# Another L makes it better? Lagrange meets LLL and may improve BKZ pre-processing

**Sébastien BALNY**, Claire DELAPLACE, Gilles DEQUEN

# Context

**Today**

factorization or discrete logarithms

# Context

**Today**

factorization or discrete logarithms

New enemy

Quantum computer

Today

~~factorization or discrete logarithms~~

New enemy

Quantum computer

# Context

Today

~~factorization or discrete logarithms~~

New enemy

Quantum computer

Solutions

Creating new schemes that are resistant to quantum attack.

# Context

## Today

~~factorization or discrete logarithms~~

New enemy

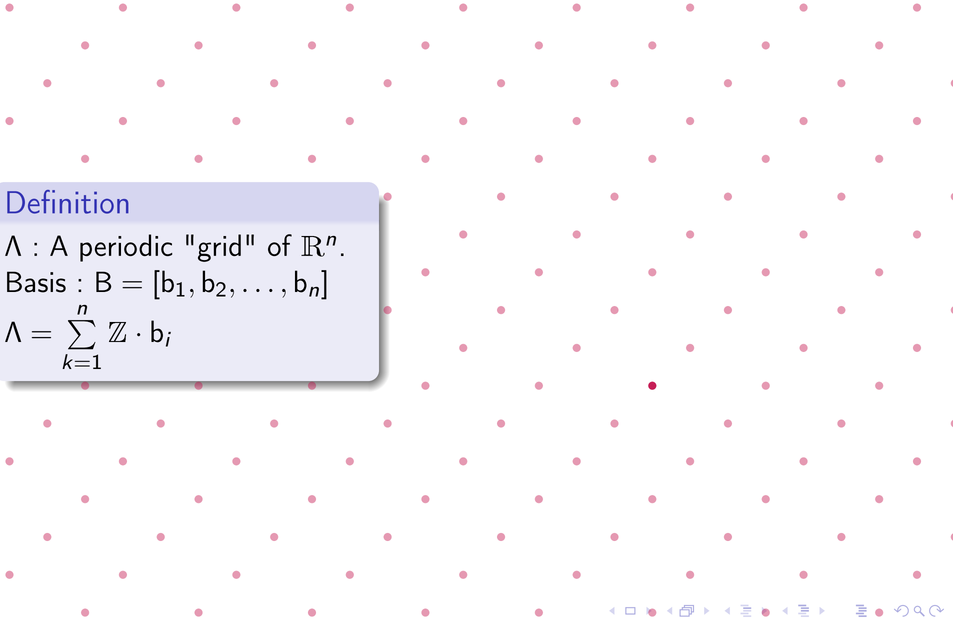
Quantum computer

## Solutions

Creating new schemes that are resistant to quantum attack.

$\boxed{\text{LATTICE BASED CRYPTOGRAPHY}}$

# What is a lattice ?



**Definition**

$\Lambda$ : A periodic "grid" of $\mathbb{R}^n$.

Basis : $B = [b_1, b_2, \ldots, b_n]$

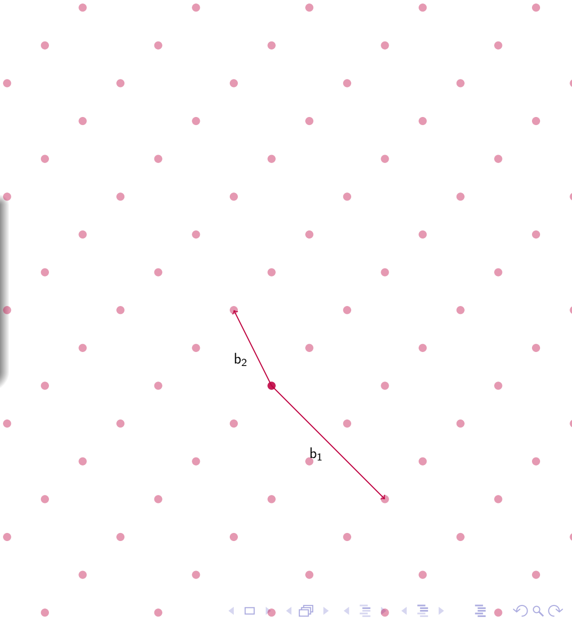
$$\Lambda = \sum_{k=1}^{n} \mathbb{Z} \cdot b_i$$

# What is a lattice ?

### Definition

$\Lambda$ : A periodic "grid" of $\mathbb{R}^n$.
Basis : $B = [b_1, b_2, \ldots, b_n]$

$$\Lambda = \sum_{k=1}^{n} \mathbb{Z} \cdot b_i$$



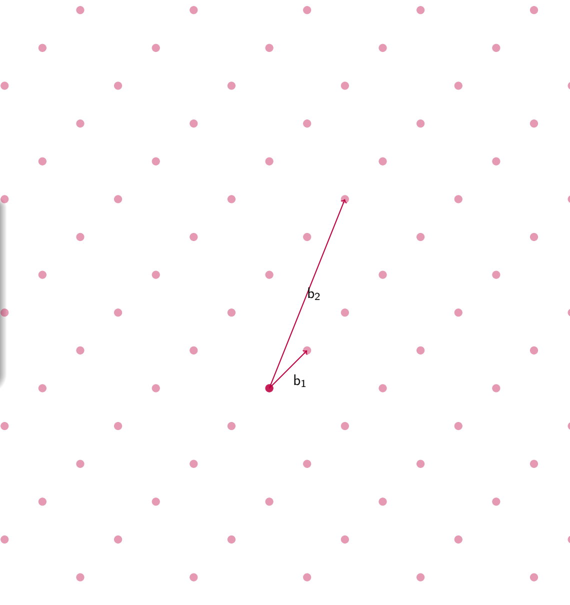
# What is a lattice ?

## Definition

$\Lambda$ : A periodic "grid" of $\mathbb{R}^n$.

Basis : $B = [b_1, b_2, \ldots, b_n]$

$$\Lambda = \sum_{k=1}^{n} \mathbb{Z} \cdot b_i$$



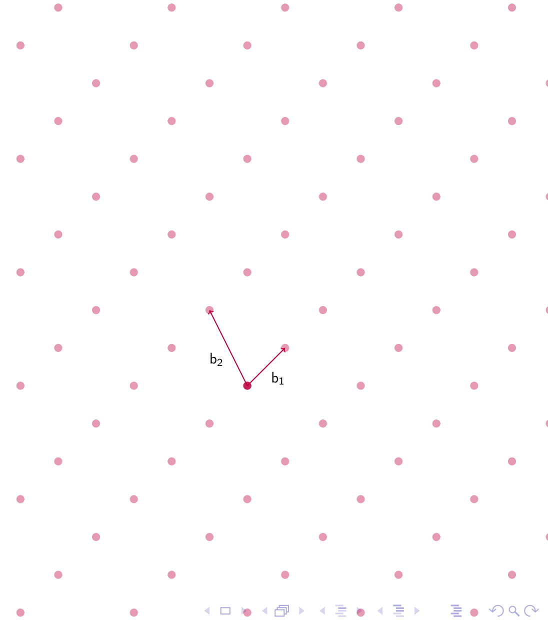
# What is a lattice ?

**Definition**

$\Lambda$ : A periodic "grid" of $\mathbb{R}^n$.

Basis : $B = [b_1, b_2, \ldots, b_n]$

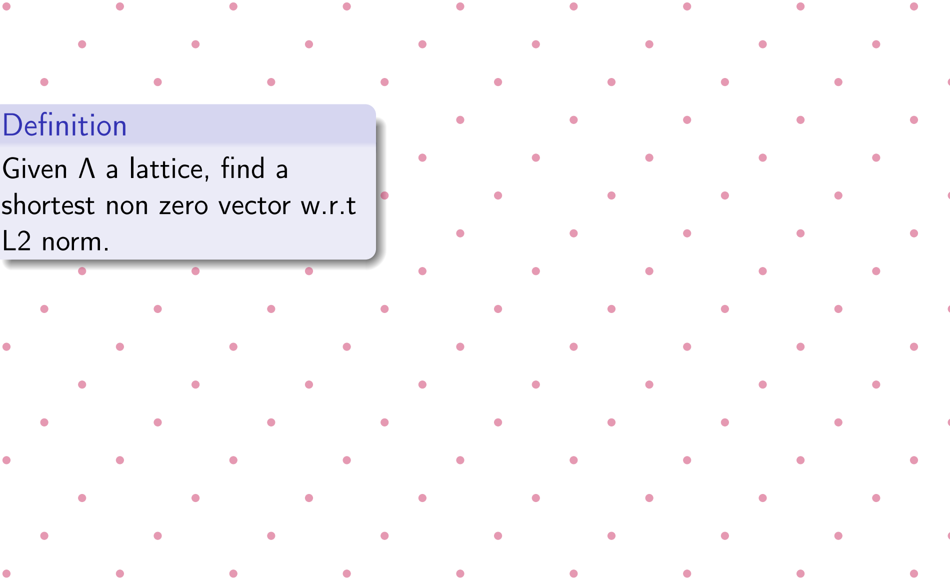
$$\Lambda = \sum_{k=1}^{n} \mathbb{Z} \cdot b_i$$



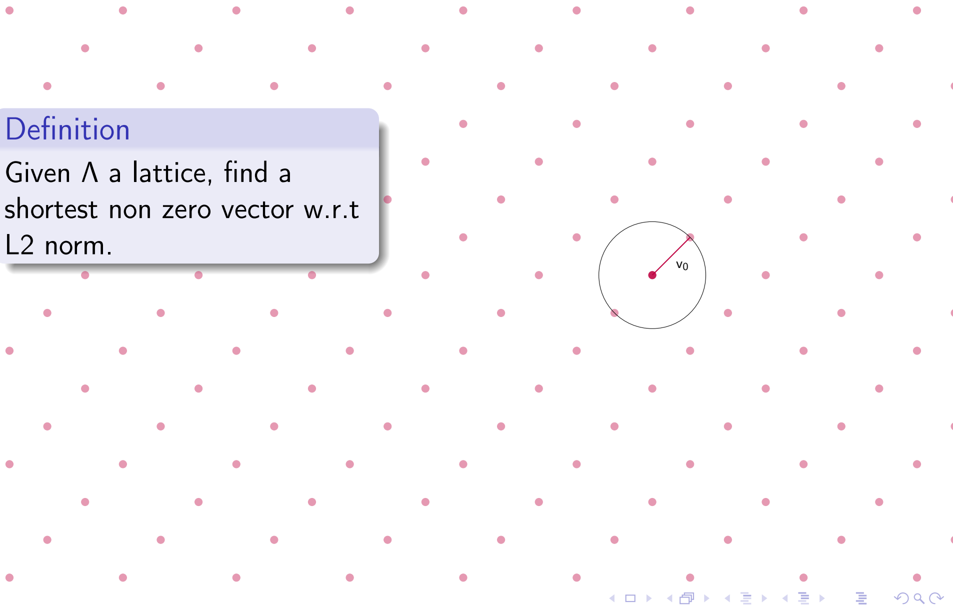
# The shortest vector problem

**Definition**

Given $\Lambda$ a lattice, find a shortest non zero vector w.r.t L2 norm.



# The shortest vector problem



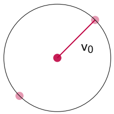

**Definition**

Given $\Lambda$ a lattice, find a shortest non zero vector w.r.t L2 norm.

# The shortest vector problem

**Definition**

Given $\Lambda$ a lattice, find a shortest non zero vector w.r.t L2 norm.




$\Downarrow$

NP-hard [Ajtai1996]

# The shortest vector problem

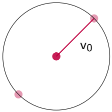
**Definition**

Given $\Lambda$ a lattice, find a shortest non zero vector w.r.t L2 norm.

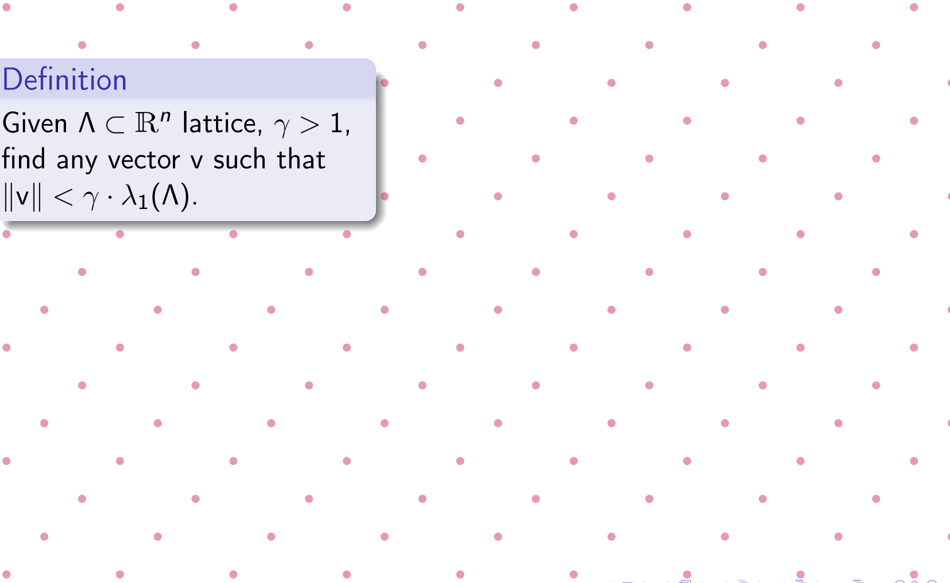
$\Downarrow$

NP-hard [Ajtai1996]

$$\boxed{\|v_0\| = \lambda_1(\Lambda)}$$



# The approximate Shortest Vector Problem



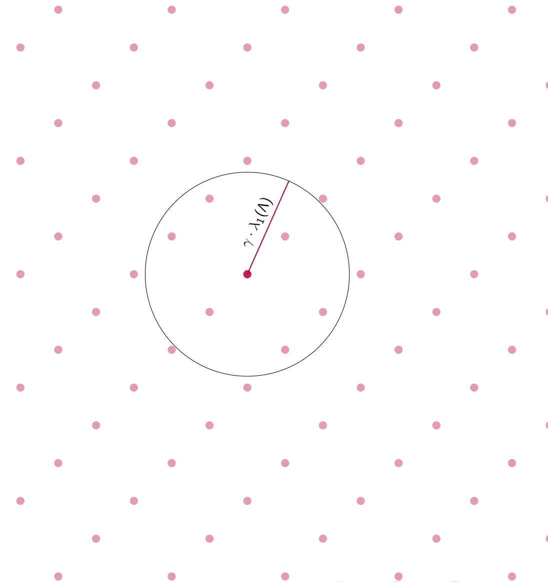
**Definition**

Given $\Lambda \subset \mathbb{R}^n$ lattice, $\gamma > 1$, find any vector v such that $\|v\| < \gamma \cdot \lambda_1(\Lambda)$.

# The approximate Shortest Vector Problem

## Definition

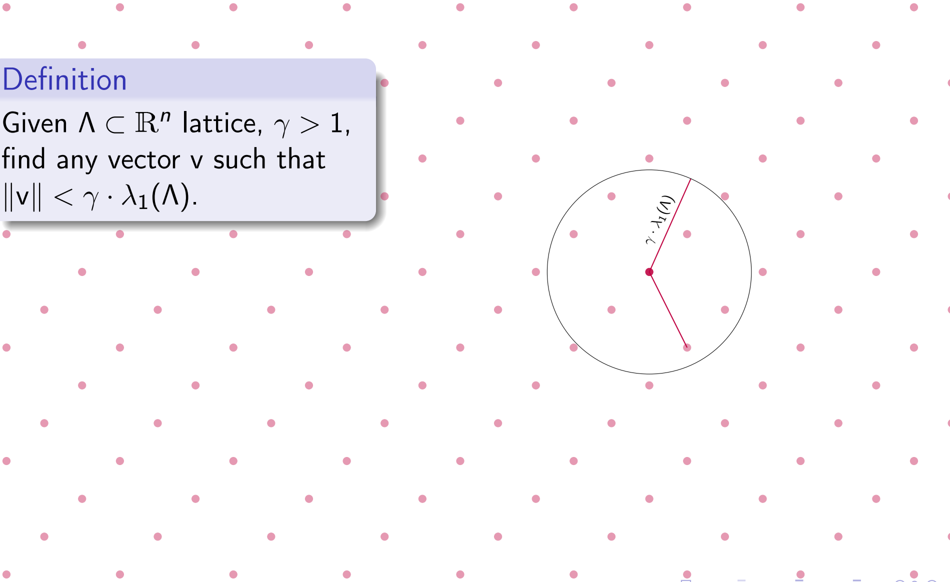
Given $\Lambda \subset \mathbb{R}^n$ lattice, $\gamma > 1$, find any vector v such that $\|\mathrm{v}\| < \gamma \cdot \lambda_1(\Lambda)$.



# The approximate Shortest Vector Problem

**Definition**

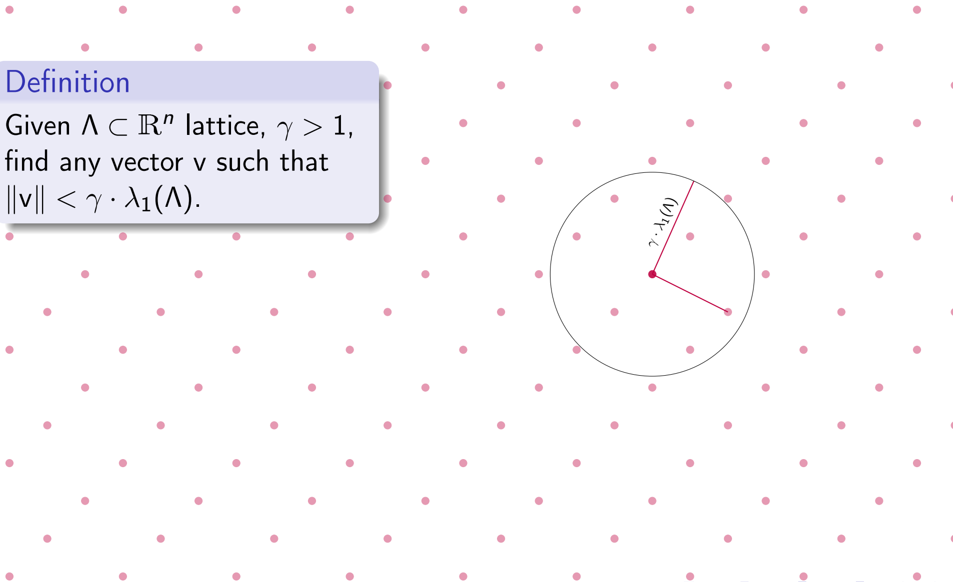
Given $\Lambda \subset \mathbb{R}^n$ lattice, $\gamma > 1$, find any vector v such that $\|v\| < \gamma \cdot \lambda_1(\Lambda)$.



# The approximate Shortest Vector Problem

### Definition

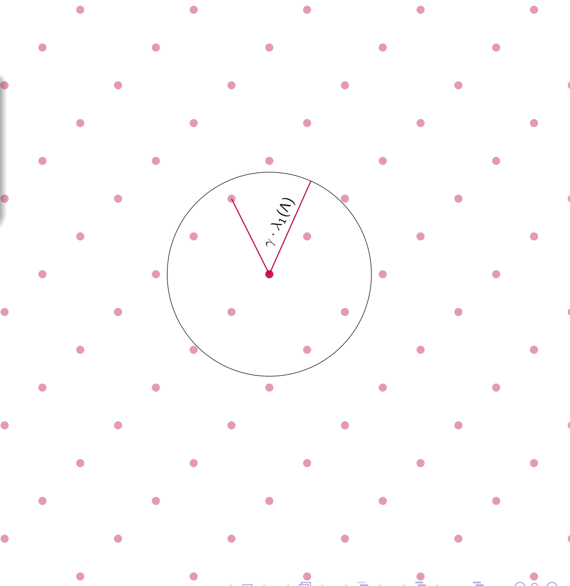
Given $\Lambda \subset \mathbb{R}^n$ lattice, $\gamma > 1$, find any vector v such that $\|\mathrm{v}\| < \gamma \cdot \lambda_1(\Lambda)$.



# The approximate Shortest Vector Problem

**Definition**

Given $\Lambda \subset \mathbb{R}^n$ lattice, $\gamma > 1$, find any vector v such that $\|v\| < \gamma \cdot \lambda_1(\Lambda)$.

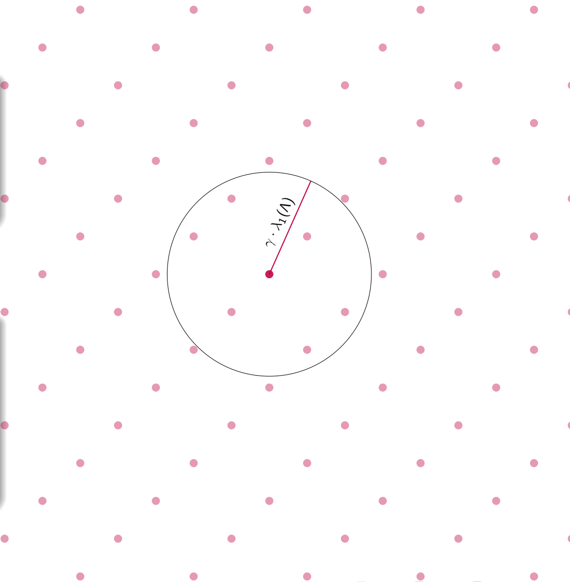


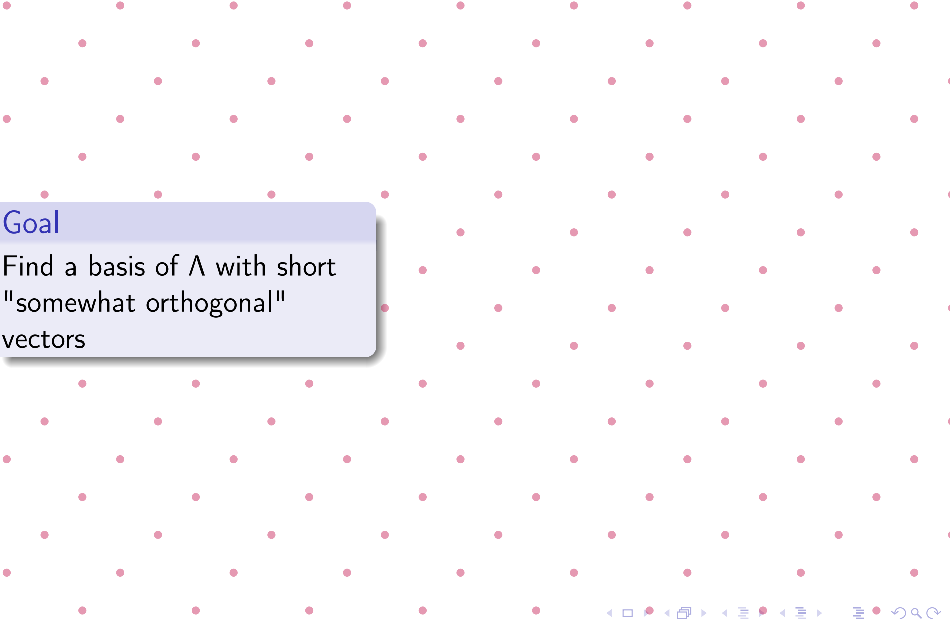
# The approximate Shortest Vector Problem

**Definition**

Given $\Lambda \subset \mathbb{R}^n$ lattice, $\gamma > 1$, find any vector v such that $\|\mathrm{v}\| < \gamma \cdot \lambda_1(\Lambda)$.

- $\gamma = \Theta(2^n) \Rightarrow \mathrm{poly}(n)$ algo [LLL1982]
- $\gamma = \Omega(n/\log(n)) \Rightarrow$ NP-hard [GG2000]

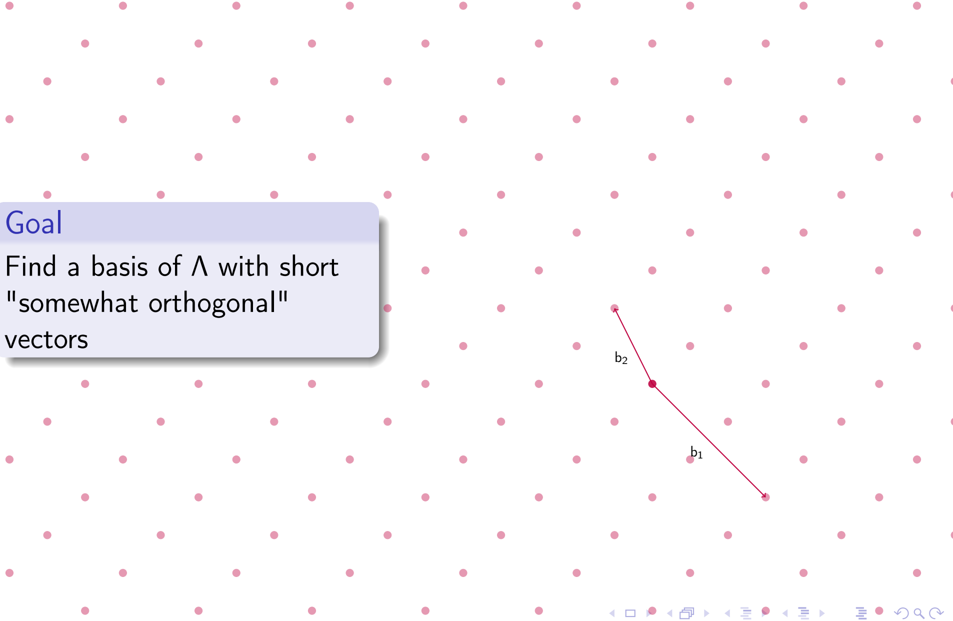




**Goal**

Find a basis of $\Lambda$ with short "somewhat orthogonal" vectors

# Lattice reduction

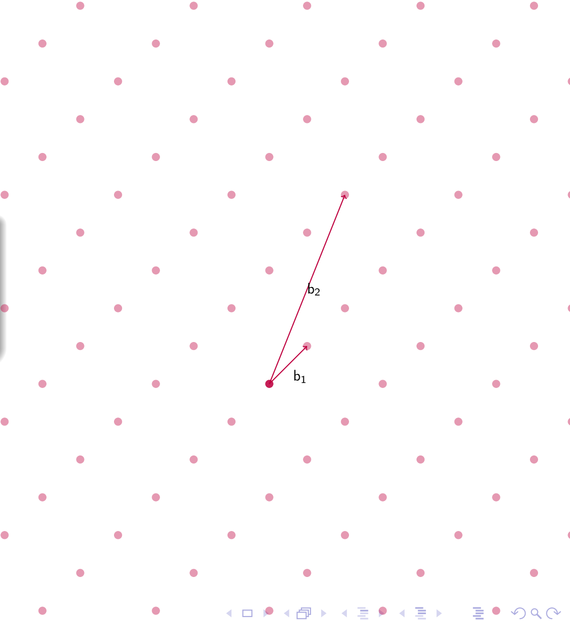


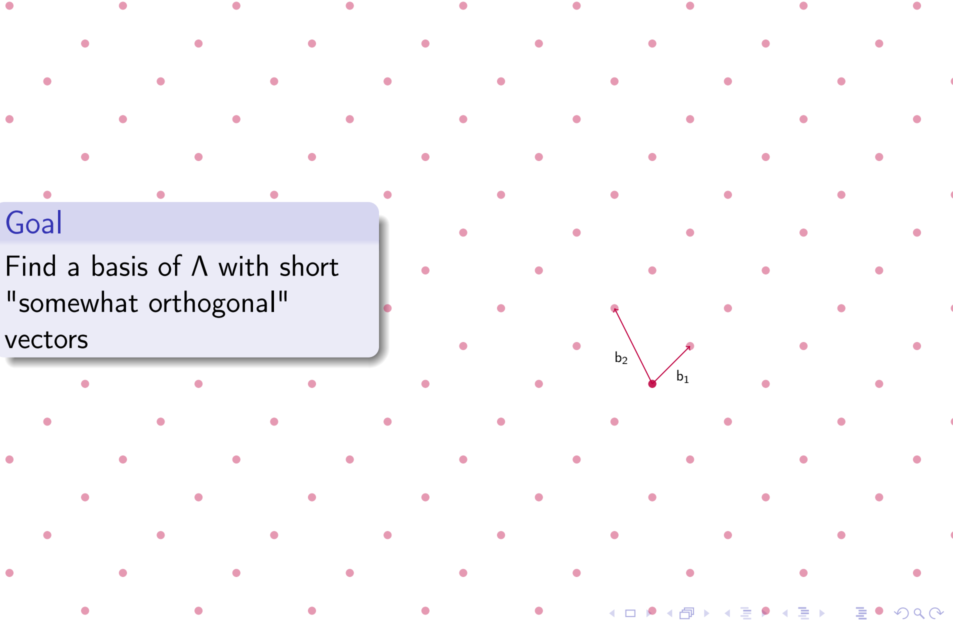

## Goal

Find a basis of $\Lambda$ with short "somewhat orthogonal" vectors

## Goal

Find a basis of Λ with short "somewhat orthogonal" vectors






### Goal

Find a basis of $\Lambda$ with short "somewhat orthogonal" vectors

# Lattice reduction

**Lagrange algorithm [Lagrange1773]**

Compute the most reduced basis en dim 2.

# Lattice reduction

## Lagrange algorithm [Lagrange1773]

Compute the most reduced basis en dim 2.

## Lenstra-Lenstra-Lovasz (LLL) [LLL1982]

Efficient in low dimension.

- Runtime : $n^{\mathcal{O}(1)}$
- memory : $\mathcal{O}(n)$

# Lattice reduction

## Lagrange algorithm [Lagrange1773]

Compute the most reduced basis en dim 2.

## Lenstra-Lenstra-Lovasz (LLL) [LLL1982]

Efficient in low dimension.

- Runtime : $n^{\mathcal{O}(1)}$
- memory : $\mathcal{O}(n)$

## Block-Korkine-Zolotarev (BKZ-$\beta$) [Schnorr1987]

$\beta$ offers a trade-off between quality and efficiency.

- Runtime : after a number of tours at most $\Theta(n^2 \log n/\beta^2)$ the first basis vector of BKZ is short. Tour complexity : $2^{\mathcal{O}(\beta^2)}$. [LN2020]
- memory : $n^{\mathcal{O}(1)}$

# dimension 2 : The Lagrange algorithm

Input : Given a basis $B = [b_1, b_2]$.
Output : A reduced basis $[b_1, b_2]$.

While $|{b_2}^t b_1| > \min(\|b_1\|^2, \|b_2\|^2)/2$

1. $b_2 \leftarrow b_2 - \lfloor \mu_{2,1} \rceil b_1$
   with $\mu_{2,1} = \dfrac{|{b_2}^t b_1|}{\|b_1\|^2}$
2. If $\|b_2\|^2 < \|b_1\|^2$ swap the two vectors and go to step 1.

# dimension 2 : The Lagrange algorithm

Input : Given a basis $B = [b_1, b_2]$.
Output : A reduced basis $[b_1, b_2]$.

While $|{b_2}^t b_1| > \min(\|b_1\|^2, \|b_2\|^2)/2$

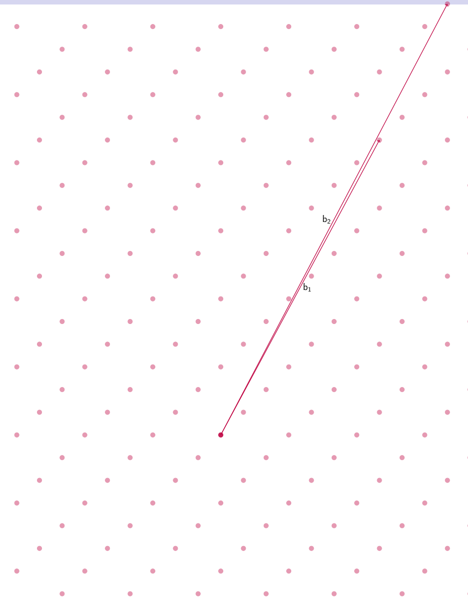
1. $b_2 \leftarrow b_2 - \lfloor \mu_{2,1} \rceil b_1$
   with $\mu_{2,1} = \dfrac{|{b_2}^t b_1|}{\|b_1\|^2}$
2. If $\|b_2\|^2 < \|b_1\|^2$ swap the two vectors and go to step 1.

# dimension 2 : The Lagrange algorithm

Input : Given a basis $B = [b_1, b_2]$.
Output : A reduced basis $[b_1, b_2]$.

While $|{b_2}^t b_1| > \min(\|b_1\|^2, \|b_2\|^2)/2$

1. $b_2 \leftarrow b_2 - \lfloor \mu_{2,1} \rceil b_1$
   with $\mu_{2,1} = \dfrac{|{b_2}^t b_1|}{\|b_1\|^2}$
2. If $\|b_2\|^2 < \|b_1\|^2$ swap the two vectors and go to step 1.

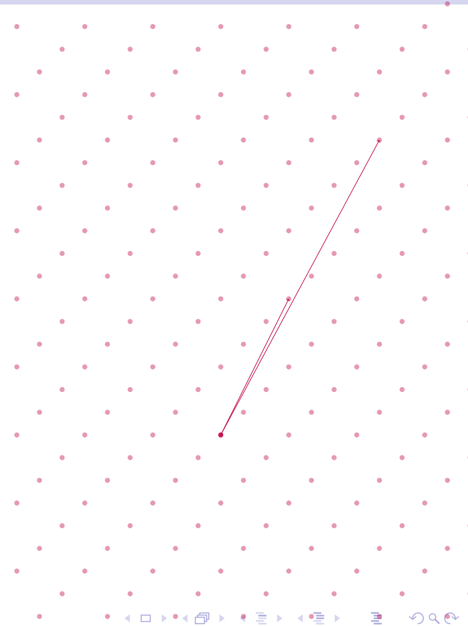


# dimension 2 : The Lagrange algorithm

Input : Given a basis $B = [b_1, b_2]$.
Output : A reduced basis $[b_1, b_2]$.

While $|{b_2}^t b_1| > \min(\|b_1\|^2, \|b_2\|^2)/2$

1. $b_2 \leftarrow b_2 - \lfloor \mu_{2,1} \rceil b_1$
   with $\mu_{2,1} = \dfrac{|{b_2}^t b_1|}{\|b_1\|^2}$
2. If $\|b_2\|^2 < \|b_1\|^2$ swap the two vectors and go to step 1.

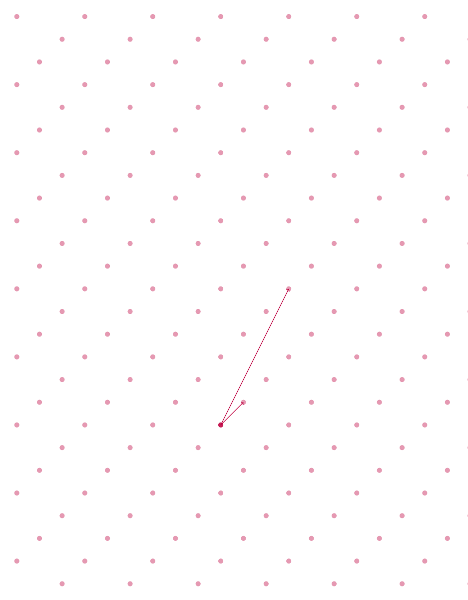


# dimension 2 : The Lagrange algorithm

Input : Given a basis $B = [b_1, b_2]$.
Output : A reduced basis $[b_1, b_2]$.

While $|{b_2}^t b_1| > \min(\|b_1\|^2, \|b_2\|^2)/2$

1. $b_2 \leftarrow b_2 - \lfloor \mu_{2,1} \rceil b_1$
   with $\mu_{2,1} = \dfrac{|{b_2}^t b_1|}{\|b_1\|^2}$
2. If $\|b_2\|^2 < \|b_1\|^2$ swap the two vectors and go to step 1.

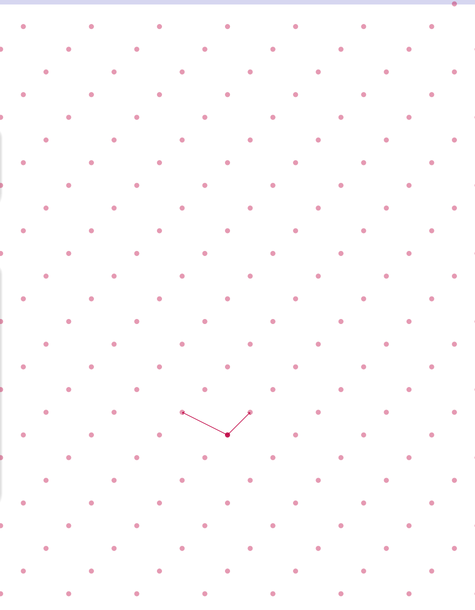


# dimension 2 : The Lagrange algorithm

Input : Given a basis $B = [b_1, b_2]$.
Output : A reduced basis $[b_1, b_2]$.

While $|{b_2}^t b_1| > \min(\|b_1\|^2, \|b_2\|^2)/2$

1. $b_2 \leftarrow b_2 - \lfloor \mu_{2,1} \rceil b_1$
   with $\mu_{2,1} = \dfrac{|{b_2}^t b_1|}{\|b_1\|^2}$
2. If $\|b_2\|^2 < \|b_1\|^2$ swap the two vectors and go to step 1.



# To sum up

- Lattice reduction uses (approx-)SVP oracles ;

# To sum up

- Lattice reduction uses (approx-)SVP oracles ;
- (approx-)SVP easier when input basis already reduced.

# Definitions

## L-reduction

A pair of vectors $(u, v)$ is L-reduced or Lagrange-reduced if

$$\|u \pm v\| \geq \max(\|u\|, \|v\|)$$

# Definitions

## L-reduction

A pair of vectors $(u, v)$ is L-reduced or Lagrange-reduced if

$$\|u \pm v\| \geq \max(\|u\|, \|v\|)$$

## Pair-wise L-reduction

A set of linearly independent vectors $\mathcal{S}$ is said to be L-reduced if for all $(u, v) \in \mathcal{S}^2$, $(u, v)$ is L-reduced.

# Definitions

### L-reduction

A pair of vectors $(u, v)$ is L-reduced or Lagrange-reduced if

$$\|u \pm v\| \geq \max(\|u\|, \|v\|)$$

### Pair-wise L-reduction

A set of linearly independent vectors $\mathcal{S}$ is said to be L-reduced if for all $(u, v) \in \mathcal{S}^2$, $(u, v)$ is L-reduced.

$$\boxed{B \text{ LLL-reduced} \quad \not\Rightarrow \quad B \text{ L-reduced.}}$$

# Our work

A new algorithm inspired by **LLL** and **Lagrange**.

# Our work

A new algorithm inspired by **LLL** and **Lagrange**.

- Test Lattices : Darmstadt SVP Challenge generator

  `https://www.latticechallenge.org/svp-challenge/`

  1000 lattices per dim : from 40 to 200, step of 10 (in almost cases)
- Comparison between our result $v_0$ and $\tilde{\lambda}_1 \approx \lambda_1(\Lambda)$ :

$$\boxed{\text{approx factor} = \frac{\|v_0\|}{\tilde{\lambda}_1}}$$

- Python implementation using FpyLLL library on MatriCS HPC Platform : `https://www.matrics.u-picardie.fr/`

# Our new L4 (Lagrange-LLL) algorithm

Input : A LLL-reduced basis B.
Output : A better LLL-reduced basis $B'$ with $\|b_1'\| \leq \|b_1\|$.

# Our new L4 (Lagrange-LLL) algorithm

Input : A LLL-reduced basis B.
Output : A better LLL-reduced basis B′ with $\|b'_1\| \leq \|b_1\|$.

**Steps :**

1. Compute a set $S = \mathrm{B} \cup \{b_i \pm b_j \mid \#(b_i, b_j) \in \mathrm{B} \times \mathrm{B}\}$ ;
2. $\mathrm{B} = \mathrm{LLLReduce}(S)$ ;
3. Repeat step 1 and step 2 as long as $\|b_1\|$ is decreasing.

# Our new L4 (Lagrange-LLL) algorithm

Input : A LLL-reduced basis B.
Output : A better LLL-reduced basis B′ with $\|\mathrm{b}'_1\| \leq \|\mathrm{b}_1\|$.

**Steps :**

1. Compute a set $S = \mathrm{B} \cup \{b_i \pm b_j \mid (b_i, b_j)\text{not L-reduced}\}$ ;
2. $\mathrm{B} = \mathrm{LLLReduce}(S)$ ;
3. Repeat step 1 and step 2 as long as $\|\mathrm{b}_1\|$ is decreasing.

# Construction of $S$

**Sample L4**

1. $S \leftarrow \mathrm{B}$ ;
2. Repeat $\alpha_1 n$ times :
   a. $\mathrm{u} = \mathrm{RandomChoose}(S)$ ;
   b. Repeat $\alpha_2 n$ times :
      i. $\mathrm{v} = \mathrm{RandomChoose}(S)$ ;
      ii. If $0 < \|\mathrm{u} \pm \mathrm{v}\| \leq \max(\|\mathrm{u}\|; \|\mathrm{v}\|)\|$ do
      $S \leftarrow S \cup \{\mathrm{u} \pm \mathrm{v}\}$ ;
      done
3. $S \leftarrow \mathrm{Sort}(S)$.

# Construction of $S$

**Sample L4**

1. $S \leftarrow \mathsf{B}$ ;
2. Repeat $\alpha_1 n$ times :
   a. $\mathsf{u} = \mathsf{RandomChoose}(S)$ ;
   b. Repeat $\alpha_2 n$ times :
      i. $\mathsf{v} = \mathsf{RandomChoose}(S)$ ;
      ii. If $0 < \|\mathsf{u} \pm \mathsf{v}\| \leq \max(\|\mathsf{u}\|; \|\mathsf{v}\|)\|$ do
      $S \leftarrow S \cup \{\mathsf{u} \pm \mathsf{v}\}$ ;
      done
3. $S \leftarrow \mathsf{Sort}(S)$.

$\alpha_1 = 1$ and $\alpha_2 = 1/2 \quad \Longrightarrow \quad$ faster than LLL.

# Our new L4 (Lagrange-LLL) algorithm

Input : A LLL-reduced basis B.
Output : A better LLL-reduced basis B′ with $\|b'_1\| \leq \|b_1\|$.

**Steps :**

1. Compute a set $S = B \cup \{b_i \pm b_j \mid (b_i, b_j) \text{not L-reduced}\}$ ;
2. $B = \mathrm{LLLReduce}(S)$ ;
3. Repeat step 1 and step 2 as long as $\|b_1\|$ is decreasing.

# Our new L4 (Lagrange-LLL) algorithm

Input : A LLL-reduced basis B.
Output : A better LLL-reduced basis B′ with $\|b'_1\| \leq \|b_1\|$.

**Steps :**

1. Compute a set $S = \mathrm{B} \cup \{b_i \pm b_j \mid (b_i, b_j) \text{not L-reduced}\}$ ;
2. $\mathrm{B} = \mathrm{LLLReduce}(S)$ ; $\longleftarrow$ complexity $C_{LLL}$
3. Repeat step 1 and step 2 as long as $\|b_1\|$ is decreasing.

# Our new L4 (Lagrange-LLL) algorithm

Input : A LLL-reduced basis B.
Output : A better LLL-reduced basis B′ with $\|b'_1\| \leq \|b_1\|$.

**Steps :**

1. Compute a set $S = \mathrm{B} \cup \{b_i \pm b_j \mid (b_i, b_j) \text{not L-reduced}\}$ ; $\ll C_{LLL}$
2. $\mathrm{B} = \mathrm{LLLReduce}(S)$ ; $\longleftarrow$ complexity $C_{LLL}$
3. Repeat step 1 and step 2 as long as $\|b_1\|$ is decreasing.

# Our new L4 (Lagrange-LLL) algorithm

Input : A LLL-reduced basis B.
Output : A better LLL-reduced basis B′ with $\|b'_1\| \leq \|b_1\|$.

**Steps :**

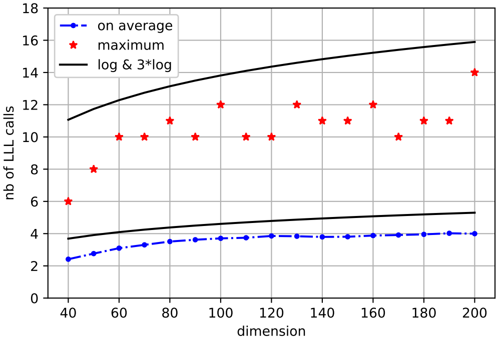
1. Compute a set $S = \mathrm{B} \cup \{b_i \pm b_j \mid (b_i, b_j) \text{not L-reduced}\}$ ; $\ll C_{LLL}$
2. $\mathrm{B} = \mathrm{LLLReduce}(S)$ ; $\longleftarrow$ complexity $C_{LLL}$
3. Repeat step 1 and step 2 as long as $\|b_1\|$ is decreasing.

**Complexity**

Time complexity : $\mathcal{O}(k \times C_{LLL})$

$k$ number of calls to LLL

# number of LLL calls

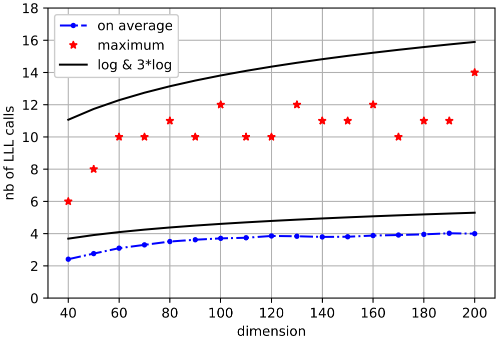



1000 tests/dim

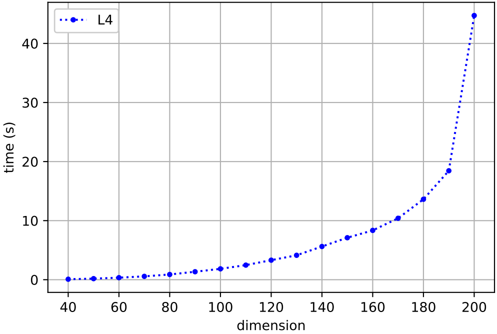
# number of LLL calls

## Conjecture

$$k = \mathcal{O}(\log(n))$$

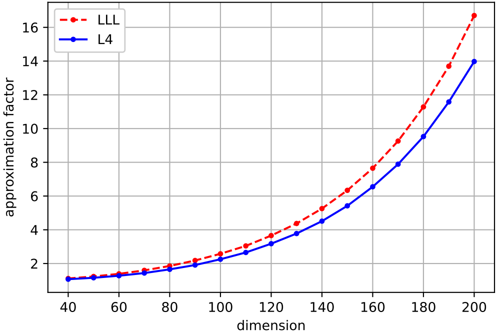



1000 tests/dim




1000 tests/dim

# L4 : Norm of the First Vector




1000 tests/dim

# Randomization

## Equivalent bases

$\Lambda = \mathcal{L}(\mathrm{B}_1) = \mathcal{L}(\mathrm{B}_2) \iff \mathrm{B}_2 = \mathrm{B}_1 \times U$ , for some $U$ unimodular.

# Randomization

## Equivalent bases

$$\Lambda = \mathcal{L}(B_1) = \mathcal{L}(B_2) \iff B_2 = B_1 \times U \text{ , for some } U \text{ unimodular.}$$

## Randomization

Input : A basis $B_1$.

Output : A new basis $B_2$.

1. Generate randomly $U$ such that $\det U = \pm 1$;
2. Compute $B_2 = B_1 \times U$.

### Equivalent bases

$\Lambda = \mathcal{L}(\mathrm{B}_1) = \mathcal{L}(\mathrm{B}_2) \quad \Longleftrightarrow \quad \mathrm{B}_2 = \mathrm{B}_1 \times U$ , for some $U$ unimodular.

### Randomization

Input : A basis $\mathrm{B}_1$.
Output : A new basis $\mathrm{B}_2$.

1. Generate randomly $U$ such that $\det U = \pm 1$ ;
2. Compute $\mathrm{B}_2 = \mathrm{B}_1 \times U$.

- L4-Rand $k$ : $k$ randomizations ;

# Randomization

## Equivalent bases

$\Lambda = \mathcal{L}(B_1) = \mathcal{L}(B_2) \iff B_2 = B_1 \times U$ , for some $U$ unimodular.

## Randomization

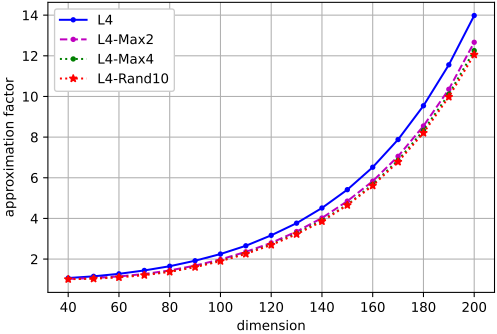
Input : A basis $B_1$.

Output : A new basis $B_2$.

1. Generate randomly $U$ such that $\det U = \pm 1$ ;
2. Compute $B_2 = B_1 \times U$.

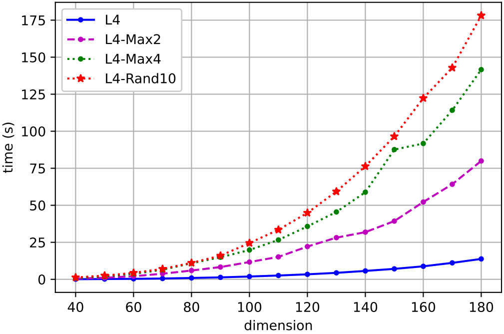
- L4-Rand $k$ : $k$ randomizations ;
- L4-Max $k$ : stop if no improvement after $k$ randomizations.

# Randomization : Norm of the First Vector




1000 tests/dim

# Randomization : Experimental runtime




1000 tests/dim

# Darmstadt SVP Challenge : $\|b_1\| \leq 1.05\tilde{\lambda}_1$ ?

| Dimension | LLL | L4 | L4-Max2 | L4-Max4 | L4-Rand10 |
|---|---|---|---|---|---|
| 40 | 161 | 355 | 760 | 842 | 915 |
| 50 | 9 | 64 | 318 | 544 | 626 |
| 60 | 0 | 4 | 34 | 103 | 93 |
| 70 | 0 | 0 | 1 | 2 | 4 |
| 80 | 0 | 0 | 0 | 0 | 0 |
| 90 | 0 | 0 | 0 | 0 | 0 |

$\vdots$

1000 tests/dim

# L4 VS BKZ

## L4 VS BKZ-12

$\rightarrow$ similar runtime

$\rightarrow$ worst approximation factor

# L4 VS BKZ

## L4 VS BKZ-12

$\rightarrow$ similar runtime

$\rightarrow$ worst approximation factor

## Idea 1 : Using BKZ instead of LLL in L4

Not working : Too many L-reduced basis vectors

# L4 VS BKZ

## L4 VS BKZ-12

$\rightarrow$ similar runtime

$\rightarrow$ worst approximation factor

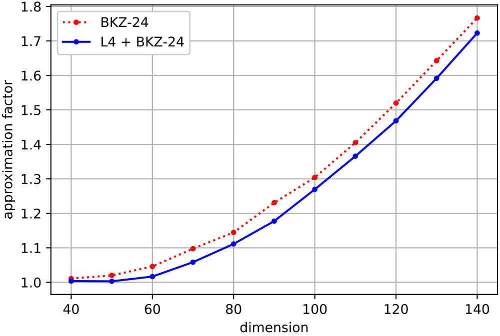
## Idea 1 : Using BKZ instead of LLL in L4

Not working : Too many L-reduced basis vectors

## Idea 2 : Using L4 instead of LLL in the pre-computation of BKZ

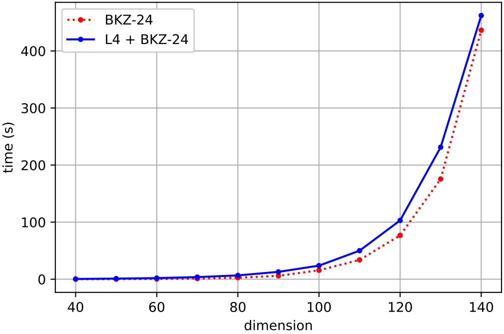
**Better results ! ! !**

# Average approximation factor




100 tests/dim

# Average runtime




100 tests/dim

# Our results

## L4+BKZ-24

- approx factor 3% better on average.
- proportion of improved basis $\nearrow$ with dim.

# Our results

## L4+BKZ-24

- approx factor 3% better on average.
- proportion of improved basis $\nearrow$ with dim.

## BKZ-24 after L4 VS BKZ-24 after LLL

- In 35$\sim$45 % of the cases, our method improves the runtime.

# Our results

## L4+BKZ-24

- approx factor 3% better on average.
- proportion of improved basis $\nearrow$ with dim.

## BKZ-24 after L4 VS BKZ-24 after LLL

- In 35$\sim$45 % of the cases, our method improves the runtime.

## L4+BKZ-24 VS BKZ-24

- For all dim, there are some cases where **everything is better**.

# Conclusion

- $\rightarrow$ A new algorithm inspired by LLL and Lagrange reduction
- $\rightarrow$ Improve the approximation factor for SVP when used as pre-processing of BKZ
- $\rightarrow$ In some cases faster than BKZ

# Conclusion

- → A new algorithm inspired by LLL and Lagrange reduction
- → Improve the approximation factor for SVP when used as pre-processing of BKZ
- → In some cases faster than BKZ

## Future work

- → Better implementation
- → Improve the sample
- → Including L4 in BKZ implementation

# Conclusion

→ A new algorithm inspired by LLL and Lagrange reduction

→ Improve the approximation factor for SVP when used as pre-processing of BKZ

→ In some cases faster than BKZ

## Future work

→ Better implementation

→ Improve the sample

→ Including L4 in BKZ implementation

# Thank you !

https://zenodo.org/records/13847623