A quick overview of lattice reduction and LLL

Andy Novocin

ENS de Lyon

September 8th

The Road Map

What LLL does

Why so useful

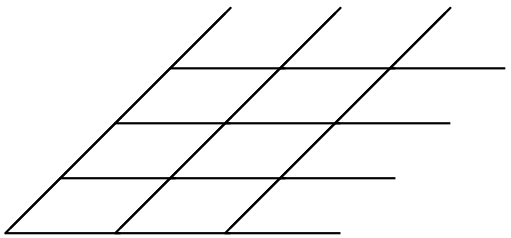
How it works

Behavior and Complexity

Overview of LLL

Introducing Lattices

A lattice, L



The same lattice, L

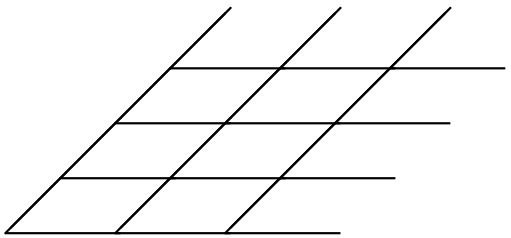
Definition

A lattice, L, is all integer combinations of some vectors.
Any minimal spanning set of L is called a basis of L.

Every lattice has many bases. . . and we want a good basis!

Introducing Lattices

A lattice, L



The same lattice, L

Definition

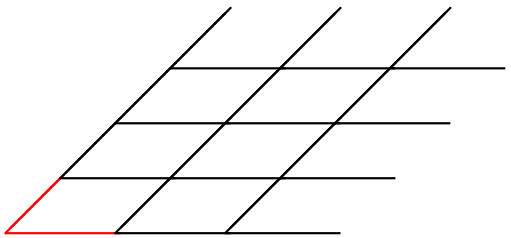
A lattice, L, is all integer combinations of some vectors.

Any minimal spanning set of L is called a basis of L.

Every lattice has many bases. . . and we want a good basis!

Introducing Lattices

A lattice, L



The same lattice, L

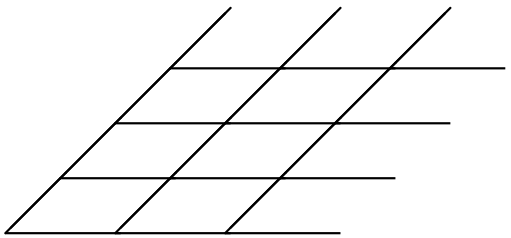
Definition

A lattice, L, is all integer combinations of some vectors.
Any minimal spanning set of L is called a basis of L.

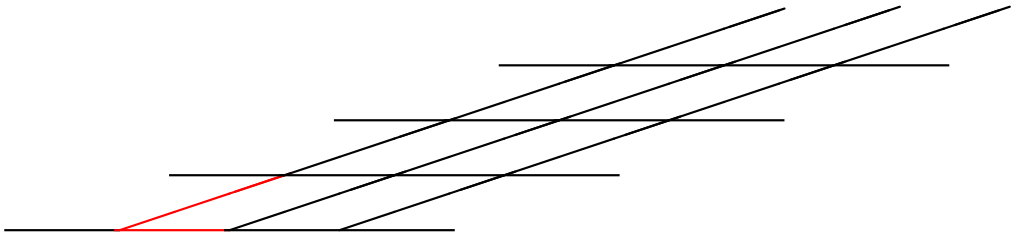
Every lattice has many bases. . . and we want a good basis!

Introducing Lattices

A lattice, L



The same lattice, L



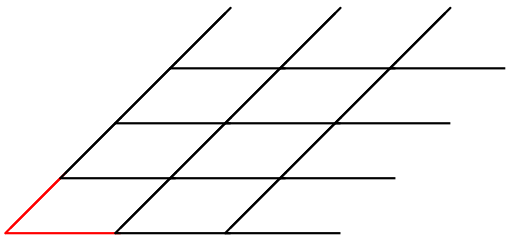
Definition

A lattice, L, is all integer combinations of some vectors.
Any minimal spanning set of L is called a basis of L.

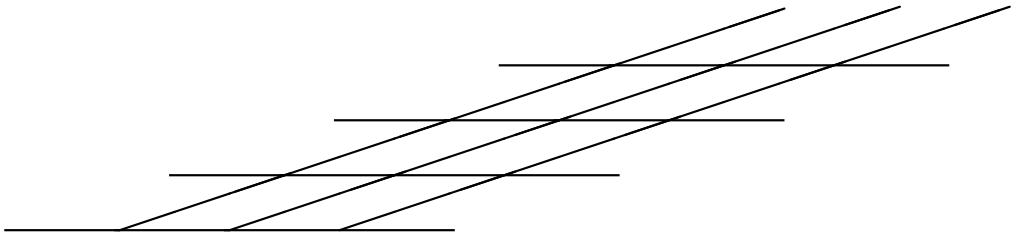
Every lattice has many bases... and we want a good basis!

Introducing Lattices

A lattice, L



The same lattice, L



Definition

A lattice, L, is all integer combinations of some vectors.
Any minimal spanning set of L is called a basis of L.

Every lattice has many bases. . . and we want a good basis!

From one basis to another

A Generic Lattice Reduction Algorithm

Input: A basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of L.
Output: A 'reduced' basis $\mathbf{b}'_1, \ldots, \mathbf{b}'_d$ of L (I'll define reduced in a moment).

- LLL [Lenstra, Lenstra, Lovász 1982] is the name of the first and most popular lattice reduction algorithm.
- Let B (resp. B') have columns $\mathbf{b}_j$ (resp. $\mathbf{b}'_j$). Then $B' = BU$ for some unimodular transformation U.

From one basis to another

A Generic Lattice Reduction Algorithm

Input: A basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of L.
Output: A 'reduced' basis $\mathbf{b}'_1, \ldots, \mathbf{b}'_d$ of L (I'll define reduced in a moment).

- LLL [Lenstra, Lenstra, Lovász 1982] is the name of the first and most popular lattice reduction algorithm.
- Let B (resp. B') have columns $\mathbf{b}_j$ (resp. $\mathbf{b}'_j$). Then $B' = BU$ for some unimodular transformation U.

From one basis to another

A Generic Lattice Reduction Algorithm

Input: A basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of L.
Output: A 'reduced' basis $\mathbf{b}'_1, \ldots, \mathbf{b}'_d$ of L (I'll define reduced in a moment).

- LLL [Lenstra, Lenstra, Lovász 1982] is the name of the first and most popular lattice reduction algorithm.
- Let B (resp. B') have columns $\mathbf{b}_j$ (resp. $\mathbf{b}'_j$). Then $B' = BU$ for some unimodular transformation U.

Quick reminder of Gram-Schmidt

Given a set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ the Gram-Schmidt (G-S) process returns orthogonal vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$. Inductively:

- $\mathbf{b}_1^* := \mathbf{b}_1$
- For i from 2 to d:
  - For j from 1 to $i-1$ let $\mu_{i,j}$ be $\frac{<\mathbf{b}_i, \mathbf{b}_j^*>}{<\mathbf{b}_j^*, \mathbf{b}_j^*>}$ (coefficient of projection of $\mathbf{b}_i$ onto $\mathbf{b}_j^*$)
  - Define $\mathbf{b}_i^* := \mathbf{b}_i - \left(\sum_{j=1}^{i-1} \mu_{i,j} \cdot \mathbf{b}_j^* \right)$.

Intuition of GSO

My favorite way to think of G-S vectors is that $\mathbf{b}_i^*$ is $\mathbf{b}_i$ **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{R}$.

Quick reminder of Gram-Schmidt

Given a set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ the Gram-Schmidt (G-S) process returns orthogonal vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$. Inductively:

- $\mathbf{b}_1^* := \mathbf{b}_1$
- For i from 2 to d:
  - For j from 1 to $i-1$ let $\mu_{i,j}$ be $\frac{<\mathbf{b}_i, \mathbf{b}_j^*>}{<\mathbf{b}_j^*, \mathbf{b}_j^*>}$ (coefficient of projection of $\mathbf{b}_i$ onto $\mathbf{b}_j^*$)
  - Define $\mathbf{b}_i^* := \mathbf{b}_i - \left(\sum_{j=1}^{i-1} \mu_{i,j} \cdot \mathbf{b}_j^* \right)$.

Intuition of GSO

My favorite way to think of G-S vectors is that $\mathbf{b}_i^*$ is $\mathbf{b}_i$ **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{R}$.

Quick reminder of Gram-Schmidt

Given a set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ the Gram-Schmidt (G-S) process returns orthogonal vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$. Inductively:

- $\mathbf{b}_1^* := \mathbf{b}_1$
- For i from 2 to d:
  - For j from 1 to $i-1$ let $\mu_{i,j}$ be $\frac{<\mathbf{b}_i, \mathbf{b}_j^*>}{<\mathbf{b}_j^*, \mathbf{b}_j^*>}$ (coefficient of projection of $\mathbf{b}_i$ onto $\mathbf{b}_j^*$)
  - Define $\mathbf{b}_i^* := \mathbf{b}_i - \left(\sum_{j=1}^{i-1} \mu_{i,j} \cdot \mathbf{b}_j^* \right)$.

Intuition of GSO

My favorite way to think of G-S vectors is that $\mathbf{b}_i^*$ is $\mathbf{b}_i$ **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{R}$.

Quick reminder of Gram-Schmidt

Given a set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ the Gram-Schmidt (G-S) process returns orthogonal vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$. Inductively:

- $\mathbf{b}_1^* := \mathbf{b}_1$
- For i from 2 to d:
  - For j from 1 to $i-1$ let $\mu_{i,j}$ be $\frac{<\mathbf{b}_i, \mathbf{b}_j^*>}{<\mathbf{b}_j^*, \mathbf{b}_j^*>}$ (coefficient of projection of $\mathbf{b}_i$ onto $\mathbf{b}_j^*$)
  - Define $\mathbf{b}_i^* := \mathbf{b}_i - \left(\sum_{j=1}^{i-1} \mu_{i,j} \cdot \mathbf{b}_j^* \right)$.

Intuition of GSO

My favorite way to think of G-S vectors is that $\mathbf{b}_i^*$ is $\mathbf{b}_i$ **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{R}$.

Quick reminder of Gram-Schmidt

Given a set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ the Gram-Schmidt (G-S) process returns orthogonal vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$. Inductively:

- $\mathbf{b}_1^* := \mathbf{b}_1$
- For i from 2 to d:
  - For j from 1 to $i-1$ let $\mu_{i,j}$ be $\frac{<\mathbf{b}_i, \mathbf{b}_j^*>}{<\mathbf{b}_j^*, \mathbf{b}_j^*>}$ (coefficient of projection of $\mathbf{b}_i$ onto $\mathbf{b}_j^*$)
  - Define $\mathbf{b}_i^* := \mathbf{b}_i - \left(\sum_{j=1}^{i-1} \mu_{i,j} \cdot \mathbf{b}_j^* \right)$.

Intuition of GSO

My favorite way to think of G-S vectors is that $\mathbf{b}_i^*$ is $\mathbf{b}_i$ **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{R}$.

LLL Reduced Bases

Size-reduced basis

A lattice basis is said to be size-reduced if $|\mu_{i,j}| \leq \frac{1}{2}$ for all $j < i$.

Roughly $\mathbf{b}_i$ is **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{Z}$ (similar to G-S but remaining in the lattice).

LLL-reduced basis

A lattice basis is said to be LLL-reduced if it is **size-reduced** and $\| \mathbf{b}^*_{i+1} \|^2 \geq (\frac{3}{4} - \mu_{i+1,i}) \| \mathbf{b}^*_i \|^2 \geq \frac{1}{2} \| \mathbf{b}^*_i \|^2$ for $1 \leq i < d$

Intuitively this means that the G-S norms don't drop 'too fast'.

LLL Reduced Bases

Size-reduced basis

A lattice basis is said to be size-reduced if $|\mu_{i,j}| \leq \frac{1}{2}$ for all $j < i$.

Roughly $\mathbf{b}_i$ is **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{Z}$ (similar to G-S but remaining in the lattice).

LLL-reduced basis

A lattice basis is said to be LLL-reduced if it is **size-reduced** and $\| \mathbf{b}^*_{i+1} \|^2 \geq (\frac{3}{4} - \mu_{i+1,i}) \| \mathbf{b}^*_i \|^2 \geq \frac{1}{2} \| \mathbf{b}^*_i \|^2$ for $1 \leq i < d$

Intuitively this means that the G-S norms don't drop 'too fast'.

LLL Reduced Bases

Size-reduced basis

A lattice basis is said to be size-reduced if $|\mu_{i,j}| \leq \frac{1}{2}$ for all $j < i$.

Roughly $\mathbf{b}_i$ is **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{Z}$ (similar to G-S but remaining in the lattice).

LLL-reduced basis

A lattice basis is said to be LLL-reduced if it is **size-reduced** and $\| \mathbf{b}^*_{i+1} \|^2 \geq (\frac{3}{4} - \mu_{i+1,i}) \| \mathbf{b}^*_i \|^2 \geq \frac{1}{2} \| \mathbf{b}^*_i \|^2$ for $1 \leq i < d$

Intuitively this means that the G-S norms don't drop 'too fast'.

LLL Reduced Bases

Size-reduced basis

A lattice basis is said to be size-reduced if $|\mu_{i,j}| \leq \frac{1}{2}$ for all $j < i$.

Roughly $\mathbf{b}_i$ is **modded out** by $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ over $\mathbb{Z}$ (similar to G-S but remaining in the lattice).

LLL-reduced basis

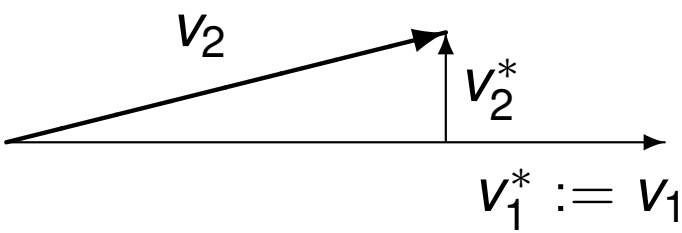
A lattice basis is said to be LLL-reduced if it is **size-reduced** and $\| \mathbf{b}^*_{i+1} \|^2 \geq (\frac{3}{4} - \mu_{i+1,i}) \| \mathbf{b}^*_i \|^2 \geq \frac{1}{2} \| \mathbf{b}^*_i \|^2$ for $1 \leq i < d$

Intuitively this means that the G-S norms don't drop 'too fast'.

Reduced is near-Orthogonal

In this picture there are two vectors which are far from orthogonal.

Small G-S Length



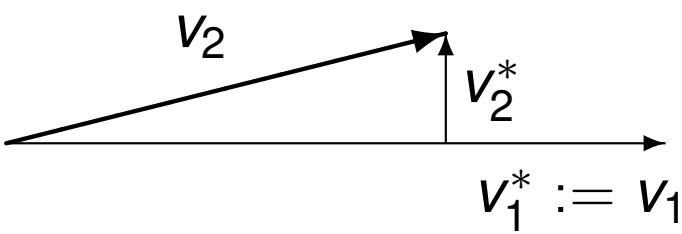
In this one the vectors are closer to orthogonal.

- A reduced basis is roughly sorted by G-S norms
- It is also a 'nearly orthogonal' basis

Reduced is near-Orthogonal

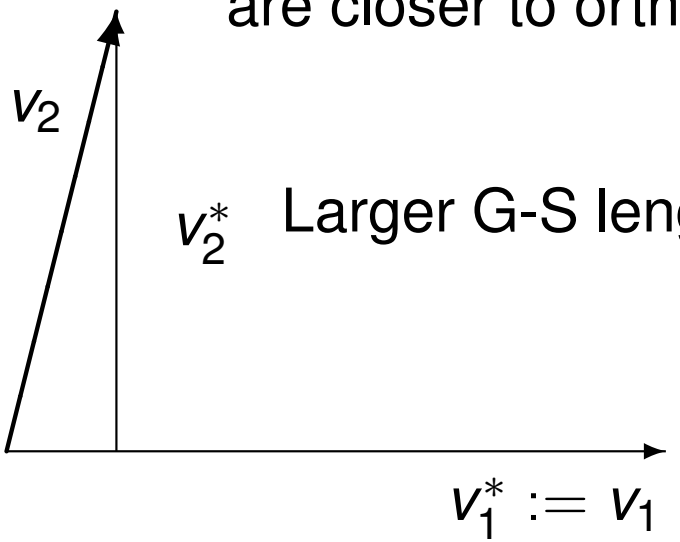
In this picture there are two vectors which are far from orthogonal.

Small G-S Length



In this one the vectors are closer to orthogonal.

Larger G-S length

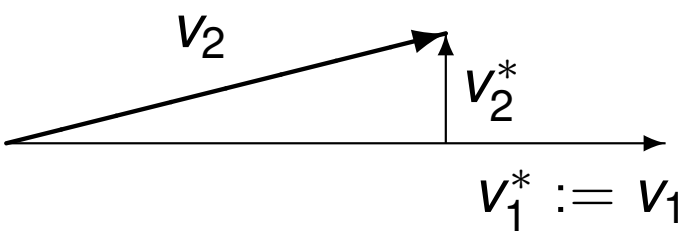


- A reduced basis is roughly sorted by G-S norms
- It is also a 'nearly orthogonal' basis

Reduced is near-Orthogonal

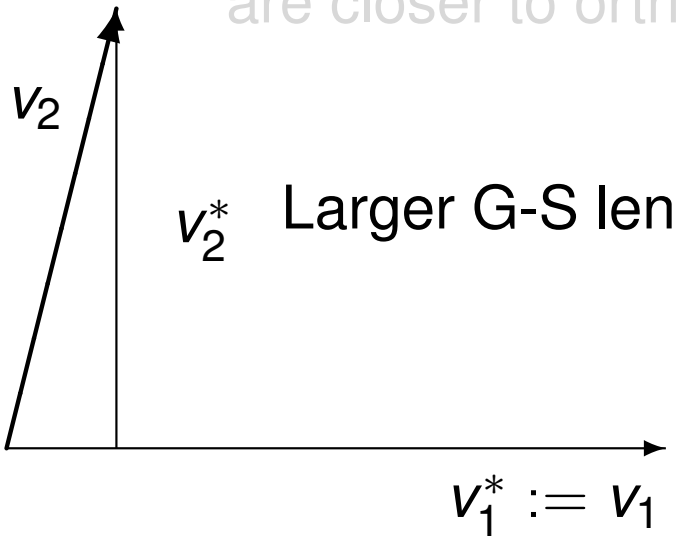
In this picture there are two vectors which are far from orthogonal.

Small G-S Length



In this one the vectors are closer to orthogonal.

Larger G-S length

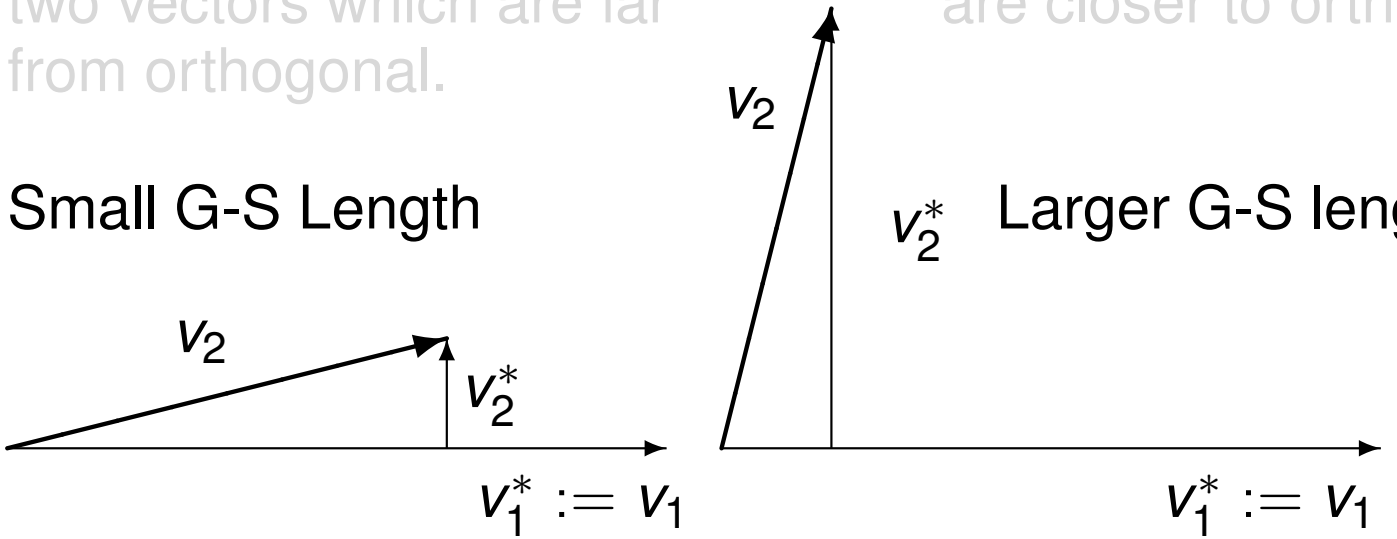


- A reduced basis is roughly sorted by G-S norms
- It is also a 'nearly orthogonal' basis

Reduced is near-Orthogonal

In this picture there are two vectors which are far from orthogonal.

In this one the vectors are closer to orthogonal.



- A reduced basis is roughly sorted by G-S norms
- It is also a 'nearly orthogonal' basis

Overview of LLL

Why a Reduced Basis is a Nice Basis

- The first vector of a reduced basis is small.
- For every $\mathbf{v} \in L$ we have:

$$\| \mathbf{b}_1 \| \leq 2^{(d-1)/2} \| \mathbf{v} \|$$

- For every basis of s vectors the following holds:
- Let $\mathbf{v} \in L$ with $\| \mathbf{v} \|^2 \leq B$. If $\| \mathbf{b}_d^* \|^2 > B$ then $\mathbf{v} \in \mathrm{SPAN}_{\mathbb{Z}}(\mathbf{b}_1, \ldots, \mathbf{b}_{s-1})$
- Reduced bases push large G-S norms to the rear helping to separate large and small.

Why a Reduced Basis is a Nice Basis

- The first vector of a reduced basis is small.
- For every $\mathbf{v} \in L$ we have:

$$\| \mathbf{b}_1 \| \leq 2^{(d-1)/2} \| \mathbf{v} \|$$

- For every basis of s vectors the following holds:
- Let $\mathbf{v} \in L$ with $\| \mathbf{v} \|^2 \leq B$. If $\| \mathbf{b}_d^* \|^2 > B$ then $\mathbf{v} \in \mathrm{SPAN}_{\mathbb{Z}}(\mathbf{b}_1, \ldots, \mathbf{b}_{s-1})$
- Reduced bases push large G-S norms to the rear helping to separate large and small.

Why a Reduced Basis is a Nice Basis

- The first vector of a reduced basis is small.
- For every $\mathbf{v} \in L$ we have:

$$\| \mathbf{b}_1 \| \leq 2^{(d-1)/2} \| \mathbf{v} \|$$

- For every basis of s vectors the following holds:
- Let $\mathbf{v} \in L$ with $\| \mathbf{v} \|^2 \leq B$. If $\| \mathbf{b}_d^* \|^2 > B$ then $\mathbf{v} \in \mathrm{SPAN}_{\mathbb{Z}}(\mathbf{b}_1, \ldots, \mathbf{b}_{s-1})$
- Reduced bases push large G-S norms to the rear helping to separate large and small.

Why a Reduced Basis is a Nice Basis

- The first vector of a reduced basis is small.
- For every $\mathbf{v} \in L$ we have:

$$\| \mathbf{b}_1 \| \leq 2^{(d-1)/2} \| \mathbf{v} \|$$

- For every basis of s vectors the following holds:
- Let $\mathbf{v} \in L$ with $\| \mathbf{v} \|^2 \leq B$. If $\| \mathbf{b}_d^* \|^2 > B$ then $\mathbf{v} \in \mathrm{SPAN}_{\mathbb{Z}}(\mathbf{b}_1, \ldots, \mathbf{b}_{s-1})$
- Reduced bases push large G-S norms to the rear helping to separate large and small.

The Most Common Lattice Question

The Shortest Vector Problem

Given a lattice, L, find the Shortest Vector in L.

- The Shortest Vector Problem (SVP) is NP-hard to even approximate to within a constant.
- The are many interesting research problems which can be connected to the SVP.
- One of the primary uses of lattice reduction algorithms is to approximately solve the SVP in polynomial time.
- Sometimes approximating can be enough to solve these interesting problems.

The Most Common Lattice Question

The Shortest Vector Problem

Given a lattice, L, find the Shortest Vector in L.

- The Shortest Vector Problem (SVP) is NP-hard to even approximate to within a constant.
- The are many interesting research problems which can be connected to the SVP.
- One of the primary uses of lattice reduction algorithms is to approximately solve the SVP in polynomial time.
- Sometimes approximating can be enough to solve these interesting problems.

The Most Common Lattice Question

The Shortest Vector Problem

Given a lattice, L, find the Shortest Vector in L.

- The Shortest Vector Problem (SVP) is NP-hard to even approximate to within a constant.
- The are many interesting research problems which can be connected to the SVP.
- One of the primary uses of lattice reduction algorithms is to approximately solve the SVP in polynomial time.
- Sometimes approximating can be enough to solve these interesting problems.

The Most Common Lattice Question

The Shortest Vector Problem

Given a lattice, L, find the Shortest Vector in L.

- The Shortest Vector Problem (SVP) is NP-hard to even approximate to within a constant.
- The are many interesting research problems which can be connected to the SVP.
- One of the primary uses of lattice reduction algorithms is to approximately solve the SVP in polynomial time.
- Sometimes approximating can be enough to solve these interesting problems.

The Most Common Lattice Question

The Shortest Vector Problem

Given a lattice, L, find the Shortest Vector in L.

- The Shortest Vector Problem (SVP) is NP-hard to even approximate to within a constant.
- The are many interesting research problems which can be connected to the SVP.
- One of the primary uses of lattice reduction algorithms is to approximately solve the SVP in polynomial time.
- Sometimes approximating can be enough to solve these interesting problems.

An Example: Algebraic Number Reconstruction

Finding a minpoly: Given an approximation $\tilde{\alpha} = \mathrm{Re}(\tilde{\alpha}) + i \cdot \mathrm{Im}(\tilde{\alpha})$.

Make a lattice, L, like this:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^0}) & C \cdot \mathrm{Im}(\tilde{\alpha^0}) \\ 0 & 1 & 0 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^1}) & C \cdot \mathrm{Im}(\tilde{\alpha^1}) \\ 0 & 0 & 1 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^2}) & C \cdot \mathrm{Im}(\tilde{\alpha^2}) \\ 0 & 0 & 0 & 1 & C \cdot \mathrm{Re}(\tilde{\alpha^3}) & C \cdot \mathrm{Im}(\tilde{\alpha^3}) \end{pmatrix}$$

Where C is a very large constant.

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

Then $(c_0, c_1, c_2, c_3, 0, 0) \in L$ and is smaller in size than the other vectors.

An Example: Algebraic Number Reconstruction

Finding a minpoly: Given an approximation $\tilde{\alpha} = \mathrm{Re}(\tilde{\alpha}) + i \cdot \mathrm{Im}(\tilde{\alpha})$.
Make a lattice, L, like this:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^0}) & C \cdot \mathrm{Im}(\tilde{\alpha^0}) \\ 0 & 1 & 0 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^1}) & C \cdot \mathrm{Im}(\tilde{\alpha^1}) \\ 0 & 0 & 1 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^2}) & C \cdot \mathrm{Im}(\tilde{\alpha^2}) \\ 0 & 0 & 0 & 1 & C \cdot \mathrm{Re}(\tilde{\alpha^3}) & C \cdot \mathrm{Im}(\tilde{\alpha^3}) \end{pmatrix}$$

Where C is a very large constant.
Let $\mathtt{minpoly}(\alpha) =: c_0 + c_1 x + c_2 x^2 + c_3 x^3$.
Then $(c_0, c_1, c_2, c_3, 0, 0) \in L$ and is smaller in size than the other vectors.

An Example: Algebraic Number Reconstruction

Finding a minpoly: Given an approximation $\tilde{\alpha} = \mathrm{Re}(\tilde{\alpha}) + i \cdot \mathrm{Im}(\tilde{\alpha})$.
Make a lattice, L, like this:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^0}) & C \cdot \mathrm{Im}(\tilde{\alpha^0}) \\ 0 & 1 & 0 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^1}) & C \cdot \mathrm{Im}(\tilde{\alpha^1}) \\ 0 & 0 & 1 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^2}) & C \cdot \mathrm{Im}(\tilde{\alpha^2}) \\ 0 & 0 & 0 & 1 & C \cdot \mathrm{Re}(\tilde{\alpha^3}) & C \cdot \mathrm{Im}(\tilde{\alpha^3}) \end{pmatrix}$$

Where C is a very large constant.
Let $\texttt{minpoly}(\alpha) =: c_0 + c_1 x + c_2 x^2 + c_3 x^3$.
Then $(c_0, c_1, c_2, c_3, 0, 0) \in L$ and is smaller in size than the other vectors.

An Example: Algebraic Number Reconstruction

Finding a minpoly: Given an approximation $\tilde{\alpha} = \mathrm{Re}(\tilde{\alpha}) + i \cdot \mathrm{Im}(\tilde{\alpha})$.
Make a lattice, L, like this:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^0}) & C \cdot \mathrm{Im}(\tilde{\alpha^0}) \\ 0 & 1 & 0 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^1}) & C \cdot \mathrm{Im}(\tilde{\alpha^1}) \\ 0 & 0 & 1 & 0 & C \cdot \mathrm{Re}(\tilde{\alpha^2}) & C \cdot \mathrm{Im}(\tilde{\alpha^2}) \\ 0 & 0 & 0 & 1 & C \cdot \mathrm{Re}(\tilde{\alpha^3}) & C \cdot \mathrm{Im}(\tilde{\alpha^3}) \end{pmatrix}$$

Where C is a very large constant.
Let $\texttt{minpoly}(\alpha) =: c_0 + c_1 x + c_2 x^2 + c_3 x^3$.
Then $(c_0, c_1, c_2, c_3, 0, 0) \in L$ and is smaller in size than the other vectors.

A quick ‘hit-list’ of some applications

- Polynomial Factoring
- Integer Relations/Algebraic Relations
- Cryptography (both attacking and encoding)
- Diophantine Equations
- Linear Programming
- Conjecture testing
- the Knapsack problem
- GCDs and kernels
- Table Maker’s Dilemma :)

Overview of LLL

A Lattice Reduction Algorithm

Most variants of LLL perform the following steps in one form or another:

1. *(Gram-Schmidt over $\mathbb{Z}$)*. By subtracting suitable $\mathbb{Z}$-linear combinations of $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ from $\mathbf{b}_k$.
2. *(LLL Switch)*. If interchanging $\mathbf{b}_k$ and $\mathbf{b}_{k+1}$ will increase $\| \mathbf{b}^*_{k+1} \|^2$ by a factor 2, then do so.
3. *(Repeat)*. If there was no such k in Step 2, then the algorithm stops. Otherwise go back to Step 1.

The cost of this algorithm has been roughly approximated as: '**the number of switches**' times '**the cost per switch**'

A Lattice Reduction Algorithm

Most variants of LLL perform the following steps in one form or another:

1. *(Gram-Schmidt over $\mathbb{Z}$)*. By subtracting suitable $\mathbb{Z}$-linear combinations of $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ from $\mathbf{b}_k$.
2. *(LLL Switch)*. If interchanging $\mathbf{b}_k$ and $\mathbf{b}_{k+1}$ will increase $\| \mathbf{b}^*_{k+1} \|^2$ by a factor 2, then do so.
3. *(Repeat)*. If there was no such k in Step 2, then the algorithm stops. Otherwise go back to Step 1.

The cost of this algorithm has been roughly approximated as: '**the number of switches**' times '**the cost per switch**'

A Lattice Reduction Algorithm

Most variants of LLL perform the following steps in one form or another:

1. *(Gram-Schmidt over $\mathbb{Z}$)*. By subtracting suitable $\mathbb{Z}$-linear combinations of $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ from $\mathbf{b}_k$.
2. *(LLL Switch)*. If interchanging $\mathbf{b}_k$ and $\mathbf{b}_{k+1}$ will increase $\| \mathbf{b}^*_{k+1} \|^2$ by a factor 2, then do so.
3. *(Repeat)*. If there was no such k in Step 2, then the algorithm stops. Otherwise go back to Step 1.

The cost of this algorithm has been roughly approximated as: '**the number of switches**' times '**the cost per switch**'

A Lattice Reduction Algorithm

Most variants of LLL perform the following steps in one form or another:

1. *(Gram-Schmidt over $\mathbb{Z}$)*. By subtracting suitable $\mathbb{Z}$-linear combinations of $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ from $\mathbf{b}_k$.
2. *(LLL Switch)*. If interchanging $\mathbf{b}_k$ and $\mathbf{b}_{k+1}$ will increase $\| \mathbf{b}^*_{k+1} \|^2$ by a factor 2, then do so.
3. *(Repeat)*. If there was no such k in Step 2, then the algorithm stops. Otherwise go back to Step 1.

The cost of this algorithm has been roughly approximated as: **'the number of switches'** times **'the cost per switch'**

A Tightly Packed Example

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

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

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

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

A Tightly Packed Example

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

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

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

A Tightly Packed Example

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

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

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

A Tightly Packed Example

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

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

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

A Tightly Packed Example

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

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

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

A Tightly Packed Example

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

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

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

A Tightly Packed Example

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

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

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

A Tightly Packed Example

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

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

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

A Tightly Packed Example

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

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

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

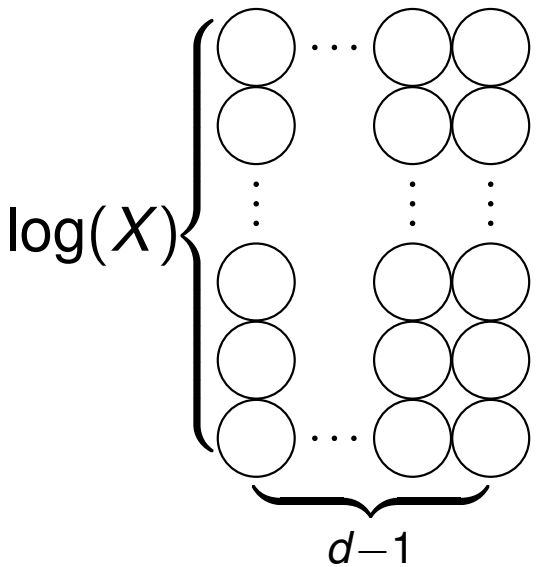
Overview of LLL

Counting Switches

LLL[82] counts switches:

$\mathcal{O}(d^2 \log(X))$

0 switches

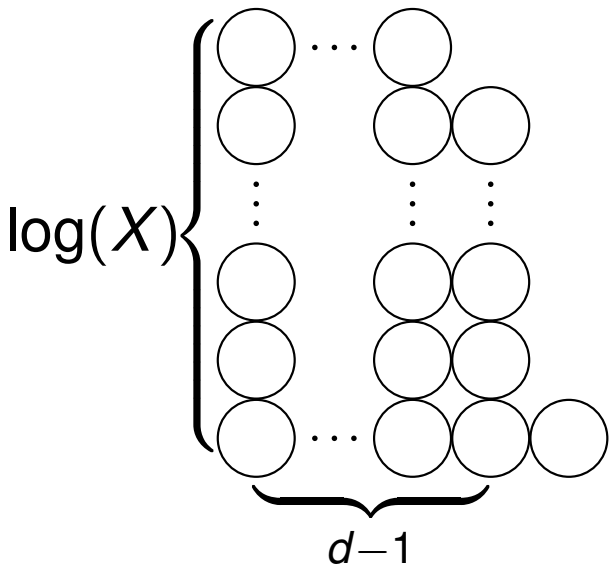


Counting Switches

LLL[82] counts switches:

$$\mathcal{O}(d^2 \log(X))$$

1 switch

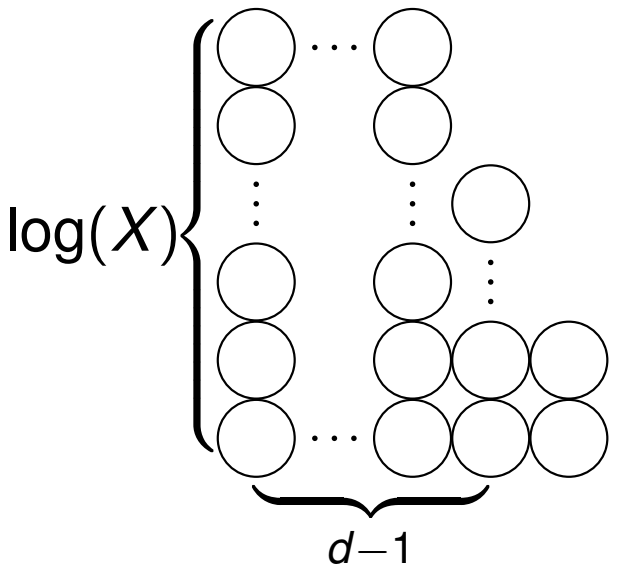


Counting Switches

LLL[82] counts switches:

$$\mathcal{O}(d^2 \log(X))$$

2 switches

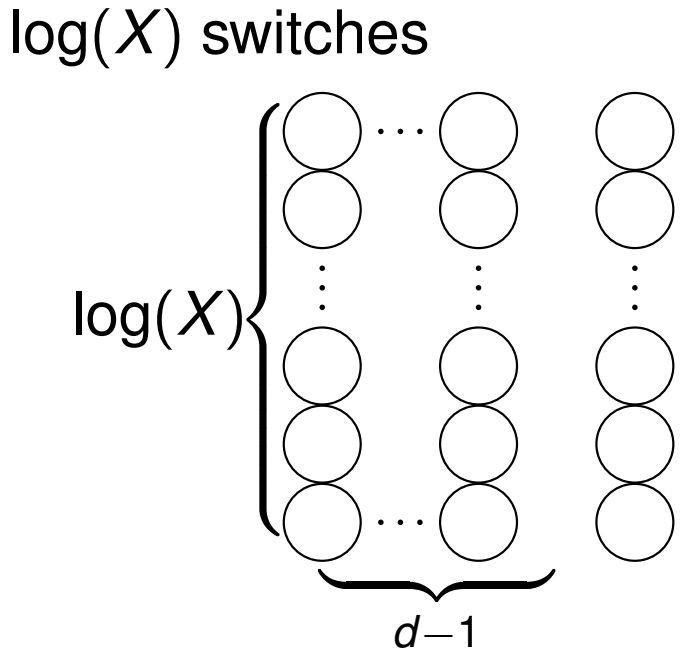


Counting Switches

LLL[82] counts switches:

$$\mathcal{O}(d^2 \log(X))$$

$$= \log(X) + \cdots$$

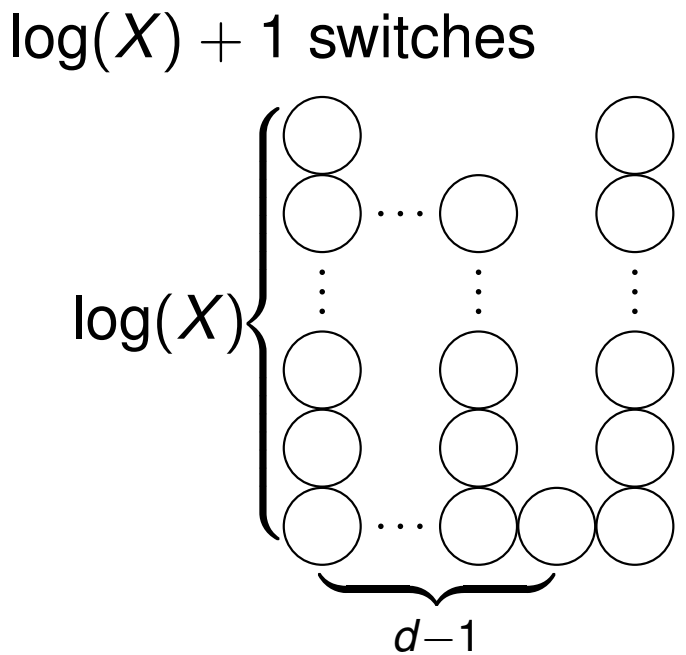


Counting Switches

LLL[82] counts switches:

$$\mathcal{O}(d^2 \log(X))$$

$$= \log(X) + \cdots$$

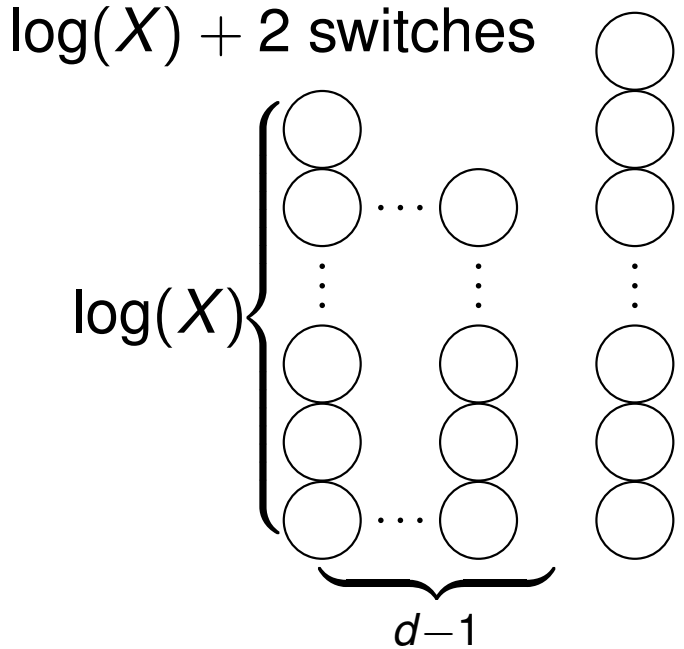


Counting Switches

LLL[82] counts switches:

$$\mathcal{O}(d^2 \log(X))$$

$$= \log(X) + \cdots$$

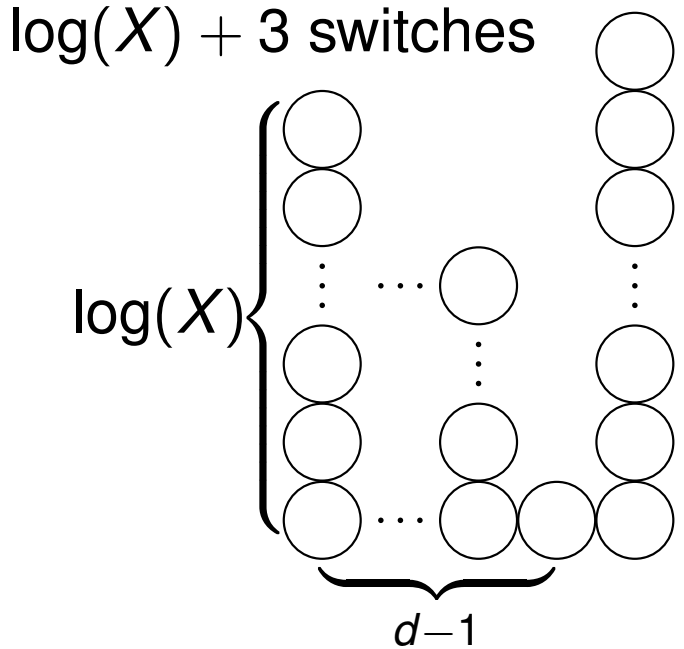


Counting Switches

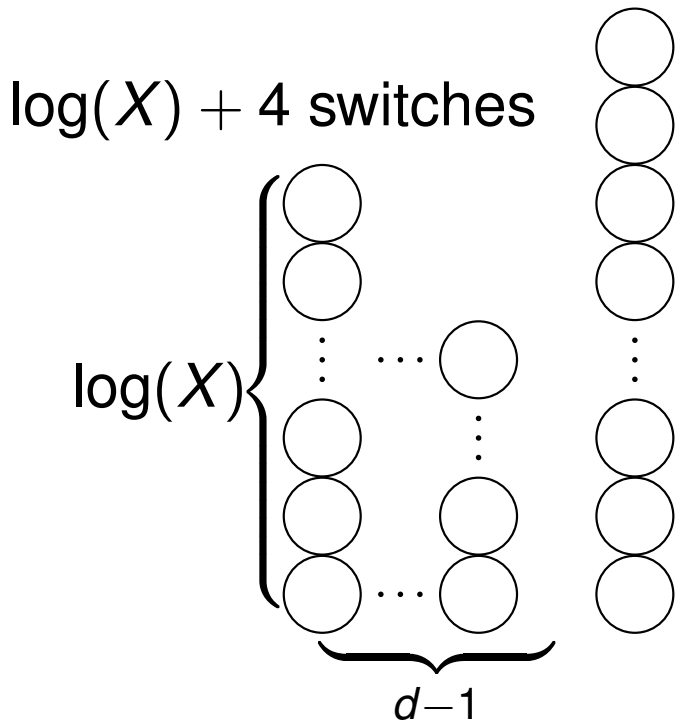
LLL[82] counts switches:

$$\mathcal{O}(d^2 \log(X))$$

$$= \log(X) + 2\log(X) + \cdots$$



Counting Switches



LLL[82] counts switches:

$$\mathcal{O}(d^2 \log(X))$$

$$= \log(X) + 2\log(X) +$$

$$\cdots + (d-1)\log(X)$$

Complexity of LLL

Parameters

Given a lattice basis, $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ with $\| \mathbf{b}_i \|^2 \leq X$ for all i. Return a reduced basis.

- The 1982 LLL paper does this in $\mathcal{O}(d^5 n \log^3(X)) = \mathcal{O}(d^2 \log X) \times \mathcal{O}(d) \times \mathcal{O}(n) \times \mathcal{M}(d \log X)$
- Several results weaken the strength of reduction to gain a complexity savings.
- Recent floating-point versions of LLL have attacked the cost of the size-reductions.
- We've attacked the number of switches if the user can bound the size of target vectors.

Complexity of LLL

Parameters

Given a lattice basis, $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ with $\| \mathbf{b}_i \|^2 \leq X$ for all i. Return a reduced basis.

- The 1982 LLL paper does this in $\mathcal{O}(d^5 n \log^3(X)) = \mathcal{O}(d^2 \log X) \times \mathcal{O}(d) \times \mathcal{O}(n) \times \mathcal{M}(d \log X)$
- Several results weaken the strength of reduction to gain a complexity savings.
- Recent floating-point versions of LLL have attacked the cost of the size-reductions.
- We've attacked the number of switches if the user can bound the size of target vectors.

Complexity of LLL

Parameters

Given a lattice basis, $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ with $\| \mathbf{b}_i \|^2 \leq X$ for all i. Return a reduced basis.

- The 1982 LLL paper does this in $\mathcal{O}(d^5 n \log^3(X)) = \mathcal{O}(d^2 \log X) \times \mathcal{O}(d) \times \mathcal{O}(n) \times \mathcal{M}(d \log X)$
- Several results weaken the strength of reduction to gain a complexity savings.
- Recent floating-point versions of LLL have attacked the cost of the size-reductions.
- We've attacked the number of switches if the user can bound the size of target vectors.

Complexity of LLL

Parameters

Given a lattice basis, $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ with $\| \mathbf{b}_i \|^2 \leq X$ for all i. Return a reduced basis.

- The 1982 LLL paper does this in $\mathcal{O}(d^5 n \log^3(X)) = \mathcal{O}(d^2 \log X) \times \mathcal{O}(d) \times \mathcal{O}(n) \times \mathcal{M}(d \log X)$
- Several results weaken the strength of reduction to gain a complexity savings.
- Recent floating-point versions of LLL have attacked the cost of the size-reductions.
- We've attacked the number of switches if the user can bound the size of target vectors.

An Example of an improvement strategy

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}$$ has a vector of length $\sqrt{102}$

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

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

A single call to LLL uses 24 swaps.

An Example of an improvement strategy

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}$$ has a vector of length $\sqrt{102}$

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

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

A single call to LLL uses 24 swaps.

An Example of an improvement strategy

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}$$ has a vector of length $\sqrt{102}$

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

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

A single call to LLL uses 24 swaps.

An Example of an improvement strategy

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}$$ has a vector of length $\sqrt{102}$

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

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

A single call to LLL uses 24 swaps.

An Example of an improvement strategy

$$\begin{pmatrix} 0 & 0 & 0 & 200001 \\ 1 & 0 & 0 & 90102 \\ 0 & 1 & 0 & 90403 \\ 0 & 0 & 1 & 90904 \end{pmatrix}$$ has a vector of length $\sqrt{102}$

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

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

A single call to LLL uses 24 swaps.

Thanks!

Thank you for your attention.