Gradual Sub-Lattice Reduction *
(now with more applications!)

Andy Novocin
andy@novocin.com

LIP, ENS de Lyon

September 23rd
The (gimmicky) Road Map
Gradual Sub-Lattice Reduction *

The Old Stuff
Lattice
Reduction
Lattice Reduction

The New Concepts
* 
Sub-
Gradual

The Bottom Line
The Complexity Result
New Complexities for Factoring Polynomials
Why give this talk?

- I want my work to be as *useful* as possible.
- This began as a new complexity for factoring polynomials.
- The result is actually much more about lattice reductions.
- Lattice reduction is used for more than just factoring.
- So I want to show you how this result *might* be applied.
  ... in the hope that you will find it useful.
Why give this talk?

- I want my work to be as *useful* as possible.
- This began as a new complexity for factoring polynomials
- The result is actually much more about lattice reductions
- Lattice reduction is used for more than just factoring
- So I want to show you how this result *might* be applied
  ... in the hope that you will find it useful
Why give this talk?

- I want my work to be as *useful* as possible.
- This began as a new complexity for factoring polynomials
- The result is actually much more about lattice reductions
- Lattice reduction is used for more than just factoring
- So I want to show you how this result *might* be applied
  ... in the hope that you will find it useful
Why give this talk?

- I want my work to be as *useful* as possible.
- This began as a new complexity for factoring polynomials
- The result is actually much more about lattice reductions
- Lattice reduction is used for more than just factoring
- So I want to show you how this result *might* be applied
  ... in the hope that you will find it useful
Why give this talk?

- I want my work to be as *useful* as possible.
- This began as a new complexity for factoring polynomials
- The result is actually much more about lattice reductions
- Lattice reduction is used for more than just factoring
- So I want to show you how this result *might* be applied
  ... in the hope that you will find it useful
Why give this talk?

- I want my work to be as *useful* as possible.
- This began as a new complexity for factoring polynomials.
- The result is actually much more about lattice reductions.
- Lattice reduction is used for more than just factoring.
- So I want to show you how this result *might* be applied...
  ... in the hope that you will find it *useful*.
Gradual Sub-Lattice Reduction *

The Old Stuff
Lattice Reduction

The New Concepts
Sub-Gradual

The Bottom Line
The Complexity Result
New Complexities for Factoring Polynomials
Introducing Lattices

A lattice, $L$ 

Definition

A lattice, $L$, is the set of all integer combinations of some set of vectors in $\mathbb{R}^n$

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

Every lattice has many bases... and we want to find a good basis!
Introducing Lattices

A lattice, $L$

Definition

A lattice, $L$, is the set of all integer combinations of some set of vectors in $\mathbb{R}^n$

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

Every lattice has many bases... and we want to find a good basis!
Introducing Lattices

A lattice, \( L \)

The same lattice, \( L \)

**Definition**

A lattice, \( L \), is the set of all integer combinations of some set of vectors in \( \mathbb{R}^n \)

Any minimal spanning set of \( L \) is called a *basis* of \( L \)

Every lattice has many bases... and we want to find a good basis!
Introducing Lattices

A lattice, $L$

The same lattice, $L$

**Definition**

A lattice, $L$, is the set of all integer combinations of some set of vectors in $\mathbb{R}^n$. Any minimal spanning set of $L$ is called a basis of $L$.

Every lattice has many bases... and we want to find a good basis!
Introducing Lattices

A lattice, \( L \)

The same lattice, \( L \)

---

Definition

A lattice, \( L \), is the set of all integer combinations of some set of vectors in \( \mathbb{R}^n \). Any minimal spanning set of \( L \) is called a basis of \( L \).

Every lattice has many bases... and we want to find a good basis!
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 areas 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.
- The algorithm in this talk is well suited for approximating the SVP (in some specific lattices).
- Sometimes approximating can be enough.
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.
- There are many interesting research areas 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.
- The algorithm in this talk is well suited for approximating the SVP (in some specific lattices).
- Sometimes approximating can be enough.
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 areas 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.
- The algorithm in this talk is well suited for approximating the SVP (in some specific lattices).
- Sometimes approximating can be enough.
### 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.
- There are many interesting research areas 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.
- The algorithm in this talk is well suited for approximating the SVP (in some specific lattices).
- Sometimes approximating can be enough.
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.
- There are many interesting research areas 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.
- The algorithm in this talk is well suited for approximating the SVP (in some specific lattices).
- Sometimes approximating can be enough.
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.
- There are many interesting research areas 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.
- The algorithm in this talk is well suited for approximating the SVP (in some specific lattices).
- Sometimes approximating can be enough.
An Example: Algebraic Number Reconstruction

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

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & C \cdot \text{Re}(\tilde{\alpha}^0) & C \cdot \text{Im}(\tilde{\alpha}^0) \\
0 & 1 & 0 & 0 & C \cdot \text{Re}(\tilde{\alpha}^1) & C \cdot \text{Im}(\tilde{\alpha}^1) \\
0 & 0 & 1 & 0 & C \cdot \text{Re}(\tilde{\alpha}^2) & C \cdot \text{Im}(\tilde{\alpha}^2) \\
0 & 0 & 0 & 1 & C \cdot \text{Re}(\tilde{\alpha}^3) & C \cdot \text{Im}(\tilde{\alpha}^3)
\end{pmatrix}
\]
Where \( C \) is a very large constant.
Let \( \text{minpoly}(\alpha) =: c_0 + c_1 x + c_2 x^2 + c_3 x^3 \).
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
\[ \tilde{\alpha} = \text{Re}(\tilde{\alpha}) + i \cdot \text{Im}(\tilde{\alpha}). \]
Make a lattice, \( L \), like this:

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

Where \( C \) is a very large constant.

Let \( \text{minpoly}(\alpha) =: c_0 + c_1 x + c_2 x^2 + c_3 x^3 \).
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
\[ \tilde{\alpha} = \text{Re}(\tilde{\alpha}) + i \cdot \text{Im}(\tilde{\alpha}). \]
Make a lattice, \( L \), like this:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & C \cdot \text{Re}(\tilde{\alpha}_0) & C \cdot \text{Im}(\tilde{\alpha}_0) \\
0 & 1 & 0 & 0 & C \cdot \text{Re}(\tilde{\alpha}_1) & C \cdot \text{Im}(\tilde{\alpha}_1) \\
0 & 0 & 1 & 0 & C \cdot \text{Re}(\tilde{\alpha}_2) & C \cdot \text{Im}(\tilde{\alpha}_2) \\
0 & 0 & 0 & 1 & C \cdot \text{Re}(\tilde{\alpha}_3) & C \cdot \text{Im}(\tilde{\alpha}_3)
\end{pmatrix}
\]

Where \( C \) is a very large constant.
Let \( \text{minpoly}(\alpha) =: c_0 + c_1 x + c_2 x^2 + c_3 x^3. \)
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
\( \tilde{\alpha} = \text{Re}(\tilde{\alpha}) + i \cdot \text{Im}(\tilde{\alpha}) \).
Make a lattice, \( L \), like this:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & C \cdot \text{Re}(\tilde{\alpha}_0) & C \cdot \text{Im}(\tilde{\alpha}_0) \\
0 & 1 & 0 & 0 & C \cdot \text{Re}(\tilde{\alpha}_1) & C \cdot \text{Im}(\tilde{\alpha}_1) \\
0 & 0 & 1 & 0 & C \cdot \text{Re}(\tilde{\alpha}_2) & C \cdot \text{Im}(\tilde{\alpha}_2) \\
0 & 0 & 0 & 1 & C \cdot \text{Re}(\tilde{\alpha}_3) & C \cdot \text{Im}(\tilde{\alpha}_3)
\end{pmatrix}
\]

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

The Old Stuff

Lattice Reduction

The New Concepts

Sub-Gradual

The Bottom Line

The Complexity Result
New Complexities for Factoring Polynomials
First we need to recall Gram-Schmidt Orthogonalization

Given a set of vectors $b_1, \ldots, b_d \in \mathbb{R}^n$ the Gram-Schmidt (G-S) process returns a set of orthogonal vectors $b_1^*, \ldots, b_d^*$ with the following properties:

- $b_1 = b_1^*$
- $\text{SPAN}_\mathbb{R}\{b_1, \ldots, b_i\} = \text{SPAN}_\mathbb{R}\{b_1^*, \ldots, b_i^*\}$

Intuition of GSO

My favorite way to think of G-S vectors is that $b_i^*$ is $b_i$ modded out by $b_1, \ldots, b_{i-1}$ over $\mathbb{R}$. 
First we need to recall Gram-Schmidt Orthogonalization

Given a set of vectors \( b_1, \ldots, b_d \in \mathbb{R}^n \) the Gram-Schmidt (G-S) process returns a set of orthogonal vectors \( b_1^*, \ldots, b_d^* \) with the following properties:

1. \( b_1 = b_1^* \)
2. \( \text{SPAN}_\mathbb{R}\{b_1, \ldots, b_i\} = \text{SPAN}_\mathbb{R}\{b_1^*, \ldots, b_i^*\} \)

Intuition of GSO

My favorite way to think of G-S vectors is that \( b_i^* \) is \( b_i \) modded out by \( b_1, \ldots, b_{i-1} \) over \( \mathbb{R} \).
First we need to recall Gram-Schmidt Orthogonalization

Given a set of vectors $b_1, \ldots, b_d \in \mathbb{R}^n$ the Gram-Schmidt (G-S) process returns a set of orthogonal vectors $b_1^*, \ldots, b_d^*$ with the following properties:

- $b_1 = b_1^*$
- $\text{SPAN}_{\mathbb{R}} \{b_1, \ldots, b_i\} = \text{SPAN}_{\mathbb{R}} \{b_1^*, \ldots, b_i^*\}$

Intuition of GSO

My favorite way to think of G-S vectors is that $b_i^*$ is $b_i$ modded out by $b_1, \ldots, b_{i-1}$ over $\mathbb{R}$. 
First we need to recall Gram-Schmidt Orthogonalization

Given a set of vectors \( b_1, \ldots, b_d \in \mathbb{R}^n \) the Gram-Schmidt (G-S) process returns a set of orthogonal vectors \( b_1^*, \ldots, b_d^* \) with the following properties:

\[
\begin{align*}
&\bullet \; b_1 = b_1^* \\
&\bullet \; \text{SPAN}_\mathbb{R}\{b_1, \ldots, b_i\} = \text{SPAN}_\mathbb{R}\{b_1^*, \ldots, b_i^*\}
\end{align*}
\]

**Intuition of GSO**

My favorite way to think of G-S vectors is that \( b_i^* \) is \( b_i \) modded out by \( b_1, \ldots, b_{i-1} \) over \( \mathbb{R} \).
Introducing A Reduced Basis

The goal of lattice reduction is to find a ‘nice’ basis for a given lattice.

A Reduced Basis

Let $b_1, \ldots, b_d$ be a basis for a lattice, $L$, and let $b_j^*$ be the $j^{th}$ G-S vector.
Then we call the basis $(\delta, \eta)$-reduced, for $\delta \in (1/4, 1], \eta \in [1/2, \sqrt{\delta})$, when:

$$\| b_i^* \|^2 \leq \left( \frac{1}{\delta - \eta^2} \right) \cdot \| b_{i+1}^* \|^2 \ \forall i < d$$

In the original LLL paper the values $(\delta, \eta) := (3/4, 1/2)$ were chosen so that $\| b_i^* \|^2 \leq 2 \| b_{i+1}^* \|^2$.
A reduced basis cannot be too far from orthogonal. In particular the G-S lengths do not drop ‘too’ fast.
Introducing A Reduced Basis

The goal of lattice reduction is to find a ‘nice’ basis for a given lattice.

A Reduced Basis

Let \( b_1, \ldots, b_d \) be a basis for a lattice, \( L \), and let \( b_j^* \) be the \( j^{th} \) G-S vector. Then we call the basis \((\delta, \eta)\)-reduced, for \( \delta \in (1/4, 1], \eta \in [1/2, \sqrt{\delta}) \), when:

\[
\| b_i^* \|^2 \leq \left( \frac{1}{\delta - \eta^2} \right) \cdot \| b_{i+1}^* \|^2 \quad \forall i < d
\]

In the original LLL paper the values \((\delta, \eta) := (3/4, 1/2)\) were chosen so that \( \| b_i^* \|^2 \leq 2 \| b_{i+1}^* \|^2 \). A reduced basis cannot be too far from orthogonal. In particular the G-S lengths do not drop ‘too’ fast.
Introducing A Reduced Basis

The goal of lattice reduction is to find a ‘nice’ basis for a given lattice.

A Reduced Basis

Let \( b_1, \ldots, b_d \) be a basis for a lattice, \( L \), and let \( b_j^* \) be the \( j^{th} \) G-S vector. Then we call the basis \((\delta, \eta)\)-reduced, for \( \delta \in (1/4, 1], \eta \in [1/2, \sqrt{\delta}) \), when:

\[
\| b_i^* \|_2^2 \leq \left( \frac{1}{\delta - \eta^2} \right) \cdot \| b_{i+1}^* \|_2^2 \quad \forall i < d
\]

In the original LLL paper the values \((\delta, \eta) := (3/4, 1/2)\) were chosen so that \( \| b_i^* \|_2^2 \leq 2 \| b_{i+1}^* \|_2^2 \).

A reduced basis cannot be too far from orthogonal. In particular the G-S lengths do not drop ‘too’ fast.
Introducing A Reduced Basis

The goal of lattice reduction is to find a ‘nice’ basis for a given lattice.

A Reduced Basis

Let $b_1, \ldots, b_d$ be a basis for a lattice, $L$, and let $b^*_j$ be the $j^{th}$ G-S vector. Then we call the basis $(\delta, \eta)$-reduced, for $\delta \in (1/4, 1], \eta \in [1/2, \sqrt{\delta})$, when:

$$\| b^*_i \|^2 \leq \left( \frac{1}{\delta - \eta^2} \right) \cdot \| b^*_{i+1} \|^2 \quad \forall i < d$$

In the original LLL paper the values $(\delta, \eta) := (3/4, 1/2)$ were chosen so that $\| b^*_i \|^2 \leq 2 \| b^*_{i+1} \|^2$.

A reduced basis cannot be too far from orthogonal. In particular the G-S lengths do not 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.

Larger G-S length

- LLL searches for a nearly orthogonal basis.
- It does this by rearranging basis vectors such that latter vectors have long G-S lengths and 'modding out' by previous vectors over \( \mathbb{Z} \).
Reduced is near-Orthogonal

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

\[ \mathbf{v}_1^* := \mathbf{v}_1 \]

\[ \mathbf{v}_2^* := \mathbf{v}_2 \]

Small G-S Length

In this one the vectors are closer to orthogonal.

\[ \mathbf{v}_1^* := \mathbf{v}_1 \]

\[ \mathbf{v}_2^* \]

Larger G-S length

- LLL searches for a nearly orthogonal basis.
- It does this by rearranging basis vectors such that latter vectors have long G-S lengths and 'modding out' by previous vectors over \( \mathbb{Z} \).
Reduced is near-Orthogonal

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

Small G-S Length

\[ v_2^* := v_1 \]

Larger G-S length

\[ v_2^* := v_1 \]

- LLL searches for a nearly orthogonal basis.
- It does this by rearranging basis vectors such that latter vectors have long G-S lengths and ’modding out’ by previous vectors over \( \mathbb{Z} \).
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

- LLL searches for a nearly orthogonal basis.
- It does this by rearranging basis vectors such that latter vectors have long G-S lengths and ’modding out’ by previous vectors over $\mathbb{Z}$. 
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

- LLL searches for a nearly orthogonal basis.
- It does this by rearranging basis vectors such that latter vectors have long G-S lengths and 'modding out' by previous vectors over $\mathbb{Z}$. 
Reduced is near-Orthogonal

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

Small G-S Length

\[ v_1^* := v_1 \]

\[ v_2^* :\]

\[ v_2 \]

In this one the vectors are closer to orthogonal.

Larger G-S length

\[ v_2^* \]

\[ v_1^* := v_1 \]

• LLL searches for a nearly orthogonal basis.
• It does this by rearranging basis vectors such that latter vectors have long G-S lengths and ‘modding out’ by previous vectors over \( \mathbb{Z} \).
Reduced is near-Orthogonal

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

**Small G-S Length**

\[ v_2^* : = v_1 \]

In this one the vectors are closer to orthogonal.

**Larger G-S length**

\[ v_2^* : = v_1 \]

- LLL searches for a nearly orthogonal basis.
- It does this by rearranging basis vectors such that latter vectors have long G-S lengths and 'modding out' by previous vectors over \( \mathbb{Z} \).
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

- LLL searches for a nearly orthogonal basis.
- It does this by rearranging basis vectors such that latter vectors have long G-S lengths and ‘modding out’ by previous vectors over \( \mathbb{Z} \).
Reduced is near-Orthogonal

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

Small G-S Length

\[ v_1^* := v_1 \]

\[ v_2 \]

\[ v_2^* \]

In this one the vectors are closer to orthogonal.

Larger G-S length

\[ v_1^* := v_1 \]

\[ v_2^* \]

• LLL searches for a nearly orthogonal basis.
• It does this by rearranging basis vectors such that latter vectors have long G-S lengths and ’modding out’ by previous vectors over \( \mathbb{Z} \).
A Reduced Basis is a Nice Basis

Nice traits of a reduced basis:

- The first vector is not far from the shortest vector in the lattice. For every $v \in L$ we have:
  \[ \| b_1 \| \leq 2^{(d-1)/2} \| v \| \]

- The later vectors have longer Gram-Schmidt length than when LLL began. This is useful because of the following property which is true for any basis, $b_1, \ldots, b_d$:
  
  For every $v \in L$ with $\| v \|_2^2 \leq B$. If $\| b_d^* \|_2^2 > B$ then $v \in \text{SPAN}_\mathbb{Z}(b_1, \ldots, b_{d-1})$.

- The basic idea is that LLL can separate the small vectors from the large vectors, if we can create a large enough gap in their sizes.
A Reduced Basis is a Nice Basis

Nice traits of a reduced basis:

- The first vector is not far from the shortest vector in the lattice. For every $v \in L$ we have:
  \[ \| b_1 \| \leq 2^{(d-1)/2} \| v \| \]

- The later vectors have longer Gram-Schmidt length than when LLL began. This is useful because of the following property which is true for any basis, $b_1, \ldots, b_d$:

  For every $v \in L$ with $\| v \| ^2 \leq B$. If $\| b_d^* \| ^2 > B$ then $v \in \text{SPAN}_\mathbb{Z}(b_1, \ldots, b_{d-1})$.

- The basic idea is that LLL can separate the small vectors from the large vectors, if we can create a large enough gap in their sizes.
A Reduced Basis is a Nice Basis

Nice traits of a reduced basis:

- The first vector is not far from the shortest vector in the lattice. For every $v \in L$ we have:

  $$\| b_1 \| \leq 2^{(d-1)/2} \| v \|$$

- The later vectors have longer Gram-Schmidt length than when LLL began. This is useful because of the following property which is true for any basis, $b_1, \ldots, b_d$:

  For every $v \in L$ with $\| v \|^2 \leq B$. If $\| b_d^* \|^2 > B$ then $v \in \text{SPAN}_{\mathbb{Z}}(b_1, \ldots, b_{d-1})$.

- The basic idea is that LLL can separate the small vectors from the large vectors, if we can create a large enough gap in their sizes.
Gradual Sub-Lattice Reduction *

The Old Stuff
Lattice Reduction

The New Concepts
* Sub-Gradual

The Bottom Line
The Complexity Result
New Complexities for Factoring Polynomials
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 \( b_1, \ldots, b_{i-1} \) from \( b_i \).

2. *(LLL Switch).* If there is a \( k \) such that interchanging \( b_{k-1} \) and \( b_k \) will increase \( \| b_k^* \|^2 \) by a factor \( 1/\delta \), 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 \( b_1, \ldots, b_{i-1} \) from \( b_i \).

2. *(LLL Switch).* If there is a \( k \) such that interchanging \( b_{k-1} \) and \( b_k \) will increase \( \| b_k^* \|_2 \) by a factor \( 1/\delta \), 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 \( b_1, \ldots, b_{i-1} \) from \( b_i \).

2. *(LLL Switch).* If there is a \( k \) such that interchanging \( b_{k-1} \) and \( b_k \) will increase \( \| b_k^* \|^2 \) by a factor \( 1/\delta \), 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 $b_1, \ldots, b_{i-1}$ from $b_i$.

2. *(LLL Switch).* If there is a $k$ such that interchanging $b_{k-1}$ and $b_k$ will increase $\|b_k^*\|^2$ by a factor $1/\delta$, 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

\[
\begin{pmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
\end{pmatrix}
\]
### A Tightly Packed Example

\[
\begin{pmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
0 & 0 & 5 & 0 \\
10 & 20 & 5 & 1 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
0 & 20 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
0 & 20 & 0 & 0 \\
0 & 20 & 0 & 0 \\
\end{pmatrix}
\]
A Tightly Packed Example

\[
\begin{pmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
0 & 0 & 5 & 0 \\
10 & 20 & 5 & 1 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 0 & 5 & 1 \\
10 & 0 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 5 & 0 \\
0 & 0 & 0 & 1 \\
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0 \\
0 & 20 & 0 & 0 \\
\end{pmatrix}
\]
A Tightly Packed Example

\[
\begin{bmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0 \\
10 & 0 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 0 & 0 \\
0 & 20 & 0 & 0
\end{bmatrix}
\]
A Tightly Packed Example

\[
\begin{pmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
0 & 0 & 0 & 1 \\
10 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0 \\
0 & 20 & 0 & 0
\end{pmatrix}
\]
### A Tightly Packed Example

\[
\begin{pmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 0 & 1 \\
10 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0
\end{pmatrix}
\]
A Tightly Packed Example

\[
\begin{pmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0 \\
\end{pmatrix}
\]
A Tightly Packed Example

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

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

\[
\begin{pmatrix}
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0 
\end{pmatrix}
\]
A Tightly Packed Example

\[
\begin{bmatrix}
10 & 0 & 0 & 0 \\
10 & 20 & 0 & 0 \\
10 & 20 & 5 & 0 \\
10 & 20 & 5 & 1 \\
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
0 & 20 & 0 & 0 \\
10 & 20 & 5 & 1 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 \\
10 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 20 & 0 & 0
\end{bmatrix}
\]
Today’s Complexity Goal (kind of . . .)

Parameters

Given a lattice basis, \( b_1, \ldots, b_d \in \mathbb{R}^n \) with \( \| b_i \|^2 \leq X \) for all \( i \).
Return a reduced basis.

- The 1982 LLL paper does this in \( \mathcal{O}(d^5n \log^3(X)) \)
- The 2005 Nguyen and Stehlé paper does this in \( \mathcal{O}(d^4n(d + \log(X)) \log(X)) \)
- We will try to do something like this on some types of input in something like \( \mathcal{O}(d^7 + d^5 \log(X)) \)
- It’s actually \( \mathcal{O}((r + N)r^3(r + \log(B))(\log(X) + (r + N)(r + \log(B)))) \) for a reduced basis of a sub-lattice.
- My goal today is to explain this result, and why/how to use it in applications.
Today’s Complexity Goal (kind of . . .)

Parameters

Given a lattice basis, $b_1, \ldots, b_d \in \mathbb{R}^n$ with $\| b_i \|^2 \leq X$ for all $i$. Return a reduced basis.

- The 1982 LLL paper does this in $O(d^5 n \log^3 (X))$
- The 2005 Nguyen and Stehlé paper does this in $O(d^4 n(d + \log(X)) \log(X))$
- We will try to do something like this on some types of input in something like $O(d^7 + d^5 \log(X))$
- It’s actually $O((r + N)r^3(r + \log(B))(\log(X) + (r + N)(r + \log(B))))$ for a reduced basis of a sub-lattice.
- My goal today is to explain this result, and why/how to use it in applications.
Today’s Complexity Goal (kind of. . .)

Parameters

Given a lattice basis, \( b_1, \ldots, b_d \in \mathbb{R}^n \) with \( \| b_i \|^2 \leq X \) for all \( i \).
Return a reduced basis.

- The 1982 LLL paper does this in \( \mathcal{O}(d^5 n \log^3(X)) \)
- The 2005 Nguyen and Stehlé paper does this in \( \mathcal{O}(d^4 n(d + \log(X)) \log(X)) \)
- We will try to do something like this on some types of input in something like \( \mathcal{O}(d^7 + d^5 \log(X)) \)
- It’s actually \( \mathcal{O}((r + N)r^3(r + \log(B))(\log(X) + (r + N)(r + \log(B)))) \) for a reduced basis of a sub-lattice.
- My goal today is to explain this result, and why/how to use it in applications.
Today’s Complexity Goal (kind of . . .)

Parameters

Given a lattice basis, $b_1, \ldots, b_d \in \mathbb{R}^n$ with $\| b_i \|^2 \leq X$ for all $i$. Return a reduced basis.

- The 1982 LLL paper does this in $O(d^5 n \log^3(X))$
- The 2005 Nguyen and Stehlé paper does this in $O(d^4 n(d + \log(X)) \log(X))$
- We will try to do something like this on some types of input in something like $O(d^7 + d^5 \log(X))$
- It’s actually $O(((r + N)r^3(r + \log(B))(\log(X) + (r + N)(r + \log(B))))$ for a reduced basis of a sub-lattice.
- My goal today is to explain this result, and why/how to use it in applications.
Gradual Sub-Lattice Reduction *

The Old Stuff
Lattice Reduction

The New Concepts
Sub-Gradual

The Bottom Line
The Complexity Result
New Complexities for Factoring Polynomials
Knapsack Lattices

The asterisk

So far our algorithm only operates on the following types of lattices:

$$
\begin{pmatrix}
1 & 0 & \cdots & 0 & x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\
0 & 1 & \cdots & 0 & x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & x_{r,1} & x_{r,2} & \cdots & x_{r,N}
\end{pmatrix}
$$

Although many interesting problems can fit these formats.
Knapsack Lattices

The asterisk

So far our algorithm only operates on the following types of lattices:

\[
\begin{pmatrix}
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & P_N \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & P_2 & \cdots & 0 \\
0 & 0 & \cdots & 0 & P_1 & 0 & \cdots & 0 \\
1 & 0 & \cdots & 0 & x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\
0 & 1 & \cdots & 0 & x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & x_{r,1} & x_{r,2} & \cdots & x_{r,N}
\end{pmatrix}
\]

Although many interesting problems can fit these formats.
Knapsack Lattices

The asterisk

So far our algorithm only operates on the following types of lattices:

\[
\begin{pmatrix}
1 & 0 & \cdots & 0 & x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\
0 & 1 & \cdots & 0 & x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & x_{r,1} & x_{r,2} & \cdots & x_{r,N}
\end{pmatrix}
\]

Although many interesting problems can fit these formats.
Gradual Sub-Lattice Reduction *

The Old Stuff
Lattice Reduction

The New Concepts
* Sub-Gradual

The Bottom Line
The Complexity Result
New Complexities for Factoring Polynomials
The Switch Picture

LLL[82] counts switches:

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

0 switches

\[
\begin{align*}
\log(X) & \quad \ldots \\
\vdots & \quad \ldots \\
\ldots & \quad \ldots \\
\ldots & \quad \ldots \\
\ldots & \quad \ldots \\
0 & \quad \ldots
\end{align*}
\]
The Switch Picture

LLL[82] counts switches:

\[ O(d^2 \log(X)) \]

1 switch

\[ \log(X) \]

\[ d - 1 \]
The Switch Picture

LLL[82] counts switches:

$O(d^2 \log(X))$

2 switches

$log(X)$

$d-1$
The Switch Picture

LLL[82] counts switches:

\[ \mathcal{O}(d^2 \log(X)) = \log(X) + \cdots \]
The Switch Picture

LLL[82] counts switches:

$$O(d^2 \log(X))$$

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

- $d-1$ switches
LLL[82] counts switches:

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

$$= \log(X) + \ldots$$
LLL[82] counts switches:

$$O(d^2 \log(X))$$

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

The Switch Picture

The Old Stuff

The New Concepts

The Bottom Line
LLL[82] counts switches:

\[ O(d^2 \log(X)) \]

\[ = \log(X) + 2 \log(X) + \cdots + (d - 1) \log(X) \]
It’s a Better Picture with a Sub-Lattice

In problems where we want vectors of length \( \leq B \),
We can prove a ‘better’ bound for the number of switches.

\[
\leq O(d^2(d + \log(B)))
\]
Gradual Sub-Lattice Reduction *

The Old Stuff
Lattice Reduction

The New Concepts
* Sub-Gradual

The Bottom Line
The Complexity Result
New Complexities for Factoring Polynomials
Ideas from Factoring

- Hoeij and Belabas experimented on factoring polynomials using knapsack type lattices.
- Hoeij’s approach uses the pertinent columns one at a time rather than the entire lattice.
- Further, Belabas’ approach uses each column’s data in sections, most significant bits first.
- So to reduce the basis, many calls are used to LLL, not just one.
- The total time, in practice, for many calls is better than a single call.
- The CPU’s work was not distributed evenly between the many calls to LLL.
Ideas from Factoring

- Hoeij and Belabas experimented on factoring polynomials using knapsack type lattices.
- Hoeij’s approach uses the pertinent columns one at a time rather than the entire lattice.
- Further, Belabas’ approach uses each column’s data in sections, most significant bits first.
- So to reduce the basis, many calls are used to LLL, not just one.
- The total time, in practice, for many calls is better than a single call.
- The CPU’s work was not distributed evenly between the many calls to LLL.
Ideas from Factoring

- Hoeij and Belabas experimented on factoring polynomials using knapsack type lattices.
- Hoeij’s approach uses the pertinent columns one at a time rather than the entire lattice.
- Further, Belabas’ approach uses each column’s data in sections, most significant bits first.
- So to reduce the basis, many calls are used to LLL, not just one.
- The total time, in practice, for many calls is better than a single call.
- The CPU’s work was not distributed evenly between the many calls to LLL.
Ideas from Factoring

- Hoeij and Belabas experimented on factoring polynomials using knapsack type lattices.
- Hoeij’s approach uses the pertinent columns one at a time rather than the entire lattice.
- Further, Belabas’ approach uses each column’s data in sections, most significant bits first.
- So to reduce the basis, many calls are used to LLL, not just one.
- The total time, in practice, for many calls is better than a single call.
- The CPU’s work was not distributed evenly between the many calls to LLL.
Ideas from Factoring

- Hoeij and Belabas experimented on factoring polynomials using knapsack type lattices.
- Hoeij’s approach uses the pertinent columns one at a time rather than the entire lattice.
- Further, Belabas’ approach uses each column’s data in sections, most significant bits first.
- So to reduce the basis, many calls are used to LLL, not just one.
- The total time, in practice, for many calls is better than a single call.
- The CPU’s work was not distributed evenly between the many calls to LLL.
Ideas from Factoring

- Hoeij and Belabas experimented on factoring polynomials using knapsack type lattices.
- Hoeij's approach uses the pertinent columns one at a time rather than the entire lattice.
- Further, Belabas' approach uses each column's data in sections, most significant bits first.
- So to reduce the basis, many calls are used to LLL, not just one.
- The total time, in practice, for many calls is better than a single call.
- The CPU's work was not distributed evenly between the many calls to LLL.
An Example

\[
\begin{pmatrix}
0 & 0 & 0 & 200001 \\
1 & 0 & 0 & 90102 \\
0 & 1 & 0 & 90403 \\
0 & 0 & 1 & 90904
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
0 & 0 & 0 & 200001 \\
1 & 0 & 0 & 90102 \\
0 & 1 & 0 & 90403 \\
0 & 0 & 1 & 90904
\end{pmatrix}
\] 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

\[
\begin{pmatrix}
0 & 0 & 0 & 200001 \\
1 & 0 & 0 & 90102 \\
0 & 1 & 0 & 90403 \\
0 & 0 & 1 & 90904 \\
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
0 & 0 & 0 & 200001 \\
1 & 0 & 0 & 90102 \\
0 & 1 & 0 & 90403 \\
0 & 0 & 1 & 90904 \\
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
0 & 0 & 0 & 200001 \\
1 & 0 & 0 & 90102 \\
0 & 1 & 0 & 90403 \\
0 & 0 & 1 & 90904 \\
\end{pmatrix}
\]
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.
Gradual Sub-Lattice Reduction *

The Old Stuff
Lattice Reduction

The New Concepts
* Sub-Gradual

The Bottom Line
The Complexity Result
New Complexities for Factoring Polynomials
A sketch of the Algorithm I

Input:

\[ B \text{ and } L = \text{RowSpace} \begin{pmatrix} 
0 & \cdots & 0 & 0 & \cdots & P_N \\
0 & \cdots & 0 & 0 & \ddots & 0 \\
0 & \cdots & 0 & P_1 & \cdots & 0 \\
1 & \cdots & 0 & x_{1,1} & \cdots & x_{1,N} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 1 & x_{r,1} & \cdots & x_{r,N} 
\end{pmatrix} \]

Output: A reduced basis which generates a sub-lattice \( L' \subseteq L \) such that if \( \mathbf{v} \in L \) and \( \| \mathbf{v} \|^2 \leq B \) then \( \mathbf{v} \in L' \).
A sketch of the Algorithm II

The Main Algorithm:

1. $s := r; M := I_{r \times r}$

2. for $j = 1 \ldots N$ do:
   2.1 $y_j := M[1, \ldots, r] \cdot x_j; \quad l := \left\lceil \log_2 \left( \max(P_j, |y_j|_\infty, 2) \right) \right\rceil$
   2.2 $M := \begin{bmatrix} 0 & \frac{P_j/2^l}{M} \\ M & \frac{y_j/2^l}{M} \end{bmatrix}; \quad s := s + 1$

2.3 while $(l \neq 0)$ do:
   2.3.1 $y_j := 2^l \cdot M \cdot [0, \ldots, 0, 1]^T; \quad l := \max\{0, \left\lceil \log_2 \left( \frac{|y_j|_\infty}{\alpha^2cB} \right) \right\rceil \}$
   2.3.2 $M := \begin{bmatrix} M[1, \ldots, r + j - 1] & y_j/2^l \end{bmatrix}$
   2.3.3 Call LLL_with_removals$(M)$ set $M$ to the output; Adjust $s$
The Complexity Comparison

\[ O((r + N)c^3(c + \log(B))(\log(X) + (r + N)(c + \log(B)))) \]
with \( c = r + N \) or \( O(r) \) a bound on the number of vectors

On a square input for some typical values of \( B \):

<table>
<thead>
<tr>
<th></th>
<th>( L^2 )</th>
<th>( O(d^6 \log(X) + d^5 \log^2(X)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B = O(X) )</td>
<td>( O(d^7 + d^5 \log^2(X)) )</td>
<td></td>
</tr>
<tr>
<td>( B = O(X^{1/d}) )</td>
<td>( O(r^5 d^2 + r^3 \log^2(X)) )</td>
<td></td>
</tr>
<tr>
<td>( B = 2^{O(d)} )</td>
<td>( O(d^4 r^3 + d^2 r^3 \log(X)) )</td>
<td></td>
</tr>
</tbody>
</table>
Features of the Proof

\[ O((r + N)c^3(c + \log(B))(\log(X) + (r + N)(c + \log(B)))) \]
with \( c = r + N \) or \( O(r) \)

- The size of the vectors remains \( O(c + \log(B)) \)
- The total number of scalings is \( O(r + N) \)
- The total number of switches is \( O((r + N)c(c + \log(B))) \)
Gradual Sub-Lattice Reduction

The Old Stuff
Lattice Reduction

The New Concepts
* Sub-Gradual

The Bottom Line
The Complexity Result
New Complexities for Factoring Polynomials
Belabas, Kleuners, van Hoeij, and Steel showed that reducing the following basis will factor a polynomial.

$$\begin{pmatrix}
1 & * & \cdots & * \\
\vdots & \vdots & \ddots & \vdots \\
1 & * & \cdots & * \\
\end{pmatrix}$$

Any vector which corresponds with a factor has size $\leq r + 1$, so we choose $B = r + 1$.
Comparing with Schönhage

If we apply our algorithm to the [BHKS] result then we can factor a polynomial with degree \( N \) and height \( H \) with complexity:

\[ \mathcal{O}(N^3 r^4 + N^2 r^4 \log(H)) \]

This is the first improvement since 1984 when Schönhage gives:

\[ \mathcal{O}(N^8 + N^5 \log^3(H)) \]
Other applications

Algebraic Number Reconstruction

- Suppose $h(x)$ is an unknown polynomial of degree $d$ and maximal coefficient $\leq H$.
- Now give me $O(d^2 + d \log H)$ bits of a root of $h$.
- Using the lattice I showed you earlier we can find $h(x)$.
- Our algorithm improves the complexity bound of this problem a factor $O(d^2)$ to $O(d^7 + d^5 \log^2 H)$.
Thanks!

Thank you for your attention.