Master Thesis

University of Bergen

Department of Informatics

Introduction to Lattices and Its Applications in Compute-and-Forward Strategy

Author:
Maria van der Reek Lidsheim

Supervisors:
Hsuan-Yin Lin,
Maiara Francine Bollauf,
Øyvind Ytrehus

June 1, 2023

Acknowledgements

I want to thank my supervisors Hsuan-Yin Lin, Maiara Francine Bollauf and Øyvind Ytrehus for the guidance and mentoring throughout the thesis and the courses related to it. I would also like to thank my friends and family for being there for me and supporting me along the way.

Abstract

The Compute-and-Forward (CF) strategy was proposed as a physical layer network coding (PNC) framework by Nazer and Gastpar in 2011. CF exploits interference to obtain higher rates between users in a network. This thesis focuses on studying the application of the lattice network coding (LNC) for CF strategy using maximum-likelihood (ML) decoding through four influential papers in the area.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

Nazer and Gastpar proposed a new strategy for communication in wireless networks called Compute-and-Forward (CF) [1]. CF works in a scenario where each of the transmitting nodes sends a message to a distant receiver through an intermediate relay. The relay receives a noisy linear combination of the incoming packets and forwards a representation of this combination to the receiver. In the traditional approach, each packet is viewed as an annoying interference to the others, and in the common interpretation, interference equates to noise. Such an approach lowers the possible transmission rate. Instead, the CF strategy aims to recover the noiseless linear equations of the transmitted messages using the noisy linear combination input provided by the channel. The receiver at the destination can solve the linear system for its desired messages with enough linear combinations and recover the message. Lattice Network Coding (LNC) was proposed as a promising approach by applying lattices for the CF strategy [1]. In particular, by mapping transmitted messages over a finite field into points of several lattices, LNC ensures that the linear combination of the lattice points can be decoded reliably and works in a way that the decoder recovers a linear system of equations of the transmitted lattices' points.

In [2,3], the authors study LNC for CF using Maximum Likelihood (ML) decoding, where the goal is to maximize the conditional probability of the received signal at one relay, given the sum of the transmitted signals through the channel. The ML decoder aims to maximize the conditional probability formula, which is shown to be equivalent to studying the so-called *flatness factor* for the additive independent and identically distributed (i.i.d.) Gaussian noise. We will discuss the details in Chapter 3.2.4. The authors in [3] explain why it is deemed important that the flatness factor of the designed lattice of LNC has to be as large as possible to prevent bad ML decoding performance [3], making it possible to recover the correct message. One important thing to note from [3] is that they conjecture that maximizing the ML metric is equivalent to solving an inhomogeneous Diophantine approximation in large dimensions. Note that the flatness factor definition presented in [2] is slightly different from that of [3], which has studied the flatness factor's properties more deeply and how it can be related to LNC for CF using ML decoding.

It is worth mentioning that a closely related parameter of the flatness factor, called the *smoothing parameter* of a lattice, has been introduced to be a key feature in [4] for lattice-based cryptography. The close relations between the flatness factor and smoothing parameter have also been studied in [5], where the authors' has focused on applying the flatness factor for security application over Gaussian wiretap channels.

In [6], the authors consider the same LNC approach over Gaussian channels as [2, 3] for CF strategy using ML decoding, but focus on the implementation in practical aspects, where only a one-dimensional lattice and two users are considered. Moreover, it is shown that in the one-dimensional case, the implementation of the ML decoding is equivalent to using an *Inhomogeneous Diophantine Approximation* algorithm.

In [7], the authors consider the practical aspects of the CF strategy with LNC based on ML decoding like [6]. For one-dimensional lattice, they examined sub-optimal decoding techniques like the Diophantine approximation and the Minimum Mean Square Error Generalized Decision Feedback Equalizer (MMSE-GDFE), applied by the Sphere Decoder [7, Sec. IV]. Moreover, they extended the study to the multi-dimensional case for the design of LNC. They concluded the paper by saying that more research is required on the error probability function and lattice constructions that optimize the flatness factor for multi-dimensional lattices.

In [2], the author presented an analysis of the LNC for ML decoding. The author has introduced a notion of the flatness factor. In [3], they studied the properties of the flatness factor after changing the definition to be more consistent with [4, 5]. In [6], the authors studied a decoding technique for one-dimensional finite constellations. In [7], they consider the practical implementation of the CF strategy and study the case for a two-dimensional lattice coding. Despite using different techniques, all papers [2, 3, 6, 7] build on the application of LNC for CF using ML decoding but for different lattices, where the ideas are based on the inhomogeneous Diophantine approximation, the use of the flatness factor of a lattice and the HNF algorithm. In this thesis, we want to understand the differences in the approaches presented in the papers.

1.2 Objective

We present a literature review together with an analysis of the implemented methods, including the CF framework and the connection with lattices. In terms of specific objectives, we aim to:

- Explain how lattice coding can be applied in LNC and how it can improve the achievable coding rate.
- Describe how the HNF algorithm can help analyze LNC using ML decoding.
- Provide practical and theoretical examples through code snippets, graphs, and tables.
- Show how the diversity order and constellation size affect the results.

Finally, we also point out perspectives of future work.

1.3 Thesis Organization

The thesis is organized as follows:

- **Chapter 2:** Preliminary concepts from information theory
- **Chapter 3:** Review of the proposed papers [2,3,6,7]
- **Chapter 4:** Conclusions and future work
- **Chapter 5:** Appendix

Nomenclature

EM	Electro-magnetic
QPSK	Quadrature Phase Shift Keying
BPSK	Binary Phase Shift Keying
BER	Bit Error Rate
PAM	Pulse Amplitude Modulation
CF	Compute-and-Forward
SNR	Signal-to-Noise Ratio
AWGN	Additive White Gaussian Noise
NN	Nearest-Neighbor
BEP	Bit Error Probability
WLAN	Wireless Local-Area Network
HNF	Hermite Normal Form
LLL	Lenstra-Lenstra-Lovasz algorithm
SVP	Shortest Vector Problem
CVP	Closest Vector Problem
ML	Maximum-likelihood
PNC	Physical Layer Network Coding
MIMO	Multiple-input Multiple-output
PID	Principle Ideal Domain
LNC	Lattice-based Network Coding
VNR	Volume-to-Noise Ratio
CSI	Channel Side Information
MMSE	Minimum Mean Square Error
dB	Decibel

gcd	Greatest Common Divisor
$\boldsymbol{x}$	n-dimensional vector
X	Matrix
$\mathcal{X}$	Set
X	Random variable
$\mathbb{R}$	Set of real numbers
$\mathbb{Z}$	Set of Integers
$\mathrm{FT}\{\cdot\}$	Fourier Transform
$\dagger$	Hermitian Transpose
$d_{\mathrm{E}}(\boldsymbol{x}, \boldsymbol{y})$	Euclidean Distance
$\lVert\boldsymbol{x}\rVert$	The Euclidean norm of a vector $\boldsymbol{x}$
$\langle\cdot\rangle$	Inner Product
R	Bit Rate
C	Shannon Capacity
$\mathcal{R}$	Ring
i	Imaginary number $(\sqrt{-1})$
$\mathsf{R}_{\mathrm{comp}}$	Computation Rate
GF	Galois Field/Finite Field
Λ	Lattice
$\boldsymbol{\lambda}$	Lattice point
$\Lambda^{\star}$	Dual lattice
$\Lambda_{\boldsymbol{x}}$	Coset of a lattice Λ
Q_{Λ}	Lattice Quantizer
$r_{\mathrm{pack}}(\Lambda)$	Packing Radius of a lattice Λ
$r_{\mathrm{cov}}(\Lambda)$	Covering Radius of a lattice Λ
$\mathcal{V}_0$	Voronoi cell
$\mathcal{P}(\Lambda)$	Fundamental Region of a lattice Λ
$Q_{\Lambda}^{\mathrm{NN}}$	Nearest-Neighbor Quantizer
ε_{Λ}	Flatness Factor of a lattice Λ

G	Lattice Generator Matrix
$p(\boldsymbol{y}/\boldsymbol{\lambda})$	Conditional Probability Function, where $\boldsymbol{\lambda}$ is a point of a lattice
$\mathsf{G}\mathsf{G}^{\mathsf{T}}$	Gram Matrix
$\mathrm{GL}_n(\mathbb{Z})$	Group of invertible matrices with integer coefficients
r	Radius
$\mathbb{Z}[i]$	Gaussian Integers
$\mathbb{Z}[\omega]$	Eisenstein Integers, where $\omega = e^{2\pi i/3}$
$\mathrm{Vol}(\Lambda)$	Volume of Fundamental Region of a lattice Λ
Θ_Λ	Theta series of a lattice Λ

Chapter 2

Preliminary Concepts from Codes, Lattices, and Information Theory

2.1 Algebra

2.1.1 Finite Fields

Definition 2.1.1. A finite field is a field that contains only finitely many elements. A field is an integral domain, which is a ring with identity and no zero divisors. For example, $\mathbb{Z}/(p)$ with p a prime has p elements. It is also called a Galois field. The order is always a prime or a power of a prime. For each prime power, there exists exactly one finite field GF(p^n).

Example 2.1.1. For example, the finite field GF(2) consists of elements 0 and 1 which satisfies the following addition and multiplication tables:

$+$	0	1
0	0	1
1	1	0

Table 2.1: Addition table for GF(2)

$\times$	0	1
0	0	0
1	0	1

Table 2.2: Multiplication table for GF(2)

2.1.2 Principal Ideal Domains

From [8], Principle Ideal Domain (PID) is explained as follows.

Definition 2.1.2. For a ring $\mathcal{R}$, its nonzero elements is denoted by $\mathcal{R}/\{0\}$. If $ab = 0$, for some $b \in \mathcal{R}$, an element in $\mathcal{R}/\{0\}$ is called a zero-divisor. The integers $\mathbb{Z}$ form a PID, where a PID is an integral domain where every ideal is principle.

Definition 2.1.3. Gaussian integers are the set $\mathbb{Z}[i] \triangleq \{a + bi \colon a, \in \mathbb{Z}\}$ [8].

Remark. The Gaussian integers have four units ($\pm 1, \pm i$), and are called a Gaussian prime if it is a prime in $\mathbb{Z}[i]$. This happens if and only if it satisfies one of the following [8]:

- $|a| = |b| = 1$
- Either $|a|$ or $|b|$ are zero and the other is a prime number in $\mathbb{Z}$ of the form $4j + 3$.
- Both $|a|$ and $|b|$ are nonzero and $a^2 + b^2$ is a prime number in $\mathbb{Z}$ of the form $4j + 1$.

Definition 2.1.4. Eisenstein integers are the set $\mathbb{Z}[\omega] \triangleq \{a + b\omega \colon a, b \in \mathbb{Z}\}$. Eisenstein integers are complex numbers with integer real and imaginary parts in a particular lattice in the complex plane [8].

Remark. The Eisenstein integers $\mathbb{Z}[w]$ have six units, and are called an Eisenstein prime if it is a prime in $\mathbb{Z}[w]$ [8]. This happens if and only if

- $a + b\omega$ is a product of a unit in $\mathbb{Z}[\omega]$ and a prime number in $\mathbb{Z}$ of the form $3j + 2$.
- $|a + b\omega| = a^2 - ab + b^2$ is a prime number in $\mathbb{Z}$.

2.1.3 Inhomogeneous Diophantine Approximation

The inhomogeneous Diophantine approximation approximates real numbers by rationals with a given denominator. We are given a real number α and a positive integer q:

$$\left|\alpha - \frac{p}{q}\right| < \left|\alpha - \frac{p'}{q'}\right|. \tag{2.1}$$

One way to find the inhomogeneous Diophantine approximation is to use the extended Euclidean algorithm [9, Ch. 4.3].

2.2 Lattices

A lattice is a discrete set of points within a additive subgroup. If a point $\boldsymbol{\lambda}$ is in the lattice, then its inverse is also in the lattice. Since a lattice is an additive subgroup of $\mathbb{R}^n$, then for two points $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$, their respective sum is also in the lattice. The lattice Λ contains all integer multiples of any lattice point $\boldsymbol{\lambda}$ and all integer linear combination of all combinations of any two lattice points since it is an infinite set [10].
The definition of a lattice Λ is given by:

Definition 2.2.1. A lattice is a discrete subset of $\mathbb{R}^n$, which can be represented as

$$\Lambda = \{\boldsymbol{x} = \boldsymbol{u}\mathsf{G}_{m\times n} \colon \boldsymbol{u} = (u_1, ..., u_m) \in \mathbb{Z}^m\}, \tag{2.2}$$

where the m rows of G is a linearly independent set of vectors in $\mathbb{R}^n$. When $m = n$, the lattice Λ is said to be full-rank.

The generator matrix is not unique for each lattice.

Now we will present an example from [11].

Example 2.2.1. Consider the set $\mathcal{A}_2$ of all vectors in $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) \in \mathbb{Z}^3$ such that $\boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3 = 0$. This set is parameterized by letting two coordinates be free and forcing the third one to be negative sum of the two free coordinates (if we let $\boldsymbol{x}_1, \boldsymbol{x}_3$ free, then $\boldsymbol{x}_2 = -\boldsymbol{x}_1 - \boldsymbol{x}_3$), showing that we can describe $\mathcal{A}_2$ by integer linear combinations of two independent vectors. This is a rank 2 lattice in $\mathbb{R}^3$, since a generator matrix for it is

$$\mathsf{G} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{pmatrix}. \tag{2.3}$$

Definition 2.2.2. The Gram matrix is $\mathsf{G}\mathsf{G}^\intercal$. If the elements of the Gram matrix are integers, then the lattice is called integral. If the determinant of the Gram matrix is 1, then the lattice is called unimodular [10, Ch. 2.5].

Example 2.2.2. We will now give an example of a Gram Matrix. Suppose we are given two vectors:

$$\boldsymbol{g}_1 = (1, 3, 5), \boldsymbol{g}_2 = (2, 4, 6). \tag{2.4}$$

To compute the Gram matrix, we need to compute $\mathsf{G}\mathsf{G}^\intercal$, where $\mathsf{G}^\intercal$ denotes the transpose of G.

$$\begin{aligned} \mathsf{G} &= \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \\ \mathsf{G}^\intercal &= \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \\ \mathsf{G}\mathsf{G}^\intercal &= \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \\ &= \begin{pmatrix} 1\cdot 1 + 3\cdot 3 + 5\cdot 5 & 1\cdot 2 + 3\cdot 4 + 5\cdot 6 \\ 2\cdot 1 + 4\cdot 3 + 6\cdot 5 & 2\cdot 2 + 4\cdot 4 + 6\cdot 6 \end{pmatrix} \\ &= \begin{pmatrix} 35 & 44 \\ 44 & 56 \end{pmatrix} \end{aligned}$$

Therefore, we know the Gram matrix $\mathsf{G}\mathsf{G}^\intercal$ for the vectors $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ is

$$\mathsf{G}\mathsf{G}^\intercal = \begin{pmatrix} 35 & 44 \\ 44 & 56 \end{pmatrix} \tag{2.5}$$

Definition 2.2.3. The Euclidean distance [11, Ch. 2] between two points $\boldsymbol{x} = (x_1, ..., x_n)$, $\boldsymbol{y} = (y_1, ...y_n) \in \mathbb{R}^n$ is given by:

$$d_{\mathrm{E}}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \cdots + (y_n - x_n)^2}. \tag{2.6}$$

Example 2.2.3. Say we are given two vectors $\boldsymbol{x} = (1, 2)$ and $\boldsymbol{y} = (3, 4)$. The Euclidean distance between them are:

$$\begin{aligned} d_{\mathrm{E}}(\boldsymbol{x}, \boldsymbol{y}) &= \sqrt{(3-1)^2 + (4-2)^2} \\ &= \sqrt{2^2 + 2^2} \\ &= \sqrt{8} = 2\sqrt{2}. \end{aligned}$$

A lattice is a periodic arrangement of points in the n-dimensional Euclidean space. For higher dimensions, the problems of packing and covering are not equivalent. Packing and covering defined as given in [10, Ch. 3.1.1 and 3.1.2]:

Definition 2.2.4. For a given lattice Λ and a radius r, the set $\Lambda + B_r$ is a packing [10, Ch. 3.1.1] in the Euclidean space, where $\mathcal{B}$ is the ball surrounding the lattice point λ, if for all distinct lattice points $\boldsymbol{\lambda}, \boldsymbol{\lambda}' \in \Lambda$, we have

$$(\boldsymbol{\lambda} + B_r) \cap (\boldsymbol{\lambda}' + B_r) = \emptyset. \tag{2.7}$$

This means that as long as the spheres do not intersect, it is a packing of the lattice Λ. The packing radius $r_{\text{pack}}(\Lambda)$ of the lattice is defined by the largest balls the lattice can pack:

$$r_{\text{pack}}(\Lambda) = \sup\{r \colon \Lambda + B_r \text{ is a packing}\}. \tag{2.8}$$

It is determined by how big the balls can get without overlapping with each other.

Now, we will define the covering of a lattice Λ.

Definition 2.2.5. The set $\Lambda + \mathcal{B}_r$ of spheres centered around a lattice point, is a covering [10, Ch. 3.1.2] in the Euclidean space if

$$\mathbb{R}^n \subseteq \Lambda + \mathcal{B}_r, \tag{2.9}$$

which means that each point λ is covered by at least one sphere, and the spheres can overlap and there is no space in the lattice Λ not covered by the spheres. The radius of the covering $r_{\text{cov}}(\Lambda)$ of the lattice Λ is

$$r_{\text{cov}}(\Lambda) = \min\{r \colon \Lambda + \mathcal{B}_r \text{ is a covering}\} \tag{2.10}$$

which is also the outer radius of the Voronoi cell, i.e., the minimum radius of a (closed) ball containing the fundamental Voronoi cell $\mathcal{V}_0$.

Definition 2.2.6. The volume of a (full rank) lattice Λ is given by:

$$\text{Vol}(\Lambda) = \det(\Lambda) = |\det(\mathsf{G})|, \tag{2.11}$$

where $\det(\cdot)$ denotes the determinant and G denotes the generator matrix of a lattice Λ.

2.2.1 Voronoi Cell

Broadly speaking, the Voronoi cell of a point is the set of all points in a space closer to that point than any other point in the set. The fundamental Voronoi cell is denoted as $\mathcal{V}_0$, and it is the Voronoi cell associated with the origin, which is the lattice point that equals the zero-vector $\boldsymbol{\lambda} = \mathbf{0}$. All Voronoi cells are shifted versions of the fundamental Voronoi cell $\mathcal{V}_0$ [10, Ch. 4]. A Voronoi partition uses a nearest-neighbour (NN) rule, which is the most important division of a given lattice Λ. We now define Voronoi partition as given in [10, Ch. 4].

Definition 2.2.7. Let $\|\cdot\|$ denote the Euclidean norm. The distance of a point $\boldsymbol{x}$ in $\mathbb{R}^n$ from Λ is defined as

$$\|\boldsymbol{x} - \Lambda\| \triangleq \min_{\boldsymbol{\lambda} \in \Lambda} \|\boldsymbol{x} - \boldsymbol{\lambda}\|. \tag{2.12}$$

The NN quantizer $Q_\Lambda^{(\mathrm{NN})}(\cdot)$ maps $\boldsymbol{x}$ to its closest lattice point:

$$Q_\Lambda^{(\mathrm{NN})}(\boldsymbol{x}) = \arg\min_{\boldsymbol{\lambda} \in \Lambda} \|\boldsymbol{x} - \boldsymbol{\lambda}\|, \tag{2.13}$$

and the Voronoi cell $\mathcal{V}_{\boldsymbol{\lambda}}$ is the set of all points which are quantized to $\boldsymbol{\lambda}$:

$$\mathcal{V}_{\boldsymbol{\lambda}} = \{\boldsymbol{x} \colon \; Q_\Lambda^{(\mathrm{NN})}(\boldsymbol{x}) = \boldsymbol{\lambda}\}. \tag{2.14}$$

Example 2.2.4. We are given three points:

$$\boldsymbol{u}_1 = (-1, 1), \; \boldsymbol{u}_2 = (2, 3), \; \boldsymbol{u}_3 = (0, -2).$$

The partition here involves three cells, one for each point, so we know that the points in each cell are closer to that point than to the other two points. The boundaries between each cell are called Voronoi edges. The perpendicular bisectors of the connecting points form the Voronoi edges. We need to find the perpendicular bisectors of the segments connecting the points. The perpendicular bisector of the segment connecting $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ is the line passing through in the middle of that segment, which is $(1/2, 2)$. Perpendicular to the vector pointing from $\boldsymbol{u}_1$ to $\boldsymbol{u}_2$ is $(2, 2)$. We can call these two lines $\boldsymbol{x}$ and $\boldsymbol{y}$, then we get the equation $\boldsymbol{x} - \boldsymbol{y} = (1/2, 2) - (2, 2) = (-3/2, 0)$, and therefore divides the plane into two. One cell is now the Voronoi cell for $\boldsymbol{u}_1$, while the other is the Voronoi cell for $\boldsymbol{u}_2$. We can do the same to find the Voronoi cell for $\boldsymbol{u}_3$ by considering the perpendicular bisectors of the segments connecting $\boldsymbol{u}_3$ with $\boldsymbol{u}_1$ or $\boldsymbol{u}_2$.

Figure 2.1 shows the cell around $\boldsymbol{u}_1$ will contain the set of points closer to $\boldsymbol{u}_1$ than to $\boldsymbol{u}_2$ or $\boldsymbol{u}_3$. The cell around $\boldsymbol{u}_2$ will contain the set of points closer to $\boldsymbol{u}_2$ than $\boldsymbol{u}_1$ or $\boldsymbol{u}_3$. The cell around $\boldsymbol{u}_3$ will contain the set of points closer to $\boldsymbol{u}_3$ than to $\boldsymbol{u}_1$ or $\boldsymbol{u}_2$.

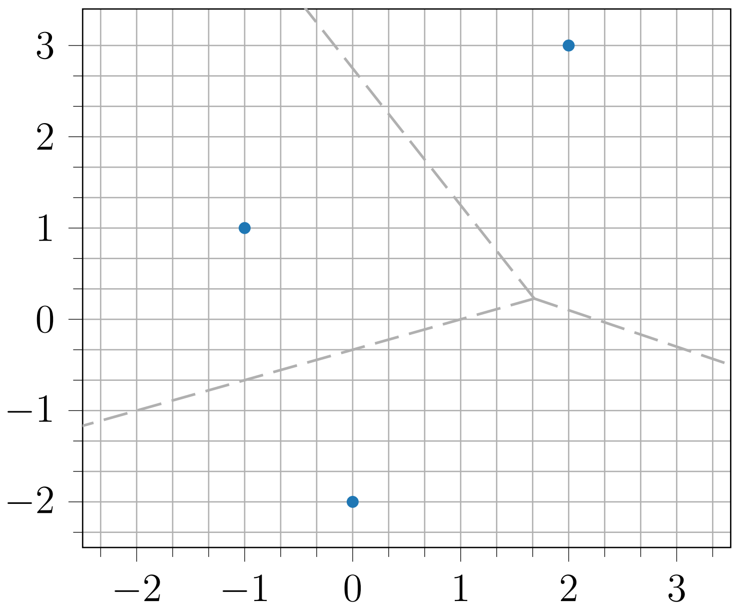
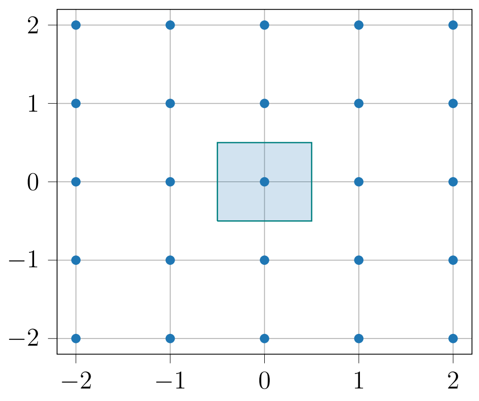
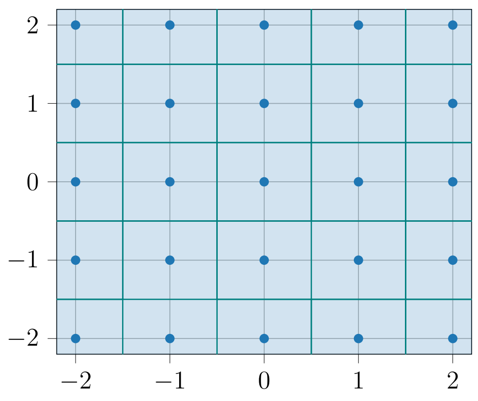


Figure 2.1: Voronoi partition of the points $\boldsymbol{u}_1$, $\boldsymbol{u}_2$ and $\boldsymbol{u}_3$.



(a) Voronoi cell of a cubic lattice $\mathbb{Z}^2$



(b) All Voronoi cells in a cubic lattice $\mathbb{Z}^2$

Example 2.2.5. We see in Figure 2.2a, the Voronoi region of the $\mathbb{Z}^2$ lattice, $\mathcal{V}_0$. In Figure 2.2b, we see that all the Voronoi cells are shown for each lattice point $\boldsymbol{\lambda}$ for the cubic lattice $\mathbb{Z}^2$.

2.2.2 Cosets

A coset is a discrete set of points such that the difference vector between every pair of points belongs to the lattice.

A coset is defined as found in [10, Ch. 2.1.2]:

Definition 2.2.8. For a lattice Λ and the vector $\boldsymbol{x} \in \mathbb{R}^n$:

$$\Lambda_{\boldsymbol{x}} = \boldsymbol{x} + \Lambda = \{\boldsymbol{x} + \boldsymbol{\lambda} \colon \boldsymbol{\lambda} \in \Lambda\}. \tag{2.15}$$

A coset is not a lattice since it does not consist of the origin. The union of $\Lambda_{\boldsymbol{x}}$ of all shifts $\boldsymbol{x}$ covers the entire space $\mathbb{R}^n$, but with many overlaps. A Voronoi cell is the set of all points which are quantized to $\boldsymbol{\lambda}$. The Voronoi partition refers to the Euclidean norm.

2.2.3 Lattice Codes and Nested Lattices

A lattice code is a code whose codewords are represented as the points from a lattice Λ. We want to use lattice codes because they have some symmetric properties which are relevant to approach communication problems. A q-ary linear code is a subspace of $\mathbb{F}_q$

Definition 2.2.9. A q-ary linear code $\mathcal{C}$, where q-ary means the finite field has q elements, induces a modulo-q lattice, which is a lattice Λ, where all operations are done with modulo q [11, Ch. 3]:

$$\Lambda_{\mathcal{C}} = \{\boldsymbol{x} \in \mathbb{Z}^n \colon \boldsymbol{x} \bmod q \in \mathcal{C}\}. \tag{2.16}$$

We will now give an example of such construction for $q = 2$.

Example 2.2.6. Say we are given the codeword $\mathcal{C} = \{(0,0,0,0), (1,0,1,1)\} \subseteq \mathbb{F}_2^4$. Then:

$$\Lambda_{\mathcal{C}} = \{(0,0,0,0) + 2(z_1,z_2,z_3,z_4)\} \bigcup \{(1,0,1,1) + 2(z_1,z_2,z_3,z_4)\} \tag{2.17}$$

$$\boldsymbol{z} = (z_1,z_2,z_3,z_4) \in \mathbb{Z} \tag{2.18}$$

Remark. An important property of a lattice code is that the cell volume is equal to the number of integer cosets, and is given by

$$\mathrm{Vol}(\Lambda_{\mathcal{C}}) = |\mathbb{Z}^n/\Lambda_{\mathcal{C}}| = q^n/M, \tag{2.19}$$

where M is the size of the code $\mathcal{C}$ [10, Ch. 2].

Nested lattices [10, Ch. 8.1] are a lattice construction where one lattice is contained within another. A nested lattice can provide a more efficient representation of the transmitted or encrypted data and they can reduce the complexity in lattice network coding regarding encoding and decoding. They allow a reduction in the amount of information that needs to be transmitted over the network by allowing the intermediate nodes to compute the linear combinations of the lattice points closer to the points at the destination.

Definition 2.2.10. A pair of lattices are nested if

$$\Lambda_2 \subset \Lambda_1, \tag{2.20}$$

meaning that Λ_2 is a sublattice of Λ_1.

Remark. Λ_1 is called the fine lattice, and Λ_2 is called the coarse lattice [10, Ch. 8.1].

2.2.4 Dual Lattice

A dual lattice defined as given in [10, Def. 4.2.3]:

Definition 2.2.11. Two lattices Λ and $\Lambda^\star$ in $\mathbb{R}^n$ are dual if their inner products of the points are integers. This means that $\langle \boldsymbol{\lambda}, \boldsymbol{\lambda}^\star \rangle \in \mathbb{Z}$ for all $\boldsymbol{\lambda} \in \Lambda$, $\boldsymbol{\lambda}^\star \in \Lambda^\star$. In other words, if G is a generator matrix of Λ, then $(\mathsf{G}^{-1})^\intercal$ is the generator matrix of $\Lambda^\star$.

2.2.5 Hermite Normal Form

We will now explain the HNF as given in [12].

Definition 2.2.12. We are given an $m \times n$ matrix $\mathsf{M} = m_{i,j}$ with integer coefficients. The matrix M is in HNF if $r \leq n$ and there exist a strictly increasing map from $[r+1, n]$ to $[1, m]$ which satisfies:

- For $r+1 \leq j \leq n$, $m_{f(j),j} \geq 1$, $m_{i,j} = 0$ if $i \geq f(j)$ and $0 \leq m_{f(k),j} < m_{f(k),k}$ if $k < j$.
- The first r columns of M are equal to 0.

Remark. In the important special case where $m = n$ and $f(k) = k$(or equivalently $\det(\mathsf{M}) \neq 0$), M is in HNF if it satisfies the following conditions.
(1) M is an upper triangular matrix, i.e. $m_{i,j} = 0$ if $i > j$.
(2) For every i, we have $m_{i,i} > 0$.
(3) For every $j > i$ we have $0 \leq m_{i,j} < m_{i,i}$.

More generally, if $n \geq m$, a matrix M in HNF has the following shape

$$\begin{pmatrix} 0 & 0 & \dots & 0 & * & * & \dots & * \\ 0 & 0 & \dots & 0 & 0 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & * \end{pmatrix}, \tag{2.21}$$

where the last m columns form a matrix in HNF.

We will now give a theorem for HNF [12, Thm. 2.4.3]:

Theorem 2.2.7. Let A be an $m \times n$ matrix with integer coefficients. Then there exists a unique $m \times n$ matrix $\mathsf{B} = (b_{i,j})$ in HNF of the form $\mathsf{B} = \mathsf{AU}$ with $\mathsf{U} \in \mathrm{GL}_n(\mathbb{Z})$, where $\mathrm{GL}_n(\mathbb{Z})$ is the group of matrices with integer coefficients which are invertible, i.e. whose determinant is equal to ± 1.

Remark. Even though B is unique, matrix U is not. The matrix W formed by the non-zero columns of B is the HNF of A [12].

We give the proof of Theorem 2.2.6 as Algorithm 1 from [12], where we are given an $m \times n$ matrix A with integer coefficients $(a_{i,j})$. The algorithm finds the HNF of A.

Algorithm 1: Finding the Hermite Normal Form

```
Input  : An $m \times n$ matrix $\mathsf{A}$ with integer coefficients $a_{i,j}$
Output: The matrix $\mathsf{W}$, which is the HNF of $\mathsf{A}$
$i \leftarrow m,\ k \leftarrow n$;

if $m \leq n$ then
    $l \leftarrow 1$;
    else
        $l \leftarrow m - n + 1$;
while Not finished do
    for $j < k$ do
        if $a_{i,j} = 0$ then
            if $a_{i,k} < 0$ then
                Replace column $A_k$ by $-A_k$;
for $j \leq k$ do
    if $a_{i,j} \neq 0$ then
        Choose min $|a_{i,j}|$ if $j_0 \leq k$ then
            Exchange column $A_k$ by $-A_k$;
$b \leftarrow a_{i,k}$ ;
for $j = 1, ..., k-1$ do
    $q \leftarrow \lfloor a_{i,j}/b \rceil,\ A_j \leftarrow qA_k$, go to step 2;
$b \leftarrow a_{i,k}$;
if $b = 0$ then
    $k \leftarrow k + 1$, go to step 6;
    if $j > k$ then
        $q \leftarrow \lfloor a_{i,j}/b \rceil,\ A_j \leftarrow A_j - qA_k$;
if $i = l$ then
    for $j = 1, ..., n - k + 1$ do
        $W_j = A_{j+k-1}$, Terminate algorithm;
    $i = i - 1,\ k = k - 1$, go to step 2;
return $\mathsf{W}$
```

The algorithm terminates since one can easily prove that $|a_{i,k}|$ is strictly decreasing each time we return to step 2 from step 4. Upon termination, it is clear that W is in HNF, and since it has been obtained from A by elementary column operations of determinant ± 1, W is the HNF of A.

We will now try to demonstrate how this can be done:
We give the basis for a lattice Λ by the following vectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ and $\boldsymbol{v}_3$:

$$\boldsymbol{v}_1 = (6, -4, -7), \tag{2.22}$$
$$\boldsymbol{v}_2 = (3, 5, 14), \tag{2.23}$$
$$\boldsymbol{v}_3 = (-3, 7, 16). \tag{2.24}$$

The first step is now to find the Greatest Common Divisor (gcd) of the first entries of the three vectors. We see that the gcd$(6, 3, -3)$ is 3. Since the gcd $= 3$, we want a vector that has a 3, which in our case is $\boldsymbol{v}_2$, in the first position to get rid of the leading coefficients of the other vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_3$.

$$\boldsymbol{v}_4 = \boldsymbol{v}_1 - 2\boldsymbol{v}_2 = (0, -14, -35) \tag{2.25}$$
$$\boldsymbol{v}_5 = \boldsymbol{v}_3 + \boldsymbol{v}_2 \quad = (0, 12, 30). \tag{2.26}$$

Next, we want to shorten the second entries of the vectors. We see that the gcd$(-14, 12)$ is 2, so we want to get as close to 2 as possible, so we get

$$\boldsymbol{v}_7 = -(\boldsymbol{v}_4 + \boldsymbol{v}_5) = (0, 2, 5). \tag{2.27}$$

Next, we see that if we add together $\boldsymbol{v}_4 + 7\boldsymbol{v}_7$, we get the $(0, 0, 0)$-vector.
At last, we want to look at $\boldsymbol{v}_2$. Here we can take

$$\boldsymbol{v}_2 - 2\boldsymbol{v}_7 = (3, 1, 4) \tag{2.28}$$

and we end up with the basis

$$\begin{pmatrix} 3 & 1 & 4 \\ 0 & 2 & 5 \\ 0 & 0 & 0 \end{pmatrix}. \tag{2.29}$$

We can check this with a program which calculates the HNF of a matrix.
The program used here is a short program using already existing functions which we will explain. The program can be found in Appendix B.1 and is as follows with the built-in libraries taken from [13]:

```
from hsnf import row_style_hermite_normal_form
import numpy as np

M = np.array([[6, -4, -7],
              [3, 5, 14],
              [-3, 7, 16]])

H, R = row_style_hermite_normal_form(M)

print(f'H = {H}')
```

The program uses the function `row_style_hermite_normal_form()` from Appendix B.1.1, which uses the class-method `with_standard_basis()` from Appendix B.1.2 and the function
`hermite_normal_form()` from Appendix B.1.3 in the function
`ZmoduleHomomorphish()` from Appendix B.1.4. The first function
`row_style_hermite_form()` from Appendix B.1.1 calculates the row-style HNF of the given matrix M. It uses the library `hsnf` to calculate the HNF.
The function takes one argument M, which is a NumPy array of shape (m, n), and contains the integer matrix for which the HNF is to be calculated.
The function returns a tuple (H, R), where H is the HNF of M and R is a unimodular matrix, so $\mathsf{H} = \mathsf{R} \cdot \mathsf{M}$. The HNF is an upper-triangular integer matrix that reduces the basis M to a better basis. The unimodular matrix R is a matrix with integer entries and determinant ± 1.

The function first creates an instance of the class `ZmoduleHomomorphishm()` from Appendix B.1.4, by using the basis M as the matrix representation. It then uses the method `hermite_normal_form()` from Appendix B.1.3 in the class
`ZmoduleHomomorphishm()` from Appendix B.1.4 to calculate the HNF and return the basis.
The class `ZmoduleHomomorphishm()` from Appendix B.1.4 has three instance variables:

- `_A`, which is a NumPy array containing the matrix representation of the homomorphism.
- `_basis_from`, which is a NumPy array containing the standard basis vector for the domain(source) of the homomorphism.
- `_basis_to`, which is a NumPy array containing the standard basis vector for the co-domain(target) of the homomorphism.

The class `ZmoduleHomomorphishm()` from Appendix B.1.4 uses two methods in this case, which are `_hnf_row()` from Appendix B.1.5 and
`hermite_normal_form()` from Appendix B.1.3.

- The method `_hnf_row()` calculates the HNF using row operations and returns the HNF of the matrix and the unimodular matrix.
- The method `hermite_normal_form()` copies the instance variables
`_A`, `_basis_from` and `_basis_to` and then calls the method `_hnf_row()` to calculate the HNF of the matrix. It then reverts the instance variables to their original values and returns the result.

When we run the program, we get that the better basis, or the HNF for

$$\boldsymbol{v}_1 = (6, -4, -7), \tag{2.30}$$
$$\boldsymbol{v}_2 = (3, 5, 14), \tag{2.31}$$
$$\boldsymbol{v}_3 = (-3, 7, 16), \tag{2.32}$$

turns out to be

$$\begin{pmatrix} 3 & 1 & 4 \\ 0 & 2 & 5 \\ 0 & 0 & 0 \end{pmatrix}. \tag{2.33}$$

If we run the program on the second matrix, we see that we get the same basis so we know that it is the best basis, since it is already in HNF.

2.2.6 The use for Hermite Normal Form

We can use HNF to solve Diophantine equations [6] and to determine if a Diophantine equation has solutions and what the solutions are. The HNF yields a basis where the vectors are shorter and more orthogonal, which makes the computations simpler and more accurate. The Lenstra-Lenstra-Lovász (LLL) algorithm is the most used and is generally faster than the HNF algorithm, especially when it comes to larger matrices. The HNF on the other hand can provide a unique representation of the lattice, whereas the LLL only provides an approximation of the lattice basis. We can also use the HNF to find a better basis for a lattice and to solve for the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP) [11]. The SVP is the problem of finding the shortest non-zero vector in a given lattice. The problem wants to find the non-zero vector $\boldsymbol{x} \in \Lambda$ with the smallest Euclidean norm $\|\boldsymbol{x}\|$, which is the lattice point $\boldsymbol{\lambda}$ closest to the origin.
The CVP is the problem of finding the lattice point closest to a given $\boldsymbol{y} \in \mathbb{R}^n$. For a given lattice Λ and a vector $\boldsymbol{y} \in \mathbb{R}^n$, the problem aims to find $\boldsymbol{\lambda} = \arg\min_{\boldsymbol{x}\in\Lambda} \|\boldsymbol{y} - \boldsymbol{x}\| \in \Lambda$. The same applies here to the basis optimization problem. The HNF will transform the problems of SVP and CVP into simpler problems by transforming the basis into an HNF basis, all though for some small dimensions the SVP and CVP will still be hard problems.

The LLL algorithm is a commonly used algorithm for basis reduction in lattice problems. The algorithm takes as input a basis for a lattice and produces a new basis that is almost as good as the HNF basis but with a much lower computational cost. It reduces the size of the basis vectors and improves the orthogonality of the basis. We will not go into more depth on the LLL algorithm, as it is outside the scope of this thesis, but if the reader wants to learn more we refer to this book [14, p. 439]. The HNF algorithm works well for small matrices, while for bigger matrices, the algorithm can be computationally expensive, since the number of operations required to perform the algorithm increases.

2.2.7 Theta Series of a Lattice

The theta series [15] is the generating function for the number of vectors with norm n in the lattice and is called the theta function of a lattice. There are tables where you can find the first terms of the theta series for various lattices, indicating how the lattice looks and is structured. The theta series is a power series with coefficients that tell you the number of lattice points $\boldsymbol{\lambda}$ at each distance from the origin of the lattice. The function of the theta series of a lattice is given by [2]:

$$\Theta_\Lambda(q) = \sum_{\boldsymbol{x}\in\Lambda} q^{\|\boldsymbol{x}\|^2}, \tag{2.34}$$

where $q = e^{i\pi z}$, $\mathrm{Im}(z) > 0$ is a complex variable, Λ is the lattice. The theta series is a modular form which means the modular group is a discrete subgroup of the group of linear transformations, which satisfies some properties. Properties of the theta series are:

- It is a modular form of weight $n/2$, where n is the dimension of the lattice, which means it is invariant under modular transformations of a complex variable q.
- It satisfies the Jacobi inversion formula [16]. The formula let the theta series be a product of certain theta functions associated with the dual lattice of the original lattice.

2.2.8 The Jacobi Theta Function

The definition of the Jacobi theta function is as follows [17]:

Definition 2.2.13.

$$\vartheta_2(z) = \sum_{m=-\infty}^{\infty} q^{(m+1/2)^2}, \tag{2.35}$$

$$\vartheta_3(z) = \sum_{m=-\infty}^{\infty} q^{m^2}, \tag{2.36}$$

$$\vartheta_4(z) = \sum_{m=-\infty}^{\infty} (-q)^{m^2}. \tag{2.37}$$

Jacobi theta functions are the elliptic analogs of exponential functions. The theta functions are quasi-doubly periodic, which means that it satisfies the following two conditions of periodicity with two independent periods:

- $f(z + \omega_1) = f(z)$
- $f(z + \omega_2) = f(z)$

The value of $f(z)$ is the same when z is shifted by an integer multiple of either ω_1 or ω_2. Since ω_1 and ω_2 are quasi, it means that they are not necessarily multiples of each other.

2.2.9 The Fourier Transform

The Fourier transform is the mapping that transforms a discrete-time signal from a time domain to a frequency domain, which is represented by a lattice.

Definition 2.2.14. For a function $g : \mathbb{R}^n \to \mathbb{R}^1$, given that g is periodic, its Fourier transform $\hat{g} = \text{FT}\{g\}$ is given by [10]:

$$\hat{g}(\boldsymbol{\lambda}^\star) = \frac{1}{\text{Vol}(\Lambda)} \int_{\mathcal{P}_0} g(\boldsymbol{x}) \cdot e^{-i2\pi\langle \boldsymbol{\lambda}^\star, \boldsymbol{x}\rangle} d\boldsymbol{x}, \ \text{ for } \boldsymbol{\lambda}^\star \in \Lambda^\star, \tag{2.38}$$

where $i = \sqrt{-1}$ and $\mathcal{P}_0$ is the fundamental cell of Λ.

If g is not periodic, then its Fourier transform $\hat{g}(f) = \text{FT}\{g\}$ is defined as:

$$\hat{g}(f) = \int_{\mathbb{R}^n} g(\boldsymbol{x}) \cdot e^{-i2\pi\langle \boldsymbol{f}, \boldsymbol{x}\rangle} d\boldsymbol{x}, \ \text{ for } \boldsymbol{f} \in \mathbb{R}^n. \tag{2.39}$$

To find an algorithm that maximizes $\rho(\boldsymbol{y}/\boldsymbol{\lambda})$ in an efficient way, we need the Fourier transform and the Jacobi Theta functions. If $\rho(\boldsymbol{y}/\boldsymbol{\lambda})$ is constant or flat, there will be a high probability of error, which is why we want to maximize it.

Since $f_{\sigma,\Lambda}$ is periodic on Λ, its Fourier transform is defined on the dual lattice $\Lambda^{\star}$. $f_{\sigma,\Lambda}(\boldsymbol{y})$ is the n-variable Gaussian measure over a lattice. The Fourier expansion of $f_{\sigma,\Lambda}(\boldsymbol{y})$ is:

$$f_{\sigma,\Lambda}(\boldsymbol{y}) = \frac{1}{\mathrm{Vol}(\Lambda)} \sum_{\boldsymbol{q}^{\star}\in\Lambda^{\star}} e^{-2\pi^2\sigma^2\|\boldsymbol{q}^{\star}\|^2} e^{2\pi i\langle \boldsymbol{q}^{\star},\boldsymbol{y}\rangle}, \text{ for } \boldsymbol{y}\in\mathbb{R}^n. \tag{2.40}$$

We can then use the Fourier transform to express the Gaussian measure and the Jacobi Theta function.

2.3 Information Theory and Coding Preliminaries

2.3.1 Entropy and mutual information

We will now give the definition for the entropy and mutual information as given by [10, Appendix. A.1].

Definition 2.3.1. The entropy of a discrete random variable X is given by

$$\mathsf{H}(X) = -\sum_{x} p(x)\log_2 p(x), \tag{2.41}$$

where $p(x)$ denotes the probability distribution for X. H is a non-negative and upper bounded by the logarithm of the size of the alphabet $\mathcal{A}$:

$$0 \le \mathsf{H}(X) \le \log|\mathcal{A}|. \tag{2.42}$$

Definition 2.3.2. The conditional entropy:

$$\mathsf{H}(X|Y) = \sum_{y} p(y)\cdot \mathsf{H}(X|Y=y) = \sum_{x,y} p(x,y)\log p(x|y), \tag{2.43}$$

where $p(x,y)$ represents the joint probability distribution between X and Y.

Definition 2.3.3. The joint entropy satisfies the chain rule given by:

$$\mathsf{H}(X|Y) = \mathsf{H}(X) + \mathsf{H}(Y|X) = \mathsf{H}(Y) + \mathsf{H}(X|Y). \tag{2.44}$$

Definition 2.3.4. The mutual information between two random variables is the reduction in the entropy of one of them when the other becomes available:

$$\begin{aligned}\mathsf{I}(X;Y) &= \mathsf{H}(X) - \mathsf{H}(X|Y) && (2.45)\\ &= \mathsf{H}(Y) - \mathsf{H}(Y|X) && (2.46)\\ &= \mathsf{H}(X) + \mathsf{H}(Y) - \mathsf{H}(X,Y). && (2.47)\end{aligned}$$

2.3.2 Channel Coding

In channel coding, one transmits a noisy channel or stores the information on a storage device. The goal is to add redundancy to the transmitted signal, making it distinguishable from the noise. Coding over a channel consists of two stages which are the error-correcting coding stage and the modulation stage. In the error-correcting coding stage, redundancy is added in the discrete alphabet domain, whereas in the modulation stage, the codeword is mapped into the vector $\boldsymbol{x}$. The constellation is the set of all possible input vectors $\boldsymbol{x}$. A lattice constellation is a shortened version of an n-dimensional lattice. The Shannon capacity is defined as the maximum amount of information sent over a channel. This means that the higher the Signal-to-Noise Ratio (SNR) and more channel bandwidth, the higher the possible data rate.
We will now give a definition of Shannon's capacity as given in [10]:

Definition 2.3.5. The Shannon capacity is defined as the maximum amount of information sent over a channel per channel use:

$$\mathsf{C} = \frac{1}{n} \max \mathrm{I}(X; X + Z), \tag{2.48}$$

where the maximization is over all valid random channel inputs X.

An Additive White Gaussian Noise (AWGN) channel mimics the effect of random processes that can occur in a channel. It adds white Gaussian noise to the signal which passes through the channel. It represents the effects of noise on a signal during transmission. It is added to the original signal and has a flat power spectral density across all frequencies, meaning it has the same power at all frequencies. It follows a normal distribution, also referred to as a Gaussian distribution. The amount of noise added to the signal depends on the SNR, which is the ratio of the original signal's power to the noise's power.

2.3.3 Wiretap Channel

The wiretap channel is a setting where one aims to provide information-theoretic privacy of communicated data based solely on the assumption that the channel from sender to the adversary is "noisier" than the channel from sender to receiver [18]. Say we are given a sender called Alice, a receiver called Bob, and an eavesdropper or wiretapper called Eve. The goal of Eve is to listen to the channel and recover the message sent between Alice and Bob. The goal of Alice and Bob is to communicate over a channel without being intercepted or listened to by a third party, namely Eve, meaning that Alice can send a message, and Bob will successfully recover this message at his end. Ideally, Eve is further away from Alice than Bob is, so the interference of the channel will not affect Bob's received message as much as it will affect Eve. The goal in such a channel is, therefore, to minimize the decoding error of Bob and maximize the decoding error of Eve. This way, we can ensure that Bob will recover the message without Eve being able to know what the message is.

2.3.4 Network Coding

Network coding is used to improve the efficiency and performance of data transmission by allowing nodes to combine and manipulate packets to increase the amount of information transmitted and reduce the amount of redundant data. Each node in the network can encode packets it receives before transmitting them to the next node. This technique lets network coding achieve higher throughput and lower response time than similar techniques. The complexity of network coding has to do with the encoding and decoding of packets and the need for synchronization between nodes. In a standard packet-switching network, nodes act as routers and wish to find the best route under the current conditions. A physical layer network coding strategy aims to exploit the linear combination taken by the channel as part of an end-to-end network code. By using appropriately chosen lattice codes, it is possible for each receiver to decode a linear combination of the codewords directly [1].

2.3.5 Physical Layer Network Coding

PNC improves the network throughput and efficiency. PNC combines and processes signals at the physical layer of the communication system, whereas usually, it is done in higher network layers. PNC lets two wireless nodes simultaneously transmit data to a third node by allowing overlapping of signals transmitted by the two nodes in the air. The two signals are superimposed and transmitted as a single signal by the two nodes. The third node receives the superimposed signal and separates the two original signals. By using PNC, you can get reduced latency and improved network throughput. The decoding part of the PNC method is complex, which imposes a challenge. One method to use for PNC is the CF method, which has been developed as a candidate for realizing PNC [19]. We will explain more about PNC in Section 3.3.1.

2.3.6 Nested-lattice-based Physical Layer Network Coding

LNC, as described in [20], is a type of CF relaying strategy. It is an information transmission scheme in Gaussian relay networks. LNC exploits the property that integer linear combinations of lattice points are again lattice points. Relays in LNC attempt to decode their received signals into integer linear combinations of codewords, which can be forwarded. The transmitted information can then be recovered by solving a linear system.

2.3.7 Lattice Network Coding

Let $\mathcal{T}$ be a PID. A Lattice Network Coding (LNC) scheme is a $\mathcal{T}$-linear PNC scheme based on a finite lattice quotient, where each transmitter sends an information-embedding coset through a coset representative, and each receiver recovers one or more $\mathcal{T}$-linear combinations for the transmitted coset representatives. The destination decodes all information-embedding cosets from the transmitters.
We will now define an LNC scheme as done in [8]:

Definition 2.3.6. An LNC scheme is a lattice-partition-based approach to a PNC which generalizes Nazer and Gastpars' CF approach. Here, $\mathcal{E}$ denotes the encoder. φ denotes the homomorphishm $\Lambda \to R_\pi^k$, such that Λ is the kernel of φ. $\bar{\varphi}$ denotes the injective map $R_\pi^k \to \Lambda$ that the transmitter l sends [8, Eq. 1]:

$$\boldsymbol{x}_l = \mathcal{E}(\boldsymbol{w}_l) = \bar{\varphi}(\boldsymbol{w}_l) + \boldsymbol{d}_l - Q_\Lambda(\bar{\varphi})(\boldsymbol{w}_l) + \boldsymbol{d}_l. \tag{2.49}$$

$Q_\Lambda\colon \mathbb{C}^n \to \Lambda'$ is a lattice quantizer for the sublattice Λ and $\boldsymbol{d}_l$ is a dither uniformly distributed over the fundamental region $\mathcal{P}(\Lambda)$. $\boldsymbol{d}_l$ is known to transmitter l and to the receiver. The transmitted signal $\boldsymbol{x}_l \in \mathcal{P}(\Lambda)$. The average power of the transmitted signal $\boldsymbol{x}_l$ is determined by the second moment of the fundamental region $\mathcal{P}(\Lambda)$. The receiver then computes:

$$\hat{\boldsymbol{u}} = \mathfrak{D}(\boldsymbol{y}|\boldsymbol{h}, \bar{\boldsymbol{a}}) = \varphi\Bigg(\mathfrak{D}_\Lambda\Bigg(\alpha\boldsymbol{y} - \sum_{l=1}^{L} \boldsymbol{a}_l\boldsymbol{d}_l\Bigg)\Bigg), \tag{2.50}$$

where $\alpha \in \mathbb{C}$ and $\boldsymbol{a} = (a_1, ..., a_L) \in R^L$ are receiver parameters. The linear combination $\boldsymbol{u}$ is computed correctly if and only if the effective noise $\boldsymbol{n}$ satisfies $\mathfrak{D}_\Lambda(n) \in \Lambda$ The connection between the decoder $\mathfrak{D}(\cdot|\boldsymbol{h}, \boldsymbol{a})$ for LNC and the decoder $\mathfrak{D}(\cdot|\boldsymbol{h}, \bar{\boldsymbol{a}})$ for PNC is that for any receiver parameter $\boldsymbol{a}$, the corresponding coefficient vector $\bar{\boldsymbol{a}}$ is given by $\bar{\boldsymbol{a}} = \sigma(\boldsymbol{a})$.

2.3.8 Maximum Likelihood Decoding

Maximum Likelihood (ML) decoding minimizes the average error probability by maximizing the probability of the noise over all input vectors $\boldsymbol{x}$. For AWGN, this amounts to the lattice point closest to $\boldsymbol{y}$. The ML decoder is optimal but complex. The complexity of the ML decoder when using lattices is that it must take into account the shaping region $\mathcal{V}_0 = \mathcal{V}_0(\Lambda)$. Unless Λ is a simple cubic lattice, ML decoding of a Voronoi constellation is complicated because of the joint structure of the two nested lattices [2].
A simple cubic lattice can be efficiently solved using a lattice reduction algorithm, reducing the search space for the closest lattice point to a sphere centered at the received point. For this we can use, for example, the Fourier transform which breaks down a complex signal into several frequencies. This makes it less comprehensive in terms of analysis and manipulation of the signal. The complexity of such a decoding algorithm is polynomial in the dimension of the lattice. For other lattice structures, for example, with high degree of symmetry, the minimum distance between the lattice points are significantly smaller, and it may take several iterations to find the closest lattice point $\boldsymbol{\lambda}$, which makes the computational complexity significantly higher than for simpler structures.
A problem with ML decoding is that since codewords near the edge of a shaping region have fewer neighbours than the ones closer to the middle, the complexity is usually more minor. It is also more consuming and complex for the inner, since they are more significant, rather than for the outer codewords.

2.3.9 Lattice Decoding

Lattice decoding is less complicated than ML decoding and does not consider the shaping region $\mathcal{V}_0(\Lambda)$ [2]. The output for a lattice decoder is quantized to the fine lattice Λ_1 [10, Ch. 8.1] and the associated coset is found. For AWGN, the quantization is to its nearest lattice point. The lattice decoder can be improved by pre-processing the channel output before decoding. We then get the Euclidean lattice decoder, which considers the estimation of the channel output and the Euclidean distance. The complexity for a lattice decoder is identical for all messages, whereas, for the ML decoder, it varies depending on where in the Voronoi region the point is.

In lattice decoding, a lattice structure represents multiple possible paths through a system. It then gives the path a score for probability or likelihood, to see how likely the path is to be the correct one.

Chapter 3

Review of the proposed papers

In this chapter we have studied papers [2,3,6,7] with the help of the authors in [21], and now we present them in a comparative basis, describing how the authors examine the data-transfer in PNC-based networking, and how it is vulnerable to wiretap attacks. We describe how the CF protocol is used and applied in the papers and how the flatness factor is defined. We also introduce the channel model given for these cases in Section 3.1.1 and we derive the conditional probability formula $p(\boldsymbol{y}/\boldsymbol{\lambda})$ in Section 3.1.4.

3.1 Comparison of the papers

In [21] the authors examine how data transfer in PNC-based networking is vulnerable to wiretapping attacks. It is possible to apply the existing lattice coset coding to obstruct attackers from obtaining information from the data communicated over the network. The paper introduces the major drawbacks of conventional data security schemes based on cryptography. A requirement in these schemes is that the keys must be shared somehow among the communicating parties, and sent over the communication channel. There is always a risk that the keys may be revealed to the adversary. Wiretap channels are vulnerable to eavesdropping attacks because of their broadcast characteristics. A wiretap channel is a communication channel that is vulnerable to eavesdropping by a third party, often referred to as a wiretapper. The wiretappers goal is to listen to or intercept the communication between the legitimate sender and the receiver without them knowing.

Wyner first introduced a degraded wiretap channel where he applied coset coding to protect the message against an attacker whose receiver SNR is less than at the intended destination. Csiszar later generalized the idea where he made no crucial assumptions on the quality of the channels experienced by the legitimate receiver and the attacker. Another technique, called cooperative jamming, uses interference from several network nodes to combat the attacks. This is done by using Gaussian noise or structured codes as the interfering signal or through interference alignment methods. Interference alignment methods is a linear precoding technique that attempts to align interfering signals in time, frequency, or space. In MIMO networks [10], interference alignment uses the spatial dimension multiple antennas offer for alignment.

Another way is to use artificial noise in the MIMO channel. A sender will generate an auxiliary signal from the null space of the receiver's channel and sends it along with its information-bearing signal. The signal will be noise at the attacker's receiver while it is cancelled out at the intended receiver.

3.1.1 Channel model

One relay receives messages from k sources $\boldsymbol{s}_1, ..., \boldsymbol{s}_k$ and transmits a linear combination of these k messages. We will now present the CF strategy as studied by [3].

The received signal at the relay is expressed as [3, Eq. 1]:

$$\boldsymbol{y} = \sum_{j=1}^{k} \boldsymbol{h}_j \boldsymbol{x}_j + \boldsymbol{z}. \tag{3.1}$$

The variable $\boldsymbol{h}_j$ is the channel coefficient between source $\boldsymbol{s}_j$ and the relay. $\boldsymbol{x}_j$ is the vector transmitted by source $\boldsymbol{s}_j$. The variable $\boldsymbol{z}$ is the noise at the relay. The first part $\sum_{j=1}^{k} \boldsymbol{h}_j \boldsymbol{x}_j$ represents the signal transmitted by all k users, while the second part $\boldsymbol{z}$ is the noise at the relay, which is an AWGN channel.

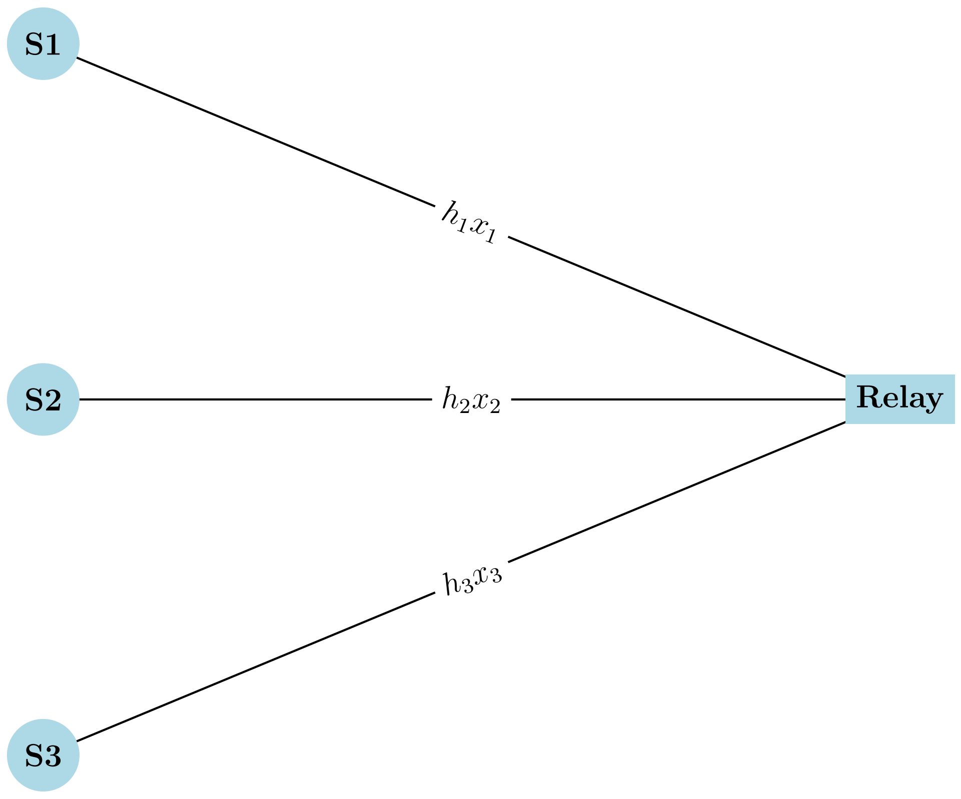


Figure 3.1: A channel model given three sources and one relay.

3.1.2 CF protocol for PNC

Consider a wireless network comprising transmitters, multiple relays, and receivers. Each transmitter consist of an encoder that maps the message vector $\boldsymbol{m}$ from a finite field to a codeword vector $\boldsymbol{x}$ in $\mathbb{R}^n$. The channel coefficients $\boldsymbol{h}$ are only required at the relays, not at the transmitters. Upon receiving the vector $\boldsymbol{y}$, which consists of the transmitted messages, we aim to decode a linear combination. The first component of $\boldsymbol{y}$ represents a linear combination of the transmitted codewords with integer coefficients, while the remaining components represent the effective noise. For this purpose, the codebook must be closed under integer linear combinations to ensure that the sum results in a valid codeword. Lattices satisfy this property.

The effective noise should be independent of the codeword vector $\boldsymbol{x}$, and the codebook must correspond to the message space $\mathbb{F}^n$. To achieve these desired properties, we can select a codebook with a linear structure, such as nested lattices. In this case, the codewords closely resemble points from an n-dimensional lattice partition. The unknown integer coefficients can be determined by maximizing the achievable rate. If the message rate is lower than the achievable rate, the relays can successfully recover an integer linear combination of the transmitted codewords [1].

Imagine a scenario where Alice wants to send a message to Bob, while ensuring that Eve cannot intercept the message. In this setup, there are two separate Additive White Gaussian Noise (AWGN) channels: one between Alice and Bob, and another between Alice and Eve. Alice takes her message vector, denoted as $\boldsymbol{m}$, consisting of k data symbols, and encodes it into a codeword $\boldsymbol{x}$ over the real numbers. This codeword represents a point selected from an n-dimensional lattice $\Lambda_{\mathrm{B}} \subset \mathbb{R}^n$ that is designated for Bob. Within this lattice, there exists a sublattice $\Lambda_{\mathrm{E}} \subset \Lambda_{\mathrm{B}}$. To define a coset of Λ_{E}, we consider a translation of Λ_{E} in the form of $\Lambda_{\mathrm{E}} + \boldsymbol{\lambda}$, where $\boldsymbol{\lambda} \in \mathbb{R}^n$ represents the coset leader. This coset leader is an n-dimensional vector that does not belong to Λ_{E}. Each coset $\Lambda_{\mathrm{E}} + \boldsymbol{\lambda}$ is associated with an information vector $\boldsymbol{m} \in \mathbb{F}^k p$, where $\mathbb{F}^k p$ denotes a finite field. The number of disjoint cosets is equal to p^k, which ensures that each coset is uniquely assigned to an information vector. Consequently, the original lattice Λ_{B} is divided into p^k separate and non-overlapping cosets. This partitioning is presented as:

$$\Lambda_{\mathrm{B}} = U_{m \in \mathbb{F}_p^k} (\Lambda_{\mathrm{E}} + \boldsymbol{\lambda}_m) \tag{3.2}$$

For Gaussian wiretap coding, Alice selects a point $\boldsymbol{x}$ at random from the coset $\Lambda_{\mathrm{E}} + \boldsymbol{\lambda}_m$ and transmits $\boldsymbol{c}$ subsequently. Bob and Eve receive $\boldsymbol{y}_{\mathrm{B}}$ and $\boldsymbol{y}_{\mathrm{E}}$, both in R^n as [21, Eq. 6]:

$$\boldsymbol{y}_{\mathrm{B}} = \boldsymbol{x} + \boldsymbol{z}_{\mathrm{B}} \tag{3.3}$$

$$\boldsymbol{y}_{\mathrm{E}} = \boldsymbol{x} + \boldsymbol{z}_{\mathrm{E}}. \tag{3.4}$$

Let $\boldsymbol{z}_{\mathrm{B}} \sim \mathcal{N}(0, \sigma_{\mathrm{B}^2})$ and $\boldsymbol{z}_{\mathrm{E}} \sim \mathcal{N}(0, \sigma_{\mathrm{E}^2})$ represent the noise terms associated with Bob and Eve's channels, respectively, with noise powers per dimension. Upon receiving $\boldsymbol{y}$, Bob's objective is to decode it by finding the nearest lattice point in Λ_{B} to $\boldsymbol{y}_{\mathrm{B}}$, while Eve aims to do the same. They must then determine the coset to which the decoded lattice point belongs and extract the message m.

The primary objective in wiretap code design is twofold: to maximize the probability of Bob's successful decoding and to minimize the probability of Eve's successful decoding. To ensure Bob's successful decoding, Λ_{B} must be an AWGN-good lattice. On the other hand, for Eve's probability of successful decoding to be low, Λ_{E} must be chosen as a secrecy-good lattice, characterized by a small flatness factor [2]. By assuming a discrete Gaussian distribution for Bob's lattice and selecting a secrecy-good lattice with a small flatness factor for Eve, we can achieve nearly uniformly distributed cosets. This ensures that Bob can achieve a high rate of information transfer since the cosets appear to be evenly distributed. In contrast, Eve will encounter maximum confusion because the cosets seem to be selected almost randomly from her perspective.

Now, let's examine the PNC implemented using the CF relaying strategy in the context of a Gaussian wiretap channel. The wireless network comprises transmitters, relays, and receivers. The relays utilize the CF strategy to transmit data from information source nodes to their respective destinations.
When an eavesdropper intends to gain information from the signal of a certain transmitter, the signal received by the attacker is [21, Eq. 8]:

$$\boldsymbol{y}_{\mathrm{E}} = \boldsymbol{h}_l \boldsymbol{x}_l + \sum_{\ell \in \Lambda \setminus \{l\}} \boldsymbol{h}_\ell \boldsymbol{x}_\ell + \boldsymbol{z}_{\mathrm{E}}, \tag{3.5}$$

where the coefficients $\boldsymbol{h}_\ell$ and $\boldsymbol{h}_l$ refer to the channel coefficients corresponding to the channels between the transmitters and the attacker, and $\boldsymbol{z}_{\mathrm{E}} \sim N(0, \sigma_{\mathrm{E}}^2 I)$. The goal of the attacker is to recover the $\boldsymbol{x}$ [21].

The CF technique involves a relay decoding a linear combination of messages instead of individually recovering them. This prevents attackers from extracting specific users' data, as they can only access a mixture of information from multiple users.

The most significant vulnerability in a CF-based PNC network arises when attackers emulate relays by processing received signals. This poses a substantial security risk to the network.

To address these concerns, lattice Gaussian coding provides a solution for secure information transfer in networks that employ physical layer network coding as their data routing strategy. By incorporating lattice-based PNC, higher data rates can be achieved compared to existing multi-user wireless communication schemes. However, ensuring information security becomes a primary concern in such scenarios.

By incorporating nested lattices that are both AWGN-good and secrecy-good, a multi-user transmission strategy can be established. This approach not only enables higher data rates but also ensures the privacy of each user's information within the network.

Denoting $\boldsymbol{y}_i$ as the received signal at the i^{th} relay Rel_i, we have [21, Eq. 1]:

$$\boldsymbol{y}_i = \sum_{l=1}^{L} \boldsymbol{h}_{i,l} \boldsymbol{x}_l + \boldsymbol{z}_i, \tag{3.6}$$

where $\boldsymbol{h}_{i,l} \in \mathbb{R}$ is a real-valued channel given corresponding to the relay Rel_i and the transmitter $T\boldsymbol{x}_l$.

Properties of the channel:

- $\boldsymbol{x}_l$ is the codeword $\in \mathbb{R}^n$
- $T\boldsymbol{x}_l$ is the l'th transmitter that maps the message $\boldsymbol{m}_l \in \mathbb{F}_p^k$ into $\boldsymbol{x}_l \in \mathbb{R}^n$
- $\boldsymbol{z} \sim N(0, 1)$ denotes the AWGN

We are given the following [21, Eq. 3]:

$$\boldsymbol{y}_i = \sum_{1}^{\Lambda} \boldsymbol{h}_{i,l} \boldsymbol{x}_l + \boldsymbol{z}_l \tag{3.7}$$

$$= \sum_{1}^{\Lambda} \boldsymbol{a}_{i,l} \boldsymbol{x}_l + \sum_{1}^{\Lambda} (\boldsymbol{h}_{i,l} - \boldsymbol{a}_{i,l}) \boldsymbol{x}_l + \boldsymbol{z}_i. \tag{3.8}$$

The first part of the equation is the linear combination of the transmitted codewords with integer coefficients $\boldsymbol{a}_{i,l} \in \mathbb{Z}$, and the second part is the effective noise, which is independent of $\boldsymbol{x}_i$ [2], [6].

The papers [2, 6] are based on Nazer and Gastpar's proposed CF strategy as a PNC scheme. Nazer and Gastpar described a code structure based on nested lattices whose algebraic structure makes the scheme reliable and efficient. The CF strategy exploits interference to obtain higher end-to-end transmission rates between users in a network. The relays are required to decode noiseless linear equations of the transmitted messages using the noisy linear equations provided by the channel. In [6], Osmane and Belfiore consider implementing their scheme for real Gaussian channels and one-dimensional lattices. They relate the maximization of the transmission rate to the lattice SVP. They show that the ML criterion can be implemented using an inhomogeneous Diophantine approximation algorithm, as we will explore in the next section.

3.1.3 CF Protocol for ML decoding

The paper [2] focuses on ML decoding. The system model and assumptions are common for all four papers [2, 3, 6, 7], which are the main focus of the thesis. They consider one relay receiving messages from k sources $\boldsymbol{s}_1, ..., \boldsymbol{s}_k$ and transmitting a linear combination of these k messages. The relay observes a noisy linear combination of the transmitted signals through the channel. Received signal at the relay is expressed as Formula 3.6. $\boldsymbol{h}_j$ is the channel coefficient between source $\boldsymbol{s}_j$ and the relay. $\boldsymbol{x}_j$ is the vector transmitted by source $\boldsymbol{s}_j$. The relay searches for the integer coefficient vector $\boldsymbol{a} = [a_1\ a_2 \cdots a_k]^\intercal$ that maximizes the transmission rate. It then decodes a noiseless linear combination of the transmitted vectors [2, Eq. 2]:

$$\boldsymbol{x}_R = \sum_{j=1}^{k} \boldsymbol{a}_j \boldsymbol{x}_j \tag{3.9}$$

and retransmits it to the destination or another relay. The channel is complex-valued with complex inputs and output. The channel coefficients $\boldsymbol{h}_j$ are complex, circular, i.i.d. Gaussian. The fact that the channel coefficients are complex means that they have both magnitude and phase. The circular part means that the phase of the channel coefficients is uniformly distributed between 0 and 2π radians. This assumption is made to simplify the analysis and to provide a better approximation for the practical aspect. The channel coefficients are also i.i.d. Gaussian, which means the real and imaginary parts of the channel coefficients are independent and identically distributed random variables with a Gaussian distribution, so $\boldsymbol{h}_i \sim \mathcal{N}(0, 1)$. $\boldsymbol{z}$ is Gaussian, zero mean, with variance $\sigma^2 = 1(\boldsymbol{z} \sim \mathcal{N}(0, 1))$. Let $\boldsymbol{h} = [h_1\ h_2 \cdots h_k]^\intercal$ denote the vector of channel coefficients. The source vectors $\boldsymbol{x}_j$ are taken from a lattice code. The sources have no channel-side information (CSI). The CSI is only available at the relay [2].

For the CF strategy, the paper uses the computation rate R_{comp} given by [1] to find the vector $\boldsymbol{a}$ maximizing the computation rate. This is found by applying the SVP in a lattice. The SVP in a lattice is the problem of finding the shortest non-zero vector $\boldsymbol{v}$ in a lattice such that the length $\|\boldsymbol{v}\|$ is minimized. the achievable computation rate relays can recover any set of linear equations with coefficient vector

$\boldsymbol{a}$ as long as the message rates are less than the computation rate [2, Eq. 3]:

$$\mathsf{R}_{\text{comp}}(\boldsymbol{h}, \boldsymbol{a}) = \log\left(\left(|\boldsymbol{a}|^2 - \frac{SNR|\boldsymbol{h}^\dagger \boldsymbol{a}|^2}{1 + SNR\|\boldsymbol{h}\|^2}\right)^{-1}\right). \tag{3.10}$$

The rate is achievable by scaling the received signal by the MMSE coefficient, which is the Minimum Mean Square Error (MMSE) coefficient. The MMSE coefficient minimizes the mean square error between the estimated and accurate signals. We want to find the coefficient vector with the highest computation rate. From [2, Theorem 1]:

Theorem 3.1.1. For a given $\boldsymbol{h} \in \mathbb{C}^N$, $\mathsf{R}_{\text{comp}}(\boldsymbol{h}, \boldsymbol{a})$ is maximized by choosing $\boldsymbol{a} \in \mathbb{Z}[i]$ as

$$\boldsymbol{a} = \arg\min_{\boldsymbol{a} \neq 0}(\boldsymbol{a}^\dagger \mathsf{G} \boldsymbol{a}) \tag{3.11}$$

where,

$$\mathsf{G} = \mathsf{I} - \frac{SNR}{1 + SNR\|\boldsymbol{h}\|^2}\mathsf{H}. \tag{3.12}$$

$\boldsymbol{H} = [H_{ij}]$, $H_{ij} = h_i h_j^\star$, $1 \leq i, j \leq N$.

We will now present the proof as found in [6]:

Proof. Maximizing $\mathsf{R}_{\text{comp}}(\boldsymbol{h}, \boldsymbol{a})$ is equivalent to the following minimization

$$\min_{\boldsymbol{a} \neq \boldsymbol{0}} \left\{\|\boldsymbol{a}\|^2 + \text{SNR}\|\boldsymbol{h}\|^2\|\boldsymbol{a}\|^2 - \text{SNR}|\boldsymbol{h}^\dagger \boldsymbol{a}|^2\right\}. \tag{3.13}$$

We can write

$$|\boldsymbol{h}^\dagger \boldsymbol{a}|^2 = \sum_{i,j} h_i h_j^\star a_i^\star a_j. \tag{3.14}$$

As $\mathsf{H} = [H_{ij}], H_{ij} = h_i h_j^\star$, $1 \leq i, j \leq N$, it follows that $\sum_{i,j} h_i h_i^\star a_i^\star a_j = \boldsymbol{a}^\dagger \mathsf{H} \boldsymbol{a}$. Using these notations, we can write (3.15) as

$$(1 + \text{SNR}\|\boldsymbol{h}\|^2) \min_{\boldsymbol{a} \neq \boldsymbol{0}} \boldsymbol{a}^\dagger \left[\mathsf{I} - \frac{\text{SNR}}{1 + \text{SNR}\|\boldsymbol{h}\|^2}\mathsf{H}\right]\boldsymbol{a}. \tag{3.15}$$

$\mathsf{I} - \frac{\text{SNR}}{1+\text{SNR}\|\boldsymbol{h}\|^2}\mathsf{H}$ has N strictly positive eigenvalues. It is then positive definite. Now, the problem is reduced to the minimization of $\boldsymbol{a}^\dagger \mathsf{G} \boldsymbol{a}$. □

Searching for the vector that minimizes $\boldsymbol{a}$ is equivalent to the SVP for the lattice Λ whose Gram matrix is G. The proof for this can be found in [2].

3.1.4 Deriving the conditional probability formula from the paper

The conditional probability formula $p(\boldsymbol{y}/\boldsymbol{\lambda})$ is maximized in the ML decoder by [2, Eq. 10-11].

$$p(\boldsymbol{y}/\boldsymbol{\lambda}) = \sum_{x_1,...,x_k,\ \sum_{j=1}^{k} a_j x_j = \boldsymbol{\lambda}} p(\boldsymbol{y}/x_1, ...x_k)p(x_1, ..., x_k) \tag{3.16}$$

$$p(\boldsymbol{y}/x_1, ..., x_k) \propto \exp\left[-\frac{\|\boldsymbol{y} - \sum_{j=1}^{k} \boldsymbol{h}_j \boldsymbol{x}_j\|^2}{2\sigma^2}\right]. \tag{3.17}$$

$\propto$ denotes the proportionality between the two expressions. Further, we have:

$$= \sum_{t \in \mathbb{Z}[i]^{(k-1)n}} \exp\left[-\frac{\|\boldsymbol{y} - \mathsf{C}\boldsymbol{t} - \mathsf{D}\mathsf{B}^{-1}\boldsymbol{\lambda}\|^2}{2\sigma^2}\right] \tag{3.18}$$

$$\mathsf{C} = \sum_{j=1}^{k} \frac{\boldsymbol{h}_j}{\boldsymbol{a}_j} \mathsf{M}_j \mathsf{U}_j \tag{3.19}$$

$$\mathsf{D} = \left(\sum_{j=1}^{k} \frac{\boldsymbol{h}_j}{\boldsymbol{a}_j} \mathsf{M}_j \mathsf{V}_j\right) \mathsf{B}^{-1}\boldsymbol{\lambda} \tag{3.20}$$

$$\omega(\boldsymbol{\lambda}) = \boldsymbol{y} - \mathsf{D} \tag{3.21}$$

$$g(\boldsymbol{\lambda}) = \sum_{q \in \Lambda} \exp\left[-\frac{\|\omega - q\|^2}{2\sigma^2}\right]. \tag{3.22}$$

For some values of the parameters, $g(\boldsymbol{\lambda})$ can be flat, which prevents it from maximizing.

3.2 The Flatness Factor

The flatness factor is an essential parameter for a lattice Λ when it comes to decoding, and more specifically, the CF scheme, using ML decoding. The flatness factor depends on the theta series of a lattice Λ. The goal of finding the ML decoding solution is to minimize the flatness factor [2].

The flatness factor provides a measure of how curved a lattice is at a given point. The measure is related to the minimum distance between lattice points, which is important mainly for pure communication of lattice-based crypto schemes. The larger the flatness factor, the more flat the lattice is and the smaller the minimum distance between lattice points. A lattice with a large flatness factor is more vulnerable to attacks that exploit the shortness of lattice vectors, such as the LLL algorithm. This is why analyzing the flatness factor of a lattice is important in general security schemes.

The advantage of the flatness factor is that it gives precise characterization by the theta series, and it can handle both large and small values, whereas, for example, the smoothing parameter only deals with small values [5].

3.2.1 The Flatness Factor Definition

The flatness factor is defined by [2, Def. 3].

Definition 3.2.1. The flatness factor ε_Λ of the lattice Λ is defined as the ratio

$$\varepsilon_\Lambda \triangleq \frac{E_{\boldsymbol{y}}(\phi_\Lambda(\boldsymbol{y}))}{\max_{\boldsymbol{y}\in\mathbb{R}} \phi_\Lambda(\boldsymbol{y})}, \tag{3.23}$$

where the n-variable function $\phi_\Lambda(\boldsymbol{y})$ is defined as:

$$\phi_\Lambda(\boldsymbol{y}) = \sum_{\boldsymbol{x}\in\Lambda} e^{-\frac{\|\boldsymbol{y}-\boldsymbol{x}\|^2}{2\sigma^2}}. \tag{3.24}$$

The average, which is given by [2, Eq. 15]:

$$E_{\boldsymbol{y}}(\phi_\Lambda(\boldsymbol{y})), \tag{3.25}$$

is the average value of the n-variable function ϕ_Λ. The other function is the maximum of the n-variable function ϕ_Λ given by [2, Eq.16]:

$$\max_{\boldsymbol{y}\in\mathbb{R}} \phi_\Lambda(\boldsymbol{y}) = \Theta_\Lambda\left(e^{\frac{1}{2\sigma^2}}\right). \tag{3.26}$$

Equation 2.34 is the theta series of the lattice Λ, and σ^2 is the variance. The proofs for these can be seen in [2]. The flatness factor tells you how evenly spaced the lattice points $\boldsymbol{\lambda}$ are in the lattice Λ. The smaller the flatness factor, the more evenly spaced the points $\boldsymbol{\lambda} \in \Lambda$ are. The definition for the Flatness Factor as given by Equation 3.23 is related to the flatness factor given in [3]. The authors in [3] has changed the definition for the flatness factor for it to be more consistent with the problem of physical-layer security. In [5], the author defines the flatness factor sightly different, where they take into account the minimum variation of $f_{\sigma,\Lambda}(\boldsymbol{x})$, whereas in the definition from the author of [2], they only consider the maximum variation of $f_{\sigma,\Lambda}(\boldsymbol{x})$.

3.2.2 The Smoothing Parameter

The Smoothing Parameter of a lattice Λ is the smallest amount of Gaussian noise that smooths out the discrete structure of Λ up to the number of errors. From [4], we give the definition for the smoothing parameter of a lattice:

Definition 3.2.2. For a lattice Λ and real $\varepsilon > 0$, the smoothing parameter $\eta_\varepsilon(\Lambda)$ is the smallest $s > 0$ such that $Q_{1/s}(\Lambda^\star\backslash\{0\}) \leq \varepsilon$, where Q is the Gaussian function not including the zero-vector.

Remark. $Q_{1/s}(\Lambda^\star\backslash\{0\})$ [4, Sect. 3] is a continous and strictly decreasing function of s, such that

$$\lim_{s\to 0} Q_{1/s}(\Lambda^\star\backslash\{0\}) = \infty \tag{3.27}$$

$$\lim_{s\to\infty} Q_{1/s}(\Lambda^\star\backslash\{0\}) = 0. \tag{3.28}$$

The smoothing parameter was first introduced by the authors in [4]. They explained it as a new lattice parameter with a few fundamental properties. If one were to pick a noise vector from a Gaussian distribution with radius at least as large as the smoothing parameter, and reduce the noise vector modulo the fundamental parallelepiped of the lattice, then the resulting distribution is very close to uniform [4, Sect. 1.1].

3.2.3 The Flatness Factor in LNC

Nazer and Gastpar proposed a CF strategy as a PNC scheme. The code structure is based on nested lattices with an algebraic structure making the scheme reliable and efficient. Niesen and Whiting later revealed an important limitation of the decoder used in Nazer and Gastpar's CF strategy. For this paper, Belfiore and Ling consider maximum-likelihood decoding of the CF strategy as a way to get around the limitation of Nazer and Gastpar limitation. The new decoding algorithm is based on the inhomogeneous Diophantine approximation.

Before this section, we described the PNC concept introduced in [19], which explains how PNC is used to turn the broadcast property of the wireless channel into a capacity-boosting advantage.

In [2], an analysis of the CF strategy was introduced where ML decoding is used. The flatness factor, which we introduced in Definition 3.2.1 in the context of wiretap attacks is now being implemented to measure the quality of ML decoding. In [3], the properties of the flatness factor are studied more deeply and give more knowledge of lattice network coding.

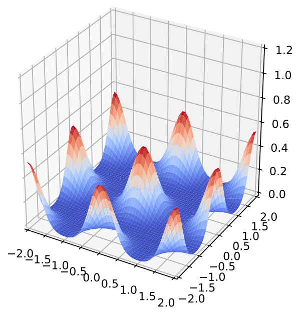
3.2.4 Visualizing different values for the flatness factor

We have now defined the flatness factor. Further, the flatness factor of the Gaussian measure is defined as the maximum ratio $f_{\sigma,\Lambda}(\boldsymbol{y})$ that can deviate from the uniform distribution.

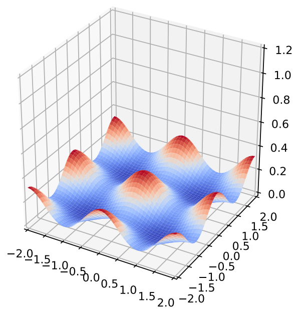
Definition 3.2.3. For a given lattice Λ and a vector $\boldsymbol{y}$, we can consider the n-variable function given in [3]:

$$f_{\sigma,\Lambda}(\boldsymbol{y}) = \frac{1}{(2\pi\sigma)^n} \sum_{\boldsymbol{q}\in\Lambda} e^{\|\boldsymbol{y}-\boldsymbol{q}\|^2/2\sigma^2}. \tag{3.29}$$

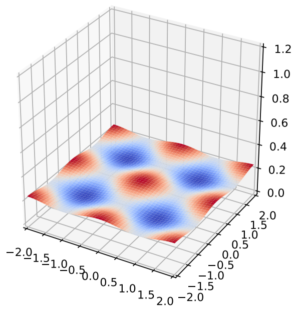
The function represents a sum of Gaussian functions centred at each point $\boldsymbol{q}$ in the lattice Λ. The Gaussian function at each point $\boldsymbol{q}$ is weighted by the distance between $\boldsymbol{y}$ and $\boldsymbol{q}$. The parameter σ decides how far apart the Gaussian functions are. Larger values of σ result in smoother and more spread-out functions, whereas smaller values of σ give sharper functions. We will show this in the following figures reproduced from [3, Fig. 1].



(a) $\sigma^2 = 0.3$



(b) $\sigma^2 = 0.5$



(c) $\sigma^2 = 0.8$

Figure 3.2: $f_{\sigma,\Lambda}(\boldsymbol{y})$ for $\Lambda = 2\mathbb{Z}^2$, $\sigma^2 = 0.3,\ 0.5$, and 0.8.

We notice that when σ^2 is lower, the waves in the graphs becomes higher and steeper, which is what we want for the flatness factor. We want to avoid that the n-variable function $f_{\sigma,\Lambda}(\boldsymbol{y})$ becomes flat for the value of Λ. We know that the flatness factor is the factor in which the maximum ratio $f_{\sigma,\Lambda}(\boldsymbol{y})$ can deviate from the uniform distribution $\epsilon_\Lambda(\sigma)$, and the flatness factor can handle both small and large values compared to the smoothing parameter which only deals with small values. When we have the flatness factor of Λ as large values, we ensure that the maximum likelihood metric is as different as possible from the other likelihood metrics of the point of Λ. For the decoding of the CF strategy to be good, we need a large theta series. The reason for this is that the theta series, potentially making the structure more complex, also provides more degrees of freedom and improve the performance of the CF. It may require more computational resource in order to get the desired outcome. So, depending on what is important, a smaller theta series gives a less complex structure of the lattice, and can be more efficient, while a larger theta series will give better performance, but be less efficient [7].

This is a lot of the problem when it comes to lattices. Having bigger lattice structures, leads to better performance, but in a lot of the cases it ends up being inefficient, and therefore we need to come up with strategies being just as good, but efficient enough for us to get the right trade-off out of it.

3.3 The CF method

Interference Alignment

Interference alignment [11] is a linear precoding technique that attempts to align interfering signals in time, frequency, or space. In Multiple-input Multiple-output (MIMO) networks, interference alignment uses the spatial dimension offered by multiple antennas for alignment.

$$\begin{aligned}\boldsymbol{y}_i &= \sum_{l=1}^{\mathsf{L}} \boldsymbol{a}_{i,l}\boldsymbol{x}_l + \sum_{l=1}^{\mathsf{L}} (\boldsymbol{h}_{i,l} - \boldsymbol{a}_{i,l})\boldsymbol{x}_l + \boldsymbol{z}_i \qquad (3.30)\\ &= x^2 + y^2. \qquad (3.31)\end{aligned}$$

The term $\sum_{l=1}^{\mathsf{L}} \boldsymbol{a}_{i,l}\boldsymbol{x}_l$ corresponds to a linear combination of the transmitted codewords with integer coefficients $\boldsymbol{a}_{i,l} \in \mathbb{Z}$. The term $\sum_{l=1}^{\mathsf{L}}(\boldsymbol{h}_{i,l} - \boldsymbol{a}_{i,l})\boldsymbol{x}_l + \boldsymbol{z}_i$ is the effective noise. The relay computes $\boldsymbol{u}_i$ by decoding the first term in Equation 3.30. If we choose nested lattices, we know that all necessary conditions for the sum to result in a valid codeword will be met. We find the unknown integer coefficients $\boldsymbol{a}$ by maximizing the achievable rate $\mathsf{R}(\boldsymbol{h}, \boldsymbol{a})$, which is given by Equation 3.10. According to [21], any message rate R_l satisfying [21, Eq. 5]

$$\mathsf{R}_l < \mathsf{R}(\boldsymbol{h}, \boldsymbol{a}), \qquad (3.32)$$

is achievable so that the relays can recover an integer linear combination of the transmitted codewords.

3.3.1 How does PNC work?

PNC, described in [19], applies network coding directly within the radio channel at the physical layer. The authors proposes a strategy related to network coding, which deals with reception and modulation of Electro-Magnetic (EM) signals. [19] show that the capacity of a two-way relay channel increases by 100% compared to a scheme without network coding and by 50% compared to network coding.

We will now give the same assumptions of the channel as given by [19]. The authors assume the use of a Quadrature Phase Shift Keying (QPSK) modulation, symbol-level and carrier-phase synchronization and the use of power control, so N_1 and N_3 arrive at N_2 with the same phase and amplitude, where the QPSK data stream can be considered as two Binary Phase-Shift Keying (BPSK) data streams. For the data stream we have one in-phase stream and one quadrature-phase stream. The time slot is defined as the time required for the transmission of one fixed-size frame and the transmission and reception at a particular node must occur in different time slots. The passband signal received by N_2 during one symbol period is by given by [19, Eq. 3]:

$$\begin{aligned} r_2(t) &= s_1(t) + s_3(t) && (3.33)\\ &= [a_1\cos(\omega t) + b_1\sin(\omega t)] + [a_3\cos(\omega t) + b_3\sin(\omega t)] && (3.34)\\ &= (a_1 + a_3)\cos(\omega t) + (b_1 b_3)\sin(\omega t). && (3.35) \end{aligned}$$

The symbol period can be either 1 or 3. It refers to the bandpass signal transmitted by N_i, while $\boldsymbol{r}_2(t)$ represents the bandpass signal received by N_2 during a single symbol period. $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ are the QPSK modulated information bits of N_i, and ω denotes the carrier frequency. Consequently, N_2 will receive two baseband signals. The first one, called the in-phase signal and denoted as P, corresponds to the input. The second one is the quadrature phase signal, denoted as Q, which corresponds to the output [19, Eq. 4].

$$\begin{aligned} P &= \boldsymbol{a}_1 + \boldsymbol{a}_3 && (3.36)\\ Q &= \boldsymbol{b}_1 + \boldsymbol{b}_3. && (3.37) \end{aligned}$$

A relay node plays a crucial role in facilitating data transfer within a network or between different networks. In the context of this scenario, N_2 functions as a relay node. However, it does not have the capability to extract the individual information transmitted by N_1 and N_3. Instead, it can only access the combined signals P and Q. To ensure that the end-to-end delivery of information is equivalent to GF(2), a modulation/demodulation scheme described in [19] is necessary. This scheme enables the summation of bits from N_1 and N_3 at the physical layer, resulting in the desired information delivery.

When $\boldsymbol{s}_1$ and $\boldsymbol{s}_3$ go through modulation, we see clearly from Table 3.1, that 1 maps to 1 and 0 maps to -1. After the mapping of the vectors, N_2 obtain the information bits [19, Eq. 5]:

$$\begin{aligned} \boldsymbol{s}_2^P &= \boldsymbol{s}_1^P \oplus \boldsymbol{s}_3^P && (3.38)\\ \boldsymbol{s}_2^Q &= \boldsymbol{s}_1^Q \oplus \boldsymbol{s}_3^Q, && (3.39) \end{aligned}$$

which transmits [19, Eq. 6]:

$$\boldsymbol{s}_2(t) = \boldsymbol{a}_2\cos(\omega t) + \boldsymbol{b}_2\sin(\omega t). \qquad (3.40)$$

From the received signal $\boldsymbol{s}_2(t)$, we can perform QPSK demodulation to obtain $\boldsymbol{s}_2^P$ and $\boldsymbol{s}_2^Q$. The bits successfully extracted from $\boldsymbol{s}_2^P$ and $\boldsymbol{s}_2^Q$ within a specific time slot

Modulation mapping at N_1 and N_3				Demodulation mapping at N_2		
				Input	Output	
Input		Output			Modulation mapping at N_2	
					Input	Output
$\boldsymbol{s}_1^P$	$\boldsymbol{s}_3^Q$	$\boldsymbol{a}_1$	$\boldsymbol{a}_3$	$\boldsymbol{a}_1 + \boldsymbol{a}_3$	$\boldsymbol{s}_2^P$	$\boldsymbol{a}_2$
1	1	1	1	2	0	-1
0	1	-1	1	0	1	1
1	0	1	-1	0	1	1
0	0	-1	-1	-2	0	-1

Table 3.1: Reproduced from Zhang, Liew and Lam, PNC mapping overview

will be used to create S_2. To achieve this, we can employ PNC-mapping and utilize the network coding operation $S_2 = S_1 \oplus S_3$.

Table 3.1 is reproduced from [19, Table 1] and shows a modulation mapping at N_1 and N_2, and demodulation and modulation mappings at N_3. It shows how the PNC strategy works, so when the inputs $\boldsymbol{s}_1^P$ and $\boldsymbol{s}_3^P$ are modulated, they receive the values for $\boldsymbol{a}_1$ and $\boldsymbol{a}_3$. $\boldsymbol{s}_1$ and $\boldsymbol{s}_3$ are modulated to either 1 or -1, which is the set of $\boldsymbol{a}_j = \{-1, 1\}$, which is the variable representing the BPSK modulated bit of $\boldsymbol{s}_j^P$. If $\boldsymbol{s}_j^P$ gives input value 1, the output $\boldsymbol{a}_j$ is 1 and if $\boldsymbol{s}_j$ is 0, the output $\boldsymbol{a}_j$ is modulated to -1. When we have received the values for $\boldsymbol{a}_1$ and $\boldsymbol{a}_3$, we can demodulate N_2, and the input will be $\boldsymbol{a}_1 + \boldsymbol{a}_3$. The input $\boldsymbol{s}_2^{\mathsf{P}}$ becomes the sum of these two modulated by 2. As explained, the input $\boldsymbol{s}_2^I$ is then modulated to either 1 or -1. This is all part of the demodulation of the input $\boldsymbol{a}_1 + \boldsymbol{a}_3$, where the final output is the last column of Table 3.1 for $\boldsymbol{a}_2$.

We will now compare the Bit Error Rate (BER) performance for standard QPSK modulation, the XOR bit for straightforward network coding and PNC modulation. We suppose that the received signal energy for one bit is unity, which means that the energy of the received signal carrying one bit of information is normalized to one value. This is to simplify the analysis of the performance. The noise is Gaussian white with density $N_0/2$. When transmitting from N_2 to N_1 or N_3, the BER is standard BPSK modulation for all three, which is the Q-function $Q(\sqrt{2/N_0})$. The Q function is the complementary cumulative distribution function of the zero-mean, unit-variance Gaussian random variable. When receiving at N_2, the BER for the XOR bit for straightforward network coding is [19, Sect. 2.D]:

$$2Q(\sqrt{2/N_0})(1 - Q(\sqrt{2/N_0})). \tag{3.41}$$

For the PNC modulation, we need to use the maximum posterior probability criterion. We know that $\boldsymbol{a}_2$ is mapped to either 1 or -1 from the table. IEEE 802.11 requires 100% of the time slots that PNC needs because of the differences in the underlying techniques used for data transmission and network coordination. PNC uses the combination of XOR and superposition coding to transmit data from two sources to a destination, as explained above. In the first time slot, the two sources transmit their signals simultaneously, and in the second, the destination node decodes the signals using the PNC technique described. The IEEE 802.11 is a standard for wireless local area networks (WLAN), and it specifies the physical and data link layer protocols used in wireless communication. IEEE 802.11 uses a technique called carrier sense multiple access with collision avoidance (CSMA/CA) to avoid the wireless nodes colliding. CSMA/CA requires time slots for network coordination. These include Request to Send (RTS), Clear to Send (CTS) and Acknowledgement (ACK) messages. As shown in [19, Ch. 2.D], for one frame exchange, the PNC modulation requires two timeslots, the IEEE 802.11 modulation requires four timeslots and the straightforward network coding requires three, which means that PNC can improve the system by a factor of 100% and 50%, respectively.

3.3.2 Analysis of the paper by Belfiore and method of reproduction

Assumptions for the model [2] is that the channel coefficients are real, i.i.d. Gaussian. $\boldsymbol{z}$ is Gaussian, zero-mean, with variance $\sigma^2 = 1$:

$$\boldsymbol{h}_i \sim \mathcal{N}(0,1), \quad \boldsymbol{z} \sim \mathcal{N}(0,\sigma^2) = \mathcal{N}(0,1)$$

First, we assign values to the variables S_m, ρ, σ_N, $\boldsymbol{h}$ and $\boldsymbol{a}$ as seen in Appendix A.1. Next, we calculate the gcd by using the extended Euclidean algorithm from Appendix A.2. We then determine the solutions of x_1 and x_2 by using [6, Eq. 10]:

Definition 3.3.1. The set of all solutions is obtained as followed

$$\begin{cases} x_1 = \frac{u_1}{g}\lambda + \frac{a_2}{g}k \\ x_2 = \frac{u_2}{g}\lambda - \frac{a_1}{g}k \end{cases} \tag{3.42}$$

$g = a_1 \wedge a_2$ is the gcd of a_1 and a_2, $k \in \mathbb{Z}$.

The solutions for x_1 and x_2 are calculated by Appendix A.3. We have now calculated the gcd, the set (u_1, u_2), γ and β and can then start plotting Figure 3.3 using the following code from Appendix A.3.1.

- Set the value $\boldsymbol{y}$, which is the inner product of vector $\boldsymbol{h}$ and $\boldsymbol{x}$ which are given to be $(x_1, x_2) = (-2, 3)$ from Appendix A.3.
- Define lambda λ with the help of the prorgram from Appendix A.3.1.
- Compute the probability for a given $\boldsymbol{y}$, value of λ, σ_{N}, $\boldsymbol{u}$ and $\boldsymbol{a}$.
- Compute the sum of Gaussians over the given range.
- Give as output the sum returned.
- Set the step size and range for the lambda values, loop over every lambda value and compute the corresponding probability.
- Plot the curve of the figure.

By changing the constellation S_m, the graph changes the amount of maximum point, and by setting S_m down to 30 we see that we get that curve is maximized for one point which is approximately $\lambda = -7$, which we can see in Figure 3.3. When we set $S_m = 90$, we get that λ is maximized for several points.
The figure deviates from [2, Fig. 1], such that for the values λ is maximized are lower for our figure. In our figure, the highest point for $p(\boldsymbol{y}/\boldsymbol{\lambda})$ is right above 14, while for [2, Fig. 1], the maximum point is at approximately 23 for $p(\boldsymbol{y}/\boldsymbol{\lambda})$. The difference between Figure 3.3 and Figure 3.4 is the constellation size S_m.

The figures below are reproduced from the same function from [2, Fig. 1] implemented by the code from Appendix A but for different S_m and ρ.

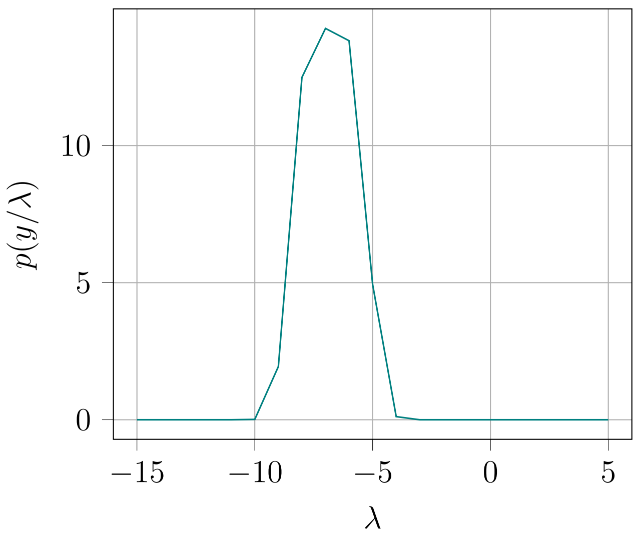


Figure 3.3: $S_m = 30,\ \rho = 40$

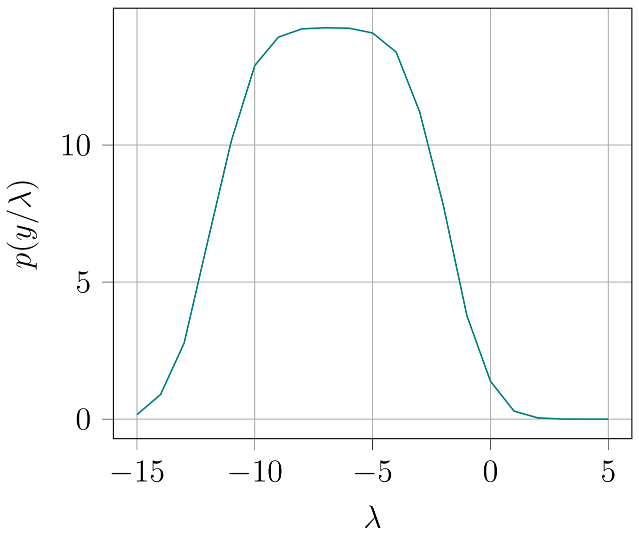


Figure 3.4: $S_m = 90,\ \rho = 40$

3.3.3 Constellation size

The diversity is used to mitigate the effects of a signal fading when travelling through a wireless channel. This involves transmitting multiple copies of the same signal through multiple paths and combining the signal again at the receiver to improve the signal [7]. The diversity is defined as how steep or not the curve for error probability is with respect to the SNR, which can be seen in [6, Fig. 3], where they are using the inhomogeneous Diophantine approximation, which we have mentioned previously. So when the constellation size is small, the symbols used to represent the data have a lower number of bits compared to when the constellation size is bigger. The diversity order is 1. Having a low constellation size means that there are limited possibilities to improve the quality of the signal through the diversity order. The more the constellation size increases, the more the diversity increases. This can lead to more possibilities for improving the quality of the signal through diversity. Since the constellation size of the first figure is small, it is easier to decode the maximum $\rho(\boldsymbol{\lambda})$, since we can see the peek in the figure which corresponds to a unique λ where ρ is maximum unlike for the other figure 3.4, which have a larger constellation size, and diversity order and therefore have several maximum points along the x-axis.

In the second figure 3.4, the $S_m = 90$, which means the constellation size is larger and the diversity order is higher as well. Since ρ depends on the constellation, we get a large set of the couple (x_1, x_2) which needs to be calculated and gone through. The width for ρ, which is the SNR in terms of dB, then becomes larger and therefore becomes flat on the top of the plot, which we see in Figure 3.4 and Figure 3.5. Since we can no longer see the value of λ, the diversity becomes worse. The diversity is less and drops to 1/2, as we can see clearly when putting $S_m = 80$ and $\rho = 50$. The figure has the maximum points for $\lambda = -4$ and $\lambda = -10$.

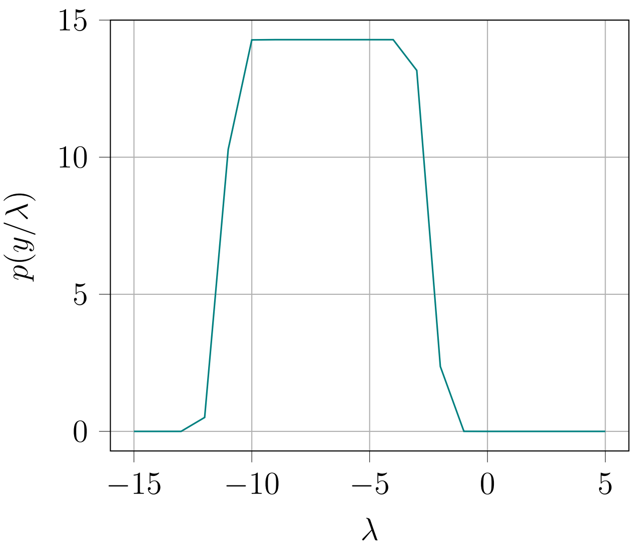


Figure 3.5: $S_m = 80,\ \rho = 50$

Figure taken from [6, Fig. 1]

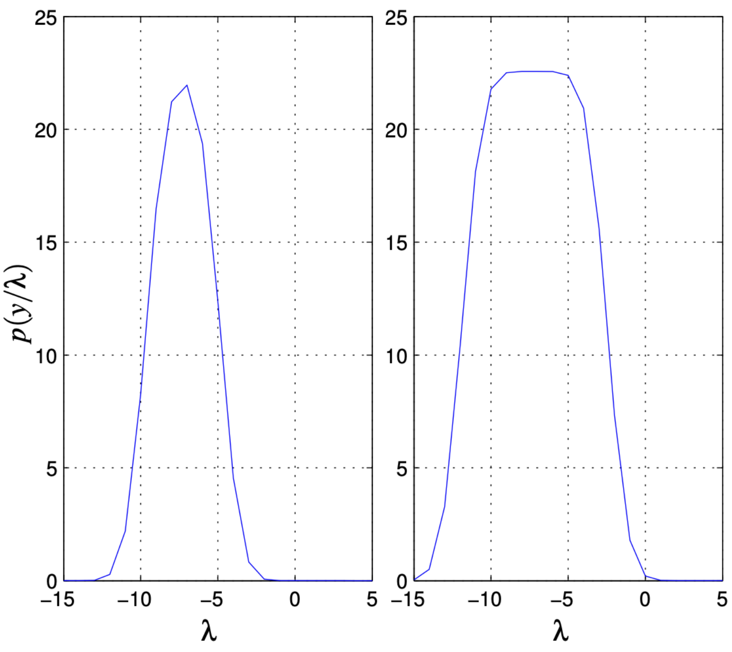


Figure 3.6: $p(\boldsymbol{y}/\boldsymbol{\lambda})$ for $\boldsymbol{h} = [-1.274\ 0.602]^\intercal$, $\boldsymbol{a} = [2\ \ -1]^\intercal$, $SNR = 40dB$, $x_1 = -2$ and $x_2 = 3$. $p(\boldsymbol{y}/\boldsymbol{\lambda})$ is maximized for one value, $\lambda = -7$ in the left sub-figure, while it is maximized for several values of λ in the right one.

We know the y-axis to be $p(\boldsymbol{y}/\boldsymbol{\lambda})$ and the x-axis to be λ. We are given the values

- $\boldsymbol{h} = [-1.274\ 0.602]^\intercal$
- $\boldsymbol{a} = [2\ \ -1]^\intercal$
- $\text{SNR} = 40dB$
- $x_1 = -2$
- $x_2 = 3$

The achievable computation rate relays can recover any set of linear equations with coefficient vector $\boldsymbol{a}$ as long as the message rates are less than the computation rate from Equation 3.10. This rate is achievable by scaling the received signal by the MMSE coefficient. We want to find the coefficient vector with the highest computation rate, as shown in Theorem 3.1.1. Searching for the vector that minimizes $\boldsymbol{a}$ is equivalent to the SVP for the lattice Λ whose Gram matrix is G. Using the values given, we get

$$\exp\left[-\frac{\|1-\sum_{j=1}^{2}(\boldsymbol{h}_j\boldsymbol{x}_j)\|^2}{2\sigma^2}\right] \tag{3.43}$$

$$= \exp\left[-\frac{\|1-((-1.274\cdot -2)+(0.602\cdot 3))\|^2}{2^2}\right]. \tag{3.44}$$

The channel-side information (CSI) is the channel property. It describes how a signal propagates from the transmitter to the receiver and represents the combined

effect. CSI is estimated at the receiver, and it is crucial for achieving reliable communications with high data rates in multi-antenna systems.
The smoothing parameter of a Euclidean lattice Λ is the smallest Gaussian noise that smooths out the discrete structure of Λ.
In [6], the authors relate the maximization of the transmission rate to the lattice SVP. It shows that the ML criterion can be implemented by using the inhomogeneous Diophantine approximation algorithm. The authors in [6] also implement the CF protocol described by Nazer and Gastpar, and they also explain how to obtain the integer coefficients that maximize the rate. They propose a decoding technique based on ML. All the practical aspects are demonstrated for one-dimensional real constellations. By using [2,6,7], we will try to break down the problems into simpler problems and explain what is done in the papers in simpler terms.

3.3.4 Understanding the methods

As seen in Figure 3.1, the model used considers one relay receiving messages from two sources $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ and transmitting a linear combination of the two messages. The received signal at the relay is expressed as Equation 3.1, which is used in the extended Euclidean algorithm to calculate the x's and y's for the plot. The relay will search for the integer coefficient vector $\boldsymbol{a}$ that maximizes the transmission rate. It will then decode the noiseless linear combination of the transmitted signals and retransmit them to the destination or another relay. The channel coefficients h_1 and h_2 are real, i.i.d. Gaussian $h_1 \sim \mathcal{N}(0,1)$. $\boldsymbol{z}$ is Gaussian, zero-mean, with variance $\sigma^2 = 1$. $\boldsymbol{h}$ denotes the vector of channel coefficients $\boldsymbol{h} = [h_1\ h_2]^\intercal$. So, from this, we know that $\sigma^2 = 1$. The source symbols x_i are integers and verify

$$|x_i| \in \mathcal{S} = -s_m, -s_m + 1, ..., s_m. \tag{3.45}$$

The s_m is used later in the plot. The source does not have CSI.

To find the vector $\boldsymbol{a}$ maximizing the computation rate, we use the expression of the computation rate R_{comp} as found in Equation 3.10.
All though they explain in [6] that the rate is achievable by scaling the received signal, the scale is the MMSE-coefficient and is as follows [7, Eq. 10]:

$$\alpha = \alpha_{MMSE} = \frac{\rho \boldsymbol{h}^\dagger \boldsymbol{a}}{1 + \rho ||\boldsymbol{h}||^2}. \tag{3.46}$$

The MMSE coefficient is proved to approach the capacity of linear Gaussian channels in lattice-based strategies. It is unnecessary for the plotting of this figure we are trying to explain because it is a one-dimensional lattice. The MMSE-coefficient is found in [7]. We are also given the vector $\boldsymbol{a}$ in Figure 3.6, so it is not necessary to calculate the achievable computation rate R_{comp} to plot it. From [7, Sect. B.System Model], we are given the formula for the SNR given by ρ:

$$\rho = \frac{\mathsf{E}_{\text{av}}}{\sigma^2}, \tag{3.47}$$

where ρ denotes the SNR at the relay node and E_{av} denotes the average energy per symbol, which is also denoted as S_m, given as one of the variables in Figure 3.6. The last variable σ^2, retrieved from [7] is initially 1. Figure 3.6 uses a one-dimensional lattice, so the transmitted x_i are scalars that belong to real Pulse Amplitude Modulation (PAM) [7] constellations defined by

$$\mathsf{C} = \{x_i \in [-S_m, S_m], S_m \in \mathbb{Z}\}. \tag{3.48}$$

A PAM [7] is a one-dimensional real constellation. It is a set of points representing different discrete amplitude levels for a given signal. For more complex systems, one would use two-dimensional or higher constellations to represent the more complex values.

3.3.5 Recovering the linear equations

The goal of the relay is to decode a linear equation of the transmitted messages and pass it to the destination or another relay. The received signal at the relay is written on the form

$$\boldsymbol{y} = \lambda + \xi_1 x_1 + \xi_2 x_2 + \boldsymbol{z}, \tag{3.49}$$

where λ is an integer , $\xi_i = \boldsymbol{h}_i - \boldsymbol{a}_i$ and $\boldsymbol{z}$ is an additive white noise. To recover the linear equation λ, we have to use a linear Diophantine equation to get

$$\lambda = a_1 x_1 + a_2 x_2. \tag{3.50}$$

A Diophantine equation involves only sums, products and power where all the constants are integers. A Diophantine equation will be on the form

$$a\boldsymbol{x} + b\boldsymbol{y} = \boldsymbol{c}, \tag{3.51}$$

where integers are the only solution of interest.
By using the extended Euclidean algorithm, we get the set of all solutions obtained as follows

$$\begin{cases} x_1 = \frac{u_1}{g}\lambda + \frac{a_2}{g}k \\ x_2 = \frac{u_2}{g}\lambda - \frac{a_1}{g}k. \end{cases} \tag{3.52}$$

$g = a_1 \wedge a_2$ is the gcd of a_1 and a_2, $k \in \mathbb{Z}$.
We want to recover the message $\boldsymbol{m}$. If λ is a multiple of the gcd of a_1 and a_2, then the Diophantine equation has infinite solutions.

Now we choose values for A, U.

$$\mathsf{A} = \begin{bmatrix} 3 & 3 \\ 1 & 0 \end{bmatrix}, \mathsf{U} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}. \tag{3.53}$$

We see from [2] that $\mathsf{H} = \mathsf{A} \cdot \mathsf{U}$, so

$$\mathsf{H} = \begin{bmatrix} 3 & 3 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 1 & 1 \end{bmatrix}. \tag{3.54}$$

From Figure 3.6, we have that $k = 2$, $\boldsymbol{j} = [1,\ 2]$, $\boldsymbol{a} = [2\ \ -1]$ and $\Lambda_j = a_1\Lambda_1 + a_2\Lambda_2$, for $k = 2$.

Since we know from Theorem 3.1.1, we can now define G:

$$\mathsf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathsf{H} = \begin{bmatrix} 3 & 6 \\ 1 & 1 \end{bmatrix}. \tag{3.55}$$

From Figure 3.6, we are also given SNR $= 40dB$ and $\boldsymbol{h} = [-1.274\ 0.602]^{\mathsf{T}}$,

$$\mathsf{G} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{40}{1 + 40\|[-1.274\ 0.602]\|} \cdot \begin{bmatrix} 3 & 6 \\ 1 & 1 \end{bmatrix} \tag{3.56}$$

$$\mathsf{G} = \begin{bmatrix} -1.09 & -4.1839 \\ -0.6973 & 0.3027 \end{bmatrix}. \tag{3.57}$$

Then we can calculate

$$\lambda = \sum_{j=1}^{k=2} \boldsymbol{a}_j \boldsymbol{x}_j = (2 \cdot -2) + (-1 \cdot 3) = -7. \tag{3.58}$$

M is the generator matrix for Λ.

$$\begin{bmatrix} M_1 & M_2 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 \end{bmatrix}^{\mathsf{T}} = \lambda \tag{3.59}$$

$$\mathsf{M} = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{3.60}$$

$$\mathsf{M} \cdot \mathsf{U} = \begin{bmatrix} \mathbf{0} & \mathsf{B} \end{bmatrix} \tag{3.61}$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathsf{B} \end{bmatrix}, \mathsf{B} = 1 \tag{3.62}$$

Now, from [2, Eq. 8] can be expressed as:

$$\begin{bmatrix} \mathbf{0} & \mathsf{B} \end{bmatrix} \cdot \mathsf{U}^{-1} \cdot \begin{bmatrix} r_1 & r_2 \end{bmatrix}^{\mathsf{T}} = \lambda. \tag{3.63}$$

We input the values from Figure 3.6 to find $[\boldsymbol{r}_1\ \boldsymbol{r}_2]$:

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = -7. \tag{3.64}$$

From $[0\ \mathsf{B}] \cdot \mathsf{U}^{-1}$, we get $[0\ 1]$. To get $\lambda = -7$, we set $[r_1\ r_2]^{\mathsf{T}}$ to be $[1\ -7]^{\mathsf{T}}$. Further we define $\tilde{\boldsymbol{t}} = \boldsymbol{U}^{-1}[r_1\ r_2]$. We see that the vector composed of the last n components of $\tilde{\boldsymbol{t}}$ is equal to $\mathsf{B}^{-1}\lambda$.
Therefore we can write:

$$\tilde{\boldsymbol{t}} = \begin{bmatrix} \boldsymbol{t} & \mathsf{B}^{-1}\lambda \end{bmatrix} \tag{3.65}$$

$$\tilde{\boldsymbol{t}} = \mathsf{U}^{-1} \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 \end{bmatrix}^{\mathsf{T}} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -7 \end{bmatrix} = \begin{bmatrix} 8 \\ -7 \end{bmatrix}. \tag{3.66}$$

Further, we can now find $\boldsymbol{t}$, which is a $(k-1)n$-dimensional vector with components in $\mathbb{Z}[i]$:

$$\tilde{\boldsymbol{t}} = \begin{bmatrix} \boldsymbol{t} & \mathsf{B}^{-1}\lambda \end{bmatrix} = \begin{bmatrix} 8 & -7 \end{bmatrix}, \tag{3.67}$$

so $t = 8$.
We then decompose the unimodular matrix U into blocks, where matrices V_j are square $n \times n$ matrices:

$$\mathsf{U} = \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}. \tag{3.68}$$

We use this information further in the formula for $\boldsymbol{r}_j$

$$\boldsymbol{r}_j = \mathsf{U}_j t + \mathsf{V}_j \mathsf{B}^{-1}\lambda. \tag{3.69}$$

For $j = 1$,

$$\boldsymbol{r}_1 = u_1\boldsymbol{t} + v_1\mathsf{B}^{-1}\lambda \tag{3.70}$$

$$\boldsymbol{r}_1 = 1 \cdot 8 + 1 \cdot 1 \cdot -7 = 1, \tag{3.71}$$

which checks out from what we already have for $\boldsymbol{r}_1$.
For $j = 2$,

$$\boldsymbol{r}_2 = \mathsf{U}_2\boldsymbol{t} + \mathsf{V}_2\mathsf{B}^{-1}\lambda\boldsymbol{r}_2 = 0 \cdot 8 + 1 \cdot 1 \cdot -7 = -7, \tag{3.72}$$

which checks out for what we already have for $\boldsymbol{r}_2$.
Now we can look at the final equality

$$\boldsymbol{x}_j = \frac{1}{\boldsymbol{a}_j}(\mathsf{M}_j\mathsf{U}_j\boldsymbol{t} + \mathsf{M}_j\mathsf{V}_j\mathsf{B}^{-1}\lambda), \tag{3.73}$$

where $\boldsymbol{t}$ is any vector in $\mathbb{Z}[i]^{(k-1)n}$.
With the same numbers, we get:

$$x_1 = \frac{1}{2}(0 \cdot 1 \cdot 8 + 0 \cdot 1 \cdot 1 \cdot -7) = 0 \tag{3.74}$$

$$x_2 = \frac{1}{-1}(1 \cdot 0 \cdot 8 + 1 \cdot 1 \cdot 1 \cdot -7) = 7. \tag{3.75}$$

We now see that we have recovered the original message $\boldsymbol{m} = (x_1, x_2) = (0, 7)$

Chapter 4

Conclusions and future work

While there is a lot of research on the area of using PNC, LNC related to CF, the issue is still that the algorithms remains computationally expensive when the dimension of the lattice grows. Many algorithms work well and is suitable for lower dimension lattices, but the reliability gain does not weigh up for the computational cost of the operations as the dimensions increases in size.

4.1 Conclusion

What we have done in this thesis is to present the key points from the papers in focus, which are mainly [2, 3, 6, 7]. We also want to mention [5], to emphasize the importance of secrecy good lattice and the use of the flatness factor to gain the optimal lattice structure to be secrecy-good. The authors in [2, 3, 6, 7] discuss using the HNF algorithm, LNC and the flatness factor of lattices regarding the CF strategy and PNC. We have shown how the HNF algorithm can be used in theory and practically applied to find better basis for lattices, where the vectors are shorter and more orthogonal, with some coding examples. We have explained how a lattice's flatness factor can affect the codings security, and the importance of minimizing the flatness factor to be able to recover the message sent. We showed three figures with different notions of constellation size to show how much easier it can be to find a desired lattice point when the diversity order of a constellation size is one versus when the constellation size increases, the diversity order halves and there is no longer one point maximizing the graph. The author in [2] talks about how the metric of the ML decoder needs to be different from the likelihood metric for the intended receiver to recover the correct message. We have also seen that the flatness factor needs to be minimized for the ML decoding to work correctly and for the decoders to recover the correct message. The flatness factor can be defined for different purposes, where the author in [5] proposes a definition more suiting for a secrecy-good lattice.

4.2 Future work

For future work, it would be an interesting approach to work more on the graphs displaying the conditional probability, which are maximized for both one point and one for several depending on the value for SNR, S_m and ρ and also to know exactly why the maximized points in the figures we have tried to reproduce are much lower compared to the figure displayed in the paper by [2].

Another interesting approach is to implement other basis-reduction algorithms to optimize the computational cost and still retain the advantages of using lattices to work on the approaches introduced in the thesis. As mentioned by the authors in [7], it is in fact important to optimize the flatness factor and the error probability function for lattice constructions. Regarding security, the flatness factor makes it harder for an attacker to exploit a lattice structure and it is therefore important to research on the area to find ways to use the flatness factor in efficient ways making it more secure. In terms of efficiency, a lower flatness factor also allows for compact representations of lattice points, meaning the lattice points are more evenly distributed and closer to each other in terms of the Euclidean distance. It can also reduce the computational complexity of the lattice operations, making the implementations of lattices more efficient.

I think it is important to mention a conjecture from the authors of [3], where they state that "Maximizing the ML metric is equivalent to solving a multidimensional inhomogeneous Diophantine approximation problem". All though the conjecture is not proved, it is an important notion due to the fact that most of the decoding techniques presented in the papers introduced are presented for one-dimensional lattices, and it is necessary for future research to solve the issue for higher dimensions as well.

Bibliography

[1] B. Nazer and M. Gastpar, “Compute-and-forward: Harnessing interference through structured codes,” 2009.

[2] J.-C. Belfiore, “Lattice codes for the compute-and-forward protocol: The flatness factor,” 2011.

[3] J.-C. Belfiore and C. Ling, “The flatness factor in lattice network coding: design criterion and decoding algorithm,” 2012.

[4] D. Micciancio and O. Regev, “Worst-case to average-case reductions based on gaussian measures,” in *45th Annual IEEE Symposium on Foundations of Computer Science*, pp. 372–381, 2004.

[5] C. Ling, L. Luzzi, J.-C. Belfiore, and D. Stehlé, “Semnatically secure lattice codes for the gaussian wiretap channel,” 2014.

[6] A. Osmane and J.-C. Belfiore, “The compute-and-forward protocol: Implementation and practical aspects,” 2011.

[7] A. Mejri, G. R.-B. Othman, and J. C. Belfiore, “Lattice decoding for the compute-and-forward protocol,” 2012.

[8] C. Feng, D. Silva, and F. R. Kschischang, “Design criteria for lattice network coding,” 2011.

[9] K. H. Rosen, *Discrete Mathematics and Its Applications.* McGraw Hill, 2011.

[10] R. Zamir, *Lattice Coding for Signals and Networks.* Cambridge, U.K.: cambridge, 2014.

[11] S. I. Costa, F. Oggier, A. Campello, J.-C. Belfiore, and E. Viterbo, *Lattices Applied to Coding for Reliable and Secure Communications.* Cham, Switzerland: springer, 2017.

[12] H. Cohen, *A Course in Computational Algebraic Number Theory.* Germany: Springer-Verlag Berlin Heidelberg, 1996.

[13] K. Shinohara, “Hsnf 0.3.16.”

[14] J. Hoffstein, J. Pipher, and J. H. Silverman, *An introduction to Mathematical Cryptography.* New York, NY, USA: Springer, 2004.

[15] E. W. Weisstein, “Theta series.”

[16] J. Koekoek and R. Koekoek, "The Jacobi inversion formula," 1999.

[17] J. Conway and N. Sloane, *Sphere Packings, Lattices and Groups*. Springer, 1999.

[18] A. D. Wyner, "The wire-tap channel," *The Bell System Technical Journal*, vol. 54, pp. 1355–1387, October 1975.

[19] Z. Shengli, S.-C. Liew, and P. P. K. Lam, "Physical layer network coding," 2007.

[20] C. Feng, D. Silva, and F. R. Kschischang, "An algebraic approach to physical-layer network coding," *IEEE Transactions on Information Theory*, vol. 59, pp. 7576–7596, nov 2013.

[21] V. Forutan and R. F. H. Fischer, "On the security of lattice-based physical-layer network coding against wiretap attacks," 2015.

Appendices

Appendix A

Code for reproducing flatness factor figure

A.1 Defining the variables

```
import numpy as np
# the upper bound of the source symbols
Sm = 70;
# the SNR in terms of dB
rho = 40;
# the variance of the Gaussian noise
sigma_N = np.sqrt((Sm**2)/(10**(rho/10)));
# the channel coefficients
h = np.array([-1.274, 0.602]);
# a maximized
a = np.array([2,-1]);
```

A.2 Extended Euclidean Algorithm

```
# Python program of extended Euclidean Algorithm
# Cf.
    https://www.geeksforgeeks.org/python-program-for-basic-and-extended-euclidean-algori
# Cf. CormenLeisersonRivestStein22_1
# function for extended Euclidean Algorithm
def gcdExtended(a, b):
    # Base Case
    if b == 0 :
        return a,1,0
    gcd,x1,y1 = gcdExtended(b, a%b)
    # Update x and y using recursive results
    x = y1 # //: floor Division
    y = x1 - (a//b) * y1

    if (a<0 and b<0):
        gcd = -gcd;
        x = -x;
        y = -y;
    if (a<0 and b>0):
        gcd = -gcd;
        x = -x;
        y = -y;
    if (a>0 and b<0):
        gcd = -gcd;
        x = -x;
        y = -y;
    u = np.array([x,y]);

    return gcd,u
```

A.3 Determining the gcd

```
# determine the solutions of x1 and x2 by (10)
g, u = gcdExtended(a[0], a[1])
print("gcd(", a[0] , "," , a[1], ") = ", g)
print("(u_1,u_2) = (",u[0], ",", u[1],")")
# Other parameters:
gamma_h = (h[0]*u[0]+h[1]*u[1])/g;
print("gamma =",gamma_h);
beta_h = (h[1]*a[0]-h[0]*a[1])/g;
print("beta =",beta_h);
```

A.3.1 Plot the figure

```
from mpmath import *
mp.dps = 25; mp.pretty = True;
import matplotlib.pyplot as plt
# set the y value
y = np.inner(h, [-2,3]);
# Check OsmaneBelfiore11_1sub: Cf. https://arxiv.org/abs/1107.0300
def p_lambda(y,lambda_val,sigma_N,u,a):
# test of lambda_val
# lambda_val = -14;
    sum_Gaussians = 0;
    # get all the valid boundaries for k
    kBounds_x1 =
        np.sort([(-Sm*g-u[0]*lambda_val)/a[1],(Sm*g-u[0]*lambda_val)/a[1]]);
    kBounds_x2 =
        np.sort([(-Sm*g+u[1]*lambda_val)/a[0],(Sm*g+u[1]*lambda_val)/a[0]]);
    # get the valid values of k for x1 and x2
    k_x1Range=np.arange(np.ceil(kBounds_x1[0]),np.floor(kBounds_x1[1])+1,1);
    k_x2Range=np.arange(np.ceil(kBounds_x2[0]),np.floor(kBounds_x2[1])+1,1);
    kRange = np.intersect1d(k_x1Range,k_x2Range);
    for k in kRange:
        x = [(u[0]*lambda_val+a[1]*k)/g,(u[1]*lambda_val-a[0]*k)/g];
        sum_Gaussians = sum_Gaussians + exp(-(y-np.inner(h,
            x))**2/(2*sigma_N**2))/(sqrt(2*pi)*sigma_N);
    return sum_Gaussians
step_lambda = 1;
lambdaRange = list(np.arange(-15,5+step_lambda,step_lambda));
# the curve in terms of lambda
p_lambda_curve = [];
# compute all the values of lambda
for lambda_val in lambdaRange:
    # evaluate the values of rho_lambda
    p_lambda_curve.append(p_lambda(y,lambda_val,sigma_N,u,a));
plt.figure()
plot_curve = plt.plot(lambdaRange,p_lambda_curve,'b-');
plt.grid(True)
plt.xlabel('lambda')
```

Appendix B

Hermite Style Normal Form Algorithm

B.1 Hermite Normal Style Form Algorithm

```
from hsnf import row_style_hermite_normal_form
import numpy as np

M = np.array([[6, -4, -7],
              [3, 5, 14],
              [-3, 7, 16]])

H, R = row_style_hermite_normal_form(M)

print(f'H = {H}')
```

B.1.1 Row style hermite normal form

```
def row_style_hermite_normal_form(M: NDArrayInt) -> tuple[NDArrayInt,
    NDArrayInt]:
    """
    Calculate row-style Hermite normal form of `M`.
    Return matrices `(H, L)` satisfy ``H = np.dot(L, M)``.

    Parameters
    ----------
    M: array, (m, n)
        Integer matrix

    Returns
    -------
    H: Array, (m, n)
        Hermite normal form of M, upper-triangular integer matrix
    L: array, (m, m)
        unimodular matrix
    """
    zmh = ZmoduleHomomorphishm.with_standard_basis(M)
    return zmh.hermite_normal_form()
```

B.1.2 With standard basis

```
def with_standard_basis(cls, A):
    """
    Create homomorphish with regard A as a matrix representation with
        standard basis

    Parameters
    ----------
    A: array, (m ,n)
        matrix representation of homomorphish: Z^m -> Z^n
    """
    A = np.array(A, dtype=int)
    if A.ndim !=2:
        raise ValueError("matrix representation must be 2s")

    m, n = A.shape
    basis_from = cls._standard_basis(m)
    basis_to = cls._standard_basis(n)

    return cls(A, basis_from, basis_to)
```

B.1.3 Hermite Normal Form

```
def hermite_normal_form(self):
    """
    calculate row-style Hermite normal form

    Returns
    --------
    H: array, (m, n)
        Hermite normal form of M, upper-triangular integer matrix
    L: array, (m, n)
        unimodular matrix s.t. H = np.dot(L, M)
    """
    A = self._A.copy()
    basis_from = self._basis_from.copy()
    basis_to = self._basis_to.copy()

    H, L = self._hnf_row(si=0, sj=0)

    #revert A, basis_from and basis_to
    self._A = A
    self._basis_from = basis_from
    self._basis_to = basis_to
    return H, L
```

B.1.4 ZmoduleHomomorphishm

```
    # Copyright (c) 2019 Kohei Shinohara
# Distributed under the terms of the MIT License.
from __future__ import annotations

import warnings

import numpy as np

from hsnf.utils import NDArrayInt, get_nonzero_min_abs_full,
    get_nonzero_min_abs_row


class ZmoduleHomomorphism:
    """
    homomorphism between Z-modules

    Parameters
    ----------
    A: array, (m, n)
        matrix representation of homomorhism: Z^m -> Z^n
    basis_from: array, (m, )
        basis of Z^m
    basis_to: array, (n, )
        basis of Z^n
    """

    def __init__(self, A, basis_from, basis_to):
        if A.dtype not in [np.int32, np.int64]:
            warnings.warn("Decomposed matrix should be integer.")

        self._A = A
        self._basis_from = basis_from
        self._basis_to = basis_to

    @property
    def num_row(self):
        return self._A.shape[0]

    @property
    def num_column(self):
        return self._A.shape[1]

    def _swap_from(self, axis1, axis2):
        self._basis_from[[axis1, axis2]] = self._basis_from[[axis2, axis1]]
        self._A[[axis1, axis2]] = self._A[[axis2, axis1]]
```

```
def _swap_to(self, axis1, axis2):
    self._basis_to[:, [axis1, axis2]] = self._basis_to[:, [axis2,
       axis1]]
    self._A[:, [axis1, axis2]] = self._A[:, [axis2, axis1]]

def _change_sign_from(self, axis):
    self._basis_from[axis] *= -1
    self._A[axis, :] *= -1

def _change_sign_to(self, axis):
    self._basis_to[:, axis] *= -1
    self._A[:, axis] *= -1

def _add_from(self, axis1, axis2, k):
    """
    add k times axis2 to axis1
    """
    self._basis_from[axis1] += self._basis_from[axis2] * k
    self._A[axis1, :] += self._A[axis2, :] * k

def _add_to(self, axis1, axis2, k):
    """
    add k times axis2 to axis1
    """
    self._basis_to[:, axis1] += self._basis_to[:, axis2] * k
    self._A[:, axis1] += self._A[:, axis2] * k

def _is_lone(self, s):
    """
    check if all s-th row elements column elements become zero
    """
    if np.nonzero(self._A[s, (s + 1) :])[0].size != 0:
        return False
    if np.nonzero(self._A[(s + 1) :, s])[0].size != 0:
        return False
    return True

def _get_nextentry(self, s):
    """
    return entry which is not diviable by A[s, s]
    assume A[s, s] is not zero.
    """
    for i in range(s + 1, self.num_row):
        for j in range(s + 1, self.num_column):
            if self._A[i, j] % self._A[s, s] != 0:
                return i, j
    return None

def _snf(self, s):
    """
```

```
        determine SNF up to the s-th row and column elements
        """
        if s == min(self._A.shape):
            return self._A, self._basis_from, self._basis_to

        # choose a pivot
        row, col = get_nonzero_min_abs_full(self._A, s)
        if col is None:
            # if there does not remain non-zero elements, this procesure
                ends.
            return self._A, self._basis_from, self._basis_to
        self._swap_from(s, row)
        self._swap_to(s, col)

        # eliminate the s-th column entries
        for i in range(s + 1, self.num_row):
            if self._A[i, s] != 0:
                k = self._A[i, s] // self._A[s, s]
                self._add_from(i, s, -k)

        # eliminate the s-th row entries
        for j in range(s + 1, self.num_column):
            if self._A[s, j] != 0:
                k = self._A[s, j] // self._A[s, s]
                self._add_to(j, s, -k)

        # if there does not remain non-zero element in s-th row and column,
            find a next entry
        if self._is_lone(s):
            res = self._get_nextentry(s)
            if res:
                i, j = res
                self._add_from(s, i, 1)
                return self._snf(s)
            elif self._A[s, s] < 0:
                self._change_sign_from(s)
            return self._snf(s + 1)
        else:
            return self._snf(s)

    def smith_normal_form(self):
        """
        calculate Smith normal form

        see the following awesome post for a description of this algorithm:
            http://www.dlfer.xyz/post/2016-10-27-smith-normal-form/

        Returns
        -------
        D: array, (m, n)
        L: array, (m, m)
        R: array, (n, n)
            D = np.dot(L, np.dot(M, R))
            L, R are unimodular.
        """
        A = self._A.copy()
        basis_from = self._basis_from.copy()
        basis_to = self._basis_to.copy()

        D, L, R = self._snf(s=0)

        # revert A, basis_from, and basis_to
        self._A = A
        self._basis_from = basis_from
        self._basis_to = basis_to

        return D, L, R

    def _hnf_row(self, si, sj):
        """
        determine row-style HNF up to the si-th row and the sj-th column
            elements
        """
        if (si == self.num_row) or (sj == self.num_column):
            return self._A, self._basis_from

        # choose a pivot
        row, _ = get_nonzero_min_abs_row(self._A, si, sj)

        if row is None:
            # if there does not remain non-zero elements, go to a next
                column
            return self._hnf_row(si, sj + 1)
        self._swap_from(si, row)

        # eliminate the s-th column entries
        for i in range(si + 1, self.num_row):
            if self._A[i, sj] != 0:
                k = self._A[i, sj] // self._A[si, sj]
                self._add_from(i, si, -k)

        # if there does not remain non-zero element in s-th row, find a
            next entry
        if np.count_nonzero(self._A[(si + 1) :, sj]) == 0:
            if self._A[si, sj] < 0:
                self._change_sign_from(si)

            if self._A[si, sj] != 0:
                for i in range(si):
                    k = self._A[i, sj] // self._A[si, sj]
                    if k != 0:
                        self._add_from(i, si, -k)
            return self._hnf_row(si + 1, sj + 1)
        else:
            return self._hnf_row(si, sj)

    def hermite_normal_form(self):
        """
        calculate row-style Hermite normal form

        Returns
        -------
        H: array, (m, n)
            Hermite normal form of M, upper-triangular integer matrix
        L: array, (m, m)
            unimodular matrix s.t. H = np.dot(L, M)
        """
        A = self._A.copy()
        basis_from = self._basis_from.copy()
        basis_to = self._basis_to.copy()

        H, L = self._hnf_row(si=0, sj=0)

        # revert A, basis_from, and basis_to
        self._A = A
        self._basis_from = basis_from
        self._basis_to = basis_to

        return H, L

    @classmethod
    def _standard_basis(cls, n):
        return np.eye(n, dtype=int)

    @classmethod
    def with_standard_basis(cls, A):
        """
        create homomorhism with regard A as a matrix representation with
            standard basis

        Parameters
        ----------
        A: array, (m, n)
            matrix representation of homomorhism: Z^m -> Z^n
        """
        A = np.array(A, dtype=int)
        if A.ndim != 2:
            raise ValueError("matrix representation must be 2d")

        m, n = A.shape
        basis_from = cls._standard_basis(m)
        basis_to = cls._standard_basis(n)
        return cls(A, basis_from, basis_to)


def smith_normal_form(M: NDArrayInt) -> tuple[NDArrayInt, NDArrayInt,
    NDArrayInt]:
    """
    Calculate Smith normal form of integer matrix `M`.
    Returned matrices `(D, L, R)` satisfy ``D = np.dot(L, np.dot(M, R))``.

    Parameters
    ----------
    M: array, (m, n)
        Integer matrix

    Returns
    -------
    D: array, (m, n)
        Smith normal form of `M`
    L: array, (m, m)
        Unimodular matrix
    R: array, (n, n)
        Unimodular matrix
    """
    zmh = ZmoduleHomomorphism.with_standard_basis(M)
    return zmh.smith_normal_form()


def row_style_hermite_normal_form(M: NDArrayInt) -> tuple[NDArrayInt,
    NDArrayInt]:
    """
    Calculate row-style Hermite normal form of `M`.
    Returned matrices `(H, L)` satisfy ``H = np.dot(L, M)``.

    Parameters
    ----------
    M: array, (m, n)
        Integer matrix

    Returns
    -------
    H: array, (m, n)
        Hermite normal form of M, upper-triangular integer matrix
    L: array, (m, m)
        Unimodular matrix
    """
    zmh = ZmoduleHomomorphism.with_standard_basis(M)
    return zmh.hermite_normal_form()
```

```
def column_style_hermite_normal_form(M: NDArrayInt) -> tuple[NDArrayInt,
    NDArrayInt]:
    """
    Calculate column-style Hermite normal form of ‘M‘
    Returned matrices ‘(H, R)‘ satisfy ‘‘H = np.dot(M, R)‘‘

    Parameters
    ----------
    M: array, (m, n)
        Integer matrix

    Returns
    -------
    H: array, (m, n)
        Hermite normal form of M, lower-triangular integer matrix
    R: array, (n, n)
        Unimodular matrix
    """
    zmh = ZmoduleHomomorphism.with_standard_basis(M.T)
    H_T, R_T = zmh.hermite_normal_form()
    H = H_T.T
    R = R_T.T
    return H, R
```

B.1.5 HNF row

```
def _hnf_row(self, si, sj):
"""
determine row-style HNF up to the si-th row and the sj-th column elements
"""
if (si == self.num_row) or (sj == self.num_column):
    return self._A, self._basis_from

# choose a pivot
row, _ = get_nonzero_min_abs_row(self._A, si, sj)

if row is None:
    # if there does not remain non-zero elements, go to a next column
    return self._hnf_row(si, sj + 1)
self._swap_from(si, row)

# eliminate the s-th column entries
for i in range(si + 1, self.num_row):
    if self._A[i, sj] != 0:
        k = self._A[i, sj] // self._A[si, sj]
        self._add_from(i, si, -k)

# if there does not remain non-zero element in s-th row, find a next entry
if np.count_nonzero(self._A[(si + 1) :, sj]) == 0:
    if self._A[si, sj] < 0:
        self._change_sign_from(si)

    if self._A[si, sj] != 0:
        for i in range(si):
            k = self._A[i, sj] // self._A[si, sj]
            if k != 0:
                self._add_from(i, si, -k)

    return self._hnf_row(si + 1, sj + 1)
else:
    return self._hnf_row(si, sj)
```

B.2 Voronoi partition

```
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.spatial import Voronoi, voronoi_plot_2d

    A = (-1,1)
    B = (2,3)
    C = (0,-2)
    points = (A,B,C)
    voronoi = Voronoi(points)
    figure = voronoi_plot_2d(voronoi, show_vertices=False,
        line_colors='grey', line_width=1, line_alpha=1, point_size=3)

    plt.show()
```