Lattice Codes and Sphere Decoding

Pasi Pyrrö

School of Science

Bachelor's special assignment
Espoo 14.8.2017

Thesis supervisor:

Prof. Camilla Hollanti

Thesis advisors:

Prof. Marcus Greferath

Dr. Oliver W. Gnilke

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE BACHELOR'S
SPECIAL ASSIGNMENT

Author: Pasi Pyrrö		
Title: Lattice Codes and Sphere Decoding		
Date: 14.8.2017	Language: English	Number of pages: 4+28
Degree programme: Mathematics and Systems Analysis		
Supervisor: Prof. Camilla Hollanti		
Advisors: Prof. Marcus Greferath, Dr. Oliver W. Gnilke		

Keywords: sphere decoding, space–time lattice codes, wireless communications

Contents

Symbols and abbreviations

Symbols

$\mathbb{Z}$	Set of integers
$\mathbb{C}$	Field of complex numbers
$\mathbf{x}$	Vector
$\mathbf{X}$	Matrix

Operators

$\|\cdot\|$	Euclidean norm
$\lceil \cdot \rfloor$	Round to nearest integer
$\det(\mathbf{X})$	Determinant of matrix $\mathbf{X}$

Abbreviations

SNR	Signal to noise ratio
CVP	Closest vector problem
MIMO	Multiple input multiple output
PAM	Pulse amplitude modulation
i.i.d.	independent and identically distributed
BLER	Block error rate

1 Introduction

Wireless communication has been a crucial part of modern information technology for a couple of decades now. It is facing a lot of practical everyday problems which motivate the ongoing extensive research on the field. One of these problems is the noise and fading that occurs on wireless channels due to obstacles and radiation from the surroundings. To avoid data loss during transmission via wireless link one has to encode the data to be sent in such a robust way that it can still be recovered at the receiving end even in the presence of noise and fading of reasonable scale.

One way to tackle this problem, and the method this thesis focuses on, is the use of space–time lattice codes and sphere decoding. From a mathematical point of view the process of decoding can then be viewed as a problem of finding the closest lattice point to a given input vector, that is, the possibly noisy vector containing the data we receive from the wireless channel. This problem in its general form is known to be NP-hard, but for communications applications, where the dimensionality and the shape of the lattice are kept reasonable, there exist algorithms, like the sphere decoder, that offer polynomial expected complexity with respect to the size of the codeword constellation [1].

2 Lattices in communications technology

Before we go into communications applications of lattices such as lattice codes, let us start off with the definition of a lattice. Let n and m be positive integers such that $n \leq m$ and $\mathbf{b}_1, ..., \mathbf{b}_n \in \mathbb{R}^m$ be linearly independent vectors. A discrete Abelian subgroup Λ of $\mathbb{R}^m$ is called a *lattice* of dimension n if it is defined as

$$\Lambda = \sum_{i=1}^{n} \mathbb{Z}\mathbf{b}_i = \{a_1\mathbf{b}_1 + \ldots + a_n\mathbf{b}_n \mid a_i \in \mathbb{Z}, 1 \leq i \leq n\} \tag{1}$$

where the set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ is called the basis of the lattice Λ. All points of the lattice can be obtained as a linear combination of the basis vectors $\mathbf{b}_i$ and integer coefficients $\mathbf{a}_i$ as stated in (1). The number of basis vectors n is also called the rank of the lattice and if $n = m$ the lattice is said to have full rank. The points of the lattice Λ form a group under addition which means that if $\mathbf{x} \in \Lambda$ then $-\mathbf{x} \in \Lambda$ and if $\mathbf{x}, \mathbf{y} \in \Lambda$ then $\mathbf{x} \pm \mathbf{y} \in \Lambda$. [2]

There is also a matrix representation for the same lattice

$$\Lambda = \{\mathbf{B}a \mid a \in \mathbb{Z}^n\} \tag{2}$$

where $\mathbf{B} = [\mathbf{b}_1, ..., \mathbf{b}_n]$ is an $m \times n$ matrix called the generator matrix of the lattice Λ and $\mathbf{a}$ is an n-dimensional integer column vector. Note that $\mathbf{B}$ is not uniquely determined by the lattice as in fact there exists infinitely many bases for the same lattice [2]. Also the origin is part of every lattice regardless of the choice of $\mathbf{B}$. If $\mathbf{B}$ is a basis for lattice Λ then $\mathbf{B}'$ is also a basis for the same lattice if the following holds

$$\mathbf{B}' = \mathbf{W}\mathbf{B}, \tag{3}$$

$$\det(\mathbf{W}) = \pm 1 \tag{4}$$

where $\mathbf{W}$ is an $m \times m$ matrix with integer entries [3]. Some bases are however better in some sense than others, especially in communications applications, as one could imagine. Usually reasonable orthogonality and relatively small norm for the basis vectors are desired [3]. A better basis can be obtained via a process called basis reduction, which is illustrated in figure 1. Clearly basis vectors $\mathbf{u}_i$ obtained from the basis reduction are more pleasant to work with than the original vectors $\mathbf{v}_i$ although they both span the same lattice.

What we considered earlier were lattices spanned over real m-dimensional space $\mathbb{R}^m$ but in similar sense we can consider a lattice consisting of complex valued vectors from $\mathbb{C}^m$. The principles of such complex lattice are almost the same, however, now elements of $\mathbf{B}$ and $\mathbf{a}$ take values from $\mathbb{C}$. Note that a complex lattice has an equivalent real representation which has double the rank of the corresponding complex lattice. Consider a lattice $\Lambda \subset \mathbb{C}^m$ with a generator matrix $\mathbf{B} = [\mathbf{z}_1, ..., \mathbf{z}_n]$. Now we can always express it with a real valued lattice $\Lambda_{\text{real}} \subset \mathbb{R}^{2m}$ and the corresponding $2m \times n$ generator matrix is given by

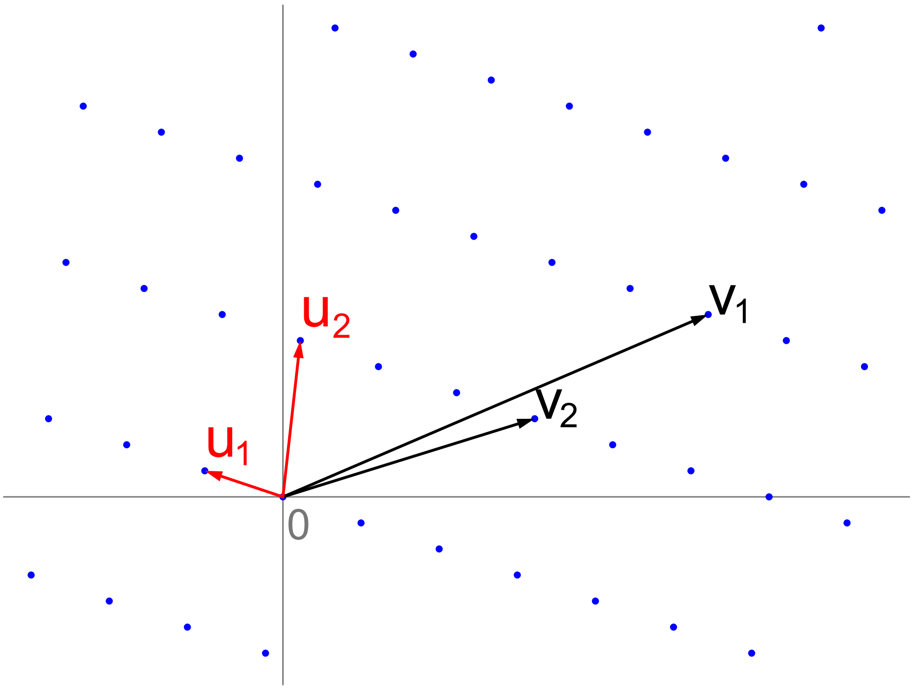

Figure 1: Two different bases for the same two dimensional lattice.

$$\mathbf{B}_{\text{real}} = \begin{bmatrix} \mathrm{Re}(z_{11}) & \dots & \mathrm{Re}(z_{1n}) \\ \mathrm{Im}(z_{11}) & \dots & \mathrm{Im}(z_{1n}) \\ \vdots & \ddots & \vdots \\ \mathrm{Re}(z_{m1}) & \dots & \mathrm{Re}(z_{mn}) \\ \mathrm{Im}(z_{m1}) & \dots & \mathrm{Im}(z_{mn}) \end{bmatrix}. \tag{5}$$

In other words we convert each column of $\mathbf{B}$ to real vector by separating each of their complex elements into two adjacent real elements, that is the real and imaginary parts of the original complex element. This process doubles the amount of rows in $\mathbf{B}_{\text{real}}$ but both representations ultimately describe the same lattice. [4]

2.1 Closest vector problem

The most relevant mathematical problem related to sphere decoding, a communications application of interest in this paper, is the closest vector problem (CVP). Let J be a general vector space. Given a lattice $\Lambda \subset J$ and an input point $\mathbf{y} \in J$ the problem is to find a lattice point $\hat{\mathbf{x}} \in \Lambda$ that is closest to $\mathbf{y}$. More precisely $\hat{\mathbf{x}} \in \Lambda$ has to meet the following condition

$$\|\mathbf{y} - \hat{\mathbf{x}}\| \leq \|\mathbf{y} - \mathbf{x}\|, \quad \forall \mathbf{x} \in \Lambda. \tag{6}$$

In this thesis we mostly consider $J = \mathbb{C}^n$.

For a fixed point $\mathbf{x}' \in \Lambda$ the set of vectors $\mathbf{y}_i \in J$ that satisfy the inequality (15) is called the Voronoi region $\mathcal{V}_{\mathbf{x}'} \subset J$ (see figure 2) of lattice point $\mathbf{x}'$ [5]. This means that if $\mathbf{y} \in \mathcal{V}_{\mathbf{x}'}$ then the solution to CVP is $\mathbf{x}'$. An example of lattice's Voronoi cells is illustrated in figure 2. If $\mathbf{y} \in \partial\mathcal{V}_{\mathbf{x}'}$ the solution to CVP is a random choice between points whose Voronoi region contains that boundary. The volume of the $\mathcal{V}_{\mathbf{x}'}$ is given by $\det(\Lambda) = \sqrt{\det(\mathbf{B}^T\mathbf{B})}$, where $\mathbf{B}$ is the generator matrix of Λ.

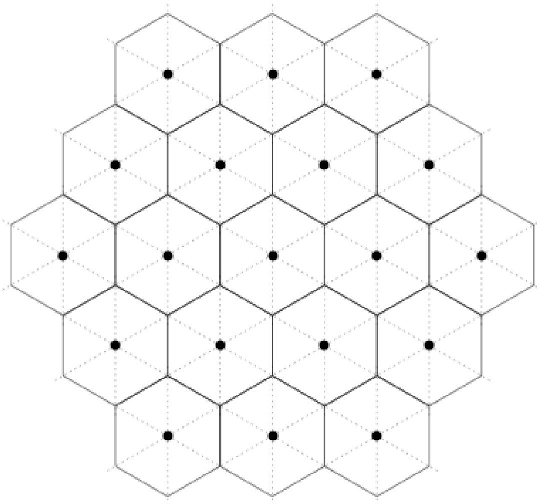

Figure 2: The Voronoi cells of the two dimensional hexagonal lattice A_2.

CVP is trivial for orthogonal lattices as one could just round every coordinate of $\mathbf{y}$ to nearest integer an get the correct answer, but CVP has been shown to be NP-hard as the function of the lattice rank in general case when we consider skewed lattices and thus all known algorithms for CVP have exponential worst case complexity [3]. Non-trivial lattices appear in most communications applications, for example when using maximum likelihood decoding for lattice codes sent via channel that induces fading. Such decoding process can be thought as a CVP.

2.2 Space–time lattice codes

Space–time lattice codes are a robust way of encoding the data on a wireless channel that uses multiple transmit and receiver antennas, i.e. a MIMO-system. Wireless channels are usually very error prone due to interference and fading caused by the surrounding electrical devices, obstacles and environment. This method helps to improve the reliability of the data transmission on such channel, meaning we get decoding errors less likely. Codebook from a certain space–time lattice code constellation can be represented as a set of matrices, which is defined as

$$\mathcal{L} = \left\{ \sum_{i=1}^{k} a_i \mathbf{X}_i \;\middle|\; \mathbf{X}_i \in \mathcal{M}_{m\times l}(\mathbb{C}),\ a_i \in S \subset \mathbb{Z} \right\}. \tag{7}$$

The $\mathbf{X}_i, i = 1, ..., k$ denote the constant basis matrices of the lattice code and the coefficients $a_i, i = 1, ..., k$ are integers from a finite signal set S which represent the data we want to send. We call $\mathcal{L}$ our codebook as it contains all possible codewords for a certain finite signal set. In this thesis we consider the pulse amplitude modulation (q-PAM) signal set of size q. It is defined as

$$S = \{a = 2u - q + 1 \mid u \in \mathbb{Z}_q\} \tag{8}$$

with $\mathbb{Z}_q = \{0, 1, ..., q-1\}$. So for example 4-PAM gives $S = \{-3, -1, 1, 3\}$ i.e. all the odd integers which absolute value is smaller than 4. The size of the signal set is usually chosen to be some power of two, i.e. $q = 2^b$. This way b information bits are mapped to each codeword we send. [1]

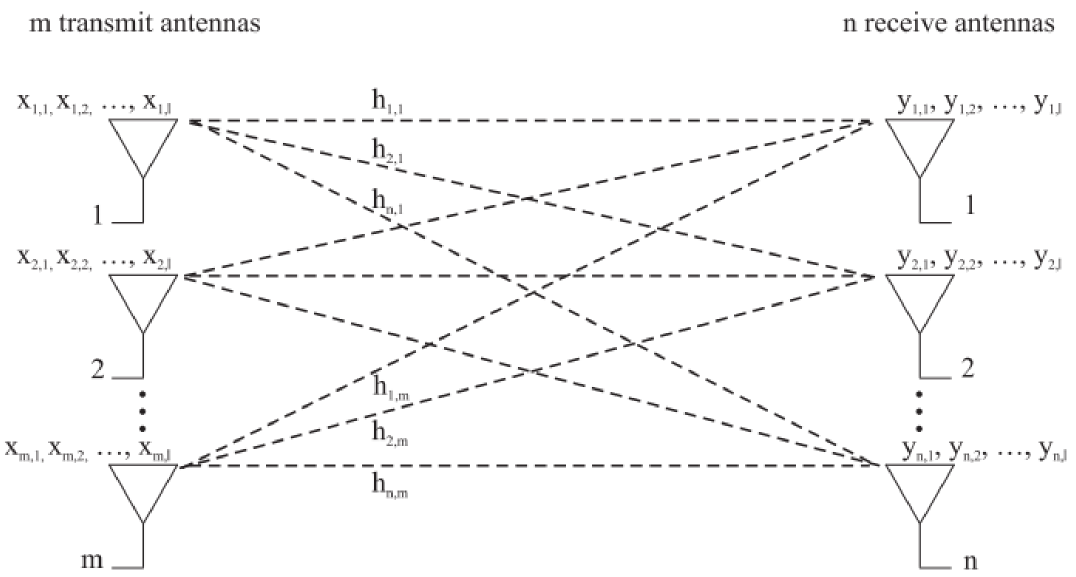

Figure 3: The MIMO-channel model with quasi-static Rayleigh fading [1].

Consider a multiple antenna system as in figure 3 of m transmit and n receive antennas. We choose an $m \times l$ complex matrix $\mathbf{X} \in \mathcal{L}$ constructed from (7) as the codeword which we send through this wireless channel, where l denotes the number of channel uses (time slots) required to send the codeword. In this thesis we consider quasi-static channel model i.e. path gains $h_{i,j}$ in figure 3 are assumed to stay constant during the sending of single codeword i.e. during l channel uses and then change.

The simulated received noisy $m \times l$ codeword matrix can be obtained like this

$$\mathbf{Y} = \mathbf{HX} + \mathbf{N} \tag{9}$$

where $\mathbf{H}$ is $n \times m$ channel matrix that simulates the path gains in figure 3 and $\mathbf{N}$ is a $n \times l$ noise matrix. $\mathbf{H}$ and $\mathbf{N}$ is generated from independent and identically distributed (i.i.d.) complex Gaussian random entries with zero mean for each simulation round. Their distributions use different variance though.

The component $y_{i,j}$ of the received matrix corresponds to the j-th signal received by antenna i, which is a superposition of faded versions of all m transmitted signals with additive noise, i.e. it can be obtained like

$$y_{i,j} = h_{i,1}x_{1,j} + h_{i,2}x_{2,j} + ... + h_{i,m}x_{m,j} + n_{i,j} \tag{10}$$

for all $1 \leq i \leq n,\ 1 \leq j \leq m$. The $n_{i,j}$ is the corresponding noise matrix component.

The problem of decoding is now: Given $\mathbf{Y}$, which $\mathbf{X} \in \mathcal{L}$ was sent? According to maximum likehood principle this can be viewed as CVP [1] and is discussed in the next chapter in further detail.

3 Sphere decoder

In this chapter we will discuss an efficient decoding algorithm for space–time block codes known as the sphere decoder. The name *sphere decoder* actually comes from Pohst's idea of conducting the search within a finite hypersphere in 1981 [3]. This thesis is particularly interested in the Schnorr-Euchner implementation of the algorithm, which is based on Pohst strategy and the Babai nearest plane algorithm [3].

The problem of decoding the correct codeword from the set of all possible codewords from the received noisy codeword can be reduced to CVP discussed earlier [1]. But exhaustive search over all points (codewords) in the finite lattice of interest (codebook) is just not feasible solution as the number of possible codewords grows exponentially with the rank of the lattice code k in equation (7) and the size of the signaling set q in equation (8).

Luckily the sphere decoder algorithm exists, which instead of searching through all lattice points within the finite signal set boundaries only considers lattice points within a hypersphere around the received vector of certain radius like in figure 4. This significantly reduces the complexity of the search algorithm.

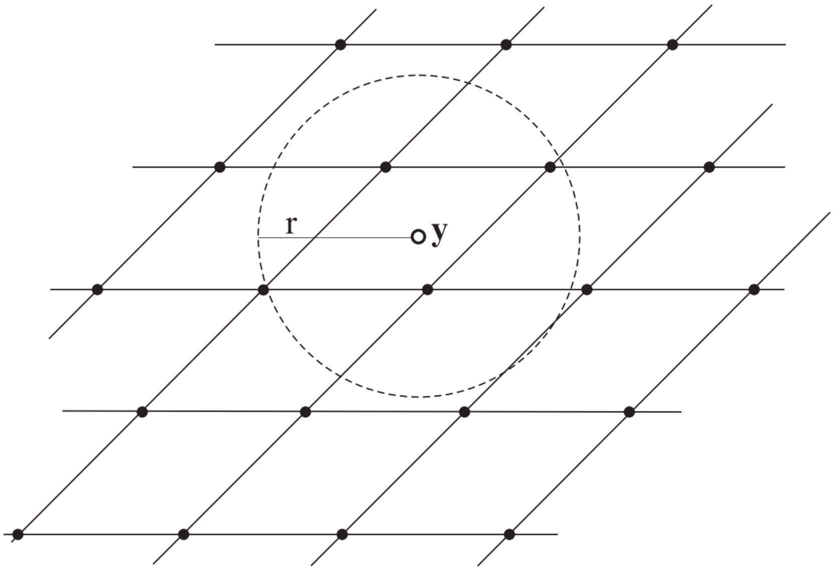

Figure 4: The CVP with sphere decoding in two dimensional lattice. The search complexity can be reduced by giving an upper bound for the closest point distance [1].

Although one can do even better than that: everytime the algorithm finds a lattice point within the hypersphere and one is interested in finding the closest one, one can obviously shrink down the search radius. It is now sufficient to only consider lattice points that are either as far away as the point the algorithm just found, or lattice points that are closer, thus the search radius can be set to equal the distance to that point.

There is however still the problem of choosing the initial radius for the sphere decoder. In simulations one can always *cheat* and set the radius roughly equal the power of the noise we apply to the system. In real life applications however, where we cannot predict the power of the noise in the channel, this is not trivial task for some implementations of sphere decocer. Luckily the Schnorr-Euchner strategy, which this paper is interested in, uses Babai nearest plane algorithm as a part of its implementation, which guarantees the algorithm always finds a nearby point without any initial radius at all. Once it finds a lattice point that is reasonably near the codeword that was received, the algorithm can use its distance as the upper bound for the sphere decoder and only consider lattice points that are within that radius from that point on.

However since in practise we consider finite subsets of the lattice, there is a possibility that the Babai point lies outside of the codebook region, which causes the algorithm to loop infinitely [3]. To avoid this the Algorithm 1 and 2 introduced in this paper, which are both modifications of the Schnorr-Euchner strategy, consider only lattice points that have values taken from the chosen finite signal set as their components. The real benefit of the integrated Babai algorithm in Schnorr-Euchner strategy comes from the increased probability of finding the closest lattice point early at the cost of slightly increased computational complexity compared to Pohst strategy [3].

3.1 Recursive sublattice decomposition

For understanding how the sphere decoder works let us introduce the concept of sublattice decomposition from n-dimensional lattice to a stack of $(n-1)$-dimensional sublattices. As one could guess this can also be done recursively and ultimately we end up with 0-dimensional lattices also known as the lattice points. The motivation to this recursive representation of a lattice is that one only needs to consider intervals of sublattice layer indices when doing the distance comparison in each dimension from n to 0.

Let us consider a $m \times n$ lattice generator matrix $\mathbf{G}$ for a lattice Λ and write it like

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}' & \mathbf{v}_n \end{bmatrix} \tag{11}$$

where $\mathbf{G}'$ is a $m \times (n-1)$ matrix and the last column vector can be written as $\mathbf{v}_n = \mathbf{v}_{\|} + \mathbf{v}_{\perp}$, with $\mathbf{v}_{\|}$ in the column space of $\mathbf{G}'$ and $\mathbf{v}_{\perp}$ in the null space of $\mathbf{G}'$. With this notation we can decompose any n-dimensional lattice as follows

$$\Lambda(\mathbf{G}) = \bigcup_{u_n=-\infty}^{+\infty} \Big\{ \mathbf{c} + u_n\mathbf{v}_{\|} + u_n\mathbf{v}_{\perp} \mid \mathbf{c} \in \Lambda(\mathbf{G}'),\ u_n \in \mathbb{Z} \Big\}. \tag{12}$$

Equation (12) basically describes a stack of $(n-1)$-dimensional translated sublattices, where u_n is the index of each sublattice of an n-dimensional lattice Λ. The hyperplanes over which the sublattices are spanned are called *layers*. For example you can think of the horizontal lines in figure 4 as one dimensional layers and the discrete points in them as sublattices of the whole two dimensional lattice.

To better understand the given expression in equation (12) let's consider the sublattice in the base layer $u_n = 0$, which is just all the points in the sublattice generated by $\mathbf{G}'$. Similarly we can get all other sublattices from this base sublattice by offsetting all the points in it by linear combination $u_n\mathbf{v}_{\parallel} + u_n\mathbf{v}_{\perp}$, where $\mathbf{v}_{\parallel}$ denotes a translation of all points in that same hyperplane and $\|\mathbf{v}_{\perp}\|$ is the orthogonal distance between two adjacent layers in the n-th dimension. By taking the union of all these sublattices we get the whole n-dimensional lattice. In figure 4 for example you can think of the middle horizontal line as the base layer.

If $\mathbf{G}$ is an upper triangular matrix, we can easily express $\mathbf{v}_{\perp}$ as $\mathbf{v}_{\perp} = (0, 0, ..., v_{nn})^T$ and thus we can write simply $\|\mathbf{v}_{\perp}\| = |v_{nn}|$. In fact any generator matrix $\mathbf{G}$ can be rotated to upper triangular form [3] with $v_{nn} > 0$ so we can simply denote the distance between $(k-1)$-dimensional layers with v_{kk} later on.

With the concept of recursive sublattice decomposition, CVP in n-dimensional lattice $\Lambda(\mathbf{G})$ can be done recursively with a finite number of $(n-1)$-dimensional search operations. If $\mathbf{x} \in \mathbb{R}^m$ is a vector to decode in the lattice $\Lambda(\mathbf{G})$, which is decomposed as in (12), the orthogonal distance y_n from $\mathbf{x}$ to the $(n-1)$-dimensional layer with index u_n is given by the formula

$$y_n = |u_n - \hat{u}_n| \cdot \|\mathbf{v}_{\perp}\| \tag{13}$$

where

$$\hat{u}_n = \frac{\mathbf{x}\mathbf{v}_{\perp}^t}{\|\mathbf{v}_{\perp}\|^2} \tag{14}$$

i.e. $\hat{u}_n$ is a scalar projection of $\mathbf{x}$ onto $\mathbf{v}_{\perp}$. Inserting this scalar projection i.e. the magnitude of $\mathbf{x}$ vector's orthogonal component into the equation (13) pretty obviously yields the orthogonal distance of $\mathbf{x}$ to the layer with index u_n. This is illustrated in figure 5.

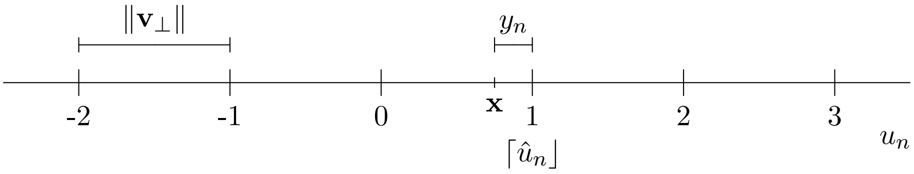

Figure 5: A stack of parallel layers of an n-dimensional lattice. u_n denotes the layer index.

3.2 Schnorr-Euchner lattice search algorithm

Now regarding CVP it is sufficient to consider these orthogonal distances in equation (13) and we can thus write the CVP as

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \in \Lambda(\mathbf{G})}{\operatorname{argmin}} \sum_{n=0}^{m} y_n \tag{15}$$

where $\hat{\mathbf{x}}$ is the lattice point closest to $\mathbf{x}$. This can be thought as traversing a search tree of depth m from root node to the leaf node at lowest level with smallest sum of vertex weights. In the figure 6 a search tree for 4-dimensional CVP is illustrated. The tree is not full as not all subtrees need to be considered in smart implementations of the lattice search algorithm as the search can be stopped early. Note that only paths from root to level 4 nodes are considered feasible solutions to CVP as the solution vector $\hat{\mathbf{x}}$ has to have 4 components. The node values on each level k of the search tree represent candidate value for the $(m-k+1)$-th component of the solution vector selected from some finite signal set. Here $m = 4$.

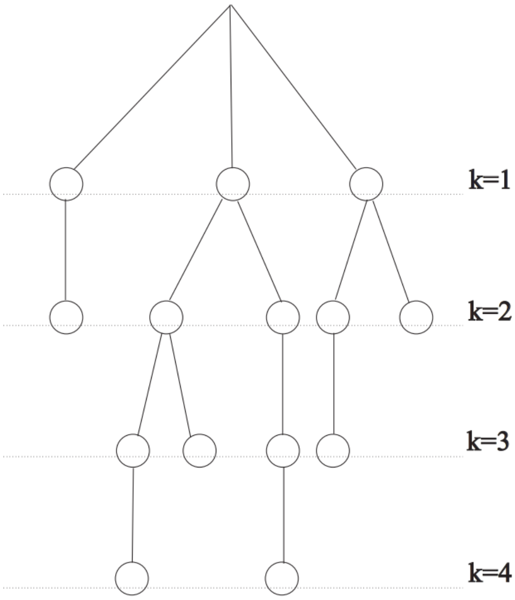

Figure 6: Traversal of a lattice search algorithm in a four dimensional lattice [1].

So how does one select the layer, which is closest to $\hat{\mathbf{x}}$ in each dimension n? As the layer indices u_n take values from $\mathbb{Z}$, clearly selecting $u_n = \lceil \hat{u}_n \rfloor$ in equation (13) gives you the smallest y_n in that dimension. Considering only this option in each dimension yields the Babai nearest plane algorithm, which reduces the search problem to a $(m-1)$-dimensional one. It is a quick way to find a nearby lattice point, but it is not necessarily the closest one [3]. The point found by this algorithm is called the Babai point.

To ensure one indeed finds the closest lattice point, which in decoding applications like the sphere decoder is crucial, one needs to be a bit more throughout than this. Indeed Schnorr-Euchnerr sphere decoder does not stop here but instead climbs up the search tree and enumerates through all feasible points that are closer than the first point found also known as the *Babai point*. This what the sphere decoder using Pohst strategy would also do. What's different in Schnorr-Euchner strategy from Pohst apart from not requiring initial radius thanks to the Babai nearest plane algorithm is the way of enumerating sublattice indices in equation (13). While Pohst method would just go through the layers in ascending order within the distance upper bound (sphere radius), Schnorr-Euchnerr starts from the midpoint i.e. $u_n = \lceil \hat{u}_n \rfloor$ and zigzags around it until it hits the upper bound (see figure 5). In other words the Schnorr-Euchner enumeration goes through the layers in ascending order of the orthogonal distances y_n. The benefit of doing this is quite obvious: if one finds a layer u_n that is outside the current sphere radius one can stop the search in that sublattice early, as clearly there are no longer points within radius in that sublattice, thanks to the ascending distance ordering. This significantly reduces the search tree complexity depicted in figure 6 and thus makes the algorithm more efficient [1][3].

3.3 Sphere decoder implementation

The sphere decoder algorithm used by the program is a modification of the Schnorr-Euchner strategy described earlier which also accounts for finite signal set boundaries with some optimizations. It is based on the algorithms introduced in [1] and [7], but there are a couple of key differences. The new implementation natively handles q-PAM signal sets defined in equation (8), so the signal set does not need to be mapped to interval $[0, q-1]$ first like in previous algorithms, which was confusing when comparing the input and output vectors. The program also uses the dedicated and efficient linear algebra library called *Armadillo* [6]. The library also has intuitive syntax, which makes the code rather easy to read for those who wish to understand every detail of it. While the original algorithm in [7] also handled spherically shaped codebooks, this program introduces two separate algorithms: one for parallelotope and one for spherically shaped codebooks. Pseudocodes for both algorithms are given in this paper. This division saves a lot of unnecessary comparisons when using the program with a normal pallelotope shaped codebook, which not surpricingly requires a much simpler algorithm.

The way the algorithms in the program are structured, provides a lot of modularity. This gives the experienced user with programming background an option to program his or her own custom simulations using this application programming interface within the main program source file.

The sphere decoder consist of two parts: *input preprocessing* and *decoding* as can be seen in figure 7. Preprocessing converts the input complex matricies into their real representation using equation (5) and rotates the faded channel lattice generator matrix into upper triangular form with QR decomposion in order to use the sphere decoder. The received vectorised codeword $\mathbf{y}$ also has be mapped to same coordinates as $\mathbf{R}$ by multiplying it with inverse of the rotation matrix $\mathbf{Q}$. In other words, instead of solving $\mathbf{B}\hat{\mathbf{x}} = \mathbf{y}$ one can solve the equivalent problem $\mathbf{R}\hat{\mathbf{x}} = \mathbf{y}'$. Doing this conversion is essential in order for the sphere decoder to work, as upper triangular form enables simple backsubstitution for partial radius calculations in each dimension.

The decoding process is divided into two algorithms, first one for normal codebooks (Algorithm 1) and the second one for spherical codebooks (Algorithm 2). The algorithms are very similar, so the differences in algorithm 2 to algorithm 1 are framed to emphasize them. Both algorithms are based on the ones introduced in [1],

[7] and [8]. They both output, if successful, a vector $\hat{\mathbf{x}}$ that is closest lattice point to $\mathbf{y}'$ in a lattice generated by $\mathbf{R}$. According to maximum likehood principle and the good behavior of the lattice codes under fading and noise, this should also very likely be the correct codeword that was sent through the wireless channel [1]. Then the program goes back to flowchart 8, compares the result and checks if this is actually the case.

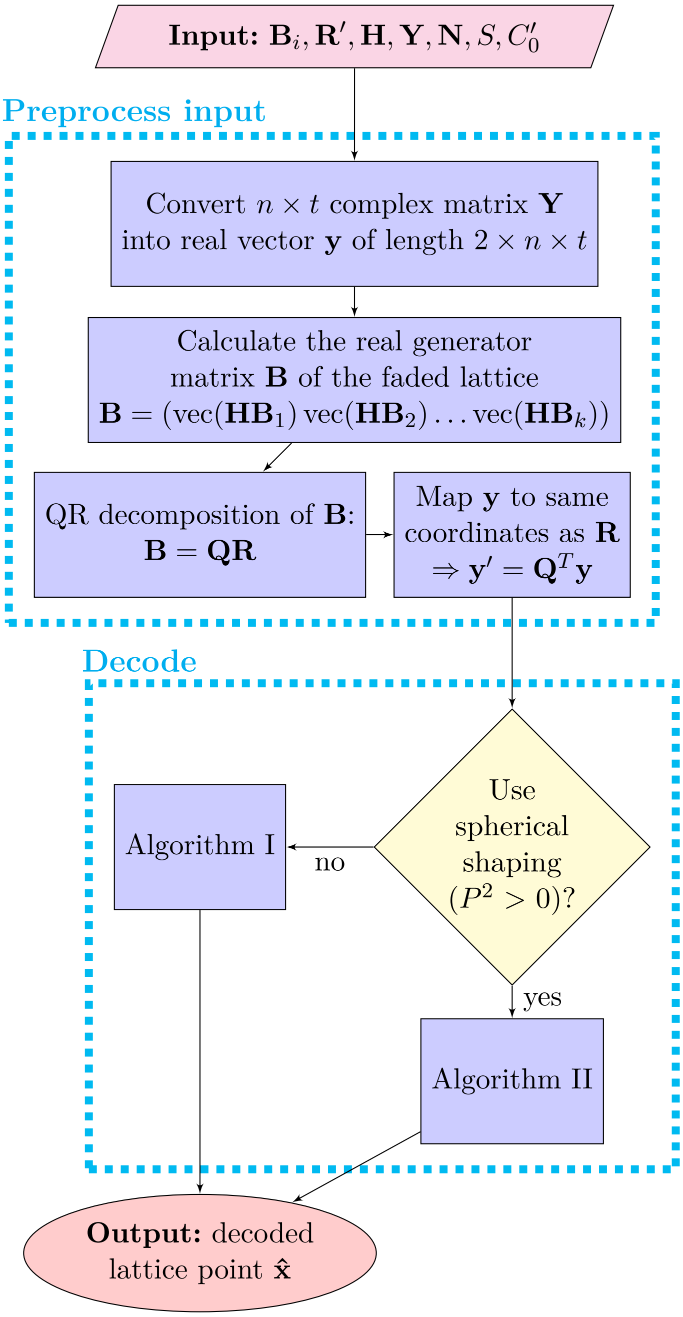
Input: $\mathbf{B}_i, \mathbf{R}', \mathbf{H}, \mathbf{Y}, \mathbf{N}, S, C_0'$
Preprocess input
Convert $n \times t$ complex matrix $\mathbf{Y}$ into real vector $\mathbf{y}$ of length $2 \times n \times t$
Calculate the real generator matrix $\mathbf{B}$ of the faded lattice $\mathbf{B} = (\mathrm{vec}(\mathbf{HB}_1)\,\mathrm{vec}(\mathbf{HB}_2)\ldots\mathrm{vec}(\mathbf{HB}_k))$
QR decomposition of $\mathbf{B}$: $\mathbf{B} = \mathbf{QR}$
Map $\mathbf{y}$ to same coordinates as $\mathbf{R}$ $\Rightarrow \mathbf{y}' = \mathbf{Q}^T\mathbf{y}$
Decode
Use spherical shaping $(P^2 > 0)$?
no
yes
Algorithm I
Algorithm II
Output: decoded lattice point $\hat{\mathbf{x}}$

Figure 7: The sphere decoder algorithm flowchart.

Algorithm I, (Input $C_0', \mathbf{y}', \mathbf{R}$, S. Output $\hat{\mathbf{x}}$)

```
Step 1. Initialization) Set $i := k, T_k := 0, \xi_k := 0$ and $d_c := C_0'$ (current sphere squared radius).

Step 2. (DFE on $x_i$) Set $x_{\text{temp}} := \underset{a \in S}{\text{argmin}} \left| \frac{y_i' - \xi_i}{r_{i,i}} - a \right|$.
    If $x_{\text{temp}} \leq S_1$ set $\Delta_i := 2$.
    Else if $x_{\text{temp}} \geq S_q$ set $\Delta_i := -2$.
    Else set $\Delta_i := 2 \cdot \text{sign}\,(y_i' - \xi_i - r_{i,i}x_i)$.

Step 3. (Main step) If $d_c < T_i + |y_i' - \xi_i - r_{i,i}x_i|^2$, then go to Step 4 (i.e., we are outside the sphere).
    Else if $x_i \notin S$ (i.e., we are inside the sphere but outside the signal set boundaries)
    then {Set $x_{\text{next}} := x_i + \Delta_i$. If ($x_i < S_1$ and $x_{\text{next}} > S_q$) or
    ($x_i > S_q$ and $x_{\text{next}} < S_1$) then go to Step 4, else go to Step 6.}
    Else (i.e., we are inside the sphere and signal set boundaries)
        If $i > 1$ then
            {let $\xi_{i-1} := \sum_{j=1}^{k} r_{i-1,j}x_j$,
            $T_{i-1} := T_i + |y_i' - \xi_i - r_{i,i}x_i|^2$,
            $i := i - 1$ and go to Step 2}.
        Else if $i = 1$ then go to Step 5.

Step 4. If $i = k$, terminate, else set $i := i + 1$ and go to Step 6.

Step 5. (A valid point is found) Let $d_c := T_1 + |y_1' - \xi_1 - r_{1,1}x_1|^2$, save $\hat{\mathbf{x}} := \mathbf{x}$.
    Then, let $i := i + 1$ and go to Step 6.

Step 6. (Schnorr-Euchner enumeration of level $i$) Let $x_i := x_i + \Delta_i$,
    $\Delta_i := -\Delta_i - 2 \cdot \text{sign}(\Delta_i)$ and go to Step 3.
```

Algorithm II, (Input $C_0, \mathbf{y}', \mathbf{R}, S$, $\boxed{\mathbf{R}', P^2}$. Output $\hat{\mathbf{x}}$)

```
Step 1. Initialization) Set $i := k, T_k := 0, \xi_k := 0$, $\boxed{P_k := 0, \eta_k := 0}$ and $d_c := C_0'$
    (current sphere squared radius).

Step 2. (DFE on $x_i$) Set $x_{\text{temp}} := \underset{a \in S}{\text{argmin}} \left| \frac{y_i' - \xi_i}{r_{i,i}} - a \right|$.
    If $x_{\text{temp}} \leq S_1$ set $\Delta_i := 2$.
    Else if $x_{\text{temp}} \geq S_q$ set $\Delta_i := -2$.
    Else set $\Delta_i := 2 \cdot \text{sign}\,(y_i' - \xi_i - r_{i,i}x_i)$.

Step 3. (Main step) If $d_c < T_i + |y_i' - \xi_i - r_{i,i}x_i|^2$, then go to Step 4 (i.e., we are
    outside the sphere).
    Else if $x_i \notin S$ or $\boxed{P^2 < P_i + \left|r'_{i,i}x_i + \eta_i\right|^2}$ (i.e., we are inside the sphere but outside
    the signal set boundaries)
    then {Set $x_{\text{next}} := x_i + \Delta_i$. If ($x_i < S_1$ and $x_{\text{next}} > S_q$) or
    ($x_i > S_q$ and $x_{\text{next}} < S_1$) then go to Step 4, else go to Step 6.}
    Else (i.e., we are inside the sphere and signal set boundaries)
        If $i > 1$ then
            {let $\xi_{i-1} := \sum_{j=1}^{k} r_{i-1,j}x_j$,
            $T_{i-1} := T_i + |y'_i - \xi_i - r_{i,i}x_i|^2$ ,
            $\boxed{\eta_{n-1} := \sum_{j=1}^{k} r'_{i-1,j}x_j,}$
            $\boxed{P_{i-1} := P_i + \left|r'_{i,i}x_i + \eta_i\right|^2 ,}$
            $i := i - 1$ and go to Step 2}.
        Else if $i = 1$ then go to Step 5.

Step 4. If $i = k$, terminate, else set $i := i + 1$ and go to Step 6.

Step 5. (A valid point is found) Let $d_c := T_1 + |y'_1 - \xi_1 - r_{1,1}x_1|^2$, save $\hat{\mathbf{x}} := \mathbf{x}$.
    Then, let $i := i + 1$ and go to Step 6.

Step 6. (Schnorr-Euchner enumeration of level $i$) Let $x_i := x_i + \Delta_i$,
    $\Delta_i := -\Delta_i - 2 \cdot \text{sign}(\Delta_i)$ and go to Step 3.
```

4 Lattice code simulations

4.1 Program implementation

The program discussed in this paper is essentially a lattice code simulator implemented as a command line application which the aforementioned sphere decoder algorithm is also part of. In this section a rough outline of the implementation is presented, which is most concisely presented in figure 8. With this insight one ought to create better simulations, especially when experimenting simulations outside the normal parameter range.

What the program actually simulates is the block error rate (BLER) of the lattice code under different levels of signal-to-noise ratios (SNR). Block error occurs when the decoded codeword is not exactly the same as the one we sent and the program counts these. Then BLER is simply the number of block errors divided by the number of simulation runs. The program can run multiple simulations in parallel, all of which simulate one SNR level each. The SNR is used to calculate the variance of the Rayleigh channel matrix coefficients in figure 3. The channel noise matrix variance was chosen to be fixed one during all simulations. Note that the channel model is not limited to MIMO like in figure 3 but can also be chosen to be SISO system which simulates one transmit and one receiver antenna.

The program requires at least two input files: the settings file and a basis matrix file, both located in their dedicated folders. Depending on the type of simulation one can also specify two additional input files for the sublattice basis matrices (for wiretap simulations) and simulation error limits (i.e. how many errors per SNR). To put it simply, the basis matrices are *the lattice code* you want to test and the other two input files are just parameters for the simulation. There are many possible combinations of parameters to use, we'll go through some of them in the next section with example simulation setups.

One key feature of the program is the support for codebooks of spherical shape. Unlike the codebook described in (7), which is clearly a parallelotope shaped subset of a lattice $\Lambda(\mathbf{X}_i)$ as the codeword coordinates take values from finite subset of $\mathbb{Z}$. In contrast to this, a spherical codebook only considers codewords in the lattice, that lie within certain radius of the origin. The simulations work rather differently depending on which type of codebook is used, this can also be clearly seen in flow charts 7 and 8. The idea of using a spherical codebook is to keep the maximum energy of the codebook reasonable as in skewed lattices parallelotope vertices and therefore some of the possible codewords can be rather far from the origin and require significantly more energy to send than the ones near origin. The problem however is estimating the correct radius to cover certain amount of codewords within the codebook boundary hypersphere. The program implements an option for estimating the radius automatically as seen in the program flowchart 8. In the settings file user can specify the exponent s and also the number or different radiuses considered around the initial radius when doing the search. The larger this variable is the better the radius estimate usually is (at the cost of increased computation time). If the estimation fails for some reason, it rolls back to user defined radius or using the parallelotope shaped codebook instead.

Once this is done the program creates the codebook, either the whole thing or a random sampled part of it, which can be decided in the settings file. The last option is usually recommended as the codebooks tend to be really large and attempting to create too large codebooks can even cause your computer to freeze. Although using random sampled codewords especially with spherical shaped codebooks introduces a certain bias as codewords near the boundary of the hypersphere tend to get picked more often. The pregenererated random sampled codebook is not used in the actual simulations, instead random codewords are generated on the fly. If one chooses to generate the whole codebook then the codewords are also picked from it during simulation, which might slightly speed up the actual simulations and also eliminate any biases given that the codebook was generated correctly.

The codebook $\mathcal{L}$ is then used for calculating its average and maximum energy. The energy of a single codeword matrix $\mathbf{X}$ is its squared Frobenius norm. The codebook average energy is later used in calculating the variance for the channel matrix. After all preparations are done, the program prints out the info relevant to current simulation and asks the user to verify it before starting the actual simulations. If something is out of place (e.g. wrong signaling set or ridiculously high average energy) you can abort the simulation here and double check your input files.

If user decides to continue, parallel SNR simulations (up to number of CPU cores in the machine) are started. In the settings file the SNR range and step can be defined, which ultimately decides the number of simulations to run. Each simulation can print out its current status every n simulation rounds if n is specified in the settings file. The each simulation continues until both error minimum and required number of rounds are reached, as seen in the figure 8. These can be set in the settings file as well. As the number of errors decreases significantly with the increase of SNR, one can use a dedicated csv input file for specifying error limits for each SNR simulation explicitly (an example is given in the /settings/ folder). This way one can require more errors in lower SNR levels and only few at high SNR as with constant error limit for each simulation the high SNR simulations might take really long time to complete. If a simulation takes too long to finish it can be forcefully aborted by hitting the key combination CTRL-C.

Once all simulations are finished the results are displayed in the console output and also printed in the /output/ folder into a csv file. The simulation results in the console output are in no particular order due to the parallelism involved, however the csv file is sorted in the ascending order of SNR level used in the simulation. If the plotting utility is used, plots for BLER and average search complexity are displayed immediately based on the simulation results. The csv file is also easy to import in Matlab for example for further analysis.

4.2 Simulation examples

To better grasp the concept of the lattice code simulations using this program, let's give a couple of common examples. To run successful simulations one needs to pay close attention to the simulation paramaters and the basis matrices of the lattice code.

4.2.1 Wiretap simulation

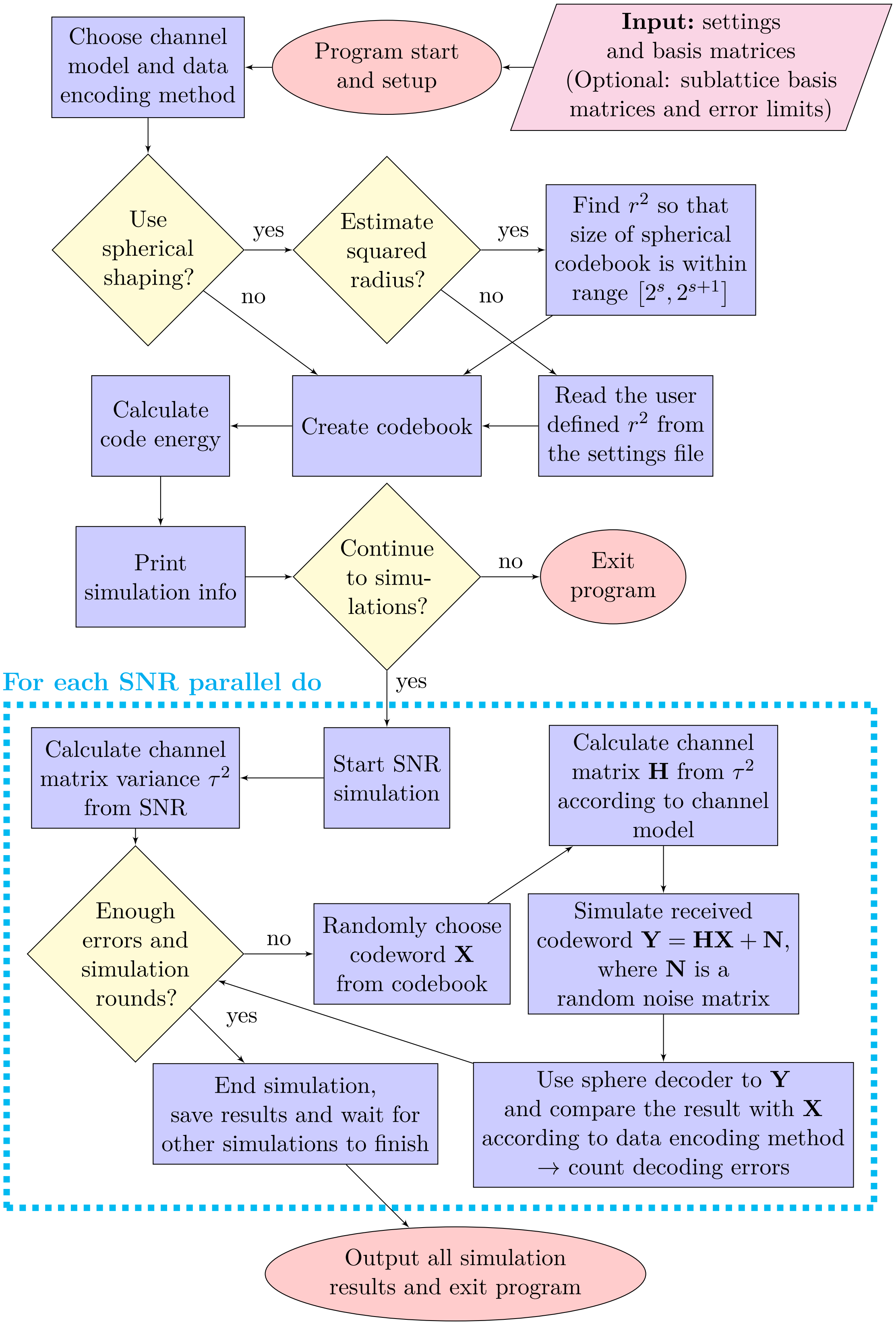
Input: settings and basis matrices (Optional: sublattice basis matrices and error limits)
Program start and setup
Choose channel model and data encoding method
Use spherical shaping?
yes
no
Estimate squared radius?
yes
no
Find r^2 so that size of spherical codebook is within range $[2^s, 2^{s+1}]$
Read the user defined r^2 from the settings file
Create codebook
Calculate code energy
Print simulation info
Continue to simulations?
no
Exit program
yes
For each SNR parallel do
Start SNR simulation
Calculate channel matrix variance τ^2 from SNR
Enough errors and simulation rounds?
no
Randomly choose codeword $\mathbf{X}$ from codebook
Calculate channel matrix $\mathbf{H}$ from τ^2 according to channel model
Simulate received codeword $\mathbf{Y} = \mathbf{HX} + \mathbf{N}$, where $\mathbf{N}$ is a random noise matrix
Use sphere decoder to $\mathbf{Y}$ and compare the result with $\mathbf{X}$ according to data encoding method → count decoding errors
yes
End simulation, save results and wait for other simulations to finish
Output all simulation results and exit program

Figure 8: The program flowchart.

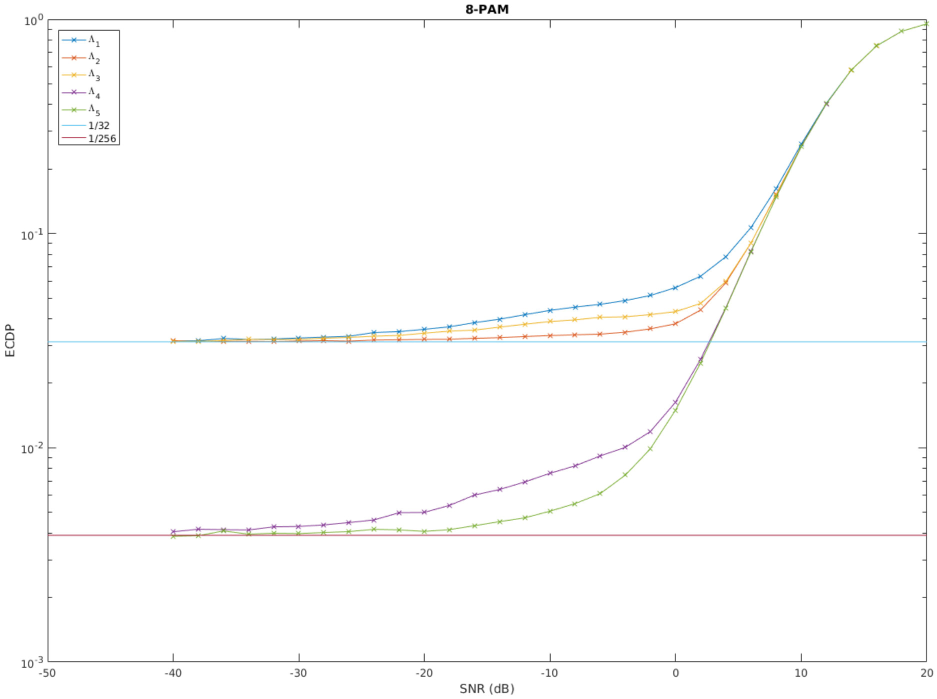

Figure 9: Five different sublattices for coset encoding in wiretap simulations. ECDP stands for "Eve's correct decoding probability".

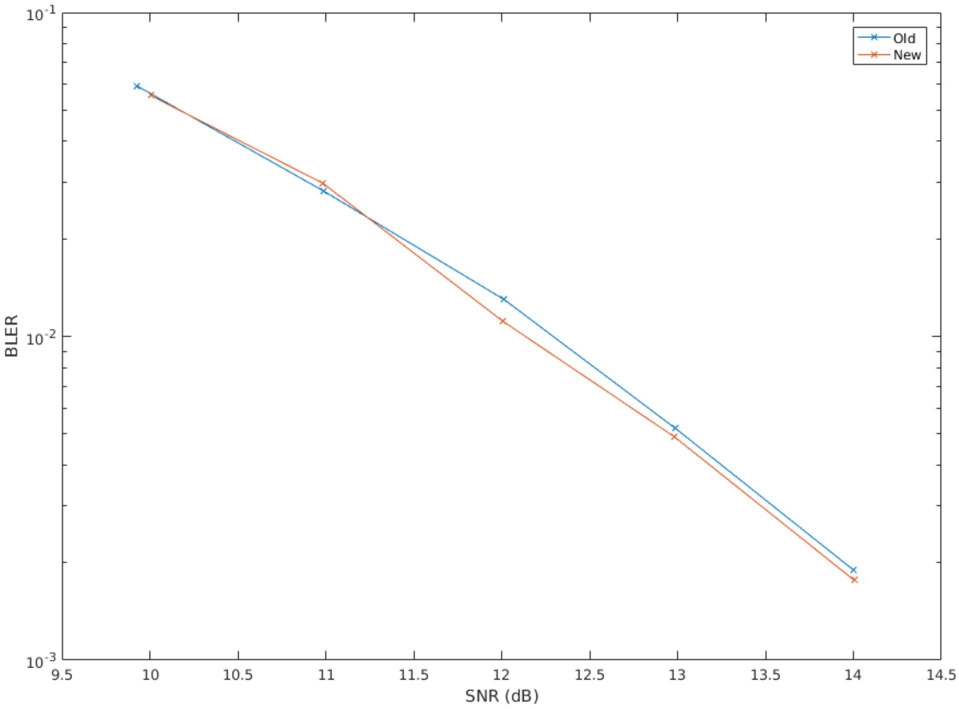

Figure 10: Comparing the MIDO (8-PAM, MIMO) simulation results with the old sphere decoder implementation from [7]. The curves are roughly the same.

5 Summary

References

[1] Mäki, M. *Space-time block codes and the complexity of sphere decoding.* Doria, Referenced 10.7.2017. Available at https://www.doria.fi/bitstream/handle/10024/54404/gradu2008maki-miia.pdf

[2] Cassels, J.W.S. *An introduction to the Geometry of Numbers*, New York, Springer–Verlag, 1971.

[3] Agrell, E., Eriksson, T. and Zeger, K. *Closest point search in lattices* IEEE Transactions on Information Theory, Vol. 48, pp.2201-2214, August 2002.

[4] Conway, J.H. and Sloane, N.J.A. *Sphere packings, lattices and groups.* Third edition, New York, Springer–Verlag, 1998.

[5] Zamir, R. *Lattices are Everywhere.* EE - Systems, Referenced 15.8.2017. Available: http://ita.ucsd.edu/workshop/09/files/paper/paper_312.pdf

[6] Conrad Sanderson and Ryan Curtin. *Armadillo: a template-based C++ library for linear algebra.* Journal of Open Source Software, Vol. 1, pp. 26, 2016. http://dx.doi.org/10.21105/joss.00026

[7] C. Hollanti and K. Ranto, *Maximal orders in space-time coding: Construction and decoding.* International Symposium on Information Theory and Its Applications, Auckland, 2008, pp. 1-5. Available at http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4895634

[8] M.O. Damen, H. El Gamal, G. Caire *On maximum-likelihood detection and the search for the closest lattice point* IEEE Transactions on Information Theory, Vol. 49, pp.2389-2402, October 2003. Available at http://ieeexplore.ieee.org/document/1237128/

A Program User Guide

A.1 Preface

The purpose of this program is to simulate the performance of different space–time lattice codes under different levels of SNR. Its implementation heavily depends on C++ linear algebra template library called Armadillo [6]. This program is a console application (i.e. it is operated from command line) that uses simple text based files for input and output. It also has a optional plotting utility which provides graphical interface for the simulation results. The program is primalily intented to be used on Linux based operating systems but should be portable to other platforms as well (there are some slight differences in the required compiler instructions and used libraries though). If there are compatibility issues optional parts of the program can be left uncompiled intentionally.

This guide assumes Linux based operating system, the steps described here might differ slightly when the program is used on other operating systems. The details regarding to those differences are not discussed here, but references to additional documentation are given. In the following chapters we will go through the installation and usage of the program in detail.

A.2 System requirements

The program runs optimally on Linux and Mac OS and can make use of multiple CPU cores for parallel computing of the simulations. Windows is not recommended due to the limitations posed by the Armadillo library [6]. The hardware requirements for running small scale simulations are relatively low, any modern laptop should do.

One can optionally make use of the computing power of the GPU, which could potentially speed up large matrix operations that the linear algebra library Armadillo does. This requires CUDA toolkit and NVBLAS, see details here: http://docs.nvidia.com/cuda/nvblas/index.html. Keep in mind that this feature is somewhat experimental, so if you encounter any problems you can always recompile the program without this feature.

Before trying to install and run the program itself, you need to make sure you have GNU Make and a compatible C++ compiler installed. The recommended GNU C++ compiler (g++) and GNU Make should be preinstalled in most Linux distributions but in case they are not, try installing the *build-essentials* package from the package manager. In Ubuntu you can do this by typing in terminal:

```
$ sudo apt-get install build-essential
```

If this does not work please refer to https://gcc.gnu.org/wiki/InstallingGCC and https://www.gnu.org/software/make/.

The next thing is to gather the required library packages:

1. OpenBLAS library: https://github.com/xianyi/OpenBLAS/wiki/Installation-Guide

2. LAPACK – Linear Algebra PACKage: http://www.netlib.org/lapack

3. ARPACK library: http://www.caam.rice.edu/software/ARPACK
4. Armadillo C++ Linear algebra library: http://arma.sourceforge.net/download.html
5. Boost C++ library (optional, for plotting only): http://www.boost.org

You can follow the links in the packege list for detailed instructions on how install those packages if you're facing problems (e.g. you're not installing the program on Linux) but the most straightforward way of installing them is by opening a terminal (in Linux CTRL+ALT+T) and using a package manager utility (e.g. apt-get in Ubuntu) to install those packages. The first three packages are required by the Armadillo library so you should install the packeges in the given order. Also note that not Linux based operating systems might require different libraries for Armadillo to work, in this case please refer to the Armadillo documentation by following the given link in the package list.

In the Linux package manager the libraries listed should go by names: *libopenblas-dev*, *liblapack-dev*, *libarpack-dev*, *libarmadillo-dev*, *libboost-all-dev*. Note that there might be some version number before the first dash in those names. The Boost library can safely be omitted for compatibility reasons. If all the install processes succeed you are now all set up to install the program itself.

A.3 Installing the program

Open terminal (CTRL+ALT+T) and create a folder for the program

```
$ mkdir sphere-decoder
$ cd sphere-decoder
```

Once you are in that folder copy the contents of this Gitlab repository there: https://version.aalto.fi/gitlab/pasi.pyrro/sphere-decoder. This requires you to have Aalto University login credentials. If you have trouble accessing the repository contact your supervisor who instructed you to use this program, they should have a copy of it. The copying from the repository can be done by either downloading the repository as a compressed folder (e.g. zip) and extracting it into the folder created earlier or by using the following command in the terminal (requires git to be installed on the system):

```
$ git clone git@version.aalto.fi:pasi.pyrro/sphere-decoder.git
```

After copying the files check the contents of that folder by typing this command in the terminal:

```
$ ll
```

Check if the contents of that folder look similar to those of the Gitlab repository. If everything looks all right, you're using Linux and want to do the plain installation (without plotting or GPU support), just type the following in terminal

```
$ make
```

to compile the program. For other operating systems you likely need to modify the Makefile located in the /src/ directory. For convenience there is also a file called *Makefile.mac* in /src/ directory. Replacing the default Makefile with that (by removing the file extension) should work for most MAC users.

There are a couple of options how to compile the program if you did manage to install the required libraries. If you installed everything listed above and want to do the full installation, then type this instead.

```
$ make with=plotting+gpu
```

You can choose also choose just one of those options, for example

```
$ make with=plotting
```

but keep in mind the requirements for them:

1. **plotting**: C++ Boost library
2. **gpu**: CUDA Toolkit and NVBLAS, see section A.2

If the compiling succeeded, now check if the program runs correctly by typing:

```
$ make run
```

If there are no errors regarding missing libraries or such then you have just correctly installed the program! If there are such errors during the compilation or program runtime, go back to previous section and double check that you have installed the correct libraries.

A.4 Setup and Input

Now that the program is installed it is time to take a look at the simulation parameters and the program input files in order to run successful simulations. In the /bases/ folder there are text files containing the lattice code basis matrices for the simulation. These files are really important as they contain the lattice code itself, the performance of which you want to simulate. Note that the program supports complex matrices as input, which are the preferred way of inputting your basis matrices. You can (and should) add your own basis matrix text files there. The program should not be too picky about the input format but do make sure you have something other than white spaces separating the matrix entries. The most tested input format is the Wolfram Mathematica complex matrix format, which should work in all cases. This format is also used in most of the example basis matrix files provided with the program, which you can use as a reference, like the one in figure A1.

For simulation parameters and other options one should navigate to /settings/ folder and open the default settings file called *settings.ini*. The key contents of the file are displayed in figure A2. There are usually also comments in those settings files that start after the character sequence '//' and end in the line end but these

```
{{1, 0},
{0, 1}}

{{0, -1},
{1, 0}}

{{0+I, 0},
{0, 0-I}}

{{0, 0+I},
{0+I, 0}}
```

Figure A1: An example basis matrix input file (alamouti.txt) for the Alamouti lattice code. It contains four 2x2 complex matrices, which form the base of the lattice code.

are left out of the figures for clarity reasons. The comments are of course ignored by the program and only hold meaning for the user. Make sure not to remove any of the 22 options or the program will complain you about missing options. If you want to disable an option, it is done by either leaving the option value empty (string options) or setting it to -1 (number options) as seen in the figure A2. In case you end up messing up the settings file in some way, easy way to reset it is to just delete the file and run the program normally. This will generate a new default settings file for you, like the one in figure A2.

When editing the settings, only modify the right hand side of the equality sign. If you change the option variable names, the program will not recognise them. You can always check the correct variable names from figure A2 or just generate a new default settings file if you happen to change them accidentally. The order of the setting variables does not matter, just make sure they're all on their own separate line. Remember to save the changes you make to that file. The program doesn't need to be recompiled for these changes to take effect, just run it again after saving the settings file.

One can also create multiple settings files in that folder. To use a different settings file one just needs to give its name as a command line argument for the program. This will be explained in the next chapter. Using multiple settings files is really useful for organizing different simulations so you don't have to touch every parameter everytime you want to do a different simulation.

A.5 Running the program

Once the program is set up correctly, running the program is a relatively easy task. Just make sure you're in the root folder of the local repository (the folder where the program is located) and type the following in terminal:

```
$ make run
```

```
basis_file=alamouti.txt
output_file=
coset_file=
error_file=
channel_model=mimo
x-PAM=4
energy_estimation_samples=-1
no_of_matrices=4
matrix_coefficient=1.0
time_slots=2
no_of_transmit_antennas=2
no_of_receiver_antennas=2
snr_min=-6
snr_max=20
snr_step=2
simulation_rounds=10000
required_errors=-1
plot_results=-1
stat_display_interval=-1
spherical_shaping_max_power=-1
codebook_size_exponent=-1
radius_search_density=100
```

Figure A2: The default settings file (settings.ini) for the sphere decoder program.

or

```
$ ./sphdec [settings_file.ini*]
```

The first command uses the default settings file and in the second one can optionally spesify the name of the settings file to be used (do not give the whole file path, just the name of a file in /settings/ folder). If no command line argument is given the first two commands are equal.

Once the program has started it prints useful data related to the current actions it is taking to the terminal as well as in the log.txt file located in the /logs/ folder. Visiting the log.txt file can be useful for reviewing previous program behaviour in the future. In the log file each program runtime should be separated by an empty line.

If the program gets stuck somewhere or takes too long to finish a large simulation, pressing CTRL+C in the terminal will terminate the execution of the program almost instantly. The program attempts to output the current simulation data even in this situation but it is not quaranteed to succeed so keep that in mind when doing that. For large simulations one needs to be patient as the simulation complexity grows exponentially with the lattice code dimension. It might be required to hit this key combination several times if the program does not terminate within five seconds or so.

A.6 Output and Plots

If the program exits without errors its output should be found from /output/ folder in csv format. This standard data format is easy to import into Matlab and other programs you might need. The filenames are by default named so that they are in chronological order so the latest simulation output should be the last one in the file listing. The csv format is easy to import in other programs like Matlab.

If the program was compiled successfully with the plotting make option and the following settings variable is set to

```
plot_results=1
```

then a couple of graphical interfaces for plots generated from the simulation data should pop up. If there are errors check that you also have GNUPlot installed on your system. Plotting gives you a quick review on how the simulation went. For more rigorous data analysis it is recommended to import the output csv file to some numerical scientific calculation software like Matlab.

A.7 General troubleshooting

Some general miscellaneous tips are listed here to help you troubleshoot errors while using the program:

1. Do not include the dollar sign in terminal commands. It's just there to indicate terminal input.