}$ it is implied that the first up to r-

[1] https://simons.berkeley.edu/talks/successive-minima-type-inequalities

th coordinate of a vector are being guessed to be zero. As a guess is made, a "success probability" of guessing correctly can be related. Also, a specific pattern could appear once or more than once in a vector if rotations of the vector are allowed. This fact is referred as distinct and multiple winners respectively. In [MS01] the following open problems were mentioned regarding the option of guessing a number of zeros in a secret vector $\mathbf{s}$:

1. Does the pattern $\{1, 2, \dots, r\}$ give the lowest success probability among all patterns of length r?
2. Which patterns of length r achieve the highest success probability among all patterns of length r?
3. Among patterns of length r, does the pattern $\{1, 2, \dots, r\}$ of consecutive zeros give the fewest distinct winners?
4. What is the average number of distinct winners over all patterns of length r?
5. What pattern (or patterns) gives the maximum number of distinct winners?

5.1.1 – Contributions. In this chapter we examine combinatorial aspects of guessing a pattern of zeros in a secret vector as discussed by May and Silverman in [MS01]. We start by providing further insight into the experimental observation stating that random patterns seem to behave better than the consecutive zeros pattern. We define what we call the *index of symmetry* of a pattern. We discuss the relevance of this notion to some open problems mentioned in [MS01] as well as its use in aiding the choice of a "good" pattern.

5.2 — The consecutive zeros pattern

One of the first questions which emerges when one considers the guessing of zeros game is if the consecutive zeros pattern is a good choice or not. In [MS01] it is mentioned that experimentally it appears that random patterns possess a higher success probability over the consecutive pattern. However this is only an experimental observation. We attempt to make this observation more clear by adding a proven result. We start by computing the success probability of the consecutive zeros pattern $\{1, 2, \dots, r\}$ in a binary vector.

Definition 5.1. *Let* $N \in \mathbb{Z}_{>0}$ *and* $\mathbf{x} \in \mathbb{Z}_2^N$. *We denote by* $\mathbf{x}^{(k)}$ *the* k*-fold right rotation of* $\mathbf{x}$.

Definition 5.2. *Let* $N, d \in \mathbb{Z}_{>0}$ *with* $N \geqslant d$. *We define*

$$B_N(d) := \{\mathbf{x} \in \mathbb{Z}_2^N \text{ with exactly } d \text{ ones}\}.$$

We will use the notation $[n]$ for the set of integers $\{0, 1, \dots, n-1\}$.

Definition 5.3. *Let* $N, d \in \mathbb{Z}_{>0}$ *with* $N \geqslant d$ *and* $\mathbf{a} \in B_N(d)$. *We define*

$$\mathrm{Pat}(\mathbf{a}) := \{S \subseteq [N] \mid \exists k \in [N] \text{ s.t. } \mathbf{a}^{(k)}_{s_i} = 0 \; \forall s_i \in S\}$$

Proposition 5.4. *Let* $N, d, r \in \mathbb{Z}_{>0}$ *with* $r \leqslant N - d$ *and* $\mathbf{a} \in B_N(d)$. *Then*

$$\Pr([r] \in \mathrm{Pat}(\mathbf{a})) = \begin{cases} 1 & \text{if } r < \lceil N/d \rceil \\ \dfrac{1}{\binom{N-1}{d-1}} \displaystyle\sum_{l=1}^{\lfloor \frac{N-d}{r} \rfloor} \binom{d}{l}\binom{N-lr-1}{d-1}(-1)^{l+1} & \text{if } r \geqslant \lceil N/d \rceil. \end{cases}$$

Proof. Case $r < \lceil N/d \rceil$: Let $\mathbf{a} \in B_N(d)$. As $[r] \in \mathrm{Pat}(\mathbf{a})$ requires at least one rotation of $\mathbf{a}$ to contain r consecutive zeros in positions $0, \dots, r-1$, we consider the vector $\mathbf{a}$ as a placement of $N-d$ 0s and d 1s on a circle with N positions. That implies that there will always exist a block of consecutive zeros with size $\lceil (N-d)/d \rceil = \lceil N/d \rceil - 1$. Assume on the contrary that each substring has at most $r - 1 \leqslant \lceil N/d \rceil - 2$ zeros. Together with the d bounding 1s this covers at most $d(\lceil N/d \rceil - 1) < N$ positions out of N contradicting the assumption.
Case $r \geqslant \lceil N/d \rceil$: In this case it is not always guaranteed that $[r] \in \mathrm{Pat}(\mathbf{a})$. Let $\mathbf{a} \in B_N(d)$, then

$$\Pr([r] \in \mathrm{Pat}(\mathbf{a})) = \frac{|\{\mathbf{v} \in B_N(d) \text{ s.t. } [r] \in \mathrm{Pat}(\mathbf{v})\}|}{|B_N(d)|}.$$

If for $\mathbf{a} \in B_N(d)$ it holds that $[r] \in \mathrm{Pat}(\mathbf{a})$, the same holds for all $\mathbf{a}^{(k)}$. We define the equivalence relation $\sim$ in $B_N(d)$ by $\mathbf{a} \sim \mathbf{b}$ iff $\exists k$ such that $\mathbf{b} = \mathbf{a}^{(k)}$. We denote by M a set of representatives[2] of $B_N(d)/\sim$. Thus we get

$$\begin{aligned}\Pr([r] \in \mathrm{Pat}(\mathbf{a})) &= N\frac{|\{\mathbf{v} \in M \text{ s.t. } [r] \in \mathrm{Pat}(\mathbf{v})\}|}{|B_N(d)|} \\ &= \frac{N}{\binom{N}{d}}|\{\mathbf{v} \in M \text{ s.t. } [r] \in \mathrm{Pat}(\mathbf{v})\}|.\end{aligned}$$

We are left with the task to count the elements in the set $A \coloneqq \{\mathbf{v} \in M \text{ s.t. } [r] \in \mathrm{Pat}(\mathbf{v})\}$. We do consider again vectors in $B_N(d)$ as a placement of $N-d$ 0s and d 1s on a circle with N positions. For a vector $\mathbf{a} \in B_N(d)$ to belong to A it should possess at least one block of at least r zeros.
We first consider the "base case" $\lfloor (N-d)/r \rfloor = 1$. In this case having more than one block of at least r zeros is not possible, and thus a vector in A will contain exactly one block of at least r zeros. A way to count the number of these vectors is the following. Reserve r zeros, then there are $N-d-r$ zeros and d ones to be placed on a circle of length $N-r$. There are $\binom{N-r}{d}$ ways to do that. As we are interested in choices which are rotation invariant we get $\frac{1}{N-r}\binom{N-r}{d}$. Then it is left to choose one out of the d blocks of zeros in the circle with $N-r$ positions and insert the initially reserved block of r zeros. Hence we end up with $|A| = \frac{1}{N-r}\binom{N-r}{d}\binom{d}{1}$.

Then we consider the case $\lfloor (N-d)/r \rfloor > 1$. In order to deal with this case we will use an inclusion-exclusion type argument. For $1 \leqslant k \leqslant \lfloor (N-d)/r \rfloor$, we define

$$m_k \coloneqq |\{\mathbf{v} \in A \mid \mathbf{v} \text{ contains exactly } k \text{ blocks of at least } r \text{ zeros}\}|.$$

This implies that $|A| = \sum_{k=1}^{\lfloor (N-d)/r \rfloor} m_k$. Let $1 \leqslant l \leqslant \lfloor (N-d)/r \rfloor$. As a first step of our inclusion-exclusion argument we use the same method as in the "base case" in order to count the number of vectors containing at least l blocks of at least r zeros. Hence, we initially reserve $l \cdot r$ zeros. Then there are $N-d-lr$ zeros and d ones to be placed on a circle of length $N-lr$. There are $\frac{1}{N-lr}\binom{N-lr}{d}$ rotation invariant ways to do that. Then it is left to choose l out of the d blocks of zeros in the circle with $N-lr$ positions and insert the initially reserved l blocks of r zeros. So we end up with $\frac{1}{N-lr}\binom{N-lr}{d}\binom{d}{l}$ choices. However during the process of counting, for $l \leqslant k \leqslant \lfloor (N-d)/r \rfloor$ we double

[2] In combinatorial terms, the set M contains the binary necklaces of length N with exactly d 1s.

counted by a factor of $\binom{k}{l}$ all cases were there are exactly k blocks of at least r zeros. Therefore it follows that

$$\frac{1}{N-lr}\binom{N-lr}{d}\binom{d}{l} = \sum_{k=l}^{\left\lfloor \frac{N-d}{r} \right\rfloor} \binom{k}{l} m_k.$$

In order to compute the sum of m_k from the weighted sum given above, we consider the alternating sum

$$\begin{aligned}
\sum_{l=1}^{\left\lfloor \frac{N-d}{r} \right\rfloor} \frac{(-1)^{l+1}}{N-lr}\binom{N-lr}{d}\binom{d}{l} &= \sum_{l=1}^{\left\lfloor \frac{N-d}{r} \right\rfloor} (-1)^{l+1} \sum_{k=l}^{\left\lfloor \frac{N-d}{r} \right\rfloor} \binom{k}{l} m_k \\
&= \sum_{k=1}^{\left\lfloor \frac{N-d}{r} \right\rfloor} \left(\sum_{l=1}^{k} \binom{k}{l} (-1)^{l+1} \right) m_k \\
&= \sum_{k=1}^{\left\lfloor \frac{N-d}{r} \right\rfloor} -((1-1)^k - 1) m_k \\
&= \sum_{k=1}^{\left\lfloor \frac{N-d}{r} \right\rfloor} m_k = |A|
\end{aligned}$$

This concludes the proof. □

Remark 5.5. *For $a \in B_N(d)$ and small r, namely $r < \lceil N/d \rceil$ it was shown that $\Pr([r] \in \mathrm{Pat}(a)) = 1$. However, we can extend this result to any pattern J with $|J| < \lceil N/d \rceil$. For any such J this can be done by considering a bijection from $B_N(d)$ to $B_N(d)$ where the elements at indices in $[|J|]$ are mapped to the indices in J. This could be considered a slight improvement of [MS01, Theorem 1] as it gives an exact result for "small" patterns.*

In [MS01, Theorem 1] it was shown that for a randomly chosen pattern J of length r and $a \in B_N(d)$ it is expected that

$$\Pr(J \in \mathrm{Pat}(a)) \approx 1 - \left(1 - \frac{\binom{N-r}{d}}{\binom{N}{d}}\right)^N. \qquad (5.1)$$

This was also supported by experimental results. As a pattern with a higher success probability could lead to a faster attack, a natural question arises. Which success probability is higher: the one in [MS01, Theorem 1] or the one in Proposition 5.4? Unfortunately, due the complexity of the sum in Proposition 5.4 we were not able to prove a result on this. Therefore we resort to making the following assumption, which is just a more compact algebraic reformulation of the comparison between the formula in Proposition 5.4 and the right part of equation (5.1).

Assumption 5.6. *For $N, d, r \in \mathbb{N}$ with $2 \leqslant d < N$ and $\lceil \frac{N}{d} \rceil \leqslant r < N - d$ it holds that*

$$\left(1 - \frac{\binom{N-r}{d}}{\binom{N}{d}}\right)^N \leqslant \frac{1}{\binom{N-1}{d-1}} \sum_{l=0}^{\left\lfloor \frac{N-d}{r} \right\rfloor} \binom{d}{l}\binom{N-lr-1}{d-1}(-1)^l.$$

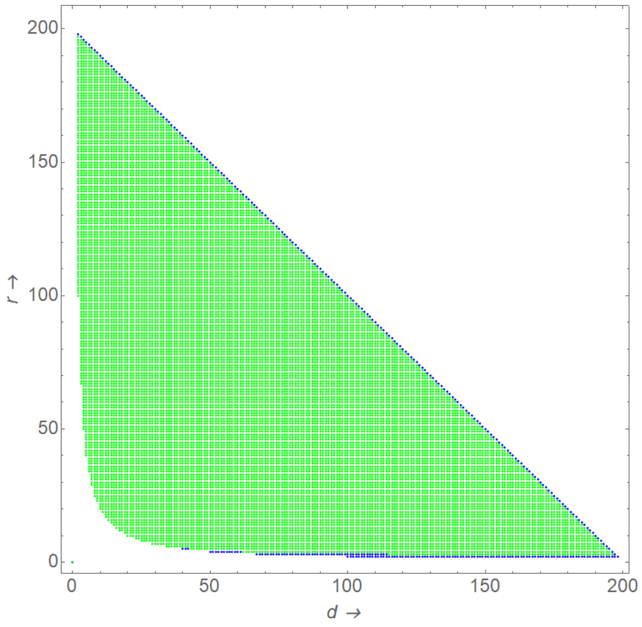


Figure 5.1: Experimental results on the validity of Heuristic Assumption 5.6. In this example we took $N = 200$ and checked the validity for all pairs (d, r) with $2 \leqslant d < N$ and $\lceil \frac{N}{d} \rceil \leqslant r \leqslant N - d$. Green points indicate that the assumption is true whereas blue points indicate that it is false.

We checked the validity of our assumption experimentally for various triplets (N, d, r). In Figure 5.1 the result of one such experiment is shown. We consider it to be a good representative for getting an intuition on the validity of our heuristic assumption.

As it is shown in Figure 5.1 the assumption is false on the diagonal $r = N - d$. However this is not a problem in practice as we would not try to guess all zeros in a vector anyway. Therefore we excluded this corner case from our assumption. However the assumption appears to be false in many cases on the lower boundary of Figure 5.1 as well. This is a different case as these points could correspond to practical values which an attacker might decide to use. Hence, in these cases it is not obvious if using the consecutive zeros pattern or a random pattern is more efficient. This depends on the accuracy of the model in [MS01, Theorem 1] for the success probability in these particular cases. Experimentally it appears that blue points only occur for $r \leqslant 6$. Under this extra restriction Heuristic Assumption 5.6 was found to be always true for $N < 10^4$.

This brings us to the following heuristic result, which follows directly by Proposition 5.4 and the Heuristic Assumption 5.6.

Heuristic result 5.7. *Let* $N, d \in \mathbb{Z}_{>0}$*,* $\mathfrak{a} \in B_N(d)$ *and* $J \subset [N]$ *chosen randomly such that* $|J| = r$ *with* $\lceil \frac{N}{d} \rceil \leqslant r < N - d$*. Assuming that [MS01, Theorem 1] accurately predicts the* $\Pr(J \in \mathrm{Pat}(\mathfrak{a}))$ *and the heuristic assumption 5.6 holds, then*

$$\Pr([r] \in \mathrm{Pat}(\mathfrak{a})) \leqslant \Pr(J \in \mathrm{Pat}(\mathfrak{a})).$$

5.3 — A pattern's index of symmetry

In this section we define an index of symmetry of a pattern. We attempt to use this notion in order to gain insight into questions 3 and 5 from the introduction. We do that by examining the somewhat dual of questions 3 and 5. Namely we ask about multiple winners instead of distinct winners, as this appears easier to handle. We will also examine the adequacy of our index of symmetry as a measure aiding in choosing patters with higher success probability.

5.3.1 – Multiple winners. By multiple winners we refer to the fact that a pattern may appear more than once in a vector (taking into account rotations). We consider the problem where we try to minimise the number of multiple winners. A pattern J of length r may appear k times in a vector $\mathbf{a} \in B_N(d)$ as long as $kr - (k-1) \leqslant N - d$ so $k \leqslant (N-d-1)/(r-1)$. Hence, we cannot expect to minimise the maximum number of times a pattern can occur in a vector $\mathbf{a} \in B_N(d)$ below this bound. For a pattern J to occur more than $(N-d-1)/(r-1)$ times in a vector $\mathbf{a} \in B_N(d)$ it might need to be more symmetric. By that we mean that there exists a rotation of the pattern which partially overlaps with the pattern. We define

$$S_r(N) := \{\{x_i\}_{i=1}^r \mid x_i \in [N] \text{ and } x_i \neq x_j \text{ for } i \neq j\}.$$

Let $J = \{x_i\}_{i=1}^r \in S_r(N)$ and $g \in \mathbb{Z}_N$. We denote as gJ the rotation of J by g, hence $gJ = \{x_i + g \pmod N\}_{i=1}^r$ where representatives $\pmod N$ are taken in $[N]$.

Definition 5.8. *Let $N, r \in \mathbb{Z}_{>0}$ and $J \in S_r(N)$. We define the index of symmetry of the pattern J to be*

$$\eta(J) := \max_{g \in \mathbb{Z}_N \setminus \{0\}} |gJ \cap J|$$

Remark 5.9. *The index $\eta(J)$ of a pattern J does not depend on a specific vector $\mathbf{a} \in B_N(d)$ but solely on the pattern J.*

Proposition 5.10. *Let $N, r \in \mathbb{Z}_{>0}$ and $J \in S_r(N)$ be a pattern. It holds that* $\frac{r(r-1)}{N} \leqslant \eta(J) \leqslant r-1$.

Proof. The upper bound follows directly as two patterns can share at most $r-1$ elements without being identical. For the lower bound we consider a graph theoretical approach. Let $J = \{x_i\}_{i=1}^r \in S_r(N)$ be a pattern and $g \in \mathbb{Z}_N \setminus \{0\}$. We ask for a lower bound to $|gJ \cap J|$. But,

$$\begin{aligned} |gJ \cap J| &= |\{x_i + g \pmod N\}_{i=1}^r \cap \{x_i\}_{i=1}^r| \\ &= |\{x_i \in J \mid \exists x_j \in J \text{ with } x_i + g \equiv x_j \pmod N\}| \\ &= |\{(x_i, x_j) \in J \times J \mid x_j - x_i \equiv g \pmod N\}|. \end{aligned}$$

Assign the elements of J to the vertices of the complete graph K_r. Each edge of the graph corresponds to the fact that we can move from an element in $\mathbb{Z}_N$ to another by adding an element in $\mathbb{Z}_N$. We map each element $g \in \mathbb{Z}_N$ to a color but such that g and $-g$ get the same color. We consider the following edge colouring of the graph K_r. Color the edge (x_i, x_j) according to the color corresponding to $x_i - x_j \pmod N$. This is well defined as $x_i - x_j$ and $x_j - x_i$ correspond to the same color. It then follows that $|gJ \cap J|$ equals to the number of occurrences of the color corresponding to g and $\eta(J)$ is the maximum among all colors in the graph. The graph has $\binom{r}{2}$ edges and there are at most $\lceil (N-1)/2 \rceil$ colors being used. Therefore the maximum times a color appears is at least $\binom{r}{2}/\lceil (N-1)/2 \rceil$ which implies the result. $\square$

Remark 5.11. *For the consecutive zeros pattern* $[r]$ *it holds that* $\eta([r]) = r-1$ *showing that the upper bound is tight. This also provides a good reason why the consecutive zeros pattern appears to maximize the number of multiple winners. Figure 5.2 provides some indication that by random sampling of patterns we observe an increase of the multiple winners as the value of* $\eta(J)$ *increases.*

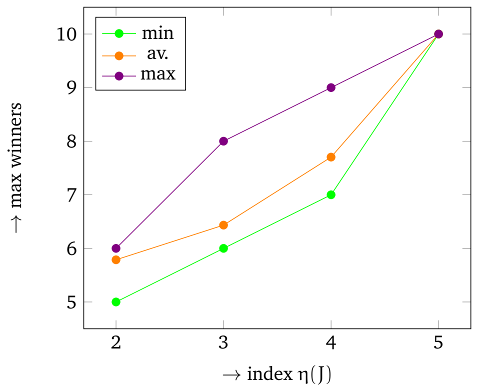


Figure 5.2: Experimental results on the maximum number of winners for $(N, d, r) = (19, 4, 6)$. Patterns $J \in S_r(N)$ are considered in classes according their value $\eta(J)$. For each such class the graph shows the minimum, average and maximum of the maximum number of times a pattern in the class can occur in a vector of $B_N(d)$.

5.3.2 – The distribution of patterns according to $\eta(J)$. Having already defined the index $\eta(J)$ of a pattern J, a first question that arises is how are the patterns $J \in S_r(N)$ distributed according to this index. Computing this distribution exactly seems to be a hard combinatorial problem. However, it is possible to get an approximation using probabilistic techniques.

Let Z be a random variable modelling the experiment; pick a pattern $J \in S_r(N)$ and compute $\eta(J)$. We are going to approximate the probability mass function of Z. The index $\eta(J)$ was defined as $\eta(J) = \max_{g \in \mathbb{Z}_N \setminus \{0\}} |gJ \cap J|$ where $J = \{x_i\}_{i=1}^{r}$. The first step is for some fixed $g \in \mathbb{Z}_N \setminus \{0\}$ and for some random $x \in J$ to define a Bernoulli trial of whether x is in gJ or not. It follows that the success probability is $\Pr(x \in gJ) = \frac{r-1}{N}$. Next, let Y be a random variable modelling the experiment; choose a fixed $g \in \mathbb{Z}_N \setminus \{0\}$ and a random pattern $J \in S_r(N)$, then compute $|gJ \cap J|$. The random variable Y is actually counting the number of successes for repeating the aforementioned Bernoulli trial r times. If these Bernoulli trials were independent we could conclude that Y follows a binomial distribution. Therefore, this is the point where from an exact model we will move to an approximation by making the assumption that these Bernoulli trials are actually independent. Under this assumption we can deduce that $\Pr(Y = k) = \binom{r}{k} p^k (1-p)^{r-k}$ where $p = (r-1)/N$.

By the definition of the index $\eta(J)$ we get that $Z = \max_{1\leqslant i\leqslant N-1} Y_i$ where $Y_i \sim Y$ and each Y_i corresponds to an intersection $|gJ \cap J|$ implied by an element in $g \in \mathbb{Z}_N\backslash\{0\}$. However, the random variable Z could be considered as $Z = \max_{1\leqslant i\leqslant \lceil (N-1)/2\rceil} Y_i$ rather than $Z = \max_{1\leqslant i\leqslant N-1} Y_i$. This is due to the fact that for a pattern J and $g \in \mathbb{Z}_N$ it holds that $|gJ \cap J| = |(-g)J \cap J|$. Therefore half of the random variables Y_i initially considered in the definition of Z would be redundant. So, for the cumulative distribution function of Z we get that

$$\begin{aligned}\Pr(Z \leqslant m) &= \prod_{i=1}^{\lceil \frac{N-1}{2}\rceil} \Pr(Y_i \leqslant m)\\ &= \left(\sum_{k=0}^{m}\binom{r}{k}p^k(1-p)^{r-k}\right)^{\lceil \frac{N-1}{2}\rceil}. \qquad (5.2)\end{aligned}$$

Hence, finally we can get an approximation of $\Pr(Z = m)$ using the fact that $\Pr(Z = m) = \Pr(Z \leqslant m) - \Pr(Z \leqslant m-1)$.

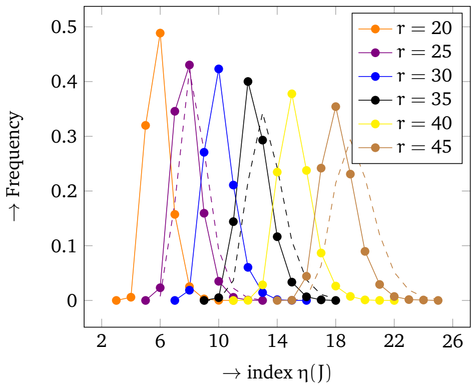
This should only be considered as a simplified rough approximation of the distribution of patterns according to the index $\eta(J)$. As we already mentioned, we made an assumption on the random variable Y in order to be able to draw a conclusion. However we cannot predict how much that affects the quality of the approximation neither if it is even possible to get an exact formula if this assumption is not made. Experimental results testing the presented model are shown in Figure 5.3.

5.3.3 – Using $\eta(J)$ in practice. In this section we provide experimental results supporting the definition and use of the index η in practice. A question already mentioned in Section 5.3.2 is how patterns J of a specific length r in dimension N are distributed according to their value $\eta(J)$. For the experiments we used most values of (N, r) from the experiments of [MS01, Section 6] and an extra value of N which is closer to currently used parameters, combined with some values $r > \sqrt{N}$. We selected $r > \sqrt{N}$ in order to ensure that the bound $r(r-1)/N$ is greater than 1.

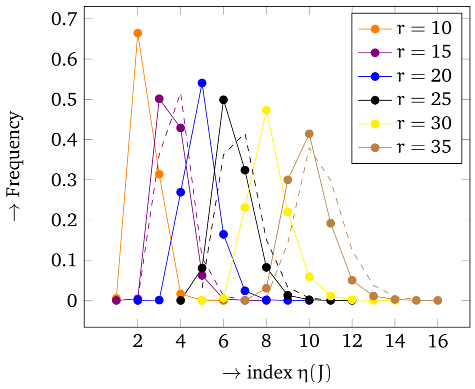
Figure 5.3 shows that patterns are mainly concentrated towards the lower bound of $\eta(J)$, namely $\frac{r(r-1)}{N}$. So, sampling patterns randomly will result in patterns with a relatively small index η. In addition Figure 5.3 provides good evidence that additionally to the upper bound given in Proposition 5.10 being tight, also the lower bound of $\eta(J)$ appears to be quite tight. However, we should mention that in practice we observe only patterns with $\eta(J)$ up to roughly $r/2$ instead of $r-1$.

Finally, Figure 5.3 presents some preliminary experimental results on the accuracy of the model adopted in Section 5.3.2 in order to describe the distribution of patterns according to $\eta(J)$. A first conclusion could be that as r decreases the accuracy of the model improves. Of course this is only an experimental observation. The reason for our model not being exact is an assumption we made in Section 5.3.2, namely that the random variable Y follows a binomial distribution. The random variable Y models $|gJ \cap J|$ for a fixed $g \in \mathbb{Z}_N\backslash\{0\}$ and a random pattern $J \in S_r(N)$. We tested that assumption separately and one set of results is shown in Figure 5.4.[3] The results in Figure 5.4 suggest that

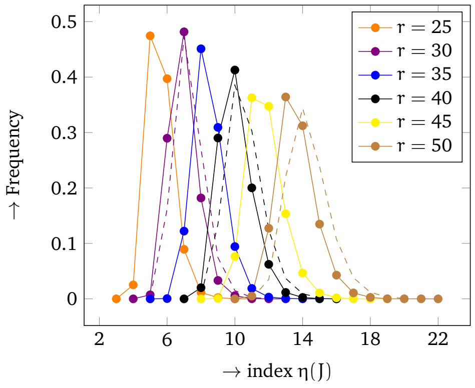
[3]We performed the experiment for $N = 167, 251, 347$ but we only show the case $N = 251$ as the other cases are similar.



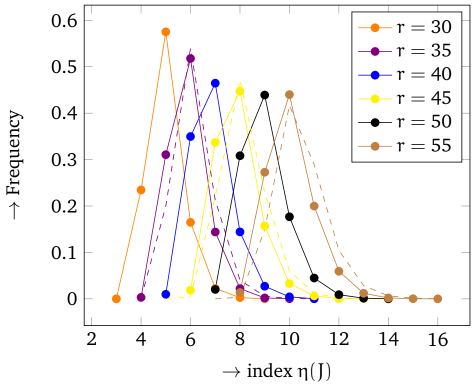
(a) Distributions for $N = 167$.



(b) Distributions for $N = 251$.



(c) Distributions for $N = 347$.



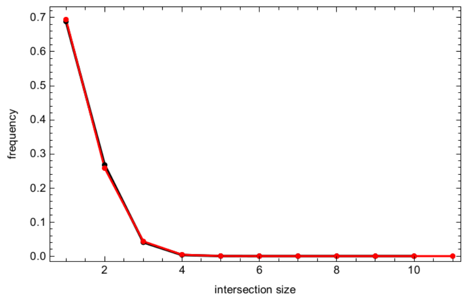
(d) Distributions for $N = 761$.

Figure 5.3: Experimental results for the distribution of patterns according to the index $\eta(J)$. For each pair (N, r) we sampled 10,000 random patterns of length r in dimension N and plotted the observed distribution (full lines). The dashed lines show the approximation of the distribution considered in Section 5.3.2 for some of the pairs (N, r).

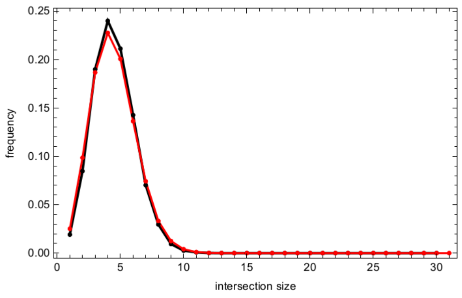
for smaller values of r the binomial model matches quite closely the behaviour of the random variable Y. However for bigger r there are significant deviations. This behaviour could explain that of the approximations shown in Figure 5.3. Finally we notice that in Figure 5.4 the red curves corresponding to the binomial model seem to have a peak at the same point as the black ones.

Apart from the distribution of patterns according to the index η, an interesting question is how the success probability of a pattern J relates to its value $\eta(J)$. Patterns with higher success probability could potentially speed-up attacks. In order to investigate this behaviour we performed the following experiments.

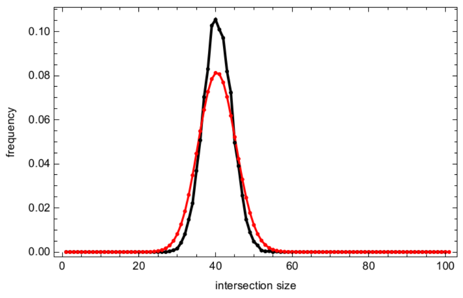
For many of the triplets (N, d, r) used in [MS01, Section 6] we sampled 10,000 random patterns of length r and 10 vectors from $B_N(d)$. Similarly to [MS01], for each vector we counted how many patterns out of all observed patterns J with $\eta(J) = k$ occurred in



(a) $r = 10$.



(b) $r = 30$.



(c) $r = 100$.

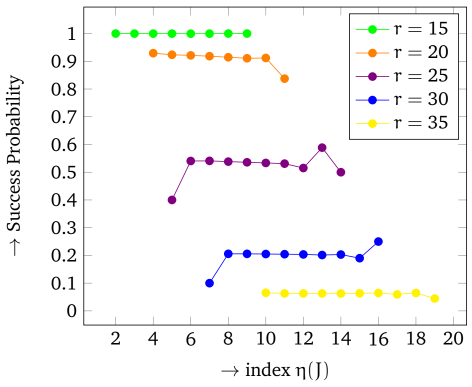
Figure 5.4: Experimental results comparing the probability mass function of the random variable Y defined in Section 5.3.2 to the suggested binomial model. For this experiment we chose $N = 251$ and $r = 10, 30, 100$. The black lines indicate the experimentally observed distribution whereas the red ones the suggested binomial model.

the vector. For each triplet (N, d, r) and value k of the index η this resulted in 10 pairs of index-value–probability. We finally considered the average amongst the 10 pairs as the average success probability of a pattern J with $\eta(J) = k$. As this computation was run on a 48-core machine we could run this experiment in parallel 48 times. The average of all 48 results is shown in Figures 5.5a, 5.5b, 5.5c.

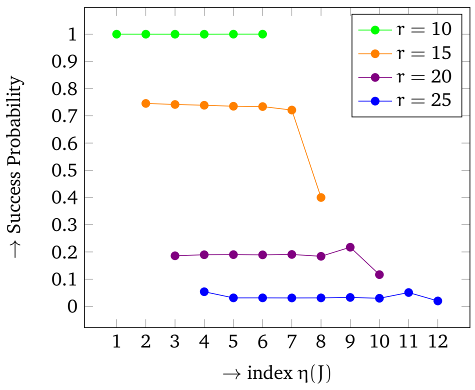
As the above experiments do only approximate the true behaviour of the success probability of patterns according to the index η, we also performed some experiments with small parameters. In these cases we were able to go through all patterns of length r instead of just a fraction of them and compute their exact success probability by iterating through all vectors in $B_N(d)$. The result of only one of the "small" experiments we performed is shown in Figure 5.5d.

Starting with easiest experiment, i.e. Figure 5.5d we can see that the success probability behaves almost always monotonically (apart from the case $r = 4$ and $\eta(J) = 1, 2$). This "almost" monotonic behaviour turned out to occur in more such "small" experiments which we performed with $(N, d) = (13, d)$ and $(N, d) = (11, d)$ for several values of d. We do not include all of these results as they all point to the same behaviour.

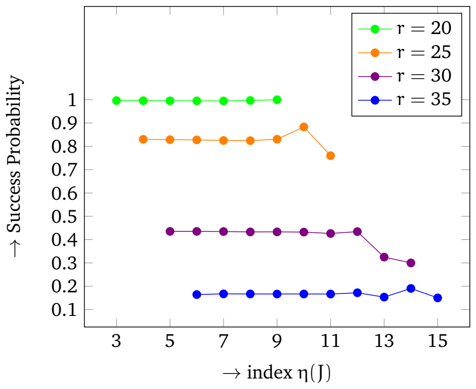
However, if we examine Figures 5.5a, 5.5b, 5.5c we cannot observe the same nice monotonic behaviour where the maximum is reached for the smallest value of $\eta(J)$. For



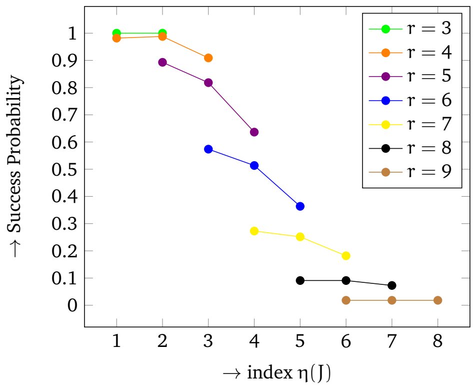
(a) Success prob. for $(N, d) = (167, 30)$.



(b) Success prob. for $(N, d) = (251, 72)$.



(c) Success prob. for $(N, d) = (347, 64)$.



(d) Success prob. for $(N, d) = (13, 4)$.

Figure 5.5: Experimental results for the success probability of a pattern J with regard to its value $\eta(J)$. The plotted points correspond to the average success probability observed among a set of patterns with the same value of the index η.

the higher values of (N, d) used in these experiments the success probability appears to behave more erratically. Even though, there are still some conclusions which can be drawn. It seems that for each value of r the middle section of the curve corresponding to that value does not deviate a lot. Also, in many cases we notice a peak towards the highest values of $\eta(J)$.

The results shown in Figures 5.5a, 5.5b, 5.5c are approximations of the truth and therefore they could be not completely accurate. Extrapolating from observations like the ones in Figure 5.5d (monotonic behaviour) appears to be a dangerous exercise. So, what should an attacker do if he wants to choose patterns J with regard to their value $\eta(J)$ and maximize the success probability?

One strategy would be to perform an experiment like the ones shown in Figures 5.5a, 5.5b, 5.5c, 5.5d in order to create a profile of the behaviour of the success probability with regard to the index η. It seems that the curves in Figure 5.5 are not completely flat for "longer" patterns, thus the index $\eta(J)$ could be of use in the choice of patterns. For example in Figure 5.5a for $r = 25$ it appears that patterns with $\eta(J) = 13$ have on average a clearly higher success probability than the rest. Computing the value $\eta(J)$ takes time polynomial in N and hence (depending on the desired accuracy of the profile) this should take only negligible time compared to a lattice attack which would have time complexity exponential in N. However, creating a profile of the behaviour of the success probability with regard to the index η can be considered a trade-off in a more relaxed setting. This is due to the fact that such a profile would only have to be computed once and then it can be reused in any number of attacks which use the same parameters. Finally, it should be pointed out that computing such profiles is independent of the lattice and for a guessed pattern of length r the dimension of the lattice is reduced from N to $N - r$.

An advantage of this method to just finding some specific pattern is that in this way we identify a whole class of patterns with a presumably higher success probability compared to a randomly chosen pattern. Therefore it will be harder for a lattice creator to generate a lattice whose shortest vector will contain none of the patterns in such a class.

As an attacker, another take-away point of Figure 5.5 is that for "small enough" values of r it appears that the success probability of a pattern J is independent of $\eta(J)$ and close to 1. Therefore, an attacker can guess a "small enough" pattern for free: i.e. without a loss in the success probability of the lattice attack he will be running. Also, this seems to be independent of $\eta(J)$ thus in this case the attacker does not have to bother with it. Once the length of the pattern grows the success probability decreases, meaning that the lattice attack will have to be repeated for various "guessed" patterns. This actually describes a success probability-time trade-off. Of course, whether the overall performance can be improved depends on the performance of the chosen lattice attack and the success probability induced by the guessed patterns. In addition, guessing patterns could be viewed as a means of achieving a success probability-memory trade-off. If there are hardware limitations (i.e. available memory) an attack could be made feasible, at the cost of a smaller success probability.

Chapter 6

Exploiting generalised Ring-LWE lattices

This chapter contains work that is still work in progress.

6.1 — Introduction

The NTRU cryptosystem introduced by Hoffstein–Pipher–Silverman [HPS98] is one of the most well-known lattice-based cryptography schemes. Thus, it naturally attracted a lot of cryptanalytic attention. One of the research lines explored in the past few years is the so called *overstretched* NTRU. This line of research focuses on the study of NTRU with a "large" modulus q. The motivation behind this line of research is that most attacks against NTRU are better when q is large than when q is small.

The first published works investigating this path were by Albrecht–Bai–Ducas [ABD16] and Cheon–Jeong–Lee [CJL16] in 2016. In these works it was shown that if the modulus q is "large" enough then the underlying algebraic structure of NTRU can be used in order to improve upon already existing attacks at the time. Later, in 2017 Kirchner–Fouque [KF17] showed that the existence of an unusually dense sublattice (a sublattice of large dimension and small determinant) in the NTRU lattice was the key element for "better" lattice-basis reduction attacks in the regime of "large" q. Improving upon [KF17], Ducas–van Woerden in [DW21] provided improved asymptotic and concrete estimations of what is "large" q.

Therefore, what should be clarified next is the meaning of "large" modulus q. We will follow the same definitions adopted by Ducas–van Woerden in [DW21] for this notion. Namely, the standard regime (not "large" modulus q) is distinguished from the overstretched regime ("large" modulus q) by classifying according to which event occurs first during strong basis reduction (like BKZ [Sch87]):

– Secret Key Recovery (SKR_κ): Let $0 \leqslant \kappa < d$ be a fixed position, a vector as short as a secret key vector is inserted in the basis at position κ.
– Dense Sublattice Discovery (DSD_κ): Let $0 \leqslant \kappa < d$ be a fixed position, a vector strictly longer than the secret key but belonging to the dense (secret) sublattice generated by the secret key is inserted in the basis at position κ.

In their work [KF17] Kirchner and Fouque showed that asymptotically $q \geqslant n^{2.783+o(1)}$ is already "large" enough (i.e. overstretched) for ternary secrets. However recently Ducas and van Woerden in [DW21] brought this down to $q \geqslant n^{2.484+o(1)}$ and also provided a concrete analysis of where the fatigue point lies (i.e. the value of q separating the standard regime from the overstretched regime).

The Ring-LWE problem is closely related to NTRU and is also used as a building block in modern lattice-based cryptography. Thus the question arises, if the overstretched concept and corresponding analysis can be applied to the Ring-LWE problem. According to [KF17] the aforementioned analysis is possible for NTRU due to the inherent property of its lattice to contain an unusually dense sublattice of large dimension. However, the Ring-LWE problem is mainly modelled in literature by some lattice which does not possess this crucial property. The assumption that this is the *only* lattice modelling Ring-LWE, led to some claims (e.g. see [KF17, Section 7]) stating that Ring-LWE cannot get affected by attacks relying on the existence of an unusually dense sublattice. In this chapter we revisit such claims and try to provide evidence of whether they are true or not.

6.1.1 – Contributions. In this chapter we adopt a suggestion made in [BCLv19] by Bernstein–Chuengsatiansup–Lange–Vredendaal which leads to a series of lattices modelling Ring-LWE. These lattices are of increasing dimension and symmetry. This underlying symmetry allows the gradual formation of a dense sublattice of relatively large dimension. Hence, the fertile ground for an overstretched case analysis is created.

Initially, we examine the asymptotic requirement (block size) for a DSD event to occur. For our analysis we mainly adopt the techniques from [DW21]. We differ from [DW21] at the heuristic assumption used to model the profile of the basis at the point of detection. In particular, instead of using the ZGSA assumption we opt for the GSA assumption as it appears to be more suitable for the case of Ring-LWE (Figure 6.2). Finally, we compare the asymptotic estimate implied by the DSD analysis to the asymptotic requirement for a SKR event in order to decide the existence of a fatigue point in the case of Ring-LWE. We conclude that under a specific model of comparison for the cost of SKR and DSD events, there does not exist a fatigue point for the case of Ring-LWE. We further support our conclusion by experimental results.

Furthermore, we briefly examine (experimentally) the idea from [BCLv19] within a second framework. Namely, we examine how this larger set of lattices modelling Ring-LWE could in practice affect the performance of lattice-basis reduction attacks. Using a modification of the implementation from [DW21] we ran experiments where we tried to solve Ring-LWE instances while considering many lattices modelling the same Ring-LWE instance. Our experiments indicate that the best attack performance (success probability, block size) is not reached for the commonly used lattice of rank $2n + 1$.

6.2 — Preliminaries

6.2.1 – Notation. We adopt the already introduced notation from Chapter 1 for basic notions regarding lattices and lattice problems. As lattices are groups, sublattices are defined as subgroups contained in the full lattice and maintaining the additive group structure. For a basis $\mathbf{B}$ and $i \in \{0, 1, \ldots, d-1\}$ we define π_i as the projection onto the orthogonal complement of $\{\mathbf{b}_0, \ldots, \mathbf{b}_{i-1}\}$. The Gram–Schmidt vectors $\mathbf{b}_0^*, \ldots, \mathbf{b}_{d-1}^*$ are defined as $\mathbf{b}_i^* := \pi_i(\mathbf{b}_i)$. The sequence $(\|\mathbf{b}_i^*\|)_{i=0}^{d-1}$ is called the profile of a basis. We recall that $\mathrm{Vol}(\mathcal{L}) = \prod_{i=0}^{d-1} \|\mathbf{b}_i^*\|$. We write $\mathbf{B}_{[l:r)}$ for the matrix $[\pi_l(\mathbf{b}_l), \ldots, \pi_l(\mathbf{b}_{r-1})]$, and denote the projected sublattice $\mathcal{L}(\mathbf{B}_{[l:r)})$ as $\mathcal{L}_{[l:r)}$ when the basis is clear from the context. If $l = 0$, there is no projection involved, thus $\mathcal{L}_{[0:r)}$ is just the sublattice generated by $[\mathbf{b}_0, \dots, \mathbf{b}_r]$.

6.2.2 – The Ring-LWE problem and its lattices. Let q be a prime and let $f(X) \in \mathbb{Z}[X]$ be an irreducible polynomial of degree n. We set $R = \mathbb{Z}[X]/\langle f(X)\rangle$ and $R_q = \mathbb{Z}_q[X]/\langle f(X)\rangle$. In order to ease the exposition we will consider the specific case $f(X) = X^n + 1$ with $n = 2^l$, known as the $2n$-th cyclotomic polynomial.

Definition 6.1 (The Ring-LWE Problem). *Let $a, e \in R$ with "small" coefficients*[1]*, sampled according to some distributions χ_1 and χ_2. Let G be a uniformly random element of R_q and $A \in R_q$ such that $aG + e = A$ in R_q. Given, G, A find a.*

This problem can be modelled over R using some lifting from R_q such as

$$\begin{pmatrix} q & A & -G \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r \\ t \\ a \end{pmatrix} = \begin{pmatrix} e \\ t \\ a \end{pmatrix},$$

for some $r \in R$. Also, in this case $t \in R$ is set to 1. As a next step, one could further model the problem over $\mathbb{Z}$. In order to do so, it is useful to consider the multiplication matrix $\mathbf{M}(g)$ of a $g \in R$, i.e. the j-th column of $M(g)$ is formed by the coefficients of $x^j g$. Consider the lattice generated by the columns of the following $(2n+1) \times (2n+1)$ matrix:

$$\begin{pmatrix} q\mathbf{I}_n & \boldsymbol{A} & \mathbf{M}(-G) \\ \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_n \end{pmatrix} \tag{6.1}$$

where $\boldsymbol{A}$ is the coefficient vector of A. Then a solution to the Ring-LWE problem is given by a short vector in this lattice. This is a canonical choice for a lattice modeling Ring-LWE, in the spirit of [BG14], but one could also use more "generalised" lattices of dimension up to $3n$. For example, one can consider the matrix representing the "natural" homogenisation over $\mathbb{Z}$ (6.2) , not taking $t \in \mathbb{Z}$ but $\mathbf{t} \in \mathbb{Z}^n$:

$$\begin{pmatrix} q\mathbf{I}_n & \mathbf{M}(A) & \mathbf{M}(-G) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_n \end{pmatrix}. \tag{6.2}$$

However, this is merely one choice for $\mathbf{t} \in \mathbb{Z}^n$. More lattices can be obtained by taking $\mathbf{t} \in \mathbb{Z}^\nu$ with $\nu \in \{1, \dots, n\}$. In this way, the number of short vectors representing a solution to the Ring-LWE problem is increasing with ν, which may help in cryptanalysis. The "extra" vectors representing a solution to the Ring-LWE problem correspond to multiples of (e, t, a) by a "small" element in R (i.e. x^j). This implies the use of a more structured lattice at the cost of a higher dimension. If we wished no extra structure in the used lattice we could just use one column out of the middle block of the basis matrix (6.2). Hence, the choice of ν can offer a trade-off between the dimension and the amount of structure in the lattice modelling Ring-LWE. This idea was described in [BCLv19, Section 6.3]. Hence, we give the following definition.

[1]Here "small" usually refers to some set like $\{0, \pm 1\}$.

Definition 6.2. *Let* (n, q, G, A, a, e) *be a Ring-LWE instance and* $\nu \in \{1, \ldots, n\}$. *We define the rank* $2n + \nu$ *Ring-LWE lattice as*

$$\mathcal{L}^{G,A,q,\nu} := \begin{pmatrix} q\mathbf{I}_n & \mathbf{M}(A)_{[0:\nu)} & \mathbf{M}(-G) \\ \mathbf{0} & \mathbf{I}_\nu & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_n \end{pmatrix} \cdot \mathbb{Z}^{2n+\nu}, \tag{6.3}$$

and its rank ν *(secret) dense sublattice by:*

$$\mathcal{L}^{a,e} := \mathbf{B}^{a,e} \cdot \mathbb{Z}^\nu \subset \mathcal{L}^{G,A,q,\nu}, \text{ where } \mathbf{B}^{a,e} := \begin{pmatrix} \mathbf{M}(e)_{[0:\nu)} \\ \mathbf{I}_\nu \\ \mathbf{M}(a)_{[0:\nu)} \end{pmatrix}.$$

For the ease of exposition we will denote a random column of $\mathbf{B}^{a,e}$ as $(\mathbf{e}|\mathbf{t}|\mathbf{a})$. The basis $\mathbf{B}^{a,e}$ is expected to be close to orthogonal and hence the Hadamard bound 6.10 can provide a “good” approximation of $\mathrm{Vol}(\mathcal{L}^{a,e})$.

Remark 6.3. *Instead of choosing the first* ν *columns of* $\mathbf{M}(A)$ *in order to define* $\mathcal{L}^{G,A,q,\nu}$, *one could also make a random choice of* ν *columns of* $\mathbf{M}(A)$. *We chose to use the first* ν *columns of* $\mathbf{M}(A)$ *in the definition of* $\mathcal{L}^{G,A,q,\nu}$ *for the ease of exposition.*

6.2.3 – Useful definitions, results and assumptions. We start by mentioning some commonly adopted heuristic assumptions in the study of lattices and lattice basis reduction which were introduced in Section 1.2. In particular, we will use the Gaussian heuristic 1.18 and the Geometric Series Assumption 1.20. We also recall the definition of a q-ary lattice.

Definition 6.4 (q-ary lattice). *A lattice* $\mathcal{L}$ *of dimension* d *is called* q*-ary if for some* $q > 0$ *we have*

$$q\mathbb{Z}^d \subseteq \mathcal{L} \subseteq \mathbb{Z}^d.$$

Assumption 6.5 (ZGSA). *Let* $\mathbf{B}$ *be a basis of a* $(2n + \nu)$*-dimensional* q*-ary lattice* $\mathcal{L}$ *with* n q*-vectors. After BKZ-*β *reduction, the profile has the following shape*

$$\|\mathbf{b}_i^*\| = \begin{cases} q & \text{if } i \leqslant n - m \\ \sqrt{q}\,\alpha_\beta^{(n-i)} & \text{if } n - m < i < n + m \\ 1 & \text{if } i \geqslant n + m \end{cases} \tag{6.4}$$

where $\alpha_\beta = (\beta/2\pi e)^{1/(\beta-1)}$ *and* $m = \frac{1}{2}\frac{\log(q)}{\log(\alpha_\beta)}$.

Heuristic assumption 6.5 is the analogue of a similar assumption about NTRU. We refer to [HG07, Section 3.1] and [DW21, Section 2.3] for a more detailed description. The version regarding NTRU was introduced in [DW21] under the name “ZGSA” in order to emphasise the Z-like shape of it. Of course, as it can be seen in Figure 6.1 it is not exactly a Z shape, but we decide to keep the name for the uniformity of notation.

Remark 6.6. *The value of* m *in Heuristic assumption 6.5 can exceed* n *for big “enough” blocksizes* β. *This would break the “Z” shape of the profile, and lead to a profile which would be closer to the prediction of the GSA instead of the ZGSA.*

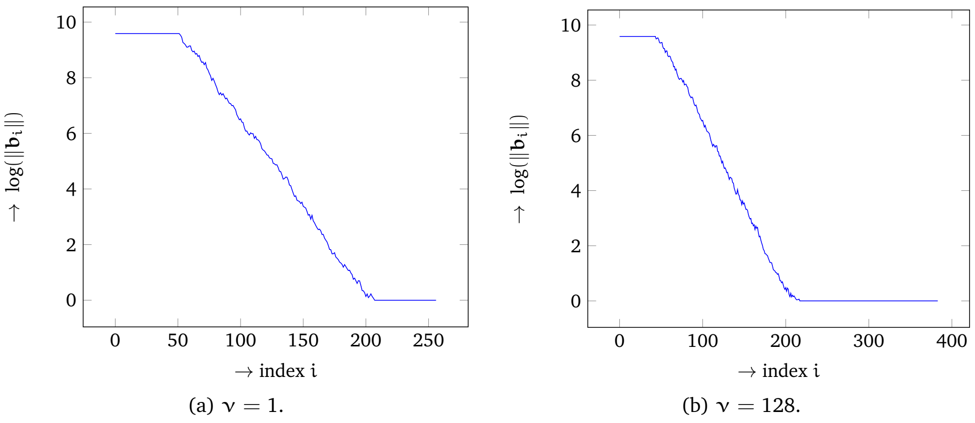


Figure 6.1: An example of the basis' profile for $\mathcal{L}^{G,A,q,\nu}$ after BKZ reduction with blocksize 10. The shown example corresponds to $n = 128$, $q = 769$ and $\nu = 1, 128$.

Definition 6.7 (BKZ Events)**.** *For a progressive BKZ-β run (the block size β increases progressively) on a Ring-LWE lattice $\mathcal{L}^{G,A,q,\nu}$ with dense sublattice $\mathcal{L}^{a,e}$ we define two events:*

1. ***Secret Key Recovery (SKR)**: The first time one of the secret keys $(\mathbf{e}|\mathbf{t}|\mathbf{a})$ is inserted into the basis.*
2. ***Dense Sublattice Discovery (DSD)**: The first time a vector $\mathbf{v} \in \mathcal{L}^{a,e}$ strictly longer than the secret key(s) is inserted into the basis.*

We further specify SKR_κ and DSD_κ when the insertion takes place at position κ in the basis.

Heuristic result 6.8 (2016 Estimate [ADPS16])**.** *Let $\mathcal{L}$ be a lattice of dimension d and $\mathbf{v} \in \mathcal{L}$ be an unusually short vector $\|\mathbf{v}\| \ll \mathrm{gh}(\mathcal{L})$. Then under the Geometric Series Assumption BKZ recovers $\mathbf{v}$ if*

$$\sqrt{\beta/d}\,\|\mathbf{v}\| < \alpha_\beta^{(2\beta-d-1)/2}\,\mathrm{Vol}(\mathcal{L})^{1/d}, \tag{6.5}$$

where $\alpha_\beta = (\beta/2\pi e)^{1/(\beta-1)}$.

Lemma 6.9 (Sublattice Volume [DW21])**.** *Let $\mathcal{L}$ be a d-dimensional lattice with basis $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$, and consider the sublattice $\mathcal{L}_{[0:s)}$. For any n-dimensional sublattice $\mathcal{L}' \subset \mathcal{L}$ we have*

$$\mathrm{Vol}(\mathcal{L}_{[0:s)} \cap \mathcal{L}') \leqslant \mathrm{Vol}(\mathcal{L}') \left(\min_J \prod_{j \in J} \|\mathbf{b}_j^*\| \right)^{-1}, \tag{6.6}$$

where $k := \dim(\mathcal{L}_{[0:s)} \cap \mathcal{L}')$ and J ranges over the $(n-k)$-size subsets of $\{s, \ldots, d-1\}$.

Finally we recall the Hadamard bound.

Proposition 6.10 (Hadamard bound)**.** *Let $\mathbf{B}$ be a $d \times d$ real matrix with columns $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$ then*

$$|\det(\mathbf{B})| \leqslant \prod_{i=0}^{d-1} \|\mathbf{b}_i\|$$

6.3 — Can Ring-LWE get overstretched?

6.3.1 – Dense sublattice detection for Ring-LWE. In order to estimate the minimum block size β at which a DSD event occurs in $\mathcal{L}^{G,A,q,\nu}$, we are going to imitate the analysis of [DW21]. Thus, the first step is to adopt the following claim.

Claim 6.11 (DSD-PT). *A tour of BKZ-β triggers the DSD event for $\mathcal{L}^{G,A,q,\nu}$ if*

$$\|\pi_{n+l-\beta}(\mathbf{v})\| < \|\mathbf{b}^*_{n+l-\beta}\|, \tag{6.7}$$

where $\mathbf{v}$ is a shortest vector of $\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}$ for some $0 < l \leqslant n + \nu$.

This claim is the analogue of a similar claim used about NTRU in [DW21] and provides a quantification of when a DSD event is triggered. Hence, we are left with the task of estimating the minimum β for which the inequality (6.7) holds. Following the analysis of [DW21] the right hand side (i.e. $\|\mathbf{b}^*_{n+l-\beta}\|$) could be estimated by the (Z)GSA. On the other hand, the $\|\pi_{n+l-\beta}(\mathbf{v})\|$ can be upper bounded by Minkowski's bound. In order to use Minkowski's bound, the volume of $\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}$ is bounded using the following corollary of Lemma 6.9.

Corollary 6.12. *Let $\mathcal{L}^{G,A,q,\nu}$ be a Ring-LWE lattice with dense sublattice $\mathcal{L}^{a,e}$ of dimension ν. If* $\dim(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}) = k$ *for some $l \geqslant 0$ then*

$$\mathrm{Vol}(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}) \leqslant \mathrm{Vol}(\mathcal{L}^{a,e}) \left(\prod_{j=2n+k}^{2n+\nu-1} \|\mathbf{b}^*_j\| \right)^{-1}. \tag{6.8}$$

In order to make use of Corollary 6.12 an estimation of $\dim(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e})$ is needed. This dimension does not rely only on the lattice $\mathcal{L}^{G,A,q,\nu}$ but also on the basis $\mathbf{B}$ of it being used, as it considers $\mathcal{L}^{G,A,q,\nu}_{[0:n+l)}$, which is basis-dependent. As BKZ runs, it modifies the basis of $\mathcal{L}^{G,A,q,\nu}$ and hence also $\mathcal{L}^{G,A,q,\nu}_{[0:n+l)}$. Therefore we can conclude that $\dim(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e})$ could vary as BKZ runs. Our goal is to use Corollary 6.12 for the moment at which BKZ detects a vector in $\mathcal{L}^{a,e}$ for the first time. We will make the assumption that just before the detection BKZ will have "mixed" the initial basis vectors to a point that they look like "random" vectors compared to $\mathcal{L}^{a,e}$.

Therefore, similarly to [DW21], we will assume that the span of the first $2n$ vectors of the basis and that of $\mathcal{L}^{a,e}$ behave like random $2n$ and ν dimensional spaces accordingly. Therefore their intersection is expected to be trivial with high probability in the $(2n+\nu)$-dimensional space. Hence, if $l \leqslant n$ we can conclude that $\dim(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}) = 0$ and if $l > n$ we get $\dim(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}) = l - n$.

We performed some preliminary experiments in order to verify this estimation. We varied the parameter l and computed the dimension of the intersection of $\mathcal{L}^{G,A,q,\nu}_{[0:n+l)}$ with the secret sublattice $\mathcal{L}^{a,e}$. For $l < n$ we observed a series of zeros. Then at roughly $l = n$ we noticed the first non-trivial intersection. From that point on, as l was increasing the dimension of the intersection was increasing. Usually increasing l by 1 increased the dimension of the intersection by 1. In some cases increasing l by 1 would increase the dimension of the intersection by 2 but then in the next step the dimension would stay the same, "balancing" the overall increase of the dimension. These experiments seem to match our suggested model.

We summarise our claim in the following heuristic assumption.

Assumption 6.13. *Let $\mathcal{L}^{G,A,q,\nu}$ be a Ring-LWE lattice with dense sublattice $\mathcal{L}^{a,e}$ of dimension ν. At the stage when BKZ detects a vector in $\mathcal{L}^{a,e}$ for the first time, it holds that* $\dim(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}) = \max(l-n, 0)$.

Therefore we are interested in the cases $n < l \leqslant n + \nu$.

Heuristic result 6.14. *Let (n, q, G, A, a, e) be a Ring-LWE instance with parameters $q = \Theta(n^{\mathcal{Q}})$ and $\|(\mathbf{e}|\mathbf{t}|\mathbf{a})\| = O(n^{\mathcal{S}})$. For $1 < \mathcal{Q}$ and $0 < \mathcal{S} \leqslant 1$, the BKZ algorithm with block size $\beta = \mathcal{B}n$ applied to $\mathcal{L}^{G,A,q,\nu}$ triggers a DSD event if*

$$\mathcal{B} = \frac{2}{1 + \mathcal{Q} - 2\mathcal{S}} + o(1). \tag{6.9}$$

This is the asymptotically minimum block size for $1 \leqslant \nu \leqslant n$, and is attained at $\nu = 1$.

Proof. Equation (6.7) provides a sufficient condition for a DSD event to occur. Therefore, following the analysis in [DW21] we will estimate $\ln\|\pi_{n+l-\beta}(\mathbf{v})\|$ and $\ln\|\mathbf{b}^*_{n+l-\beta}\|$. Set $l = Ln$ and $\nu = \mathcal{V}n$.
Step 1. We start by estimating $\ln\|\pi_{n+l-\beta}(\mathbf{v})\|$:

$$\begin{aligned}\ln\|\pi_{n+l-\beta}(\mathbf{v})\| &\approx \ln\sqrt{\frac{\beta}{n+l}} + \ln\|\mathbf{v}\| \\ &= \frac{1}{2}\ln\frac{\mathcal{B}}{1+L} + \ln\|\mathbf{v}\| \\ &= \ln\|\mathbf{v}\| + O(1).\end{aligned}$$

In order to bound $\ln\|\mathbf{v}\|$ we use Theorem 1.15:

$$\begin{aligned}\ln\|\mathbf{v}\| &= \ln\left\|\lambda_1\left(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}\right)\right\| \\ &\leqslant \frac{1}{2}\ln(l-n) + \frac{1}{l-n}\ln\mathrm{Vol}\left(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e}\right) \qquad \text{(by Cor. 6.12)} \\ &\leqslant \frac{1}{2}\ln(l-n) + \frac{1}{l-n}\left(\ln\mathrm{Vol}(\mathcal{L}^{a,e}) - \ln\left(\prod_{j=n+l}^{2n+\nu-1}\|\mathbf{b}^*_j\|\right)\right).\end{aligned}$$

For $\ln\mathrm{Vol}(\mathcal{L}^{a,e})$ we use the Hadamard bound and thus we get

$$\begin{aligned}\ln\|\mathbf{v}\| &\leqslant \frac{1}{2}\ln(l-n) + \frac{1}{l-n}\left(\mathcal{S}\nu\ln n - \sum_{j=n+l}^{2n+\nu-1}\ln\|\mathbf{b}^*_j\| + O(n)\right) &(6.10)\\ &\leqslant \left(\frac{1}{2} + \frac{\mathcal{S}\mathcal{V}}{L-1}\right)\ln n - \frac{1}{(L-1)n}\sum_{j=n+l}^{2n+\nu-1}\ln\|\mathbf{b}^*_j\| + O(1). &(6.11)\end{aligned}$$

At this point the analysis cannot follow the same path as in the case of NTRU. In particular, if we were to continue the analysis as in [DW21] then we should use the ZGSA in order to estimate the sum of $\ln \|\mathbf{b}_j^*\|$. However for the Ring-LWE case the summation starts at $j = n + l$ with $l > n$ which leads to $2n < j$. According to the ZGSA for $n + m \leqslant i$ it holds that $\ln \|\mathbf{b}_i^*\| = 0$. As we have assumed a "Z" shape for the profile which implies that $m < n$, and that $2n < j$ we conclude that $m+n < 2n < j$. Therefore, the sum of $\ln \|\mathbf{b}_j^*\|$ in (6.11) will be equal to zero. The main problem implied by a trivial sum over $\ln \|\mathbf{b}_j^*\|$ is that it actually means that Corollary 6.12 gives a uniform bound to $\mathrm{Vol}(\mathcal{L}^{G,A,q,\nu}_{[0:n+l)} \cap \mathcal{L}^{a,e})$ which is independent of l, thus not a bound that can be used to determine the best value (some preliminary experiments verify this). Such a weak bound leads to a loose analysis and therefore it is not useful.

Apart from not being useful, using the ZGSA assumption for the analysis of DSD in Ring-LWE would also not be realistic. This is illustrated by our experimental results (see Figure 6.2). According to our experiments the profile of the basis at the point of detection follows the GSA assumption model instead of the ZGSA assumption model in that there are no flat parts. Hence, we will continue our analysis under the GSA assumption. Thus equation (6.11) gives that

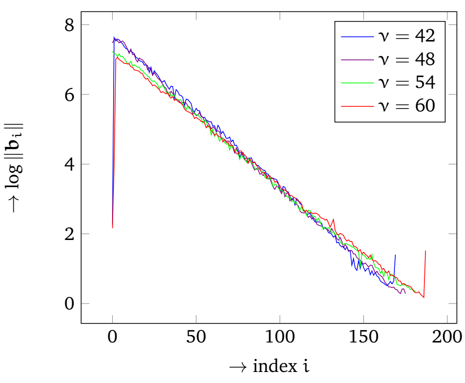
$$\begin{aligned}
\ln \|\boldsymbol{v}\| &\leqslant \left(\frac{1}{2} + \frac{\mathcal{S}\mathcal{V}}{L-1}\right) \ln n - \frac{1}{(L-1)n} \sum_{j=n+l}^{2n+\nu-1} \ln \|\mathbf{b}_j^*\| + O(1) \\
&\leqslant \left(\frac{1}{2} + \frac{\mathcal{S}\mathcal{V}}{L-1}\right) \ln n \\
&\quad - \frac{1}{(L-1)n} \sum_{j=n+l}^{2n+\nu-1} \left[\left(\frac{\mathcal{Q}}{2+\mathcal{V}} + \frac{2+\mathcal{V}}{2\mathcal{B}} - \frac{j}{\mathcal{B}n} \right) \ln n - jO\left(\frac{1}{n}\right) \right] + O(1) \\
&\leqslant \left(\frac{1}{2} + \frac{\mathcal{S}\mathcal{V}}{L-1}\right) \ln n - \frac{1+\mathcal{V}-L}{L-1} \left(\frac{\mathcal{Q}}{2+\mathcal{V}} - \frac{L+1}{2\mathcal{B}} + o(1) \right) \ln n + O(1) \\
&\leqslant \left[\frac{1}{2} + \frac{\mathcal{S}\mathcal{V}}{L-1} - \frac{1+\mathcal{V}-L}{L-1} \left(\frac{\mathcal{Q}}{2+\mathcal{V}} - \frac{L+1}{2\mathcal{B}} \right) + o(1) \right] \ln n + O(1).
\end{aligned}$$

<u>Step 2.</u> Estimating $\ln \|\mathbf{b}_{n+l-\beta}^*\|$.
Estimating this quantity can be done by directly applying the GSA:

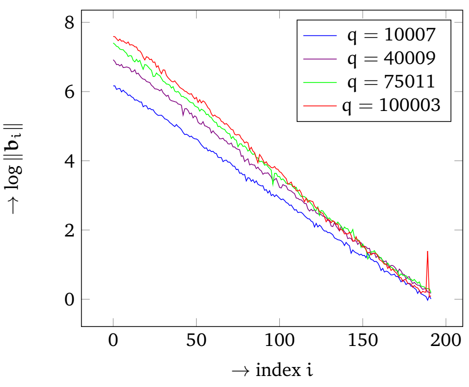
$$\begin{aligned}
\ln \|\mathbf{b}_{n+l-\beta}^*\| &= \ln \left(q^{n/(2n+\nu)} \alpha_\beta^{(2\beta-2l+\nu-1)/2} \right) \\
&= \frac{n}{2n+\nu} \ln q + \frac{2\beta - 2l + \nu - 1}{2} \ln \alpha_\beta \\
&= \left(\frac{\mathcal{Q}}{2+\mathcal{V}} + \frac{\mathcal{V} - 2L}{2\mathcal{B}} + 1 \right) \ln n + O(1) \qquad (6.12)
\end{aligned}$$

as $\ln \alpha_\beta = \frac{\ln n}{\mathcal{B}n} + O(\frac{1}{n})$.

<u>Step 3.</u> Comparing the estimations from Step 1 and Step 2.



(a) Profiles for $q = 40009$.



(b) Profiles for $\nu = 64$.

Figure 6.2: Experimental results on the profile of the basis at the point a DSD event occurs. We ran progressive BKZ with 8 tours on Ring-LWE lattices with $n = 64$ and for several values of q and ν. The secret and error distributions for all instances were chosen to be the Gaussian with $\sigma^2 = \frac{2}{3}$. The progressive BKZ was allowed to use a maximum block size $\min(n + \nu, 60)$. For each pair (q, ν) we tried 20 instances. Each plot represents a randomly chosen profile, out of those cases where DSD occurred before SKR. In all cases the profiles were similar, that is why show only one randomly chosen.

So, considering equation (6.7), a sufficient condition to be satisfied asymptotically is,

$$\frac{1}{2} + \frac{\mathcal{S}\mathcal{V}}{L-1} - \frac{1+\mathcal{V}-L}{L-1}\left(\frac{\mathcal{Q}}{2+\mathcal{V}} - \frac{L+1}{2\mathcal{B}}\right) \leqslant \frac{\mathcal{Q}}{2+\mathcal{V}} + \frac{\mathcal{V}-2L}{2\mathcal{B}} + 1 \Rightarrow$$
$$\frac{(2+\mathcal{V})(L^2-2L+2\mathcal{V}+1)}{(2+\mathcal{V})(L-1-2\mathcal{S}\mathcal{V}) + 2\mathcal{V}\mathcal{Q}} \leqslant \mathcal{B} \tag{6.13}$$

under the condition that the denominator is positive, i.e. $\mathcal{S} - \mathcal{Q}/(2+\mathcal{V}) < (L-1)/2\mathcal{V}$. Under this condition and $0 < \mathcal{V} \leqslant 1$ and $1 \leqslant L \leqslant 1 + \mathcal{V}$ we can minimize the left hand side of inequality (6.13) as a function of $\mathcal{V}$ and L. We set

$$f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, \mathcal{L}) = \frac{(2+\mathcal{V})(L^2-2L+2\mathcal{V}+1)}{(2+\mathcal{V})(L-1-2\mathcal{S}\mathcal{V}) + 2\mathcal{V}\mathcal{Q}}$$

and study this function in the aforementioned domain. It turns out (see Lemma 6.15) that

$$\frac{2}{1+\mathcal{Q}-2\mathcal{S}} < f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, \mathcal{L}) \tag{6.14}$$

and $f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, \mathcal{L})$ approaches the lower bound as $\mathcal{V} \to 0^+$. □

Lemma 6.15. *Let*

$$f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, L) = \frac{(\mathcal{V}+2)\left(L^2 - 2L + 2\mathcal{V} + 1\right)}{(\mathcal{V}+2)(L - 2\mathcal{S}\mathcal{V} - 1) + 2\mathcal{Q}\mathcal{V}}$$

on the domain $0 < \mathcal{V} \leqslant 1$*,* $1 \leqslant L \leqslant 1 + \mathcal{V}$*, with the denominator being positive, and with* $0 < \mathcal{S} \leqslant 1$*,* $\mathcal{Q} > 1$*. For all* $\mathcal{Q}, \mathcal{S}, \mathcal{V}, L$ *with the conditions above, we have*

$$f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, L) > \frac{2}{1 + \mathcal{Q} - 2\mathcal{S}},$$

where the inequality is strict.

Proof. We write $L = 1 + x\mathcal{V}$, where $0 \leqslant x \leqslant 1$. Then we get

$$f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, 1 + x\mathcal{V}) = \frac{(\mathcal{V}+2)(x^2\mathcal{V}+2)}{2\mathcal{Q} - (\mathcal{V}+2)(2\mathcal{S} - x)},$$

where in numerator and denominator a factor $\mathcal{V}$ has conveniently dropped out. Clearly we have

$$f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, 1 + x\mathcal{V}) \geqslant \frac{2(\mathcal{V}+2)}{2\mathcal{Q} - (\mathcal{V}+2)(2\mathcal{S} - 1)},$$

and

$$\frac{d}{d\mathcal{V}}\left(\frac{2(\mathcal{V}+2)}{2\mathcal{Q} - (\mathcal{V}+2)(2\mathcal{S} - 1)}\right) = \frac{4\mathcal{Q}}{(2\mathcal{Q} - (\mathcal{V}+2)(2\mathcal{S} - 1))^2}$$

is positive, so

$$f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, 1 + x\mathcal{V}) \geqslant \left.\frac{2(\mathcal{V}+2)}{2\mathcal{Q} - (\mathcal{V}+2)(2\mathcal{S} - 1)}\right|_{\mathcal{V}=0} = \frac{2}{1 + \mathcal{Q} - 2\mathcal{S}}.$$

Clearly this lower bound is indeed reached as a limit value for $f_{\mathcal{Q},\mathcal{S}}(\mathcal{V}, L)$. □

Remark 6.16. *In Heuristic result 6.14 we chose to focus on Ring-LWE instances with* $q = \Theta(n^{\mathcal{Q}})$ *and* $\|(\mathbf{e}|\mathbf{t}|\mathbf{a})\| = O(n^{\mathcal{S}})$*, where* $1 < \mathcal{Q}$ *and* $0 < \mathcal{S} \leqslant 1$*. This is due to mainly two reasons. Firstly, these choices are interesting from a cryptographic point of view and especially the choice of* $\mathcal{S} = 1/2$ *(ternary secrets). Like in the NTRU case, we expect the DSD events to occur for "higher"* q *and choosing* $1 < \mathcal{Q}$ *is also helpful in the proof of inequality* (6.14).

6.3.2 – The fatigue point. Heuristic result 6.14 implies some direct conclusions on the occurrence of DSD events. In particular, the fact that the minimum block size is reached for $\nu = 1$ implies that in this case the DSD event is "trivial". That is due to the fact that detecting a DSD event with a 1-dimensional secret sublattice will actually result in a SKR event. Namely, the DSD detection will output a multiple of the secret vector. Therefore it should not be expected for $\nu = 1$ to get a DSD event before a SKR event. Also, the results of our experiments (Figure 6.4) suggest that for "small" ν and "small" q it is unlikely to witness a DSD event before a SKR event. This is similar to the case of NTRU.

Nevertheless, our experiments in Figure 6.4 do not rule out the case for higher values of q or ν to allow a DSD event before a SKR event. For this reason we ran further experiments with $n = 64$, $q \in \{10007, 40009, 75011, 100003\}$ and $\nu \in \{30, \ldots, 64\}$ and focused on the order of the two events. Figure 6.3 shows the result of our experiments.

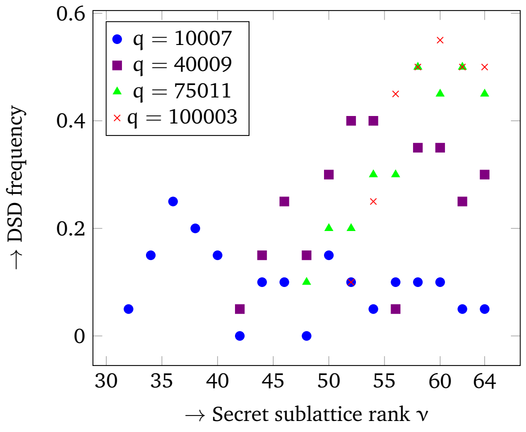


Figure 6.3: Experimental results on the frequency of DSD events occurring before SKR events. We ran progressive BKZ with 8 tours on Ring-LWE lattices with $n = 64$ and for several values of q and ν. The secret and error distribution for all instances was chosen to be the Gaussian with $\sigma^2 = \frac{2}{3}$. The progressive BKZ was allowed to use a maximum block size $\min(n+\nu, 60)$. Each point in the graph represents the frequency of DSDs amongst 20 instances.

Two main conclusions can be drawn from Figure 6.3. Initially, it appears that as q increases, also the minimum ν increases for which DSD events will start occurring before SKR events. This behaviour could be explained by the fact that as q increases SKR gets "easier" (Figure 6.4d) and thus for DSD to get even "easier" it requires a higher dimensional dense (secret) sublattice, hence a higher ν. It also seems that as q increases, the maximum frequency with which DSD events occur increases as well. This second observation resembles to the behaviour observed for NTRU in [DW21]. However in the case of Ring-LWE there does not seem to exist a *useful* fatigue point (i.e. a critical value of q after which DSD can happen before SKR).

In order to discuss a fatigue point in the case of Ring-LWE, the choice of ν should also be taken into account. In Figure 6.3 it was shown that for higher values of q and ν DSD events can occur before SKR events. A naive definition of a fatigue point in the case of Ring-LWE could be the following: the smallest q for which there exists a ν_0 such that DSD is triggered at a lower block size than SKR for the specific choice of ν. However, the value of ν_0 could be large and it is not necessary to consider the same value of ν when studying SKR and DSD.

As it is shown in Figure 6.4d, for each q, the minimum block size needed for SKR is expected to be achieved for a "small" value of ν. However, in Figure 6.3 it was made clear that DSD events start to get triggered before SKR only for relatively large values of ν. Thus, a more accurate definition of a fatigue point would be the following: the smallest q for which the minimum block size needed for DSD over all $\nu \in \{1, \dots, n\}$ is smaller than the minimum block size needed for SKR over all $\nu \in \{1, \dots, n\}$.

Let (n, q, G, A, a, e) be a Ring-LWE instance with parameters $q = \Theta(n^{\mathcal{Q}})$ and $\|(\mathbf{e}|\mathbf{t}|\mathbf{a})\| = O(n^{\mathcal{S}})$, where $1 < \mathcal{Q}$ and $0 < \mathcal{S} \leqslant 1$. We set $\mathcal{L}_{SKR} = \mathcal{L}^{G,A,q,\nu_0}$ for some "small" $\nu_0 = o(n)$ to be the lattice used for the SKR (i.e. it requires the minimum block size over all $\nu \in \{1, \dots, n\}$) and similarly $\mathcal{L}_{DSD} = \mathcal{L}^{G,A,q,\nu_1}$ be the lattice used for the DSD, for some ν_1. Thus, for a fatigue point to exist, there should exist values of q for which the block size needed for SKR in $\mathcal{L}_{\mathrm{SKR}}$ is higher than the block size for the DSD in $\mathcal{L}_{\mathrm{DSD}}$. In Heuristic result 6.14 we obtained a minimum block size for DSD. Hence, if there was a fatigue point, the block size needed for SKR in $\mathcal{L}_{\mathrm{SKR}}$ should be at least higher than this minimum.

Under the assumption that $\nu_0 = o(n)$, the $\mathcal{L}_{\mathrm{SKR}}$ lattice is of dimension $(2+o(1))n$, it is a q-ary lattice and has $\mathrm{Vol}(\mathcal{L}_{\mathrm{SKR}}) = q^n$. Therefore, we can consider it to asymptotically behave as the NTRU lattice does in [DW21]. In particular, we can directly transfer the asymptotic estimate on the necessary block size for SKR implied by the "2016-estimate". Namely, for $\mathcal{S} \leqslant 1$ we get a block size estimate of $2\mathcal{Q}/(1+\mathcal{Q}-\mathcal{S})^2$. Hence, for a DSD event to occur at a lower block size than a SKR event, asymptotically it should hold that,

$$\frac{2}{1+\mathcal{Q}-2\mathcal{S}} < \frac{2\mathcal{Q}}{(1+\mathcal{Q}-\mathcal{S})^2} \quad \Rightarrow \quad \mathcal{Q} < -(\mathcal{S}-1)^2$$

which gives a contradiction. Therefore, under the assumption that the best block size for SKR is reached for some $\nu_0 = o(n)$ we conclude that there is no useful fatigue point for the case of Ring-LWE.

6.4 — Implications for attack performance

Considering a larger family of lattices modelling Ring-LWE could have implications for the performance of known lattice attacks. In particular, the "generalised" lattices which were introduced in Section 6.2.2 are of increasing dimension and symmetry. In Section 6.3 we took advantage of this property in order to describe an analysis of whether Ring-LWE can get overstretched. However, this new trade-off between the dimension of the used lattice and its symmetry yields a new "dimension" in the attack surface of the problem. Therefore, it should be investigated if the best attack to Ring-LWE does lie in the "trivial" case where the used lattice has the minimum dimension but also the minimum symmetry.

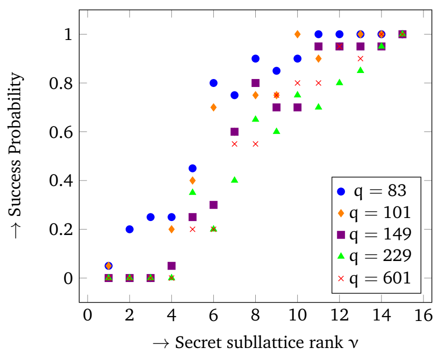
Pursuing this line of research we ran a set of experiments. For this purpose, we adapted the implementations of [DW21][3] to the case of Ring-LWE. In our experiments we focused on two aspects. The first one concerns the success probability of the algorithm when given specific resources (i.e. block size up to $\min(n+\nu, 60)$), while the second concerns the actual required block size for Secret Key Recovery (SKR). The results of our experiments are shown in Figure 6.4.

However, before we start discussing our findings there is a specific technical detail which we would like to point out. Initially, we tried to run our experiments using BKZ without any restrictions in sage [TSD19] (i.e. progressive, limited number of tours). Without these restrictions the BKZ was "too strong" and could solve all cases, no matter the rank of the lattice. In the implementation from [DW21] the authors built a progressive version of BKZ and also provided the choice of restricting BKZ to a predetermined number of tours. This is a choice that we kept. Apparently, this was a good idea as it "weakened" BKZ and thus the algorithm became more "sensitive" to the actual difficulty of the problem.

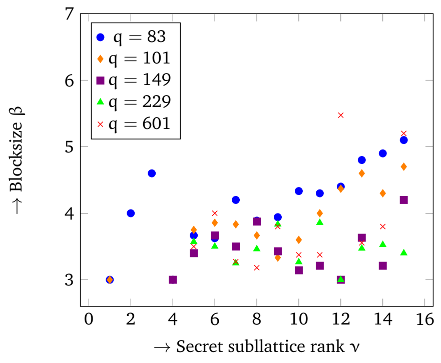
Our experiments are divided in two main groups, those with length $n = 32$ (Figures 6.4a, 6.4b) and those with $n = 64$ (Figures 6.4c, 6.4d). For both cases the entries

[2]Except for a couple of cases for $q = 601$, where DSD occurred first.

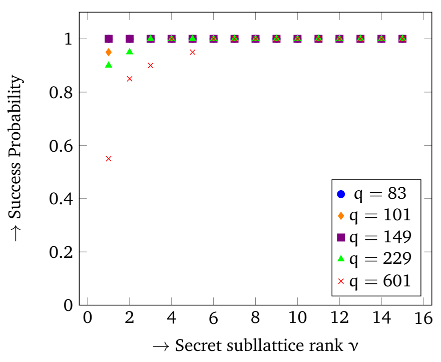
[3]Available at `https://github.com/WvanWoerden/NTRUFatigue`



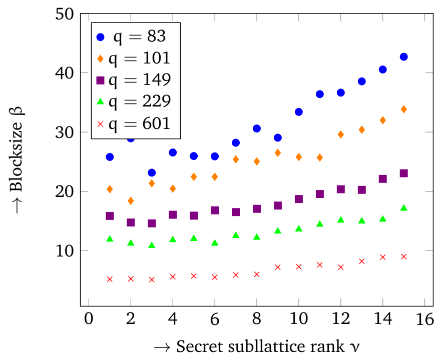
(a) $n = 32$.



(b) $n = 32$.



(c) $n = 64$.



(d) $n = 64$.

Figure 6.4: Experimental results on the performance of progressive BKZ with 8 tours on Ring-LWE with $n = 32, 64$ and for several values of q and ν. The secret and error distribution for all instances was chosen to be the Gaussian with $\sigma^2 = \frac{2}{3}$. The progressive BKZ was allowed to use a maximum block size $\min(n + \nu, 60)$. Each point in all four graphs represents the average amongst 20 instances. On the left (Figures 6.4a and 6.4c) the success probability of the algorithm on Secret Key Recovery (SKR) is shown, whereas on the right (Figures 6.4b and 6.4d) the minimum block size at which the SKR occurred is shown.[2]

of the secret and error vector were sampled from the discrete Gaussian distribution with $\sigma^2 = \frac{2}{3}$. For each n, we ran a series of experiments for $q \in \{83, 101, 149, 229, 601\}$ and $\nu \in \{1, 2 \ldots 15\}$. Regarding the values of q, we decided to stop at $q = 601$ as for the aforementioned values of n, ν and higher values of q the problem becomes trivial (see Figures 6.4b, 6.4d).

Starting with the case of $n = 32$, Figure 6.4a shows how the success probability of BKZ can be affected by the choice of ν. The most obvious observation is the overall increase of the success probability as ν increases. However, if we focus more closely on the "plots" for each value of q, we can see that they interleave. Thus, we cannot conclude in this case that one plot is clearly above another. Nevertheless, it seems that the blue dots ($q = 83$) are above all others for almost all cases. In addition we can observe that for some values of q we do get a zero success probability for $\nu \leqslant 4$.

In Figure 6.4b it is shown how the minimum block size for SKR is affected by the choice of ν. Apparently, for the case ($n = 32$), the problem was quite easy when the success probability was greater than zero, as we notice block sizes only in the range $3-5$. Although this is trivial in practice, it is worth noticing that the necessary block size did not significantly increase as the rank of the used lattice (and the success probability) increased.

For the case of $n = 64$ (Figures 6.4c, 6.4d), we get a more clear picture of the effect of ν in the performance of the attack. In particular, in Figure 6.4c we can clearly notice that as q increases the effect of ν is getting greater. However, for the case of $n = 64$ the effect of ν seems to kick in for higher values of q compared to the case $n = 32$. In Figure 6.4d we get a clear picture of how the block size is affected by ν. In this case the data points behave less erratically compared to the case $n = 32$. As it can be seen, the plots for each q are clearly above each other without intersecting. Thus, we notice that as q increases the problem gets easier, like in the case of NTRU. Also, the data suggest that the block size tends to (finally) increase as ν increases.

However, it seems that the minimum block size is not reached at $\nu = 1$. Instead, the minimum is reached at some $\nu \in \{2, 3\}$. This supports our intuition, namely that the minimum block size does not lie at $\nu = 1$ but it could also not be reached “too far” from $\nu = 1$ as this would result to higher block sizes. Of course, at this point it would be interesting to get an asymptotic analysis suggesting which is the optimal choice of ν which minimizes the block size. However, we chose to focus on this problem from an experimental point of view and leave such an analysis for future work. The only claim we want to make at this point is the following. If the optimal ν which minimizes the block size is not $\nu = 1$ (which is suggested by our experiments) then it probably will not be constant, but should increase as n increases.

Chapter 7

Conclusions and open problems

As a final chapter of this thesis we summarise research questions and open problems emerging from Chapters 2 through 6.

7.1 — Research questions on approximate Voronoi cells

In Chapter 2 we studied CVPP in terms of approximate Voronoi cells, and obtained better time and space complexities using randomized slicing, which is similar in spirit to using randomized bases in lattice enumeration [GNR10]. With this approach, an improvement was reached upon previously known complexities for CVPP, both theoretically and experimentally. The main research paths left for future work from Chapter 2 were (i) obtain a tighter asymptotic analysis from the one in Lemma 2.12, (ii) the utilisation of a CVPP solver to build faster hybrid enumeration methods, and (iii) the optimisation of CVPP methods in moderate dimensions, where the performance is studied from a practical point of view instead of an asymptotic one.

Chapter 2 contains the chronologically first work of this thesis. As it turned out, the time between the publication of that work and these lines are being written was enough for the scientific community to deal with two out of the three open questions mentioned above. In particular, the heuristic proof/derivation of Lemma 2.12 turned out to not be tight, as it was later shown by Ducas, Laarhoven and van Woerden in [DLvW20]. In addition, in Chapter 4 of this thesis we studied potential use of a CVPP solver in building faster hybrid algorithms. Thus, it is only problem (iii) of the above paragraph which actually remains open.

7.2 — Research questions on irreducible vectors

In Chapter 3 we introduced a notion of irreducible vectors with the belief that it will motivate further research in the field of lattices. In this chapter we focused only on pairwise irreducibility of vectors, even though a definition of higher order irreducibility was also given. In particular pairwise irreducibility appears to have a close relation to lattice sieving algorithms. Thus, it could be that the set $\mathrm{P}(\mathcal{L})$ can provide further insight in this area. An interesting question would be if the usage of the set $\mathrm{P}(\mathcal{L})$ (under some heuristic assumptions on its size) enables the proof of an upper bound on the time complexity of the GaussSieve [MV10b]. Examining the properties and the utility of higher order irreducibility is left for future research.

The implications of $P(\mathcal{L})$ in cryptanalytic attacks could be an interesting topic to investigate. The set $P(\mathcal{L})$ is expected to be affected by an underlying structure in the lattice $\mathcal{L}$. It can thus be expected that structured lattices end up with a smaller set $P(\mathcal{L})$ than "average-case" lattices. Many of the modern lattice-based cryptosystems possess such underlying structures and hence they could serve as interesting cases to examine from this point of view.

In Section 3.4.1 we argued that computing $P(\mathcal{L})$ by "brute force" can take up to $\tilde{O}(2^{2n})$ time. Therefore, this can serve as an upper bound. However, this bound may not be tight as discussed in Section 3.4.2. In Section 3.4.2 modified sieving algorithms were utilised in order to show how to compute $P(\mathcal{L})$ asymptotically. But the question of how to compute it exactly or approximately in practice remains open. Such a result would also imply the ability to compute a subset of $R(\mathcal{L})$ (of heuristically exponential size) without requiring the set $R(\mathcal{L})$.

The set $P(\mathcal{L})$ can be used as a tool in proving a behaviour of a lattice algorithm but could also be used by itself (e.g. as preprocessing data of a CVPP algorithm). In Section 3.5 we proposed the use of the "tuple slicer" in order to utilise the set $P(\mathcal{L})$ in the CVPP framework. However this algorithm introduces a new question, namely what size of tuples should be considered during this algorithm. Figure 3.4 attempts to give some preliminary experimental evidence on this problem. However, a theoretical analysis of this question is left for future work.

Section 3.7 provided some experimental evidence showing that the size of a set $P(\mathcal{L})$ could vary a lot in some cases. An "average-case" result implying that if the underlying lattice is not "special" then the size of $P(\mathcal{L})$ cannot vary a lot would be of interest. A potential tool to reaching such a result could be lattice theta functions [Elk09]. This is due to the fact that the coefficients in a lattice's theta function actually represent the number of lattice vectors of a specific length. Therefore this property reveals the connection to the definition of $P(\mathcal{L})$.

7.3 — Research questions on hybrid algorithms

In Chapter 4 we examined hybrid lattice algorithms for SVP based on algorithms for CVPP. Our study examined the (heuristic) asymptotic performance of these hybrids as well as their practical performance. However, apart from our results there is a number of questions arising from the research in Chapter 4.

Besides performing more extensive experiments, which may assist in obtaining estimates for the crossover points between these hybrids and plain lattice sieving, open problems arising from Chapter 4 include (i) finding a way to effectively incorporate pruning into the enumeration parts of the proposed hybrids; (ii) further studying the theoretical and practical relevance of the proposed nested hybrid algorithms, and their relation with progressive sieving ideas [Duc18,LM18]; and (iii) finding improvements for CVPP, potentially using a dual distinguisher. After the publication of the work described in Chapter 4, Laarhoven and Walter studied (iii) in their work "Dual lattice attacks for closest vector problems (with preprocessing) [LW21]".

7.4 — Research questions on guessing patterns

In Chapter 5 we discussed some combinatorial aspects in guessing a pattern of zeros in a lattice vector. In Proposition 5.4 we provided an exact formula for the success probability of the consecutive zeros pattern. This enabled a comparison between the consecutive zeros pattern and random patterns. Even though this comparison is not exact in the sense that the result in [MS01] is approximate, it provides more concrete evidence in what was before just an experimental observation. Therefore we have addressed an interesting special case of question 1 in Section 5.1. Proving Heuristic Assumption 5.6 would make this claim even more robust. However, this task is probably interesting only as an intellectual challenge, as an attacker can always plug in values for N, d, r and check its validity for his case of interest.

By defining the index of symmetry $\eta(J)$ of a pattern J we attempted to provide a concept which would reflect some of its properties, like the maximum number of times it can occur in a vector (multiple winners). Whether or not this concept is an optimal choice remains an interesting open combinatorial problem. What would definitely be very interesting is a theoretical result predicting the success probability of a pattern J with regard to its value $\eta(J)$. This would explain Figure 5.5 and it would make the computation of such profiles unnecessary for attackers.

Finally, in Chapter 5 we described how guessing patterns of zeros can provide a success probability-time trade-off or even a probability-memory trade-off. Testing the potential benefits of such trade-offs in practice with concrete parameters and attacks is left for future research.

7.5 — Research questions on Ring-LWE's lattices

In Chapter 6 we investigated how a larger set of lattices could be used in the analysis of the Ring-LWE problem. We mainly focused on applying this idea within the so-called "overstretched" framework. In our study we approached this problem both from a theoretical point of view (heuristic asymptotics) as well as from an experimental point of view. We adopted the approach of [DW21] and studied if Ring-LWE could get overstretched, like in the case of NTRU. Even though we concluded in a negative answer, our result covers this existing gap in the analysis of Ring-LWE.

Additionally, we examined how this larger set of lattices modelling Ring-LWE could affect in practice the performance of lattice reduction attacks. Some preliminary experiments suggested that the optimal choice may not lie at the commonly used lattice of rank $2n + 1$. It would be interesting to get an asymptotic analysis suggesting which lattice modelling Ring-LWE provides the best performance (block size, success probability). Of course, further experiments supporting such an asymptotic analysis would be of interest.

Summary

Lattice cryptanalysis: theoretical and practical aspects

Cryptography is of utmost importance in our connected and digitalised world. It enables the secure implementation of many online services which empower the technological development and financial growth of our society. Some examples are online payments, sending emails, sending messages from our phone or having online meetings. Cryptography guarantees the confidentiality, integrity and authenticity of our data against malicious entities while these data are transmitted through an insecure channel (e.g. the internet).

Currently used public-key encryption and digital signature schemes depend on the hardness of mathematical problems, such as the factorization of integers which are the product of two (big) primes, or the discrete logarithm problem. These problems have been studied extensively for the past 45 years and seem to be adequately hard (up to now). However, that could change within the next years or decades. The advent of large scale quantum computers would enable the use of quantum algorithms, breaking these problems extremely faster than the currently used classical algorithms. Under this assumption, it is made clear that a new generation of cryptography has to be designed, based on mathematical problems which should withstand classical as well as quantum attacks. One such option for post-quantum cryptography is lattice-based cryptography.

This thesis is devoted to the study of some lattice problems underlying lattice-based cryptography. The first part of this thesis discusses some theoretical aspects of lattice problems. In particular, in Chapter 2 the use of approximate Voronoi cells is examined in solving the closest vector problem with preprocessing (CVPP). The use of approximate Voronoi cells combined with a so-called randomised slicer results in better time and space complexities for CVPP. Besides that, a notion of irreducibility for lattice vectors is introduced in Chapter 3. It turns out that the set of irreducible vectors of a lattice is a subset of its Voronoi relevant vectors. Apart from this property which allows a connection to already known theory we also notice that there is a close relation to the output of lattice sieving algorithms.

The second part of this thesis is devoted to a more practical study of lattice problems. In Chapter 4 we examine hybrid algorithms for the Shortest Vector Problem (SVP) through ones for the closest vector problem with preprocessing. This idea takes advantage of the work in Chapter 2 and combines it (mainly) with lattice enumeration algorithms. In lattice-based cryptography it is a common practice to choose secret lattice vectors containing many zeros. This fact motivated a study in Chapter 5 of combinatorial aspects in guessing a pattern of zeros in a vector. More precisely, we focus on a comparison between

the consecutive zeros pattern and random patterns while also introducing a potential "measure" of "good" patterns. Finally, one of the widely used building blocks in modern lattice-based cryptography is the Ring-LWE problem. Chapter 6 focuses on this problem and investigates the potential benefits of a "wider" set of lattices modelling Ring-LWE. In more detail, we examine this idea within the so-called "overstretched" framework. Furthermore we investigate the practical implications of this idea within lattice-basis reduction attacks.

Curriculum Vitae

Emmanouil Doulgerakis was born on November 20, 1993, in Rethymno, Greece. In 2015, he completed his bachelor's degree in Mathematics at University of Crete. For his performance in his undergraduate studies he received the Pixoridi award. In the same year, he started his master in Pure Mathematics at University of Crete which he completed in May of 2017 under the supervision of Jannis Antoniadis. The title of his master thesis was The Number Field Sieve and its Applications in Factorization. For his performance in his master studies he received the Manasaki scholarship.

Also in May 2017, he started his PhD project in the Coding theory and Cryptology group at the Eindhoven University of Technology under the supervision of Benne de Weger and Tanja Lange. The project was funded by the NWO grant 628.001.028 (FASOR). During his PhD studies he studied some theoretical as well as practical aspects of solving lattice problems, focusing on those of cryptographic interest.

Bibliography

[ABD16] Martin Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In *Proceedings of the 36th CRYPTO*, pages 153–178. Springer, 2016. doi:10.1007/978-3-662-53018-4_6.

[ADH$^+$19] Martin Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *Proceedings of the 38th EUROCRYPT*, pages 717–746. Springer, 2019. doi:10.1007/978-3-030-17656-3_25.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange: a new hope. In *Proceedings of the 25th USENIX Security Symposium*, pages 327–343, 2016. doi:10.5555/3241094.3241120.

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time via discrete Gaussian sampling. In *Proceedings of the 47th STOC*, pages 733–742, 2015. doi:10.1145/2746539.2746606.

[ADSD15] Divesh Aggarwal, Daniel Dadush, and Noah Stephens-Davidowitz. Solving the closest vector problem in 2^n time – the discrete Gaussian strikes again! In *Proceedings of the 56th FOCS*, pages 563–582, 2015. doi:10.1109/FOCS.2015.41.

[AEVZ02] Erik Agrell, Thomas Eriksson, Alexander Vardy, and Kenneth Zeger. Closest point search in lattices. *Transactions on Information Theory*, 48(8):2201–2214, 2002. doi:10.1109/TIT.2002.800499.

[AKS01] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the 33rd STOC*, pages 601–610. ACM Press, 2001. doi:10.1145/380752.380857.

[AN17] Yoshinori Aono and Phong Q. Nguyen. Random sampling revisited: lattice enumeration with discrete pruning. In *Proceedings of the 36th EUROCRYPT*, pages 65–102. Springer, 2017. doi:10.1007/978-3-319-56614-6_3.

[Bab86] László Babai. On Lovasz lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. doi:10.1007/BF02579403.

[BCLV17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: reducing attack surface at low cost. In *Proceedings of the 24th SAC*, pages 235–260, 2017. doi:10.1007/978-3-319-72565-9_12.

[BCLv19] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime, 2019. Submission to NIST post-quantum call for proposals. URL: https://ntruprime.cr.yp.to/nist/ntruprime-20190330.pdf.

[BD15] Nicolas Bonifas and Daniel Dadush. Short paths on the Voronoi graph and the closest vector problem with preprocessing. In *Proceedings of the 26th SODA*, pages 295–314. ACM-SIAM, 2015. doi:10.1137/1.9781611973730.22.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the 27th SODA*, pages 10–24. ACM-SIAM, 2016. doi:10.1137/1.9781611974331.ch2.

[BG14] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In *Proceedings of the 19th Information Security and Privacy*, pages 322–337. Springer, 2014. doi:10.1007/978-3-319-08344-5_21.

[BL21] Daniel J. Bernstein and Tanja Lange. Non-randomness of s-unit lattices. Cryptology ePrint Archive, Paper 2021/1428, 2021. https://eprint.iacr.org/2021/1428. URL: https://eprint.iacr.org/2021/1428.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *LMS Journal of Computation and Mathematics*, 19(A):146–162, 2016. doi:10.1112/S1461157016000292.

[Cha02] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th STOC*, pages 380–388. ACM Press, 2002. doi:10.1145/509907.509965.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low-level encoding of zero. *LMS Journal of Computation and Mathematics*, 19(A):255–266, 2016. doi:10.1112/S1461157016000371.

[CMF19] Filipe Cabeleira, Artur Mariano, and Gabriel Falcao. Memory-optimized Voronoi cell-based parallel kernels for the shortest vector problem on lattices. In *Proceedings of the 27th EUSIPCO*, pages 1–5. IEEE, 2019. doi:10.23919/EUSIPCO.2019.8902635.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Proceedings of the 17th ASIACRYPT*, pages 1–20. Springer, 2011. doi:10.1007/978-3-642-25385-0_1.

[CS92] John H. Conway and Neil J.A. Sloane. Low-dimensional lattices. VI. Voronoi reduction of three-dimensional lattices. In *Proceedings of the Mathematical and Physical Sciences*, volume 436, pages 55–68. The Royal Society, 1992. doi:10.1098/rspa.1992.0004.

[CS98] John H. Conway and Neil J.A. Sloane. *Sphere packings, lattices and groups*. Springer, 1998.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.

[DLdW19] **Emmanouil Doulgerakis**, Thijs Laarhoven, and Benne de Weger. Finding closest lattice vectors using approximate Voronoi cells. In *Proceedings of the 10th PQCRYPTO*, pages 3–22. Springer, 2019. doi:10.1007/978-3-030-25510-7_1.

[DLdW20] **Emmanouil Doulgerakis**, Thijs Laarhoven, and Benne de Weger. Sieve, enumerate, slice, and lift: Hybrid lattice algorithms for svp via cvpp. In *Proceedings of the 12th AFRICACRYPT*, pages 301–320. Springer, 2020. doi:10.1007/978-3-030-51938-4_15.

[DLdW21] **Emmanouil Doulgerakis**, Thijs Laarhoven, and Benne de Weger. The irreducible vectors of a lattice: Some theory and applications. *Preprint*, 2021. https://eprint.iacr.org/2021/1203.pdf.

[DLvW20] Léo Ducas, Thijs Laarhoven, and Wessel van Woerden. The randomized slicer for CVPP: sharper, faster, smaller, batchier. In *Proceedings of the 23rd PKC*, pages 3–36. Springer, 2020. doi:10.1007/978-3-030-45388-6_1.

[Duc18] Léo Ducas. Shortest vector from lattice sieving: a few dimensions for free. In *Proceedings of the 37th EUROCRYPT*, pages 125–145. Springer, 2018. doi:10.1007/978-3-319-78381-9_5.

[DW21] Léo Ducas and Wessel van Woerden. NTRU fatigue: How stretched is overstretched ? In *Proceedings of the 27th ASIACRYPT*, pages 3–32. Springer, 2021. doi:10.1007/978-3-030-92068-5_1.

[Elk09] Noam Elkies. Theta functions and weighted theta functions of Euclidean lattices, with some applications. http://people.math.harvard.edu/~elkies/aws09.pdf, 2009.

[FCMP19] Gabriel Falcao, Filipe Cabeleira, Artur Mariano, and Luis Paulo Santos. Heterogeneous implementation of a Voronoi cell-based SVP solver. *IEEE Access*, 7:127012–127023, 2019. doi:10.1109/ACCESS.2019.2939142.

[FK15] Masaharu Fukase and Kenji Kashiwabara. An accelerated algorithm for solving SVP based on statistical analysis. *Journal of Information Processing*, 23(1):67–80, 2015. doi:10.2197/ipsjjip.23.67.

[FP85] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice. *Mathematics of Computation*, 44(170):463–471, 1985. doi:10.2307/2007966.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *Proceedings of the 29th EUROCRYPT*, pages 257–278. Springer, 2010. doi:10.1007/978-3-642-13190-5_13.

[Ham50] Richard Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950. doi:10.1002/j.1538-7305.1950.tb00463.x.

[HG07] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *Proceedings of the 27th CRYPTO*, pages 150–169, 2007. doi:10.1007/978-3-540-74143-5_9.

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in Euclidean norm. In *Proceedings of the 20th PKC Part I*, pages 16–40. Springer, 2017. doi:10.1007/978-3-662-54365-8_2.

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving. In *Proceedings of the 21st PKC*, pages 407–436. Springer, 2018. doi:10.1007/978-3-319-76578-5_14.

[Hop18] Max Hopkins. Representation-theoretic techniques for independence bounds of Cayley graphs. Bachelor thesis, 2018.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory Symposium*, pages 267–288, 1998. doi:10.1007/BFb0054868.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Proceedings of the 3rd IWCC*, pages 159–190, 2011. doi:10.1007/978-3-642-20901-7_10.

[HRS19] Christoph Hunkenschröder, Gina Reuland, and Matthias Schymura. On compact representations of Voronoi cells of lattices. In *Proceedings of the 20th IPCO*, volume 11480 of *Lecture Notes in Computer Science*, pages 261–274. Springer, 2019. doi:10.1007/s10107-019-01463-3.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm (extended abstract). In *Proceedings of the 27th CRYPTO*, pages 170–186. Springer, 2007.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th STOC*, pages 193–206. ACM Press, 1983. doi:10.1145/800061.808749.

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In *Proceedings of the 36th EUROCRYPT*, pages 3–26. Springer, 2017. doi:10.1007/978-3-319-56620-7_1.

[KL78] Grigory Kabatiansky and Vladimir Levenshtein. Bounds for packings on a sphere and in space. *Problemy Peredachi Informatsii*, 14:3–25, 1978.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Proceedings of the 35th CRYPTO*, pages 3–22. Springer, 2015. doi:10.1007/978-3-662-47989-6_1.

[Laa16a] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2016. https://pure.tue.nl/ws/portalfiles/portal/14673128/20160216_Laarhoven.pdf.

[Laa16b] Thijs Laarhoven. Sieving for closest lattice vectors (with preprocessing). In *Proceedings of the 23rd SAC*, pages 523–542. Springer, 2016. doi:10.1007/978-3-319-69453-5_28.

[Laa16c] Thijs Laarhoven. Tradeoffs for nearest neighbors on the sphere, 2016. https://arxiv.org/abs/1511.07527.

[Laa19] Thijs Laarhoven. Approximate Voronoi cells for lattices, revisited. In *Proceedings of the 1st MATHCRYPT*, 2019. https://arxiv.org/pdf/1907.04630.pdf.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. doi:10.1007/BF01457454.

[LM18] Thijs Laarhoven and Artur Mariano. Progressive lattice sieving. In *Proceedings of the 9th PQCRYPTO*, pages 292–311. Springer, 2018. doi:10.1007/978-3-319-79063-3_14.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986. doi:10.1145/22145.22146.

[LvdPdW12] Thijs Laarhoven, Joop van de Pol, and Benne de Weger. Solving hard lattice problems and the security of lattice-based cryptosystems. *Cryptology ePrint Archive, Report 2012/533*, pages 1–43, 2012. http://eprint.iacr.org/2012/533.

[LW21] Thijs Laarhoven and Michael Walter. Dual lattice attacks for closest vector problems (with preprocessing). In *Proceedings of CT-RSA*, pages 478–502, 2021. doi:10.1007/978-3-030-75539-3_20.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*. Kluwer Academic Publishers, Boston, Massachusetts, 2002.

[Mic01] Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, 2001. doi:10.1109/18.915688.

[Min11] Hermann Minkowski. In *Gesammelte Abhandlungen von Hermann Minkowski*, volume 2, pages 103–121. 1911.

[MM65] J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965. doi:10.1007/BF02760024.

[MS01] Alexander May and Joseph H. Silverman. Dimension reduction methods for convolution modular lattices. In *Proceedings of Cryptography and Lattices*, pages 110–125. Springer, 2001. doi:10.1007/3-540-44670-2_10.

[MV10a] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proceedings of the 42nd STOC*, pages 351–358. ACM Press, 2010. doi:10.1145/1806689.1806739.

[MV10b] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the 21st SODA*, pages 1468–1480. ACM-SIAM, 2010. doi:10.1137/1.9781611973075.119.

[MW15] Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *Proceedings of the 26th SODA*, pages 276–294, 2015. doi:10.1137/1.9781611973730.21.

[Ngu09] Phong Q. Nguyen. Hermite's constant and lattice algorithms. In *The LLL Algorithm: Survey and Applications*, pages 19–69. Springer, 1st edition, 2009.

[NS09] Phong Q. Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. *ACM Transactions on Algorithms*, 5(4):46:1–46:48, 2009. doi:10.1145/1597036.1597050.

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. doi:10.1515/JMC.2008.009.

[Ros64] Mosche Rosenfeld. Independent sets in regular graphs. *Israel Journal of Mathematics*, 2:262–272, 1964. doi:10.1007/BF02759743.

[RS96] Dayanand S. Rajan and Anil M. Shende. A characterization of root lattices. *Discrete Mathematics*, 161:309–314, 1996. doi:10.1016/0012-365X(95)00239-S.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. doi:10.1145/359340.359342.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2):201–224, 1987. doi:10.1016/0304-3975(87)90064-8.

[SFS09] Naftali Sommer, Meir Feder, and Ofir Shalvi. Finding the closest lattice point by iterative slicing. *SIAM Journal of Discrete Mathematics*, 23(2):715–731, 2009. doi:10.1137/060676362.

[Sho94] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th FOCS*, pages 124–134. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365700.

[Sin00] Simon Singh. *The code book: the secret history of codes and code-breaking*. Fourth Estate, 2000.

[Ste16] Noah Stephens-Davidowitz. Dimension-preserving reductions between lattice problems. http://noahsd.com/latticeproblems.pdf, 2016.

[SUR19] SURFsara. The Lisa cluster. https://userinfo.surfsara.nl/systems/lisa, 2019.

[svp19] SVP Challenge, 2019. https://www.latticechallenge.org/svp-challenge/.

[The19a] The FPLLL development team. fplll, a lattice reduction library. Available at https://github.com/fplll/fplll, 2019.

[The19b] The g6k development team. The general sieve kernel (G6K). Available at https://github.com/fplll/g6k, 2019.

[TSD19] The Sage Developers. Sagemath, the Sage Mathematics Software System. https://www.sagemath.org, 2019.

[VB96] Emanuele Viterbo and Ezio Biglieri. Computing the Voronoi cell of a lattice: The diamond-cutting algorithm. *IEEE Transactions on Information Theory*, 42:161–171, 1996. doi:10.1109/18.481786.

[Vor08] Georgy Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.