The state of the art in polynomial factorization

Andy Novocin
École Normale Supérieure de Lyon

EVA-flo 2010

Outline:

The Basic Algorithmic Problem:

Given $f \in \mathbb{Z}[x]$ find a complete irreducible factorization, $g_1 \cdots g_k = f$ in $\mathbb{Z}[x]$ (as quickly as possible).

My Personal Research Objective:

Prove near-sharp complexity bounds for highly practical (useful and optimized) algorithms.

The Historical Gap in Polynomial Factoring

The best algorithm in theory vs. in practice:

Year	**Best Provable Bound**	**Best Probable Bound**
1969	Zassenhaus	Zassenhaus
1982	LLL	Zassenhaus
2002	LLL	van Hoeij
2004	LLL	Belabas
2010	Hoeij/Novocin	Hart/Hoeij/Novocin
Summer 2010	Hart/Hoeij/Novocin	Hart/Hoeij/Novocin

Now we'll explore the behavior (in theory and in practice) of Hart/Hoeij/Novocin.

The basic conepts

- $f \in \mathbb{Z}[x] \subset \mathbb{Z}_p[x]$
- Finding a factorization in $\mathbb{Z}_p[x]$ can be practical
- Let $f_1, \ldots, f_r$ be the factorization of f in $\mathbb{Z}_p[x]$.
- True factors $g|f$ correspond with 0–1 vectors in $\{0,1\}^r$.
- True factors have boundable coefficients $\| g \|_\infty \leq L$.
- Given a vector in $\{0,1\}^r$ we can quickly test it
- Provided we know $f_1, \ldots, f_r$ to sufficient precision ($2L$).
- A Technique called Hensel lifting can increase p-adic precision

The basic conepts

- $f \in \mathbb{Z}[x] \subset \mathbb{Z}_p[x]$
- Finding a factorization in $\mathbb{Z}_p[x]$ can be practical
- Let $f_1, \ldots, f_r$ be the factorization of f in $\mathbb{Z}_p[x]$.
- True factors $g|f$ correspond with 0–1 vectors in $\{0,1\}^r$.
- True factors have boundable coefficients $\| g \|_\infty \leq L$.
- Given a vector in $\{0,1\}^r$ we can quickly test it
- Provided we know $f_1, \ldots, f_r$ to sufficient precision ($2L$).
- A Technique called Hensel lifting can increase p-adic precision

The basic conepts

- $f \in \mathbb{Z}[x] \subset \mathbb{Z}_p[x]$
- Finding a factorization in $\mathbb{Z}_p[x]$ can be practical
- Let $f_1, \ldots, f_r$ be the factorization of f in $\mathbb{Z}_p[x]$.
- True factors $g|f$ correspond with 0–1 vectors in $\{0, 1\}^r$.
- True factors have boundable coefficients $\| g \|_\infty \leq L$.
- Given a vector in $\{0, 1\}^r$ we can quickly test it
- Provided we know $f_1, \ldots, f_r$ to sufficient precision ($2L$).
- A Technique called Hensel lifting can increase p-adic precision

The basic conepts

- $f \in \mathbb{Z}[x] \subset \mathbb{Z}_p[x]$
- Finding a factorization in $\mathbb{Z}_p[x]$ can be practical
- Let $f_1, \ldots, f_r$ be the factorization of f in $\mathbb{Z}_p[x]$.
- True factors $g|f$ correspond with 0–1 vectors in $\{0,1\}^r$.
- True factors have boundable coefficients $\| g \|_\infty \leq L$.
- Given a vector in $\{0,1\}^r$ we can quickly test it
- Provided we know $f_1, \ldots, f_r$ to sufficient precision ($2L$).
- A Technique called Hensel lifting can increase p-adic precision

The basic conepts

- $f \in \mathbb{Z}[x] \subset \mathbb{Z}_p[x]$
- Finding a factorization in $\mathbb{Z}_p[x]$ can be practical
- Let $f_1, \ldots, f_r$ be the factorization of f in $\mathbb{Z}_p[x]$.
- True factors $g|f$ correspond with 0–1 vectors in $\{0,1\}^r$.
- True factors have boundable coefficients $\| g \|_\infty \leq L$.
- Given a vector in $\{0,1\}^r$ we can quickly test it
- Provided we know $f_1, \ldots, f_r$ to sufficient precision ($2L$).
- A Technique called Hensel lifting can increase p-adic precision

The basic conepts

- $f \in \mathbb{Z}[x] \subset \mathbb{Z}_p[x]$
- Finding a factorization in $\mathbb{Z}_p[x]$ can be practical
- Let $f_1, \ldots, f_r$ be the factorization of f in $\mathbb{Z}_p[x]$.
- True factors $g|f$ correspond with 0–1 vectors in $\{0,1\}^r$.
- True factors have boundable coefficients $\parallel g \parallel_\infty \leq L$.
- Given a vector in $\{0,1\}^r$ we can quickly test it
- Provided we know $f_1, \ldots, f_r$ to sufficient precision ($2L$).
- A Technique called Hensel lifting can increase p-adic precision

The basic conepts

- $f \in \mathbb{Z}[x] \subset \mathbb{Z}_p[x]$
- Finding a factorization in $\mathbb{Z}_p[x]$ can be practical
- Let $f_1, \ldots, f_r$ be the factorization of f in $\mathbb{Z}_p[x]$.
- True factors $g|f$ correspond with 0–1 vectors in $\{0, 1\}^r$.
- True factors have boundable coefficients $\| g \|_\infty \leq L$.
- Given a vector in $\{0, 1\}^r$ we can quickly test it
- Provided we know $f_1, \ldots, f_r$ to sufficient precision ($2L$).
- A Technique called Hensel lifting can increase p-adic precision

An Example

- Let $f = x^4 - 11$ and $p = 5$.
- A bound (Landau-Mignotte) on the coefficients of any factors of f is 12.05.
- $f = x^4 - 11 \equiv (x+1)(x+2)(x+3)(x+4) \bmod 5$
- Using Hensel Lifting we find $f \equiv (x+16)(x+12)(x+13)(x+9) \equiv (x-9)(x+12)(x-12)(x+9) \bmod 5^2$.
- Could brute force combinations, such as: $(x-9)\cdot(x+9) \equiv x^2 - 6 \bmod 25$, but the GCD of $x^2 - 6$ and f performed in $\mathbb{Z}[x]$ is 1.
- After testing certain combinations we would determine that f is irreducible.

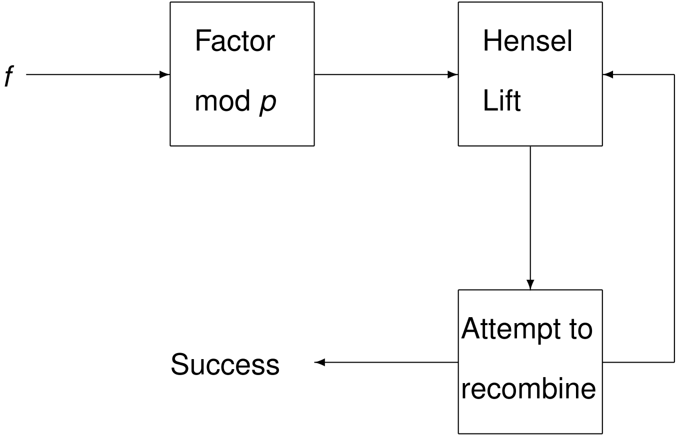
Behavior/Cost of the parts

- Let f have degree N and $\| f \|_\infty \leq 2^H$.
- Factoring modulo p costs $\mathcal{O}(N^2 + N \log p)$ CPU ops
- Hensel Lifting to precision a is $\mathcal{O}(\mathcal{M}(N)\mathcal{M}(a \cdot \log p) \cdot \log r)$.
- Checking a 0–1 vector is cheap, in worst-case, $\mathcal{O}(N^2 + NH)$.
- There are 2^r such 0–1 vectors.
- We call the process of finding 0–1 vectors, recombination.
- Zassenhaus uses brute force, modern approaches (since van Hoeij) use LLL.
- The behavior of LLL and recombination is practical but mysterious.

Our first philosophical dilemma

- Treat ‘modern’ recombination as a black box.
- It accepts a p-adic factorization of f with precision a.
- It returns either the complete factorization of f over $\mathbb{Z}$ or it gives up.
- There are some (obscure) worst-case polynomials when recombination dominates Hensel lifting (as it does in theory).
- The average polynomials tend to be dominated by Hensel lifting in practice.
- So what should we use for the first precision?
- Aim too low then recombination might fail.
- Aim too high Hensel lifting could dominate running times.

The Hensel Picture



Before opening the box

Practical Design Goal

In practice we would like to always minimize the cost of Hensel lifting.

Theoretical Design Goal

We must show that, in the worst-case, any failed attempts do not impact the complexity bound.

Balanced Design Goal

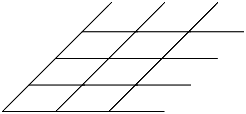
We show that, in the worst-cases, failed attempts do not impact the running times.

Opening the recombination box

At the heart of our modern recombination technique is an application of the LLL algorithm.
LLL is a somewhat mysterious algorithm with many useful applications (cryptography, number theory, integer programming, Diophantine approximations, relation finding, Table Maker's Dilemma).

Introducing Lattices

A lattice, L



The same lattice, L

Definition

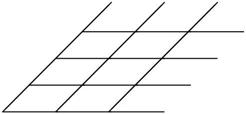
A lattice, L, is the set of all integer combinations of some set of vectors in $\mathbb{R}^n$

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

Every lattice has many bases. . . and LLL wants to find a good basis!

Introducing Lattices

A lattice, L



The same lattice, L

Definition

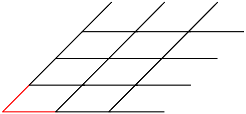
A lattice, L, is the set of all integer combinations of some set of vectors in $\mathbb{R}^n$

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

Every lattice has many bases... and LLL wants to find a good basis!

Introducing Lattices

A lattice, L



The same lattice, L

Definition

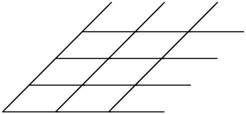
A lattice, L, is the set of all integer combinations of some set of vectors in $\mathbb{R}^n$

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

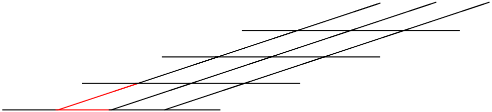
Every lattice has many bases… and LLL wants to find a good basis!

Introducing Lattices

A lattice, L



The same lattice, L



Definition

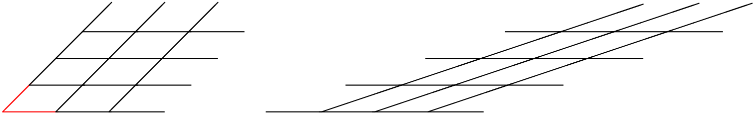
A lattice, L, is the set of all integer combinations of some set of vectors in $\mathbb{R}^n$

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

Every lattice has many bases. . . and LLL wants 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 LLL wants 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 ($\approx$ very difficult / not polynomial time) to solve.
- The are many interesting research areas which can be connected to the SVP.
- One of the primary uses of a 'good basis' is to approximately solve the SVP in polynomial time.
- 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 ($\approx$ very difficult / not polynomial time) to solve.
- The are many interesting research areas which can be connected to the SVP.
- One of the primary uses of a 'good basis' is to approximately solve the SVP in polynomial time.
- 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 ($\approx$ very difficult / not polynomial time) to solve.
- The are many interesting research areas which can be connected to the SVP.
- One of the primary uses of a 'good basis' is to approximately solve the SVP in polynomial time.
- 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 ($\approx$ very difficult / not polynomial time) to solve.
- The are many interesting research areas which can be connected to the SVP.
- One of the primary uses of a 'good basis' is to approximately solve the SVP in polynomial time.
- 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 ($\approx$ very difficult / not polynomial time) to solve.
- The are many interesting research areas which can be connected to the SVP.
- One of the primary uses of a 'good basis' is to approximately solve the SVP in polynomial time.
- Sometimes approximating can be enough.

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.

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.

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:

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

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 $\mathbb{R}$.

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:

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

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 $\mathbb{R}$.

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:

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

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 $\mathbb{R}$.

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:

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

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 $\mathbb{R}$.

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 **LLL-reduced** when:

$$\| b_i^* \|^2 \leq 2 \| b_{i+1}^* \|^2 \ \ \forall i < d$$

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

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 **LLL-reduced** when:

$$\| b_i^* \|^2 \leq 2 \| b_{i+1}^* \|^2 \quad \forall i < d$$

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

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 **LLL-reduced** when:

$$\| b_i^* \|^2 \leq 2 \| b_{i+1}^* \|^2 \quad \forall i < d$$

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

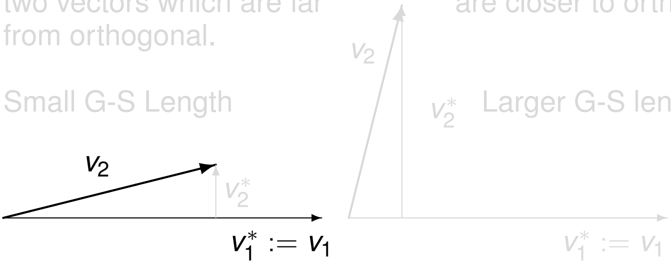
Gram-Schmidt Length versus Orthogonality

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

In this one the vectors are closer to orthogonal.

Small G-S Length

Larger G-S length

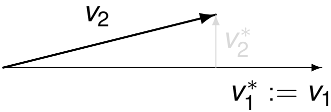


- LLL searches for a nearly orthogonal basis.
- It does this by ‘rearranging’ basis vectors such that later vectors have longer G-S lengths and ’modding out’ by previous vectors over $\mathbb{Z}$.

Gram-Schmidt Length versus Orthogonality

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

In this one the vectors are closer to orthogonal.



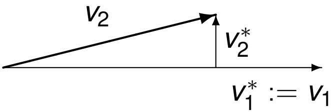
- LLL searches for a nearly orthogonal basis.
- It does this by 'rearranging' basis vectors such that later vectors have longer G-S lengths and 'modding out' by previous vectors over $\mathbb{Z}$.

Gram-Schmidt Length versus Orthogonality

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

In this one the vectors are closer to orthogonal.

Small G-S Length



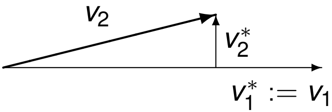
Larger G-S length

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

Gram-Schmidt Length versus Orthogonality

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.

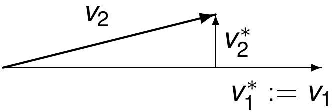
Larger G-S length

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

Gram-Schmidt Length versus Orthogonality

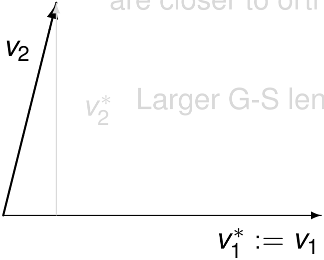
In this picture there are two vectors which are far from orthogonal.

Small G-S Length



In this one the vectors are closer to orthogonal.

Larger G-S length

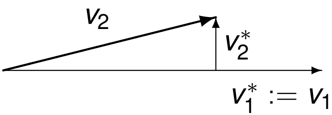


- LLL searches for a nearly orthogonal basis.
- It does this by 'rearranging' basis vectors such that later vectors have longer G-S lengths and 'modding out' by previous vectors over $\mathbb{Z}$.

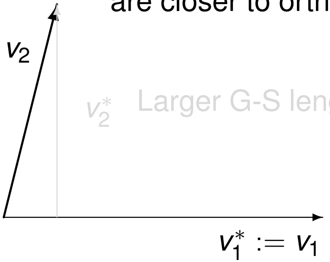
Gram-Schmidt Length versus Orthogonality

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.



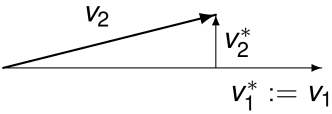
Larger G-S length

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

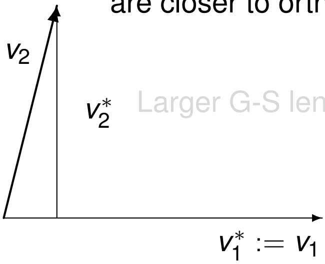
Gram-Schmidt Length versus Orthogonality

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.



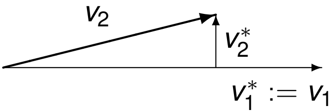
Larger G-S length

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

Gram-Schmidt Length versus Orthogonality

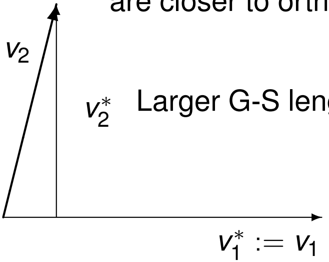
In this picture there are two vectors which are far from orthogonal.

Small G-S Length



In this one the vectors are closer to orthogonal.

Larger G-S length

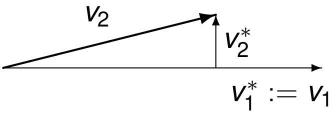


- LLL searches for a nearly orthogonal basis.
- It does this by 'rearranging' basis vectors such that later vectors have longer G-S lengths and 'modding out' by previous vectors over $\mathbb{Z}$.

Gram-Schmidt Length versus Orthogonality

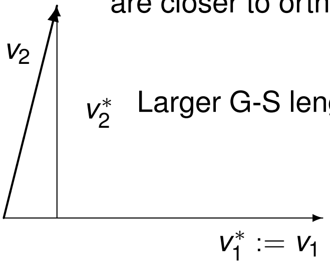
In this picture there are two vectors which are far from orthogonal.

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 later vectors have longer G-S lengths and ’modding out’ by previous vectors over $\mathbb{Z}$.

Properties of a reduced 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 $v \in \mathrm{SPAN}_{\mathbb{Z}}(b_1, \ldots, b_{d-1})$.

- 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.

Properties of a reduced 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 $v \in \mathrm{SPAN}_{\mathbb{Z}}(b_1, \ldots, b_{d-1})$.

- 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.

Properties of a reduced 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 $v \in \mathrm{SPAN}_{\mathbb{Z}}(b_1, \ldots, b_{d-1})$.
- 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 Rough Sketch of LLL

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. In fpLLL this step is also known as the Babbai step.
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 CPU cost of this algorithm will be roughly:
'**the number of switches**' times '**the cost per switch**'

A Rough Sketch of LLL

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. In fpLLL this step is also known as the Babbai step.
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 CPU cost of this algorithm will be roughly:
‘**the number of switches**’ times ‘**the cost per switch**’

A Rough Sketch of LLL

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. In fpLLL this step is also known as the Babbai step.
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 CPU cost of this algorithm will be roughly:
'the number of switches' times **'the cost per switch'**

A Rough Sketch of LLL

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. In fpLLL this step is also known as the Babbai step.
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 CPU cost of this algorithm will be roughly:
'the number of switches' times **'the cost per switch'**

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

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

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

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

The Ideas of the van Hoeij Recombination:

Mark van Hoeij had the clever idea of using LLL to find these $0-1$ vectors. Here's an overview:

- Every true factor, g_j, corresponds with a 0–1 vector, w_j, with r entries. Let $\mathrm{SPAN}_{\mathbb{Z}}(w_1, \ldots, w_s) =: W \subset \mathbb{Z}^r$.
- If we know any basis for W then we can find a reduced row echelon form of the basis to find the w_j and solve the problem.
- We can create a lattice, L, which contains W. If we make sure that the vectors in W are short while the vectors in $L \setminus W$ are long, then LLL can find W.
- So we begin by taking the standard basis for $\mathbb{Z}^r$ as our basis for L. Then we will add entries which should be short for true factors and perhaps long for the others.

But what can we use for this task?

The Ideas of the van Hoeij Recombination:

Mark van Hoeij had the clever idea of using LLL to find these $0-1$ vectors. Here's an overview:

- Every true factor, g_j, corresponds with a 0–1 vector, w_j, with r entries. Let $\mathrm{SPAN}_{\mathbb{Z}}(w_1, \ldots, w_s) =: W \subset \mathbb{Z}^r$.
- If we know any basis for W then we can find a reduced row echelon form of the basis to find the w_j and solve the problem.
- We can create a lattice, L, which contains W. If we make sure that the vectors in W are short while the vectors in $L \setminus W$ are long, then LLL can find W.
- So we begin by taking the standard basis for $\mathbb{Z}^r$ as our basis for L. Then we will add entries which should be short for true factors and perhaps long for the others.

But what can we use for this task?

The Ideas of the van Hoeij Recombination:

Mark van Hoeij had the clever idea of using LLL to find these $0-1$ vectors. Here's an overview:

- Every true factor, g_j, corresponds with a 0–1 vector, w_j, with r entries. Let $\mathrm{SPAN}_{\mathbb{Z}}(w_1, \ldots, w_s) =: W \subset \mathbb{Z}^r$.
- If we know any basis for W then we can find a reduced row echelon form of the basis to find the w_j and solve the problem.
- We can create a lattice, L, which contains W. If we make sure that the vectors in W are short while the vectors in $L \setminus W$ are long, then LLL can find W.
- So we begin by taking the standard basis for $\mathbb{Z}^r$ as our basis for L. Then we will add entries which should be short for true factors and perhaps long for the others.

But what can we use for this task?

The Ideas of the van Hoeij Recombination:

Mark van Hoeij had the clever idea of using LLL to find these $0-1$ vectors. Here's an overview:

- Every true factor, g_j, corresponds with a 0–1 vector, w_j, with r entries. Let $\mathrm{SPAN}_{\mathbb{Z}}(w_1, \ldots, w_s) =: W \subset \mathbb{Z}^r$.
- If we know any basis for W then we can find a reduced row echelon form of the basis to find the w_j and solve the problem.
- We can create a lattice, L, which contains W. If we make sure that the vectors in W are short while the vectors in $L \setminus W$ are long, then LLL can find W.
- So we begin by taking the standard basis for $\mathbb{Z}^r$ as our basis for L. Then we will add entries which should be short for true factors and perhaps long for the others.

But what can we use for this task?

The Ideas of the van Hoeij Recombination:

Mark van Hoeij had the clever idea of using LLL to find these $0-1$ vectors. Here's an overview:

- Every true factor, g_j, corresponds with a 0–1 vector, w_j, with r entries. Let $\mathrm{SPAN}_{\mathbb{Z}}(w_1, \ldots, w_s) =: W \subset \mathbb{Z}^r$.
- If we know any basis for W then we can find a reduced row echelon form of the basis to find the w_j and solve the problem.
- We can create a lattice, L, which contains W. If we make sure that the vectors in W are short while the vectors in $L \setminus W$ are long, then LLL can find W.
- So we begin by taking the standard basis for $\mathbb{Z}^r$ as our basis for L. Then we will add entries which should be short for true factors and perhaps long for the others.

But what can we use for this task?

The van Hoeij Approach

The i^{th} Trace of a polynomial

We define the i^{th} trace of g:

$$\mathrm{Tr}_i(g) := \sum_{j=1}^{N} \alpha_j^i$$

where α_j are the roots of g.
So if g_1, g_2 are polynomials then
$\mathrm{Tr}_1(g_1 \cdot g_2) = \mathrm{Tr}_1(g_1) + \mathrm{Tr}_1(g_2)$.
Also it is a fact that $\mathrm{Tr}_i(g)$ is always in the coefficient ring of g.

This works well because:

- The Trace is additive.
- The Trace of a polynomial in $\mathbb{Z}[x]$ is boundable, while in $\mathbb{Z}_p$ this can be arbitrarily 'large'.

The van Hoeij Approach

The i^{th} Trace of a polynomial

We define the i^{th} trace of g:

$$\mathrm{Tr}_i(g) := \sum_{j=1}^{N} \alpha_j^i$$

where α_j are the roots of g.
So if g_1, g_2 are polynomials then
$\mathrm{Tr}_1(g_1 \cdot g_2) = \mathrm{Tr}_1(g_1) + \mathrm{Tr}_1(g_2)$.
Also it is a fact that $\mathrm{Tr}_i(g)$ is always in the coefficient ring of g.

This works well because:

- The Trace is additive.
- The Trace of a polynomial in $\mathbb{Z}[x]$ is boundable, while in $\mathbb{Z}_p$ this can be arbitrarily 'large'.

The same example but with van Hoeij

- Let's show van Hoeij's approach on the previous example: $f = x^4 - 11$.
- $f \equiv (x - 41)(x + 41)(x - 38)(x + 38) \bmod 125$
- The absolute value of any root of f cannot exceed $\sqrt[4]{11} \approx 1.82116$.
- So $|\mathrm{Tr}_1(g_k)| \leq 4 \cdot 1.82116 \approx 7.2846$ and $|\mathrm{Tr}_2(g_k)| \leq 4 \cdot 1.82116^2 \approx 13.266$.
- Now find the Traces for our local factors: $\mathrm{Tr}_1(f_1) = 41, \mathrm{Tr}_1(f_2) = -41, \mathrm{Tr}_1(f_3) = 38, \mathrm{Tr}_1(f_4) = -38$. While $\mathrm{Tr}_2(f_1) = \mathrm{Tr}_2(f_2) = 56$ and $\mathrm{Tr}_2(f_3) = \mathrm{Tr}_2(f_4) = -56$.

Example Continued; $f \equiv (x-41)(x+41)(x-38)(x+38)$

Because of space we will show both Tr_1 and Tr_2 in our lattice, although in practice the second trace would not be added until after the first LLL run.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 41/7.2846 & 56/13.266 \\ 0 & 1 & 0 & 0 & -41/7.2846 & 56/13.266 \\ 0 & 0 & 1 & 0 & 38/7.2846 & -56/13.266 \\ 0 & 0 & 0 & 1 & -38/7.2846 & -56/13.266 \\ 0 & 0 & 0 & 0 & 125/7.2846 & 0 \\ 0 & 0 & 0 & 0 & 0 & 125/13.266 \end{pmatrix}$$

$(1,1,0,0,0)$ and $(0,0,1,1,0)$ are small vectors, but trial division will show that they do not correspond with true factors. Whereas using both traces will show $(1,1,1,1,0,0)$ will be the smallest vector in this lattice.

Example Continued; $f \equiv (x-41)(x+41)(x-38)(x+38)$

Because of space we will show both Tr_1 and Tr_2 in our lattice, although in practice the second trace would not be added until after the first LLL run.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 41/7.2846 & 56/13.266 \\ 0 & 1 & 0 & 0 & -41/7.2846 & 56/13.266 \\ 0 & 0 & 1 & 0 & 38/7.2846 & -56/13.266 \\ 0 & 0 & 0 & 1 & -38/7.2846 & -56/13.266 \\ 0 & 0 & 0 & 0 & 125/7.2846 & 0 \\ 0 & 0 & 0 & 0 & 0 & 125/13.266 \end{pmatrix}$$

$(1,1,0,0,0)$ and $(0,0,1,1,0)$ are small vectors, but trial division will show that they do not correspond with true factors. Whereas using both traces will show $(1,1,1,1,0,0)$ will be the smallest vector in this lattice.

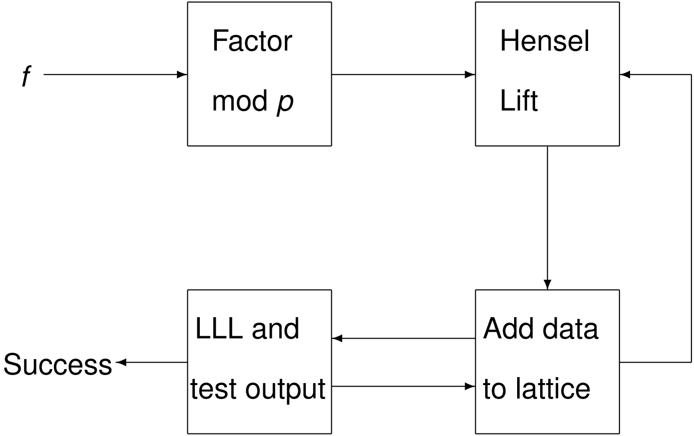
Example Continued; $f \equiv (x-41)(x+41)(x-38)(x+38)$

Because of space we will show both Tr_1 and Tr_2 in our lattice, although in practice the second trace would not be added until after the first LLL run.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 41/7.2846 & 56/13.266 \\ 0 & 1 & 0 & 0 & -41/7.2846 & 56/13.266 \\ 0 & 0 & 1 & 0 & 38/7.2846 & -56/13.266 \\ 0 & 0 & 0 & 1 & -38/7.2846 & -56/13.266 \\ 0 & 0 & 0 & 0 & 125/7.2846 & 0 \\ 0 & 0 & 0 & 0 & 0 & 125/13.266 \end{pmatrix}$$

$(1,1,0,0,0)$ and $(0,0,1,1,0)$ are small vectors, but trial division will show that they do not correspond with true factors. Whereas using both traces will show $(1,1,1,1,0,0)$ will be the smallest vector in this lattice.

Amortizing Costs



Gradual reduction

In a LATIN2010 we have analysed the complexity of a useful LLL technique.

B-reduction

We call a basis, $b_1, \ldots, b_s$ a **B-reduced basis** if:

- $b_1, \ldots, b_s$ form a reduced basis.
- $\| b_s^* \|^2 \leq B$.

B-reducing the following $r \times r$ matrix uses less than $\mathcal{O}(r^2(r+B))$ LLL switches. No matter how large the entries are!

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

Gradual reduction

In a LATIN2010 we have analysed the complexity of a useful LLL technique.

B-reduction

We call a basis, $b_1, \ldots, b_s$ a **B-reduced basis** if:

- $b_1, \ldots, b_s$ form a reduced basis.
- $\| b_s^* \|^2 \leq B$.

B-reducing the following $r \times r$ matrix uses less than $\mathcal{O}(r^2(r+B))$ LLL switches. No matter how large the entries are!

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

B-reduction, cont.

Gradual B-reduce:

1. Scale down the last entries by 2^{kr} so that all final entries have absolute value $\leq 2^r$.
2. Run LLL.
3. Throw out the final vectors with G-S length $> B$.
4. Scale the last entries back up by 2^r, return to step 2.

- This approach uses many calls to LLL, but the entries will always have a bounded size.
- We can bound the total number of switches.
- When the input is a van Hoeij matrix with one trace we can use $B = r + 1$.
- This allows us to bound LLL switches $\mathcal{O}(r^3)$ instead of $\mathcal{O}(r^2 H)$

B-reduction, cont.

Gradual B-reduce:

1. Scale down the last entries by 2^{kr} so that all final entries have absolute value $\leq 2^r$.
2. Run LLL.
3. Throw out the final vectors with G-S length $> B$.
4. Scale the last entries back up by 2^r, return to step 2.

- This approach uses many calls to LLL, but the entries will always have a bounded size.
- We can bound the total number of switches.
- When the input is a van Hoeij matrix with one trace we can use $B = r + 1$.
- This allows us to bound LLL switches $\mathcal{O}(r^3)$ instead of $\mathcal{O}(r^2 H)$

B-reduction, cont.

Gradual B-reduce:

1. Scale down the last entries by 2^{kr} so that all final entries have absolute value $\leq 2^r$.
2. Run LLL.
3. Throw out the final vectors with G-S length $> B$.
4. Scale the last entries back up by 2^r, return to step 2.

- This approach uses many calls to LLL, but the entries will always have a bounded size.
- We can bound the total number of switches.
- When the input is a van Hoeij matrix with one trace we can use $B = r + 1$.
- This allows us to bound LLL switches $\mathcal{O}(r^3)$ instead of $\mathcal{O}(r^2 H)$

B-reduction, cont.

Gradual B-reduce:

1. Scale down the last entries by 2^{kr} so that all final entries have absolute value $\leq 2^r$.
2. Run LLL.
3. Throw out the final vectors with G-S length $> B$.
4. Scale the last entries back up by 2^r, return to step 2.

- This approach uses many calls to LLL, but the entries will always have a bounded size.
- We can bound the total number of switches.
- When the input is a van Hoeij matrix with one trace we can use $B = r + 1$.
- This allows us to bound LLL switches $\mathcal{O}(r^3)$ instead of $\mathcal{O}(r^2 H)$

B-reduction, cont.

Gradual B-reduce:

1. Scale down the last entries by 2^{kr} so that all final entries have absolute value $\leq 2^r$.
2. Run LLL.
3. Throw out the final vectors with G-S length $> B$.
4. Scale the last entries back up by 2^r, return to step 2.

- This approach uses many calls to LLL, but the entries will always have a bounded size.
- We can bound the total number of switches.
- When the input is a van Hoeij matrix with one trace we can use $B = r + 1$.
- This allows us to bound LLL switches $\mathcal{O}(r^3)$ instead of $\mathcal{O}(r^2 H)$

An Example

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

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

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

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

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

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

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

A single call to LLL uses 24 swaps.

An Example

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

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

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

A single call to LLL uses 24 swaps.

An Example

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

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

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

A single call to LLL uses 24 swaps.

The LATIN2010 algorithm for factoring

This method generalizes to the following type of matrix:

$$\begin{pmatrix} & & & & & D_N \\ & & & & \iddots & \\ & & & D_1 & & \\ 1 & & & * & \cdots & * \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & * & \cdots & * \end{pmatrix}$$

- A matrix of this particular form provably solves factorization. Uses an absurd amount of Hensel lifting.
- Now we can factor polynomials using $\mathcal{O}(Nr^2)$ LLL switches.
- The original LLL paper used $\mathcal{O}(N^2(N+H))$ switches.
- This gave a new complexity for factoring. Still not with a very practical algorithm (because of the absurd amount of Hensel Lifting).

The LATIN2010 algorithm for factoring

This method generalizes to the following type of matrix:

$$
\begin{pmatrix}
 & & & & & D_N \\
 & & & & \iddots & \\
 & & & D_1 & & \\
1 & & & * & \cdots & * \\
 & \ddots & & \vdots & \ddots & \vdots \\
 & & 1 & * & \cdots & *
\end{pmatrix}
$$

- A matrix of this particular form provably solves factorization. Uses an absurd amount of Hensel lifting.
- Now we can factor polynomials using $\mathcal{O}(Nr^2)$ LLL switches.
- The original LLL paper used $\mathcal{O}(N^2(N+H))$ switches.
- This gave a new complexity for factoring. Still not with a very practical algorithm (because of the absurd amount of Hensel Lifting).

The LATIN2010 algorithm for factoring

This method generalizes to the following type of matrix:

$$\begin{pmatrix} & & & & & D_N \\ & & & & \iddots & \\ & & & D_1 & & \\ 1 & & & * & \cdots & * \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & * & \cdots & * \end{pmatrix}$$

- A matrix of this particular form provably solves factorization. Uses an absurd amount of Hensel lifting.
- Now we can factor polynomials using $\mathcal{O}(Nr^2)$ LLL switches.
- The original LLL paper used $\mathcal{O}(N^2(N+H))$ switches.
- This gave a new complexity for factoring. Still not with a very practical algorithm (because of the absurd amount of Hensel Lifting).

The LATIN2010 algorithm for factoring

This method generalizes to the following type of matrix:

$$\begin{pmatrix} & & & & & D_N \\ & & & & \iddots & \\ & & & D_1 & & \\ 1 & & & * & \cdots & * \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & * & \cdots & * \end{pmatrix}$$

- A matrix of this particular form provably solves factorization. Uses an absurd amount of Hensel lifting.
- Now we can factor polynomials using $\mathcal{O}(Nr^2)$ LLL switches.
- The original LLL paper used $\mathcal{O}(N^2(N+H))$ switches.
- This gave a new complexity for factoring. Still not with a very practical algorithm (because of the absurd amount of Hensel Lifting).

Being Practical

My goal

To get the best complexity for the most practical algorithm we must show that early attempts at recombination do not hurt the complexity.

The tools:

- LLL never alters AD$:= \prod \parallel b_i^* \parallel$
- We are searching for very small vectors
- A B-reduced basis must have vectors with bounded norm
- We can control the size of our new data (with gradual feeding)
- This allows us to know that progress is being made with each LLL call.

Nearing the end

LLL costs

We can now prove a complexity of $\mathcal{O}(r^7)$ for the total cost of LLL. (Using fast arithmetic this is $\mathcal{O}(r^6 \log r)$).

Other costs

The cost of Hensel lifting can be bounded* by $\mathcal{O}(N^4 \cdot (N+H)^2)$. (Using fast arithmetic this drops to $\mathcal{O}(N^2(N+H)))$. We have introduced some matrix multiplications with the new technique adding a cost of $\mathcal{O}(r^3 N^2(N+H))$ (fast: $\mathcal{O}(r^2 N^2(N+H)))$.

Showing Practicality

We must show that the algorithm is practical. This can only be done with an implementation of the algorithm as it was proved.

Some timings

Poly	r	NTL	H-bnd	FLINT	H-bnd
P1	60	.248	29^{311}	.136	89^{33}
P2	20	.376	11^{437}	.144	$11^{22/44}$
P3	28	1.036	11^{629}	.320	$11^{31/62}$
P4	42	1.956	13^{745}	1.452	$7^{80/160}$
P5	32	.088	19^{51}	.036	23^{26}
P6	48	.276	19^{152}	.160	$23^{38/76}$
P7	76	1.136	37^{78}	.900	19^{74}
P8	54	3.428	13^{324}	1.700	11^{84}
M12_5	72	12.429	13^{1171}	4.156	11^{180}
M12_6	84	21.697	13^{1555}	7.780	$13^{190/380}$
S7	64	.340	29^{78}	.336	47^{41}
T1	30	3.848	7^{495}	1.180	7^{40}
T2	32	3.18	7^{200}	1.216	7^{43}

Thank You

Thank you for your time!