Computing with Euclidean lattices

Damien Stehlé

ÉNS de Lyon

Monash University, November 2012

Goals and plan of the talk

Goals:

- An introduction to the computational aspects of lattices
- An example of how floating-point arithmetic can be used to accelerate an algebraic computation

Plan of the talk:

1. **Euclidean lattices**
2. Applications of lattices
3. The LLL algorithm
4. Speeding up LLL

Goals and plan of the talk

Goals:

- An introduction to the computational aspects of lattices
- An example of how floating-point arithmetic can be used to accelerate an algebraic computation

Plan of the talk:

1. **Euclidean lattices**
2. Applications of lattices
3. The LLL algorithm
4. Speeding up LLL

Euclidean lattices

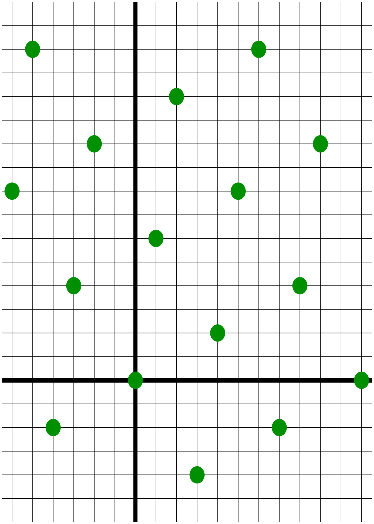
Lattice $\equiv$ discrete subgroup of $\mathbb{R}^n$

$\equiv \quad \{\sum_{i \leq n} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$

If the $\mathbf{b}_i$'s are linearly independent, they are called a **basis**.

Bases are not unique, but they can be obtained from each other by integer transforms of determinant ± 1:

$$\begin{bmatrix} -2 & 1 \\ 10 & 6 \end{bmatrix} = \begin{bmatrix} 4 & -3 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}.$$



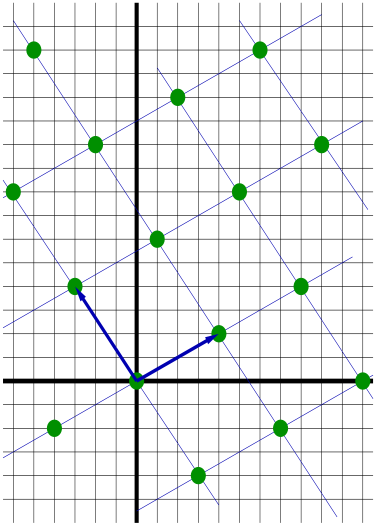
Euclidean lattices

$$\text{Lattice} \equiv \text{discrete subgroup of } \mathbb{R}^n$$
$$\equiv \left\{ \sum_{i \leq n} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\}$$

If the $\mathbf{b}_i$'s are linearly independent, they are called a **basis**.

Bases are not unique, but they can be obtained from each other by integer transforms of determinant ± 1:

$$\begin{bmatrix} -2 & 1 \\ 10 & 6 \end{bmatrix} = \begin{bmatrix} 4 & -3 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}.$$



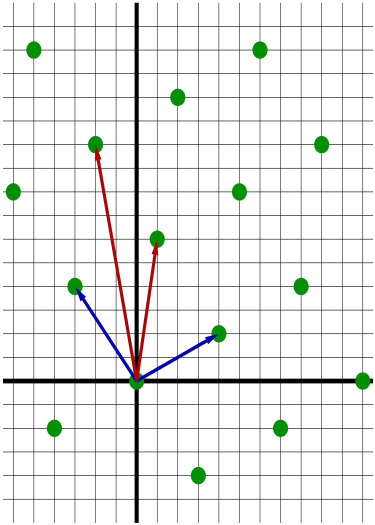
Euclidean lattices

Lattice $\equiv$ discrete subgroup of $\mathbb{R}^n$
$\equiv \{\sum_{i \leq n} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$

If the $\mathbf{b}_i$'s are linearly independent, they are called a **basis**.

Bases are not unique, but they can be obtained from each other by integer transforms of determinant ± 1:

$$\begin{bmatrix} -2 & 1 \\ 10 & 6 \end{bmatrix} = \begin{bmatrix} 4 & -3 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}.$$



Lattice invariants

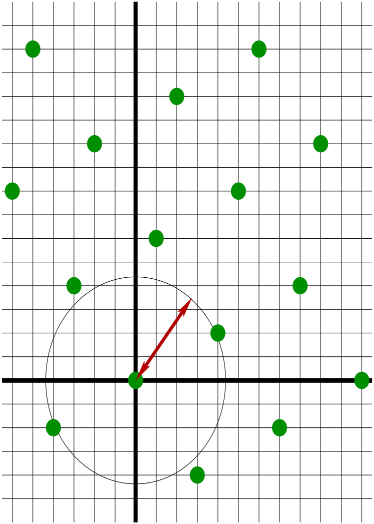
Minimum:
$\lambda(L) = \min(\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0})$

Determinant:
$\det L = |\det(\mathbf{b}_i)_i|$, for any basis

Minkowski theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$

Algorithmic approach: lattice reduction

Start from a basis, and progressively improve its norm/orthogonality



Lattice invariants

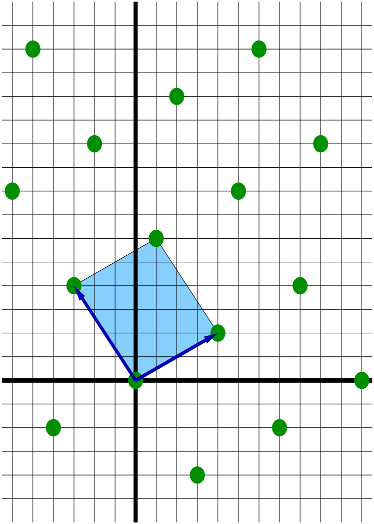
Minimum:
$\lambda(L) = \min(\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0})$

Determinant:
$\det L = \left|\det(\mathbf{b}_i)_i\right|$, for any basis

Minkowski theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$

Algorithmic approach: lattice reduction

Start from a basis, and progressively improve its norm/orthogonality



Lattice invariants

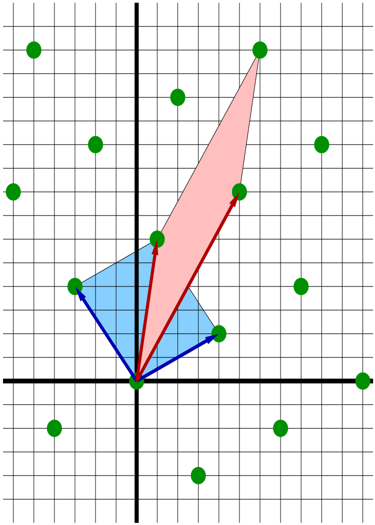
Minimum:
$\lambda(L) = \min(\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0})$

Determinant:
$\det L = \left|\det(\mathbf{b}_i)_i\right|$, for any basis

Minkowski theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$

Algorithmic approach: lattice reduction

Start from a basis, and progressively improve its norm/orthogonality



Lattice invariants

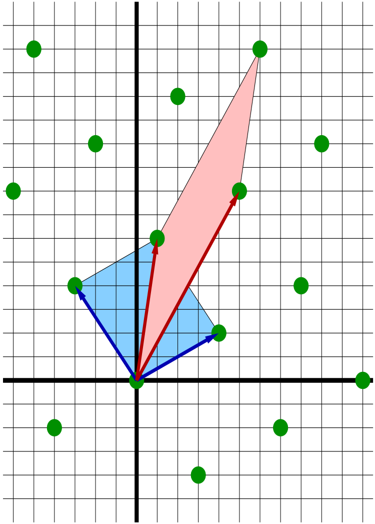
Minimum:
$\lambda(L) = \min(\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0})$

Determinant:
$\det L = |\det(\mathbf{b}_i)_i|$, for any basis

Minkowski theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$

Algorithmic approach: lattice reduction

Start from a basis, and progressively improve its norm/orthogonality



Why do we care about lattices?

- Computer algebra: factorisation of rational polynomials.
- Cryptanalysis of variants of RSA.
- Lattice-based cryptography.
- Communications theory: MIMO, GPS.
- Combinatorial optimisation, algorithmic group theory, algorithmic number theory, computer arithmetic, etc.

Lattices tend to pop out when one wants to use linear algebra but is restricted to discrete transformations.

Why do we care about lattices?

- Computer algebra: factorisation of rational polynomials.
- Cryptanalysis of variants of RSA.
- Lattice-based cryptography.
- Communications theory: MIMO, GPS.
- Combinatorial optimisation, algorithmic group theory, algorithmic number theory, computer arithmetic, etc.

Lattices tend to pop out when one wants to use linear algebra but is restricted to discrete transformations.

Main computational problem: SVP

- SVP$_\gamma$: Given a basis of L, find $\mathbf{b} \in L$ with

$$0 < \|\mathbf{b}\| \leq \gamma \cdot \lambda(L).$$

- Dec-SVP$_\gamma$: Given a basis of L and $t > 0$, reply:

$$\textbf{YES} \text{ if } \lambda(L) \leq t \quad \text{and} \quad \textbf{NO} \text{ if } \lambda(L) > \gamma \cdot t.$$

Dec-SVP$_\gamma$ on the hardness scale

- NP-hard for any $\gamma \leq \mathcal{O}(1)$, under randomized reductions
- In NP$\cap$coNP for $\gamma \geq \sqrt{n}$
- In P for $\gamma \geq 2^{\frac{n \log \log n}{\log n}}$

Main computational problem: SVP

- SVP$_\gamma$: Given a basis of L, find $\mathbf{b} \in L$ with

$$0 < \|\mathbf{b}\| \leq \gamma \cdot \lambda(L).$$

- Dec-SVP$_\gamma$: Given a basis of L and $t > 0$, reply:

$$\textbf{YES} \text{ if } \lambda(L) \leq t \quad \text{and} \quad \textbf{NO} \text{ if } \lambda(L) > \gamma \cdot t.$$

Dec-SVP$_\gamma$ on the hardness scale

- NP-hard for any $\gamma \leq \mathcal{O}(1)$, under randomized reductions
- In NP$\cap$coNP for $\gamma \geq \sqrt{n}$
- In P for $\gamma \geq 2^{\frac{n \log \log n}{\log n}}$

Main computational problem: SVP

- SVP$_\gamma$: Given a basis of L, find $\mathbf{b} \in L$ with

$$0 < \|\mathbf{b}\| \leq \gamma \cdot \lambda(L).$$

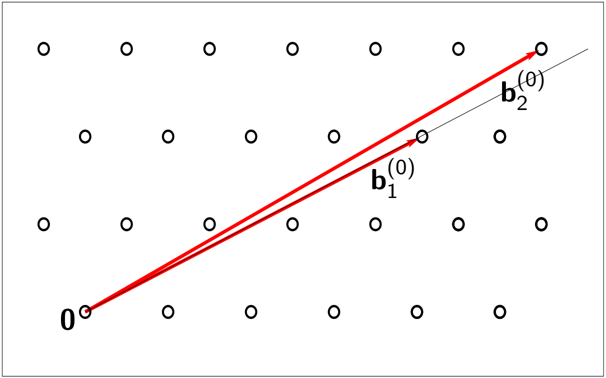
- Dec-SVP$_\gamma$: Given a basis of L and $t > 0$, reply:

$$\textbf{YES} \text{ if } \lambda(L) \leq t \quad \text{and} \quad \textbf{NO} \text{ if } \lambda(L) > \gamma \cdot t.$$

Dec-SVP$_\gamma$ on the hardness scale

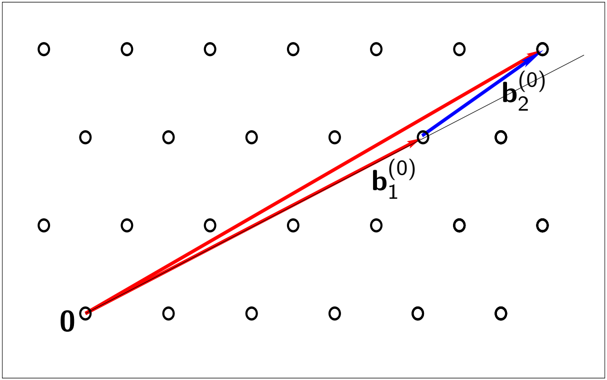
- NP-hard for any $\gamma \leq \mathcal{O}(1)$, under randomized reductions
- In NP$\cap$coNP for $\gamma \geq \sqrt{n}$
- In P for $\gamma \geq 2^{\frac{n \log \log n}{\log n}}$

SVP is easy in small dimensions!



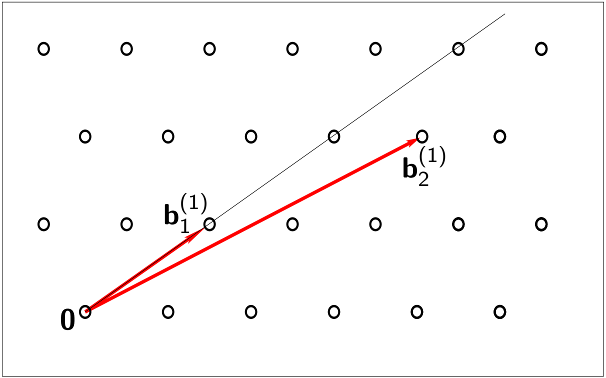
- That's almost Euclid's algorithm!
- Returns a vector reaching $\lambda(L)$
- Runs in polynomial time

SVP is easy in small dimensions!



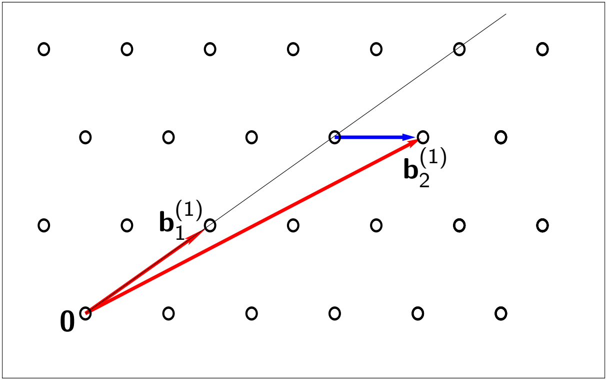
- That's almost Euclid's algorithm!
- Returns a vector reaching $\lambda(L)$
- Runs in polynomial time

SVP is easy in small dimensions!



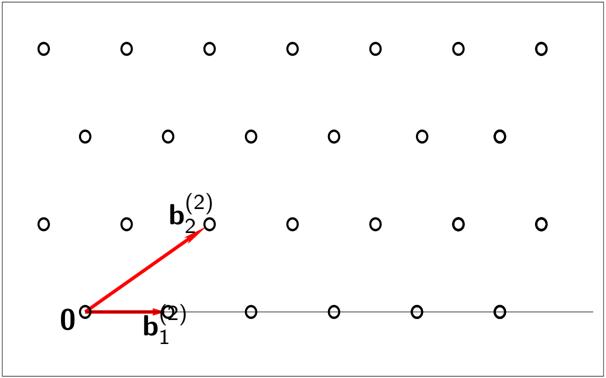
- That's almost Euclid's algorithm!
- Returns a vector reaching $\lambda(L)$
- Runs in polynomial time

SVP is easy in small dimensions!



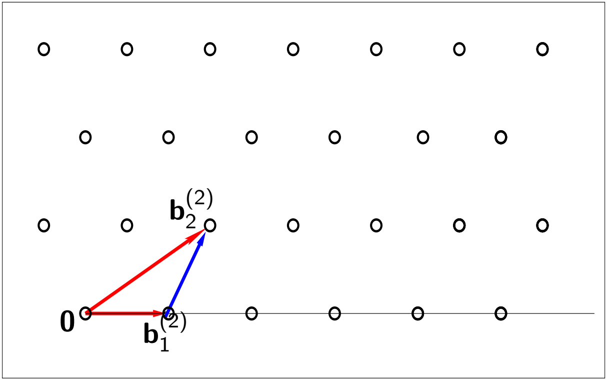
- That's almost Euclid's algorithm!
- Returns a vector reaching $\lambda(L)$
- Runs in polynomial time

SVP is easy in small dimensions!



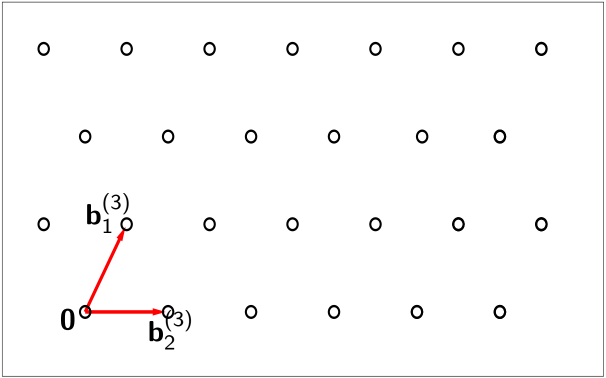
- That's almost Euclid's algorithm!
- Returns a vector reaching $\lambda(L)$
- Runs in polynomial time

SVP is easy in small dimensions!



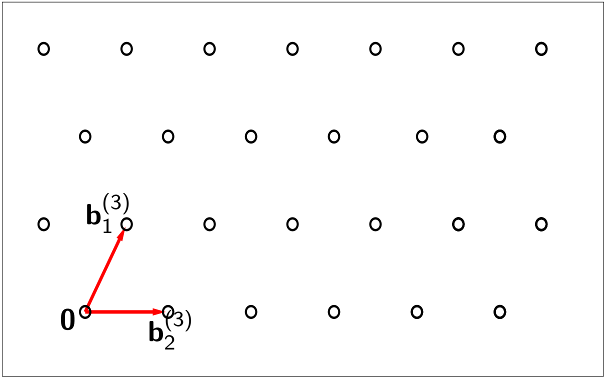
- That's almost Euclid's algorithm!
- Returns a vector reaching $\lambda(L)$
- Runs in polynomial time

SVP is easy in small dimensions!



- That's almost Euclid's algorithm!
- Returns a vector reaching $\lambda(L)$
- Runs in polynomial time

SVP is easy in small dimensions!



- That's almost Euclid's algorithm!
- Returns a vector reaching $\lambda(L)$
- Runs in polynomial time

Plan of the talk

Plan of the talk:

1. Euclidean lattices
2. **Applications of euclidean lattices**
3. The LLL algorithm
4. Speeding up LLL

- Integer relation detection
- Polynomial factorisation
- Cryptanalysis

Plan of the talk

Plan of the talk:

1. Euclidean lattices
2. **Applications of euclidean lattices**
3. The LLL algorithm
4. Speeding up LLL

- Integer relation detection
- Polynomial factorisation
- Cryptanalysis

Finding small integer relations between real numbers

BBP formula: $$\pi = \sum_{i \geq 0} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right)$$

$\rightsquigarrow$ To compute a base-16 digit of π at any given position.

Assume we search a small $\mathbb{Z}$-relation between $y_1, \ldots, y_d \in \mathbb{R}$

Take $L := L[(\mathbf{b}_i)_i]$, with $B = \begin{pmatrix} y_1 & y_2 & \ldots & y_d \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$

A $\mathbb{Z}$-relation $\sum x_i y_i = 0$ leads to a small vector $(0, x_1, \ldots, x_d)^T$

Finding small integer relations between real numbers

BBP formula: $\pi = \sum_{i \geq 0} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right)$

$\rightsquigarrow$ To compute a base-16 digit of π at any given position.

Assume we search a small $\mathbb{Z}$-relation between $y_1, \ldots, y_d \in \mathbb{R}$

$$\text{Take } L := L[(\mathbf{b}_i)_i], \quad \text{with } B = \begin{pmatrix} y_1 & y_2 & \ldots & y_d \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$$

A $\mathbb{Z}$-relation $\sum x_i y_i = 0$ leads to a small vector $(0, x_1, \ldots, x_d)^T$

Finding small integer relations between real numbers

BBP formula: $$\pi = \sum_{i \geq 0} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right)$$

$\rightsquigarrow$ To compute a base-16 digit of π at any given position.

Assume we search a small $\mathbb{Z}$-relation between $y_1, \ldots, y_d \in \mathbb{R}$

Take $L := L[(\mathbf{b}_i)_i]$, with $B = \begin{pmatrix} y_1 & y_2 & \ldots & y_d \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$

A $\mathbb{Z}$-relation $\sum x_i y_i = 0$ leads to a small vector $(0, x_1, \ldots, x_d)^T$

Using a large C makes it a shortest vector of $L \setminus 0$

Finding small integer relations between real numbers

BBP formula: $$\pi = \sum_{i \geq 0} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right)$$

$\rightsquigarrow$ To compute a base-16 digit of π at any given position.

Assume we search a small $\mathbb{Z}$-relation between $y_1, \ldots, y_d \in \mathbb{R}$

Take $L := L[(\mathbf{b}_i)_i]$, with $$B = \begin{pmatrix} y_1 & y_2 & \ldots & y_d \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$$

A $\mathbb{Z}$-relation $\sum x_i y_i = 0$ leads to a small vector $(0, x_1, \ldots, x_d)^T$

Using a large C makes it a shortest vector of $L \setminus \mathbf{0}$

Finding small integer relations between real numbers

BBP formula: $$\pi = \sum_{i \geq 0} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right)$$

$\rightsquigarrow$ To compute a base-16 digit of π at any given position.

Assume we search a small $\mathbb{Z}$-relation between $y_1, \ldots, y_d \in \mathbb{R}$

Take $L := L[(\mathbf{b}_i)_i]$, with $$B = \begin{pmatrix} Cy_1 & Cy_2 & \ldots & Cy_d \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$$

A $\mathbb{Z}$-relation $\sum x_i y_i = 0$ leads to a small vector $(0, x_1, \ldots, x_d)^T$
Using a large C makes it a shortest vector of $L \setminus \mathbf{0}$

Factoring integer polynomials

The previous idea may be used to factor polynomials in $\mathbb{Z}[x]$

Factoring polynomials with rational coefficients,
A. K. Lenstra, H. W. Lenstra Jr. and L. Lovász. Math. Ann., 1982

$\Rightarrow$ Cited $\geq$ 2500 times!!!

Given $P \in \mathbb{Z}[x]$:

0- If $\deg P \leq 1$, then stop

1- Compute a root $\alpha \in \mathbb{C}$ of P

2- Find the minimal polynomial $P_\alpha(x)$ of α, by searching for $\mathbb{Z}$-combinations between $1, \alpha, \ldots, \alpha^i$ for increasing i

3- Divide P by P_α and restart

Factoring integer polynomials

The previous idea may be used to factor polynomials in $\mathbb{Z}[x]$

Factoring polynomials with rational coefficients,
A. K. Lenstra, H. W. Lenstra Jr. and L. Lovász. Math. Ann., 1982

$\Rightarrow$ Cited $\geq$ 2500 times!!!

Given $P \in \mathbb{Z}[x]$:

0- If $\deg P \leq 1$, then stop
1- Compute a root $\alpha \in \mathbb{C}$ of P
2- Find the minimal polynomial $P_\alpha(x)$ of α, by searching for $\mathbb{Z}$-combinations between $1, \alpha, \ldots, \alpha^i$ for increasing i
3- Divide P by P_α and restart

Cryptographic design and cryptanalysis

Lattice-based cryptography:

- Secret key: very short basis of a lattice
- Public key: long basis of the same lattice
- Relies on the assumed hardness of SVP

Very popular research topic:

- More secure: post-quantum
- More efficient: no modular exponentiation
- More versatile: fully homomorphic encryption

Lattice reduction algorithms are the best known attack.

Cryptographic design and cryptanalysis

Lattice-based cryptography:

- Secret key: very short basis of a lattice
- Public key: long basis of the same lattice
- Relies on the assumed hardness of SVP

Very popular research topic:

- More secure: post-quantum
- More efficient: no modular exponentiation
- More versatile: fully homomorphic encryption

Lattice reduction algorithms are the best known attack.

Cryptographic design and cryptanalysis

Lattice-based cryptography:

- Secret key: very short basis of a lattice
- Public key: long basis of the same lattice
- Relies on the assumed hardness of SVP

Very popular research topic:

- More secure: post-quantum
- More efficient: no modular exponentiation
- More versatile: fully homomorphic encryption

Lattice reduction algorithms are the best known attack.

Plan of the talk

Plan of the talk:

1. Euclidean lattices
2. Applications of euclidean lattices
3. **The LLL algorithm**
4. Speeding up LLL

Gram-Schmidt orthogonalization (GSO)

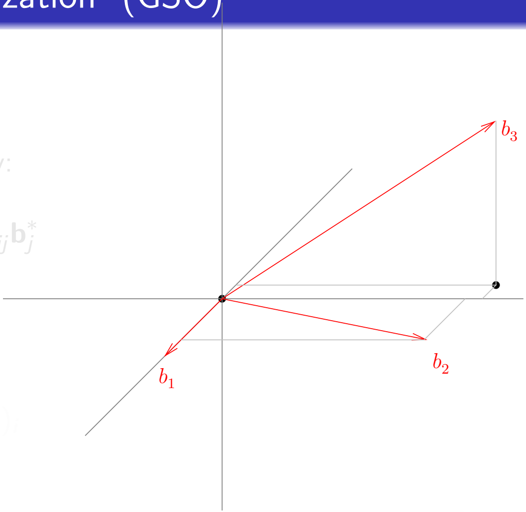
$(\mathbf{b}_i)_i$ linearly independent

The GSO $(\mathbf{b}_i^*)_i$ is defined by:

$$\forall i: \quad \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j<i} \mu_{ij} \mathbf{b}_j^*$$

$$\forall i > j: \quad \mu_{ij} = \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_j^*\|^2}$$

Triangularization of $B = (\mathbf{b}_i)_i$



For any basis $(\mathbf{b}_i)_i$ of L, we have $\lambda(L) \geq \min_i \|\mathbf{b}_i^*\|$

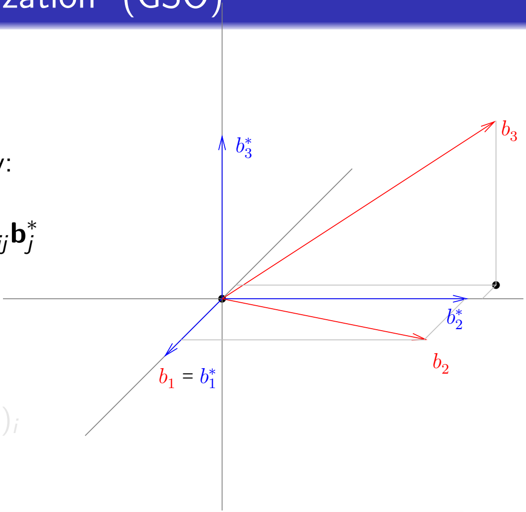
Gram-Schmidt orthogonalization (GSO)

$(\mathbf{b}_i)_i$ linearly independent

The GSO $(\mathbf{b}_i^*)_i$ is defined by:

$$\forall i : \quad \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j<i} \mu_{ij} \mathbf{b}_j^*$$

$$\forall i > j : \quad \mu_{ij} = \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_j^*\|^2}$$



Triangularisation of $B = (\mathbf{b}_i)_i$

For any basis $(\mathbf{b}_i)_i$ of L, we have $\lambda(L) \geq \min_i \|\mathbf{b}_i^*\|$

Gram-Schmidt orthogonalization (GSO)

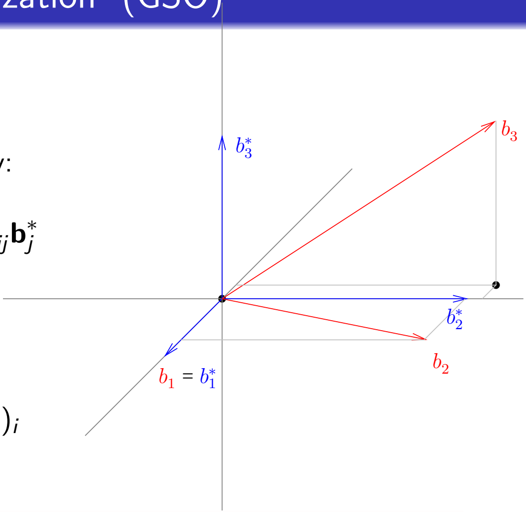
$(\mathbf{b}_i)_i$ linearly independent

The GSO $(\mathbf{b}_i^*)_i$ is defined by:

$$\forall i: \quad \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j<i} \mu_{ij} \mathbf{b}_j^*$$

$$\forall i > j: \quad \mu_{ij} = \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_j^*\|^2}$$

Triangularisation of $B = (\mathbf{b}_i)_i$



For any basis $(\mathbf{b}_i)_i$ of L, we have $\lambda(L) \geq \min_i \|\mathbf{b}_i^*\|$

Gram-Schmidt orthogonalization (GSO)

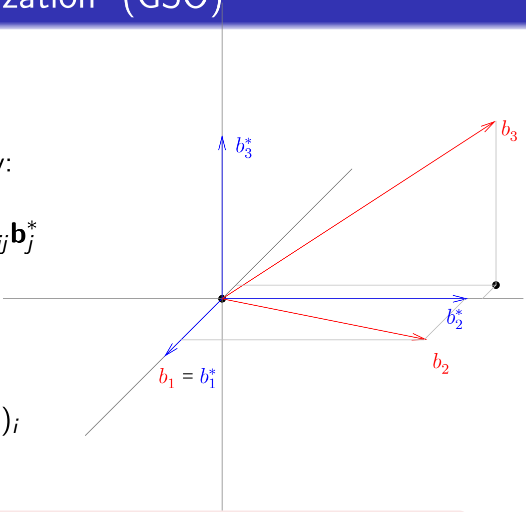
$(\mathbf{b}_i)_i$ linearly independent

The GSO $(\mathbf{b}_i^*)_i$ is defined by:

$$\forall i: \quad \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j<i} \mu_{ij} \mathbf{b}_j^*$$

$$\forall i > j: \quad \mu_{ij} = \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_j^*\|^2}$$

Triangularisation of $B = (\mathbf{b}_i)_i$



For any basis $(\mathbf{b}_i)_i$ of L, we have $\lambda(L) \geq \min_i \|\mathbf{b}_i^*\|$

The Lenstra-Lenstra-Lovász reduction

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ is said LLL-reduced if

- $\forall i, j: \ |\mu_{ij}| \leq 1/2$ [Size-reduction]
- $\forall i: \ \delta \cdot \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$ [Lovász' condition]

The Lenstra-Lenstra-Lovász reduction

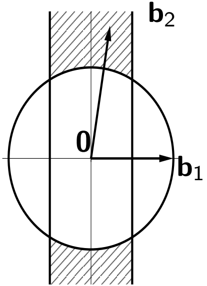
Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ is said LLL-reduced if

- $\forall i, j: \ |\mu_{ij}| \leq 1/2$ [Size-reduction]
- $\forall i: \ \delta \cdot \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$ [Lovász' condition]

The $\|\mathbf{b}_i^*\|$'s can't drop too fast:

$\forall i: \ \|\mathbf{b}_{i+1}^*\|^2 \geq (\delta - \frac{1}{4}) \|\mathbf{b}_i^*\|^2$

$\Rightarrow \lambda(L) \leq \|\mathbf{b}_1\| \leq 2^{\mathcal{O}(n)} \cdot \lambda(L)$



$\delta < 1$ is important to get a polynomial complexity

The Lenstra-Lenstra-Lovász reduction

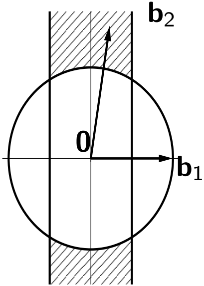
Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ is said LLL-reduced if

- $\forall i, j: \; |\mu_{ij}| \leq 1/2$ [Size-reduction]
- $\forall i: \; \delta \cdot \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$ [Lovász' condition]

The $\|\mathbf{b}_i^*\|$'s can't drop too fast:

$$\forall i: \; \|\mathbf{b}_{i+1}^*\|^2 \geq (\delta - \tfrac{1}{4}) \|\mathbf{b}_i^*\|^2$$

$$\Rightarrow \lambda(L) \leq \|\mathbf{b}_1\| \leq 2^{\mathcal{O}(n)} \cdot \lambda(L)$$



$\delta < 1$ is important to get a polynomial complexity

The Lenstra-Lenstra-Lovász reduction

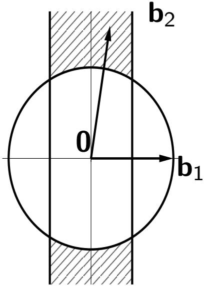
Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ is said LLL-reduced if

- $\forall i, j: \ |\mu_{ij}| \leq 1/2$ [Size-reduction]
- $\forall i: \ \delta \cdot \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$ [Lovász' condition]

The $\|\mathbf{b}_i^*\|$'s can't drop too fast:

$$\forall i: \ \|\mathbf{b}_{i+1}^*\|^2 \geq (\delta - \tfrac{1}{4}) \|\mathbf{b}_i^*\|^2$$

$$\Rightarrow \lambda(L) \leq \|\mathbf{b}_1\| \leq 2^{\mathcal{O}(n)} \cdot \lambda(L)$$



$\delta < 1$ is important to get a polynomial complexity

The Lenstra-Lenstra-Lovász reduction

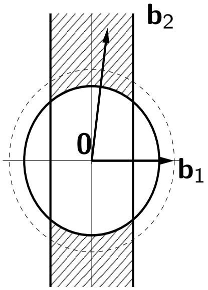
Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ is said LLL-reduced if

- $\forall i, j : \ |\mu_{ij}| \leq 1/2$ [Size-reduction]
- $\forall i : \ \delta \cdot \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$ [Lovász' condition]

The $\|\mathbf{b}_i^*\|$'s can't drop too fast:

$$\forall i : \ \|\mathbf{b}_{i+1}^*\|^2 \geq (\delta - \tfrac{1}{4}) \|\mathbf{b}_i^*\|^2$$

$$\Rightarrow \lambda(L) \leq \|\mathbf{b}_1\| \leq 2^{\mathcal{O}(n)} \cdot \lambda(L)$$



$\delta < 1$ is important to get a polynomial complexity

The Lenstra-Lenstra-Lovász algorithm

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ is said LLL-reduced if

- $\forall i, j: \ |\mu_{ij}| \leq 1/2$ [Size-reduction]
- $\forall i: \ \delta \cdot \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$ [Lovász' condition]

1. Enforce size-reduction, using a modified Gaussian elimination
2. If there is an i with $\delta \cdot \|\mathbf{b}_i^*\|^2 > \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$, then swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$, and go to Step 1
3. Return the current basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$

⇒ Correctness is trivial

⇒ Termination is much less so:

$$\mathcal{O}(n^2 \beta) \text{ loop iterations, with } \beta = \max_i \|\mathbf{b}_i^{init}\|$$

The Lenstra-Lenstra-Lovász algorithm

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ is said LLL-reduced if

- $\forall i, j : \ |\mu_{ij}| \leq 1/2$ [Size-reduction]
- $\forall i : \ \delta \cdot \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$ [Lovász' condition]

1. Enforce size-reduction, using a modified Gaussian elimination
2. If there is an i with $\delta \cdot \|\mathbf{b}_i^*\|^2 > \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$, then swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$, and go to Step 1
3. Return the current basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$

⇒ Correctness is trivial

⇒ Termination is much less so:

$$\mathcal{O}(n^2 \beta) \text{ loop iterations, with } \beta = \max_i \|\mathbf{b}_i^{init}\|$$

The Lenstra-Lenstra-Lovász algorithm

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ is said LLL-reduced if

- $\forall i, j : \ |\mu_{ij}| \leq 1/2$ [Size-reduction]
- $\forall i : \ \delta \cdot \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$ [Lovász' condition]

1. Enforce size-reduction, using a modified Gaussian elimination
2. If there is an i with $\delta \cdot \|\mathbf{b}_i^*\|^2 > \|\mathbf{b}_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|\mathbf{b}_i^*\|^2$, then swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$, and go to Step 1
3. Return the current basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$

$\Rightarrow$ Correctness is trivial

$\Rightarrow$ Termination is much less so:

$$\mathcal{O}(n^2 \beta) \text{ loop iterations, with } \beta = \max_i \|\mathbf{b}_i^{init}\|$$

Bit-complexity of LLL and practical run-time

[LLL82,Kaltofen83]

LLL terminates in $\mathcal{O}(n^4\beta^2(n+\beta))$ operations, with $\beta = \log \max_i \|\mathbf{b}_i^{init}\|$

With MAGMA V2.16:

```
> n := 25; B := RMatrixSpace(Integers(),n,n)!0;
> for i:=1 to 25 do
>    B[i][i]:=1; B[i][1]:=RandomBits(2000);
> end for;
> time C := LLL(B:Method:=''Integral'');
```

Time: 11.700

```
> time C := LLL(B);
```

Time: 0.240

Plan of the talk

Plan of the talk:

1. Euclidean lattices
2. Applications of euclidean lattices
3. The LLL algorithm
4. **Speeding up LLL**

Section based on joint works with X.-W. Chang, I. Morel, P. Q. Nguyen, A. Novocin, X. Pujol and G. Villard

LLL in practice: the numeric-symbolic approach

The Gram-Schmidt computations dominate the cost

Odlyzko's hybrid approach

Replace the rational computations on the GSO by floating-point approximations, but keep the basis operations exact

Floating-point numbers: $x_1.x_2x_3\ldots x_p \cdot B^e$, where:

- p is the precision
- B is the base, and $x_i \in \{0, \ldots, B-1\}$
- $e \in \mathbb{Z}$ is the exponent

Floating-point arithmetic:
$fl(a\ op\ b)$ is a nearest fp number to $a\ op\ b$, for any $op \in \{+, -, /, \times\}$

LLL in practice: the numeric-symbolic approach

The Gram-Schmidt computations dominate the cost

Odlyzko's hybrid approach

Replace the rational computations on the GSO by floating-point approximations, but keep the basis operations exact

Floating-point numbers: $x_1.x_2x_3\dots x_p \cdot B^e$, where:

- p is the precision
- B is the base, and $x_i \in \{0,\dots,B-1\}$
- $e \in \mathbb{Z}$ is the exponent

Floating-point arithmetic:

$fl(a\ op\ b)$ is a nearest fp number to $a\ op\ b$, for any $op \in \{+,-,/,\times\}$

LLL in practice: the numeric-symbolic approach

The Gram-Schmidt computations dominate the cost

Odlyzko's hybrid approach

Replace the rational computations on the GSO by floating-point approximations, but keep the basis operations exact

Floating-point numbers: $x_1.x_2x_3\ldots x_p \cdot B^e$, where:

- p is the precision
- B is the base, and $x_i \in \{0,\ldots,B-1\}$
- $e \in \mathbb{Z}$ is the exponent

Floating-point arithmetic:
$fl(a \; op \; b)$ is a nearest fp number to $a \; op \; b$, for any $op \in \{+,-,/,\times\}$

Odlyzko's hybrid approach is only heuristic

Odlyzko's hybrid approach

Replace the rational computations on the GSO by floating-point approximations, but keep the basis operations exact

Principle: For p small, fp arith. may efficiently simulate rational arith.
$\Rightarrow$ In practice: we aim for 53-bit machine precision

But Odlyzko's approach is heuristic:

- Fp arithmetic is inexact
- Small errors can be amplified

$\Rightarrow$ Infinite loops

$\Rightarrow$ Incorrect outputs

Odlyzko's hybrid approach is only heuristic

Odlyzko's hybrid approach

Replace the rational computations on the GSO by floating-point approximations, but keep the basis operations exact

Principle: For p small, fp arith. may efficiently simulate rational arith.
⇒ In practice: we aim for 53-bit machine precision

But Odlyzko's approach is heuristic:

- Fp arithmetic is inexact
- Small errors can be amplified

⇒ Infinite loops

⇒ Incorrect outputs

Odlyzko's hybrid approach is only heuristic

Odlyzko's hybrid approach

Replace the rational computations on the GSO by floating-point approximations, but keep the basis operations exact

Principle: For p small, fp arith. may efficiently simulate rational arith.
$\Rightarrow$ In practice: we aim for 53-bit machine precision

But Odlyzko's approach is heuristic:

- Fp arithmetic is inexact
- Small errors can be amplified

$\Rightarrow$ Infinite loops

$\Rightarrow$ Incorrect outputs

Making the numeric-symbolic approach rigorous

Underlying mathematical phenomenon [CSV12]

Any LLL-reduced basis is well-conditioned with respect to GSO

- Well-conditioned? The GSO computed in small precision is close to the genuine GSO

$\Rightarrow$ We'd like to rely on LLL-reduced bases as much as we can

Use a greedy LLL algorithm [NS05,MSV09]:

- Consider the first i s.t. $\mathbf{b}_1, \ldots, \mathbf{b}_i$ is not LLL-reduced

$\Rightarrow$ $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ is well-conditioned

- Iterate on $\mathbf{b}_i$ until nothing happens (iterative refinement)

Making the numeric-symbolic approach rigorous

Underlying mathematical phenomenon [CSV12]

Any LLL-reduced basis is well-conditioned with respect to GSO

- Well-conditioned? The GSO computed in small precision is close to the genuine GSO

$\Rightarrow$ We'd like to rely on LLL-reduced bases as much as we can

Use a greedy LLL algorithm [NS05,MSV09]:

- Consider the first i s.t. $\mathbf{b}_1, \ldots, \mathbf{b}_i$ is not LLL-reduced

$\Rightarrow$ $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ is well-conditioned

- Iterate on $\mathbf{b}_i$ until nothing happens (iterative refinement)

Bit complexity of floating-point LLL

Small precision? $\mathcal{O}(n)$ bits suffice for correctness.

Bit-complexity:

$$\underbrace{\mathcal{O}(n^2\beta)}_{1} \cdot \underbrace{\mathcal{O}(n^2)}_{2} \cdot \left[\underbrace{\mathcal{O}(n\beta)}_{3} + \underbrace{\mathcal{O}(n^2)}_{4} \right] = \mathcal{O}\left(n^5\beta(n+\beta)\right).$$

1. loop iterations
2. size-reduction arithmetic steps
3. integer arithmetic
4. floating-point arithmetic

Asymptotically not much better than LLL's $\mathcal{O}(n^4\beta^2(n+\beta))$, but much better in practice

Bit complexity of floating-point LLL

Small precision? $\mathcal{O}(n)$ bits suffice for correctness.

Bit-complexity:

$$\underbrace{\mathcal{O}(n^2\beta)}_{1} \cdot \underbrace{\mathcal{O}(n^2)}_{2} \cdot \left[\underbrace{\mathcal{O}(n\beta)}_{3} + \underbrace{\mathcal{O}(n^2)}_{4} \right] = \mathcal{O}\left(n^5\beta(n+\beta)\right).$$

1. loop iterations
2. size-reduction arithmetic steps
3. integer arithmetic
4. floating-point arithmetic

Asymptotically not much better than LLL's $\mathcal{O}(n^4\beta^2(n+\beta))$, but much better in practice

Can we do better? [NSV10,PSV13?]

The totally numeric approach

LLL can be accelerated further by using approximations for the bases too!
$\Rightarrow \widetilde{\mathcal{O}}(n^5\beta^{1.5})$ operations

The totally numeric approach, continued

Do the same with several levels of recursion
$\Rightarrow \widetilde{\mathcal{O}}(n^5\beta)$ operations

The totally numeric approach with blocking

Consider sub-matrices of the GSO
$\Rightarrow \widetilde{\mathcal{O}}(n^4\beta)$ operations

Can we do better? [NSV10,PSV13?]

The totally numeric approach

LLL can be accelerated further by using approximations for the bases too!
$\Rightarrow \widetilde{\mathcal{O}}(n^5\beta^{1.5})$ operations

The totally numeric approach, continued

Do the same with several levels of recursion
$\Rightarrow \widetilde{\mathcal{O}}(n^5\beta)$ operations

The totally numeric approach with blocking

Consider sub-matrices of the GSO
$\Rightarrow \widetilde{\mathcal{O}}(n^4\beta)$ operations

Can we do better? [NSV10,PSV13?]

The totally numeric approach

LLL can be accelerated further by using approximations for the bases too!
$\Rightarrow \widetilde{\mathcal{O}}(n^5 \beta^{1.5})$ operations

The totally numeric approach, continued

Do the same with several levels of recursion
$\Rightarrow \widetilde{\mathcal{O}}(n^5 \beta)$ operations

The totally numeric approach with blocking

Consider sub-matrices of the GSO
$\Rightarrow \widetilde{\mathcal{O}}(n^4 \beta)$ operations

Plan of the talk

Plan of the talk:

1. Euclidean lattices
2. Applications of euclidean lattices
3. The LLL algorithm
4. Speeding up LLL
5. **Conclusion**

Open problems

On LLL:

- Lower the cost further: as fast as matrix multiplication?
- Improve current implementations

In the general area of lattices

- Faster algorithms computing shorter vectors than LLL
- Quantum algorithms
- Hardness proofs for worst-case lattice problems
- Hardness proofs for average-case lattice problems (crucial for lattice-based cryptography)

Open problems

On LLL:

- Lower the cost further: as fast as matrix multiplication?
- Improve current implementations

In the general area of lattices

- Faster algorithms computing shorter vectors than LLL
- Quantum algorithms
- Hardness proofs for worst-case lattice problems
- Hardness proofs for average-case lattice problems (crucial for lattice-based cryptography)

Open problems

On LLL:

- Lower the cost further: as fast as matrix multiplication?
- Improve current implementations

In the general area of lattices

- Faster algorithms computing shorter vectors than LLL
- Quantum algorithms
- Hardness proofs for worst-case lattice problems
- Hardness proofs for average-case lattice problems (crucial for lattice-based cryptography)

Open problems

On LLL:

- Lower the cost further: as fast as matrix multiplication?
- Improve current implementations

In the general area of lattices

- Faster algorithms computing shorter vectors than LLL
- Quantum algorithms
- Hardness proofs for worst-case lattice problems
- Hardness proofs for average-case lattice problems (crucial for lattice-based cryptography)

Open problems

On LLL:

- Lower the cost further: as fast as matrix multiplication?
- Improve current implementations

In the general area of lattices

- Faster algorithms computing shorter vectors than LLL
- Quantum algorithms
- Hardness proofs for worst-case lattice problems
- Hardness proofs for average-case lattice problems (crucial for lattice-based cryptography)