Lattice Codes and Sphere Decoding

Pasi Pyrrö

School of Science

Bachelor's special assignment
Espoo 10.7.2017

Thesis supervisor:

Prof. Camilla Hollanti

Thesis advisors:

Prof. Marcus Greferath

D.Sc. Oliver 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: 10.7.2017 Language: English Number of pages: 4+11

Degree programme: Mathematics and Systems Analysis

Supervisor: Prof. Camilla Hollanti

Advisors: Prof. Marcus Greferath, D.Sc. Oliver 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
$\det(\mathbf{X})$	Determinant of matrix $\mathbf{X}$

Abbreviations

SNR	Signal to noise ratio
CVP	Closest vector problem

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 lattice shape are kept reasonable, there exist algorithms, like the sphere decoder, that offer polynomial expected complexity [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 subset Λ of $\mathbb{R}^m$ is called a lattice of dimension n if it is defined as

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

where the set of vectors $\mathbf{b}_1, ..., \mathbf{b}_n \in \mathbb{R}^m$ 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

$$\Lambda = \{\mathbf{x} | \mathbf{x} = \mathbf{B}\mathbf{a}\} \tag{2}$$

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]. 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 $\mathbf{B}$ 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

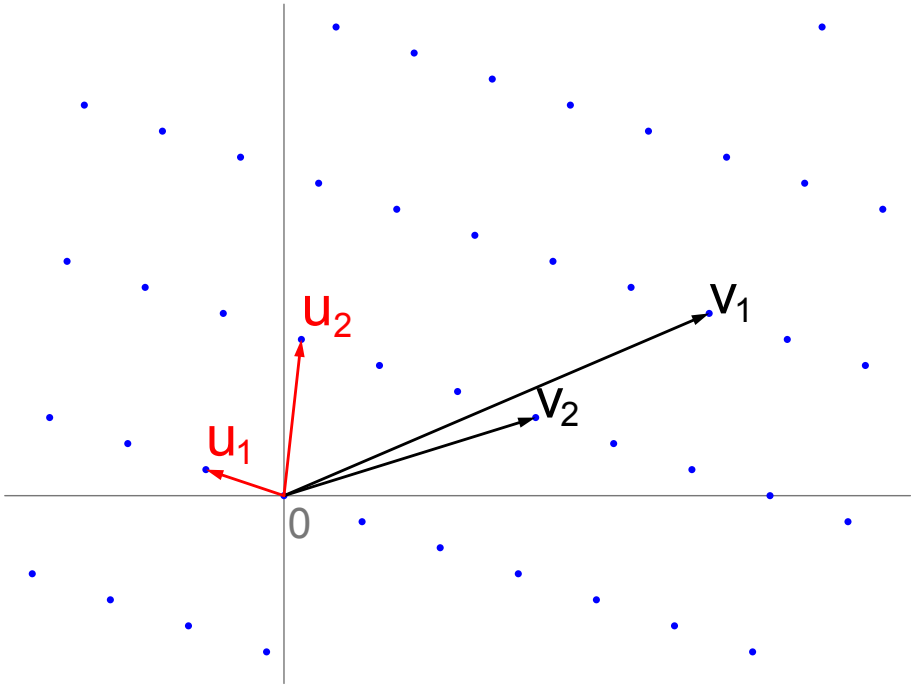

Figure 1: Two different bases for the same two dimensional lattice.

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

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

One relevant mathematical problem related to sphere decoding, a communications application of interest in this paper, is the closest vector problem (CVP). 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

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

J is the vector space, in which Λ belongs in. 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 (6) is called the Voronoi region (see figure 2) of lattice point $\mathbf{x}'$, $\mathcal{V}_{\mathbf{x}'} \subset J$. This means that if $\mathbf{y} \in \mathcal{V}_{\mathbf{x}'}$ then the solution to CVP is $\mathbf{x}'$ as can be clearly seen from 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 $\mathcal{V}_{\mathbf{x}'}$, also know as the lattice constant as it is independent of the choice of basis, is given by $\det(\Lambda) = \sqrt{\det(\mathbf{B}^T\mathbf{B})}$, where $\mathbf{B}$ is the generator matrix of Λ.

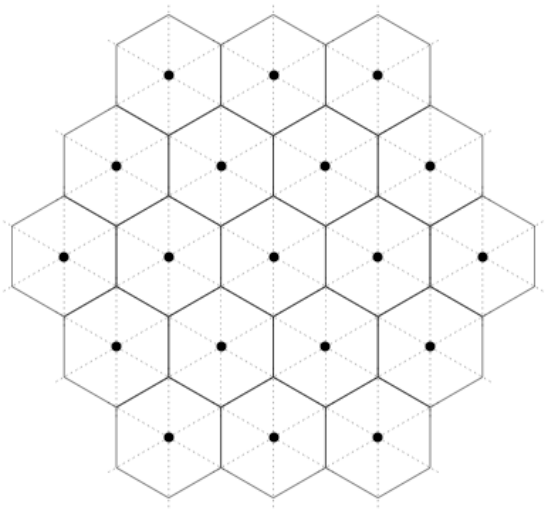

Figure 2: The voronoi cells of the two dimensional hexagonal lattice A_2.

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 a lattice codes sent via channel that induces fading. Such decoding process can be reduced to CVP and it's later explained in further detail.

2.2 Space–time lattice codes

3 Sphere decoder

4 Simulations and results

5 Summary

References

[1] Mäki, M. *Space-time block codes and the complexity of sphere decoding.* Doria, Referenced 10.7.2017. Available: 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.

A User guide

A.1 Abbreviations

SNR	Signal to noise ratio
CPU	Central processing unit
RAM	Random access memory
GPU	Graphical processing unit
C++	C++ programming language
OS	Operating system

A.2 Preface

The purpose of this program is to simulate the performance of different space–time lattice codes under different levels of SNR. It is a command line application 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). 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. In the following chapters we will go through the installation and basic use cases for the program in detail.

A.3 System requirements

Regarding the hardware as the rule of the thumb, the better the performance your computer has the faster the simulations, although nearly all modern computers with a keyboard and a monitor should suffice for small simulations. The program makes use of parallel CPU cores so one does not need to worry too much about single thread performance of the CPU. The program should not be too memory consuming except in some special use cases so no excessive amount of RAM is needed for fast simulations. GPU support might also be implemented in the future but for now the choice of GPU is irrelevant regarding the simulations.

Before trying to install and run the program itself, you need to make sure you have the all the required libraries and a compatible C++ compiler installed. The recommended GNU C++ compiler (g++) should be preinstalled in most Linux distributions but in case it is not, please refer to installation notes here: https://gcc.gnu.org/wiki/InstallingGCC.

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 for detailed instructions how install those packages if you're facing problems (e.g. you're not installing the program on Linux OS) but the most straightforward way of doing this is opening terminal in Linux and using the 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. In package manager those libraries should go by names: *libopenblas-dev*, *liblapack-dev*, *libarpack-dev*, *libarmadillo-dev* and *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 compability reasons, but requires a few modifications in the source code and Makefile for the compilation process. If all the install processes succeed you are now all set up to install the program itself.

A.4 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 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:

```
$ 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 and you're using Linux with Boost library, 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. If you didn't install the Boost C++ library you need to do the following in order to compile the program:

```
$ nano src/misc.hpp
```

In the text editor locate a line of code saying

```
#define PLOTTING // enable plotting (requires boost C++ library)
```

and either remove it or comment it out (type '//' in front of the line). After that save and exit by pressing CTRL+O, Enter and CTRL+X. After that type the command:

```
$ nano src/Makefile
```

and remove all the compiler flags that have the word 'lboost' in it. Save changes and exit the text editor as before. Now you should be able to compile the program by typing:

```
$ make
```

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, you have correctly installed the program! If there are errors during the compilation or program runtime, go back to previous section and double check that you have installed the correct libraries.

A.5 Setup and Input

A.6 Running the program

A.7 Output and Plots

A.8 General troubleshooting