$\widetilde{\mathrm{L}}^1$ a quasi-linear LLL algorithm

Andy Novocin
University of Waterloo
joint with
Damien Stehlé and Gilles Villard

FLINT Sage Days (35), December 18, 2011

Goals for this talk

- My goal: Be useful to the audience
- Two potential types:
  - competent but not LLL experts
  - LLL users, maybe experts
- A gift for non-experts: an LLL for your toolbox (over ambitious?)
- Reward for the others: the novel concepts in $\widetilde{L}^1$

Goals for this talk

- My goal: Be useful to the audience
- Two potential types:
  - competent but not LLL experts
  - LLL users, maybe experts
- A gift for non-experts: an LLL for your toolbox (over ambitious?)
- Reward for the others: the novel concepts in $\widetilde{\mathrm{L}}^1$

Goals for this talk

- My goal: Be useful to the audience
- Two potential types:
  - competent but not LLL experts
  - LLL users, maybe experts
- A gift for non-experts: an LLL for your toolbox (over ambitious?)
- Reward for the others: the novel concepts in $\widetilde{L}^1$

Goals for this talk

- My goal: Be useful to the audience
- Two potential types:
  - competent but not LLL experts
  - LLL users, maybe experts
- A gift for non-experts: an LLL for your toolbox (over ambitious?)
- Reward for the others: the novel concepts in $\widetilde{\mathrm{L}}^1$

Goals for this talk

- My goal: Be useful to the audience
- Two potential types:
  - competent but not LLL experts
  - LLL users, maybe experts
- A gift for non-experts: an LLL for your toolbox (over ambitious?)
- Reward for the others: the novel concepts in $\widetilde{\mathrm{L}}^1$

Goals for this talk

- My goal: Be useful to the audience
- Two potential types:
  - competent but not LLL experts
  - LLL users, maybe experts
- A gift for non-experts: an LLL for your toolbox (over ambitious?)
- Reward for the others: the novel concepts in $\widetilde{\mathrm{L}}^1$

LLL: A wonderful problem solving tool

To use LLL you must know when it’s possible to use LLL.

LLL: A wonderful problem solving tool

What type of problem can LLL attack?

LLL: A wonderful problem solving tool

What type of problem can LLL attack?

When you need to find an integer combination of $\{\text{some stuff}\}$ **which will satisfy** `some property`**.**

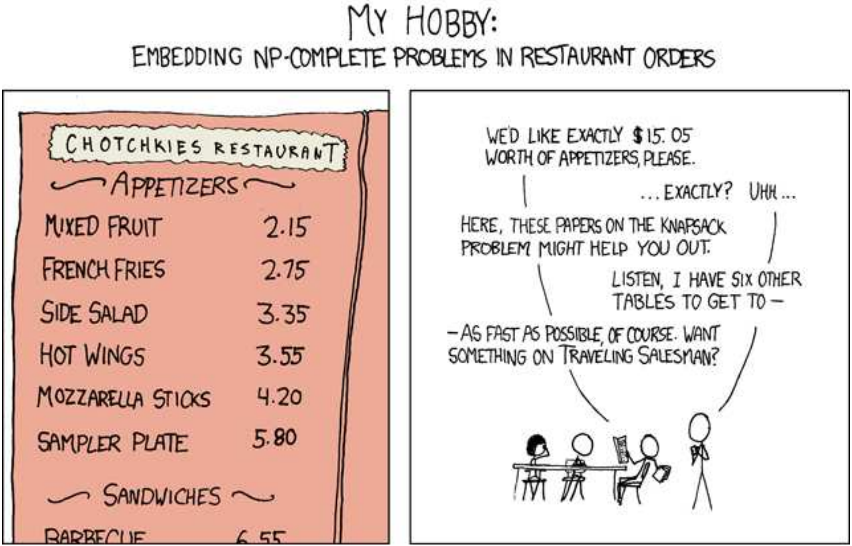
LLL: A wonderful problem solving tool

What type of problem can LLL attack?

When you need to find an integer combination of $\{\text{some stuff}\}$ **which will satisfy** `some property`**.**

Example Applications:

LLL: A wonderful problem solving tool



LLL: A wonderful problem solving tool

What type of problem can LLL attack?

When you need to find an integer combination of $\{\text{some stuff}\}$ **which will satisfy** `some property`**.**

Example Applications:

Subset-sum, Knapsack, variants, etc.

Find a combination of $2.15, 2.75, 3.35, 3.55, 4.20, 5.80$ which adds to exactly 15.05. (1 Mixed fruit, 2 orders of hot wings, and a sampler plate)

LLL: A wonderful problem solving tool

What type of problem can LLL attack?

When you need to find an integer combination of $\{\text{some stuff}\}$ **which will satisfy** `some property`**.**

Example Applications:

Minimal Polynomials

Given $\alpha \approx -.78447320 - 1.96117174 \cdot \sqrt{-1}$
find $\mathrm{minpoly}(\alpha)$. $(x^3 + 2x - 7)$

LLL: A wonderful problem solving tool

What type of problem can LLL attack?

When you need to find an integer combination of $\{\text{some stuff}\}$ **which will satisfy** `some property`**.**

Example Applications:

Algebraic number manipulation

Is there a combination of $\beta_1, \beta_2, \beta_3 \in \mathbb{Q}(\alpha)$ whose 23-adic image is $21 + 7 \cdot 23 + 11 \cdot 23^2 + \cdots$?

LLL: A wonderful problem solving tool

What type of problem can LLL attack?

When you need to find an integer combination of $\{\text{some stuff}\}$ **which will satisfy** `some property`**.**

Example Applications:

Diophantine Approximation

Given $r_1, \ldots, r_n \in \mathbb{R}$ find rationals which approximate them each with the same small denominator.

LLL: A wonderful problem solving tool

What type of problem can LLL attack?

When you need to find an integer combination of $\{\text{some stuff}\}$ **which will satisfy** `some property`**.**

Example Applications:

Euclidean Algorithm

Given a, b find $\gcd(a, b) = s \cdot a + t \cdot b$.

Obligatory lattice intro

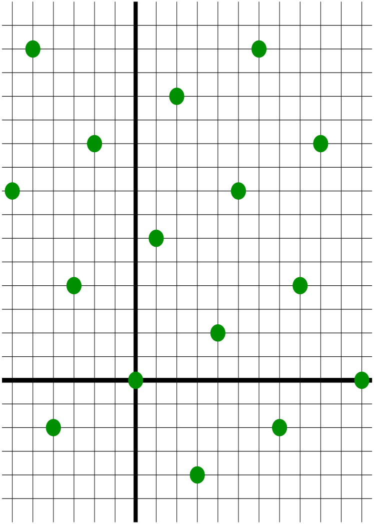
Lattice $\equiv$ discrete subgroup of $\mathbb{R}^n$

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

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:

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



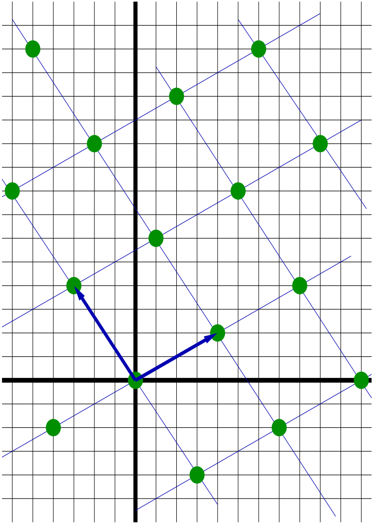
Obligatory lattice intro

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:

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



Obligatory lattice intro

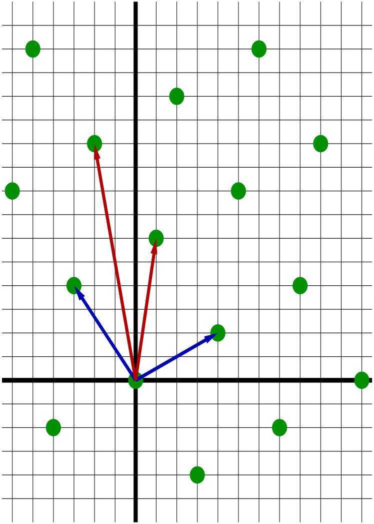
Lattice $\equiv$ discrete subgroup of $\mathbb{R}^n$

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

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

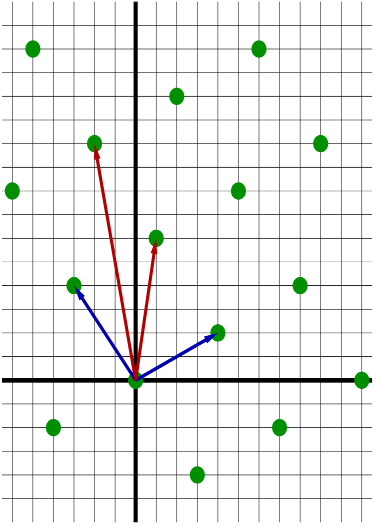
Bases are not unique, but they can be obtained from each other by integer transforms of determinant ± 1:

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



What LLL actually does.

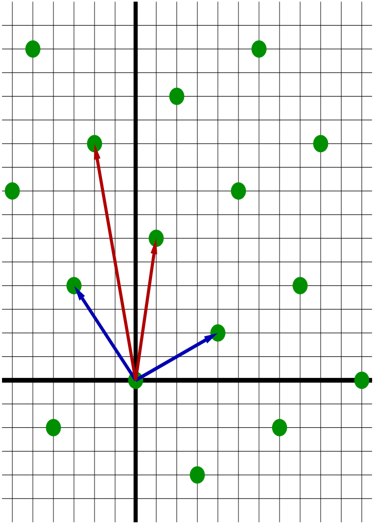
A lattice reduction algorithm is given some basis and attempts to find a better basis.



What LLL actually does.

A lattice reduction algorithm is given some basis and attempts to find a better basis.

The output is a reduced basis, which is somewhat orthogonal.

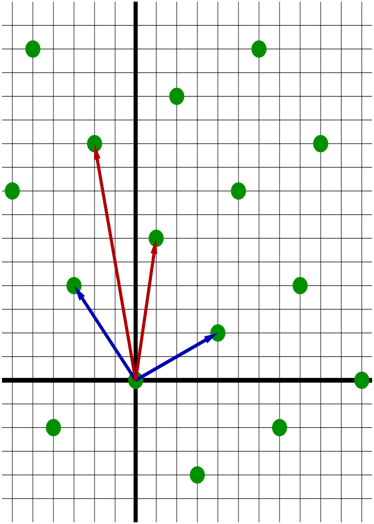


What LLL actually does.

A lattice reduction algorithm is given some basis and attempts to find a better basis.

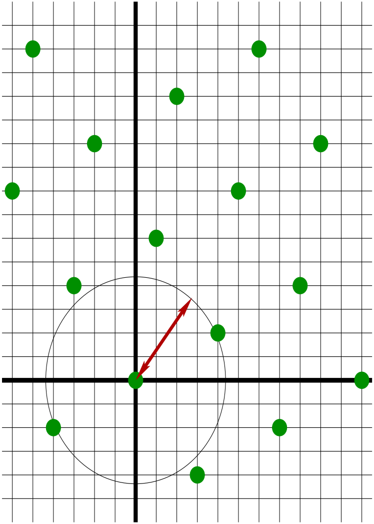
The output is a reduced basis, which is somewhat orthogonal.

In 1982 Lenstra, Lenstra, Lovász gave a polynomial time reduction algorithm (LLL).



What LLL actually does.

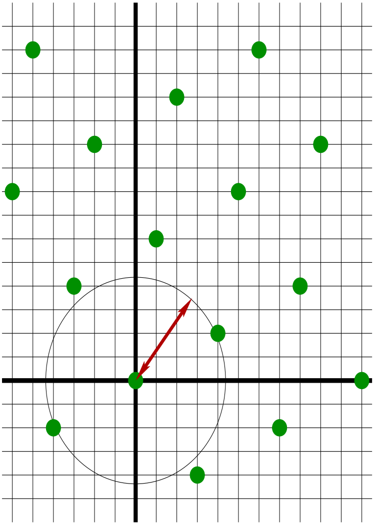
One Popular Lattice Question:



What LLL actually does.

One Popular Lattice Question:

Shortest non-zero vector (SVP)

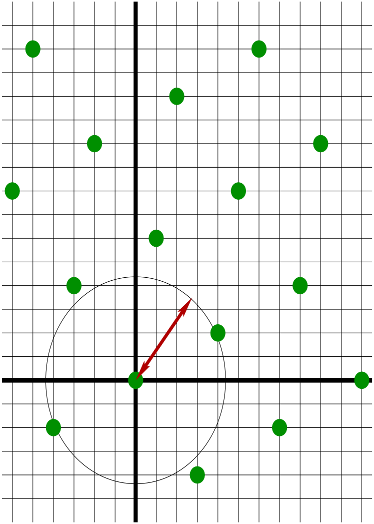


What LLL actually does.

One Popular Lattice Question:

Shortest non-zero vector (SVP)

Is NP-hard to find.



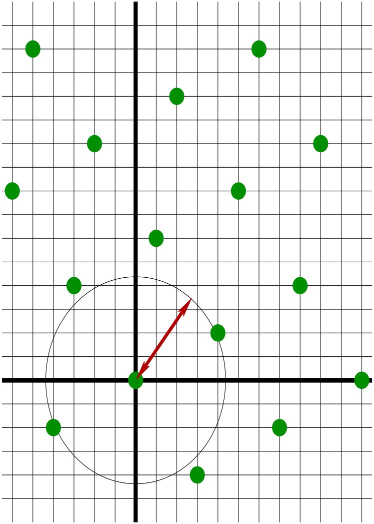
What LLL actually does.

One Popular Lattice Question:

Shortest non-zero vector (SVP)

Is NP-hard to find.

LLL approximately solves SVP in polynomial-time!



What LLL actually does.

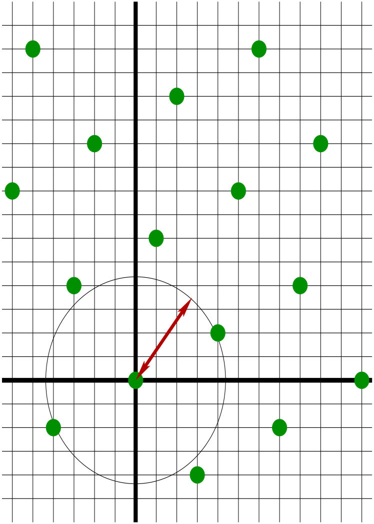
One Popular Lattice Question:

Shortest non-zero vector (SVP)

Is NP-hard to find.

LLL approximately solves SVP in polynomial-time!

When **lucky** and **creative**, approximate can be enough.



Examples of combination problems $\to$ lattice problems

Given an approximation $\alpha \approx -.78447320 + 1.96117174 \cdot \sqrt{-1}$.
Find a minimal polynomial for α.

Examples of combination problems → lattice problems

Given an approximation $\alpha \approx -.78447320 + 1.96117174 \cdot \sqrt{-1}$.
Find a minimal polynomial for α.

Make a lattice using $\alpha^0, \alpha^1, \alpha^2, \alpha^3$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 10000000000 & 0 \\ 0 & 1 & 0 & 0 & -7844732000 & -19611717400 \\ 0 & 0 & 1 & 0 & -32307963923 & 30769733412 \\ 0 & 0 & 0 & 1 & 85689463459 & 39223434588 \end{pmatrix}^T$$

Examples of combination problems $\to$ lattice problems

Given an approximation $\alpha \approx -.78447320 + 1.96117174 \cdot \sqrt{-1}$.
Find a minimal polynomial for α.

Make a lattice using $\alpha^0, \alpha^1, \alpha^2, \alpha^3$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 10000000000 & 0 \\ 0 & 1 & 0 & 0 & -7844732000 & -19611717400 \\ 0 & 0 & 1 & 0 & -32307963923 & 30769733412 \\ 0 & 0 & 0 & 1 & 85689463459 & 39223434588 \end{pmatrix}^T$$

Let $\texttt{minpoly}(\alpha) =: c_0 + c_1 x + c_2 x^2 + c_3 x^3$.

Examples of combination problems $\rightarrow$ lattice problems

Given an approximation $\alpha \approx -.78447320 + 1.96117174 \cdot \sqrt{-1}$.
Find a minimal polynomial for α.

Make a lattice using $\alpha^0, \alpha^1, \alpha^2, \alpha^3$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 10000000000 & 0 \\ 0 & 1 & 0 & 0 & -7844732000 & -19611717400 \\ 0 & 0 & 1 & 0 & -32307963923 & 30769733412 \\ 0 & 0 & 0 & 1 & 85689463459 & 39223434588 \end{pmatrix}^T$$

Let $\texttt{minpoly}(\alpha) =: c_0 + c_1 x + c_2 x^2 + c_3 x^3$.
Then $(c_0, c_1, c_2, c_3, \epsilon, \epsilon) \in L$ and is smaller in size than the other vectors.

Examples of combination problems $\rightarrow$ lattice problems

Given an approximation $\alpha \approx -.78447320 + 1.96117174 \cdot \sqrt{-1}$.
Find a minimal polynomial for α.

The first 2 vectors found by LLL are:

$$\begin{pmatrix} -7 & 2 & 0 & 1 & -541 & -212 \\ 84502 & -313827 & -101869 & -77000 & -106913 & 266772 \end{pmatrix}^T$$

Examples of combination problems $\to$ lattice problems

Given an approximation $\alpha \approx -.78447320 + 1.96117174 \cdot \sqrt{-1}$.
Find a minimal polynomial for α.

The first 2 vectors found by LLL are:

$$\begin{pmatrix} -7 & 2 & 0 & 1 & -541 & -212 \\ 84502 & -313827 & -101869 & -77000 & -106913 & 266772 \end{pmatrix}^T$$

We read this as saying that α is a root of $x^3 + 2x - 7$.

Another example of LLL solving a problem

For the knapsack menu problem we had to find a combination of $2.15, 2.75, 3.35, 3.55, 4.20, 5.80$ which adds to exactly 15.05.

The lattice I created for this one:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1505 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 215 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 275 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 335 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 355 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 420 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 580 \end{pmatrix}^T$$

Note that scaling up that last entry means that short vectors in the lattice will likely have 0 in the final column.

Another example of LLL solving a problem

For the knapsack menu problem we had to find a combination of 2.15, 2.75, 3.35, 3.55, 4.20, 5.80 which adds to exactly 15.05.

The output from LLL:

$$\begin{pmatrix} 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & -2 & -1 & 1 & 0 \\ 1 & -1 & 1 & 2 & 1 & 1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 2 & 0 & -1 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 1 & 1 & -1 & 0 & -5 \end{pmatrix}^T$$

The second vector is the solution.
The 0s in the final entries mean that this is difficult for LLL.

Intuitively, how does it work?

The input is a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

Intuitively, how does it work?

The input is a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

The goal is to push Gram-Schmidt length (length of a vector modulo the previous vectors) from early vectors to late vectors.

Intuitively, how does it work?

The input is a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

The goal is to push Gram-Schmidt length (length of a vector modulo the previous vectors) from early vectors to late vectors.

A reduced basis is, by definition, one in which G-S length never drops too fast.

Intuitively, how does it work?

The input is a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

Classical LLL works by making a succession of two elementary moves:

- **Size Reductions** Subtract integer multiples of early vectors from late vectors
- **Swaps** Switch the position of two basis vectors if a minimum amount of G-S length can be pushed.

Intuitively, how does it work?

The input is a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

Classical LLL works by making a succession of two elementary moves:

- **Size Reductions** Subtract integer multiples of early vectors from late vectors
- **Swaps** Switch the position of two basis vectors if a minimum amount of G-S length can be pushed.

Cost $\approx$ number of swaps $\times$ cost of size-reduction.

Intuitively, how does it work?

The input is a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

Classical LLL works by making a succession of two elementary moves:

- **Size Reductions** Subtract integer multiples of early vectors from late vectors
- **Swaps** Switch the position of two basis vectors if a minimum amount of G-S length can be pushed.

The moves of the algorithm combine to give a unimodular transformation.

A tight example of LLL

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}$$

A tight example of LLL

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}$$

A tight example of LLL

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}$$

A tight example of LLL

$$
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}
$$

A tight example of LLL

$$
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}
$$

A tight example of LLL

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}$$

A tight example of LLL

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}$$

A tight example of LLL

$$
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}
$$

A tight example of LLL

$$
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 10 & 20 & 5 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 20 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \\ 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \end{pmatrix}
$$

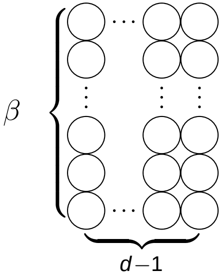
Bounding Switches/Swaps and a visualization of LLL

The height of each column is $\log(\| \mathbf{b}_i^* \|) \leq \beta$.

Every iteration/switch increases a G-S norm by a constant factor.

LLL[82] uses this to bound the number of swaps: $\mathcal{O}(d^2\beta)$.

0 switches



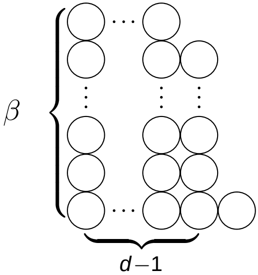
Bounding Switches/Swaps and a visualization of LLL

The height of each column is $\log(\| \mathbf{b}_i^* \|) \leq \beta$.

Every iteration/switch increases a G-S norm by a constant factor.

LLL[82] uses this to bound the number of swaps: $\mathcal{O}(d^2\beta)$.

1 switch



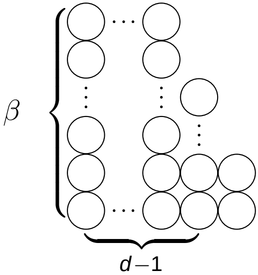
Bounding Switches/Swaps and a visualization of LLL

The height of each column is $\log(\| \mathbf{b}_i^* \|) \leq \beta$.

Every iteration/switch increases a G-S norm by a constant factor.

LLL[82] uses this to bound the number of swaps: $\mathcal{O}(d^2\beta)$.

2 switches



Bounding Switches/Swaps and a visualization of LLL

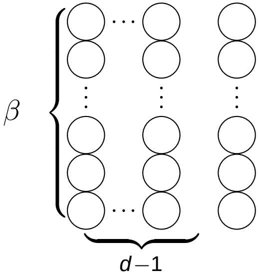
The height of each column is $\log(\| \mathbf{b}_i^* \|) \leq \beta$.

Every iteration/switch increases a G-S norm by a constant factor.

LLL[82] uses this to bound the number of swaps: $\mathcal{O}(d^2\beta)$.

β switches

$= \beta + \cdots$



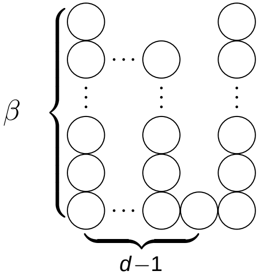
Bounding Switches/Swaps and a visualization of LLL

The height of each column is $\log(\| \mathbf{b}_i^* \|) \leq \beta$.

Every iteration/switch increases a G-S norm by a constant factor.

LLL[82] uses this to bound the number of swaps: $\mathcal{O}(d^2\beta)$.

$\beta + 1$ switches



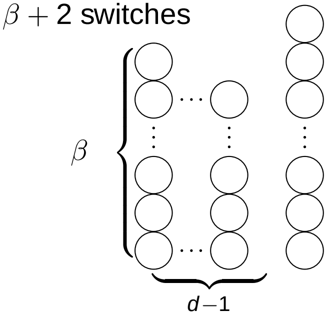
$= \beta + \cdots$

Bounding Switches/Swaps and a visualization of LLL

The height of each column is $\log(\| \mathbf{b}_i^* \|) \leq \beta$.

Every iteration/switch increases a G-S norm by a constant factor.

LLL[82] uses this to bound the number of swaps: $\mathcal{O}(d^2\beta)$.



$= \beta + \cdots$

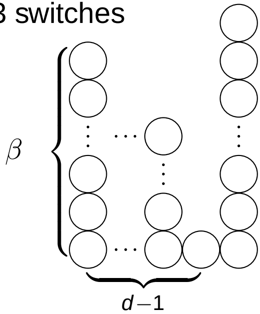
Bounding Switches/Swaps and a visualization of LLL

The height of each column is $\log(\| \mathbf{b}_i^* \|) \leq \beta$.

Every iteration/switch increases a G-S norm by a constant factor.

LLL[82] uses this to bound the number of swaps: $\mathcal{O}(d^2\beta)$.

$\beta + 3$ switches



$= \beta + 2\beta + \cdots$

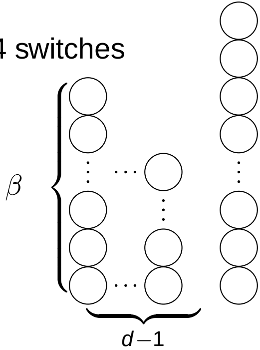
Bounding Switches/Swaps and a visualization of LLL

The height of each column is $\log(\| \mathbf{b}_i^* \|) \leq \beta$.

Every iteration/switch increases a G-S norm by a constant factor.

LLL[82] uses this to bound the number of swaps: $\mathcal{O}(d^2\beta)$.

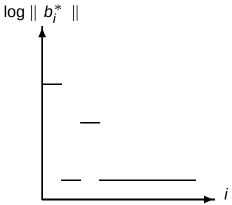
$\beta + 4$ switches



$= \beta + 2\beta +$

$\cdots + (d-1)\beta$

Visual presentation of classic LLL

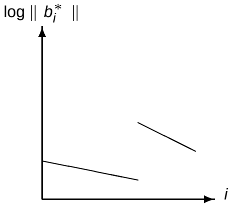


G-S norms of generic input basis

This is a picture showing logs of G-S norms.

A reduced basis would have a minimum possible slope (e.g., -1).

Visual presentation of classic LLL

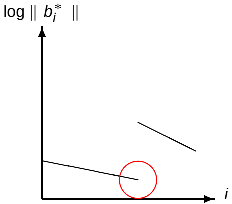


Reduced output $\Rightarrow$
G-S can't drop too fast

This is a picture showing logs of G-S norms.

A reduced basis would have a minimum possible slope (e.g., -1).

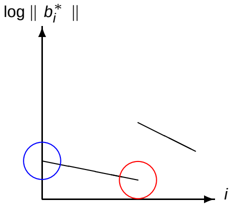
Visual presentation of classic LLL



Reduced output $\Rightarrow$
G-S can't drop too fast

This gives a short vector because:

Visual presentation of classic LLL

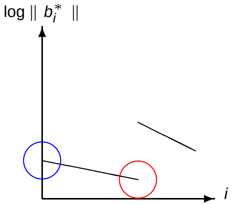


Reduced output $\Rightarrow$
G-S can't drop too fast

This gives a short vector because:

Smallest G-S vector is smaller than every vector in L

Visual presentation of classic LLL



Reduced output $\Rightarrow$
G-S can't drop too fast

This gives a short vector because:

Smallest G-S vector is smaller than every vector in L

G-S vectors aren't generally in L but $b_1^* = b_1$ is in L

Complexity Bounds for reduction algorithms

Given any matrix $B \in \mathbb{Z}_{d \times d}$ with $\| B \|_\infty \leq 2^\beta$ whose columns give the lattice basis.

Find BU whose columns are a reduced basis of the same lattice.

- L^3 costs $\mathcal{P}oly(d) \cdot \beta^3$.
- L^2/H-LLL cost $\mathcal{P}oly(d) \cdot \beta^2$.
- $\widetilde{\mathrm{L}}^1$moves this to $\mathcal{P}oly(d) \cdot \beta^{(1+\epsilon)}$

To the new stuff!

Welcome to the second chapter of the talk, the reward for experts.
A road-map of this section:

1. Present LLL as a sequence of lift-reductions:
   from reduced to reduced
2. Introduce recent truncation-friendly version of reduction.
3. Show the new beautiful tools we made for lift-reduction.
4. Give the new complexities!

To the new stuff!

Welcome to the second chapter of the talk, the reward for experts.
A road-map of this section:

1. Present LLL as a sequence of lift-reductions:
   from reduced to reduced
2. Introduce recent truncation-friendly version of reduction.
3. Show the new beautiful tools we made for lift-reduction.
4. Give the new complexities!

To the new stuff!

Welcome to the second chapter of the talk, the reward for experts.
A road-map of this section:

1. Present LLL as a sequence of lift-reductions:
   from reduced to reduced
2. Introduce recent truncation-friendly version of reduction.
3. Show the new beautiful tools we made for lift-reduction.
4. Give the new complexities!

To the new stuff!

Welcome to the second chapter of the talk, the reward for experts.
A road-map of this section:

1. Present LLL as a sequence of lift-reductions:
   from reduced to reduced
2. Introduce recent truncation-friendly version of reduction.
3. Show the new beautiful tools we made for lift-reduction.
4. Give the new complexities!

Find reduced, deform, reduce again

Old thinking:

1. Input matrix B, not reduced
2. Begin working on vectors of B
3. Until BU reduced

New thinking:

1. Begin with reduced B
2. Deform it: $\sigma_\ell B$
3. Reduce the deformation: $\sigma_\ell BU$ reduced

Find reduced, deform, reduce again

Old thinking:

1. Input matrix B, not reduced
2. Begin working on vectors of B
3. Until BU reduced

New thinking:

1. Begin with reduced B
2. Deform it: $\sigma_\ell B$
3. Reduce the deformation: $\sigma_\ell BU$ reduced

Lift-Reduction

- We call multiplying an entry of each vector by a power of 2 **a lift**.
- As a matrix that is: $\sigma_\ell = \begin{bmatrix} 2^\ell & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$
- We'll analyze the impact of this deformation on reduced bases.
- We call **Lift-Reduction** the act of reducing $\sigma_\ell B$ when B was already reduced.

Lift-Reduction

- We call multiplying an entry of each vector by a power of 2 **a lift**.
- As a matrix that is: $\sigma_\ell = \begin{bmatrix} 2^\ell & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$
- We'll analyze the impact of this deformation on reduced bases.
- We call **Lift-Reduction** the act of reducing $\sigma_\ell B$ when B was already reduced.

Lift-Reduction

- We call multiplying an entry of each vector by a power of 2 **a lift**.
- As a matrix that is: $\sigma_\ell = \begin{bmatrix} 2^\ell & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$
- We'll analyze the impact of this deformation on reduced bases.
- We call **Lift-Reduction** the act of reducing $\sigma_\ell B$ when B was already reduced.

Lift-Reduction

- We call multiplying an entry of each vector by a power of 2 **a lift**.
- As a matrix that is: $\sigma_\ell = \begin{bmatrix} 2^\ell & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$
- We'll analyze the impact of this deformation on reduced bases.
- We call **Lift-Reduction** the act of reducing $\sigma_\ell B$ when B was already reduced.

An example: Triangular

Not Reduced

$$\begin{bmatrix} 123456 & 60123 & -54127 & 23177 \\ 0 & 54321 & 21792 & -15211 \\ 0 & 0 & 321 & 123 \\ 0 & 0 & 0 & 51234 \end{bmatrix}$$

An example: Triangular

Not Reduced

$$\begin{bmatrix} 123456 & 60123 & -54127 & 23177 \\ 0 & 54321 & 21792 & -15211 \\ 0 & 0 & 321 & 123 \\ 0 & 0 & 0 & 51234 \end{bmatrix}$$

Reduced

$$\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & 51234 \end{bmatrix}$$

An example: Triangular

Not Reduced

$$\begin{bmatrix} 123456 & 60123 & -54127 & 23177 \\ 0 & 54321 & 21792 & -15211 \\ 0 & 0 & 321 & 123 \\ 0 & 0 & 0 & 51234 \end{bmatrix}$$

Reduced

$$\begin{bmatrix} & & & \\ & & & \\ & & .321 & .123 \\ & & 0 & 51234 \end{bmatrix}$$

An example: Triangular

Not Reduced

$$\begin{bmatrix} 123456 & 60123 & -54127 & 23177 \\ 0 & 54321 & 21792 & -15211 \\ 0 & 0 & 321 & 123 \\ 0 & 0 & 0 & 51234 \end{bmatrix}$$

Reduced

$$\begin{bmatrix} & & & \\ & & & \\ & & 321 & 123 \\ & & 0 & 51234 \end{bmatrix}$$

An example: Triangular

Not Reduced

$$\begin{bmatrix} 123456 & 60123 & -54127 & 23177 \\ 0 & 54321 & 21792 & -15211 \\ 0 & 0 & 321 & 123 \\ 0 & 0 & 0 & 51234 \end{bmatrix}$$

Reduced

$$\begin{bmatrix} & & \\ .54321 & .21792 & -.15211 \\ 0 & 321 & 123 \\ 0 & 0 & 51234 \end{bmatrix}$$

An example: Triangular

Not Reduced

$$\begin{bmatrix} 123456 & 60123 & -54127 & 23177 \\ 0 & 54321 & 21792 & -15211 \\ 0 & 0 & 321 & 123 \\ 0 & 0 & 0 & 51234 \end{bmatrix}$$

Reduced

$$\begin{bmatrix} & & \\ 318 & 10419 & -4156 \\ 1560 & -2184 & 1059 \\ 0 & 0 & 51234 \end{bmatrix}$$

An example: Triangular

Not Reduced

$$\begin{bmatrix} 123456 & 60123 & -54127 & 23177 \\ 0 & 54321 & 21792 & -15211 \\ 0 & 0 & 321 & 123 \\ 0 & 0 & 0 & 51234 \end{bmatrix}$$

Reduced

$$\begin{bmatrix} .123456 & .060123 & -.054127 & .023177 \\ 0 & 318 & 10419 & -4156 \\ 0 & 1560 & -2184 & 1059 \\ 0 & 0 & 0 & 51234 \end{bmatrix}$$

So what?

Now each lift reduction can be attacked aggressively.

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}^T \quad \text{(24 swaps)}$$

$$\begin{pmatrix} 0 & 0 & 0 & 200 \\ 1 & 0 & 0 & 90 \\ 0 & 1 & 0 & 90 \\ 0 & 0 & 1 & 90 \end{pmatrix}^T \begin{pmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 3 & 3 & 3 & 10 \\ -6 & -7 & -7 & 0 \end{pmatrix}^T \quad \text{(7 swaps)}$$

$$\begin{pmatrix} -1 & 1 & 0 & 301 \\ -1 & 0 & 1 & 802 \end{pmatrix}^T \begin{pmatrix} 5 & -8 & 3 & -2 \\ -8 & 13 & -5 & -97 \end{pmatrix}^T \quad \text{(2 swaps)}$$

(First block only)

So what?

Now each lift reduction can be attacked aggressively.

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}^T \quad \text{(24 swaps)}$$

$$\begin{pmatrix} 0 & 0 & 0 & 200 \\ 1 & 0 & 0 & 90 \\ 0 & 1 & 0 & 90 \\ 0 & 0 & 1 & 90 \end{pmatrix}^T \begin{pmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 3 & 3 & 3 & 10 \\ -6 & -7 & -7 & 0 \end{pmatrix}^T \quad \text{(7 swaps)}$$

$$\begin{pmatrix} -1 & 1 & 0 & 301 \\ -1 & 0 & 1 & 802 \end{pmatrix}^T \begin{pmatrix} 5 & -8 & 3 & -2 \\ -8 & 13 & -5 & -97 \end{pmatrix}^T \quad \text{(2 swaps)}$$

(First block only)

So what?

Now each lift reduction can be attacked aggressively.

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}^T \quad \text{(24 swaps)}$$

$$\begin{pmatrix} 0 & 0 & 0 & 200 \\ 1 & 0 & 0 & 90 \\ 0 & 1 & 0 & 90 \\ 0 & 0 & 1 & 90 \end{pmatrix}^T \begin{pmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 3 & 3 & 3 & 10 \\ -6 & -7 & -7 & 0 \end{pmatrix}^T \quad \text{(7 swaps)}$$

$$\begin{pmatrix} -1 & 1 & 0 & 301 \\ -1 & 0 & 1 & 802 \end{pmatrix}^T \begin{pmatrix} 5 & -8 & 3 & -2 \\ -8 & 13 & -5 & -97 \end{pmatrix}^T \quad \text{(2 swaps)}$$

(First block only)

So what?

Now each lift reduction can be attacked aggressively.

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}^T \quad \text{(24 swaps)}$$

$$\begin{pmatrix} 0 & 0 & 0 & 200 \\ 1 & 0 & 0 & 90 \\ 0 & 1 & 0 & 90 \\ 0 & 0 & 1 & 90 \end{pmatrix}^T \begin{pmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 3 & 3 & 3 & 10 \\ -6 & -7 & -7 & 0 \end{pmatrix}^T \quad \text{(7 swaps)}$$

$$\begin{pmatrix} -1 & 1 & 0 & 301 \\ -1 & 0 & 1 & 802 \end{pmatrix}^T \begin{pmatrix} 5 & -8 & 3 & -2 \\ -8 & 13 & -5 & -97 \end{pmatrix}^T \quad \text{(2 swaps)}$$

(First block only)

So what?

Now each lift reduction can be attacked aggressively.

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}^T \quad \text{(24 swaps)}$$

$$\begin{pmatrix} 0 & 0 & 0 & 200 \\ 1 & 0 & 0 & 90 \\ 0 & 1 & 0 & 90 \\ 0 & 0 & 1 & 90 \end{pmatrix}^T \begin{pmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 3 & 3 & 3 & 10 \\ -6 & -7 & -7 & 0 \end{pmatrix}^T \quad \text{(7 swaps)}$$

$$\begin{pmatrix} -1 & 1 & 0 & 301 \\ -1 & 0 & 1 & 802 \end{pmatrix}^T \begin{pmatrix} 5 & -8 & 3 & -2 \\ -8 & 13 & -5 & -97 \end{pmatrix}^T \quad \text{(2 swaps)}$$

(First block only)

General reduction as a sequence of lift-reduction

Any non-singular B can be triangularized via HNF.

Any triangular B can be reduced with a series of lift-reductions.

General reduction as a sequence of lift-reduction

Any non-singular B can be triangularized via HNF.

Any triangular B can be reduced with a series of lift-reductions.

General reduction as a sequence of lift-reduction

Any non-singular B can be triangularized via HNF.

Any triangular B can be reduced with a series of lift-reductions.

$$\left[\begin{array}{cccc|c} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & 1 & \\ \hline & & & & 1 \end{array}\right] \left[\begin{array}{cccc|c} b_{d,d} & \cdots & \# & \# & \# \\ & \ddots & & & \\ & & b_{3,3} & \# & \# \\ & & & b_{2,2} & \# \\ \hline & & & & b_{1,1} \end{array}\right] \left[\begin{array}{c|c} I & \\ \hline & I \end{array}\right]$$

General reduction as a sequence of lift-reduction

Any non-singular B can be triangularized via HNF.

Any triangular B can be reduced with a series of lift-reductions.

$$
\left[\begin{array}{ccc|cc}
1 & & & & \\
 & \ddots & & & \\
 & & 1 & & \\
\hline
 & & & 2^{\ell_2} & \\
 & & & & 1
\end{array}\right]
\left[\begin{array}{ccc|cc}
b_{d,d} & \cdots & \# & \# & \# \\
 & \ddots & & & \\
 & & b_{3,3} & \# & \# \\
\hline
 & & & \leq 1 & \# \\
 & & & & b_{1,1}
\end{array}\right]
\left[\begin{array}{c|c}
I & \\
\hline
 & U'
\end{array}\right]
$$

General reduction as a sequence of lift-reduction

Any non-singular B can be triangularized via HNF.

Any triangular B can be reduced with a series of lift-reductions.

$$
\left[\begin{array}{ccc|cc}
1 & & & & \\
 & \ddots & & & \\
 & & 1 & & \\
\hline
 & & & 1 & \\
 & & & & 1
\end{array}\right]
\left[\begin{array}{ccc|cc}
b_{d,d} & \cdots & \# & \# & \# \\
 & \ddots & & & \\
 & & b_{3,3} & \# & \# \\
\hline
 & & & \multicolumn{2}{c}{\sigma_{\ell_1} B'U'}
\end{array}\right]
\left[\begin{array}{c|c}
I & \\
\hline
 & I
\end{array}\right]
$$

General reduction as a sequence of lift-reduction

Any non-singular B can be triangularized via HNF.

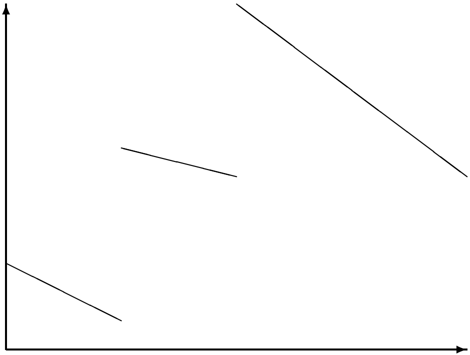
Any triangular B can be reduced with a series of lift-reductions.

$$
\left[\begin{array}{cc|ccc}
1 & & & & \\
 & \ddots & & & \\
\hline
 & & 2^{\ell_3} & & \\
 & & & 1 & \\
 & & & & 1
\end{array}\right]
\left[\begin{array}{cc|ccc}
b_{d,d} & \cdots & \# & \# & \# \\
 & \ddots & & & \\
\hline
 & & \leq 1 & \# & \# \\
 & & & \sigma_{\ell_1} B' U' &
\end{array}\right]
\left[\begin{array}{c|c}
I & \\
\hline
 & U''
\end{array}\right]
$$

Lift-reduction: $B \to \sigma_\ell B \to \sigma_\ell BU$

Graphical view of lift-reduction

$$\log R_{i,i} = \log \| b_i^* \|$$

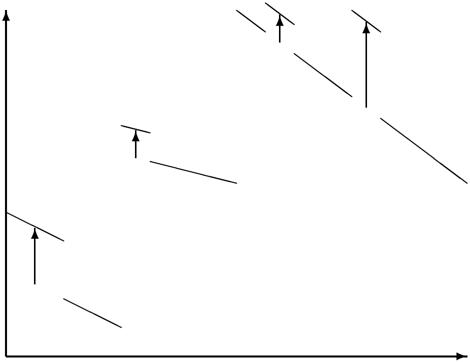


$B = QR$

Lift-reduction: $B \to \sigma_\ell B \to \sigma_\ell BU$

Graphical view of lift-reduction

$\log R'_{i,i}$

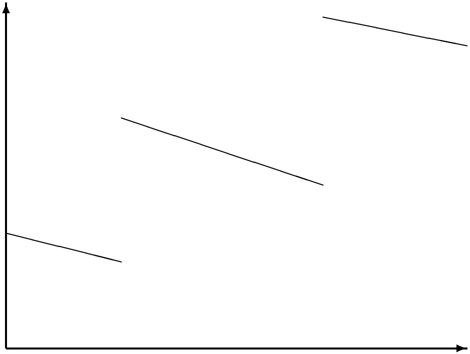


$\sigma_\ell B = Q'R'$

Lift-reduction: $B \to \sigma_\ell B \to \sigma_\ell BU$

Graphical view of lift-reduction

$\log R''_{i,i}$



$\sigma_\ell BU = Q''R''$

Truncations and a weakening of reduction

- We must work with truncated entries.
- Truncations hurt LLL-reduction (small roundings send a reduced basis to an unreduced basis).
- A new sense of reduction is truncation friendly but with all of the perks, thanks to [Chang, Stehlé, Villard]
- I'll denote a truncation of M by $M + \Delta M$
- So now, B 'reduced' $\Rightarrow B + \Delta B$ reduced.

Truncations and a weakening of reduction

- We must work with truncated entries.
- Truncations hurt LLL-reduction (small roundings send a reduced basis to an unreduced basis).
- A new sense of reduction is truncation friendly but with all of the perks, thanks to [Chang, Stehlé, Villard]
- I'll denote a truncation of M by $M + \Delta M$
- So now, B 'reduced' $\Rightarrow B + \Delta B$ reduced.

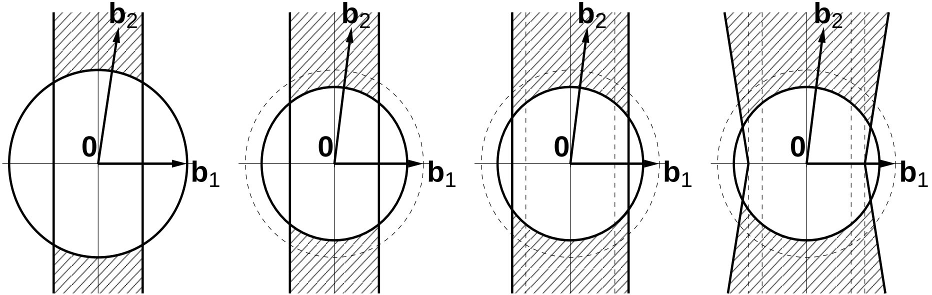
Truncations and a weakening of reduction

- We must work with truncated entries.
- Truncations hurt LLL-reduction (small roundings send a reduced basis to an unreduced basis).
- A new sense of reduction is truncation friendly
  but with all of the perks, thanks to [Chang, Stehlé, Villard]
- I'll denote a truncation of M by $M + \Delta M$
- So now, B 'reduced' $\Rightarrow B + \Delta B$ reduced.

Truncations and a weakening of reduction

- We must work with truncated entries.
- Truncations hurt LLL-reduction (small roundings send a reduced basis to an unreduced basis).
- A new sense of reduction is truncation friendly
  but with all of the perks, thanks to [Chang, Stehlé, Villard]
- I'll denote a truncation of M by $M + \Delta M$
- So now, B 'reduced' $\Rightarrow B + \Delta B$ reduced.

The new reduction, graphically



$(1, 1/2, 0)$	$(\delta, 1/2, 0)$	$(\delta, \eta, 0)$	(δ, η, θ)
Hermite	LLL’82	Schnorr’88	[C-S-V’10]

Benefits of lift reduction

Now I'll show you the (super-cool) tools we introduce for analyzing lift-reductions.

Note that these tools are more general than $\widetilde{\mathrm{L}}^1$.

So remember, use lift-reduction whenever you analyze LLL.

Overview of benefits of lift reduction

Whenever you can find a way to use ‘lift reduction’ you get all of these tools.

For B reduced, $\sigma_\ell := \mathrm{diag}(2^\ell, 1, \ldots, 1)$, and U any matrix such that

$$\sigma_\ell BU \text{ is reduced.}$$

- $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{\|b_j^*\|}{\|b_i^*\|}$
- $\sigma_\ell(B + \Delta B)U$ is reduced.
- $\sigma_\ell B(U + \Delta U)$ is reduced.
- $U + \Delta U$ is unimodular if U was.
- U can be adjusted and stored on $\ell + c \cdot d$-bits per entry
- $\mathrm{cond}(\sigma_\ell B) \leq 2^{\ell+\epsilon}\mathrm{cond}(B)$

Overview of benefits of lift reduction

Whenever you can find a way to use 'lift reduction' you get all of these tools.

For B reduced, $\sigma_\ell := \mathrm{diag}(2^\ell, 1, \ldots, 1)$, and U any matrix such that

$$\sigma_\ell B U \text{ is reduced.}$$

- $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{\|b_j^*\|}{\|b_i^*\|}$
- $\sigma_\ell (B + \Delta B) U$ is reduced.
- $\sigma_\ell B (U + \Delta U)$ is reduced.
- $U + \Delta U$ is unimodular if U was.
- U can be adjusted and stored on $\ell + c \cdot d$-bits per entry
- $\mathrm{cond}(\sigma_\ell B) \leq 2^{\ell + \epsilon} \mathrm{cond}(B)$

Overview of benefits of lift reduction

Whenever you can find a way to use ‘lift reduction’ you get all of these tools.

For B reduced, $\sigma_\ell := \mathrm{diag}(2^\ell, 1, \ldots, 1)$, and U any matrix such that

$$\sigma_\ell B U \text{ is reduced.}$$

- $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{\|b_j^*\|}{\|b_i^*\|}$
- $\sigma_\ell (B + \Delta B) U$ is reduced.
- $\sigma_\ell B (U + \Delta U)$ is reduced.
- $U + \Delta U$ is unimodular if U was.
- U can be adjusted and stored on $\ell + c \cdot d$-bits per entry
- $\mathrm{cond}(\sigma_\ell B) \leq 2^{\ell + \epsilon} \mathrm{cond}(B)$

Overview of benefits of lift reduction

Whenever you can find a way to use 'lift reduction' you get all of these tools.

For B reduced, $\sigma_\ell := \mathrm{diag}(2^\ell, 1, \ldots, 1)$, and U any matrix such that

$$\sigma_\ell B U \text{ is reduced.}$$

- $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{\|b_j^*\|}{\|b_i^*\|}$
- $\sigma_\ell (B + \Delta B) U$ is reduced.
- $\sigma_\ell B (U + \Delta U)$ is reduced.
- $U + \Delta U$ is unimodular if U was.
- U can be adjusted and stored on $\ell + c \cdot d$-bits per entry
- $\mathrm{cond}(\sigma_\ell B) \leq 2^{\ell + \epsilon} \mathrm{cond}(B)$

Overview of benefits of lift reduction

Whenever you can find a way to use 'lift reduction' you get all of these tools.

For B reduced, $\sigma_\ell := \mathrm{diag}(2^\ell, 1, \ldots, 1)$, and U any matrix such that

$$\sigma_\ell B U \text{ is reduced.}$$

- $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{\|b_j^*\|}{\|b_i^*\|}$
- $\sigma_\ell (B + \Delta B) U$ is reduced.
- $\sigma_\ell B (U + \Delta U)$ is reduced.
- $U + \Delta U$ is unimodular if U was.
- U can be adjusted and stored on $\ell + c \cdot d$-bits per entry
- $\mathrm{cond}(\sigma_\ell B) \leq 2^{\ell + \epsilon} \mathrm{cond}(B)$

Overview of benefits of lift reduction

Whenever you can find a way to use ‘lift reduction’ you get all of these tools.

For B reduced, $\sigma_\ell := \mathrm{diag}(2^\ell, 1, \ldots, 1)$, and U any matrix such that

$$\sigma_\ell B U \text{ is reduced.}$$

- $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{\|b_j^*\|}{\|b_i^*\|}$
- $\sigma_\ell (B + \Delta B) U$ is reduced.
- $\sigma_\ell B (U + \Delta U)$ is reduced.
- $U + \Delta U$ is unimodular if U was.
- U can be adjusted and stored on $\ell + c \cdot d$-bits per entry
- $\mathrm{cond}(\sigma_\ell B) \leq 2^{\ell + \epsilon} \mathrm{cond}(B)$

Overview of benefits of lift reduction

Whenever you can find a way to use 'lift reduction' you get all of these tools.

For B reduced, $\sigma_\ell := \mathrm{diag}(2^\ell, 1, \ldots, 1)$, and U any matrix such that

$$\sigma_\ell B U \text{ is reduced.}$$

- $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{\|b_j^*\|}{\|b_i^*\|}$
- $\sigma_\ell(B + \Delta B)U$ is reduced.
- $\sigma_\ell B(U + \Delta U)$ is reduced.
- $U + \Delta U$ is unimodular if U was.
- U can be adjusted and stored on $\ell + c \cdot d$-bits per entry
- $\mathrm{cond}(\sigma_\ell B) \leq 2^{\ell+\epsilon}\mathrm{cond}(B)$

Overview of benefits of lift reduction

Whenever you can find a way to use 'lift reduction' you get all of these tools.

For B reduced, $\sigma_\ell := \mathrm{diag}(2^\ell, 1, \ldots, 1)$, and U any matrix such that

$$\sigma_\ell B U \text{ is reduced.}$$

- $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{\|b_j^*\|}{\|b_i^*\|}$
- $\sigma_\ell (B + \Delta B) U$ is reduced.
- $\sigma_\ell B (U + \Delta U)$ is reduced.
- $U + \Delta U$ is unimodular if U was.
- U can be adjusted and stored on $\ell + c \cdot d$-bits per entry
- $\mathrm{cond}(\sigma_\ell B) \leq 2^{\ell + \epsilon} \mathrm{cond}(B)$

Bounding lift-reduction U-transformations

For $\sigma_\ell BU$ with $B = QR$ we prove: $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{R_{j,j}}{R_{i,i}}$

Blocks in B:

Block diagonal U:

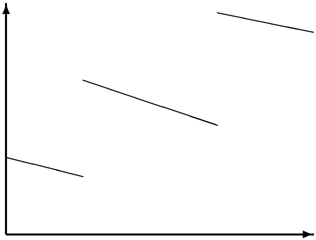
$$U = \begin{bmatrix} U_1 & U_2 & U_3 \\ & U_4 & U_5 \\ & & U_6 \end{bmatrix}$$

U_1, U_4, U_6 small
U_2, U_5 medium
U_3 large

Bounding lift-reduction U-transformations

For $\sigma_\ell BU$ with $B = QR$ we prove: $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{R_{j,j}}{R_{i,i}}$

Blocks in B:



Block diagonal U:

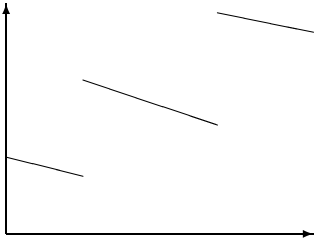
$$U = \begin{bmatrix} U_1 & U_2 & U_3 \\ & U_4 & U_5 \\ & & U_6 \end{bmatrix}$$

U_1, U_4, U_6 small
U_2, U_5 medium
U_3 large

Bounding lift-reduction U-transformations

For $\sigma_\ell BU$ with $B = QR$ we prove: $|U_{i,j}| \leq 2^{\ell + c \cdot d} \frac{R_{j,j}}{R_{i,i}}$

Blocks in B:



Block diagonal U:

$$U = \begin{bmatrix} U_1 & U_2 & U_3 \\ & U_4 & U_5 \\ & & U_6 \end{bmatrix}$$

U_1, U_4, U_6 small
U_2, U_5 medium
U_3 large

Allows truncations of U

Let B and $\sigma_\ell BU$ be reduced.

For any ΔU with $\Delta U_{i,j}/U_{i,j} \leq \epsilon$ (entry-wise perturbations)

We show:

$$\sigma_\ell B(U + \Delta U) \quad \text{is} \quad \text{also} \quad \text{reduced}$$

and:

$$(U + \Delta U) \quad \text{is} \quad \text{unimodular}$$

Allows truncations of U

Let B and $\sigma_\ell BU$ be reduced.

For any ΔU with $\Delta U_{i,j}/U_{i,j} \leq \epsilon$ (entry-wise perturbations)

We show:

$$\sigma_\ell B(U + \Delta U) \quad \text{is} \quad \text{also} \quad \text{reduced}$$

and:

$$(U + \Delta U) \quad \text{is} \quad \text{unimodular}$$

Allows truncations of U

Let B and $\sigma_\ell BU$ be reduced.

For any ΔU with $\Delta U_{i,j}/U_{i,j} \leq \epsilon$ (entry-wise perturbations)

We show:

$$\sigma_\ell B(U + \Delta U) \quad \text{is} \quad \text{also} \quad \text{reduced}$$

and:

$$(U + \Delta U) \quad \text{is} \quad \text{unimodular}$$

Allows truncations of U

Let B and $\sigma_\ell BU$ be reduced.

For any ΔU with $\Delta U_{i,j}/U_{i,j} \leq \epsilon$ (entry-wise perturbations)

We show:

$$\sigma_\ell B(U + \Delta U) \quad \text{is also reduced}$$

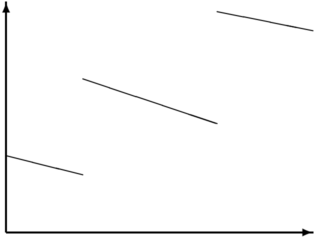
and:

$$(U + \Delta U) \quad \text{is unimodular}$$

Can create efficient U-transformations

$U + \Delta U$ will reduce so we can make an efficient U.

Visual blocks:



Block diagonal U:

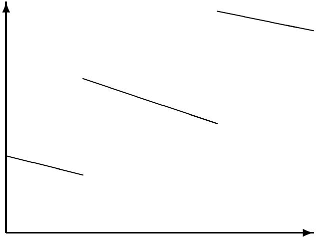
$$U = \begin{bmatrix} U_1 & U_2 & U_3 \\ & U_4 & U_5 \\ & & U_6 \end{bmatrix}$$

U_1, U_4, U_6 small
U_2, U_5 medium
U_3 large

Can create efficient U-transformations

$U + \Delta U$ will reduce so we can make an efficient U.

Visual blocks:



$U + \Delta U$:

$$\begin{bmatrix} \hat{U}_1 & \hat{U}_2 \cdot 2^{k_2} & \hat{U}_3 \cdot 2^{k_3} \\ & \hat{U}_4 & \hat{U}_5 \cdot 2^{k_2} \\ & & \hat{U}_6 \end{bmatrix}$$

$\hat{U}_i$ small

Allows adjustments of B

- By mastering U we can also master B.
- When B and $\sigma_\ell BU$ are reduced
- Then for ΔB with $\Delta B_j / B_j \leq \epsilon$ (column-wise perturbations)
- We show:

$$\sigma_\ell (B + \Delta B) U \quad \text{is} \quad \text{reduced}$$

Allows adjustments of B

- By mastering U we can also master B.
- When B and $\sigma_\ell BU$ are reduced
- Then for ΔB with $\Delta B_j / B_j \le \epsilon$ (column-wise perturbations)
- We show:

$$\sigma_\ell(B + \Delta B)U \quad \text{is} \quad \text{reduced}$$

Allows adjustments of B

- By mastering U we can also master B.
- When B and $\sigma_\ell BU$ are reduced
- Then for ΔB with $\Delta B_j / B_j \leq \epsilon$ (column-wise perturbations)
- We show:

$$\sigma_\ell (B + \Delta B) U \quad \text{is} \quad \text{reduced}$$

Allows adjustments of B

- By mastering U we can also master B.
- When B and $\sigma_\ell BU$ are reduced
- Then for ΔB with $\Delta B_j / B_j \leq \epsilon$ (column-wise perturbations)
- We show:

$$\sigma_\ell (B + \Delta B) U \quad \text{is} \quad \text{reduced}$$

Numerical Stability

- In `fpLLL` the precision needed is related to the induced Condition number of B.
- For $B = QR$ let $\mathrm{Cond}(B) := \| \, |R| \cdot |R^{-1}| \, \|$.
- The higher $\mathrm{Cond}(B)$ the more precision `fpLLL` needs.
- A reduced B is well-conditioned ($\approx 2^{\mathcal{O}(d)}$).
- We master this when deforming:
  $\mathrm{Cond}(\sigma_\ell B) = 2^{\ell + c \cdot d} \mathrm{Cond}(B)$

Numerical Stability

- In `fpLLL` the precision needed is related to the induced Condition number of B.
- For $B = QR$ let $\mathrm{Cond}(B) := \| \, |R| \cdot |R^{-1}| \, \|$.
- The higher $\mathrm{Cond}(B)$ the more precision `fpLLL` needs.
- A reduced B is well-conditioned ($\approx 2^{\mathcal{O}(d)}$).
- We master this when deforming:
  $\mathrm{Cond}(\sigma_\ell B) = 2^{\ell + c \cdot d}\mathrm{Cond}(B)$

Numerical Stability

- In `fpLLL` the precision needed is related to the induced Condition number of B.
- For $B = QR$ let $\mathrm{Cond}(B) := \| \, |R| \cdot |R^{-1}| \, \|$.
- The higher $\mathrm{Cond}(B)$ the more precision `fpLLL` needs.
- A reduced B is well-conditioned ($\approx 2^{\mathcal{O}(d)}$).
- We master this when deforming: $\mathrm{Cond}(\sigma_\ell B) = 2^{\ell + c \cdot d}\mathrm{Cond}(B)$

Numerical Stability

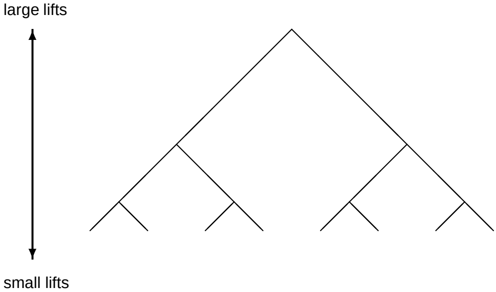
- In fpLLL the precision needed is related to the induced Condition number of B.
- For $B = QR$ let $\mathrm{Cond}(B) := \| \, |R| \cdot |R^{-1}| \, \|$.
- The higher $\mathrm{Cond}(B)$ the more precision fpLLL needs.
- A reduced B is well-conditioned ($\approx 2^{\mathcal{O}(d)}$).
- We master this when deforming: $\mathrm{Cond}(\sigma_\ell B) = 2^{\ell + c \cdot d}\mathrm{Cond}(B)$

Numerical Stability

- In `fpLLL` the precision needed is related to the induced Condition number of B.
- For $B = QR$ let $\mathrm{Cond}(B) := \| \, |R| \cdot |R^{-1}| \, \|$.
- The higher $\mathrm{Cond}(B)$ the more precision `fpLLL` needs.
- A reduced B is well-conditioned ($\approx 2^{\mathcal{O}(d)}$).
- We master this when deforming:
  $\mathrm{Cond}(\sigma_\ell B) = 2^{\ell + c \cdot d}\mathrm{Cond}(B)$

Put the tools to use

Let's try lift-reducing using recursion.

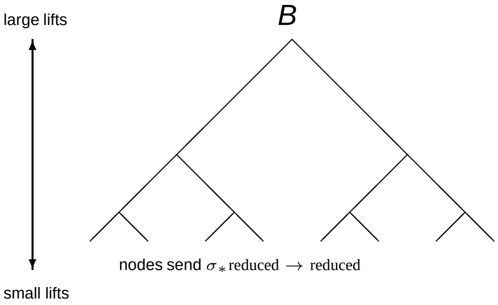


A recursive lifting tree

Put the tools to use

input: B reduced and lifting target ℓ

goal: U such that $\sigma_\ell BU$ is reduced

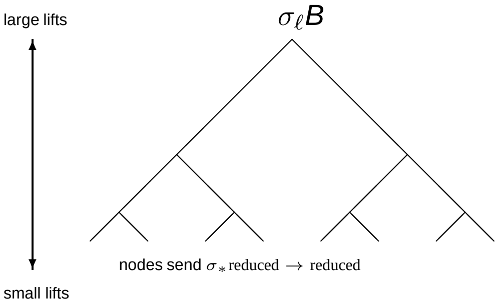


A recursive lifting tree

Put the tools to use

input: B reduced and lifting target ℓ

goal: U such that $\sigma_\ell BU$ is reduced

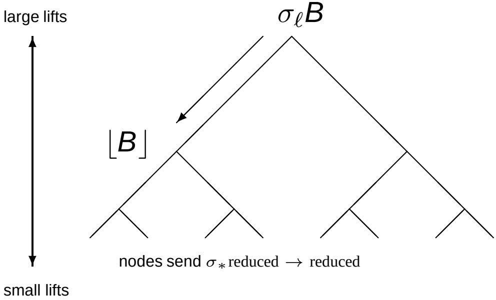


A recursive lifting tree

Put the tools to use

input: B reduced and lifting target ℓ

goal: U such that $\sigma_\ell BU$ is reduced

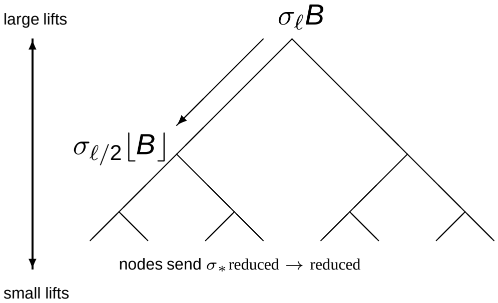


$$B \text{ reduced} \Rightarrow B + \Delta B \text{ reduced}$$

Put the tools to use

input: B reduced and lifting target ℓ

goal: U such that $\sigma_\ell B U$ is reduced

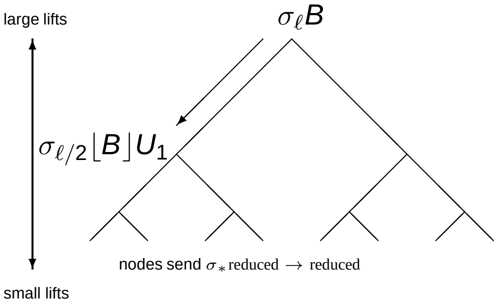


$B + \Delta B$ lifted

Put the tools to use

input: B reduced and lifting target ℓ

goal: U such that $\sigma_\ell B U$ is reduced

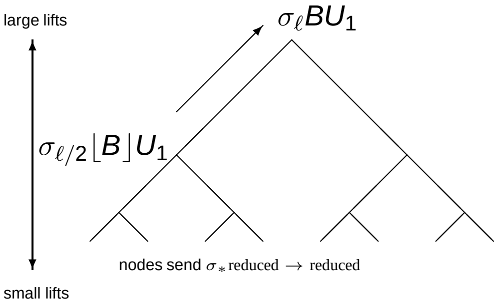


$B + \Delta B$ lift-reduced

Put the tools to use

input: B reduced and lifting target ℓ

goal: U such that $\sigma_\ell BU$ is reduced

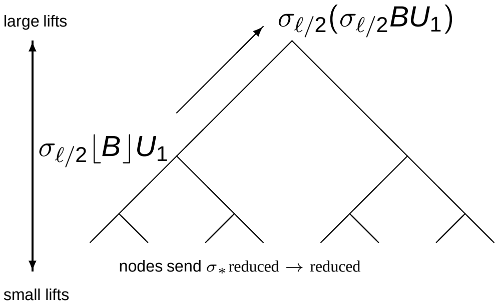


$$\sigma_{\ell/2}(B+\Delta B)U_1 \text{ red.} \Rightarrow \sigma_{\ell/2}BU_1 \text{ red.}$$

Put the tools to use

input: B reduced and lifting target ℓ

goal: U such that $\sigma_\ell B U$ is reduced

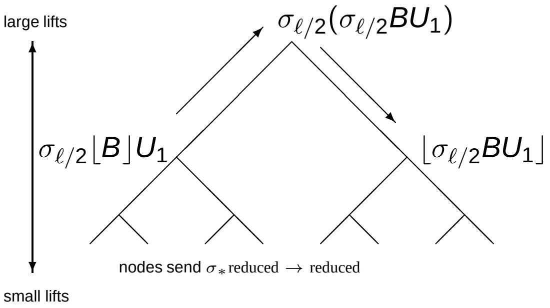


$\sigma_\ell = \sigma_{\ell/2}^2$, now a smaller lift

Put the tools to use

input: B reduced and lifting target ℓ

goal: U such that $\sigma_\ell BU$ is reduced



and so on . . .

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell BU$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} BU_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell B U$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} B U_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell BU$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} BU_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell BU$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} BU_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell BU$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} BU_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell BU$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} BU_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell B U$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} B U_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell BU$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} BU_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:
Problem 1: Are we reduced enough? (Truncations weaken)

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell BU$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} BU_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:
Problem 2: Reduce **leaf** paying ℓ not β

Recursive Lift-reduction

Pseudo-Algorithm: Lift-$\widetilde{\mathrm{L}}^1$
Input: B reduced with $\| B_j \| \leq 2^\beta$ and target lift ℓ
Output: unimodular U with $\sigma_\ell B U$ reduced

1. **leaf:** if $\ell \leq d$ then reduce $\sigma_\ell B$; return U
2. Lift-$\widetilde{\mathrm{L}}^1$ on $(B + \Delta B)$, target $\ell/2$; get U_1
3. Compute $B_1 := \sigma_{\ell/2} B U_1$ weakly reduced
4. Lift-$\widetilde{\mathrm{L}}^1$ on $(B_1 + \Delta B_1)$, target $\ell/2$; return U_2
5. return $U_1 U_2$

Three problems:
Problem 3: Perform matrix multiplications paying ℓ not β

New complexities

In these times B is $d \times d$ and $\| B_j \| \leq 2^{\beta}$.

Lift-reduction: given B Ξ-reduced we find U such that $\sigma_\ell BU$ is Ξ-reduced in time

$$\mathcal{O}\left(d^{3+\epsilon}(d+\ell+\tau) + d^{\omega}\mathcal{M}(\ell)\log \ell + \ell \log(\beta+\ell)\right)$$

Full-reduction: given any B we find U such that BU is Ξ-reduced in time

$$\mathcal{O}(d^{5+\epsilon}\beta + d^{\omega+1+\epsilon}\beta^{1+\epsilon})$$

Knapsack-reduction: for a knapsack-type lattice B we use only time

$$\mathcal{O}(d^{5+\epsilon} + d^{4+\epsilon}\beta + d^{\omega}\beta^{1+\epsilon})$$

Future Directions

Internal to Lattice Reduction:

- Better preconditioning
- Dynamic switch decisions
- Numerically stable steps (maximize practical dimension)
- Parallelize (we all need to)

Future Directions

External to Lattice Reduction:

- Challenge Problems (Homomorphic Crypto Attacks)
- Adaptable to other NP approximations?
- Given a hammer. . .

Thank You

Thank you for your time!

Problem 1: Strengthen quality

- Morel, Stehlé, and Villard have worked on quickly improving the quality of a reduced basis.
- By recognizing blocks of vectors one can carefully truncate the input lattice.

- Results in calling fpLLL on a single lattice with $\beta = O(d)$

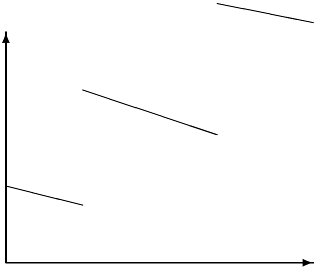
Problem 1: Strengthen quality

- Morel, Stehlé, and Villard have worked on quickly improving the quality of a reduced basis.
- By recognizing blocks of vectors one can carefully truncate the input lattice.

- Results in calling `fpLLL` on a single lattice with $\beta = \mathcal{O}(d)$

Problem 1: Strengthen quality

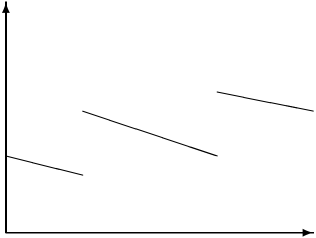
- Morel, Stehlé, and Villard have worked on quickly improving the quality of a reduced basis.
- By recognizing blocks of vectors one can carefully truncate the input lattice.



- Results in calling `fpLLL` on a single lattice with $\beta = \mathcal{O}(d)$

Problem 1: Strengthen quality

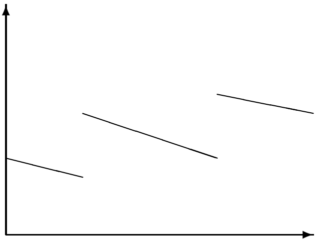
- Morel, Stehlé, and Villard have worked on quickly improving the quality of a reduced basis.
- By recognizing blocks of vectors one can carefully truncate the input lattice.



- Results in calling `fpLLL` on a single lattice with $\beta = \mathcal{O}(d)$

Problem 1: Strengthen quality

- Morel, Stehlé, and Villard have worked on quickly improving the quality of a reduced basis.
- By recognizing blocks of vectors one can carefully truncate the input lattice.



- Results in calling `fpLLL` on a single lattice with $\beta = \mathcal{O}(d)$

Problem 2: Leaf paying ℓ not β

- We have to reduce $\sigma_d B$ without a β in the complexity.
- We adapt the Strengthening algorithm to the lift-reduction case.
- Blocks are deformed by σ_ℓ but remain somewhat preserved.

- Results in single `fpLLL` with $\beta = \mathcal{O}(d + \ell)$

Problem 2: Leaf paying ℓ not β

- We have to reduce $\sigma_d B$ without a β in the complexity.
- We adapt the Strengthening algorithm to the lift-reduction case.
- Blocks are deformed by σ_ℓ but remain somewhat preserved.

- Results in single `fpLLL` with $\beta = \mathcal{O}(d + \ell)$

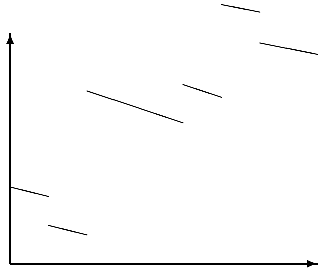
Problem 2: Leaf paying ℓ not β

- We have to reduce $\sigma_d B$ without a β in the complexity.
- We adapt the Strengthening algorithm to the lift-reduction case.
- Blocks are deformed by σ_ℓ but remain somewhat preserved.

- Results in single `fpLLL` with $\beta = \mathcal{O}(d + \ell)$

Problem 2: Leaf paying ℓ not β

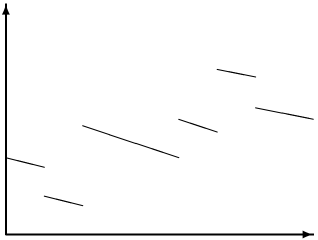
- We have to reduce $\sigma_d B$ without a β in the complexity.
- We adapt the Strengthening algorithm to the lift-reduction case.
- Blocks are deformed by σ_ℓ but remain somewhat preserved.



- Results in single `fpLLL` with $\beta = \mathcal{O}(d + \ell)$

Problem 2: Leaf paying ℓ not β

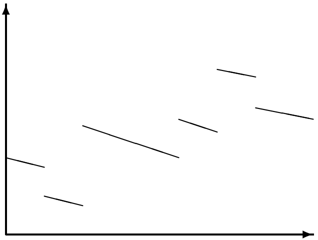
- We have to reduce $\sigma_d B$ without a β in the complexity.
- We adapt the Strengthening algorithm to the lift-reduction case.
- Blocks are deformed by σ_ℓ but remain somewhat preserved.



- Results in single `fpLLL` with $\beta = \mathcal{O}(d + \ell)$

Problem 2: Leaf paying ℓ not β

- We have to reduce $\sigma_d B$ without a β in the complexity.
- We adapt the Strengthening algorithm to the lift-reduction case.
- Blocks are deformed by σ_ℓ but remain somewhat preserved.



- Results in single `fpLLL` with $\beta = \mathcal{O}(d + \ell)$

Problem 3: Matrix products paying ℓ not β

- We have two types of products: $\sigma_\ell BU$ and U_1U_2.
- These are performed in every layer of recursion even when ℓ is small.
- We know we can adjust B, so we begin with $B := \hat{B}E$ where $\hat{B}$ has small entries and $E = \mathrm{diag}(2^{e_1}, \ldots, 2^{e_d})$
- Any U we find can also be adjusted, we choose to take $U = F\hat{U}F^{-1}$ format where $F = \mathrm{diag}(2^{f_1}, \ldots, 2^{f_d})$ and $\hat{U}$ has small entries.
- Now these products can be multiplied quickly (standard matrix multiplication with small entries).
- Any weaknesses introduced from our adjustments can be fixed by strengthening (which returns these formats too).

Problem 3: Matrix products paying ℓ not β

- We have two types of products: $\sigma_\ell BU$ and U_1U_2.
- These are performed in every layer of recursion even when ℓ is small.
- We know we can adjust B, so we begin with $B := \hat{B}E$ where $\hat{B}$ has small entries and $E = \mathrm{diag}(2^{e_1}, \ldots, 2^{e_d})$
- Any U we find can also be adjusted, we choose to take $U = F\hat{U}F^{-1}$ format where $F = \mathrm{diag}(2^{f_1}, \ldots, 2^{f_d})$ and $\hat{U}$ has small entries.
- Now these products can be multiplied quickly (standard matrix multiplication with small entries).
- Any weaknesses introduced from our adjustments can be fixed by strengthening (which returns these formats too).

Problem 3: Matrix products paying ℓ not β

- We have two types of products: $\sigma_\ell BU$ and U_1U_2.
- These are performed in every layer of recursion even when ℓ is small.
- We know we can adjust B, so we begin with $B := \hat{B}E$ where $\hat{B}$ has small entries and $E = \mathrm{diag}(2^{e_1}, \ldots, 2^{e_d})$
- Any U we find can also be adjusted, we choose to take $U = F\hat{U}F^{-1}$ format where $F = \mathrm{diag}(2^{f_1}, \ldots, 2^{f_d})$ and $\hat{U}$ has small entries.
- Now these products can be multiplied quickly (standard matrix multiplication with small entries).
- Any weaknesses introduced from our adjustments can be fixed by strengthening (which returns these formats too).

Problem 3: Matrix products paying ℓ not β

- We have two types of products: $\sigma_\ell BU$ and U_1U_2.
- These are performed in every layer of recursion even when ℓ is small.
- We know we can adjust B, so we begin with $B := \hat{B}E$ where $\hat{B}$ has small entries and $E = \mathrm{diag}(2^{e_1}, \ldots, 2^{e_d})$
- Any U we find can also be adjusted, we choose to take $U = F\hat{U}F^{-1}$ format where $F = \mathrm{diag}(2^{f_1}, \ldots, 2^{f_d})$ and $\hat{U}$ has small entries.
- Now these products can be multiplied quickly (standard matrix multiplication with small entries).
- Any weaknesses introduced from our adjustments can be fixed by strengthening (which returns these formats too).

Problem 3: Matrix products paying ℓ not β

- We have two types of products: $\sigma_\ell B U$ and $U_1 U_2$.
- These are performed in every layer of recursion even when ℓ is small.
- We know we can adjust B, so we begin with $B := \hat{B}E$ where $\hat{B}$ has small entries and $E = \mathrm{diag}(2^{e_1}, \ldots, 2^{e_d})$
- Any U we find can also be adjusted, we choose to take $U = F\hat{U}F^{-1}$ format where $F = \mathrm{diag}(2^{f_1}, \ldots, 2^{f_d})$ and $\hat{U}$ has small entries.
- Now these products can be multiplied quickly (standard matrix multiplication with small entries).
- Any weaknesses introduced from our adjustments can be fixed by strengthening (which returns these formats too).

Problem 3: Matrix products paying ℓ not β

- We have two types of products: $\sigma_\ell BU$ and U_1U_2.
- These are performed in every layer of recursion even when ℓ is small.
- We know we can adjust B, so we begin with $B := \hat{B}E$ where $\hat{B}$ has small entries and $E = \mathrm{diag}(2^{e_1}, \ldots, 2^{e_d})$
- Any U we find can also be adjusted, we choose to take $U = F\hat{U}F^{-1}$ format where $F = \mathrm{diag}(2^{f_1}, \ldots, 2^{f_d})$ and $\hat{U}$ has small entries.
- Now these products can be multiplied quickly (standard matrix multiplication with small entries).
- Any weaknesses introduced from our adjustments can be fixed by strengthening (which returns these formats too).