Lattice algorithms

Guillaume Hanrot
Some slides courtesy of X. Pujol & D. Stehlé

LIP / ENSL, CNRS, INRIA, U Lyon, UCBL

Autrans, March 2013

Goals of the course

- Give some important ideas about lattice algorithms;
- Strong focus on lattice basis reduction;
- Some (sketches of) proofs;
- but primarily ideas.
- Please interrupt for any question.

Goals of the course

- Give some important ideas about lattice algorithms;
- Strong focus on lattice basis reduction;
- Some (sketches of) proofs;
- but primarily ideas.
- Please interrupt for any question.

Goals of the course

- Give some important ideas about lattice algorithms;
- Strong focus on lattice basis reduction;
- Some (sketches of) proofs;
- but primarily ideas.
- Please interrupt for any question.

Goals of the course

- Give some important ideas about lattice algorithms;
- Strong focus on lattice basis reduction;
- Some (sketches of) proofs;
- but primarily ideas.
- Please interrupt for any question.

Goals of the course

- Give some important ideas about lattice algorithms;
- Strong focus on lattice basis reduction;
- Some (sketches of) proofs;
- but primarily ideas.
- Please interrupt for any question.

Outline

- Introduction
- Lattice basis reduction – general issues
- Lattice basis reduction – “optimal” algorithms
- Lattice basis reduction – “blockwise” algorithms
- Approximate and exact algorithms for SVP / CVP
- Application : Coppersmith’s method

Introduction

- A lattice is $L(b_1, \ldots, b_n) : \mathbb{Z}b_1 \oplus \cdots \oplus \mathbb{Z}b_n$;
- Main algorithmic problems: given $(b_1, \ldots, b_n)$, $L = L(b_1, \ldots, b_n)$,
  - SVP: find $\lambda_1(L) := \min_{x \in L - \{0\}} \|x\|$ (and x)
  - CVP: given $t \in \mathbb{R}^n$, find $d(t, L)$ and $x \in L$ st. $\|x - t\| = d(t, L)$;
  - BDD: given $t \in R^n$ and $r \in \mathbb{R}^+$, find $x \in L$ st. $d(x, t) \leq r$.
  - ... plus variants (gap variants, preprocessing...)

Introduction

- A lattice is $L(b_1, \ldots, b_n) : \mathbb{Z}b_1 \oplus \cdots \oplus \mathbb{Z}b_n$;
- Main algorithmic problems: given $(b_1, \ldots, b_n)$, $L = L(b_1, \ldots, b_n)$,
  - SVP: find $\lambda_1(L) := \min_{x \in L - \{0\}} \|x\|$ (and x)
  - CVP: given $t \in \mathbb{R}^n$, find $d(t, L)$ and $x \in L$ st. $\|x - t\| = d(t, L)$;
  - BDD: given $t \in R^n$ and $r \in \mathbb{R}^+$, find $x \in L$ st. $d(x, t) \leq r$.
  - ... plus variants (gap variants, preprocessing...)

Introduction

- A lattice is $L(b_1, \ldots, b_n) : \mathbb{Z}b_1 \oplus \cdots \oplus \mathbb{Z}b_n$;
- Main algorithmic problems: given $(b_1, \ldots, b_n)$, $L = L(b_1, \ldots, b_n)$,
  - SVP: find $\lambda_1(L) := \min_{x \in L - \{0\}} \|x\|$ (and x)
  - CVP: given $t \in \mathbb{R}^n$, find $d(t, L)$ and $x \in L$ st. $\|x - t\| = d(t, L)$;
  - BDD: given $t \in R^n$ and $r \in \mathbb{R}^+$, find $x \in L$ st. $d(x, t) \leq r$.
  - ... plus variants (gap variants, preprocessing...)

Introduction

- A lattice is $L(b_1, \dots, b_n) : \mathbb{Z}b_1 \oplus \dots \oplus \mathbb{Z}b_n$;
- Main algorithmic problems: given $(b_1, \dots, b_n)$, $L = L(b_1, \dots, b_n)$,
  - SVP: find $\lambda_1(L) := \min_{x \in L - \{0\}} \|x\|$ (and x)
  - CVP: given $t \in \mathbb{R}^n$, find $d(t, L)$ and $x \in L$ st. $\|x - t\| = d(t, L)$;
  - BDD: given $t \in R^n$ and $r \in \mathbb{R}^+$, find $x \in L$ st. $d(x, t) \le r$.
  - ... plus variants (gap variants, preprocessing...)

Introduction

- A lattice is $L(b_1, \ldots, b_n) : \mathbb{Z}b_1 \oplus \cdots \oplus \mathbb{Z}b_n$;
- Main algorithmic problems: given $(b_1, \ldots, b_n)$, $L = L(b_1, \ldots, b_n)$,
  - SVP: find $\lambda_1(L) := \min_{x \in L - \{0\}} \|x\|$ (and x)
  - CVP: given $t \in \mathbb{R}^n$, find $d(t, L)$ and $x \in L$ st. $\|x - t\| = d(t, L)$;
  - BDD: given $t \in R^n$ and $r \in \mathbb{R}^+$, find $x \in L$ st. $d(x, t) \leq r$.
  - ... plus variants (gap variants, preprocessing...)

Introduction

- A lattice is $L(b_1, \ldots, b_n) : \mathbb{Z}b_1 \oplus \cdots \oplus \mathbb{Z}b_n$;
- Main algorithmic problems: given $(b_1, \ldots, b_n)$, $L = L(b_1, \ldots, b_n)$,
  - SVP: find $\lambda_1(L) := \min_{x \in L - \{0\}} \|x\|$ (and x)
  - CVP: given $t \in \mathbb{R}^n$, find $d(t, L)$ and $x \in L$ st. $\|x - t\| = d(t, L)$;
  - BDD: given $t \in R^n$ and $r \in \mathbb{R}^+$, find $x \in L$ st. $d(x, t) \leq r$.
  - ... plus variants (gap variants, preprocessing...)

Introduction

- A lattice is $L(b_1, \dots, b_n) : \mathbb{Z}b_1 \oplus \cdots \oplus \mathbb{Z}b_n$;
- Main algorithmic problems: given $(b_1, \dots, b_n)$, $L = L(b_1, \dots, b_n)$,
  - SVP: find $\lambda_1(L) := \min_{x \in L - \{0\}} \|x\|$ (and x)
  - CVP: given $t \in \mathbb{R}^n$, find $d(t, L)$ and $x \in L$ st. $\|x - t\| = d(t, L)$;
  - BDD: given $t \in R^n$ and $r \in \mathbb{R}^+$, find $x \in L$ st. $d(x, t) \leq r$.
  - ... plus variants (gap variants, preprocessing...)

Introduction (2)

Natural problems (linear problems in integers). Eg. Knapsack application:

- $x_1, \ldots, x_n \in \mathbb{N}^n$, $M = \sum_{i=1}^n \varepsilon_i x_i$, find $(\varepsilon_i) \in \{0,1\}^n$;
- NP-complete ;

L generated by the columns of

$$M = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \ldots & 1 & 0 \\ x_1 & x_2 & \ldots & x_n & -S \end{pmatrix}.$$

- If $\eta \in \mathbb{Z}^{n+1}$, $M\eta = (\eta_1, \ldots, \eta_{n-1}, \eta_n, \sum_{i=1}^n \eta_i x_i - \eta_{n+1} S)^t$.
- A short vector in L is $M\eta$ for $\eta_i = \varepsilon_i$, $\eta_{n+1} = 1$;

Introduction (2)

Natural problems (linear problems in integers). Eg. Knapsack application:

- $x_1, \ldots, x_n \in \mathbb{N}^n$, $M = \sum_{i=1}^n \varepsilon_i x_i$, find $(\varepsilon_i) \in \{0,1\}^n$;
- NP-complete ;

L generated by the columns of

$$M = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \ldots & 1 & 0 \\ x_1 & x_2 & \ldots & x_n & -S \end{pmatrix}.$$

- If $\eta \in \mathbb{Z}^{n+1}$, $M\eta = (\eta_1, \ldots, \eta_{n-1}, \eta_n, \sum_{i=1}^n \eta_i x_i - \eta_{n+1} S)^t$.
- A short vector in L is $M\eta$ for $\eta_i = \varepsilon_i$, $\eta_{n+1} = 1$;

Introduction (2)

Natural problems (linear problems in integers). Eg. Knapsack application:

- $x_1, \dots, x_n \in \mathbb{N}^n$, $M = \sum_{i=1}^n \varepsilon_i x_i$, find $(\varepsilon_i) \in \{0,1\}^n$;
- NP-complete ;

L generated by the columns of

$$M = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \dots & 1 & 0 \\ x_1 & x_2 & \dots & x_n & -S \end{pmatrix}.$$

- If $\eta \in \mathbb{Z}^{n+1}$, $M\eta = (\eta_1, \dots, \eta_{n-1}, \eta_n, \sum_{i=1}^n \eta_i x_i - \eta_{n+1} S)^t$.
- A short vector in L is $M\eta$ for $\eta_i = \varepsilon_i$, $\eta_{n+1} = 1$;

Introduction (2)

Natural problems (linear problems in integers). Eg. Knapsack application:

- $x_1, \dots, x_n \in \mathbb{N}^n$, $M = \sum_{i=1}^n \varepsilon_i x_i$, find $(\varepsilon_i) \in \{0,1\}^n$;
- NP-complete ;

L generated by the columns of

$$M = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \dots & 1 & 0 \\ x_1 & x_2 & \dots & x_n & -S \end{pmatrix}.$$

- If $\eta \in \mathbb{Z}^{n+1}$, $M\eta = (\eta_1, \dots, \eta_{n-1}, \eta_n, \sum_{i=1}^n \eta_i x_i - \eta_{n+1} S)^t$.
- A short vector in L is $M\eta$ for $\eta_i = \varepsilon_i$, $\eta_{n+1} = 1$;

Introduction (2)

Natural problems (linear problems in integers). Eg. Knapsack application:

- $x_1, \ldots, x_n \in \mathbb{N}^n$, $M = \sum_{i=1}^{n} \varepsilon_i x_i$, find $(\varepsilon_i) \in \{0,1\}^n$;
- NP-complete ;

L generated by the columns of

$$M = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \ldots & 1 & 0 \\ x_1 & x_2 & \ldots & x_n & -S \end{pmatrix}.$$

- If $\eta \in \mathbb{Z}^{n+1}$, $M\eta = (\eta_1, \ldots, \eta_{n-1}, \eta_n, \sum_{i=1}^{n} \eta_i x_i - \eta_{n+1} S)^t$.
- A short vector in L is $M\eta$ for $\eta_i = \varepsilon_i$, $\eta_{n+1} = 1$;

Introduction (2)

Natural problems (linear problems in integers). Eg. Knapsack application:

- $x_1, \ldots, x_n \in \mathbb{N}^n$, $M = \sum_{i=1}^n \varepsilon_i x_i$, find $(\varepsilon_i) \in \{0,1\}^n$;
- NP-complete ;

L generated by the columns of

$$M = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \ldots & 1 & 0 \\ x_1 & x_2 & \ldots & x_n & -S \end{pmatrix}.$$

- If $\eta \in \mathbb{Z}^{n+1}$, $M\eta = (\eta_1, \ldots, \eta_{n-1}, \eta_n, \sum_{i=1}^n \eta_i x_i - \eta_{n+1} S)^t$.
- A short vector in L is $M\eta$ for $\eta_i = \varepsilon_i$, $\eta_{n+1} = 1$;

Introduction (2)

- ... but lattice problems are hard:
  - SVP NP-hard under randomized reductions (even with almost poly approx factor);
  - CVP NP-hard (same) ;
  - BDD NP-hard for $r = c \cdot \lambda_1(L)$.
- $\Rightarrow$ approximation algorithms, exponential algorithms.
- Lattice basis reduction.

Introduction (2)

- ... but lattice problems are hard:
  - SVP NP-hard under randomized reductions (even with almost poly approx factor);
  - CVP NP-hard (same) ;
  - BDD NP-hard for $r = c \cdot \lambda_1(L)$.
- $\Rightarrow$ approximation algorithms, exponential algorithms.
- Lattice basis reduction.

Introduction (2)

- ... but lattice problems are hard:
  - SVP NP-hard under randomized reductions (even with almost poly approx factor);
  - CVP NP-hard (same) ;
  - BDD NP-hard for $r = c \cdot \lambda_1(L)$.
- $\Rightarrow$ approximation algorithms, exponential algorithms.
- Lattice basis reduction.

Introduction (2)

- ... but lattice problems are hard:
  - SVP NP-hard under randomized reductions (even with almost poly approx factor);
  - CVP NP-hard (same) ;
  - BDD NP-hard for $r = c \cdot \lambda_1(L)$.
- $\Rightarrow$ approximation algorithms, exponential algorithms.
- Lattice basis reduction.

Introduction (2)

- ... but lattice problems are hard:
  - SVP NP-hard under randomized reductions (even with almost poly approx factor);
  - CVP NP-hard (same) ;
  - BDD NP-hard for $r = c \cdot \lambda_1(L)$.
- $\Rightarrow$ approximation algorithms, exponential algorithms.
- Lattice basis reduction.

Introduction (2)

- ... but lattice problems are hard:
  - SVP NP-hard under randomized reductions (even with almost poly approx factor);
  - CVP NP-hard (same) ;
  - BDD NP-hard for $r = c \cdot \lambda_1(L)$.
- $\Rightarrow$ approximation algorithms, exponential algorithms.
- Lattice basis reduction.

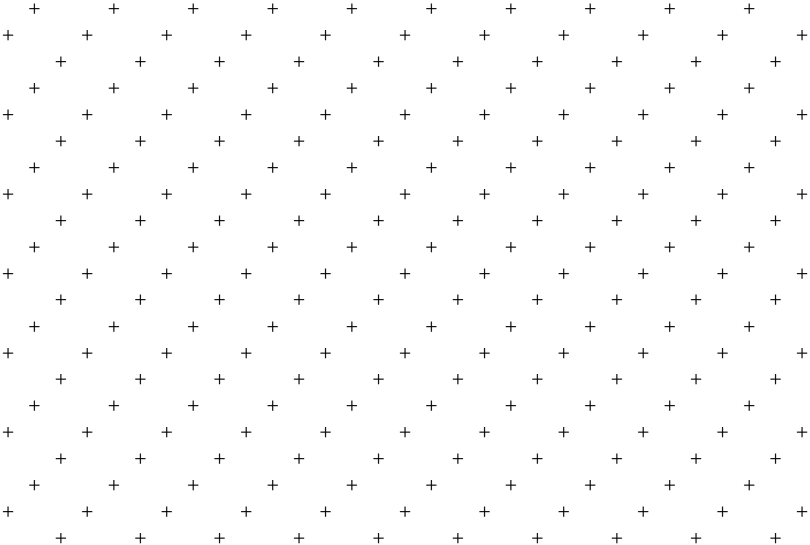
Introduction (2)

- ... but lattice problems are hard:
  - SVP NP-hard under randomized reductions (even with almost poly approx factor);
  - CVP NP-hard (same) ;
  - BDD NP-hard for $r = c \cdot \lambda_1(L)$.
- $\Rightarrow$ approximation algorithms, exponential algorithms.
- Lattice basis reduction.

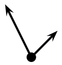
Introduction (2)

- ... but lattice problems are hard:
  - SVP NP-hard under randomized reductions (even with almost poly approx factor);
  - CVP NP-hard (same) ;
  - BDD NP-hard for $r = c \cdot \lambda_1(L)$.
- $\Rightarrow$ approximation algorithms, exponential algorithms.
- Lattice basis reduction.

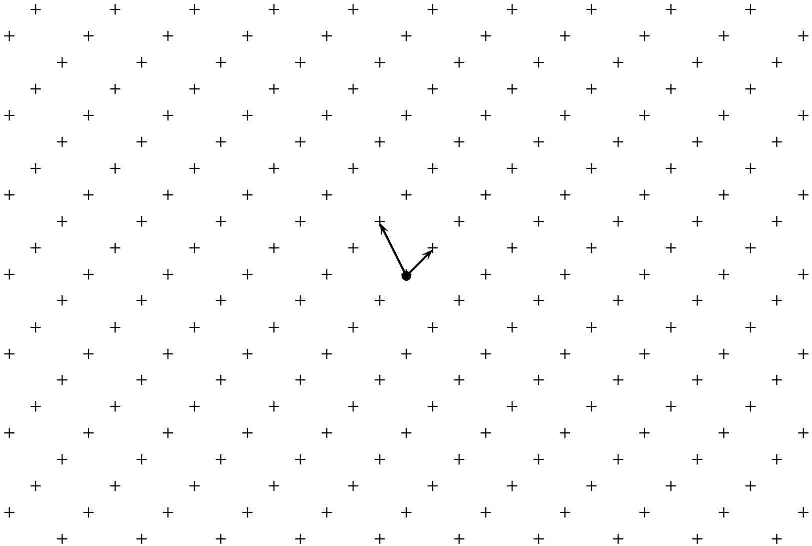
Introduction



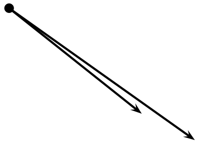
Introduction



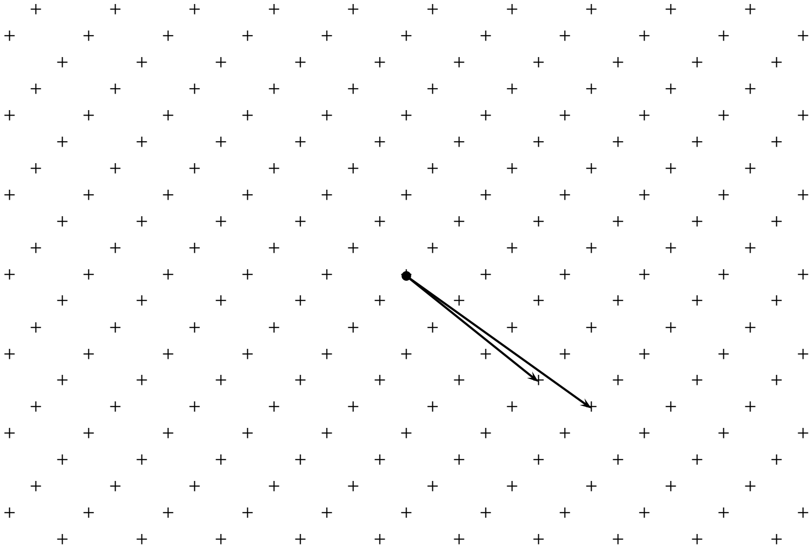
Introduction



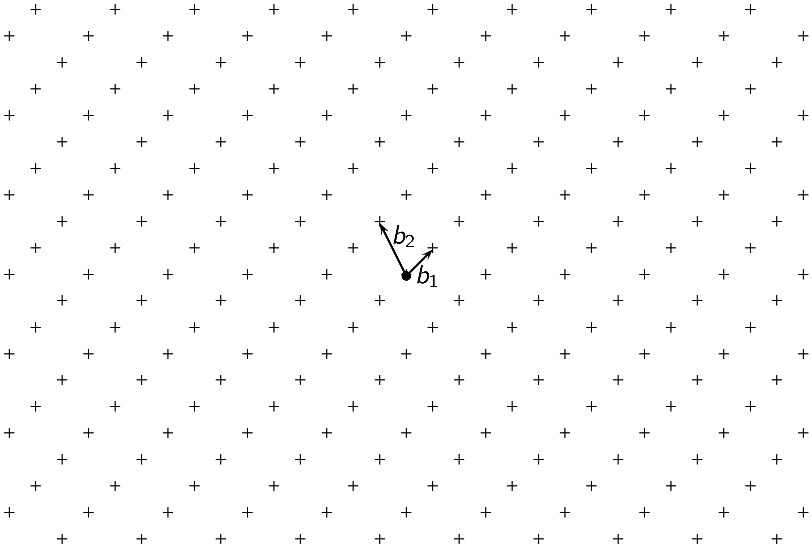
Reseaux euclidiens – bonnes et mauvaises bases



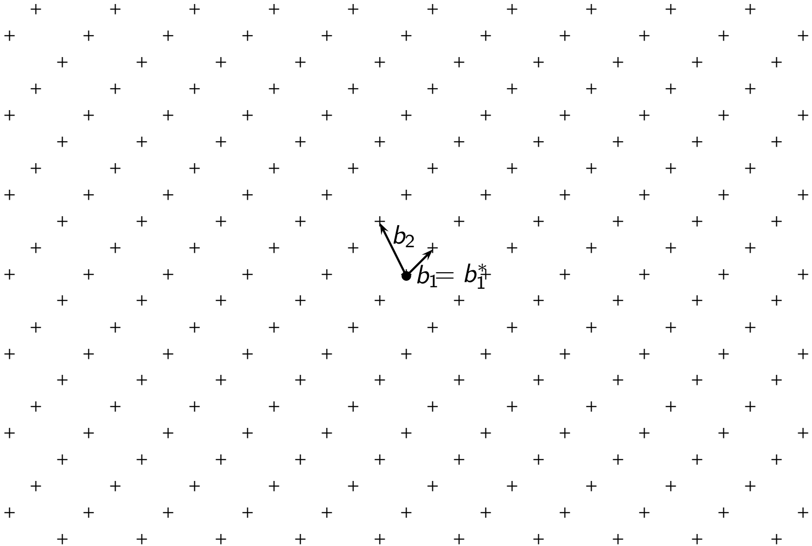
Reseaux euclidiens – bonnes et mauvaises bases



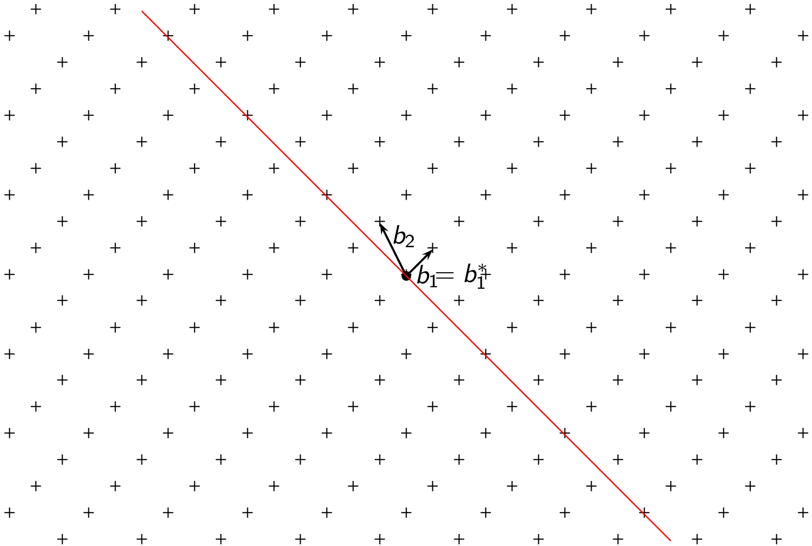
Lattices – good and bad bases



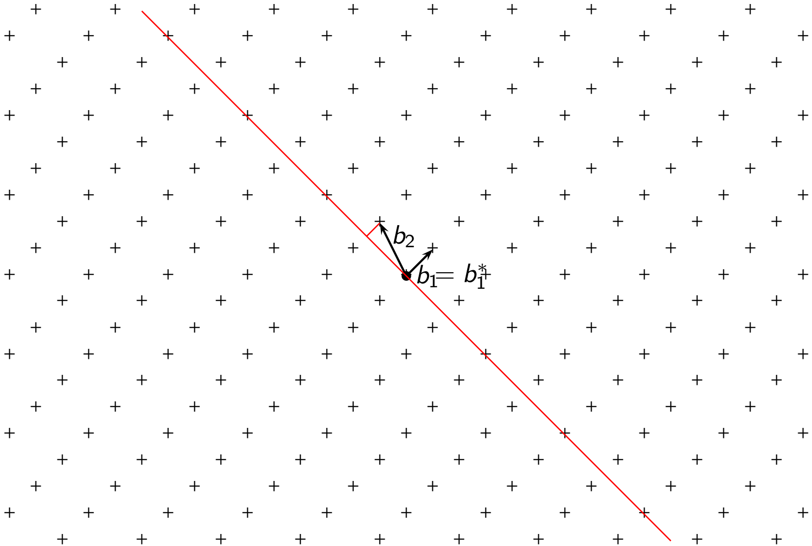
Lattices – good and bad bases



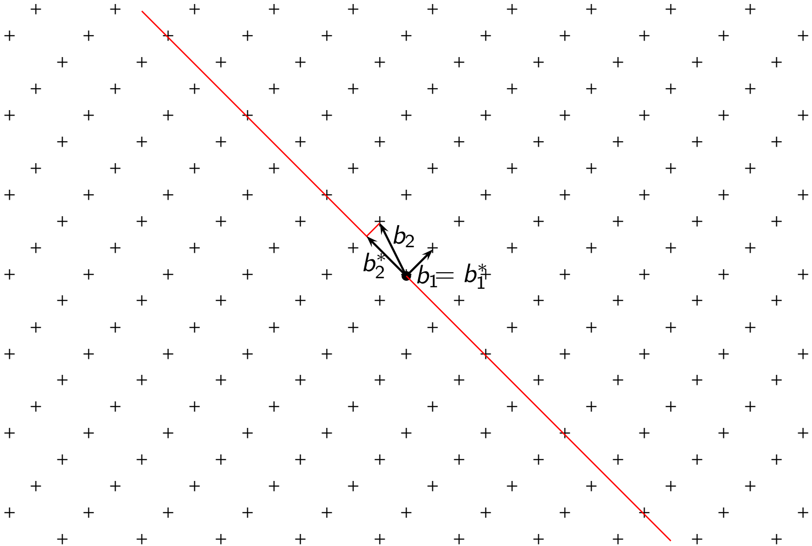
Lattices – good and bad bases



Lattices – good and bad bases

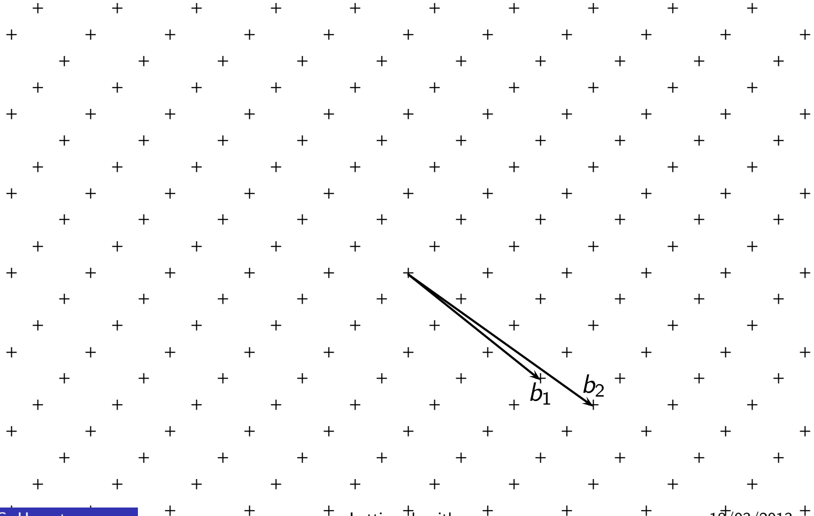


Lattices – good and bad bases



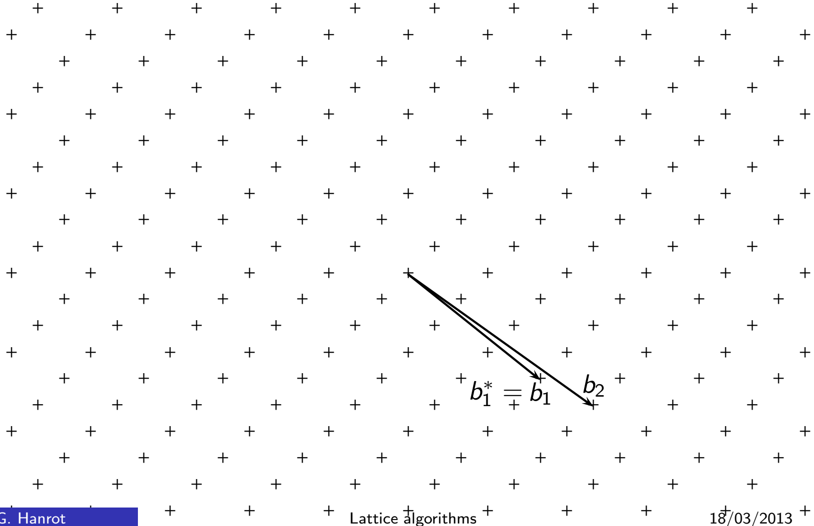
Lattices – good and bad bases

From a quantitative point of view:



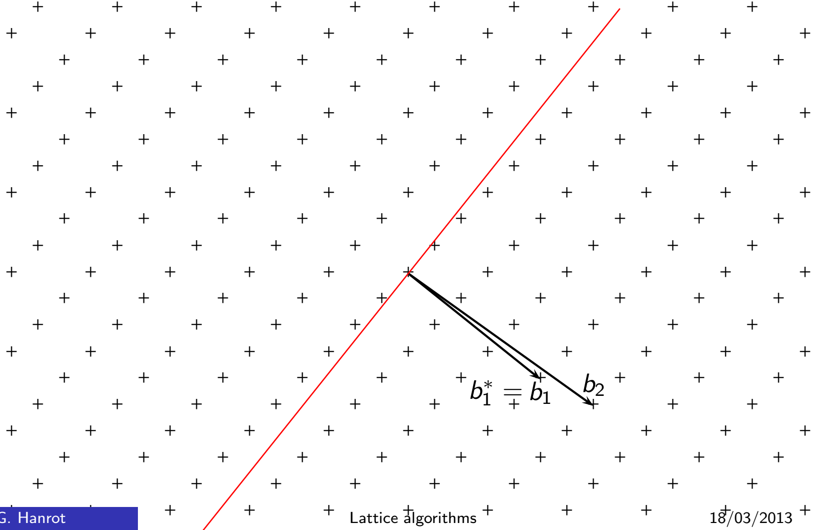
Lattices – good and bad bases

From a quantitative point of view:



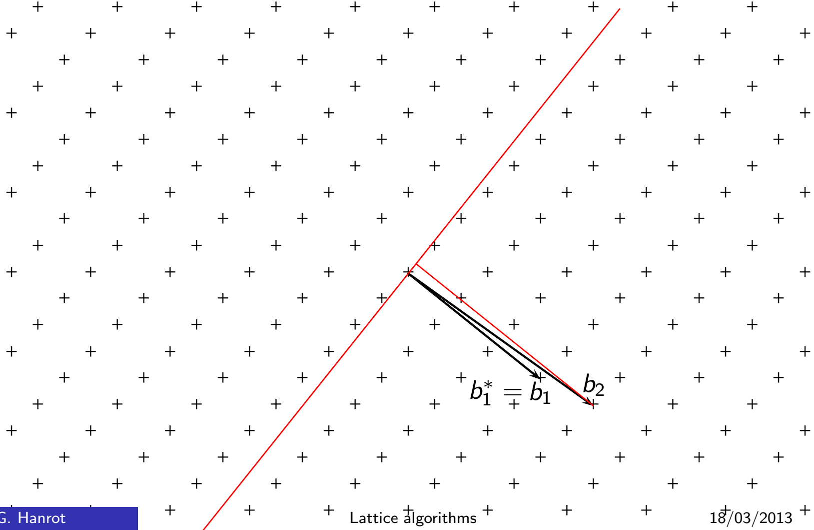
Lattices – good and bad bases

From a quantitative point of view:



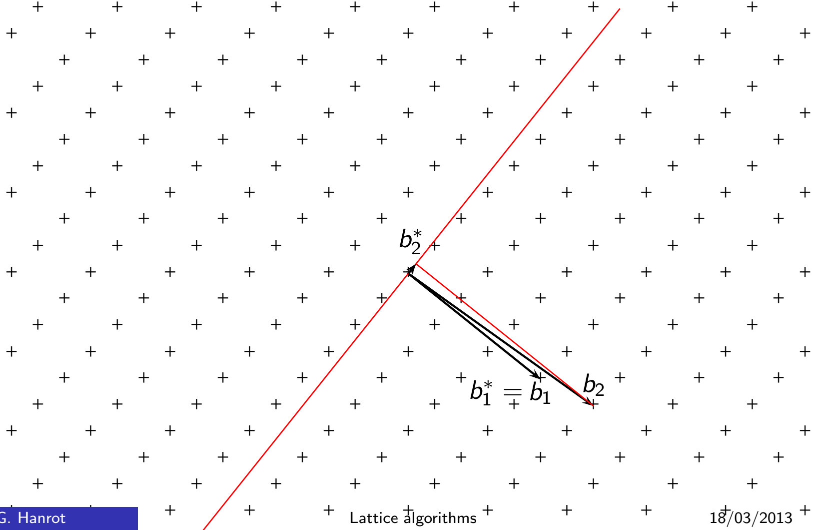
Lattices – good and bad bases

From a quantitative point of view:



Lattices – good and bad bases

From a quantitative point of view:



I. Lattice basis reduction

I. 1. Generalities on lattice basis reduction

Lattice basis reduction - Gram-Schmidt process

- $E_i := < b_1, \ldots, b_{i-1} >$, $\pi_{E_i^\perp}$.
- Put $b_i^* = b_i - \sum_{j<i} \mu_{ij} b_j^* =: \pi_{E_i^\perp}(b_i)$.
- $\mu_{ij} := (b_i, b_j^*)/\|b_j^*\|^2$.

Lattice basis reduction - Gram-Schmidt process

- $E_i := < b_1, \ldots, b_{i-1} >$, $\pi_{E_i^\perp}$.
- Put $b_i^* = b_i - \sum_{j<i} \mu_{ij} b_j^* =: \pi_{E_i^\perp}(b_i)$.
- $\mu_{ij} := (b_i, b_j^*)/\|b_j^*\|^2$.

Lattice basis reduction - Gram-Schmidt process

- $E_i := < b_1, \dots, b_{i-1} >$, $\pi_{E_i^\perp}$.
- Put $b_i^* = b_i - \sum_{j<i} \mu_{ij} b_j^* =: \pi_{E_i^\perp}(b_i)$.
- $\mu_{ij} := (b_i, b_j^*)/\|b_j^*\|^2$.

Lattice basis reduction

- What is a good basis?
- Geometrically: almost orthogonal;
  - Small orthogonality defect:

$$OD := \frac{\prod_{i=1}^{n} \|b_i\|}{\prod_{i=1}^{n} \|b_i^*\|} = \frac{\prod_{i=1}^{n} \|b_i\|}{\det L} \geq 1.$$

  - $\|b_i^*\|$ decreases slowly with i.
- Algorithmically: short vectors;
  - Length defect: $LD_i := \frac{\|b_i\|}{\lambda_i(L)}$.
  - Hermite factor: $HF := \|b_1\|/(\det L)^{1/n}$;
  - SVP approximation factor: $\|b_1\|/\lambda_1(L)$
- Algebraically, $U \in GL_n(\mathbb{Z})$ st. MU is "small".
- $\approx$ preprocessing of the lattice.

Lattice basis reduction

- What is a good basis?
- Geometrically: almost orthogonal;
  - Small orthogonality defect:

$$OD := \frac{\prod_{i=1}^{n} \|b_i\|}{\prod_{i=1}^{n} \|b_i^*\|} = \frac{\prod_{i=1}^{n} \|b_i\|}{\det L} \geq 1.$$

  - $\|b_i^*\|$ decreases slowly with i.
- Algorithmically: short vectors;
  - Length defect: $LD_i := \frac{\|b_i\|}{\lambda_i(L)}$.
  - Hermite factor: $HF := \|b_1\|/(\det L)^{1/n}$;
  - SVP approximation factor: $\|b_1\|/\lambda_1(L)$
- Algebraically, $U \in GL_n(\mathbb{Z})$ st. MU is "small".
- $\approx$ preprocessing of the lattice.

Lattice basis reduction

- What is a good basis?
- Geometrically: almost orthogonal;
  - Small orthogonality defect:

$$OD := \frac{\prod_{i=1}^n \|b_i\|}{\prod_{i=1}^n \|b_i^*\|} = \frac{\prod_{i=1}^n \|b_i\|}{\det L} \geq 1.$$

  - $\|b_i^*\|$ decreases slowly with i.
- Algorithmically: short vectors;
  - Length defect: $LD_i := \frac{\|b_i\|}{\lambda_i(L)}$.
  - Hermite factor: $HF := \|b_1\|/(\det L)^{1/n}$;
  - SVP approximation factor: $\|b_1\|/\lambda_1(L)$
- Algebraically, $U \in GL_n(\mathbb{Z})$ st. MU is "small".
- $\approx$ preprocessing of the lattice.

Lattice basis reduction

- What is a good basis?
- Geometrically: almost orthogonal;
  - Small orthogonality defect:

$$OD := \frac{\prod_{i=1}^{n} \|b_i\|}{\prod_{i=1}^{n} \|b_i^*\|} = \frac{\prod_{i=1}^{n} \|b_i\|}{\det L} \geq 1.$$

  - $\|b_i^*\|$ decreases slowly with i.
- Algorithmically: short vectors;
  - Length defect: $LD_i := \frac{\|b_i\|}{\lambda_i(L)}$.
  - Hermite factor: $HF := \|b_1\|/(\det L)^{1/n}$;
  - SVP approximation factor: $\|b_1\|/\lambda_1(L)$
- Algebraically, $U \in GL_n(\mathbb{Z})$ st. MU is "small".
- $\approx$ preprocessing of the lattice.

Lattice basis reduction

- What is a good basis?
- Geometrically: almost orthogonal;
  - Small orthogonality defect:

$$OD := \frac{\prod_{i=1}^n \|b_i\|}{\prod_{i=1}^n \|b_i^*\|} = \frac{\prod_{i=1}^n \|b_i\|}{\det L} \geq 1.$$

  - $\|b_i^*\|$ decreases slowly with i.
- Algorithmically: short vectors;
  - Length defect: $LD_i := \frac{\|b_i\|}{\lambda_i(L)}$.
  - Hermite factor: $HF := \|b_1\|/(\det L)^{1/n}$;
  - SVP approximation factor: $\|b_1\|/\lambda_1(L)$
- Algebraically, $U \in GL_n(\mathbb{Z})$ st. MU is "small".
- $\approx$ preprocessing of the lattice.

Lattice basis reduction

- What is a good basis?
- Geometrically: almost orthogonal;
  - Small orthogonality defect:

$$OD := \frac{\prod_{i=1}^n \|b_i\|}{\prod_{i=1}^n \|b_i^*\|} = \frac{\prod_{i=1}^n \|b_i\|}{\det L} \geq 1.$$

  - $\|b_i^*\|$ decreases slowly with i.
- Algorithmically: short vectors;
  - Length defect: $LD_i := \frac{\|b_i\|}{\lambda_i(L)}$.
  - Hermite factor: $HF := \|b_1\|/(\det L)^{1/n}$;
  - SVP approximation factor: $\|b_1\|/\lambda_1(L)$
- Algebraically, $U \in GL_n(\mathbb{Z})$ st. MU is "small".
- $\approx$ preprocessing of the lattice.

Lattice basis reduction

- What is a good basis?
- Geometrically: almost orthogonal;
  - Small orthogonality defect:

$$OD := \frac{\prod_{i=1}^{n} \|b_i\|}{\prod_{i=1}^{n} \|b_i^*\|} = \frac{\prod_{i=1}^{n} \|b_i\|}{\det L} \geq 1.$$

  - $\|b_i^*\|$ decreases slowly with i.
- Algorithmically: short vectors;
  - Length defect: $LD_i := \frac{\|b_i\|}{\lambda_i(L)}$.
  - Hermite factor: $HF := \|b_1\|/(\det L)^{1/n}$;
  - SVP approximation factor: $\|b_1\|/\lambda_1(L)$
- Algebraically, $U \in GL_n(\mathbb{Z})$ st. MU is "small".
- $\approx$ preprocessing of the lattice.

Lattice basis reduction – Example

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7038304916 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6175729875 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 9983710959 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 9161878375 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 9322349340 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 9870475629 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 6280159867 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2020850175 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 893775148 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 37842496080 \end{pmatrix}^T$$

Lattice basis reduction – Example

After LLL reduction

$$\begin{pmatrix} 2 & -1 & 1 & 5 & 1 & 2 & 3 & 1 & 0 & 5 \\ -3 & 6 & 0 & 0 & 8 & 5 & -3 & 1 & 3 & -3 \\ 1 & 2 & 0 & 1 & -4 & -5 & -7 & 5 & 4 & 6 \\ 2 & -6 & 4 & 3 & 0 & 0 & 1 & 3 & 5 & -3 \\ -4 & -1 & -5 & 2 & 2 & -2 & 5 & -2 & 0 & -3 \\ 7 & -3 & -4 & 1 & 1 & -2 & -4 & 1 & 4 & 3 \\ 0 & 3 & -4 & -3 & -3 & 7 & 1 & 4 & -4 & 0 \\ 0 & -4 & 4 & 1 & 4 & 3 & 0 & -7 & 2 & 2 \\ -3 & -1 & 3 & 3 & 4 & 1 & -5 & 2 & -6 & -2 \\ 3 & 0 & -4 & 7 & 0 & 0 & 0 & -1 & -2 & -7 \end{pmatrix}$$

Lattice basis reduction

Relationship "orthogonal $\Leftrightarrow$ short" through

- Minkowski's second thm :

$$\det L \leq \prod_{i=1}^{n} \lambda_i(L) \leq \sqrt{n}^n \det L,$$

since $OD / \prod LD_i = \prod_{i=1}^{n} \lambda_i(L) / \det L$.

- Recall $\lambda_1(L) \geq \min_i \|b_i^*\|$;
- $HF^n = \prod_{i=1}^{n} \|b_1\| / \|b_i^*\|$.

Reduction notion: quantitative version of all these.

Lattice basis reduction

Relationship "orthogonal $\Leftrightarrow$ short" through

- Minkowski's second thm :

$$\det L \leq \prod_{i=1}^{n} \lambda_i(L) \leq \sqrt{n}^n \det L,$$

since $OD / \prod LD_i = \prod_{i=1}^{n} \lambda_i(L) / \det L$.

- Recall $\lambda_1(L) \geq \min_i \|b_i^*\|$;
- $HF^n = \prod_{i=1}^{n} \|b_1\| / \|b_i^*\|$.

Reduction notion: quantitative version of all these.

Lattice basis reduction

Relationship "orthogonal $\Leftrightarrow$ short" through

- Minkowski's second thm :

$$\det L \leq \prod_{i=1}^{n} \lambda_i(L) \leq \sqrt{n}^n \det L,$$

since $OD/\prod LD_i = \prod_{i=1}^{n} \lambda_i(L)/\det L$.

- Recall $\lambda_1(L) \geq \min_i \|b_i^*\|$;
- $HF^n = \prod_{i=1}^{n} \|b_1\|/\|b_i^*\|$.

Reduction notion: quantitative version of all these.

Lattice basis reduction

Relationship "orthogonal $\Leftrightarrow$ short" through

- Minkowski's second thm :

$$\det L \leq \prod_{i=1}^{n} \lambda_i(L) \leq \sqrt{n}^n \det L,$$

  since $OD/\prod LD_i = \prod_{i=1}^n \lambda_i(L)/\det L$.
- Recall $\lambda_1(L) \geq \min_i \|b_i^*\|$;
- $HF^n = \prod_{i=1}^n \|b_1\|/\|b_i^*\|$.

Reduction notion: quantitative version of all these.

Lattice basis reduction

Relationship "orthogonal $\Leftrightarrow$ short" through

- Minkowski's second thm :

$$\det L \leq \prod_{i=1}^{n} \lambda_i(L) \leq \sqrt{n}^n \det L,$$

  since $OD / \prod LD_i = \prod_{i=1}^{n} \lambda_i(L) / \det L$.

- Recall $\lambda_1(L) \geq \min_i \|b_i^*\|$;
- $HF^n = \prod_{i=1}^{n} \|b_1\| / \|b_i^*\|$.

Reduction notion: quantitative version of all these.

Lattice basis reduction – notations

Notations :

- L a n-dim. lattice of $\mathbb{R}^n$, basis $(b_1, \ldots, b_n)$;
- $(u, v) := \sum_{1 \le i \le n} u_i v_i$;
- $\mathrm{vol} L := \sqrt{\det((b_i, b_j))} = |\det(b_i)|$;
- $\|\cdot\|$ is the Euclidean norm, $\|x\| = (x, x)^{1/2}$;
- size of the basis $= \beta := \log \max_{1 \le i \le n} \|b_i\|$

Lattice basis reduction – notations

Notations :

- L a n-dim. lattice of $\mathbb{R}^n$, basis $(b_1, \ldots, b_n)$;
- $(u, v) := \sum_{1 \leq i \leq n} u_i v_i$;
- $\mathrm{vol} L := \sqrt{\det((b_i, b_j))} = |\det(b_i)|$;
- $\|\cdot\|$ is the Euclidean norm, $\|x\| = (x, x)^{1/2}$;
- size of the basis $= \beta := \log \max_{1 \leq i \leq n} \|b_i\|$

Lattice basis reduction – notations

Notations :

- L a n-dim. lattice of $\mathbb{R}^n$, basis $(b_1, \ldots, b_n)$;
- $(u, v) := \sum_{1 \leq i \leq n} u_i v_i$;
- $\mathrm{vol} L := \sqrt{\det((b_i, b_j))} = |\det(b_i)|$;
- $\|\cdot\|$ is the Euclidean norm, $\|x\| = (x, x)^{1/2}$;
- size of the basis $= \beta := \log \max_{1 \leq i \leq n} \|b_i\|$

Lattice basis reduction – notations

Notations :

- L a n-dim. lattice of $\mathbb{R}^n$, basis $(b_1, \ldots, b_n)$;
- $(u, v) := \sum_{1 \le i \le n} u_i v_i$;
- $\mathrm{vol} L := \sqrt{\det((b_i, b_j))} = |\det(b_i)|$;
- $\| \cdot \|$ is the Euclidean norm, $\|x\| = (x, x)^{1/2}$;
- size of the basis $= \beta := \log \max_{1 \le i \le n} \|b_i\|$

Lattice basis reduction – notations

Notations :

- L a n-dim. lattice of $\mathbb{R}^n$, basis $(b_1, \ldots, b_n)$;
- $(u, v) := \sum_{1 \leq i \leq n} u_i v_i$;
- $\mathrm{vol} L := \sqrt{\det((b_i, b_j))} = |\det(b_i)|$;
- $\| \cdot \|$ is the Euclidean norm, $\|x\| = (x, x)^{1/2}$;
- size of the basis $= \beta := \log \max_{1 \leq i \leq n} \|b_i\|$

Lattice basis reduction – general strategy

To make b_i small and orthogonal :

- $\|b_i\|^2 = \|b_i^*\|^2 + \|b_i - b_i^*\|^2$;
- Work on $\pi_{L_i^\perp}(b_i, \ldots, b_n)$ to improve $\|b_i^*\|$.
- Make second term small $\Rightarrow$ size reduction :
  - Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $b_i - b_i^*$, $b_i \leftarrow b_i - x$.
- $\approx$ (approx-)CVP.

General philosophy:

- size-reduction + reduce projected sublattices;
- ... propagate these reductions on the whole lattice.

Lattice basis reduction – general strategy

To make b_i small and orthogonal :

- $\|b_i\|^2 = \|b_i^*\|^2 + \|b_i - b_i^*\|^2$;
- Work on $\pi_{L_i^\perp}(b_i, \ldots, b_n)$ to improve $\|b_i^*\|$.
- Make second term small $\Rightarrow$ size reduction :
  - Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $b_i - b_i^*$, $b_i \leftarrow b_i - x$.
- $\approx$ (approx-)CVP.

General philosophy:

- size-reduction + reduce projected sublattices;
- ... propagate these reductions on the whole lattice.

Lattice basis reduction – general strategy

To make b_i small and orthogonal :

- $\|b_i\|^2 = \|b_i^*\|^2 + \|b_i - b_i^*\|^2$;
- Work on $\pi_{L_i^\perp}(b_i, \ldots, b_n)$ to improve $\|b_i^*\|$.
- Make second term small $\Rightarrow$ size reduction :
  - Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $b_i - b_i^*$, $b_i \leftarrow b_i - x$.
- $\approx$ (approx-)CVP.

General philosophy:

- size-reduction + reduce projected sublattices;
- ... propagate these reductions on the whole lattice.

Lattice basis reduction – general strategy

To make b_i small and orthogonal :

- $\|b_i\|^2 = \|b_i^*\|^2 + \|b_i - b_i^*\|^2$;
- Work on $\pi_{L_i^\perp}(b_i, \ldots, b_n)$ to improve $\|b_i^*\|$.
- Make second term small $\Rightarrow$ size reduction :
  - Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $b_i - b_i^*$, $b_i \leftarrow b_i - x$.
- $\approx$ (approx-)CVP.

General philosophy:

- size-reduction + reduce projected sublattices;
- ... propagate these reductions on the whole lattice.

Lattice basis reduction – general strategy

To make b_i small and orthogonal :

- $\|b_i\|^2 = \|b_i^*\|^2 + \|b_i - b_i^*\|^2$;
- Work on $\pi_{L_i^\perp}(b_i, \ldots, b_n)$ to improve $\|b_i^*\|$.
- Make second term small $\Rightarrow$ size reduction :
  - Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $b_i - b_i^*$, $b_i \leftarrow b_i - x$.
- $\approx$ (approx-)CVP.

General philosophy:

- size-reduction + reduce projected sublattices;
- ... propagate these reductions on the whole lattice.

Lattice basis reduction – general strategy

To make b_i small and orthogonal :

- $\|b_i\|^2 = \|b_i^*\|^2 + \|b_i - b_i^*\|^2$;
- Work on $\pi_{L_i^\perp}(b_i, \dots, b_n)$ to improve $\|b_i^*\|$.
- Make second term small $\Rightarrow$ size reduction :
  - Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $b_i - b_i^*$, $b_i \leftarrow b_i - x$.
- $\approx$ (approx-)CVP.

General philosophy:

- size-reduction + reduce projected sublattices;
- ... propagate these reductions on the whole lattice.

Lattice basis reduction – size reduction

Common feature to all lattice basis reduction algorithms.

- Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $t := b_i - b_i^*$;
- Want $\pi_{L_{i-1}^{\perp}}(t)$ close to $\mathbb{Z}b_{i-1}^*$, $\approx t_{i-1}b_{i-1}^*$
- and $t - t_{i-1}b_{i-1}$ close to $\mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-2}$
    - Repeat with $t - t_{i-1}b_{i-1}$ until $i = 0$.

Algorithm:

- For j from $i-1$ downto 1 do
    - $b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$
    - For k from 1 to $j-1$ do
    - $\mu_{ik} \leftarrow \mu_{ik} - x_j \mu_{jk}$

Lattice basis reduction – size reduction

Common feature to all lattice basis reduction algorithms.

- Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $t := b_i - b_i^*$;
- Want $\pi_{L_{i-1}^\perp}(t)$ close to $\mathbb{Z}b_{i-1}^*$, $\approx t_{i-1}b_{i-1}^*$
- and $t - t_{i-1}b_{i-1}$ close to $\mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-2}$
  - Repeat with $t - t_{i-1}b_{i-1}$ until $i = 0$.

Algorithm:

- For j from $i-1$ downto 1 do
  - $b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$
  - For k from 1 to $j-1$ do
  - $\mu_{ik} \leftarrow \mu_{ik} - x_j \mu_{jk}$

Lattice basis reduction – size reduction

Common feature to all lattice basis reduction algorithms.

- Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $t := b_i - b_i^*$;
- Want $\pi_{L_{i-1}^\perp}(t)$ close to $\mathbb{Z}b_{i-1}^*$, $\approx t_{i-1}b_{i-1}^*$
- and $t - t_{i-1}b_{i-1}$ close to $\mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-2}$
    - Repeat with $t - t_{i-1}b_{i-1}$ until $i = 0$.

Algorithm:

- For j from $i-1$ downto 1 do
    - $b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$
    - For k from 1 to $j-1$ do
    - $\mu_{ik} \leftarrow \mu_{ik} - x_j\mu_{jk}$

Lattice basis reduction – size reduction

Common feature to all lattice basis reduction algorithms.

- Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $t := b_i - b_i^*$;
- Want $\pi_{L_{i-1}^{\perp}}(t)$ close to $\mathbb{Z}b_{i-1}^*$, $\approx t_{i-1}b_{i-1}^*$
- and $t - t_{i-1}b_{i-1}$ close to $\mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-2}$
  - Repeat with $t - t_{i-1}b_{i-1}$ until $i = 0$.

Algorithm:

- For j from $i-1$ downto 1 do
  - $b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$
  - For k from 1 to $j-1$ do
  - $\mu_{ik} \leftarrow \mu_{ik} - x_j \mu_{jk}$

Lattice basis reduction – size reduction

Common feature to all lattice basis reduction algorithms.

- Find $x \in L_i := \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-1}$ close to $t := b_i - b_i^*$;
- Want $\pi_{L_{i-1}^\perp}(t)$ close to $\mathbb{Z}b_{i-1}^*$, $\approx t_{i-1}b_{i-1}^*$
- and $t - t_{i-1}b_{i-1}$ close to $\mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{i-2}$
    - Repeat with $t - t_{i-1}b_{i-1}$ until $i = 0$.

Algorithm:

- For j from $i-1$ downto 1 do
    - $b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$
    - For k from 1 to $j-1$ do
    - $\mu_{ik} \leftarrow \mu_{ik} - x_j\mu_{jk}$

Lattice basis reduction – a panorama

Strong notions

- Minkowski: b_i shortest possible;
- HKZ (Hermite-Korkine-Zolotarev):
  - b_1 shortest possible $= \lambda_1(L)$
  - size-reduced
  - $\pi_{L_1}(b_2, \ldots, b_n)$ HKZ reduced

(Very) costly notions, $2^{O(n)}$ for HKZ.

Lattice basis reduction – a panorama (2)

Weaker / cheaper : blockwise algorithms :

- Use k-dim. HKZ to reduce projections of sublattices;
- size-reduced;
- $\|b_i^*\| \geq \alpha_k^{1-i} \|b_1^*\|$.

Main examples:

- LLL: $\alpha_2 \approx 2/\sqrt{3}$;
- slide, BKZ: $\alpha_k \approx k^{1/k}$;
- Hermite factor $HF = \sqrt{\alpha_k}^{n-1}$;
- Approx-SVP $= \alpha_k^{n-1}$.

Polynomial time up to $k = O(\log n)$.

I. 2. Lattice basis reduction – algorithms

Lattice basis reduction – outline

- Case $d = 2$, Gauss' algorithm.
- HKZ
- Blockwise algorithms

Lattice basis reduction – outline

- Case $d = 2$, Gauss' algorithm.
- HKZ
- Blockwise algorithms

Lattice basis reduction – outline

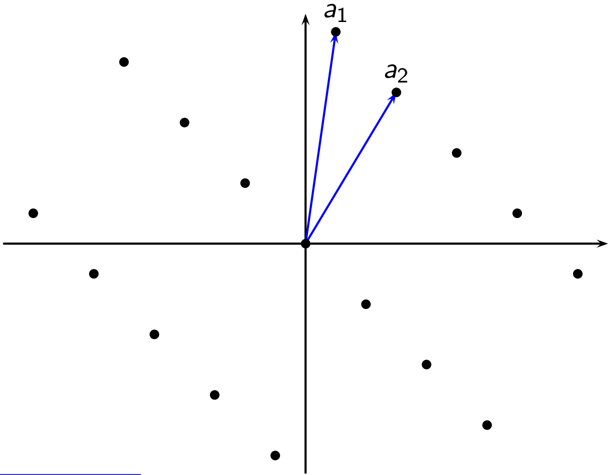
- Case $d = 2$, Gauss' algorithm.
- HKZ
- Blockwise algorithms

Lattice basis reduction – outline

- Case $d = 2$, Gauss' algorithm.
- HKZ
- Blockwise algorithms

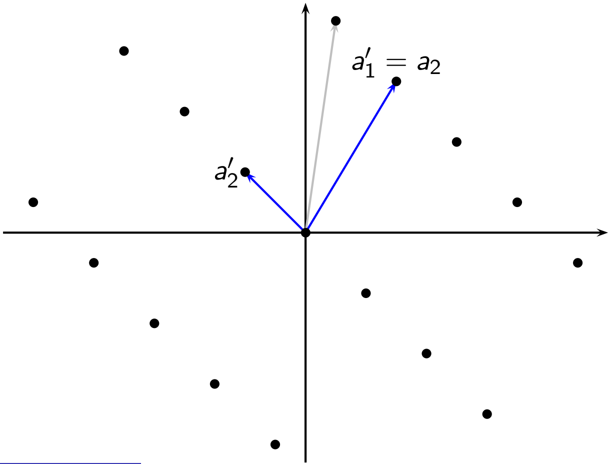
Lattice basis reduction – "optimal" algorithms

Case $d = 2$, Gauss' algorithm.
Size-reduce + swap.



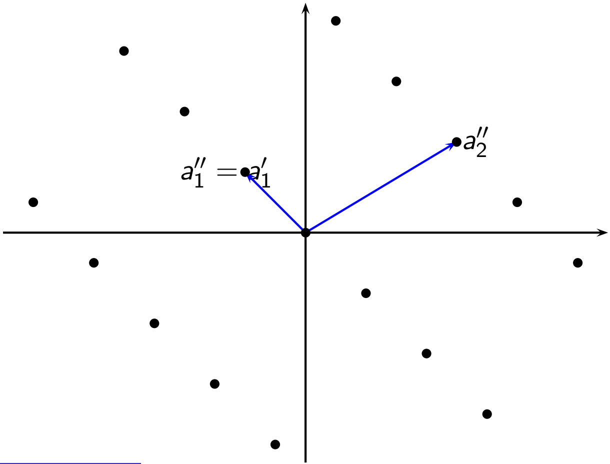
Lattice basis reduction – "optimal" algorithms

Case $d = 2$, Gauss' algorithm.
Size-reduce + swap.



Lattice basis reduction – "optimal" algorithms

Case $d = 2$, Gauss' algorithm.
Size-reduce + swap.



Lattice basis reduction – "optimal" algorithms

Analysis:

Theorem. Starting with (u, v), Gauss algorithm returns (b_1, b_2) st.

1. $\mathbb{Z}u + \mathbb{Z}v = \mathbb{Z}b_1 + \mathbb{Z}b_2$,
2. $\|b_1\| = \lambda_1(L)$, $\|b_2\| = \lambda_2(L)$
3. $\|b_2^*\| \geq \sqrt{3}/2\|b_1^*\|$
4. in time $O(\max(\|u\|, \|v\|)^2)$.

Proof (sketch).

- $\|u \pm v\|^2 \geq \|v\|^2 \Rightarrow |(u, v)| \leq \|u\|^2/2$.
- If $\|\alpha u + \beta v\|^2 < \|u\|^2$, we get

$$(\alpha^2 - |\alpha\beta| - 1)\|u\|^2 + \beta^2\|v\|^2 < 0$$

hence $(\alpha - \beta)^2 + |\alpha\beta| - 1 < 0$.

Lattice basis reduction – "optimal" algorithms

Analysis:

Theorem. Starting with (u, v), Gauss algorithm returns (b_1, b_2) st.

1. $\mathbb{Z}u + \mathbb{Z}v = \mathbb{Z}b_1 + \mathbb{Z}b_2$,
2. $\|b_1\| = \lambda_1(L)$, $\|b_2\| = \lambda_2(L)$
3. $\|b_2^*\| \geq \sqrt{3}/2\|b_1^*\|$
4. in time $O(\max(\|u\|, \|v\|)^2)$.

Proof (sketch).

- $\|u \pm v\|^2 \geq \|v\|^2 \Rightarrow |(u, v)| \leq \|u\|^2/2$.
- If $\|\alpha u + \beta v\|^2 < \|u\|^2$, we get

$$(\alpha^2 - |\alpha\beta| - 1)\|u\|^2 + \beta^2\|v\|^2 < 0$$

hence $(\alpha - \beta)^2 + |\alpha\beta| - 1 < 0$.

Lattice basis reduction – "optimal" algorithms

Analysis:

Theorem. Starting with (u, v), Gauss algorithm returns (b_1, b_2) st.

1. $\mathbb{Z}u + \mathbb{Z}v = \mathbb{Z}b_1 + \mathbb{Z}b_2$,
2. $\|b_1\| = \lambda_1(L)$, $\|b_2\| = \lambda_2(L)$
3. $\|b_2^*\| \geq \sqrt{3}/2\|b_1^*\|$
4. in time $O(\max(\|u\|, \|v\|)^2)$.

Proof (sketch).

- $\|u \pm v\|^2 \geq \|v\|^2 \Rightarrow |(u, v)| \leq \|u\|^2/2$.
- If $\|\alpha u + \beta v\|^2 < \|u\|^2$, we get

$$(\alpha^2 - |\alpha\beta| - 1)\|u\|^2 + \beta^2\|v\|^2 < 0$$

hence $(\alpha - \beta)^2 + |\alpha\beta| - 1 < 0$.

Lattice basis reduction – “optimal” algorithms

Analysis:

Theorem. Starting with (u, v), Gauss algorithm returns (b_1, b_2) st.

1. $\mathbb{Z}u + \mathbb{Z}v = \mathbb{Z}b_1 + \mathbb{Z}b_2$,
2. $\|b_1\| = \lambda_1(L)$, $\|b_2\| = \lambda_2(L)$
3. $\|b_2^*\| \geq \sqrt{3}/2\|b_1^*\|$
4. in time $O(\max(\|u\|, \|v\|)^2)$.

Proof (sketch).

- $\|u \pm v\|^2 \geq \|v\|^2 \Rightarrow |(u, v)| \leq \|u\|^2/2$.
- If $\|\alpha u + \beta v\|^2 < \|u\|^2$, we get

$$(\alpha^2 - |\alpha\beta| - 1)\|u\|^2 + \beta^2\|v\|^2 < 0$$

hence $(\alpha - \beta)^2 + |\alpha\beta| - 1 < 0$.

Lattice basis reduction – "optimal" algorithms

Analysis:

Theorem. Starting with (u, v), Gauss algorithm returns (b_1, b_2) st.

1. $\mathbb{Z}u + \mathbb{Z}v = \mathbb{Z}b_1 + \mathbb{Z}b_2$,
2. $\|b_1\| = \lambda_1(L)$, $\|b_2\| = \lambda_2(L)$
3. $\|b_2^*\| \geq \sqrt{3}/2\|b_1^*\|$
4. in time $O(\max(\|u\|, \|v\|)^2)$.

Proof (sketch).

- $\|u \pm v\|^2 \geq \|v\|^2 \Rightarrow |(u, v)| \leq \|u\|^2/2$.
- If $\|\alpha u + \beta v\|^2 < \|u\|^2$, we get

$$(\alpha^2 - |\alpha\beta| - 1)\|u\|^2 + \beta^2\|v\|^2 < 0$$

hence $(\alpha - \beta)^2 + |\alpha\beta| - 1 < 0$.

Lattice basis reduction – “optimal” algorithms

Analysis:

Theorem. Starting with (u, v), Gauss algorithm returns (b_1, b_2) st.

1. $\mathbb{Z}u + \mathbb{Z}v = \mathbb{Z}b_1 + \mathbb{Z}b_2$,
2. $\|b_1\| = \lambda_1(L)$, $\|b_2\| = \lambda_2(L)$
3. $\|b_2^*\| \geq \sqrt{3}/2\|b_1^*\|$
4. in time $O(\max(\|u\|, \|v\|)^2)$.

Proof (sketch).

- $\|u \pm v\|^2 \geq \|v\|^2 \Rightarrow |(u, v)| \leq \|u\|^2/2$.
- If $\|\alpha u + \beta v\|^2 < \|u\|^2$, we get

$$(\alpha^2 - |\alpha\beta| - 1)\|u\|^2 + \beta^2\|v\|^2 < 0$$

hence $(\alpha - \beta)^2 + |\alpha\beta| - 1 < 0$.

Lattice basis reduction – “optimal” algorithms

Complexity of Gauss' algorithm (sketch):

- Current step $\neq (u', v') \leftarrow (v \pm u, u)$;
- Otherwise, $\|u'\|^2 \leq \|v\|^2/3$
- $\Rightarrow$ logarithmic number of steps
- Cost of one step $O(\log \|v_i\|(\log \|u_i\|\|v_i\| - \log \|u_i\|^2 + 1))$
- Overall cost

$$\sum_i O(\log \|v_i\|(\log \|v_i\| - \log \|u_i\| + 1))$$

A quasi-linear time variant (similar to fast gcd) exist.

Lattice basis reduction – "optimal" algorithms

Complexity of Gauss' algorithm (sketch):

- Current step $\neq (u', v') \leftarrow (v \pm u, u)$;
- Otherwise, $\|u'\|^2 \leq \|v\|^2/3$
- $\Rightarrow$ logarithmic number of steps
- Cost of one step $O(\log \|v_i\|(\log \|u_i\|\|v_i\| - \log \|u_i\|^2 + 1))$
- Overall cost

$$\sum_i O(\log \|v_i\|(\log \|v_i\| - \log \|u_i\| + 1))$$

A quasi-linear time variant (similar to fast gcd) exist.

Lattice basis reduction – “optimal” algorithms

Complexity of Gauss' algorithm (sketch):

- Current step $\neq (u', v') \leftarrow (v \pm u, u)$;
- Otherwise, $\|u'\|^2 \leq \|v\|^2/3$
- $\Rightarrow$ logarithmic number of steps
- Cost of one step $O(\log \|v_i\|(\log \|u_i\|\|v_i\| - \log \|u_i\|^2 + 1))$
- Overall cost

$$\sum_i O(\log \|v_i\|(\log \|v_i\| - \log \|u_i\| + 1))$$

A quasi-linear time variant (similar to fast gcd) exist.

Lattice basis reduction – "optimal" algorithms

Complexity of Gauss' algorithm (sketch):

- Current step $\neq (u', v') \leftarrow (v \pm u, u)$;
- Otherwise, $\|u'\|^2 \leq \|v\|^2/3$
- $\Rightarrow$ logarithmic number of steps
- Cost of one step $O(\log \|v_i\|(\log \|u_i\|\|v_i\| - \log \|u_i\|^2 + 1))$
- Overall cost

$$\sum_i O(\log \|v_i\|(\log \|v_i\| - \log \|u_i\| + 1))$$

A quasi-linear time variant (similar to fast gcd) exist.

Lattice basis reduction – "optimal" algorithms

Summary of Gauss' algorithm:

1. Start with any (u, v) linearly independent;
2. Return (b_1, b_2) s.t. $\|b_2^*\| \geq \sqrt{3}/2\|b_1^*\|$
3. Complexity quadratic (quasi-linear)

NB. (2) is a worst case profile.

Lattice basis reduction – "optimal" algorithms

Summary of Gauss' algorithm:

1. Start with any (u, v) linearly independent;
2. Return (b_1, b_2) s.t. $\|b_2^*\| \geq \sqrt{3}/2\|b_1^*\|$
3. Complexity quadratic (quasi-linear)

NB. (2) is a worst case profile.

Lattice basis reduction – HKZ reduction

- $(b_1, \ldots, b_n)$ HKZ reduced :
  - $\|b_1\| = \lambda_1(L)$;
  - size-reduced;
  - $\pi_{L_1^\perp}(b_2, \ldots, b_n)$ again size-reduced.

Remarks:

- Need SVP oracle;
- Recursive definition $\Rightarrow$ Recursive algorithm;
- $d = 1$ is trivial.

Lattice basis reduction – HKZ reduction

- $(b_1, \ldots, b_n)$ HKZ reduced :
  - $\|b_1\| = \lambda_1(L)$;
  - size-reduced;
  - $\pi_{L_1^\perp}(b_2, \ldots, b_n)$ again size-reduced.

Remarks:

- Need SVP oracle;
- Recursive definition $\Rightarrow$ Recursive algorithm;
- $d = 1$ is trivial.

Lattice basis reduction – HKZ reduction

- $(b_1, \ldots, b_n)$ HKZ reduced :
  - $\|b_1\| = \lambda_1(L)$;
  - size-reduced;
  - $\pi_{L_1^\perp}(b_2, \ldots, b_n)$ again size-reduced.

Remarks:

- Need SVP oracle;
- Recursive definition $\Rightarrow$ Recursive algorithm;
- $d = 1$ is trivial.

Lattice basis reduction – HKZ reduction

- $(b_1, \ldots, b_n)$ HKZ reduced :
  - $\|b_1\| = \lambda_1(L)$;
  - size-reduced;
  - $\pi_{L_1^\perp}(b_2, \ldots, b_n)$ again size-reduced.

Remarks:

- Need SVP oracle;
- Recursive definition $\Rightarrow$ Recursive algorithm;
- $d = 1$ is trivial.

Lattice basis reduction – HKZ reduction

- $(b_1, \ldots, b_n)$ HKZ reduced :
  - $\|b_1\| = \lambda_1(L)$;
  - size-reduced;
  - $\pi_{L_1^\perp}(b_2, \ldots, b_n)$ again size-reduced.

Remarks:

- Need SVP oracle;
- Recursive definition $\Rightarrow$ Recursive algorithm;
- $d = 1$ is trivial.

Lattice basis reduction – HKZ reduction

- $(b_1, \ldots, b_n)$ HKZ reduced :
  - $\|b_1\| = \lambda_1(L)$;
  - size-reduced;
  - $\pi_{L_1^\perp}(b_2, \ldots, b_n)$ again size-reduced.

Remarks:

- Need SVP oracle;
- Recursive definition $\Rightarrow$ Recursive algorithm;
- $d = 1$ is trivial.

Lattice basis reduction – HKZ reduction

- $(b_1, \dots, b_n)$ HKZ reduced :
  - $\|b_1\| = \lambda_1(L)$;
  - size-reduced;
  - $\pi_{L_1^\perp}(b_2, \dots, b_n)$ again size-reduced.

Remarks:

- Need SVP oracle;
- Recursive definition $\Rightarrow$ Recursive algorithm;
- $d = 1$ is trivial.

Lattice basis reduction – HKZ reduction

- Let $e_1 = SVP(L)$.
- $(e_1, e_2, \ldots, e_n) \leftarrow Basis(e_1, b_1, \ldots, b_n)$;
- get T with $\left(\pi_{L_1^\perp}(e_2), \ldots, \pi_{L_1^\perp}(e_n)\right) \cdot T$ HKZ-reduced;
- Size-reduce $e_1, (e_2, \ldots, e_n)T$.
- Return product of transformation matrices of step 2, 3, 4.

Lattice basis reduction – HKZ reduction

- Let $e_1 = SVP(L)$.
- $(e_1, e_2, \ldots, e_n) \leftarrow Basis(e_1, b_1, \ldots, b_n)$;
- get T with $\left(\pi_{L_1^\perp}(e_2), \ldots, \pi_{L_1^\perp}(e_n)\right) \cdot T$ HKZ-reduced;
- Size-reduce $e_1, (e_2, \ldots, e_n)T$.
- Return product of transformation matrices of step 2, 3, 4.

Lattice basis reduction – HKZ reduction

- Let $e_1 = SVP(L)$.
- $(e_1, e_2, \ldots, e_n) \leftarrow Basis(e_1, b_1, \ldots, b_n)$;
- get T with $\left(\pi_{L_1^\perp}(e_2), \ldots, \pi_{L_1^\perp}(e_n)\right) \cdot T$ HKZ-reduced;
- Size-reduce $e_1, (e_2, \ldots, e_n)T$.
- Return product of transformation matrices of step 2, 3, 4.

Lattice basis reduction – HKZ reduction

- Let $e_1 = SVP(L)$.
- $(e_1, e_2, \ldots, e_n) \leftarrow Basis(e_1, b_1, \ldots, b_n)$;
- get T with $\left(\pi_{L_1^\perp}(e_2), \ldots, \pi_{L_1^\perp}(e_n)\right) \cdot T$ HKZ-reduced;
- Size-reduce $e_1, (e_2, \ldots, e_n)T$.
- Return product of transformation matrices of step 2, 3, 4.

Lattice basis reduction – HKZ reduction

- Let $e_1 = SVP(L)$.
- $(e_1, e_2, \dots, e_n) \leftarrow Basis(e_1, b_1, \dots, b_n)$;
- get T with $\left(\pi_{L_1^\perp}(e_2), \dots, \pi_{L_1^\perp}(e_n)\right) \cdot T$ HKZ-reduced;
- Size-reduce $e_1, (e_2, \dots, e_n)T$.
- Return product of transformation matrices of step 2, 3, 4.

Lattice basis reduction – "optimal" algorithms

Analysis:

Theorem. Starting with $(e_1, \dots, e_n)$, HKZ algorithm returns $(b_1, \dots, b_n)$ st.

1. $\mathbb{Z}b_1 + \dots \mathbb{Z}b_n = \mathbb{Z}e_1 + \dots + \mathbb{Z}e_n$,
2. $\|b_1\| = \lambda_1(L)$
3. $\|b_1\|/\|b_n^*\| = O(n^{(\log n)/4+1/2})$
4. in time $2^{O(n)}$.

More generally, (3) implies:

$$\|b_i^*\| \approx \exp(\log^2(n-i+1)/4)\|b_n^*\|.$$

Proof: tedious manipulation of Minkowski's inequalities.

Lattice basis reduction – “optimal” algorithms

Analysis:

Theorem. Starting with $(e_1, \dots, e_n)$, HKZ algorithm returns $(b_1, \dots, b_n)$ st.

1. $\mathbb{Z}b_1 + \dots \mathbb{Z}b_n = \mathbb{Z}e_1 + \dots + \mathbb{Z}e_n$,
2. $\|b_1\| = \lambda_1(L)$
3. $\|b_1\|/\|b_n^*\| = O(n^{(\log n)/4+1/2})$
4. in time $2^{O(n)}$.

More generally, (3) implies:

$$\|b_i^*\| \approx \exp(\log^2(n-i+1)/4)\|b_n^*\|.$$

Proof: tedious manipulation of Minkowski's inequalities.

Lattice basis reduction – "optimal" algorithms

Analysis:

Theorem. Starting with $(e_1, \ldots, e_n)$, HKZ algorithm returns $(b_1, \ldots, b_n)$ st.

1. $\mathbb{Z}b_1 + \ldots \mathbb{Z}b_n = \mathbb{Z}e_1 + \cdots + \mathbb{Z}e_n$,
2. $\|b_1\| = \lambda_1(L)$
3. $\|b_1\|/\|b_n^*\| = O(n^{(\log n)/4+1/2})$
4. in time $2^{O(n)}$.

More generally, (3) implies:

$$\|b_i^*\| \approx \exp(\log^2(n-i+1)/4)\|b_n^*\|.$$

Proof: tedious manipulation of Minkowski's inequalities.

Lattice basis reduction – "optimal" algorithms

Analysis:

Theorem. Starting with $(e_1, \ldots, e_n)$, HKZ algorithm returns $(b_1, \ldots, b_n)$ st.

1. $\mathbb{Z}b_1 + \ldots \mathbb{Z}b_n = \mathbb{Z}e_1 + \cdots + \mathbb{Z}e_n$,
2. $\|b_1\| = \lambda_1(L)$
3. $\|b_1\|/\|b_n^*\| = O(n^{(\log n)/4+1/2})$
4. in time $2^{O(n)}$.

More generally, (3) implies:

$$\|b_i^*\| \approx \exp(\log^2(n-i+1)/4)\|b_n^*\|.$$

Proof: tedious manipulation of Minkowski's inequalities.

Lattice basis reduction – blockwise algorithms overview

General strategy :

- Use a “dim k”-oracle : Gauss ($k = 2$), or HKZ (bounded k, or $k \approx \log n$);
- Use dim k-oracle over $\pi_{L_j^\perp}(b_{j+1}, \dots, b_{j+k})$, for some values of j;
- Choosing j: adaptive versions / non-adaptive ones
- Halting criterion = no room for foreseeable improvement.
- ... hence $\pi_{L_j^\perp}(b_{j+1}, \dots, b_{j+k})$ almost HKZ-reduced for all j
- ... hence bounds on $\|b_i^*\|$.

Lattice basis reduction – blockwise algorithms overview

General strategy :

- Use a "dim k"-oracle : Gauss ($k = 2$), or HKZ (bounded k, or $k \approx \log n$);
- Use dim k-oracle over $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$, for some values of j;
- Choosing j: adaptive versions / non-adaptive ones
- Halting criterion = no room for foreseeable improvement.
- ... hence $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$ almost HKZ-reduced for all j
- ... hence bounds on $\|b_i^*\|$.

Lattice basis reduction – blockwise algorithms overview

General strategy :

- Use a "dim k"-oracle : Gauss ($k = 2$), or HKZ (bounded k, or $k \approx \log n$);
- Use dim k-oracle over $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$, for some values of j;
- Choosing j: adaptive versions / non-adaptive ones
- Halting criterion = no room for foreseeable improvement.
- ... hence $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$ almost HKZ-reduced for all j
- ... hence bounds on $\|b_i^*\|$.

Lattice basis reduction – blockwise algorithms overview

General strategy :

- Use a "dim k"-oracle : Gauss ($k = 2$), or HKZ (bounded k, or $k \approx \log n$);
- Use dim k-oracle over $\pi_{L_j^\perp}(b_{j+1}, \dots, b_{j+k})$, for some values of j;
- Choosing j: adaptive versions / non-adaptive ones
- Halting criterion = no room for foreseeable improvement.
- ... hence $\pi_{L_j^\perp}(b_{j+1}, \dots, b_{j+k})$ almost HKZ-reduced for all j
- ... hence bounds on $\|b_i^*\|$.

Lattice basis reduction – blockwise algorithms overview

General strategy :

- Use a "dim k"-oracle : Gauss ($k = 2$), or HKZ (bounded k, or $k \approx \log n$);
- Use dim k-oracle over $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$, for some values of j;
- Choosing j: adaptive versions / non-adaptive ones
- Halting criterion = no room for foreseeable improvement.
- ... hence $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$ almost HKZ-reduced for all j
- ... hence bounds on $\|b_i^*\|$.

Lattice basis reduction – blockwise algorithms overview

General strategy :

- Use a "dim k"-oracle : Gauss ($k = 2$), or HKZ (bounded k, or $k \approx \log n$);
- Use dim k-oracle over $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$, for some values of j;
- Choosing j: adaptive versions / non-adaptive ones
- Halting criterion = no room for foreseeable improvement.
- ... hence $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$ almost HKZ-reduced for all j
- ... hence bounds on $\|b_i^*\|$.

Lattice basis reduction – blockwise algorithms overview

General strategy :

- Use a "dim k"-oracle : Gauss ($k = 2$), or HKZ (bounded k, or $k \approx \log n$);
- Use dim k-oracle over $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$, for some values of j;
- Choosing j: adaptive versions / non-adaptive ones
- Halting criterion = no room for foreseeable improvement.
- ... hence $\pi_{L_j^\perp}(b_{j+1}, \ldots, b_{j+k})$ almost HKZ-reduced for all j
- ... hence bounds on $\|b_i^*\|$.

Lattice basis reduction – blockwise algorithms, history

- LLL (1982) : $k = 2$, Gauss, adaptive ($j =$ smallest useful position);
- Schnorr semi-2ℓ (1987) : $k = 2\ell$, HKZ, adaptive ($j =$ smallest useful multiple of ℓ) ;
- Schnorr-Euchner's BKZ (1994) : $k = \ell$, HKZ, non adaptive ($j =$ step $\bmod n$).
- Slide-reduction (Gama-Nguyen, 2008) : HKZ, adaptive, more involved.

Lattice basis reduction – blockwise algorithms, history

- LLL (1982) : $k = 2$, Gauss, adaptive ($j =$ smallest useful position);
- Schnorr semi-2ℓ (1987) : $k = 2\ell$, HKZ, adaptive ($j =$ smallest useful multiple of ℓ) ;
- Schnorr-Euchner's BKZ (1994) : $k = \ell$, HKZ, non adaptive ($j =$ step $\bmod n$).
- Slide-reduction (Gama-Nguyen, 2008) : HKZ, adaptive, more involved.

Lattice basis reduction – blockwise algorithms, history

- LLL (1982) : $k = 2$, Gauss, adaptive ($j =$ smallest useful position);
- Schnorr semi-2ℓ (1987) : $k = 2\ell$, HKZ, adaptive ($j =$ smallest useful multiple of ℓ) ;
- Schnorr-Euchner's BKZ (1994) : $k = \ell$, HKZ, non adaptive ($j =$ step $\bmod n$).
- Slide-reduction (Gama-Nguyen, 2008) : HKZ, adaptive, more involved.

Lattice basis reduction – blockwise algorithms, history

- LLL (1982) : $k = 2$, Gauss, adaptive (j = smallest useful position);
- Schnorr semi-2ℓ (1987) : $k = 2\ell$, HKZ, adaptive (j = smallest useful multiple of ℓ) ;
- Schnorr-Euchner's BKZ (1994) : $k = \ell$, HKZ, non adaptive (j = step $\bmod n$).
- Slide-reduction (Gama-Nguyen, 2008) : HKZ, adaptive, more involved.

Lattice basis reduction – LLL reduction

[LLL82] A basis $(b_1, \ldots, b_n)$ is δ-LLL reduced ($\delta < 1$) iff.

1. $(b_1, \ldots, b_n)$ is size-reduced
2. $\pi_{L_i^\perp}(b_i, b_{i+1})$ is almost Gauss-reduced.

(2) (with (1)) $\Leftrightarrow$ Lovasz' condition

$$\begin{aligned}
\delta \|b_i^*\|^2 &\leq \|\pi_{L_{i-1}^\perp}(b_{i+1})\|^2 \\
&= \|b_{i+1}^* + \mu_{(i+1)i} b_i^*\|^2 \\
&= \|b_{i+1}^*\|^2 + \mu_{(i+1)i}^2 \|b_i^*\|^2,
\end{aligned}$$

Weaker Siegel condition

$$(\delta - 1/4)\|b_i^*\|^2 \leq \|b_{i+1}^*\|^2.$$

Geometric decrease, $\alpha = \sqrt{\delta - 1/4} < \sqrt{3}/2$.

Lattice basis reduction – LLL reduction

[LLL82] A basis $(b_1, \ldots, b_n)$ is δ-LLL reduced ($\delta < 1$) iff.

1. $(b_1, \ldots, b_n)$ is size-reduced
2. $\pi_{L_i^\perp}(b_i, b_{i+1})$ is almost Gauss-reduced.

(2) (with (1)) $\Leftrightarrow$ Lovasz' condition

$$\begin{aligned} \delta \|b_i^*\|^2 &\le \|\pi_{L_{i-1}^\perp}(b_{i+1})\|^2 \\ &= \|b_{i+1}^* + \mu_{(i+1)i} b_i^*\|^2 \\ &= \|b_{i+1}^*\|^2 + \mu_{(i+1)i}^2 \|b_i^*\|^2, \end{aligned}$$

Weaker Siegel condition

$$(\delta - 1/4)\|b_i^*\|^2 \le \|b_{i+1}^*\|^2.$$

Geometric decrease, $\alpha = \sqrt{\delta - 1/4} < \sqrt{3}/2$.

Lattice basis reduction – LLL reduction

[LLL82] A basis $(b_1, \ldots, b_n)$ is δ-LLL reduced ($\delta < 1$) iff.

1. $(b_1, \ldots, b_n)$ is size-reduced
2. $\pi_{L_i^\perp}(b_i, b_{i+1})$ is almost Gauss-reduced.

(2) (with (1)) $\Leftrightarrow$ Lovasz' condition

$$\begin{aligned} \delta \|b_i^*\|^2 &\leq \|\pi_{L_{i-1}^\perp}(b_{i+1})\|^2 \\ &= \|b_{i+1}^* + \mu_{(i+1)i} b_i^*\|^2 \\ &= \|b_{i+1}^*\|^2 + \mu_{(i+1)i}^2 \|b_i^*\|^2, \end{aligned}$$

Weaker Siegel condition

$$(\delta - 1/4)\|b_i^*\|^2 \leq \|b_{i+1}^*\|^2.$$

Geometric decrease, $\alpha = \sqrt{\delta - 1/4} < \sqrt{3}/2$.

Lattice basis reduction – LLL reduction

[LLL82] A basis $(b_1, \ldots, b_n)$ is δ-LLL reduced ($\delta < 1$) iff.

1. $(b_1, \ldots, b_n)$ is size-reduced
2. $\pi_{L_i^\perp}(b_i, b_{i+1})$ is almost Gauss-reduced.

(2) (with (1)) $\Leftrightarrow$ Lovasz' condition

$$\begin{aligned}\delta\|b_i^*\|^2 &\leq \|\pi_{L_{i-1}^\perp}(b_{i+1})\|^2 \\ &= \|b_{i+1}^* + \mu_{(i+1)i} b_i^*\|^2 \\ &= \|b_{i+1}^*\|^2 + \mu_{(i+1)i}^2 \|b_i^*\|^2,\end{aligned}$$

Weaker Siegel condition

$$(\delta - 1/4)\|b_i^*\|^2 \leq \|b_{i+1}^*\|^2.$$

Geometric decrease, $\alpha = \sqrt{\delta - 1/4} < \sqrt{3}/2$.

Lattice basis reduction – LLL algorithm

Adaptive, oracle = Gauss.

1. $j \leftarrow 1$
2. While $j \leq n-1$ do
3. If $\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1})$ (almost) reduced, then $j \leftarrow j+1$
4. else Reduce $(\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1}))$, $j \leftarrow j-1$.
5. End do.
6. Size-reduce the basis.

Actual LLL :

- Reduce $\rightarrow$ one Gauss step, i.e. size-reduce + swap
- Full size-reduction of b_j wrt $b_1, \ldots, b_{j-1}$ after each Gauss step.

Lattice basis reduction – LLL algorithm

Adaptive, oracle = Gauss.

1. $j \leftarrow 1$
2. While $j \leq n-1$ do
3. If $\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1})$ (almost) reduced, then $j \leftarrow j+1$
4. else Reduce $(\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1}))$, $j \leftarrow j-1$.
5. End do.
6. Size-reduce the basis.

Actual LLL :

- Reduce $\rightarrow$ one Gauss step, i.e. size-reduce + swap
- Full size-reduction of b_j wrt $b_1, \ldots, b_{j-1}$ after each Gauss step.

Lattice basis reduction – LLL algorithm

Adaptive, oracle = Gauss.

1. $j \leftarrow 1$
2. While $j \leq n-1$ do
3. If $\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1})$ (almost) reduced, then $j \leftarrow j+1$
4. else Reduce $(\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1}))$, $j \leftarrow j-1$.
5. End do.
6. Size-reduce the basis.

Actual LLL :

- Reduce $\rightarrow$ one Gauss step, i.e. size-reduce + swap
- Full size-reduction of b_j wrt $b_1, \ldots, b_{j-1}$ after each Gauss step.

Lattice basis reduction – LLL algorithm

Adaptive, oracle = Gauss.

1. $j \leftarrow 1$
2. While $j \leq n-1$ do
3. If $\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1})$ (almost) reduced, then $j \leftarrow j+1$
4. else Reduce $(\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1}))$, $j \leftarrow j-1$.
5. End do.
6. Size-reduce the basis.

Actual LLL :

- Reduce $\rightarrow$ one Gauss step, i.e. size-reduce + swap
- Full size-reduction of b_j wrt $b_1, \ldots, b_{j-1}$ after each Gauss step.

Lattice basis reduction – LLL algorithm

Adaptive, oracle = Gauss.

1. $j \leftarrow 1$
2. While $j \leq n-1$ do
3. If $\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1})$ (almost) reduced, then $j \leftarrow j+1$
4. else Reduce $(\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1}))$, $j \leftarrow j-1$.
5. End do.
6. Size-reduce the basis.

Actual LLL :

- Reduce $\rightarrow$ one Gauss step, i.e. size-reduce + swap
- Full size-reduction of b_j wrt $b_1, \ldots, b_{j-1}$ after each Gauss step.

Lattice basis reduction – LLL algorithm

Adaptive, oracle = Gauss.

1. $j \leftarrow 1$
2. While $j \leq n-1$ do
3. If $\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1})$ (almost) reduced, then $j \leftarrow j+1$
4. else Reduce $(\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1}))$, $j \leftarrow j-1$.
5. End do.
6. Size-reduce the basis.

Actual LLL :

- Reduce $\rightarrow$ one Gauss step, i.e. size-reduce + swap
- Full size-reduction of b_j wrt $b_1, \ldots, b_{j-1}$ after each Gauss step.

Lattice basis reduction – LLL algorithm

Adaptive, oracle = Gauss.

1. $j \leftarrow 1$
2. While $j \leq n-1$ do
3. If $\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1})$ (almost) reduced, then $j \leftarrow j+1$
4. else Reduce $(\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1}))$, $j \leftarrow j-1$.
5. End do.
6. Size-reduce the basis.

Actual LLL :

- Reduce $\rightarrow$ one Gauss step, i.e. size-reduce + swap
- Full size-reduction of b_j wrt $b_1, \ldots, b_{j-1}$ after each Gauss step.

Lattice basis reduction – LLL algorithm

Adaptive, oracle = Gauss.

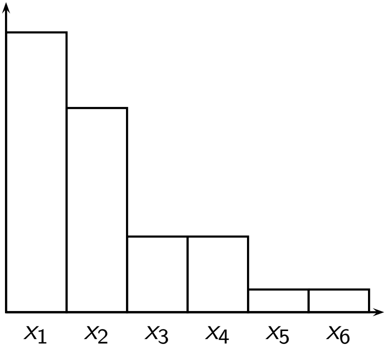
1. $j \leftarrow 1$
2. While $j \leq n-1$ do
3. If $\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1})$ (almost) reduced, then $j \leftarrow j+1$
4. else Reduce $(\pi_{L_{j-1}^{\perp}}(b_j), \pi_{L_{j-1}^{\perp}}(b_{j+1}))$, $j \leftarrow j-1$.
5. End do.
6. Size-reduce the basis.

Actual LLL :

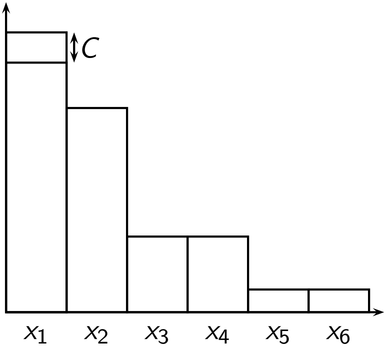
- Reduce $\rightarrow$ one Gauss step, i.e. size-reduce + swap
- Full size-reduction of b_j wrt $b_1, \ldots, b_{j-1}$ after each Gauss step.

Lattice basis reduction – A typical LLL execution

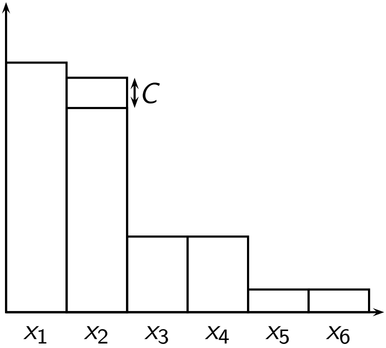
Lattice basis reduction – A typical LLL execution



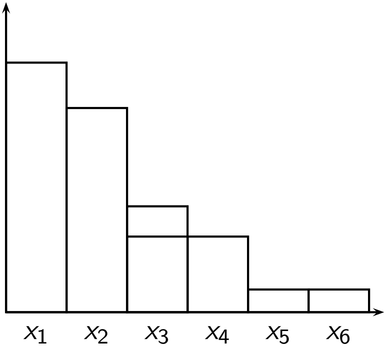
Lattice basis reduction – A typical LLL execution



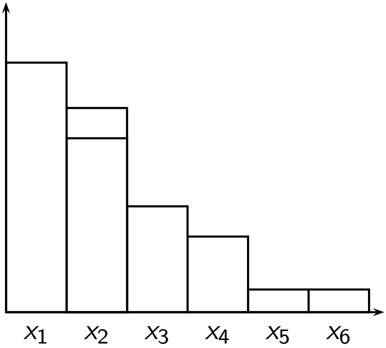
Lattice basis reduction – A typical LLL execution



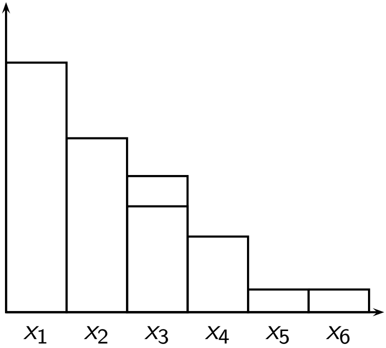
Lattice basis reduction – A typical LLL execution



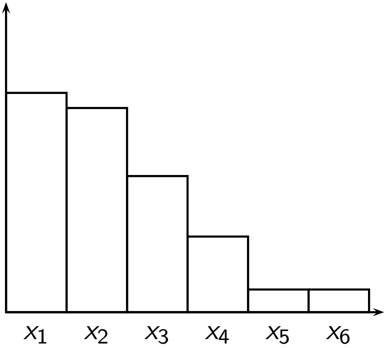
Lattice basis reduction – A typical LLL execution



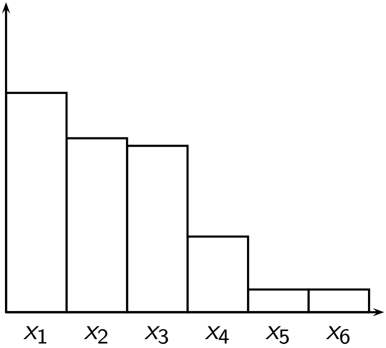
Lattice basis reduction – A typical LLL execution



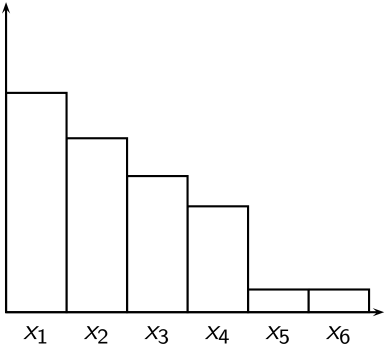
Lattice basis reduction – A typical LLL execution



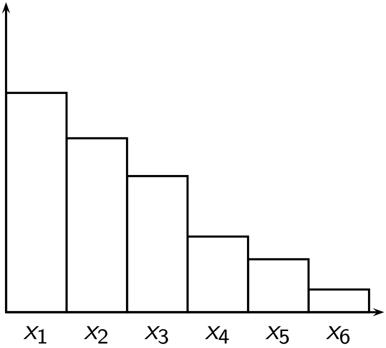
Lattice basis reduction – A typical LLL execution



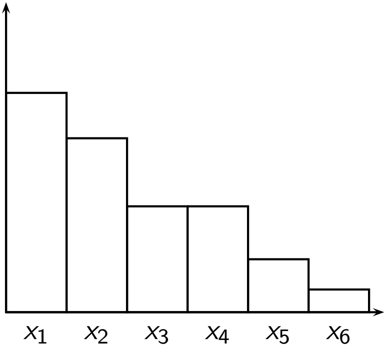
Lattice basis reduction – A typical LLL execution



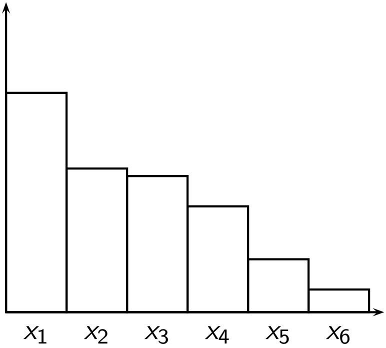
Lattice basis reduction – A typical LLL execution



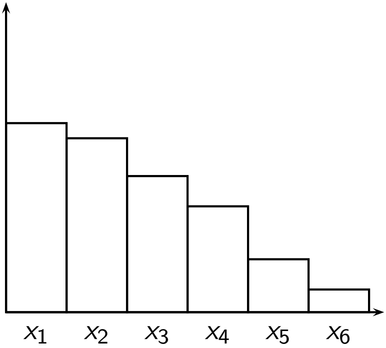
Lattice basis reduction – A typical LLL execution



Lattice basis reduction – A typical LLL execution



Lattice basis reduction – A typical LLL execution



Lattice basis reduction – LLL theorem

Theorem. On input $E = (e_1, \ldots, e_n)$ of size β, the δ-LLL algorithm computes a $\delta - LLL$ reduced basis $(b_1, \ldots, b_n)$ in time $O(n^2\beta)$ steps of cost $O(n^4(\beta + \log n)^2)$ such that

- $\|b_i^*\| \geq \sqrt{\delta - 1/4}\|b_{i-1}^*\|$,
- (approx-SVP) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/2}\,\lambda_1(L)$,
- (Hermite factor) $HF \leq (\delta - 1/4)^{(n-1)/4}$.

$\Rightarrow$ Can solve gap-SVP with gap $2^{O(n)}$.

Lattice basis reduction – LLL theorem

Theorem. On input $E = (e_1, \ldots, e_n)$ of size β, the δ-LLL algorithm computes a $\delta - LLL$ reduced basis $(b_1, \ldots, b_n)$ in time $O(n^2\beta)$ steps of cost $O(n^4(\beta + \log n)^2)$ such that

- $\|b_i^*\| \geq \sqrt{\delta - 1/4}\|b_{i-1}^*\|$,
- (approx-SVP) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/2}\, \lambda_1(L)$,
- (Hermite factor) $HF \leq (\delta - 1/4)^{(n-1)/4}$.

$\Rightarrow$ Can solve gap-SVP with gap $2^{O(n)}$.

Lattice basis reduction – LLL theorem

Theorem. On input $E = (e_1, \dots, e_n)$ of size β, the δ-LLL algorithm computes a $\delta - LLL$ reduced basis $(b_1, \dots, b_n)$ in time $O(n^2\beta)$ steps of cost $O(n^4(\beta + \log n)^2)$ such that

- $\|b_i^*\| \geq \sqrt{\delta - 1/4}\|b_{i-1}^*\|$,
- (approx-SVP) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/2}\,\lambda_1(L)$,
- (Hermite factor) $HF \leq (\delta - 1/4)^{(n-1)/4}$.

$\Rightarrow$ Can solve gap-SVP with gap $2^{O(n)}$.

Lattice basis reduction – LLL theorem

Proof. Want to prove:

1. $\|b_i^*\| \geq \sqrt{\delta - 1/4}\|b_{i-1}^*\|$
2. (approx-SVP) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/2}\,\lambda_1(L)$
3. (Hermite factor) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/4} \det L$

- First one follows by algorithm;
- 2. : $\lambda_1(L) \geq \min_{1 \leq i \leq n} \|b_i^*\|$;
- Third one : $\|b_1^*\| \leq \sqrt{\delta - 1/4}^{\,i-1}\|b_i^*\|$

Lattice basis reduction – LLL theorem

Proof. Want to prove:

1. $\|b_i^*\| \geq \sqrt{\delta - 1/4}\|b_{i-1}^*\|$
2. (approx-SVP) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/2}\,\lambda_1(L)$
3. (Hermite factor) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/4}\det L$

- First one follows by algorithm;
- 2. : $\lambda_1(L) \geq \min_{1 \leq i \leq n} \|b_i^*\|$;
- Third one : $\|b_1^*\| \leq \sqrt{\delta - 1/4}^{\,i-1}\|b_i^*\|$

Lattice basis reduction – LLL theorem

Proof. Want to prove:

1. $\|b_i^*\| \geq \sqrt{\delta - 1/4}\|b_{i-1}^*\|$
2. (approx-SVP) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/2} \lambda_1(L)$
3. (Hermite factor) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/4} \det L$

- First one follows by algorithm;
- 2. : $\lambda_1(L) \geq \min_{1\leq i\leq n} \|b_i^*\|$;
- Third one : $\|b_1^*\| \leq \sqrt{\delta - 1/4}^{i-1} \|b_i^*\|$

Lattice basis reduction – LLL theorem

Proof. Want to prove:

1. $\|b_i^*\| \geq \sqrt{\delta - 1/4}\|b_{i-1}^*\|$
2. (approx-SVP) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/2}\,\lambda_1(L)$
3. (Hermite factor) $\|b_1\| \leq (\delta - 1/4)^{(n-1)/4}\det L$

- First one follows by algorithm;
- 2. : $\lambda_1(L) \geq \min_{1\leq i\leq n}\|b_i^*\|$;
- Third one : $\|b_1^*\| \leq \sqrt{\delta - 1/4}^{\,i-1}\|b_i^*\|$

Lattice basis reduction – complexity

- When swap $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b'^*_i = \pi_{L^\perp_{i-1}}(b_i) = b^*_{i+1} + \mu_{i+1,i} b^*_i$
- Hence $\|b'^*_i\|^2 \le \delta \|b^*_i\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \dots, b_i)^2$
- One has

$$V' = V \frac{\det(b_1, \dots, b_{i-1}, b'_i)^2}{\det(b_1, \dots, b_{i-1}, b_i)^2}$$

- hence

$$V'/V = \|b'^*_i\|^2 / \|b^*_i\|^2 \le \delta$$

- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \le \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Lattice basis reduction – complexity

- When swap $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b'^*_i = \pi_{L^\perp_{i-1}}(b_i) = b^*_{i+1} + \mu_{i+1,i} b^*_i$
- Hence $\|b'^*_i\|^2 \le \delta \|b^*_i\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \ldots, b_i)^2$
- One has

$$V' = V \frac{\det(b_1, \ldots, b_{i-1}, b'_i)^2}{\det(b_1, \ldots, b_{i-1}, b_i)^2}$$

- hence

$$V'/V = \|b'^*_i\|^2 / \|b^*_i\|^2 \le \delta$$

- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \le \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Lattice basis reduction – complexity

- When swap $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b_i'^* = \pi_{L_{i-1}^\perp}(b_i) = b_{i+1}^* + \mu_{i+1,i} b_i^*$
- Hence $\|b_i'^*\|^2 \leq \delta \|b_i^*\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \dots, b_i)^2$
- One has
$$V' = V \frac{\det(b_1, \dots, b_{i-1}, b'_i)^2}{\det(b_1, \dots, b_{i-1}, b_i)^2}$$
- hence
$$V'/V = \|b_i'^*\|^2 / \|b_i^*\|^2 \leq \delta$$
- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \leq \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Lattice basis reduction – complexity

- When swap
  $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b'^*_i = \pi_{L^\perp_{i-1}}(b_i) = b^*_{i+1} + \mu_{i+1,i} b^*_i$
- Hence $\|b'^*_i\|^2 \leq \delta \|b^*_i\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \ldots, b_i)^2$
- One has

$$V' = V \frac{\det(b_1, \ldots, b_{i-1}, b'_i)^2}{\det(b_1, \ldots, b_{i-1}, b_i)^2}$$

- hence

$$V'/V = \|b'^*_i\|^2 / \|b^*_i\|^2 \leq \delta$$

- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \leq \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Lattice basis reduction – complexity

- When swap $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b'^*_i = \pi_{L^\perp_{i-1}}(b_i) = b^*_{i+1} + \mu_{i+1,i} b^*_i$
- Hence $\|b'^*_i\|^2 \leq \delta \|b^*_i\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \dots, b_i)^2$
- One has

$$V' = V \frac{\det(b_1, \dots, b_{i-1}, b'_i)^2}{\det(b_1, \dots, b_{i-1}, b_i)^2}$$

- hence

$$V'/V = \|b'^*_i\|^2 / \|b^*_i\|^2 \leq \delta$$

- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \leq \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Lattice basis reduction – complexity

- When swap $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b'^*_i = \pi_{L^\perp_{i-1}}(b_i) = b^*_{i+1} + \mu_{i+1,i} b^*_i$
- Hence $\|b'^*_i\|^2 \le \delta \|b^*_i\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \ldots, b_i)^2$
- One has

$$V' = V \frac{\det(b_1, \ldots, b_{i-1}, b'_i)^2}{\det(b_1, \ldots, b_{i-1}, b_i)^2}$$

- hence

$$V'/V = \|b'^*_i\|^2 / \|b^*_i\|^2 \le \delta$$

- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \le \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Lattice basis reduction – complexity

- When swap $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b'^*_i = \pi_{L^\perp_{i-1}}(b_i) = b^*_{i+1} + \mu_{i+1,i} b^*_i$
- Hence $\|b'^*_i\|^2 \le \delta \|b^*_i\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \ldots, b_i)^2$
- One has
$$V' = V \frac{\det(b_1, \ldots, b_{i-1}, b'_i)^2}{\det(b_1, \ldots, b_{i-1}, b_i)^2}$$
- hence
$$V'/V = \|b'^*_i\|^2 / \|b^*_i\|^2 \le \delta$$
- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \le \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Lattice basis reduction – complexity

- When swap $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b_i'^* = \pi_{L_{i-1}^\perp}(b_i) = b_{i+1}^* + \mu_{i+1,i} b_i^*$
- Hence $\|b_i'^*\|^2 \leq \delta \|b_i^*\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \ldots, b_i)^2$
- One has

$$V' = V \frac{\det(b_1, \ldots, b_{i-1}, b'_i)^2}{\det(b_1, \ldots, b_{i-1}, b_i)^2}$$

- hence

$$V'/V = \|b_i'^*\|^2 / \|b_i^*\|^2 \leq \delta$$

- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \leq \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Lattice basis reduction – complexity

- When swap $(b'_i, b'_{i+1}) \to (b_{i+1}, b_i) \Rightarrow b_i'^* = \pi_{L_{i-1}^\perp}(b_i) = b_{i+1}^* + \mu_{i+1,i} b_i^*$
- Hence $\|b_i'^*\|^2 \le \delta \|b_i^*\|^2$.
- Put $V = \prod_{i=1}^n \det(b_1, \ldots, b_i)^2$
- One has

$$V' = V \frac{\det(b_1, \ldots, b_{i-1}, b'_i)^2}{\det(b_1, \ldots, b_{i-1}, b_i)^2}$$

- hence

$$V'/V = \|b_i'^*\|^2 / \|b_i^*\|^2 \le \delta$$

- $V \in \mathbb{Z} \Rightarrow \log V_0 / \log \delta$ steps;
- But $V_0 \le \prod_{i=1}^n \|b_i\|^{2(n-i+1)}$ (Hadamard bound)
- Thus $O(n^2 \log \max \|b_i\|)$ steps.

Cost : $O(n^3)$ steps; cost of one step – close to GSO cost?

- Rational arithmetic $\Rightarrow$ control denominators;
- $b_i^* = b_i + \sum_{j<i} y_j b_j$.
- $(b_i, b_j) + \sum_{k<i} y_k (b_k, b_j) = 0 \quad (= (b_i^*, b_j))$
- ... hence $\mathbf{y} = -B_{i-1}^t b_i / \det B_{i-1}^t B_{i-1}$
- Denominators $= O(\beta'^{2n})$ (works for μ_{ij} too).
- where β' is the largest $\|b_i\|$ throughout the algorithm;
- can prove $\beta' = O(\log n + \beta)$
- Overall $O(n^2 \cdot n^2 (\log n + \beta)^2) = O(n^4 \beta^2)$ in the typical case $\beta > \log n$.

Total cost : $O(n^6 \beta^3)$.

Cost : $O(n^3)$ steps; cost of one step – close to GSO cost?

- Rational arithmetic $\Rightarrow$ control denominators;
- $b_i^* = b_i + \sum_{j<i} y_j b_j$.
- $(b_i, b_j) + \sum_{k<i} y_k (b_k, b_j) = 0 \quad (= (b_i^*, b_j))$
- ... hence $\mathbf{y} = -B_{i-1}^t b_i / \det B_{i-1}^t B_{i-1}$
- Denominators $= O(\beta'^{2n})$ (works for μ_{ij} too).
- where β' is the largest $\|b_i\|$ throughout the algorithm;
- can prove $\beta' = O(\log n + \beta)$
- Overall $O(n^2 \cdot n^2 (\log n + \beta)^2) = O(n^4 \beta^2)$ in the typical case $\beta > \log n$.

Total cost : $O(n^6 \beta^3)$.

Cost : $O(n^3)$ steps; cost of one step – close to GSO cost?

- Rational arithmetic $\Rightarrow$ control denominators;
- $b_i^* = b_i + \sum_{j<i} y_j b_j$.
- $(b_i, b_j) + \sum_{k<i} y_k (b_k, b_j) = 0 \quad (= (b_i^*, b_j))$
- ... hence $\mathbf{y} = -B_{i-1}^t b_i / \det B_{i-1}^t B_{i-1}$
- Denominators $= O(\beta'^{2n})$ (works for μ_{ij} too).
- where β' is the largest $\|b_i\|$ throughout the algorithm;
- can prove $\beta' = O(\log n + \beta)$
- Overall $O(n^2 \cdot n^2 (\log n + \beta)^2) = O(n^4 \beta^2)$ in the typical case $\beta > \log n$.

Total cost : $O(n^6 \beta^3)$.

Cost : $O(n^3)$ steps; cost of one step – close to GSO cost?

- Rational arithmetic $\Rightarrow$ control denominators;
- $b_i^* = b_i + \sum_{j<i} y_j b_j$.
- $(b_i, b_j) + \sum_{k<i} y_k (b_k, b_j) = 0 \quad (= (b_i^*, b_j))$
- ... hence $\mathbf{y} = -B_{i-1}^t b_i / \det B_{i-1}^t B_{i-1}$
- Denominators $= O(\beta'^{2n})$ (works for μ_{ij} too).
- where β' is the largest $\|b_i\|$ throughout the algorithm;
- can prove $\beta' = O(\log n + \beta)$
- Overall $O(n^2 \cdot n^2 (\log n + \beta)^2) = O(n^4 \beta^2)$ in the typical case $\beta > \log n$.

Total cost : $O(n^6 \beta^3)$.

Cost : $O(n^3)$ steps; cost of one step – close to GSO cost?

- Rational arithmetic $\Rightarrow$ control denominators;
- $b_i^* = b_i + \sum_{j<i} y_j b_j$.
- $(b_i, b_j) + \sum_{k<i} y_k (b_k, b_j) = 0 \quad (= (b_i^*, b_j))$
- ... hence $\mathbf{y} = -B_{i-1}^t b_i / \det B_{i-1}^t B_{i-1}$
- Denominators $= O(\beta'^{2n})$ (works for μ_{ij} too).
- where β' is the largest $\|b_i\|$ throughout the algorithm;
- can prove $\beta' = O(\log n + \beta)$
- Overall $O(n^2 \cdot n^2 (\log n + \beta)^2) = O(n^4 \beta^2)$ in the typical case $\beta > \log n$.

Total cost : $O(n^6 \beta^3)$.

Cost : $O(n^3)$ steps; cost of one step – close to GSO cost?

- Rational arithmetic $\Rightarrow$ control denominators;
- $b_i^* = b_i + \sum_{j<i} y_j b_j$.
- $(b_i, b_j) + \sum_{k<i} y_k (b_k, b_j) = 0 \ \ (= (b_i^*, b_j))$
- ... hence $\mathbf{y} = -B_{i-1}^t b_i / \det B_{i-1}^t B_{i-1}$
- Denominators $= O(\beta'^{2n})$ (works for μ_{ij} too).
- where β' is the largest $\|b_i\|$ throughout the algorithm;
- can prove $\beta' = O(\log n + \beta)$
- Overall $O(n^2 \cdot n^2 (\log n + \beta)^2) = O(n^4 \beta^2)$ in the typical case $\beta > \log n$.

Total cost : $O(n^6 \beta^3)$.

Cost : $O(n^3)$ steps; cost of one step – close to GSO cost?

- Rational arithmetic $\Rightarrow$ control denominators;
- $b_i^* = b_i + \sum_{j<i} y_j b_j$.
- $(b_i, b_j) + \sum_{k<i} y_k (b_k, b_j) = 0 \quad (= (b_i^*, b_j))$
- ... hence $\mathbf{y} = -B_{i-1}^t b_i / \det B_{i-1}^t B_{i-1}$
- Denominators $= O(\beta'^{2n})$ (works for μ_{ij} too).
- where β' is the largest $\|b_i\|$ throughout the algorithm;
- can prove $\beta' = O(\log n + \beta)$
- Overall $O(n^2 \cdot n^2 (\log n + \beta)^2) = O(n^4 \beta^2)$ in the typical case $\beta > \log n$.

Total cost : $O(n^6 \beta^3)$.

Lattice basis reduction – LLL, recent progresses

- Floating-point GSO;
- Quasi-linear LLL.

Lattice basis reduction – LLL, recent results

Floating-point GSO :

- GSO expensive (big rational / integer computations);
- Need little information ($\lfloor \mu_{ij} \rceil$);
- Use approximation / floating-point computations;
- ... but numerically unstable.
- Recompute GSO when instability is detected;
- Use a precise fpa model.

Theorem (Nguyen-Stehlé). LLL can be done with fpa in precision $O(d)$, giving a cost $O(d^4\beta(d+\beta))$.

Lattice basis reduction – LLL, recent results

Floating-point GSO :

- GSO expensive (big rational / integer computations);
- Need little information ($\lfloor \mu_{ij} \rceil$);
- Use approximation / floating-point computations;
- ... but numerically unstable.
- Recompute GSO when instability is detected;
- Use a precise fpa model.

Theorem (Nguyen-Stehlé). LLL can be done with fpa in precision $O(d)$, giving a cost $O(d^4\beta(d+\beta))$.

Lattice basis reduction – LLL, recent results

Floating-point GSO :

- GSO expensive (big rational / integer computations);
- Need little information ($\lfloor \mu_{ij} \rceil$);
- Use approximation / floating-point computations;
- ... but numerically unstable.
- Recompute GSO when instability is detected;
- Use a precise fpa model.

Theorem (Nguyen-Stehlé). LLL can be done with fpa in precision $O(d)$, giving a cost $O(d^4\beta(d+\beta))$.

Lattice basis reduction – LLL, recent results

Floating-point GSO :

- GSO expensive (big rational / integer computations);
- Need little information ($\lfloor \mu_{ij} \rceil$);
- Use approximation / floating-point computations;
- ... but numerically unstable.
- Recompute GSO when instability is detected;
- Use a precise fpa model.

Theorem (Nguyen-Stehlé). LLL can be done with fpa in precision $O(d)$, giving a cost $O(d^4\beta(d+\beta))$.

Lattice basis reduction – LLL, recent results

Floating-point GSO :

- GSO expensive (big rational / integer computations);
- Need little information ($\lfloor \mu_{ij} \rceil$);
- Use approximation / floating-point computations;
- ... but numerically unstable.
- Recompute GSO when instability is detected;
- Use a precise fpa model.

Theorem (Nguyen-Stehlé). LLL can be done with fpa in precision $O(d)$, giving a cost $O(d^4\beta(d+\beta))$.

Lattice basis reduction – LLL, recent results

Floating-point GSO :

- GSO expensive (big rational / integer computations);
- Need little information ($\lfloor \mu_{ij} \rceil$);
- Use approximation / floating-point computations;
- ... but numerically unstable.
- Recompute GSO when instability is detected;
- Use a precise fpa model.

Theorem (Nguyen-Stehlé). LLL can be done with fpa in precision $O(d)$, giving a cost $O(d^4\beta(d+\beta))$.

Lattice basis reduction – LLL, recent results

Quasi-linear LLL :

- Novocin-Stehlé-Villard, inspired of fast gcd – might be practical;
- H.-Pujol-Stehlé, using BKZ analysis and fast Gauss' algorithm;
- Schnorr, choosing best index j for LLL at each step + fast Gauss.

Theorem (Novocin-Stehlé-Villard). almost-LLL can be done in time $\tilde{O}(n^5\beta + n^{\omega+1}\beta)$.

Lattice basis reduction – LLL in practice

- (old) Folklore: LLL performs better than analysis;
- Often finds first minimum.

Thorough experimental studies by Nguyen and Stehlé (2007).

- In small dimensions ≤ 20, it works more or less;
- Otherwise, SVP approx factor $\approx (1.04)^n$.
- Analysis is sharp.

Lattice basis reduction – BKZ

$(b_1, \ldots, b_n)$ is k-BKZ reduced if:

- $(b_1, \ldots, b_n)$ size-reduced;
- For all i, $\pi_{L_i^\perp}(b_i, \ldots, b_{\min(n,i+k-1)})$ is HKZ-reduced.

Use a k-HKZ oracle.

Theorem(Schnorr, 1994) If $(b_1, \ldots, b_n)$ is k-BKZ reduced, then

- $\|b_i\| \leq k^{\frac{n-1}{k-1}} \frac{i+3}{4} \lambda_i(L)$.
- $\|b_i^*\| \leq k^{\frac{n-1}{k-1}} \lambda_i(L)$.
- $HF \leq \sqrt{k}^{\frac{n}{k-1}}$.

Proof. Combine Minkowski inequalities over projected sublattices.

Lattice basis reduction – BKZ, algorithmic aspects

Strategies :

- LLL-like: HKZ-reduce at the smallest possible i;
- Schnorr-Euchner: reduce at $1, 2, 3, \ldots, n-k+1, 1, 2, 3, \ldots$.

Cost :

- LLL-type arguments do not seem to work;
- hard to control a potential when reduction occurs;
- Does the LLL strategy even terminate?
- Schnorr-Euchner behaves well in practice, can prove $2^{O(n)}$ bound.

Lattice basis reduction – BKZ, algorithmic aspects

Strategies :

- LLL-like: HKZ-reduce at the smallest possible i;
- Schnorr-Euchner: reduce at 1, 2, 3, ..., $n-k+1$, 1, 2, 3, ...

Cost :

- LLL-type arguments do not seem to work;
- hard to control a potential when reduction occurs;
- Does the LLL strategy even terminate?
- Schnorr-Euchner behaves well in practice, can prove $2^{O(n)}$ bound.

Lattice basis reduction – BKZ, algorithmic aspects

Strategies :

- LLL-like: HKZ-reduce at the smallest possible i;
- Schnorr-Euchner: reduce at $1, 2, 3, \ldots, n-k+1, 1, 2, 3, \ldots$.

Cost :

- LLL-type arguments do not seem to work;
- hard to control a potential when reduction occurs;
- Does the LLL strategy even terminate?
- Schnorr-Euchner behaves well in practice, can prove $2^{O(n)}$ bound.

Lattice basis reduction – BKZ, algorithmic aspects

Strategies :

- LLL-like: HKZ-reduce at the smallest possible i;
- Schnorr-Euchner: reduce at $1, 2, 3, \ldots, n-k+1, 1, 2, 3, \ldots$.

Cost :

- LLL-type arguments do not seem to work;
- hard to control a potential when reduction occurs;
- Does the LLL strategy even terminate?
- Schnorr-Euchner behaves well in practice, can prove $2^{O(n)}$ bound.

Lattice basis reduction – BKZ, algorithmic aspects

Strategies :

- LLL-like: HKZ-reduce at the smallest possible i;
- Schnorr-Euchner: reduce at $1, 2, 3, \ldots, n-k+1, 1, 2, 3, \ldots$.

Cost :

- LLL-type arguments do not seem to work;
- hard to control a potential when reduction occurs;
- Does the LLL strategy even terminate?
- Schnorr-Euchner behaves well in practice, can prove $2^{O(n)}$ bound.

Lattice basis reduction – BKZ, algorithmic aspects

Strategies :

- LLL-like: HKZ-reduce at the smallest possible i;
- Schnorr-Euchner: reduce at $1, 2, 3, \ldots, n-k+1, 1, 2, 3, \ldots$.

Cost :

- LLL-type arguments do not seem to work;
- hard to control a potential when reduction occurs;
- Does the LLL strategy even terminate?
- Schnorr-Euchner behaves well in practice, can prove $2^{O(n)}$ bound.

Lattice basis reduction – BKZ, algorithmic aspects

A dual strategy :

- Stop after a polynomial number of steps;
- ... but what is the quality of the basis?

Theorem. After $\mathcal{O}\left(\frac{n^3}{k^2}(\log n + \log\log \beta)\right)$ calls to HKZ_k, BKZ_k returns a basis $(b_1, \ldots, b_n)$ of L such that:

$$\|b_1\| \leq 2k^{\frac{n-1}{2(k-1)}+\frac{3}{2}}(\det L)^{1/n}$$

Lattice basis reduction – BKZ, algorithmic aspects

A dual strategy :

- Stop after a polynomial number of steps;
- ... but what is the quality of the basis?

Theorem. After $\mathcal{O}\left(\frac{n^3}{k^2}(\log n + \log\log\beta)\right)$ calls to HKZ_k, BKZ_k returns a basis $(b_1, \ldots, b_n)$ of L such that:

$$\|b_1\| \leq 2k^{\frac{n-1}{2(k-1)}+\frac{3}{2}}(\det L)^{1/n}$$

Lattice basis reduction – BKZ, algorithmic aspects

A dual strategy :

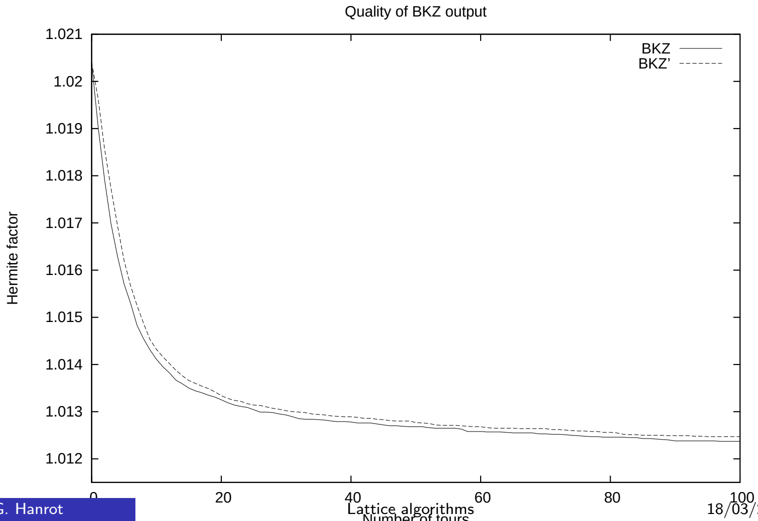
- Stop after a polynomial number of steps;
- ... but what is the quality of the basis?

Theorem. After $\mathcal{O}\left(\frac{n^3}{k^2}(\log n + \log\log\beta)\right)$ calls to HKZ_k, BKZ_k returns a basis $(b_1, \ldots, b_n)$ of L such that:

$$\|b_1\| \le 2k^{\frac{n-1}{2(k-1)}+\frac{3}{2}}(\det L)^{1/n}$$

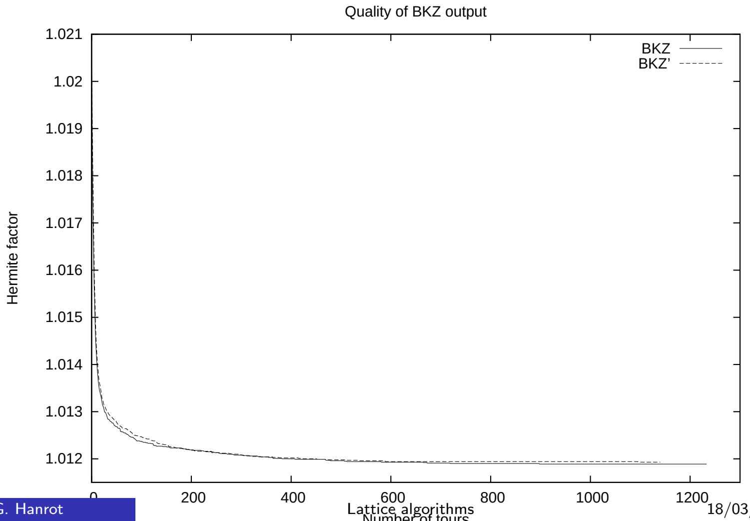
Lattice basis reduction – BKZ, a picture

Progress made during the execution of BKZ



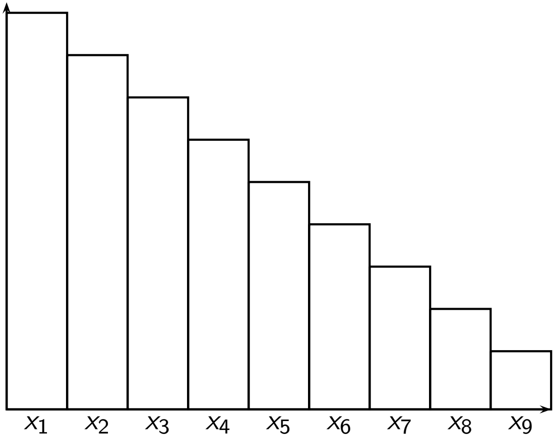
Lattice basis reduction – BKZ, a picture

Progress made during the execution of BKZ



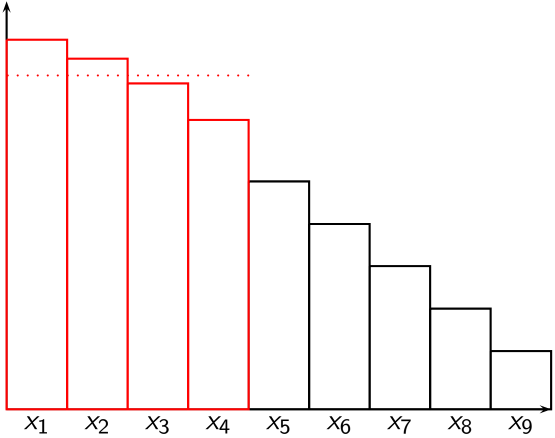
Lattice basis reduction – BKZ, analysis

- Idea: sandpile model;
- Let $x_i := \log \|b_i^*\|$.



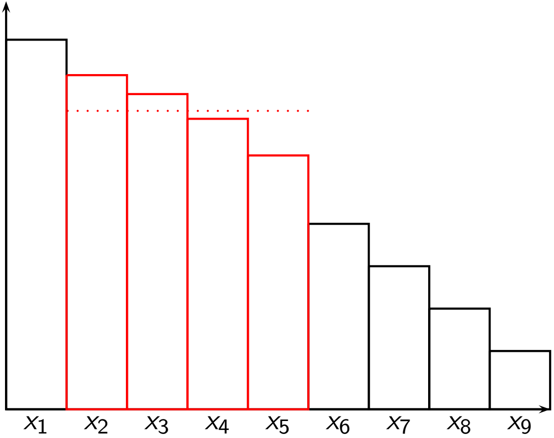
Lattice basis reduction – BKZ, analysis

- Idea: sandpile model;
- Let $x_i := \log \|b_i^*\|$.



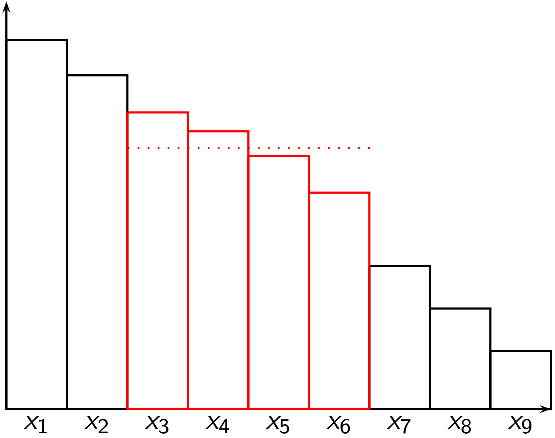
Lattice basis reduction – BKZ, analysis

- Idea: sandpile model;
- Let $x_i := \log \|b_i^*\|$.



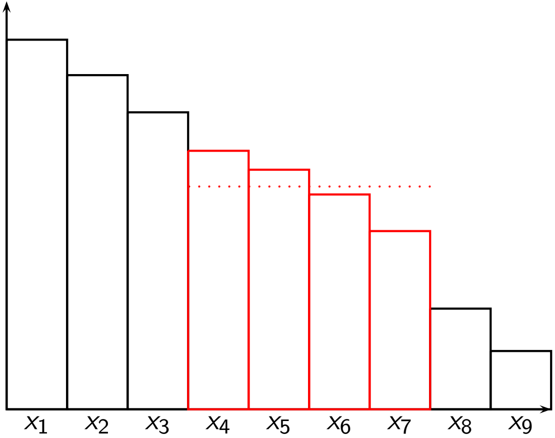
Lattice basis reduction – BKZ, analysis

- Idea: sandpile model;
- Let $x_i := \log \|b_i^*\|$.



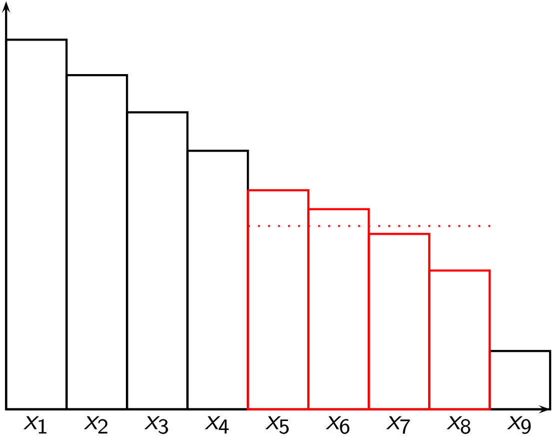
Lattice basis reduction – BKZ, analysis

- Idea: sandpile model;
- Let $x_i := \log \|b_i^*\|$.



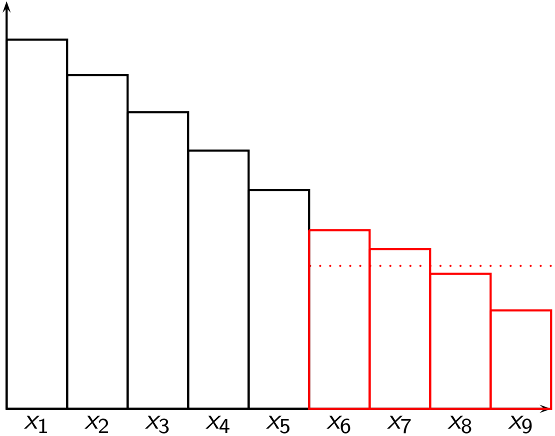
Lattice basis reduction – BKZ, analysis

- Idea: sandpile model;
- Let $x_i := \log \|b_i^*\|$.



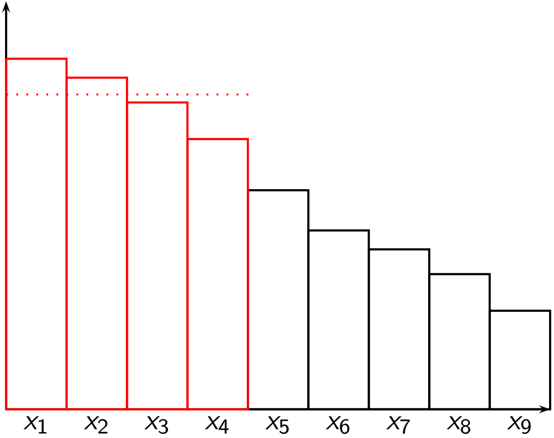
Lattice basis reduction – BKZ, analysis

- Idea: sandpile model;
- Let $x_i := \log \|b_i^*\|$.

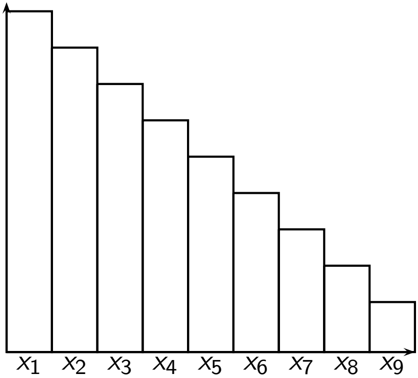


Lattice basis reduction – BKZ, analysis

- Idea: sandpile model;
- Let $x_i := \log \|b_i^*\|$.



BKZ$_k$'s sandpile as a dynamic system



$X = (x_1, \ldots, x_n)^T$

$X_{0.5} \leftarrow A_1 X$

$X_1 \leftarrow A_1 X + \Gamma_1$

$X_2 \leftarrow A_2 X_1 + \Gamma_2$

$\ldots$

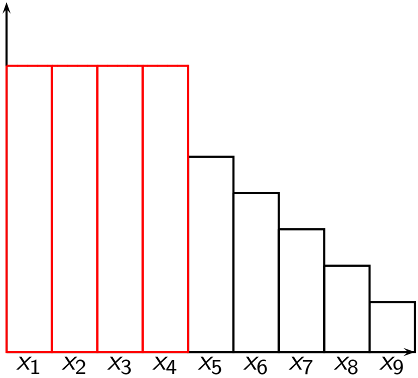
$X_j = A_j X_j + \Gamma_j$

with $j = n - k + 1$

A full tour:

$X' \leftarrow AX + \Gamma$

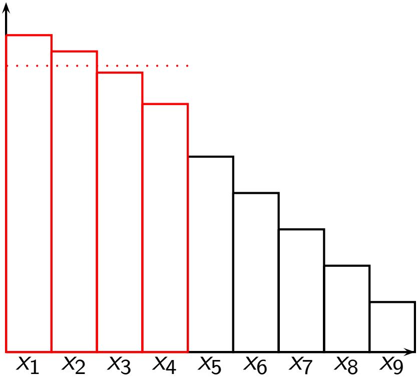
BKZ$_k$'s sandpile as a dynamic system



$X = (x_1, \ldots, x_n)^T$
$X_{0.5} \leftarrow A_1 X$
$X_1 \leftarrow A_1 X + \Gamma_1$
$X_2 \leftarrow A_2 X_1 + \Gamma_2$
$\ldots$
$X_j = A_j X_j + \Gamma_j$
with $j = n - k + 1$

A full tour:
$X' \leftarrow AX + \Gamma$

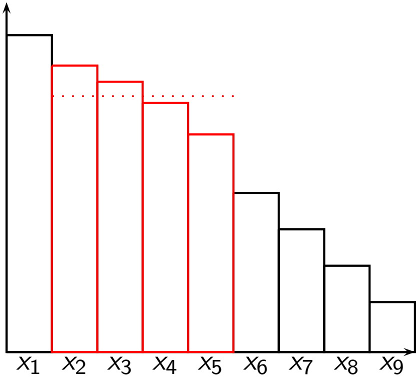
BKZ$_k$'s sandpile as a dynamic system



$X = (x_1, \dots, x_n)^T$
$X_{0.5} \leftarrow A_1 X$
$X_1 \leftarrow A_1 X + \Gamma_1$
$X_2 \leftarrow A_2 X_1 + \Gamma_2$
...
$X_j = A_j X_j + \Gamma_j$
with $j = n - k + 1$

A full tour:
$X' \leftarrow AX + \Gamma$

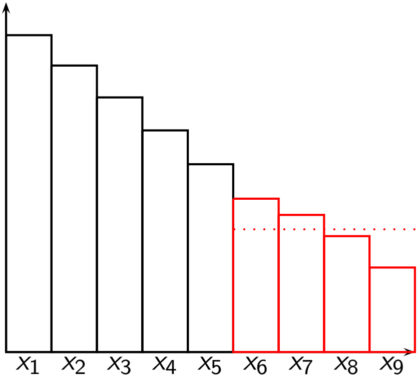
BKZ$_k$'s sandpile as a dynamic system



$X = (x_1, \ldots, x_n)^T$
$X_{0.5} \leftarrow A_1 X$
$X_1 \leftarrow A_1 X + \Gamma_1$
$X_2 \leftarrow A_2 X_1 + \Gamma_2$
$\ldots$
$X_j = A_j X_j + \Gamma_j$
with $j = n - k + 1$

A full tour:
$X' \leftarrow AX + \Gamma$

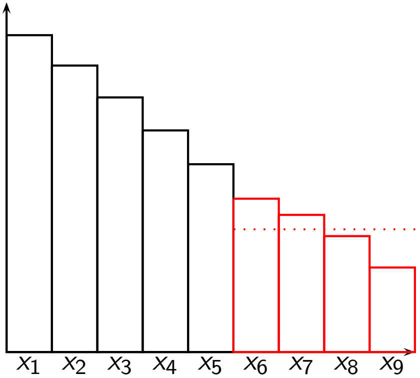
BKZ$_k$'s sandpile as a dynamic system



$X = (x_1, \ldots, x_n)^T$
$X_{0.5} \leftarrow A_1 X$
$X_1 \leftarrow A_1 X + \Gamma_1$
$X_2 \leftarrow A_2 X_1 + \Gamma_2$
$\ldots$
$X_j = A_j X_j + \Gamma_j$
with $j = n - k + 1$

A full tour:
$X' \leftarrow AX + \Gamma$

BKZ$_k$'s sandpile as a dynamic system



$X = (x_1, \ldots, x_n)^T$
$X_{0.5} \leftarrow A_1 X$
$X_1 \leftarrow A_1 X + \Gamma_1$
$X_2 \leftarrow A_2 X_1 + \Gamma_2$
$\ldots$
$X_j = A_j X_j + \Gamma_j$
with $j = n - k + 1$

A full tour:
$X' \leftarrow AX + \Gamma$

Properties of the model

$$X \leftarrow AX + \Gamma$$

- Reducedness of the output $\Rightarrow$ fixed poi nts.

- Speed of convergence $\Rightarrow$ eigenvalues of $A^T A$.

Properties of the model

$$X \leftarrow AX + \Gamma$$

- Reducedness of the output $\Rightarrow$ fixed poi nts.

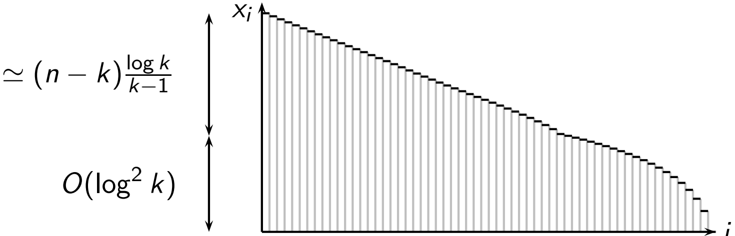
- Speed of convergence $\Rightarrow$ eigenvalues of $A^T A$.

  Geometric convergence: $\|X - X^\infty\|$ decreases by a constant factor every $\frac{n^2}{k^2}$ tours, i.e. $\frac{n^3}{k^2}$ calls to HKZ_k.

Properties of the model

$$X \leftarrow AX + \Gamma$$

- Reducedness of the output $\Rightarrow$ fixed poi nts.

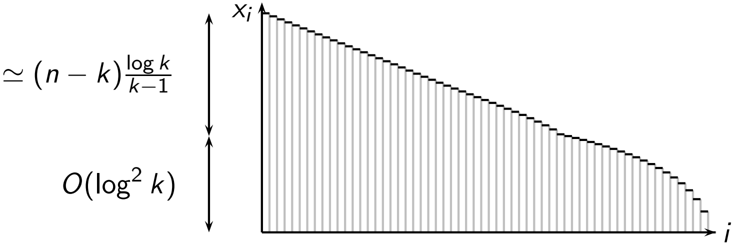


- Speed of convergence $\Rightarrow$ eigenvalues of $A^T A$.

Properties of the model

$$X \leftarrow AX + \Gamma$$

- Reducedness of the output $\Rightarrow$ fixed poi nts.



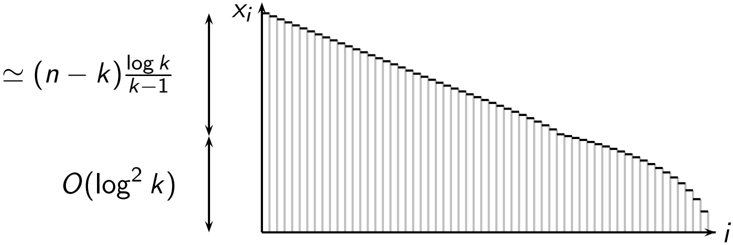
- Speed of convergence $\Rightarrow$ eigenvalues of $A^T A$.

  Geometric convergence: $\|X - X^\infty\|$ decreases by a constant factor every $\frac{n^2}{k^2}$ tours, *i.e.* $\frac{n^3}{k^2}$ calls to HKZ_k.

Properties of the model

$$X \leftarrow AX + \Gamma$$

- Reducedness of the output $\Rightarrow$ fixed poi nts.



- Speed of convergence $\Rightarrow$ eigenvalues of $A^T A$.

  Geometric convergence: $\|X - X^\infty\|$ decreases by a constant factor every $\frac{n^2}{k^2}$ tours, *i.e.* $\frac{n^3}{k^2}$ calls to HKZ_k.

Lattice basis reduction – Polynomial LLL

$b_1, \ldots, b_n \in \mathbb{K}[X]^n$, $L = K[X]b_1 \oplus \cdots \oplus K[X]b_n$.

- Orthogonality defect $= \sum_{i=1}^n \deg b_i - \deg\det(b_1, \ldots, b_n)$;
- OD $= 0 \Leftrightarrow$ up to row permutation max degrees are on the diagonal.
- also known as Popov normal form.

Theorem. There is a polynomial-time algorithm which returns a basis for which OD $= 0$, and (up to permutation) b_i is the i-th minimum of the lattice.

Lattice basis reduction – Polynomial LLL

Algorithm (see whiteboard).

II. Algorithms for SVP / CVP

II. 1. Approximate algorithms for SVP / CVP

SVP and CVP algorithms – approximate algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced;
Approx-SVP$(b_1, \ldots, b_n)$:

- Return b_1;
- Approx factor : LLL $= 2^{O(n)}$, k-BKZ $\approx k^{n/k}$;

Approx-CVP$(t, b_1, \ldots, b_n)$:

- Babai algorithms;
- Kannan's embedding technique

SVP and CVP algorithms – approximate algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced;
Approx-SVP$(b_1, \ldots, b_n)$:

- Return b_1;
- Approx factor : LLL $= 2^{O(n)}$, k-BKZ $\approx k^{n/k}$;

Approx-CVP$(t, b_1, \ldots, b_n)$:

- Babai algorithms;
- Kannan's embedding technique

SVP and CVP algorithms – approximate algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced;
Approx-SVP$(b_1, \ldots, b_n)$:

- Return b_1;
- Approx factor : LLL $= 2^{O(n)}$, k-BKZ $\approx k^{n/k}$;

Approx-CVP$(t, b_1, \ldots, b_n)$:

- Babai algorithms;
- Kannan's embedding technique

SVP and CVP algorithms – approximate algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced;
Approx-SVP$(b_1, \ldots, b_n)$:

- Return b_1;
- Approx factor : LLL $= 2^{O(n)}$, k-BKZ $\approx k^{n/k}$;

Approx-CVP$(t, b_1, \ldots, b_n)$:

- Babai algorithms;
- Kannan's embedding technique

SVP and CVP algorithms – approximate algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced;
Approx-SVP$(b_1, \ldots, b_n)$:

- Return b_1;
- Approx factor : LLL $= 2^{O(n)}$, k-BKZ $\approx k^{n/k}$;

Approx-CVP$(t, b_1, \ldots, b_n)$:

- Babai algorithms;
- Kannan's embedding technique

SVP and CVP algorithms – approximate algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced;
Approx-SVP$(b_1, \ldots, b_n)$:

- Return b_1;
- Approx factor : LLL $= 2^{O(n)}$, k-BKZ $\approx k^{n/k}$;

Approx-CVP$(t, b_1, \ldots, b_n)$:

- Babai algorithms;
- Kannan's embedding technique

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \dots, b_n)$, $(b_1, \dots, b_n)$ reduced, $t \in \mathbb{R}^n$.

Babai roundoff algorithm :

- Compute t_i such that $t = \sum_{i=1}^{n} t_i b_i$ (, ie. $B^{-1}t$)
- Return $\tilde{t} := \sum_{i=1}^{n} \lfloor t_i \rceil b_i$.

Theorem. If $(b_1, \dots, b_n)$ is LLL-reduced, then $\|\tilde{t} - t\| = 2^{O(d)} d(t, L)$.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai roundoff algorithm :

- Compute t_i such that $t = \sum_{i=1}^n t_i b_i$ (, ie. $B^{-1}t$)
- Return $\tilde{t} := \sum_{i=1}^n \lfloor t_i \rceil b_i$.

Theorem. If $(b_1, \ldots, b_n)$ is LLL-reduced, then
$\|\tilde{t} - t\| = 2^{O(d)} d(t, L)$.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai roundoff algorithm :

- Compute t_i such that $t = \sum_{i=1}^n t_i b_i$ (, ie. $B^{-1}t$)
- Return $\tilde{t} := \sum_{i=1}^n \lfloor t_i \rceil b_i$.

Theorem. If $(b_1, \ldots, b_n)$ is LLL-reduced, then $\|\tilde{t} - t\| = 2^{O(d)} d(t, L)$.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai roundoff algorithm :

- Compute t_i such that $t = \sum_{i=1}^n t_i b_i$ (, ie. $B^{-1}t$)
- Return $\tilde{t} := \sum_{i=1}^n \lfloor t_i \rceil b_i$.

Theorem. If $(b_1, \ldots, b_n)$ is LLL-reduced, then
$\|\tilde{t} - t\| = 2^{O(d)} d(t, L)$.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.

Babai nearest-plane algorithm :

- Size-reduce t wrt $(b_1, \ldots, b_n)$;
- $t_n \leftarrow \lfloor (t, b_n^*)/\|b_n^*\|^2 \rceil$;
- Continue with $(t - t_n b_n, b_1, \ldots, b_{n-1})$.
- Return $\tilde{t} := \sum t_i b_i$.

Theorem. We have

- $\|\tilde{t} - t\|^2 \leq \sum_{i=1}^n \|b_i^*\|^2/4 = 2^{O(n)}\|b_n^*\|$ (LLL-reduced);
- $\|\tilde{t} - t\|^2 = 2^{O(n)} d(t, L)$.

Much better than roundoff in theory and in practice.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai nearest-plane algorithm :

- Size-reduce t wrt $(b_1, \ldots, b_n)$;
- $t_n \leftarrow \lfloor (t, b_n^*)/\|b_n^*\|^2 \rceil$;
- Continue with $(t - t_n b_n, b_1, \ldots, b_{n-1})$.
- Return $\tilde{t} := \sum t_i b_i$.

Theorem. We have

- $\|\tilde{t} - t\|^2 \leq \sum_{i=1}^{n} \|b_i^*\|^2/4 = 2^{O(n)}\|b_n^*\|$ (LLL-reduced);
- $\|\tilde{t} - t\|^2 = 2^{O(n)} d(t, L)$.

Much better than roundoff in theory and in practice.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai nearest-plane algorithm :

- Size-reduce t wrt $(b_1, \ldots, b_n)$;
- $t_n \leftarrow \lfloor (t, b_n^*)/\|b_n^*\|^2 \rceil$;
- Continue with $(t - t_n b_n, b_1, \ldots, b_{n-1})$.
- Return $\tilde{t} := \sum t_i b_i$.

Theorem. We have

- $\|\tilde{t} - t\|^2 \le \sum_{i=1}^{n} \|b_i^*\|^2/4 = 2^{O(n)}\|b_n^*\|$ (LLL-reduced);
- $\|\tilde{t} - t\|^2 = 2^{O(n)} d(t, L)$.

Much better than roundoff in theory and in practice.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai nearest-plane algorithm :

- Size-reduce t wrt $(b_1, \ldots, b_n)$;
- $t_n \leftarrow \lfloor (t, b_n^*)/\|b_n^*\|^2 \rceil$;
- Continue with $(t - t_n b_n, b_1, \ldots, b_{n-1})$.
- Return $\tilde{t} := \sum t_i b_i$.

Theorem. We have

- $\|\tilde{t} - t\|^2 \leq \sum_{i=1}^n \|b_i^*\|^2/4 = 2^{O(n)}\|b_n^*\|$ (LLL-reduced);
- $\|\tilde{t} - t\|^2 = 2^{O(n)} d(t, L)$.

Much better than roundoff in theory and in practice.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai nearest-plane algorithm :

- Size-reduce t wrt $(b_1, \ldots, b_n)$;
- $t_n \leftarrow \lfloor (t, b_n^*)/\|b_n^*\|^2 \rceil$;
- Continue with $(t - t_n b_n, b_1, \ldots, b_{n-1})$.
- Return $\tilde{t} := \sum t_i b_i$.

Theorem. We have

- $\|\tilde{t} - t\|^2 \leq \sum_{i=1}^{n} \|b_i^*\|^2/4 = 2^{O(n)}\|b_n^*\|$ (LLL-reduced);
- $\|\tilde{t} - t\|^2 = 2^{O(n)} d(t, L)$.

Much better than roundoff in theory and in practice.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai nearest-plane algorithm :

- Size-reduce t wrt $(b_1, \ldots, b_n)$;
- $t_n \leftarrow \lfloor (t, b_n^*)/\|b_n^*\|^2 \rceil$;
- Continue with $(t - t_n b_n, b_1, \ldots, b_{n-1})$.
- Return $\tilde{t} := \sum t_i b_i$.

Theorem. We have

- $\|\tilde{t} - t\|^2 \leq \sum_{i=1}^n \|b_i^*\|^2/4 = 2^{O(n)}\|b_n^*\|$ (LLL-reduced);
- $\|\tilde{t} - t\|^2 = 2^{O(n)} d(t, L)$.

Much better than roundoff in theory and in practice.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai nearest-plane algorithm :

- Size-reduce t wrt $(b_1, \ldots, b_n)$;
- $t_n \leftarrow \lfloor (t, b_n^*)/\|b_n^*\|^2 \rceil$;
- Continue with $(t - t_n b_n, b_1, \ldots, b_{n-1})$.
- Return $\tilde{t} := \sum t_i b_i$.

Theorem. We have

- $\|\tilde{t} - t\|^2 \leq \sum_{i=1}^{n} \|b_i^*\|^2/4 = 2^{O(n)}\|b_n^*\|$ (LLL-reduced);
- $\|\tilde{t} - t\|^2 = 2^{O(n)} d(t, L)$.

Much better than roundoff in theory and in practice.

SVP and CVP algorithms – Babai algorithms

$L = L(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
Babai nearest-plane algorithm :

- Size-reduce t wrt $(b_1, \ldots, b_n)$;
- $t_n \leftarrow \lfloor (t, b_n^*)/\|b_n^*\|^2 \rceil$;
- Continue with $(t - t_n b_n, b_1, \ldots, b_{n-1})$.
- Return $\tilde{t} := \sum t_i b_i$.

Theorem. We have

- $\|\tilde{t} - t\|^2 \le \sum_{i=1}^n \|b_i^*\|^2/4 = 2^{O(n)}\|b_n^*\|$ (LLL-reduced);
- $\|\tilde{t} - t\|^2 = 2^{O(n)} d(t, L)$.

Much better than roundoff in theory and in practice.

SVP and CVP algorithms – Kannan's embedding technique

$L = L(b_1, \ldots, b_n)$, $B = (b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.

"Reduce" CVP to SVP by using

$$L' := \begin{pmatrix} B & -t \\ 0 & C \end{pmatrix},$$

for a large constant C.

- LLL-reduce L', read $\tilde{t}$ on first vector;
- Short vectors in $L' = (u - x_{n+1}t, x_{n+1}C)$, $u \in L$;
- if $C \geq 2^{O(n)}d(L, t)$, need to have $x_{n+1} = 1 \Rightarrow$ actual close vector.
- Algorithmic interpretation: further than Babai.

Can be better, somewhat less convenient.

SVP and CVP algorithms – Kannan's embedding technique

$L = L(b_1, \ldots, b_n)$, $B = (b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
"Reduce" CVP to SVP by using

$$L' := \begin{pmatrix} B & -t \\ 0 & C \end{pmatrix},$$

for a large constant C.

- LLL-reduce L', read $\tilde{t}$ on first vector;
- Short vectors in $L' = (u - x_{n+1}t, x_{n+1}C)$, $u \in L$;
- if $C \geq 2^{O(n)}d(L,t)$, need to have $x_{n+1} = 1 \Rightarrow$ actual close vector.
- Algorithmic interpretation: further than Babai.

Can be better, somewhat less convenient.

SVP and CVP algorithms – Kannan's embedding technique

$L = L(b_1, \ldots, b_n)$, $B = (b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
"Reduce" CVP to SVP by using

$$L' := \begin{pmatrix} B & -t \\ 0 & C \end{pmatrix},$$

for a large constant C.

- LLL-reduce L', read $\tilde{t}$ on first vector;
- Short vectors in $L' = (u - x_{n+1}t, x_{n+1}C)$, $u \in L$;
- if $C \geq 2^{O(n)} d(L, t)$, need to have $x_{n+1} = 1 \Rightarrow$ actual close vector.
- Algorithmic interpretation: further than Babai.

Can be better, somewhat less convenient.

SVP and CVP algorithms – Kannan's embedding technique

$L = L(b_1, \ldots, b_n)$, $B = (b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
"Reduce" CVP to SVP by using

$$L' := \begin{pmatrix} B & -t \\ 0 & C \end{pmatrix},$$

for a large constant C.

- LLL-reduce L', read $\tilde{t}$ on first vector;
- Short vectors in $L' = (u - x_{n+1}t, x_{n+1}C)$, $u \in L$;
- if $C \geq 2^{O(n)} d(L, t)$, need to have $x_{n+1} = 1 \Rightarrow$ actual close vector.
- Algorithmic interpretation: further than Babai.

Can be better, somewhat less convenient.

SVP and CVP algorithms – Kannan's embedding technique

$L = L(b_1, \ldots, b_n)$, $B = (b_1, \ldots, b_n)$ reduced, $t \in \mathbb{R}^n$.
"Reduce" CVP to SVP by using

$$L' := \begin{pmatrix} B & -t \\ 0 & C \end{pmatrix},$$

for a large constant C.

- LLL-reduce L', read $\tilde{t}$ on first vector;
- Short vectors in $L' = (u - x_{n+1}t, x_{n+1}C)$, $u \in L$;
- if $C \geq 2^{O(n)} d(L, t)$, need to have $x_{n+1} = 1 \Rightarrow$ actual close vector.
- Algorithmic interpretation: further than Babai.

Can be better, somewhat less convenient.

II. 2. Exact algorithms for SVP / CVP

SVP and CVP algorithms – outline

- **The KFP enumeration-based algorithm**
  - R. Kannan: `Improved algorithms for integer programming and related lattice problems`, STOC'83
  - U. Fincke & M. Pohst: `A procedure for determining algebraic integers of given norm`, EUROCAL'83
- Saturating the space: The AKS solver and its descendants
- Using the Voronoi cell: the Micciancio-Voulgaris algorithm

The Kannan-Fincke-Pohst enumeration algorithm

Given $(\mathbf{b}_i)_{i\leq n}$ and $\mathbf{t} \in \mathbb{R}^n$, we look for all $(x_i)_i \in \mathbb{Z}^n$ s.t.:

$$\Big\| \sum_i x_i \mathbf{b}_i - \mathbf{t} \Big\|^2 = \sum_i \Big(x_i - t_i + \sum_{j>i} \mu_{j,i} x_j \Big)^2 \|\mathbf{b}_i^*\|^2 \leq A$$

where $\mathbf{t} = \sum_i t_i \mathbf{b}_i^*$ and A is arbitrary.

By **successive projections**:

$$\begin{array}{rcl} (x_n - t_n)^2 \|\mathbf{b}_n^*\|^2 & \leq & A \\ (x_{n-1} - t_{n-1} + \mu_{n,n-1} x_n)^2 \|\mathbf{b}_{n-1}^*\|^2 + (x_n - t_n)^2 \|\mathbf{b}_n^*\|^2 & \leq & A \\ & \cdots & \\ \displaystyle\sum_{j\geq i} (x_j - t_j + \sum_{k>j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 & \leq & A \\ & \cdots & \end{array}$$

The Kannan-Fincke-Pohst enumeration algorithm

Given $(\mathbf{b}_i)_{i\leq n}$ and $\mathbf{t} \in \mathbb{R}^n$, we look for all $(x_i)_i \in \mathbb{Z}^n$ s.t.:

$$\Big\|\sum_i x_i \mathbf{b}_i - \mathbf{t}\Big\|^2 = \sum_i \Big(x_i - t_i + \sum_{j>i} \mu_{j,i} x_j\Big)^2 \|\mathbf{b}_i^*\|^2 \leq A$$

where $\mathbf{t} = \sum_i t_i \mathbf{b}_i^*$ and A is arbitrary.

By **successive projections**:

$$\begin{array}{rcl}
(x_n - t_n)^2 \|\mathbf{b}_n^*\|^2 & \leq & A \\
(x_{n-1} - t_{n-1} + \mu_{n,n-1} x_n)^2 \|\mathbf{b}_{n-1}^*\|^2 + (x_n - t_n)^2 \|\mathbf{b}_n^*\|^2 & \leq & A \\
 & \cdots & \\
\displaystyle\sum_{j \geq i} (x_j - t_j + \sum_{k>j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 & \leq & A \\
 & \cdots &
\end{array}$$

The Kannan-Fincke-Pohst enumeration algorithm

$$
\begin{array}{rcl}
(x_n - t_n)^2 \|\mathbf{b}_n^*\|^2 & \leq & A \\
(x_{n-1} - t_{n-1} + \mu_{n,n-1} x_n)^2 \|\mathbf{b}_{n-1}^*\|^2 + (x_n - t_n)^2 \|\mathbf{b}_n^*\|^2 & \leq & A \\
 & \ldots & \\
\sum_{j \geq i} (x_j - t_j + \sum_{k > j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 & \leq & A \\
 & \ldots &
\end{array}
$$

- For each value of $(x_k, \ldots, x_n)$, x_{k-1} belongs to a finite set.
- KFP is a **tree traversal**, where one is interested in the leaves.

- Cost analysis reduces to counting lattice points in balls.

Gaussian heuristic

For any "nice" K, we have $|L \cap K| \approx \frac{\mathrm{vol}\ K}{\det L}$.

The Kannan-Fincke-Pohst enumeration algorithm

$$
\begin{array}{rcl}
(x_n - t_n)^2 \|\mathbf{b}_n^*\|^2 & \leq & A \\
(x_{n-1} - t_{n-1} + \mu_{n,n-1} x_n)^2 \|\mathbf{b}_{n-1}^*\|^2 + (x_n - t_n)^2 \|\mathbf{b}_n^*\|^2 & \leq & A \\
 & \cdots & \\
\sum_{j \geq i} (x_j - t_j + \sum_{k > j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 & \leq & A \\
 & \cdots &
\end{array}
$$

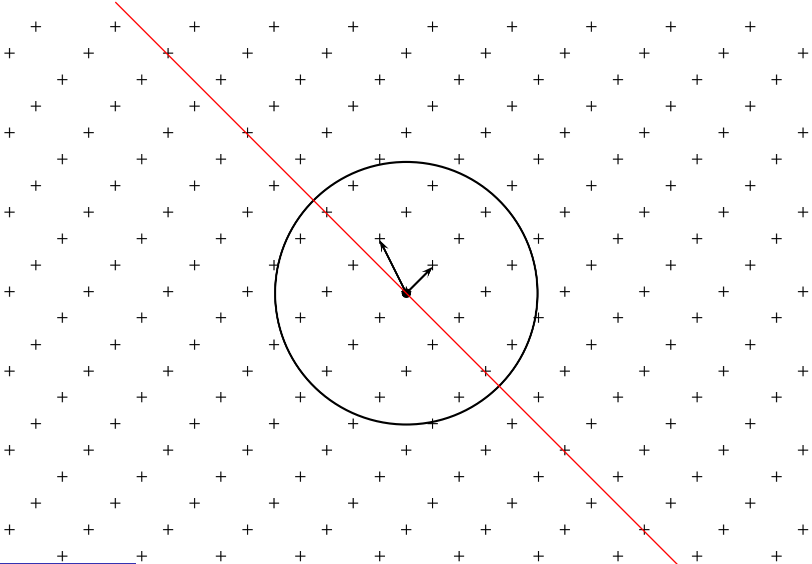
- For each value of $(x_k, \ldots, x_n)$, x_{k-1} belongs to a finite set.
- KFP is a **tree traversal**, where one is interested in the leaves.

- Cost analysis reduces to counting lattice points in balls.

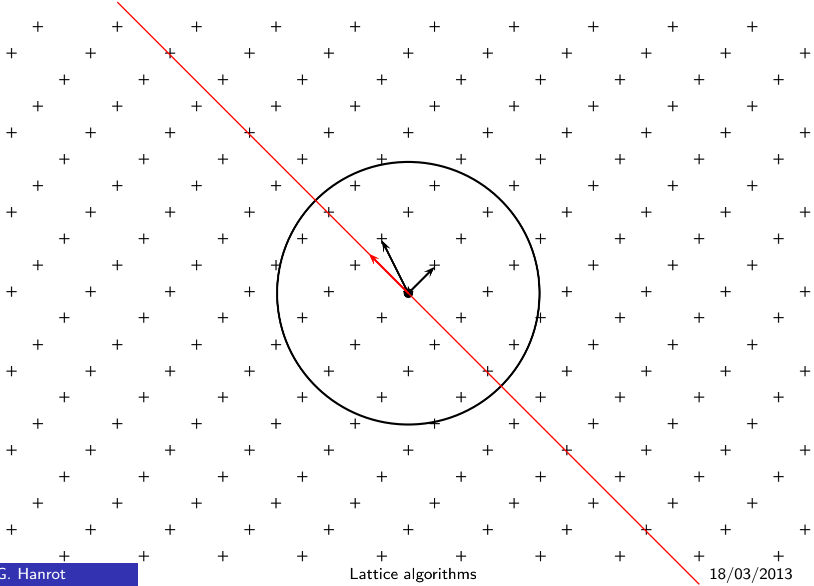
Gaussian heuristic

For any "nice" K, we have $|L \cap K| \approx \frac{\mathrm{vol}\ K}{\det L}$.

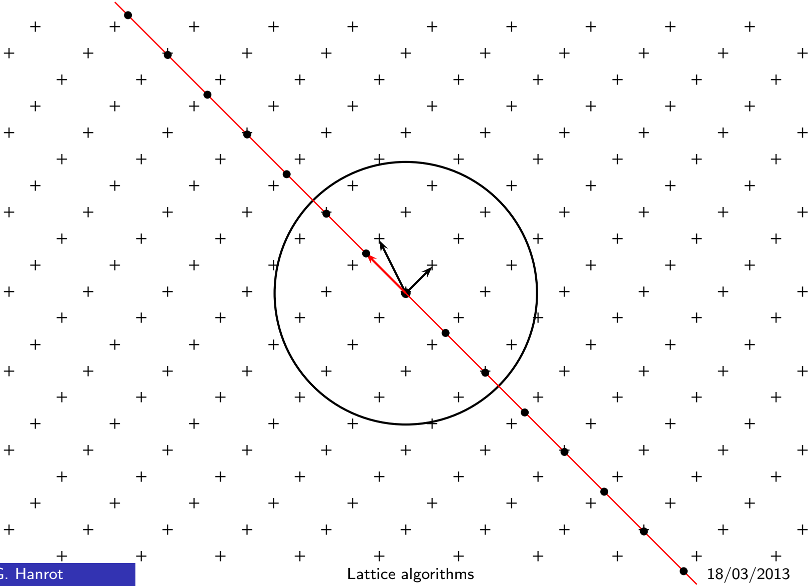
CVP, SVP – the KFP enumeration algorithm



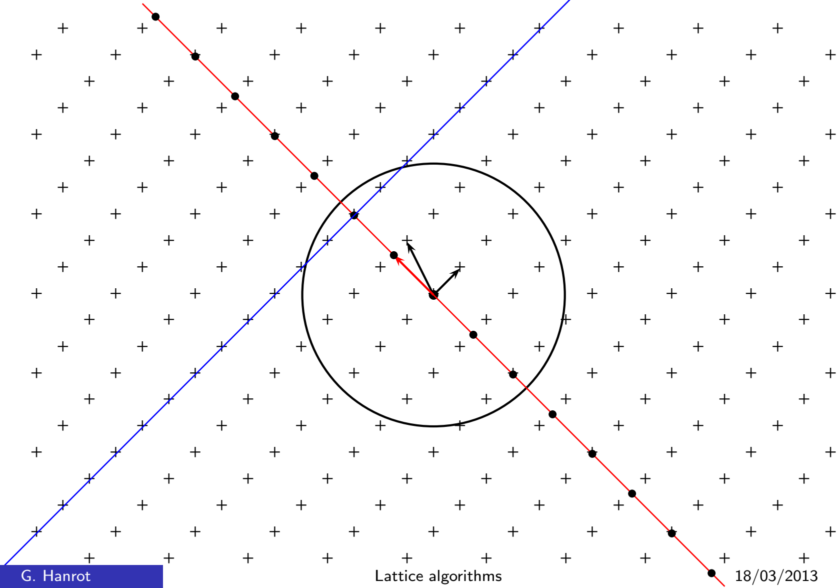
CVP, SVP – the KFP enumeration algorithm



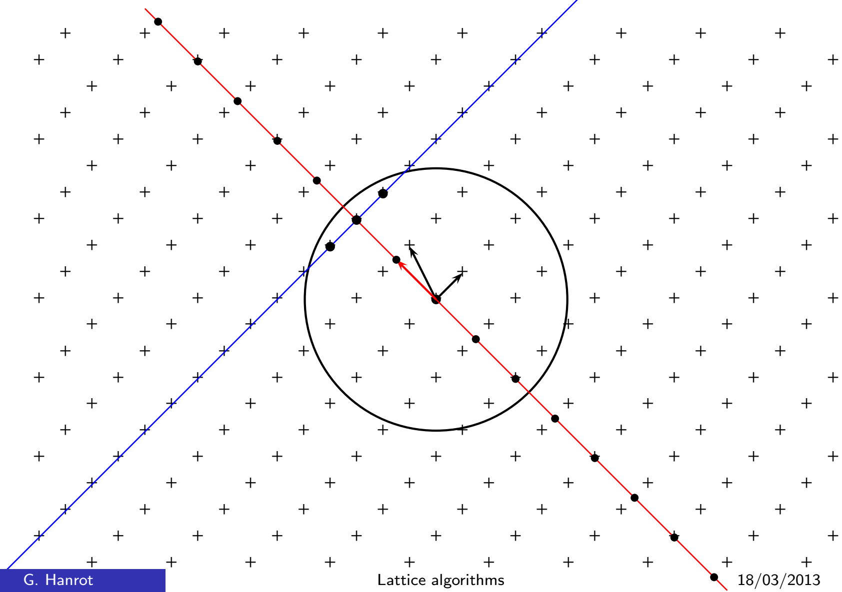
CVP, SVP – the KFP enumeration algorithm



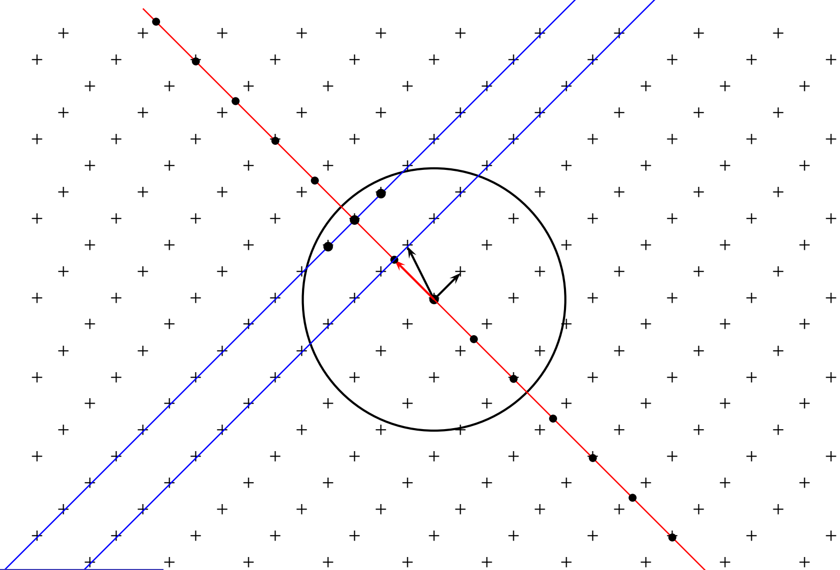
CVP, SVP – the KFP enumeration algorithm



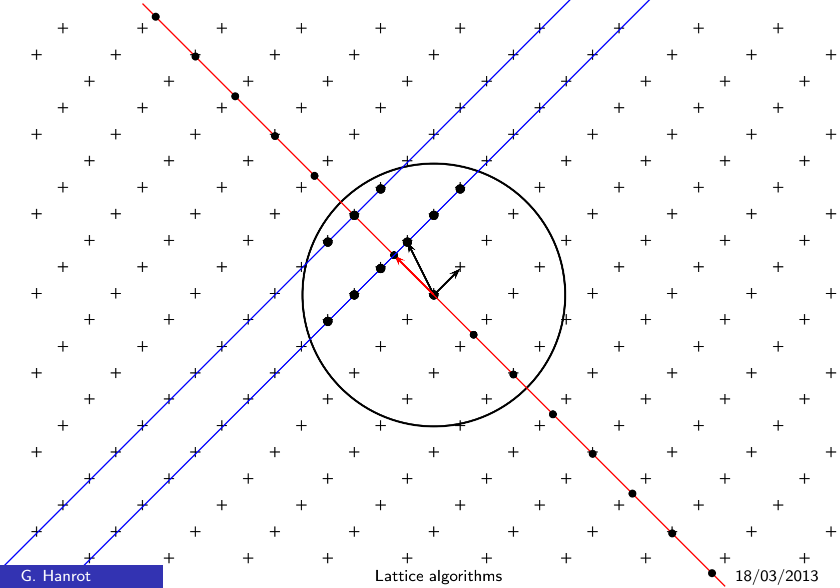
CVP, SVP – the KFP enumeration algorithm



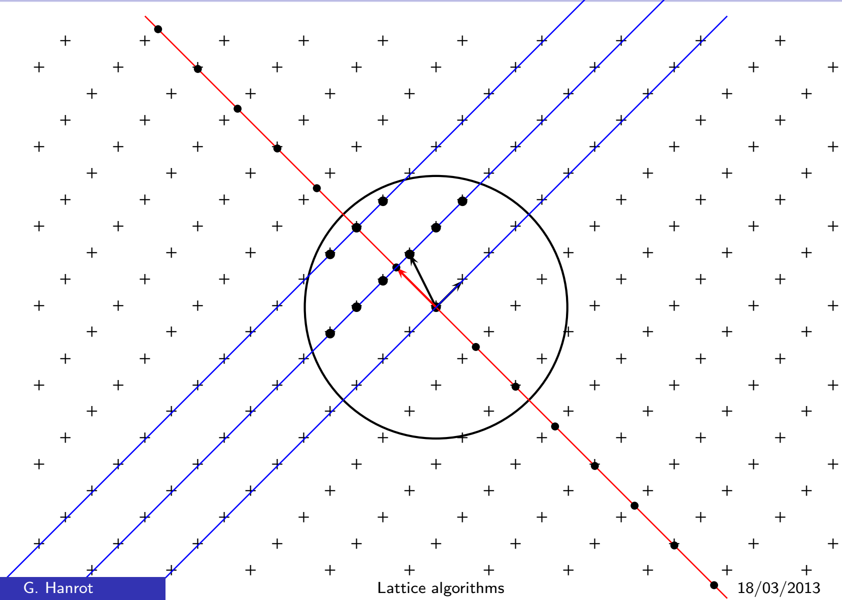
CVP, SVP – the KFP enumeration algorithm



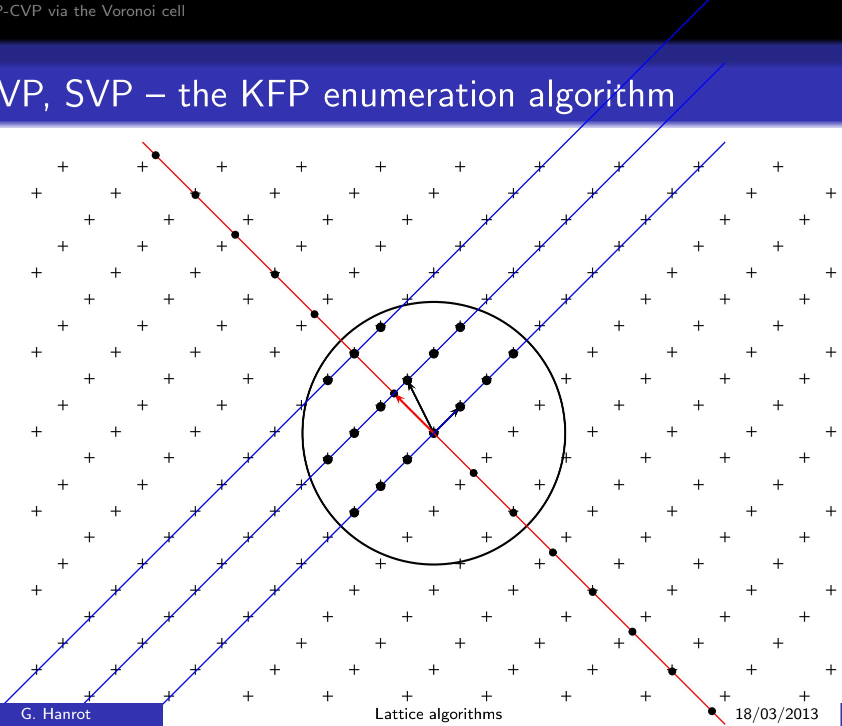
CVP, SVP – the KFP enumeration algorithm



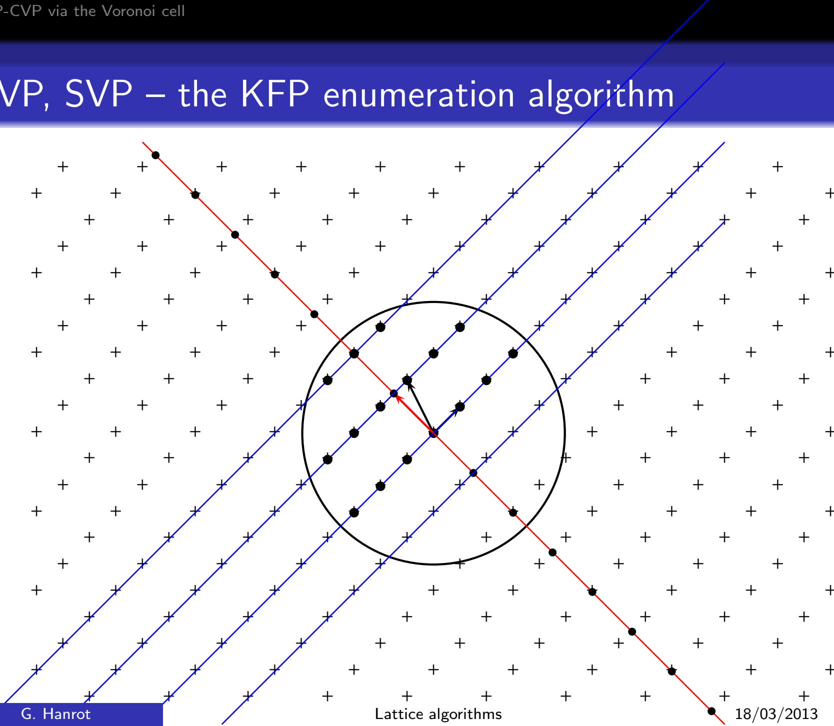
CVP, SVP – the KFP enumeration algorithm



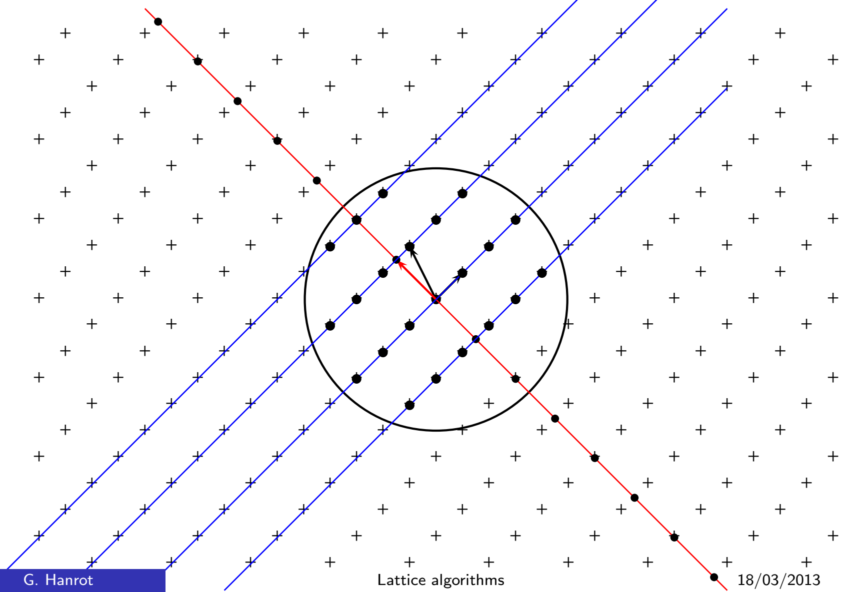
CVP, SVP – the KFP enumeration algorithm



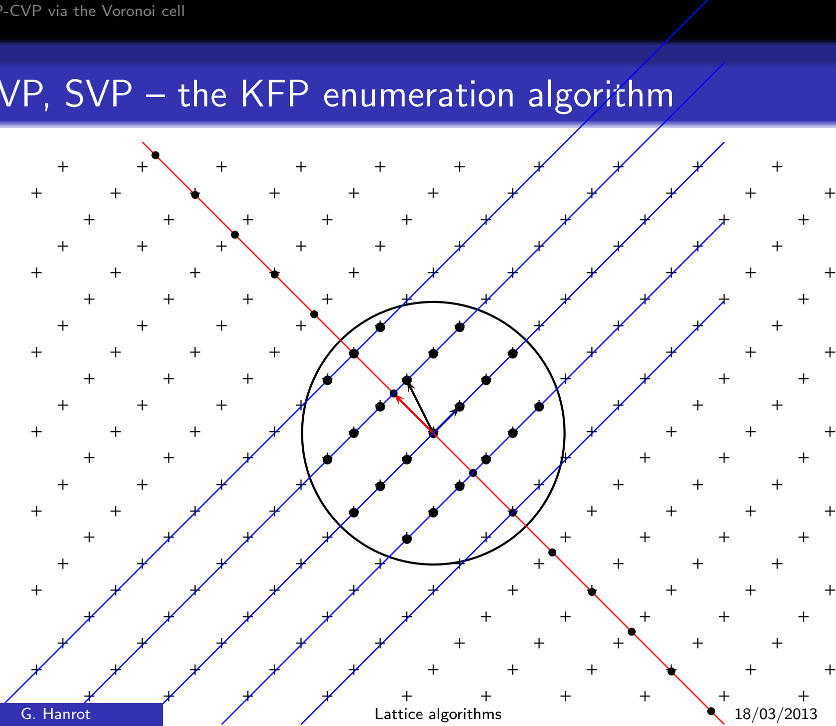
CVP, SVP – the KFP enumeration algorithm



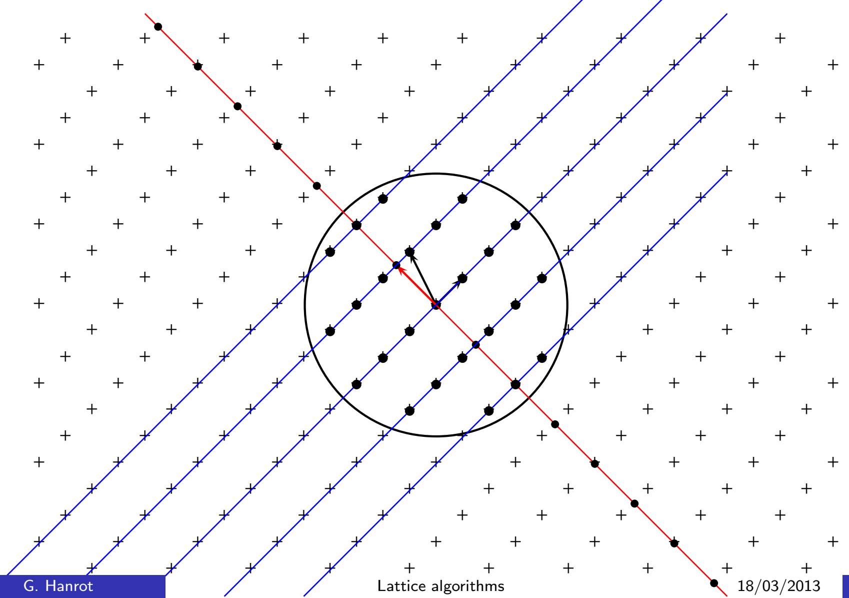
CVP, SVP – the KFP enumeration algorithm



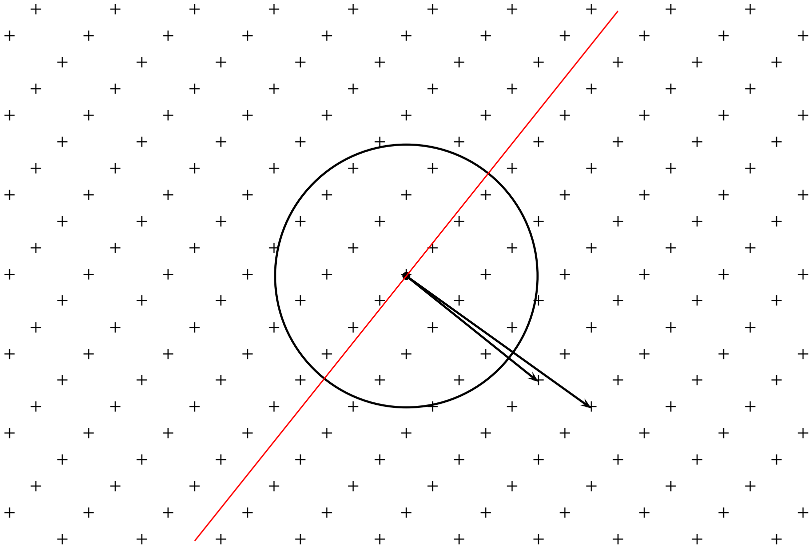
CVP, SVP – the KFP enumeration algorithm



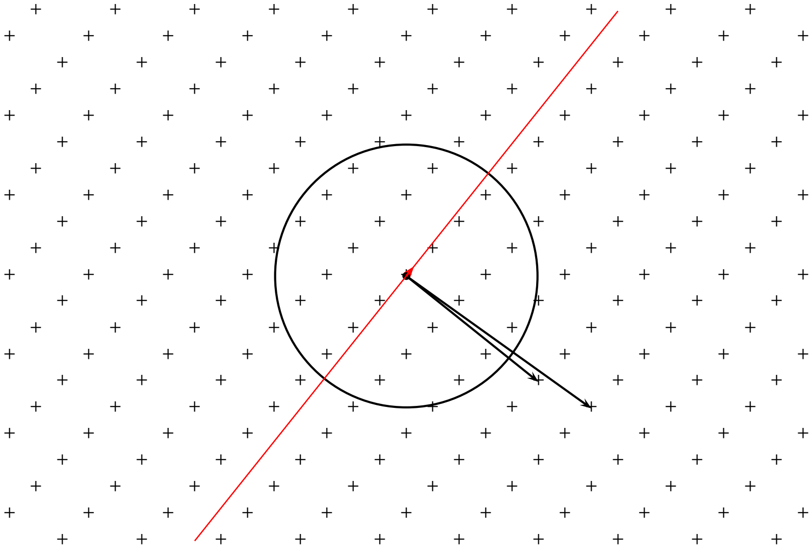
CVP, SVP – the KFP enumeration algorithm



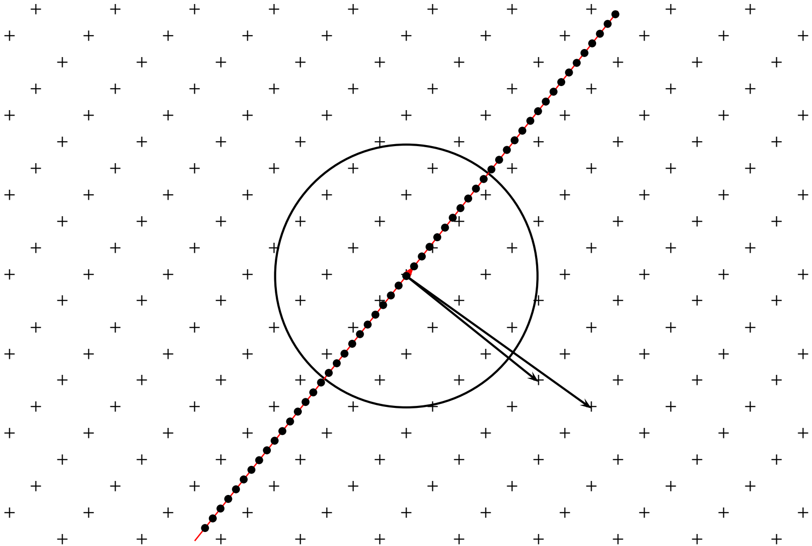
CVP, SVP – The KFP enumeration algorithm (II)



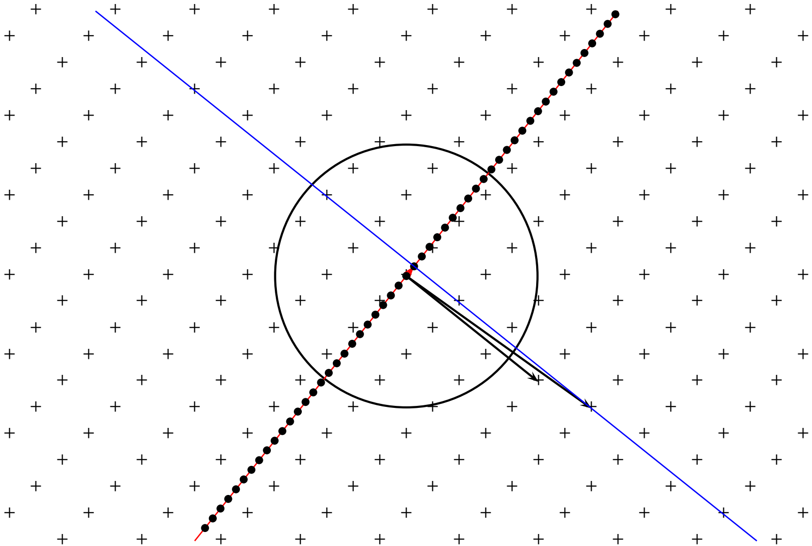
CVP, SVP – The KFP enumeration algorithm (II)



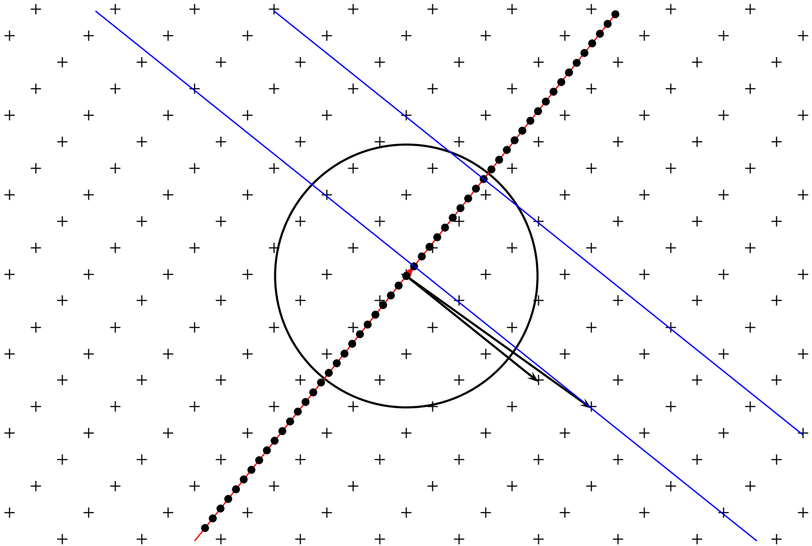
CVP, SVP – The KFP enumeration algorithm (II)



CVP, SVP – The KFP enumeration algorithm (II)



CVP, SVP – The KFP enumeration algorithm (II)



Kannan's improvement

- The shorter the basis vectors, the faster the enumeration.
- Kannan pre-processes the basis by performing enumerations in lower dimensions ;
- quasi-HKZ reduced basis as input to SVP;

$\Rightarrow$ Recursive process, using SVP solver in dim $n-1$.

- Complexity $n^{n/(2e)+o(n)}$ [HaSt07]
  (upper and lower worst-case bound).

KFP: the most practical SVP & CVP solver

- Basis is pre-processed before enumeration [Kannan83]
- Computations rely on floating-point arithmetic [PuSt08]
- The tree search can be parallelized [DHPS10]
- The choice of bound A can be optimized
- The tree search can be pruned (heuristic) [ScEu91,GaNgRe10]:

$$\forall i, \quad \sum_{j \geq i} (x_j - t_j + \sum_{k>j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 \leq p_i \cdot A,$$

where $1 \geq p_1 \geq \ldots \geq p_n > 0$.

Practical limits (a few days on a modern processor):

- For a guaranteed answer: $n \approx 70$.
- If heuristic answer suffices: $n \approx 110$.

KFP: the most practical SVP & CVP solver

- Basis is pre-processed before enumeration [Kannan83]
- Computations rely on floating-point arithmetic [PuSt08]
- The tree search can be parallelized [DHPS10]
- The choice of bound A can be optimized
- The tree search can be pruned (heuristic) [ScEu91,GaNgRe10]:

$$\forall i, \quad \sum_{j \geq i} (x_j - t_j + \sum_{k>j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 \leq p_i \cdot A,$$

  where $1 \geq p_1 \geq \ldots \geq p_n > 0$.

Practical limits (a few days on a modern processor):

- For a guaranteed answer: $n \approx 70$.
- If heuristic answer suffices: $n \approx 110$.

KFP: the most practical SVP & CVP solver

- Basis is pre-processed before enumeration [Kannan83]
- Computations rely on floating-point arithmetic [PuSt08]
- The tree search can be parallelized [DHPS10]
- The choice of bound A can be optimized
- The tree search can be pruned (heuristic) [ScEu91,GaNgRe10]:

$$\forall i, \quad \sum_{j \geq i} (x_j - t_j + \sum_{k>j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 \leq p_i \cdot A,$$

where $1 \geq p_1 \geq \ldots \geq p_n > 0$.

Practical limits (a few days on a modern processor):

- For a guaranteed answer: $n \approx 70$.
- If heuristic answer suffices: $n \approx 110$.

KFP: the most practical SVP & CVP solver

- Basis is pre-processed before enumeration [Kannan83]
- Computations rely on floating-point arithmetic [PuSt08]
- The tree search can be parallelized [DHPS10]
- The choice of bound A can be optimized
- The tree search can be pruned (heuristic) [ScEu91,GaNgRe10]:

$$\forall i, \quad \sum_{j \geq i} (x_j - t_j + \sum_{k>j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 \leq p_i \cdot A,$$

where $1 \geq p_1 \geq \ldots \geq p_n > 0$.

Practical limits (a few days on a modern processor):

- For a guaranteed answer: $n \approx 70$.
- If heuristic answer suffices: $n \approx 110$.

KFP: the most practical SVP & CVP solver

- Basis is pre-processed before enumeration [Kannan83]
- Computations rely on floating-point arithmetic [PuSt08]
- The tree search can be parallelized [DHPS10]
- The choice of bound A can be optimized
- The tree search can be pruned (heuristic) [ScEu91,GaNgRe10]:

$$\forall i, \quad \sum_{j \geq i} (x_j - t_j + \sum_{k > j} \mu_{k,j} x_k)^2 \|\mathbf{b}_j^*\|^2 \leq p_i \cdot A,$$

where $1 \geq p_1 \geq \ldots \geq p_n > 0$.

Practical limits (a few days on a modern processor):

- For a guaranteed answer: $n \approx 70$.
- If heuristic answer suffices: $n \approx 110$.

The saturation principle

Kabatyansky & Levenshtein

Let $E \subseteq \mathbb{R}^n \setminus \mathbf{0}$. Assume that for any $\mathbf{u} \neq \mathbf{v}$ in E, the angle between $\mathbf{u}$ and $\mathbf{v}$ is $\geq \phi_0$. Then $|E| \leq 2^{cn+o(n)}$ for some $c(\phi_0)$.

For $\phi_0 = 60^\circ$, we obtain $|E| \leq 2^{0.401 \cdot n}$.

Consequence: If points belong to a ball and their pairwise distances are bounded from below, then their number is $2^{O(n)}$.

The saturation principle

Kabatyansky & Levenshtein

Let $E \subseteq \mathbb{R}^n \setminus \mathbf{0}$. Assume that for any $\mathbf{u} \neq \mathbf{v}$ in E, the angle between $\mathbf{u}$ and $\mathbf{v}$ is $\geq \phi_0$. Then $|E| \leq 2^{cn+o(n)}$ for some $c(\phi_0)$.

For $\phi_0 = 60^\circ$, we obtain $|E| \leq 2^{0.401 \cdot n}$.

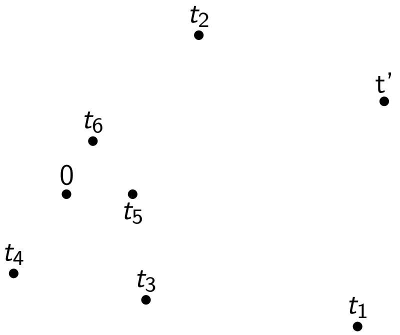
Consequence: If points belong to a ball and their pairwise distances are bounded from below, then their number is $2^{O(n)}$.

The ListSieve SVP Algorithm, heuristic version

1. Sample $\mathbf{t}_1, \ldots, \mathbf{t}_N \in L$.
2. For all $i \leq N$, if there is $j < i$ with $\|\mathbf{t}_i - \mathbf{t}_j\| < (1 - \frac{1}{n})\|\mathbf{t}_i\|$, replace $\mathbf{t}_i$ by $\mathbf{t}_i - \mathbf{t}_j$.
3. Return a shortest non-zero vector found.

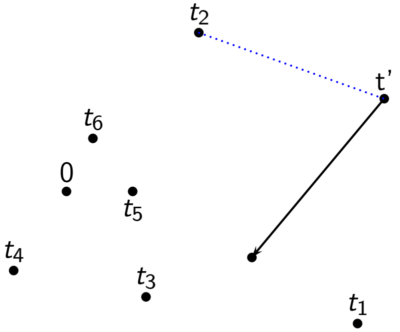
The ListSieve SVP Algorithm, heuristic version

1. Sample $\mathbf{t}_1, \ldots, \mathbf{t}_N \in L$.
2. For all $i \leq N$, if there is $j < i$ with $\|\mathbf{t}_i - \mathbf{t}_j\| < (1 - \frac{1}{n})\|\mathbf{t}_i\|$, replace $\mathbf{t}_i$ by $\mathbf{t}_i - \mathbf{t}_j$.
3. Return a shortest non-zero vector found.



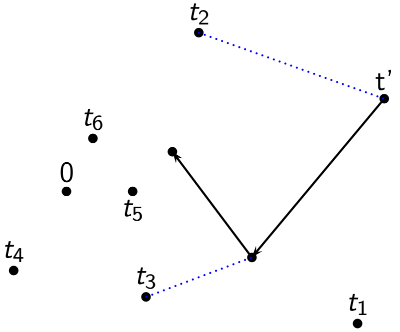
The ListSieve SVP Algorithm, heuristic version

1. Sample $\mathbf{t}_1, \ldots, \mathbf{t}_N \in L$.
2. For all $i \leq N$, if there is $j < i$ with $\|\mathbf{t}_i - \mathbf{t}_j\| < (1 - \frac{1}{n})\|\mathbf{t}_i\|$, replace $\mathbf{t}_i$ by $\mathbf{t}_i - \mathbf{t}_j$.
3. Return a shortest non-zero vector found.



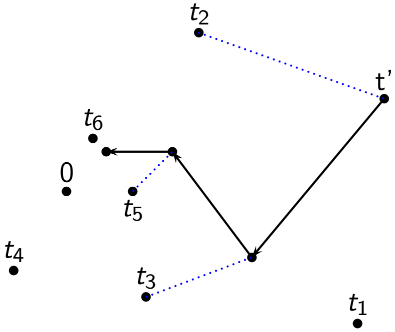
The ListSieve SVP Algorithm, heuristic version

1. Sample $\mathbf{t}_1, \ldots, \mathbf{t}_N \in L$.
2. For all $i \leq N$, if there is $j < i$ with $\|\mathbf{t}_i - \mathbf{t}_j\| < (1 - \frac{1}{n})\|\mathbf{t}_i\|$, replace $\mathbf{t}_i$ by $\mathbf{t}_i - \mathbf{t}_j$.
3. Return a shortest non-zero vector found.



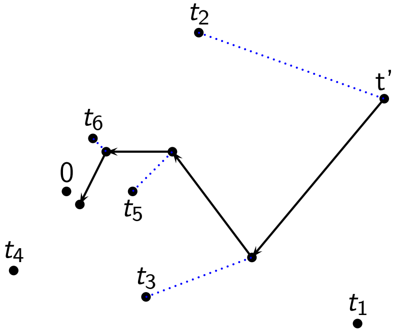
The `ListSieve` SVP Algorithm, heuristic version

1. Sample $\mathbf{t}_1, \ldots, \mathbf{t}_N \in L$.
2. For all $i \leq N$, if there is $j < i$ with $\|\mathbf{t}_i - \mathbf{t}_j\| < (1 - \frac{1}{n})\|\mathbf{t}_i\|$, replace $\mathbf{t}_i$ by $\mathbf{t}_i - \mathbf{t}_j$.
3. Return a shortest non-zero vector found.



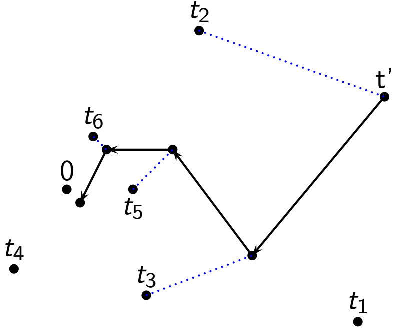
The ListSieve SVP Algorithm, heuristic version

1. Sample $\mathbf{t}_1, \ldots, \mathbf{t}_N \in L$.
2. For all $i \leq N$, if there is $j < i$ with $\|\mathbf{t}_i - \mathbf{t}_j\| < (1 - \frac{1}{n})\|\mathbf{t}_i\|$, replace $\mathbf{t}_i$ by $\mathbf{t}_i - \mathbf{t}_j$.
3. Return a shortest non-zero vector found.



The ListSieve SVP Algorithm, heuristic version

1. Sample $\mathbf{t}_1, \ldots, \mathbf{t}_N \in L$.
2. For all $i \leq N$, if there is $j < i$ with $\|\mathbf{t}_i - \mathbf{t}_j\| < (1 - \frac{1}{n})\|\mathbf{t}_i\|$, replace $\mathbf{t}_i$ by $\mathbf{t}_i - \mathbf{t}_j$.
3. Return a shortest non-zero vector found.



- Time complexity $\leq N^2$; Space complexity $\leq N^2$.
- Saturation principle $\Rightarrow$ at most $2^{c'n+o(n)}$ points at any time.

Correctness of ListSieve

How to ensure we get a shortest $\mathbf{s} \in L \setminus \mathbf{0}$?

- Principle: Hide the lattice to ListSieve by adding noise to the initial vectors: $\mathbf{t}_i \to \mathbf{t}_i + \mathbf{e}_i$.
- Once the vector has been dealt with: $\mathbf{t}_i^{end} \to \mathbf{t}_i^{end} - \mathbf{e}_i$.
- If noise $\approx \|\mathbf{s}\|$, then, with probability $\geq 2^{-\Omega(n)}$:

$$\mathbf{t}_i + \mathbf{e}_i \quad \text{could be} \quad \mathbf{t}_i' + \mathbf{e}_i' := (\mathbf{t}_i + \mathbf{s}) + (\mathbf{e}_i - \mathbf{s}).$$

- N is set large enough s.t. we get the same $\mathbf{t}^{end} \in L$ twice.

$\Rightarrow$ we get $\mathbf{t}^{end}$ and $\mathbf{t}^{end} + \mathbf{s}$ with high probability.

- We compute all pairwise differences and return the smallest.

Correctness of `ListSieve`

How to ensure we get a shortest $\mathbf{s} \in L \setminus \mathbf{0}$?

- Principle: Hide the lattice to `ListSieve` by adding noise to the initial vectors: $\mathbf{t}_i \to \mathbf{t}_i + \mathbf{e}_i$.
- Once the vector has been dealt with: $\mathbf{t}_i^{end} \to \mathbf{t}_i^{end} - \mathbf{e}_i$.
- If noise $\approx \|\mathbf{s}\|$, then, with probability $\geq 2^{-\Omega(n)}$:

$$\mathbf{t}_i + \mathbf{e}_i \text{ could be } \mathbf{t}'_i + \mathbf{e}'_i := (\mathbf{t}_i + \mathbf{s}) + (\mathbf{e}_i - \mathbf{s}).$$

- N is set large enough s.t. we get the same $\mathbf{t}^{end} \in L$ twice.

$\Rightarrow$ we get $\mathbf{t}^{end}$ and $\mathbf{t}^{end} + \mathbf{s}$ with high probability.

- We compute all pairwise differences and return the smallest.

Correctness of ListSieve

How to ensure we get a shortest $\mathbf{s} \in L \setminus \mathbf{0}$?

- Principle: Hide the lattice to ListSieve by adding noise to the initial vectors: $\mathbf{t}_i \to \mathbf{t}_i + \mathbf{e}_i$.
- Once the vector has been dealt with: $\mathbf{t}_i^{end} \to \mathbf{t}_i^{end} - \mathbf{e}_i$.
- If noise $\approx \|\mathbf{s}\|$, then, with probability $\geq 2^{-\Omega(n)}$:

$$\mathbf{t}_i + \mathbf{e}_i \text{ could be } \mathbf{t}_i' + \mathbf{e}_i' := (\mathbf{t}_i + \mathbf{s}) + (\mathbf{e}_i - \mathbf{s}).$$

- N is set large enough s.t. we get the same $\mathbf{t}^{end} \in L$ twice.

$\Rightarrow$ we get $\mathbf{t}^{end}$ and $\mathbf{t}^{end} + \mathbf{s}$ with high probability.

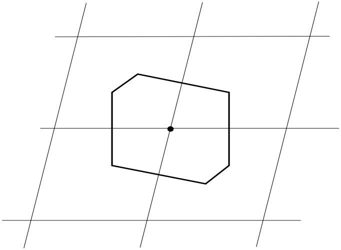
- We compute all pairwise differences and return the smallest.

Plan of the talk

- Reminders and context
- The KFP enumeration-based algorithm
- Saturating the space: The AKS solver and its descendants
- **Using the Voronoi cell: the Micciancio-Voulgaris algorithm.**
    - D. Micciancio & P. Voulgaris: `A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations`, STOC'10

The Voronoi cell of a lattice

$$\mathcal{V}(L) = \{\mathbf{x} \in \mathbb{R}^n : \forall \mathbf{b} \in L \setminus \mathbf{0}, \|\mathbf{x} - \mathbf{b}\| > \|\mathbf{x}\|\}.$$



The relevant vectors

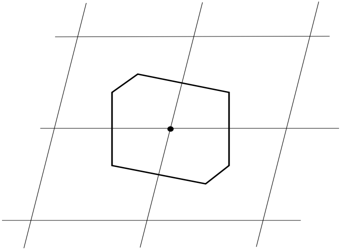
Let $(\mathbf{v}_i)_i$ be s.t. $\pm\mathbf{v}_i$ are the unique minima of a non-zero coset of $L/2L$. Then $\mathcal{V}(L) = \{\mathbf{x} \in \mathbb{R}^n : \forall i, \|\mathbf{v}_i - \mathbf{x}\| > \|\mathbf{x}\|\}$. Furthermore, these $\mathbf{v}_i$ are the smallest such set.

- A coset of $L/2L$ is of the form $(\sum_i \varepsilon_i \mathbf{b}_i) + 2L$ with $\varepsilon_i \in \{0, 1\}$.

$\Rightarrow$ $\mathcal{V}(L)$ can be described in space $\leq 2^n$.

The Voronoi cell of a lattice

$$\mathcal{V}(L) = \{\mathbf{x} \in \mathbb{R}^n : \forall \mathbf{b} \in L \setminus \mathbf{0}, \|\mathbf{x} - \mathbf{b}\| > \|\mathbf{x}\|\}.$$



The relevant vectors

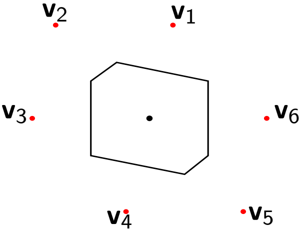
Let $(\mathbf{v}_i)_i$ be s.t. $\pm\mathbf{v}_i$ are the unique minima of a non-zero coset of $L/2L$. Then $\mathcal{V}(L) = \{\mathbf{x} \in \mathbb{R}^n : \forall i, \|\mathbf{v}_i - \mathbf{x}\| > \|\mathbf{x}\|\}$. Furthermore, these $\mathbf{v}_i$ are the smallest such set.

- A coset of $L/2L$ is of the form $(\sum_i \varepsilon_i \mathbf{b}_i) + 2L$ with $\varepsilon_i \in \{0, 1\}$.

$\Rightarrow$ $\mathcal{V}(L)$ can be described in space $\leq 2^n$.

The Voronoi cell of a lattice

$$\mathcal{V}(L) = \{\mathbf{x} \in \mathbb{R}^n : \forall \mathbf{b} \in L \setminus \mathbf{0}, \|\mathbf{x} - \mathbf{b}\| > \|\mathbf{x}\|\}.$$



The relevant vectors

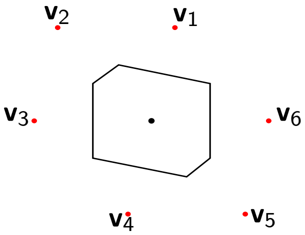
Let $(\mathbf{v}_i)_i$ be s.t. $\pm\mathbf{v}_i$ are the unique minima of a non-zero coset of $L/2L$. Then $\mathcal{V}(L) = \{\mathbf{x} \in \mathbb{R}^n : \forall i, \|\mathbf{v}_i - \mathbf{x}\| > \|\mathbf{x}\|\}$. Furthermore, these $\mathbf{v}_i$ are the smallest such set.

- A coset of $L/2L$ is of the form $(\sum_i \varepsilon_i \mathbf{b}_i) + 2L$ with $\varepsilon_i \in \{0, 1\}$.

$\Rightarrow$ $\mathcal{V}(L)$ can be described in space $\leq 2^n$.

The Voronoi cell of a lattice

$$\mathcal{V}(L) = \{\mathbf{x} \in \mathbb{R}^n : \forall \mathbf{b} \in L \setminus \mathbf{0}, \|\mathbf{x} - \mathbf{b}\| > \|\mathbf{x}\|\}.$$



The relevant vectors

Let $(\mathbf{v}_i)_i$ be s.t. $\pm\mathbf{v}_i$ are the unique minima of a non-zero coset of $L/2L$. Then $\mathcal{V}(L) = \{\mathbf{x} \in \mathbb{R}^n : \forall i, \|\mathbf{v}_i - \mathbf{x}\| > \|\mathbf{x}\|\}$. Furthermore, these $\mathbf{v}_i$ are the smallest such set.

- A coset of $L/2L$ is of the form $(\sum_i \varepsilon_i \mathbf{b}_i) + 2L$ with $\varepsilon_i \in \{0, 1\}$.

$\Rightarrow$ $\mathcal{V}(L)$ can be described in space $\leq 2^n$.

Solving SVP & CVP with the Voronoi cell

- SVP: A shortest non-zero vector is a relevant vector.
- CVP: translate $\mathbf{t}$ by a $\mathbf{b} \in L$ to map it to $\overline{\mathcal{V}}(L)$.
- It suffices to be able to do it when $\mathbf{t} \in 2\overline{\mathcal{V}}(L)$.

$\mathrm{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$:

- $\mathcal{C}(\mathbf{v}_i)$: cone of apex $\mathbf{0}$ and base the corresponding facet of $\overline{\mathcal{V}}$.
- While $\mathbf{t} \notin \overline{\mathcal{V}}$: Find i s.t. $\mathbf{t} \in \mathcal{C}(\mathbf{v}_i)$; Subtract $\mathbf{v}_i$ from $\mathbf{t}$.

Solving SVP & CVP with the Voronoi cell

- SVP: A shortest non-zero vector is a relevant vector.
- CVP: translate $\mathbf{t}$ by a $\mathbf{b} \in L$ to map it to $\overline{\mathcal{V}}(L)$.
- It suffices to be able to do it when $\mathbf{t} \in 2\overline{\mathcal{V}}(L)$.

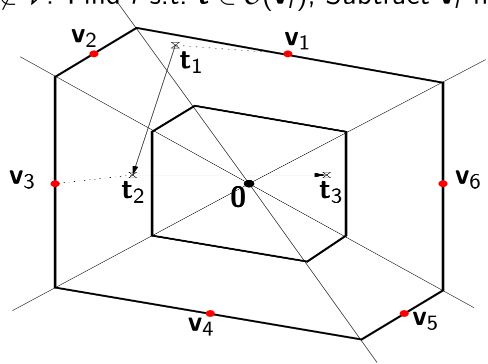
$\mathrm{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$:

- $\mathcal{C}(\mathbf{v}_i)$: cone of apex $\mathbf{0}$ and base the corresponding facet of $\overline{\mathcal{V}}$.
- While $\mathbf{t} \notin \overline{\mathcal{V}}$: Find i s.t. $\mathbf{t} \in \mathcal{C}(\mathbf{v}_i)$; Subtract $\mathbf{v}_i$ from $\mathbf{t}$.

Solving SVP & CVP with the Voronoi cell

$\mathrm{CVP}_{2\overline{\mathcal{V}} \to \overline{\mathcal{V}}}$:

- $\mathcal{C}(\mathbf{v}_i)$: cone of apex $\mathbf{0}$ and base the corresponding facet of $\overline{\mathcal{V}}$.
- While $\mathbf{t} \notin \overline{\mathcal{V}}$: Find i s.t. $\mathbf{t} \in \mathcal{C}(\mathbf{v}_i)$; Subtract $\mathbf{v}_i$ from $\mathbf{t}$.

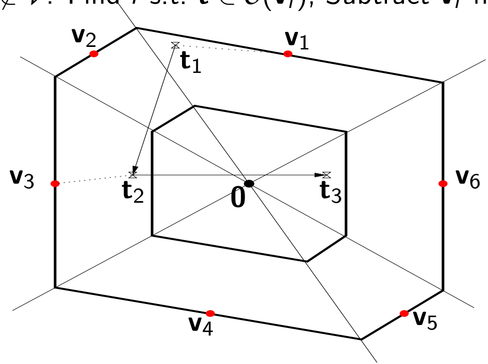


- Decreasing walk in $(\mathbf{t}_1 + L) \bmod 2L$.

Solving SVP & CVP with the Voronoi cell

$\mathrm{CVP}_{2\overline{\mathcal{V}}\to\overline{\mathcal{V}}}$:

- $\mathcal{C}(\mathbf{v}_i)$: cone of apex $\mathbf{0}$ and base the corresponding facet of $\overline{\mathcal{V}}$.
- While $\mathbf{t} \notin \overline{\mathcal{V}}$: Find i s.t. $\mathbf{t} \in \mathcal{C}(\mathbf{v}_i)$; Subtract $\mathbf{v}_i$ from $\mathbf{t}$.



- Decreasing walk in $(\mathbf{t}_1 + L) \bmod 2L$.

Computing the Voronoi cell

- A relevant vector $\mathbf{v} \in L$ is a shortest vector in a coset of $L/2L$.
- It suffices to solve $2^n - 1$ instances of CVP:

$$\text{Lattice: } 2L; \quad \text{Target } \mathbf{t} = \sum_i \varepsilon_i \mathbf{b}_i.$$

- But to solve CVP, we use the Voronoi cell...

Trick: CVP in dim n can be solved with $2^{o(n)}$ CVP's in dim $n-1$.

$$\mathrm{CVP}(\mathbf{t}, 2L) = \min_{\|\cdot\|} \left\{ \mathrm{CVP}(\mathbf{t} + x_n(2\mathbf{b}_n), L^-) \ : \ x_n \in \mathbb{Z} \right\},$$

- $L^- = 2L \cap \mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{n-1})$.
- x_n may be restricted to a small set.

Overall: Intertwined Voronoi/CVP in increasing dimensions.

Computing the Voronoi cell

- A relevant vector $\mathbf{v} \in L$ is a shortest vector in a coset of $L/2L$.
- It suffices to solve $2^n - 1$ instances of CVP:

$$\text{Lattice: } 2L; \quad \text{Target } \mathbf{t} = \sum_i \varepsilon_i \mathbf{b}_i.$$

- But to solve CVP, we use the Voronoi cell...

Trick: CVP in dim n can be solved with $2^{o(n)}$ CVP's in dim $n-1$.

$$\mathrm{CVP}(\mathbf{t}, 2L) = \min_{\|\cdot\|} \left\{ \mathrm{CVP}(\mathbf{t} + x_n(2\mathbf{b}_n), L^-) \ : \ x_n \in \mathbb{Z} \right\},$$

- $L^- = 2L \cap \mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{n-1})$.
- x_n may be restricted to a small set.

Overall: Intertwined Voronoi/CVP in increasing dimensions.

Computing the Voronoi cell

- A relevant vector $\mathbf{v} \in L$ is a shortest vector in a coset of $L/2L$.
- It suffices to solve $2^n - 1$ instances of CVP:

$$\text{Lattice: } 2L; \quad \text{Target } \mathbf{t} = \sum_i \varepsilon_i \mathbf{b}_i.$$

- But to solve CVP, we use the Voronoi cell...

Trick: CVP in dim n can be solved with $2^{o(n)}$ CVP's in dim $n-1$.

$$\mathrm{CVP}(\mathbf{t}, 2L) = \min_{\|\cdot\|} \left\{ \mathrm{CVP}(\mathbf{t} + x_n(2\mathbf{b}_n), L^-) \ : \ x_n \in \mathbb{Z} \right\},$$

- $L^- = 2L \cap \mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{n-1})$.
- x_n may be restricted to a small set.

Overall: Intertwined Voronoi/CVP in increasing dimensions.

Computing the Voronoi cell

- A relevant vector $\mathbf{v} \in L$ is a shortest vector in a coset of $L/2L$.
- It suffices to solve $2^n - 1$ instances of CVP:

$$\text{Lattice: } 2L; \quad \text{Target } \mathbf{t} = \sum_i \varepsilon_i \mathbf{b}_i.$$

- But to solve CVP, we use the Voronoi cell...

Trick: CVP in dim n can be solved with $2^{o(n)}$ CVP's in dim $n-1$.

$$\mathrm{CVP}(\mathbf{t}, 2L) = \min_{\|\cdot\|} \left\{ \mathrm{CVP}(\mathbf{t} + x_n(2\mathbf{b}_n), L^-) \ : \ x_n \in \mathbb{Z} \right\},$$

- $L^- = 2L \cap \mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{n-1})$.
- x_n may be restricted to a small set.

Overall: Intertwined Voronoi/CVP in increasing dimensions.

III. Application : Coppersmith's method

Lattice algorithms – Coppersmith's method

- $P \in \mathbb{Z}[X]$ of degree d, **monic**;
- N an integer;
- want to solve $P(x) = 0 \bmod N$;
- easy
  - if N prime (factoring mod p);
  - or factors of N known (CRT + Hensel lifting);
- hard in general: $\deg P = 2 \Leftrightarrow$ factoring N.
- look for small solutions x.
- Why small? Allow to lift the problem to $\mathbb{Z}$ (easy again).

Lattice algorithms – Coppersmith's method

- $P \in \mathbb{Z}[X]$ of degree d, **monic**;
- N an integer;
- want to solve $P(x) = 0 \bmod N$;
- easy
  - if N prime (factoring mod p);
  - or factors of N known (CRT + Hensel lifting);
- hard in general: $\deg P = 2 \Leftrightarrow$ factoring N.
- look for small solutions x.
- Why small? Allow to lift the problem to $\mathbb{Z}$ (easy again).

Lattice algorithms – Coppersmith's method

- $P \in \mathbb{Z}[X]$ of degree d, **monic**;
- N an integer;
- want to solve $P(x) = 0 \bmod N$;
- easy
  - if N prime (factoring mod p);
  - or factors of N known (CRT + Hensel lifting);
- hard in general: $\deg P = 2 \Leftrightarrow$ factoring N.
- look for small solutions x.
- Why small? Allow to lift the problem to $\mathbb{Z}$ (easy again).

Lattice algorithms – Coppersmith's method

- $P \in \mathbb{Z}[X]$ of degree d, **monic**;
- N an integer;
- want to solve $P(x) = 0 \bmod N$;
- easy
  - if N prime (factoring mod p);
  - or factors of N known (CRT + Hensel lifting);
- hard in general: $\deg P = 2 \Leftrightarrow$ factoring N.
- look for small solutions x.
- Why small? Allow to lift the problem to $\mathbb{Z}$ (easy again).

Lattice algorithms – Coppersmith's method

- $P \in \mathbb{Z}[X]$ of degree d, **monic**;
- N an integer;
- want to solve $P(x) = 0 \bmod N$;
- easy
  - if N prime (factoring mod p);
  - or factors of N known (CRT + Hensel lifting);
- hard in general: $\deg P = 2 \Leftrightarrow$ factoring N.
- look for small solutions x.
- Why small? Allow to lift the problem to $\mathbb{Z}$ (easy again).

Lattice algorithms – Coppersmith's method

- $P \in \mathbb{Z}[X]$ of degree d, **monic**;
- N an integer;
- want to solve $P(x) = 0 \bmod N$;
- easy
  - if N prime (factoring mod p);
  - or factors of N known (CRT + Hensel lifting);
- hard in general: $\deg P = 2 \Leftrightarrow$ factoring N.
- look for small solutions x.
- Why small? Allow to lift the problem to $\mathbb{Z}$ (easy again).

Lattice algorithms – Coppersmith's method

- $P \in \mathbb{Z}[X]$ of degree d, **monic**;
- N an integer;
- want to solve $P(x) = 0 \bmod N$;
- easy
  - if N prime (factoring mod p);
  - or factors of N known (CRT + Hensel lifting);
- hard in general: $\deg P = 2 \Leftrightarrow$ factoring N.
- look for small solutions x.
- Why small? Allow to lift the problem to $\mathbb{Z}$ (easy again).

Lattice algorithms – Coppersmith's method

- $P \in \mathbb{Z}[X]$ of degree d, **monic**;
- N an integer;
- want to solve $P(x) = 0 \bmod N$;
- easy
  - if N prime (factoring mod p);
  - or factors of N known (CRT + Hensel lifting);
- hard in general: $\deg P = 2 \Leftrightarrow$ factoring N.
- look for small solutions x.
- Why small? Allow to lift the problem to $\mathbb{Z}$ (easy again).

Lattice algorithms – Coppersmith's method (outline)

Simple example :

- RSA: $N = pq$, d, e integers such that $d \cdot e = (p-1)(q-1)$;
- N, e public;
- Encryption is $x \mapsto x^e$, decryption is $x \mapsto x^d$.

If $|x| < N^{1/e}$, can decrypt from $c := x^e \bmod N$:

- $|x|^e < N \Rightarrow x^e = c$ in $\mathbb{Z}$;
- Extract e-th root in $\mathbb{Z}$ (eg. Newton's method).

Key argument: if

- $|x| \leq X$ (X small),
- and $|P|(X) < N$ (P small),

then $P(x) = 0 \bmod N \Rightarrow P(x) = 0$ in $\mathbb{Z}$.

Lattice algorithms – Coppersmith's method (outline)

Simple example :

- RSA: $N = pq$, d, e integers such that $d \cdot e = (p-1)(q-1)$;
- N, e public;
- Encryption is $x \mapsto x^e$, decryption is $x \mapsto x^d$.

If $|x| < N^{1/e}$, can decrypt from $c := x^e \bmod N$:

- $|x|^e < N \Rightarrow x^e = c$ in $\mathbb{Z}$;
- Extract e-th root in $\mathbb{Z}$ (eg. Newton's method).

Key argument: if

- $|x| \leq X$ (X small),
- and $|P|(X) < N$ (P small),

then $P(x) = 0 \bmod N \Rightarrow P(x) = 0$ in $\mathbb{Z}$.

Lattice algorithms – Coppersmith's method (outline)

Simple example :

- RSA: $N = pq$, d, e integers such that $d \cdot e = (p-1)(q-1)$;
- N, e public;
- Encryption is $x \mapsto x^e$, decryption is $x \mapsto x^d$.

If $|x| < N^{1/e}$, can decrypt from $c := x^e \bmod N$:

- $|x|^e < N \Rightarrow x^e = c$ in $\mathbb{Z}$;
- Extract e-th root in $\mathbb{Z}$ (eg. Newton's method).

Key argument: if

- $|x| \leq X$ (X small),
- and $|P|(X) < N$ (P small),

then $P(x) = 0 \bmod N \Rightarrow P(x) = 0$ in $\mathbb{Z}$.

Lattice algorithms – Coppersmith's method (outline)

Simple example :

- RSA: $N = pq$, d, e integers such that $d \cdot e = (p-1)(q-1)$;
- N, e public;
- Encryption is $x \mapsto x^e$, decryption is $x \mapsto x^d$.

If $|x| < N^{1/e}$, can decrypt from $c := x^e \bmod N$:

- $|x|^e < N \Rightarrow x^e = c$ in $\mathbb{Z}$;
- Extract e-th root in $\mathbb{Z}$ (eg. Newton's method).

Key argument: if

- $|x| \leq X$ (X small),
- and $|P|(X) < N$ (P small),

then $P(x) = 0 \bmod N \Rightarrow P(x) = 0$ in $\mathbb{Z}$.

Lattice algorithms – Coppersmith's method (outline)

Simple example :

- RSA: $N = pq$, d, e integers such that $d \cdot e = (p-1)(q-1)$;
- N, e public;
- Encryption is $x \mapsto x^e$, decryption is $x \mapsto x^d$.

If $|x| < N^{1/e}$, can decrypt from $c := x^e \bmod N$:

- $|x|^e < N \Rightarrow x^e = c$ in $\mathbb{Z}$;
- Extract e-th root in $\mathbb{Z}$ (eg. Newton's method).

Key argument: if

- $|x| \leq X$ (X small),
- and $|P|(X) < N$ (P small),

then $P(x) = 0 \bmod N \Rightarrow P(x) = 0$ in $\mathbb{Z}$.

Lattice algorithms – Coppersmith's method (outline)

First attempt: Girault, Toffin, Vallée / Hastad.

- Try to find small $Q = P + SN$;
- x, Q small $\Rightarrow Q(x) < N$
- $P(x) = 0 \bmod N \Rightarrow Q(x) = 0 \bmod N \Rightarrow Q(x) = 0$
- solve equations over the integers, easy.

Lattice algorithms – Coppersmith's method (outline)

First attempt: Girault, Toffin, Vallée / Hastad.

- Try to find small $Q = P + SN$;
- x, Q small $\Rightarrow Q(x) < N$
- $P(x) = 0 \bmod N \Rightarrow Q(x) = 0 \bmod N \Rightarrow Q(x) = 0$
- solve equations over the integers, easy.

Lattice algorithms – Coppersmith's method (outline)

L lattice generated by coefficient vectors of N, Nx, $\ldots$, Nx^{d-1}, P.

$$L = \begin{pmatrix} N & & & & & p_0 \\ & N & & & & p_1 \\ & & N & & & p_2 \\ & & & \ddots & & \vdots \\ & & & & N & p_{d-1} \\ & & & & & 1 \end{pmatrix}$$

- v short vector in $L \leftrightarrow Q = P + SN$ with small coefficients;
- Want very small high order coefficients, low order coefficient are less important.
- More precisely, $|x| \leq X \Rightarrow |Q(x)| \leq \sum_{j=0}^{d} q_j X^j$.
- $\Rightarrow$ want $Q(xX) = P(xX) + S(xX)N$ with small coefficients
- $\Rightarrow$ add weights to the lattice.

Lattice algorithms – Coppersmith's method (outline)

L lattice generated by coefficient vectors of N, Nx, ..., Nx^{d-1}, P.

$$L = \begin{pmatrix} N & & & & & p_0 \\ & N & & & & p_1 \\ & & N & & & p_2 \\ & & & \ddots & & \vdots \\ & & & & N & p_{d-1} \\ & & & & & 1 \end{pmatrix}$$

- v short vector in $L \leftrightarrow Q = P + SN$ with small coefficients;
- Want very small high order coefficients, low order coefficient are less important.
- More precisely, $|x| \leq X \Rightarrow |Q(x)| \leq \sum_{j=0}^{d} q_j X^j$.
- $\Rightarrow$ want $Q(xX) = P(xX) + S(xX)N$ with small coefficients
- $\Rightarrow$ add weights to the lattice.

Lattice algorithms – Coppersmith's method (outline)

L lattice generated by coefficient vectors of N, Nx, ..., Nx^{d-1}, P.

$$L = \begin{pmatrix} N & & & & & p_0 \\ & N & & & & p_1 \\ & & N & & & p_2 \\ & & & \ddots & & \vdots \\ & & & & N & p_{d-1} \\ & & & & & 1 \end{pmatrix}$$

- v short vector in $L \leftrightarrow Q = P + SN$ with small coefficients;
- Want very small high order coefficients, low order coefficient are less important.
- More precisely, $|x| \leq X \Rightarrow |Q(x)| \leq \sum_{j=0}^{d} q_j X^j$.
- $\Rightarrow$ want $Q(xX) = P(xX) + S(xX)N$ with small coefficients
- $\Rightarrow$ add weights to the lattice.

Lattice algorithms – Coppersmith's method (outline)

L lattice generated by coefficient vectors of N, Nx, $\ldots$, Nx^{d-1}, P.

$$L = \begin{pmatrix} N & & & & & p_0 \\ & N & & & & p_1 \\ & & N & & & p_2 \\ & & & \ddots & & \vdots \\ & & & & N & p_{d-1} \\ & & & & & 1 \end{pmatrix}$$

- v short vector in $L \leftrightarrow Q = P + SN$ with small coefficients;
- Want very small high order coefficients, low order coefficient are less important.
- More precisely, $|x| \le X \Rightarrow |Q(x)| \le \sum_{j=0}^{d} q_j X^j$.
- $\Rightarrow$ want $Q(xX) = P(xX) + S(xX)N$ with small coefficients
- $\Rightarrow$ add weights to the lattice.

Lattice algorithms – Coppersmith's method (outline)

L lattice generated by coefficient vectors of N, Nx, ..., Nx^{d-1}, P.

$$L = \begin{pmatrix} N & & & & & p_0 \\ & N & & & & p_1 \\ & & N & & & p_2 \\ & & & \ddots & & \vdots \\ & & & & N & p_{d-1} \\ & & & & & 1 \end{pmatrix}$$

- v short vector in $L \leftrightarrow Q = P + SN$ with small coefficients;
- Want very small high order coefficients, low order coefficient are less important.
- More precisely, $|x| \leq X \Rightarrow |Q(x)| \leq \sum_{j=0}^{d} q_j X^j$.
- $\Rightarrow$ want $Q(xX) = P(xX) + S(xX)N$ with small coefficients
- $\Rightarrow$ add weights to the lattice.

Lattice algorithms – Coppersmith's method (outline)

L lattice generated by coefficient vectors of N, Nx, ..., Nx^{d-1}, P.

$$L = \begin{pmatrix} N & & & & & p_0 \\ & N & & & & p_1 \\ & & N & & & p_2 \\ & & & \ddots & & \vdots \\ & & & & N & p_{d-1} \\ & & & & & 1 \end{pmatrix}$$

- v short vector in $L \leftrightarrow Q = P + SN$ with small coefficients;
- Want very small high order coefficients, low order coefficient are less important.
- More precisely, $|x| \leq X \Rightarrow |Q(x)| \leq \sum_{j=0}^{d} q_j X^j$.
- $\Rightarrow$ want $Q(xX) = P(xX) + S(xX)N$ with small coefficients
- $\Rightarrow$ add weights to the lattice.

Lattice algorithms – Coppersmith's method (outline)

L lattice generated by coefficient vectors of N, Nx, ..., Nx^{d-1}, P.

$$L = \begin{pmatrix} N & & & & & p_0 \\ & NX & & & & p_1 X \\ & & NX^2 & & & p_2 X^2 \\ & & & \ddots & & \vdots \\ & & & & NX^{d-1} & p_{d-1} X^{d-1} \\ & & & & & 1X^d \end{pmatrix}$$

- v short vector in $L \leftrightarrow Q = P + SN$ with small coefficients;
- Want very small high order coefficients, low order coefficient are less important.
- More precisely, $|x| \le X \Rightarrow |Q(x)| \le \sum_{j=0}^{d} q_j X^j$.
- $\Rightarrow$ want $Q(xX) = P(xX) + S(xX)N$ with small coefficients
- $\Rightarrow$ add weights to the lattice.

Lattice algorithms – Coppersmith's method (outline)

Analysis.

- $\mathrm{vol}(L) = \prod_{i=0}^{d-1}(NX^i) = N^d X^{d(d+1)/2}$;
- LLL returns a vector v with $\|v\|_2 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- Hence $\|v\|_1 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- ok if $2^{O(d)} N^{d/(d+1)} X^{d/2} < N$
- ... $X = O(N^{2/(d(d+1))})$.

Time = poly(d, $\log N$).

Lattice algorithms – Coppersmith's method (outline)

Analysis.

- $\mathrm{vol}(L) = \prod_{i=0}^{d-1}(NX^i) = N^d X^{d(d+1)/2}$;
- LLL returns a vector v with $\|v\|_2 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- Hence $\|v\|_1 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- ok if $2^{O(d)} N^{d/(d+1)} X^{d/2} < N$
- ... $X = O(N^{2/(d(d+1))})$.

Time = poly(d, log N).

Lattice algorithms – Coppersmith's method (outline)

Analysis.

- $\mathrm{vol}(L) = \prod_{i=0}^{d-1}(NX^i) = N^d X^{d(d+1)/2}$;
- LLL returns a vector v with $\|v\|_2 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- Hence $\|v\|_1 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- ok if $2^{O(d)} N^{d/(d+1)} X^{d/2} < N$
- ... $X = O(N^{2/(d(d+1))})$.

Time = poly(d, log N).

Lattice algorithms – Coppersmith's method (outline)

Analysis.

- $\mathrm{vol}(L) = \prod_{i=0}^{d-1}(NX^i) = N^d X^{d(d+1)/2}$;
- LLL returns a vector v with $\|v\|_2 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- Hence $\|v\|_1 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- ok if $2^{O(d)} N^{d/(d+1)} X^{d/2} < N$
- ... $X = O(N^{2/(d(d+1))})$.

Time $=$ poly$(d, \log N)$.

Lattice algorithms – Coppersmith's method (outline)

Analysis.

- $\mathrm{vol}(L) = \prod_{i=0}^{d-1}(NX^i) = N^d X^{d(d+1)/2}$;
- LLL returns a vector v with $\|v\|_2 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- Hence $\|v\|_1 \leq 2^{O(d)} N^{d/(d+1)} X^{d/2}$
- ok if $2^{O(d)} N^{d/(d+1)} X^{d/2} < N$
- ... $X = O(N^{2/(d(d+1))})$.

Time = poly(d, log N).

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method.

Key idea: use multiplicities (powers of P)!

- $P(x) = 0 \bmod N \Rightarrow P^i(x)N^{k-i} = 0 \bmod N^k$ for all i;
- Look for $Q = \sum_{i=0}^{k} P^i N^{k-i} R_i(X)$, $\deg R_i < d$, $\deg R_k \leq t$;
- s.t. $|Q|(X) < N^k$

Lattice L

- generated by coefficient vectors of $S_{ij} = (xX)^j P^i(xX) N^{k-i}$;
- dimension $\delta = dk + t + 1$;
- $\deg S_{ij} = id + j$;
- hence upper triangular matrix;

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method.

Key idea: use multiplicities (powers of P)!

- $P(x) = 0 \bmod N \Rightarrow P^i(x)N^{k-i} = 0 \bmod N^k$ for all i;
- Look for $Q = \sum_{i=0}^{k} P^i N^{k-i} R_i(X)$, $\deg R_i < d$, $\deg R_k \leq t$;
- s.t. $|Q|(X) < N^k$

Lattice L

- generated by coefficient vectors of $S_{ij} = (xX)^j P^i(xX) N^{k-i}$;
- dimension $\delta = dk + t + 1$;
- $\deg S_{ij} = id + j$;
- hence upper triangular matrix;

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method.

Key idea: use multiplicities (powers of P)!

- $P(x) = 0 \bmod N \Rightarrow P^i(x)N^{k-i} = 0 \bmod N^k$ for all i;
- Look for $Q = \sum_{i=0}^{k} P^i N^{k-i} R_i(X)$, deg $R_i < d$, deg $R_k \leq t$;
- s.t.$|Q|(X) < N^k$

Lattice L

- generated by coefficient vectors of $S_{ij} = (xX)^j P^i(xX) N^{k-i}$;
- dimension $\delta = dk + t + 1$;
- deg $S_{ij} = id + j$;
- hence upper triangular matrix;

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method.

Key idea: use multiplicities (powers of P)!

- $P(x) = 0 \bmod N \Rightarrow P^i(x)N^{k-i} = 0 \bmod N^k$ for all i;
- Look for $Q = \sum_{i=0}^{k} P^i N^{k-i} R_i(X)$, $\deg R_i < d$, $\deg R_k \leq t$;
- s.t.$|Q|(X) < N^k$

Lattice L

- generated by coefficient vectors of $S_{ij} = (xX)^j P^i(xX)N^{k-i}$;
- dimension $\delta = dk + t + 1$;
- $\deg S_{ij} = id + j$;
- hence upper triangular matrix;

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method.

Key idea: use multiplicities (powers of P)!

- $P(x) = 0 \bmod N \Rightarrow P^i(x)N^{k-i} = 0 \bmod N^k$ for all i;
- Look for $Q = \sum_{i=0}^{k} P^i N^{k-i} R_i(X)$, $\deg R_i < d$, $\deg R_k \leq t$;
- s.t.$|Q|(X) < N^k$

Lattice L

- generated by coefficient vectors of $S_{ij} = (xX)^j P^i(xX)N^{k-i}$;
- dimension $\delta = dk + t + 1$;
- $\deg S_{ij} = id + j$;
- hence upper triangular matrix;

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method.

Key idea: use multiplicities (powers of P)!

- $P(x) = 0 \bmod N \Rightarrow P^i(x)N^{k-i} = 0 \bmod N^k$ for all i;
- Look for $Q = \sum_{i=0}^{k} P^i N^{k-i} R_i(X)$, $\deg R_i < d$, $\deg R_k \leq t$;
- s.t.$|Q|(X) < N^k$

Lattice L

- generated by coefficient vectors of $S_{ij} = (xX)^j P^i(xX)N^{k-i}$;
- dimension $\delta = dk + t + 1$;
- $\deg S_{ij} = id + j$;
- hence upper triangular matrix;

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method.

Key idea: use multiplicities (powers of P)!

- $P(x) = 0 \bmod N \Rightarrow P^i(x)N^{k-i} = 0 \bmod N^k$ for all i;
- Look for $Q = \sum_{i=0}^{k} P^i N^{k-i} R_i(X)$, $\deg R_i < d$, $\deg R_k \le t$;
- s.t.$|Q|(X) < N^k$

Lattice L

- generated by coefficient vectors of $S_{ij} = (xX)^j P^i(xX)N^{k-i}$;
- dimension $\delta = dk + t + 1$;
- $\deg S_{ij} = id + j$;
- hence upper triangular matrix;

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method (1996).

$$\begin{aligned}\mathrm{diagonal}(L) &= (N^k, N^k X, \ldots, N^k X^{d-1},\\ &\quad N^{k-1}X^d, N^{k-1}X^{d+1}, \ldots, N^{k-1}X^{2d-1}\\ &\quad , \ldots,\\ &\quad X^{kd}, X^{kd+1}, \ldots, X^{kd+t})\end{aligned}$$

$$\mathrm{vol} L = X^{\sum_{i=0}^{\delta-1} i} N^{d \sum_{i=0}^{k}(k-i)}$$

Lattice algorithms – Coppersmith's method (outline)

Coppersmith's method (1996).

$$
\begin{aligned}
\mathrm{diagonal}(L) \quad = \quad & (N^k, N^k X, \dots, N^k X^{d-1}, \\
& N^{k-1}X^d, N^{k-1}X^{d+1}, \dots, N^{k-1}X^{2d-1} \\
& , \dots, \\
& X^{kd}, X^{kd+1}, \dots, X^{kd+t})
\end{aligned}
$$

$$
\mathrm{vol} L = X^{\sum_{i=0}^{\delta-1} i} N^{d \sum_{i=0}^{k} (k-i)}
$$

Lattice algorithms – Coppersmith's method

$$\mathrm{vol}\, L = X^{\delta(\delta-1)/2} N^{dk(k+1)/2}.$$

- Want

$$2^{O(\delta)} X^{(\delta-1)/2} N^{dk(k+1)/2\delta} \leq N^k,$$

- ie.

$$X = O(N^{k(2\delta-d(k+1))/\delta(\delta-1)}).$$

- $\delta \approx dk$, $k \to \infty$,

$$2k(2\delta - d(k+1))/\delta(\delta-1) = 1/d - O(1/k).$$

Take $k = O(\log N)$,

- get $X = O(N^{1/d})$;
- with polynomial lattice dimension.

Lattice algorithms – Coppersmith's method

$$\mathrm{vol} L = X^{\delta(\delta-1)/2} N^{dk(k+1)/2}.$$

- Want

$$2^{O(\delta)} X^{(\delta-1)/2} N^{dk(k+1)/2\delta} \leq N^k,$$

- ie.

$$X = O(N^{k(2\delta - d(k+1))/\delta(\delta-1)}).$$

- $\delta \approx dk$, $k \to \infty$,

$$2k(2\delta - d(k+1))/\delta(\delta-1) = 1/d - O(1/k).$$

Take $k = O(\log N)$,

- get $X = O(N^{1/d})$;
- with polynomial lattice dimension.

Lattice algorithms – Coppersmith's method

$$\mathrm{vol} L = X^{\delta(\delta-1)/2} N^{dk(k+1)/2}.$$

- Want
$$2^{O(\delta)} X^{(\delta-1)/2} N^{dk(k+1)/2\delta} \leq N^k,$$

- ie.
$$X = O(N^{k(2\delta-d(k+1))/\delta(\delta-1)}).$$

- $\delta \approx dk$, $k \to \infty$,
$$2k(2\delta - d(k+1))/\delta(\delta-1) = 1/d - O(1/k).$$

Take $k = O(\log N)$,

- get $X = O(N^{1/d})$;
- with polynomial lattice dimension.

Lattice algorithms – Coppersmith's method, gcd extension

Solve $|x| \leq X$, $\gcd(P(x), N) > N^{\beta}$.

- Same strategy;
- If $|x| \leq X, \gcd(Q(x), N^k) > N^{k\beta}$ and $|Q|(X) < N^{k\beta}$
- ... then $Q(x) = 0$.

Lattice algorithms – Coppersmith's method, gcd extension

Solve $|x| \leq X$, $\gcd(P(x), N) > N^{\beta}$.

- Same strategy;
- If $|x| \leq X, \gcd(Q(x), N^k) > N^{k\beta}$ and $|Q|(X) < N^{k\beta}$
- ... then $Q(x) = 0$.

Lattice algorithms – Coppersmith's method, summary

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{1/d}, N^{1/d}]$ such that $P(x) = 0 \bmod N$.

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{\beta^2/d}, N^{\beta^2/d}]$ such that $\gcd(P(x), N) > N^{\beta}$.

Optimality:

- take $P(x) = x^d$, $N = p^d$.
- Solutions to $P(x) = 0 \bmod N$ are kp, $k \in [0, p^{d-1}[$.
- Number $= X/N^{1/d}$.

Lattice algorithms – Coppersmith's method, summary

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{1/d}, N^{1/d}]$ such that $P(x) = 0 \bmod N$.

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{\beta^2/d}, N^{\beta^2/d}]$ such that $\gcd(P(x), N) > N^{\beta}$.

Optimality:

- take $P(x) = x^d$, $N = p^d$.
- Solutions to $P(x) = 0 \bmod N$ are kp, $k \in [0, p^{d-1}[$.
- Number $= X/N^{1/d}$.

Lattice algorithms – Coppersmith's method, summary

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{1/d}, N^{1/d}]$ such that $P(x) = 0 \bmod N$.

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{\beta^2/d}, N^{\beta^2/d}]$ such that $\gcd(P(x), N) > N^{\beta}$.

Optimality:

- take $P(x) = x^d$, $N = p^d$.
- Solutions to $P(x) = 0 \bmod N$ are kp, $k \in [0, p^{d-1}[$.
- Number $= X/N^{1/d}$.

Lattice algorithms – Coppersmith's method, summary

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{1/d}, N^{1/d}]$ such that $P(x) = 0 \bmod N$.

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{\beta^2/d}, N^{\beta^2/d}]$ such that $\gcd(P(x), N) > N^\beta$.

Optimality:

- take $P(x) = x^d$, $N = p^d$.
- Solutions to $P(x) = 0 \bmod N$ are kp, $k \in [0, p^{d-1}[$.
- Number $= X/N^{1/d}$.

Lattice algorithms – Coppersmith's method, summary

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{1/d}, N^{1/d}]$ such that $P(x) = 0 \bmod N$.

Theorem. (Coppersmith). There is a polynomial-time algorithm which on input P, N, returns all $x \in [-N^{\beta^2/d}, N^{\beta^2/d}]$ such that $\gcd(P(x), N) > N^{\beta}$.

Optimality:

- take $P(x) = x^d$, $N = p^d$.
- Solutions to $P(x) = 0 \bmod N$ are kp, $k \in [0, p^{d-1}[$.
- Number $= X/N^{1/d}$.

Lattice algorithms – Coppersmith's method, application 1

Factoring with high bits known.

- Assume $N = pq$ RSA modulus, $p = p_h + p_l$, where p_h is known and p_l is "small";
- Put $R(X) = X + p_h$;
- Have $\gcd(R(p_l), N) \approx N^{1/2}$;
- Coppersmith's method $\Rightarrow$ can find p_l as soon as $|p_l| \leq N^{1/4}$.
- Can factor N in polynomial time from half the high order bits of a factor of N.

Lattice algorithms – Coppersmith's method, application 1

Factoring with high bits known.

- Assume $N = pq$ RSA modulus, $p = p_h + p_l$, where p_h is known and p_l is "small";
- Put $R(X) = X + p_h$;
- Have $\gcd(R(p_l), N) \approx N^{1/2}$;
- Coppersmith's method $\Rightarrow$ can find p_l as soon as $|p_l| \leq N^{1/4}$.
- Can factor N in polynomial time from half the high order bits of a factor of N.

Lattice algorithms – Coppersmith's method, application 1

Factoring with high bits known.

- Assume $N = pq$ RSA modulus, $p = p_h + p_l$, where p_h is known and p_l is "small";
- Put $R(X) = X + p_h$;
- Have $\gcd(R(p_l), N) \approx N^{1/2}$;
- Coppersmith's method $\Rightarrow$ can find p_l as soon as $|p_l| \leq N^{1/4}$.
- Can factor N in polynomial time from half the high order bits of a factor of N.

Lattice algorithms – Coppersmith's method, application 1

Factoring with high bits known.

- Assume $N = pq$ RSA modulus, $p = p_h + p_l$, where p_h is known and p_l is "small";
- Put $R(X) = X + p_h$;
- Have $\gcd(R(p_l), N) \approx N^{1/2}$;
- Coppersmith's method $\Rightarrow$ can find p_l as soon as $|p_l| \leq N^{1/4}$.
- Can factor N in polynomial time from half the high order bits of a factor of N.

Lattice algorithms – Coppersmith's method, application 1

Factoring with high bits known.

- Assume $N = pq$ RSA modulus, $p = p_h + p_l$, where p_h is known and p_l is "small";
- Put $R(X) = X + p_h$;
- Have $\gcd(R(p_l), N) \approx N^{1/2}$;
- Coppersmith's method $\Rightarrow$ can find p_l as soon as $|p_l| \le N^{1/4}$.
- Can factor N in polynomial time from half the high order bits of a factor of N.

Lattice algorithms – Coppersmith's method, bivariate version

- same game with $Q(x, y) = 0 \bmod N$, $|x| \leq X$, $|y| \leq Y$;
- Build a lattice from $(xX)^\ell (yY)^m Q(xX, yY)^i N^{k-i}$;
- Find **two** small vectors in this lattice $\rightarrow$

$$P_1(x, y) = 0 \bmod N, P_2(x, y) = 0 \bmod N.$$

- If P_1, P_2 small enough, $P_1(x, y) = P_2(x, y) = 0$;
- May fail if P_1 and P_2 are algebraically dependent...
- Otherwise, use your favorite method to solve (resultant, Hensel lifting, etc.).
- Details are tricky: auxiliary polynomials depend finely on shape of P and X, Y.

Lattice algorithms – Coppersmith's method, bivariate version

- same game with $Q(x, y) = 0 \bmod N$, $|x| \leq X$, $|y| \leq Y$;
- Build a lattice from $(xX)^\ell (yY)^m Q(xX, yY)^i N^{k-i}$;
- Find **two** small vectors in this lattice $\rightarrow$

$$P_1(x, y) = 0 \bmod N, P_2(x, y) = 0 \bmod N.$$

- If P_1, P_2 small enough, $P_1(x, y) = P_2(x, y) = 0$;
- May fail if P_1 and P_2 are algebraically dependent...
- Otherwise, use your favorite method to solve (resultant, Hensel lifting, etc.).
- Details are tricky: auxiliary polynomials depend finely on shape of P and X, Y.

Lattice algorithms – Coppersmith's method, bivariate version

- same game with $Q(x, y) = 0 \bmod N$, $|x| \le X$, $|y| \le Y$;
- Build a lattice from $(xX)^\ell (yY)^m Q(xX, yY)^i N^{k-i}$;
- Find **two** small vectors in this lattice $\to$

$$P_1(x, y) = 0 \bmod N, P_2(x, y) = 0 \bmod N.$$

- If P_1, P_2 small enough, $P_1(x, y) = P_2(x, y) = 0$;
- May fail if P_1 and P_2 are algebraically dependent...
- Otherwise, use your favorite method to solve (resultant, Hensel lifting, etc.).
- Details are tricky: auxiliary polynomials depend finely on shape of P and X, Y.

Lattice algorithms – Coppersmith's method, bivariate version

- same game with $Q(x, y) = 0 \bmod N$, $|x| \leq X$, $|y| \leq Y$;
- Build a lattice from $(xX)^\ell (yY)^m Q(xX, yY)^i N^{k-i}$;
- Find **two** small vectors in this lattice $\rightarrow$

$$P_1(x, y) = 0 \bmod N, P_2(x, y) = 0 \bmod N.$$

- If P_1, P_2 small enough, $P_1(x, y) = P_2(x, y) = 0$;
- May fail if P_1 and P_2 are algebraically dependent...
- Otherwise, use your favorite method to solve (resultant, Hensel lifting, etc.).
- Details are tricky: auxiliary polynomials depend finely on shape of P and X, Y.

Lattice algorithms – Coppersmith's method, bivariate version

- same game with $Q(x, y) = 0 \bmod N$, $|x| \leq X$, $|y| \leq Y$;
- Build a lattice from $(xX)^\ell (yY)^m Q(xX, yY)^i N^{k-i}$;
- Find **two** small vectors in this lattice $\rightarrow$

$$P_1(x, y) = 0 \bmod N, P_2(x, y) = 0 \bmod N.$$

- If P_1, P_2 small enough, $P_1(x, y) = P_2(x, y) = 0$;
- May fail if P_1 and P_2 are algebraically dependent...
- Otherwise, use your favorite method to solve (resultant, Hensel lifting, etc.).
- Details are tricky: auxiliary polynomials depend finely on shape of P and X, Y.

Lattice algorithms – Coppersmith's method, bivariate version

- same game with $Q(x, y) = 0 \bmod N$, $|x| \leq X$, $|y| \leq Y$;
- Build a lattice from $(xX)^\ell (yY)^m Q(xX, yY)^i N^{k-i}$;
- Find **two** small vectors in this lattice $\rightarrow$

$$P_1(x, y) = 0 \bmod N, P_2(x, y) = 0 \bmod N.$$

- If P_1, P_2 small enough, $P_1(x, y) = P_2(x, y) = 0$;
- May fail if P_1 and P_2 are algebraically dependent...
- Otherwise, use your favorite method to solve (resultant, Hensel lifting, etc.).
- Details are tricky: auxiliary polynomials depend finely on shape of P and X, Y.

Lattice algorithms – Coppersmith's method, bivariate version

- same game with $Q(x, y) = 0 \bmod N$, $|x| \leq X$, $|y| \leq Y$;
- Build a lattice from $(xX)^\ell (yY)^m Q(xX, yY)^i N^{k-i}$;
- Find **two** small vectors in this lattice $\rightarrow$

$$P_1(x, y) = 0 \bmod N, P_2(x, y) = 0 \bmod N.$$

- If P_1, P_2 small enough, $P_1(x, y) = P_2(x, y) = 0$;
- May fail if P_1 and P_2 are algebraically dependent...
- Otherwise, use your favorite method to solve (resultant, Hensel lifting, etc.).
- Details are tricky: auxiliary polynomials depend finely on shape of P and X, Y.

Lattice algorithms – Coppersmith's method, application 2

RSA modulus $N = ed$, d small, e public.

- Wiener's attack with continued fractions $\Rightarrow$ can find d from e, N as soon as $d < N^{1/4}$;
- Boneh-Durfee : $ed + k(N + 1 - p - q) = 1$, hence $k(A + s) = 1 \bmod e$, k, s unknown.
- d small $\Rightarrow e \approx N \Rightarrow |k| \leq e^{\delta}, |s| \leq e^{0.5}$.
- $f(x, y) := x(A + y) - 1$, use

$$g_{i,l} = x^i f(x, y)^l e^{k-l}, h_{i,j} = y^j f(x, y)^l e^{k-l},$$

- Triangular matrix... find $\tilde{g}$ small enough as soon as $\delta \leq 0.284$.
- Using a better (non full-rank) lattice gives $\delta \leq 0.292$.

Lattice algorithms – Coppersmith's method and RS codes

Let $\mathbb{K}$ be a finite field, n, k given.

- $(x_i)_{1 \le i \le n}$ pairwise distinct points in $\mathbb{K}$;
- Reed-Solomon code: message is P, $\deg P < k$, send $(P(x_1), \ldots, P(x_n))$, receive $(y_1, \ldots, y_n)$;
- Define $R(x)$ of degree $< k$ st. $R(x_i) = y_i$ for all i;
- Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \le k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

This is (list-)decoding of Reed-Solomon codes.

Lattice algorithms – Coppersmith's method and RS codes

Let $\mathbb{K}$ be a finite field, n, k given.

- $(x_i)_{1 \leq i \leq n}$ pairwise distinct points in $\mathbb{K}$;
- Reed-Solomon code: message is P, $\deg P < k$, send $(P(x_1), \ldots, P(x_n))$, receive $(y_1, \ldots, y_n)$;
- Define $R(x)$ of degree $< k$ st. $R(x_i) = y_i$ for all i;
- Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \leq k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

This is (list-)decoding of Reed-Solomon codes.

Lattice algorithms – Coppersmith's method and RS codes

Let $\mathbb{K}$ be a finite field, n, k given.

- $(x_i)_{1 \le i \le n}$ pairwise distinct points in $\mathbb{K}$;
- Reed-Solomon code: message is P, $\deg P < k$, send $(P(x_1), \dots, P(x_n))$, receive $(y_1, \dots, y_n)$;
- Define $R(x)$ of degree $< k$ st. $R(x_i) = y_i$ for all i;
- Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \le k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

This is (list-)decoding of Reed-Solomon codes.

Lattice algorithms – Coppersmith's method and RS codes

Let $\mathbb{K}$ be a finite field, n, k given.

- $(x_i)_{1 \leq i \leq n}$ pairwise distinct points in $\mathbb{K}$;
- Reed-Solomon code: message is P, $\deg P < k$, send $(P(x_1), \ldots, P(x_n))$, receive $(y_1, \ldots, y_n)$;
- Define $R(x)$ of degree $< k$ st. $R(x_i) = y_i$ for all i;
- Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \leq k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

This is (list-)decoding of Reed-Solomon codes.

Lattice algorithms – Coppersmith's method and RS codes

Let $\mathbb{K}$ be a finite field, n, k given.

- $(x_i)_{1 \le i \le n}$ pairwise distinct points in $\mathbb{K}$;
- Reed-Solomon code: message is P, $\deg P < k$, send $(P(x_1), \dots, P(x_n))$, receive $(y_1, \dots, y_n)$;
- Define $R(x)$ of degree $< k$ st. $R(x_i) = y_i$ for all i;
- Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \le k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

This is (list-)decoding of Reed-Solomon codes.

Lattice algorithms – Coppersmith's method and RS codes

(List) Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \leq k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

- Can use a polynomial version of Coppersmith's ideas: look for small degree linear combination of $Y^u(Y - R(X))^\ell(\prod_{i=1}^{n}(X - x_i))^{k-\ell}$
- classical decoding $k = 1$, $u = 0$: $\delta = (n + k)/2$.
- $k = 1$, u arbitrary: $\delta = \sqrt{2kn}$ (Sudan)
- Full Coppersmith's method: $\delta = \sqrt{kn}$ (Guruswami-Sudan)

Lattice algorithms – Coppersmith's method and RS codes

(List) Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \le k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

- Can use a polynomial version of Coppersmith's ideas: look for small degree linear combination of $Y^u(Y - R(X))^\ell (\prod_{i=1}^{n}(X - x_i))^{k-\ell}$
- classical decoding $k = 1$, $u = 0$: $\delta = (n + k)/2$.
- $k = 1$, u arbitrary: $\delta = \sqrt{2kn}$ (Sudan)
- Full Coppersmith's method: $\delta = \sqrt{kn}$ (Guruswami-Sudan)

Lattice algorithms – Coppersmith's method and RS codes

(List) Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \le k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

- Can use a polynomial version of Coppersmith's ideas: look for small degree linear combination of $Y^u(Y - R(X))^\ell (\prod_{i=1}^{n}(X - x_i))^{k-\ell}$
- classical decoding $k = 1$, $u = 0$: $\delta = (n + k)/2$.
- $k = 1$, u arbitrary: $\delta = \sqrt{2kn}$ (Sudan)
- Full Coppersmith's method: $\delta = \sqrt{kn}$ (Guruswami-Sudan)

Lattice algorithms – Coppersmith's method and RS codes

(List) Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \leq k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

- Can use a polynomial version of Coppersmith's ideas: look for small degree linear combination of $Y^u(Y - R(X))^\ell(\prod_{i=1}^{n}(X - x_i))^{k-\ell}$
- classical decoding $k = 1$, $u = 0$: $\delta = (n + k)/2$.
- $k = 1$, u arbitrary: $\delta = \sqrt{2kn}$ (Sudan)
- Full Coppersmith's method: $\delta = \sqrt{kn}$ (Guruswami-Sudan)

Lattice algorithms – Coppersmith's method and RS codes

(List) Decoding : Find all $P \in \mathbb{K}[X]$, $\deg P \leq k$, st. $\deg \gcd(P(X) - R(X), \prod_{i=1}^{n}(X - x_i)) > \delta$ (correct $n - \delta$ errors).

- Can use a polynomial version of Coppersmith's ideas: look for small degree linear combination of $Y^u(Y - R(X))^\ell (\prod_{i=1}^{n}(X - x_i))^{k-\ell}$
- classical decoding $k = 1$, $u = 0$: $\delta = (n + k)/2$.
- $k = 1$, u arbitrary: $\delta = \sqrt{2kn}$ (Sudan)
- Full Coppersmith's method: $\delta = \sqrt{kn}$ (Guruswami-Sudan)