The Generalized Sieve Kernel

The Algorithmic Ant and the Sandpile

Léo Ducas[1]

Based on joint work in progress with

M. Albrecht, E. Postlethwaite,

G. Herold, E. Kirshanova, M. Stevens

Cryptology Group, CWI, Amsterdam, The Netherlands

Lattice Coding Crypto Meeting

London, Sept 2018

[1]Supported by a Veni Innovational Research Grant from NWO (639.021.645).

The Algorithmic Ant and the Sandpile

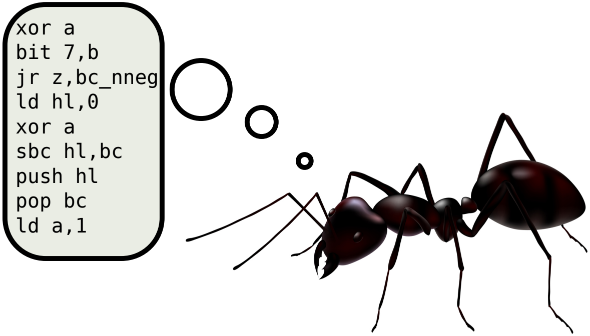
Once upon a time . . .

Once upon a time . . .

. . . there was an ant.

Once upon a time . . .

. . . there was an ant.



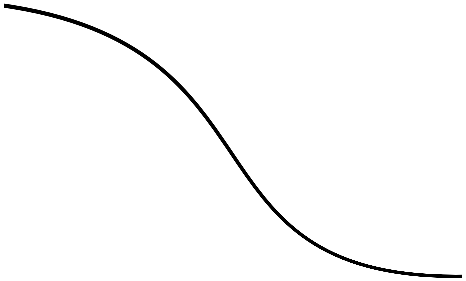
```
xor a
bit 7,b
jr z,bc_nneg
ld hl,0
xor a
sbc hl,bc
push hl
pop bc
ld a,1
```

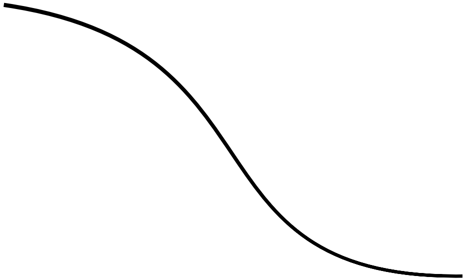
An algorithmic ant.

The Queen of ant ant,

The Queen of ant ant,

"See this sand pile."





"I want it *flat* !"

Looking clo

the algorithmic ant ponders.

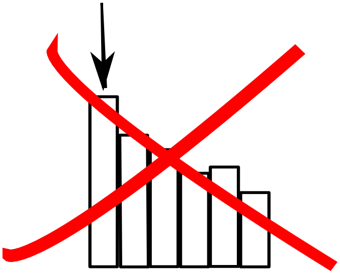
"One grain at the time,
I shall pull the sand downhill."

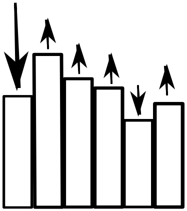
"One grain at the time,
I shall pull the sand downhill."

So ho

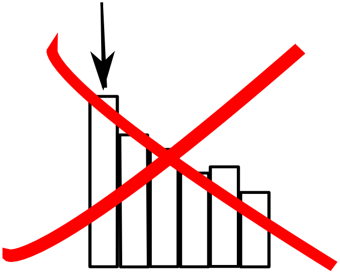
But the sandpile is whimsical,
each excavation is a puzzle of it

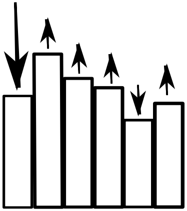
But the sandpile is whimsical, each excavation is a puzzle of it





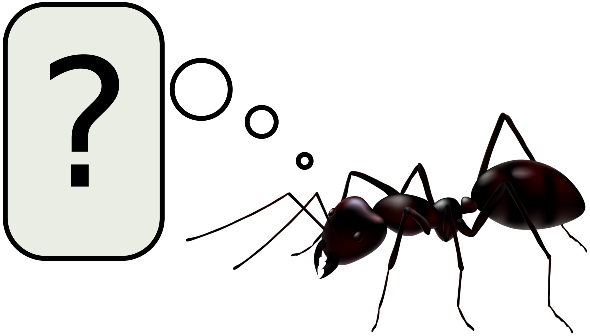
But the sandpile is whimsical,
each excavation is a puzzle of it



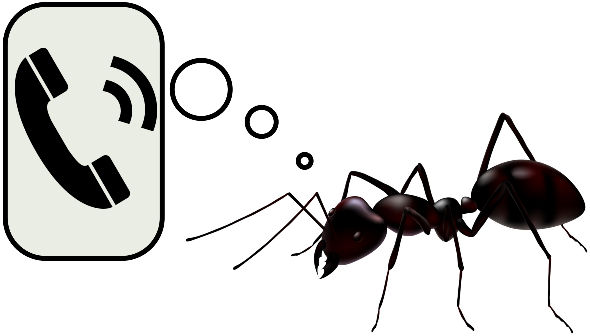


Columns are tied in my
to push one down, one must find
the right combination.

Unsure how to proceed,



Unsure how to proceed,



The ant call

"Let's break this apart."

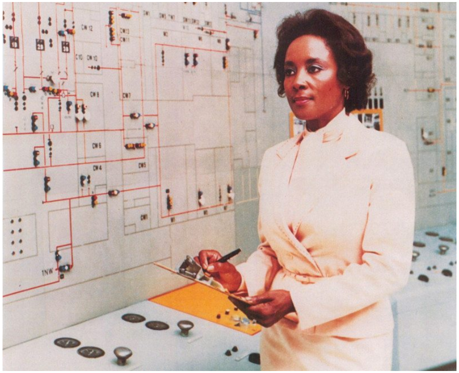
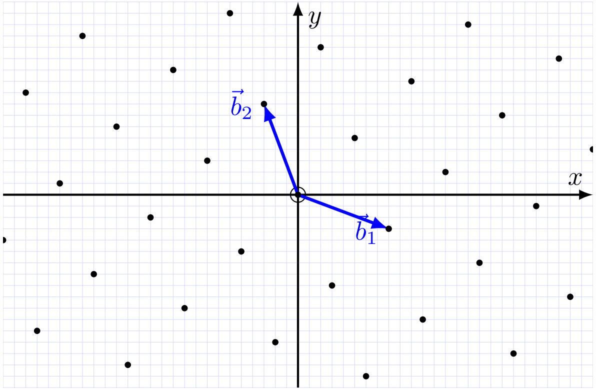


Figure: Annie Easley (NASA / NACA)

1. From Lattices to Sandpiles
2. Finding a grain of sand: Progress on SVP from Sieving
3. Flattening the Pile: Progress on lattice reduction from Sieving

From Lattices to Sandpiles

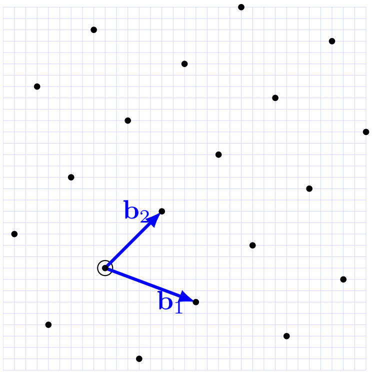
Lattices!



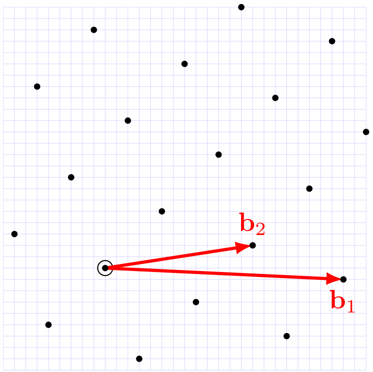
Definition

A lattice L is a discrete subgroup of a finite-dimensional Euclidean vector space.

Bases of a Lattice



Good Basis $\mathbf{G}$ of L



Bad Basis $\mathbf{B}$ of L

G $\rightarrow$ **B** : easy (randomization);
B $\rightarrow$ **G** : hard (LLL, BKZ, Lattice Sieve...).

An important invariant: the Volume

For any two bases $\mathbf{G}, \mathbf{B}$ of the same lattice Λ:

$$\det(\mathbf{G}\mathbf{G}^t) = \det(\mathbf{B}\mathbf{B}^t).$$

We can therefore define:

$$\mathrm{vol}(\Lambda) = \sqrt{\det(\mathbf{G}\mathbf{G}^t)}.$$

Geometrically: the volume of any **fundamental domain of** Λ.

Let $\mathbf{G}^\star$ be the Gram-Schmidt Orthogonalization of $\mathbf{G}$

$\mathbf{G}^\star$ is **not** a basis of Λ, nevertheless:

$$\mathrm{vol}(\Lambda) = \sqrt{\det(\mathbf{G}^\star \mathbf{G}^{\star t})} = \prod \|\mathbf{g}_i^\star\|.$$

An important invariant: the Volume

For any two bases $\mathbf{G}, \mathbf{B}$ of the same lattice Λ:

$$\det(\mathbf{G}\mathbf{G}^t) = \det(\mathbf{B}\mathbf{B}^t).$$

We can therefore define:

$$\mathrm{vol}(\Lambda) = \sqrt{\det(\mathbf{G}\mathbf{G}^t)}.$$

Geometrically: the volume of any **fundamental domain of** Λ.

Let $\mathbf{G}^\star$ be the Gram-Schmidt Orthogonalization of $\mathbf{G}$

$\mathbf{G}^\star$ is **not** a basis of Λ, nevertheless:

$$\mathrm{vol}(\Lambda) = \sqrt{\det(\mathbf{G}^\star\mathbf{G}^{\star t})} = \prod \|\mathbf{g}_i^\star\|.$$

What is a "Good" basis

Recall that, independently of the basis $\mathbf{G}$ it holds that:

$$\mathrm{vol}(\Lambda) = \prod \|\mathbf{g}_i^\star\|.$$

Therefore, it is somehow equivalent that

- $\max_i \|\mathbf{g}_i^\star\|$ is small
- $\min_i \|\mathbf{g}_i^\star\|$ is large
- $\kappa(\mathbf{G}) = \max_i \|\mathbf{g}_i^\star\| / \min_i \|\mathbf{g}_i^\star\|$ is small

Good basis

$$\max \|\mathbf{b}_i^*\| \approx \min \|\mathbf{b}_i^*\|$$

Bad basis

$$\max \|\mathbf{b}_i^*\| \gg \min \|\mathbf{b}_i^*\|$$

What is a “Good” basis

Recall that, independently of the basis $\mathbf{G}$ it holds that:

$$\mathrm{vol}(\Lambda) = \prod \|\mathbf{g}_i^\star\|.$$

Therefore, it is somehow equivalent that

- $\max_i \|\mathbf{g}_i^\star\|$ is small
- $\min_i \|\mathbf{g}_i^\star\|$ is large
- $\kappa(\mathbf{G}) = \max_i \|\mathbf{g}_i^\star\| / \min_i \|\mathbf{g}_i^\star\|$ is small

Good basis

$$\max \|\mathbf{b}_i^*\| \approx \min \|\mathbf{b}_i^*\|$$

Bad basis

$$\max \|\mathbf{b}_i^*\| \gg \min \|\mathbf{b}_i^*\|$$

What is a "Good" basis

Recall that, independently of the basis **G** it holds that:

$$\mathrm{vol}(\Lambda) = \prod \|\mathbf{g}_i^\star\|.$$

Therefore, it is somehow equivalent that

- $\max_i \|\mathbf{g}_i^\star\|$ is small
- $\min_i \|\mathbf{g}_i^\star\|$ is large
- $\kappa(\mathbf{G}) = \max_i \|\mathbf{g}_i^\star\| / \min_i \|\mathbf{g}_i^\star\|$ is small

Good basis

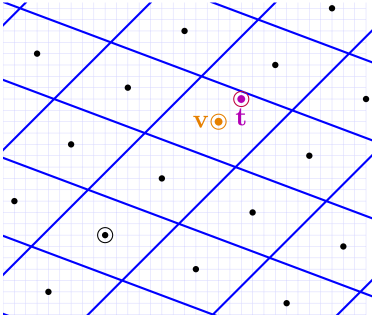
$$\max \|\mathbf{b}_i^*\| \approx \min \|\mathbf{b}_i^*\|$$

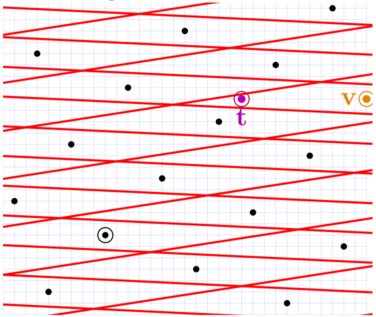
Bad basis

$$\max \|\mathbf{b}_i^*\| \gg \min \|\mathbf{b}_i^*\|$$

Bases and Fundamental Domains

Each basis defines a **parallelepipedic tiling**.



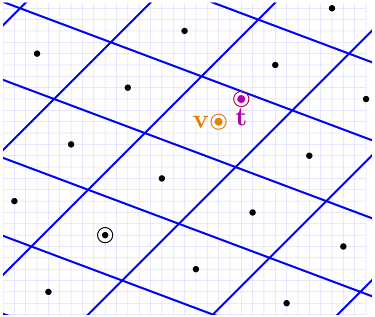


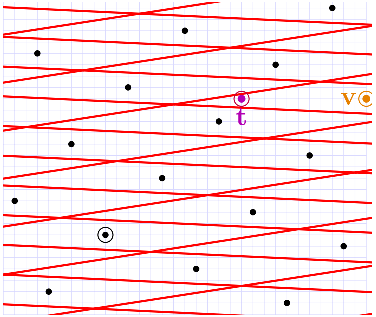
Round'off Algorithm [Lenstra, Babai]:

- Given a target $\mathbf{t}$
- Find's $\mathbf{v} \in L$ at the center the tile.

Bases and Fundamental Domains

Each basis defines a **parallelepipedic tiling**.



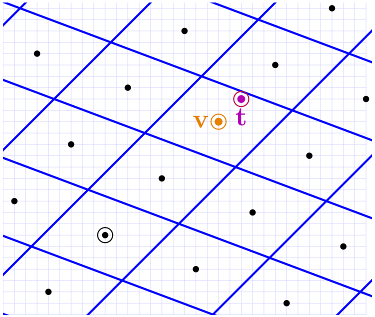


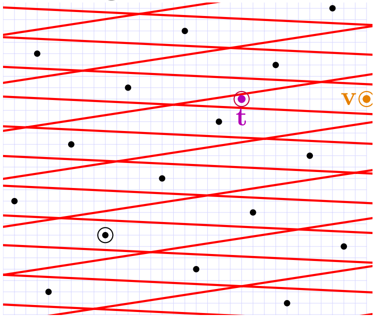
Round'off Algorithm [Lenstra, Babai]:

- Given a target **t**
- Find's **v** $\in L$ at the center the tile.

Bases and Fundamental Domains

Each basis defines a **parallelepipedic tiling**.

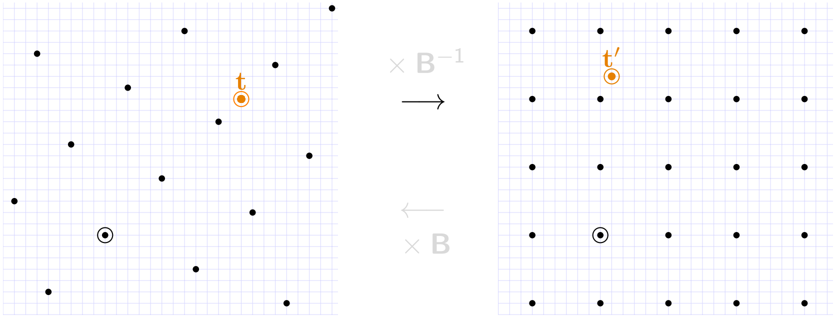




Round'off Algorithm [Lenstra, Babai]:

- Given a target **t**
- Find's **v** $\in L$ at the center the tile.

Round'off Algorithm

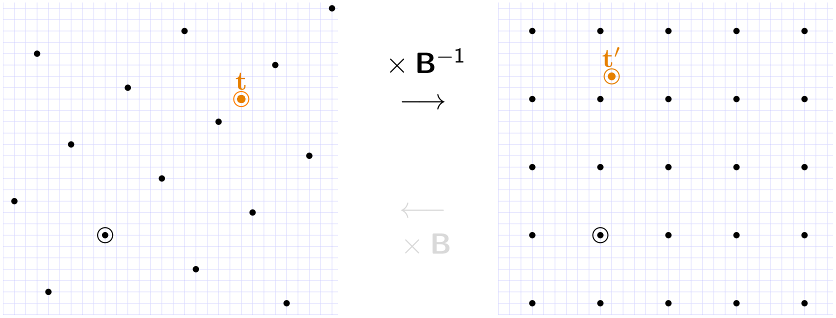


ROUNDOFF Algorithm [Lenstra,Babai]:

- Use $\mathbf{B}$ to switch to the lattice $\mathbb{Z}^n$ ($\times \mathbf{B}^{-1}$)
- round each coordinate (square tiling)
- switch back to L ($\times \mathbf{B}$)

$$\mathbf{t}' = \mathbf{B}^{-1} \cdot \mathbf{t}; \quad \mathbf{v}' = \lfloor \mathbf{t}' \rceil; \quad \mathbf{v} = \mathbf{B} \cdot \mathbf{v}'$$

Round'off Algorithm

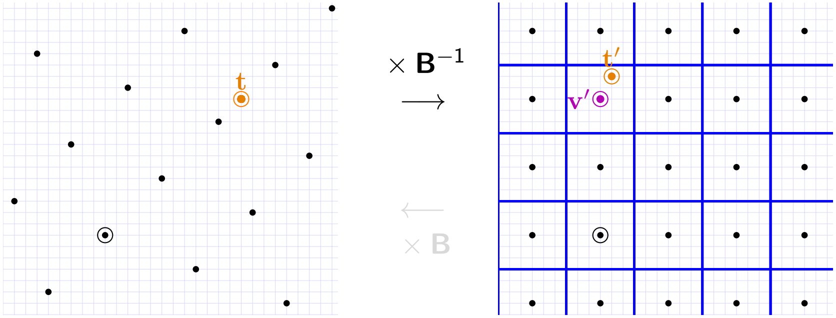


RoundOff Algorithm [Lenstra,Babai]:

- Use $\mathbf{B}$ to switch to the lattice $\mathbb{Z}^n$ $(\times \mathbf{B}^{-1})$
- round each coordinate (square tiling)
- switch back to L $(\times \mathbf{B})$

$$\mathbf{t}' = \mathbf{B}^{-1} \cdot \mathbf{t}; \quad \mathbf{v}' = \lfloor \mathbf{t}' \rceil; \quad \mathbf{v} = \mathbf{B} \cdot \mathbf{v}'$$

Round'off Algorithm

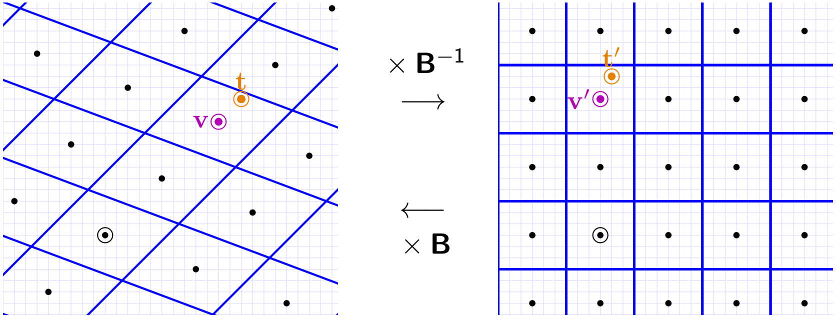


RoundOff Algorithm [Lenstra,Babai]:

- Use **B** to switch to the lattice $\mathbb{Z}^n$ ($\times \mathbf{B}^{-1}$)
- round each coordinate (square tiling)
- switch back to L ($\times \mathbf{B}$)

$$\mathbf{t}' = \mathbf{B}^{-1} \cdot \mathbf{t}; \quad \mathbf{v}' = \lfloor \mathbf{t}' \rceil; \quad \mathbf{v} = \mathbf{B} \cdot \mathbf{v}'$$

Round'off Algorithm



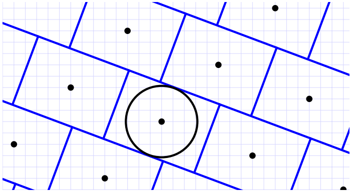
RoundOff Algorithm [Lenstra,Babai]:

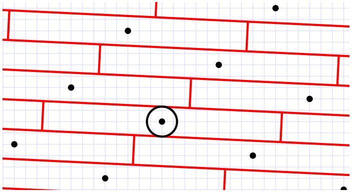
- Use **B** to switch to the lattice $\mathbb{Z}^n$ $(\times \mathbf{B}^{-1})$
- round each coordinate (square tiling)
- switch back to L $(\times \mathbf{B})$

$$\mathbf{t}' = \mathbf{B}^{-1} \cdot \mathbf{t}; \quad \mathbf{v}' = \lfloor \mathbf{t}' \rceil; \quad \mathbf{v} = \mathbf{B} \cdot \mathbf{v}'$$

Nearest-Plane Algorithm

There is a better algorithm (NEARESTPLANE) based on Gram-Schmidt Orth. $\mathbf{B}^\star$ of a basis $\mathbf{B}$:


Decoding radius with $\mathbf{G}^\star$


Decoding radius with $\mathbf{B}^\star$

- Worst-case distance: $\frac{1}{2}\sqrt{\sum \|\mathbf{b}_i^\star\|^2}$ (Approx-CVP)
- Correct decoding of $\mathbf{t} = \mathbf{v} + \mathbf{e}$ where $\mathbf{v} \in \Lambda$ if (BDD)

$$\|\mathbf{e}\| \leq \frac{1}{2} \min \|\mathbf{b}_i^\star\|$$

Profile of a Basis

Good basis

$$\max \|\mathbf{b}_i^*\| \approx \min \|\mathbf{b}_i^*\|$$

Bad basis

$$\max \|\mathbf{b}_i^*\| \gg \min \|\mathbf{b}_i^*\|$$

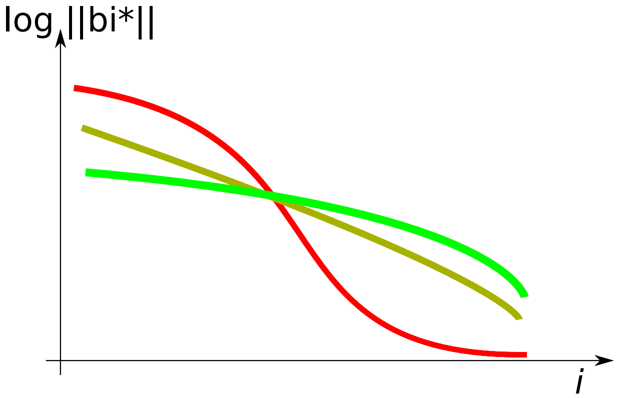
Profile of a Basis

Good basis

$$\max \|\mathbf{b}_i^*\| \approx \min \|\mathbf{b}_i^*\|$$

Bad basis

$$\max \|\mathbf{b}_i^*\| \gg \min \|\mathbf{b}_i^*\|$$



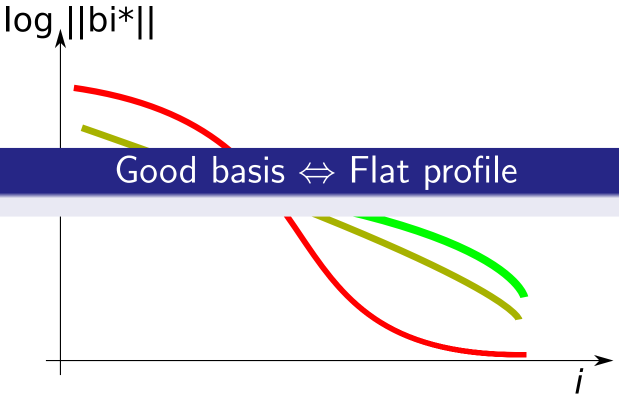
Profile of a Basis

Good basis

$$\max \|\mathbf{b}_i^*\| \approx \min \|\mathbf{b}_i^*\|$$

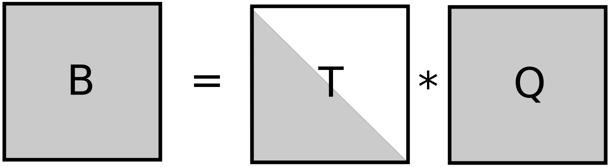
Bad basis

$$\max \|\mathbf{b}_i^*\| \gg \min \|\mathbf{b}_i^*\|$$



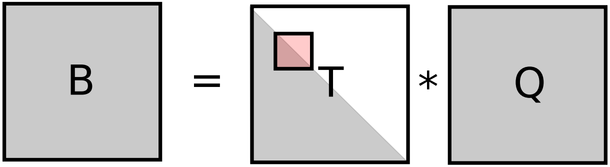
Good basis $\Leftrightarrow$ Flat profile

Local Modification



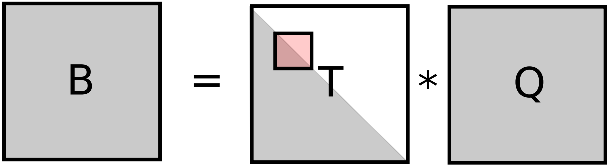
- Local blocks $[i:j]$ of T correspond to a projected sublattice $L_{[i:j]}$
- We can work locally: modify this block, affecting only $\mathbf{b}_i^* \dots \mathbf{b}_j^*$

Local Modification



- Local blocks $[i : j]$ of T correspond to a projected sublattice $L_{[i:j]}$
- We can work locally: modify this block, affecting only $\mathbf{b}_i^* \dots \mathbf{b}_j^*$

Local Modification



- Local blocks $[i:j]$ of T correspond to a projected sublattice $L_{[i:j]}$
- We can work locally: modify this block, affecting only $\mathbf{b}_i^* \ldots \mathbf{b}_j^*$

Local Improvement

- Find the shortest vector v of the projected sublattice $L_{[i:j]}$

"a puzzle of it
"the right combination."

- Construct a unimodular matrix $\mathbf{U}$ such that $\mathbf{T}_{[i:j]} \cdot \mathbf{U} = [\mathbf{v}, *, *, \dots]$. Apply $\mathbf{U}$ (locally).
- The new $\mathbf{b}_i^* = v$ got shorter!
- The other $\mathbf{b}_{i+1}^*, \dots, \mathbf{b}_j^*$ will change as well

Local Improvement

- Find the shortest vector v of the projected sublattice $L_{[i:j]}$

"a puzzle of it
"the right combination."

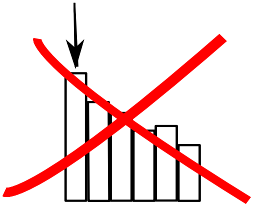
- Construct a unimodular matrix $\mathbf{U}$ such that $\mathbf{T}_{[i:j]} \cdot \mathbf{U} = [\mathbf{v}, *, *, \ldots]$. Apply $\mathbf{U}$ (locally).
- The new $\mathbf{b}_i^* = v$ got shorter!
- The other $\mathbf{b}_{i+1}^*, \ldots, \mathbf{b}_j^*$ will change as well

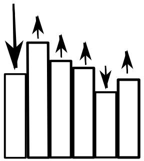
Local Improvement

- Find the shortest vector v of the projected sublattice $L_{[i:j]}$

"a puzzle of it
"the right combination."

- Construct a unimodular matrix $\mathbf{U}$ such that $\mathbf{T}_{[i:j]} \cdot \mathbf{U} = [\mathbf{v}, *, *, \ldots]$. Apply $\mathbf{U}$ (locally).
- The new $\mathbf{b}_i^* = v$ got shorter!
- The other $\mathbf{b}_{i+1}^*, \ldots, \mathbf{b}_j^*$ will change as well





Lattice reduction (e.g. BKZ-b)

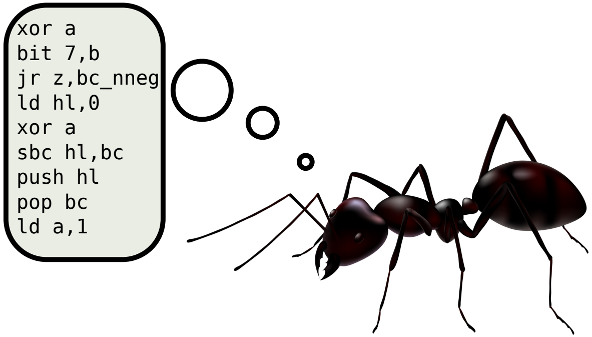
b: Blocksize
Run the local improvements for consecutive blocks:
$[1 : b], [2 : b+1], [3 : b+2], \ldots, [n-b : n], [n-b+1 : n], \ldots\ [n-1 : n]$
This is called a tour.
Repeat tours until satisfication (or convergence).

Lattice reduction (e.g. BKZ-b)

b: Blocksize
Run the local improvements for consecutive blocks:
$[1:b]$, $[2:b+1]$, $[3:b+2]$, $\ldots$, $[n-b:n]$, $[n-b+1:n]$, $\ldots$ $[n-1:n]$
This is called a tour.
Repeat tours until satisfication (or convergence).

BKZ in action

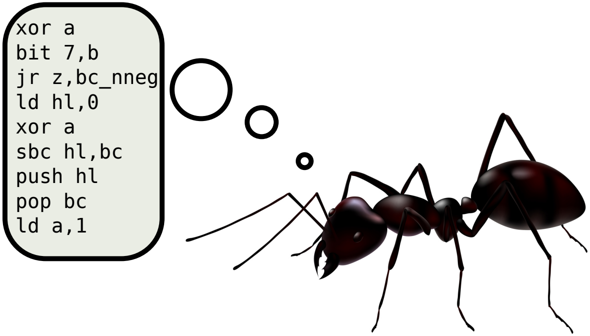
“ Thanks for the lecture, but ...



```
xor a
bit 7,b
jr z,bc_nneg
ld hl,0
xor a
sbc hl,bc
push hl
pop bc
ld a,1
```

how should I solve the SVP puzzle

" Thanks for the lecture, but ...



```
xor a
bit 7,b
jr z,bc_nneg
ld hl,0
xor a
sbc hl,bc
push hl
pop bc
ld a,1
```

how should I solve the *SVP* puzzle

Shortest Vector from Lattice Sieving: a Few Dimensions for Free[2]

[2] EUROCRYPT 2018

Two classes of Algorithms for SVP

The Shortest Vector Problem

I: The basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L}$

O: A shortest non-zero vector $\mathbf{v} \in \mathcal{L}$

Algorithm	Running time	Memory
Enumeration	$n^{n/2e} \cdot 2^{O(n)}$	$\mathrm{poly}(n)$
Sieving[3]	$[2^{.292n+o(n)}, 2^{.415n+o(n)}]$	$[2^{.2075n+o(n)}, 2^{.292n+o(n)}]$

The paradox

In theory, Sieving is faster. In pratice it is quite a lot slower.

[3]Given complexities are heuristic, heavily supported by experiments.

Two classes of Algorithms for SVP

The Shortest Vector Problem

I: The basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L}$

O: A shortest non-zero vector $\mathbf{v} \in \mathcal{L}$

Algorithm	Running time	Memory
Enumeration	$n^{n/2e} \cdot 2^{O(n)}$	$\text{poly}(n)$
Sieving[3]	$[2^{.292n+o(n)}, 2^{.415n+o(n)}]$	$[2^{.2075n+o(n)}, 2^{.292n+o(n)}]$

The paradox

In theory, Sieving is faster. In pratice it is quite a lot slower.

[3]Given complexities are heuristic, heavily supported by experiments.

Two classes of Algorithms for SVP

The Shortest Vector Problem

I: The basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L}$

O: A shortest non-zero vector $\mathbf{v} \in \mathcal{L}$

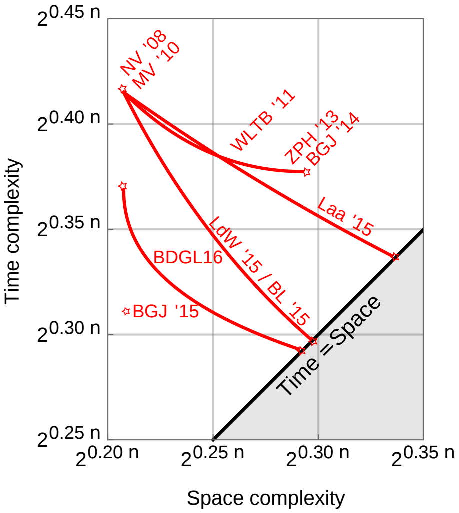
Algorithm	Running time	Memory
Enumeration	$n^{n/2e} \cdot 2^{O(n)}$	$\mathrm{poly}(n)$
Sieving[3]	$[2^{.292n+o(n)}, 2^{.415n+o(n)}]$	$[2^{.2075n+o(n)}, 2^{.292n+o(n)}]$

The paradox

In theory, Sieving is faster. In pratice it is quite a lot slower.

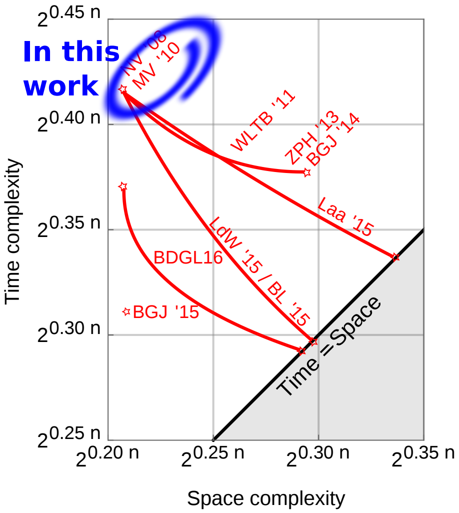
[3] Given complexities are heuristic, heavily supported by experiments.

Many trade-offs



- Our main contribution can also be applied to other sieving algorithms.
- Implementation limited to the version of **[Micciancio Voulgaris 2010]**.

Many trade-offs



- Our main contribution can also be applied to other sieving algorithms.
- Implementation limited to the version of **[Micciancio Voulgaris 2010]**.

Results

Heuristic claim, asymptotic

One can solve SVP in dimension n with a call to SIEVE in dimension $n - d$

where $d = \Theta(n/\log n)$.

Heuristic claim, concrete

One can solve SVP in dimension n making a call to SIEVE in dimension i for each $i = 2 \dots n - d$ for

$$d \approx \frac{n \cdot \ln(4/3)}{\ln(n/2\pi e)} \qquad (d \approx 15 \text{ for } n = 80)$$

Experimental claim: A bogey

A SIEVE implem. almost on par with enumeration (within a factor 4 in dims 70–80), still with room for many improvements.

Results

Heuristic claim, asymptotic

One can solve SVP in dimension n with a call to SIEVE in dimension $n - d$

$$\text{where } d = \Theta(n/\log n).$$

Heuristic claim, concrete

One can solve SVP in dimension n making a call to SIEVE in dimension i for each $i = 2 \ldots n - d$ for

$$d \approx \frac{n \cdot \ln(4/3)}{\ln(n/2\pi e)} \qquad (d \approx 15 \text{ for } n = 80)$$

Experimental claim: A bogey

A SIEVE implem. almost on par with enumeration (within a factor 4 in dims 70–80), still with room for many improvements.

Results

Heuristic claim, asymptotic

One can solve SVP in dimension n with a call to SIEVE in dimension $n - d$

$$\text{where } d = \Theta(n/\log n).$$

Heuristic claim, concrete

One can solve SVP in dimension n making a call to SIEVE in dimension i for each $i = 2 \ldots n - d$ for

$$d \approx \frac{n \cdot \ln(4/3)}{\ln(n/2\pi e)} \qquad (d \approx 15 \text{ for } n = 80)$$

Experimental claim: A bogey

A SIEVE implem. almost on par with enumeration (within a factor 4 in dims 70–80), still with room for many improvements.

Sieving

Algorithm 1 SIEVE($\mathcal{L}$)

```
$L \leftarrow$ a set of $N$ random vectors from $\mathcal{L}$ where $N \approx (4/3)^{n/2}$.
while $\exists(\mathbf{v}, \mathbf{w}) \in L^2$ such that $\|\mathbf{v} - \mathbf{w}\| < \|\mathbf{v}\|$ do
    $\mathbf{v} \leftarrow \mathbf{v} - \mathbf{w}$
end while
return $L$
```

The above runs in heuristic time $(4/3)^{n+o(n)}$.

Many concrete and asymptotic improvements:
[Nguyen Vidick 2008, Micciancio Voulgaris 2010, Laarhoven 2015, Becker Gamma Joux 2015, Becker D. Gamma Laarhoven 2015, . . .].

Sieving

Algorithm 2 $\textsc{Sieve}(\mathcal{L})$

```
$L \leftarrow$ a set of $N$ random vectors from $\mathcal{L}$ where $N \approx (4/3)^{n/2}$.
while $\exists(\mathbf{v}, \mathbf{w}) \in L^2$ such that $\|\mathbf{v} - \mathbf{w}\| < \|\mathbf{v}\|$ do
    $\mathbf{v} \leftarrow \mathbf{v} - \mathbf{w}$
end while
return $L$
```

The above runs in heuristic time $(4/3)^{n+o(n)}$.

Many concrete and asymptotic improvements:
[Nguyen Vidick 2008, Micciancio Voulgaris 2010, Laarhoven 2015, Becker Gamma Joux 2015, Becker D. Gamma Laarhoven 2015, . . .].

More than SVP

Note that $\textsc{Sieve}$ returns $N \approx (4/3)^n$ short vectors, not just a shortest vector.

Definition (Gaussian Heuristic: Expected length of the shortest vector)

$$\mathrm{gh}(\mathcal{L}) = \sqrt{n/2\pi e} \cdot \mathrm{vol}(\mathcal{L})^{1/n}.$$

Observation (heuristic & experimental)

The output of $\textsc{Sieve}$ contains almost all vectors of length $\leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$:

$$L := \textsc{Sieve}(\mathcal{L}) = \left\{ \mathbf{x} \in \mathcal{L} \text{ s.t. } \|\mathbf{x}\| \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}) \right\}.$$

Sieve then Lift

Main idea: Sieve in a projected sub-lattice, and lift all candidate solutions.

$\textsc{SubSieve}(\mathcal{L}, d)$

- Set $\mathcal{L}' = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ "left part of $\mathcal{L}$", dim=d
- Set $\mathcal{L}'' = \pi^{\perp}_{\mathcal{L}'}(\mathcal{L})$ "right part of $\mathcal{L}$", dim=$n-d$
- Compute $L = \textsc{Sieve}(\mathcal{L}'')$
- Hope that $\pi^{\perp}_{\mathcal{L}'}(\mathbf{s}) \in L$ (1)
- Lift all $\mathbf{v} \in L$ from $\mathcal{L}''$ to $\mathcal{L}$ and take the shortest (Babai alg.)

Pessimistic prediction for (1)

$$\mathrm{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}'').$$

Optimistic prediction for (1)

$$\sqrt{\frac{n-d}{n}} \cdot \mathrm{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}'').$$

Similar to linear pruning for enum.

Sieve then Lift

Main idea: Sieve in a projected sub-lattice, and lift all candidate solutions.

$\textsc{SubSieve}(\mathcal{L}, d)$

- Set $\mathcal{L}' = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ "left part of $\mathcal{L}$", dim=d
- Set $\mathcal{L}'' = \pi^{\perp}_{\mathcal{L}'}(\mathcal{L})$ "right part of $\mathcal{L}$", dim=$n-d$
- Compute $L = \textsc{Sieve}(\mathcal{L}'')$
- Hope that $\pi^{\perp}_{\mathcal{L}'}(\mathbf{s}) \in L$ (1)
- Lift all $\mathbf{v} \in L$ from $\mathcal{L}''$ to $\mathcal{L}$ and take the shortest (Babai alg.)

Pessimistic prediction for (1)

$$\mathrm{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}'').$$

Optimistic prediction for (1)

$$\sqrt{\frac{n-d}{n}} \cdot \mathrm{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}'').$$

Similar to linear pruning for enum.

Sieve then Lift

Main idea: Sieve in a projected sub-lattice, and lift all candidate solutions.

$\textsc{SubSieve}(\mathcal{L}, d)$

- Set $\mathcal{L}' = \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_d)$ "left part of $\mathcal{L}$", dim=d
- Set $\mathcal{L}'' = \pi^{\perp}_{\mathcal{L}'}(\mathcal{L})$ "right part of $\mathcal{L}$", dim=$n-d$
- Compute $L = \textsc{Sieve}(\mathcal{L}'')$
- Hope that $\pi^{\perp}_{\mathcal{L}'}(\mathbf{s}) \in L$ (1)
- Lift all $\mathbf{v} \in L$ from $\mathcal{L}''$ to $\mathcal{L}$ and take the shortest (Babai alg.)

Pessimistic prediction for (1)

$$\mathrm{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}'').$$

Optimistic prediction for (1)

$$\sqrt{\frac{n-d}{n}} \cdot \mathrm{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}'').$$

Similar to linear pruning for enum.

Sieve then Lift

Main idea: Sieve in a projected sub-lattice, and lift all candidate solutions.

$\textsc{SubSieve}(\mathcal{L}, d)$

- Set $\mathcal{L}' = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ "left part of $\mathcal{L}$", dim=d
- Set $\mathcal{L}'' = \pi^{\perp}_{\mathcal{L}'}(\mathcal{L})$ "right part of $\mathcal{L}$", dim=$n - d$
- Compute $L = \textsc{Sieve}(\mathcal{L}'')$
- Hope that $\pi^{\perp}_{\mathcal{L}'}(\mathbf{s}) \in L$ (1)
- Lift all $\mathbf{v} \in L$ from $\mathcal{L}''$ to $\mathcal{L}$ and take the shortest (Babai alg.)

Pessimistic prediction for (1)

$$\mathrm{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}'').$$

Optimistic prediction for (1)

$$\sqrt{\frac{n-d}{n}} \cdot \mathrm{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}'').$$

Similar to linear pruning for enum.

With BKZ pre-processing

- To ensure (1), we need the basis to be as reduced as possible
- We can easily afford BKZ preprocessing with block-size $b = n/2$
- Using simple BKZ models[4] we can predict $\mathrm{gh}(\mathcal{L})$ and $\mathrm{gh}(\mathcal{L}')$

Heuristic claim

$\textsc{SubSieve}(\mathcal{L}, d)$ algorithm will successfully find the shortest vector of $\mathcal{L}$ for some $d = \Theta(n/\ln n)$.

$\Rightarrow$ Improve time & memory by a sub-exponential factor $2^{\Theta(n/\log n)}$

[4] The Geometric Series Assumption

With BKZ pre-processing

- To ensure (1), we need the basis to be as reduced as possible
- We can easily afford BKZ preprocessing with block-size $b = n/2$
- Using simple BKZ models[4] we can predict $\mathrm{gh}(\mathcal{L})$ and $\mathrm{gh}(\mathcal{L}')$

Heuristic claim

$\textsc{SubSieve}(\mathcal{L}, d)$ algorithm will successfully find the shortest vector of $\mathcal{L}$ for some $d = \Theta(n/\ln n)$.

$\Rightarrow$ Improve time & memory by a sub-exponential factor $2^{\Theta(n/\log n)}$

[4]The Geometric Series Assumption

Quasi-HKZ preprocessing

Idea: Attempt stronger pre-processing.

Algorithm 3 $\textsc{SubSieve}^+(\mathcal{L}, d)$

```
$L \leftarrow \textsc{Sieve}(\mathcal{L}'')$
$L = \{\textsc{Lift}_{\mathcal{L}'' \to \mathcal{L}}(v) \text{ for } v \in L\}$
for $j = 0 \dots n/2 - 1$ do
    $\mathbf{v}_j = \arg\min_{\mathbf{s} \in L} \|\pi_{(\mathbf{v}_0 \dots \mathbf{v}_{j-1})^\perp}(\mathbf{s})\|$
end for
return $(\mathbf{v}_0 \dots \mathbf{v}_{n/2-1})$
```

- Insert $(\mathbf{v}_0 \dots \mathbf{v}_{n/2-1})$ as the new $\mathbf{b}_1 \dots \mathbf{b}_{n/2}$
- Repeat $\textsc{SubSieve}^+(\mathcal{L}, d)$ for $d = n-1, n-2, \dots, d_{\min}$
- Hope that iteration $d_{\min} + 1$ provided a quasi-HKZ basis.

Concrete prediction with quasi-HKZ preprocessing

Pessimistic prediction for (1)

$$d \approx \frac{n \ln 4/3}{\ln(n/2\pi)}$$

Optimistic prediction for (1)

$$d \approx \frac{n \ln 4/3}{\ln(n/2\pi e)}$$

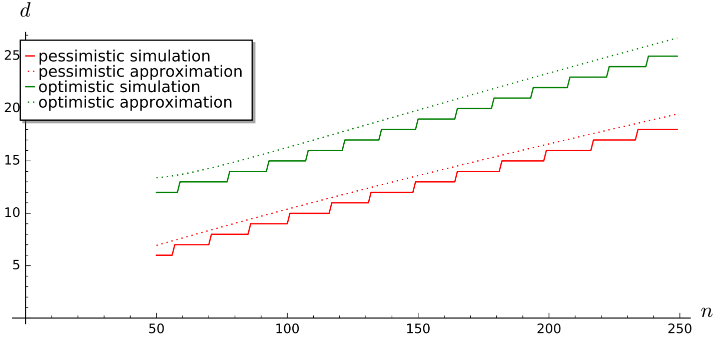


Figure: Predictions of the maximal successful choice of d_{min}.

Baseline Implementation (V0)

Re-implemented GaussSieve **[Micciancio Voulgaris 2010]**

- No gaussian sampling
  - Initial sphericity of L doesn't seem to matter
  - Initial vectors can be made much shorter $\Rightarrow$ speed-up
- Prevent collisions using a hash table
- Terminate when the ball $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$ is half-saturated
- Sort only periodically
  - Can use faster data-structures
- Vectors represented in bases $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)
  - Required to work in projected-sublattices
- Kernel in `c++`, control in `python`
  - Calls to `fpylll` to maintain $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)

Baseline Implementation (V0)

Re-implemented GaussSieve **[Micciancio Voulgaris 2010]**

- No gaussian sampling
  - Initial sphericity of L doesn't seem to matter
  - Initial vectors can be made much shorter $\Rightarrow$ speed-up
- Prevent collisions using a hash table
- Terminate when the ball $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$ is half-saturated
- Sort only periodically
  - Can use faster data-structures
- Vectors represented in bases $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)
  - Required to work in projected-sublattices
- Kernel in `c++`, control in `python`
  - Calls to `fpylll` to maintain $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)

Baseline Implementation (V0)

Re-implemented GaussSieve **[Micciancio Voulgaris 2010]**

- No gaussian sampling
    - Initial sphericity of L doesn't seem to matter
    - Initial vectors can be made much shorter $\Rightarrow$ speed-up
- Prevent collisions using a hash table
- Terminate when the ball $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$ is half-saturated
- Sort only periodically
    - Can use faster data-structures
- Vectors represented in bases $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)
    - Required to work in projected-sublattices
- Kernel in `c++`, control in `python`
    - Calls to `fpylll` to maintain $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)

Baseline Implementation (V0)

Re-implemented GaussSieve **[Micciancio Voulgaris 2010]**

- No gaussian sampling
  - Initial sphericity of L doesn't seem to matter
  - Initial vectors can be made much shorter $\Rightarrow$ speed-up
- Prevent collisions using a hash table
- Terminate when the ball $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$ is half-saturated
- Sort only periodically
  - Can use faster data-structures
- Vectors represented in bases $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)
  - Required to work in projected-sublattices
- Kernel in `c++`, control in `python`
  - Calls to `fpylll` to maintain $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)

Baseline Implementation (V0)

Re-implemented GaussSieve **[Micciancio Voulgaris 2010]**

- No gaussian sampling
  - Initial sphericity of L doesn't seem to matter
  - Initial vectors can be made much shorter $\Rightarrow$ speed-up
- Prevent collisions using a hash table
- Terminate when the ball $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L})$ is half-saturated
- Sort only periodically
  - Can use faster data-structures
- Vectors represented in bases $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)
  - Required to work in projected-sublattices
- Kernel in `c++`, control in `python`
  - Calls to `fpylll` to maintain $\mathbf{B}$ and GramSchmidt($\mathbf{B}$)

XOR−POPCNT trick (V0 $\to$ V1)

Already used in Sieving **[Fitzpatrick et al. 2015]**.
More generally know as SimHash **[Charikar 2002]**.

Idea: Pre-filter pairs $(\mathbf{v}, \mathbf{w}) \in L$ with a fast compressed test.

- Choose a spherical code $\mathcal{C} = \{\mathbf{c}_1 \dots \mathbf{c}_k\} \subset \mathcal{S}^n$ and a threshold $t \leq k/2$
- Precompute compressions $\tilde{\mathbf{v}} = \textsc{Sign}(\langle \mathbf{v}, \mathbf{c}_i \rangle) \in \{0,1\}^k$
- Only test $\|\mathbf{v} \pm \mathbf{w}\| \leq \|\mathbf{v}\|$ if

$$|\textsc{HammingWeight}(\mathbf{v} \oplus \mathbf{w}) - k/2| \geq t.$$

- Asymptotic speed-up $\Theta(n/\log n)$?
- In practice, $k = 128$ (2 words), $t = 18$: about 10 cycles per pairs.

Progressive Sieving (V1 $\rightarrow$ V2)

Concurrently and independetly invented in **[Mariano Laarhoven 2018]**.

Idea: Increase the dimension progressively.

- Recursively, Sieve in the lattice $\mathcal{L}(\mathbf{b}_1, \ldots \mathbf{b}_{n-1})$
- Start the sieve in dimension n with many short-ish vectors
- Fresh vectors get reduced much faster thanks to this initial pool.

Refer to **[Mariano Laarhoven 2018]** for a full analysis of this trick.

Dimensions for Free (V2 $\rightarrow$ V3)

- Apply the quasi-HKZ preprocessing strategy
- Do not force the choice of $d_{\min}$
- Simply increase d until the shortest vector is found.

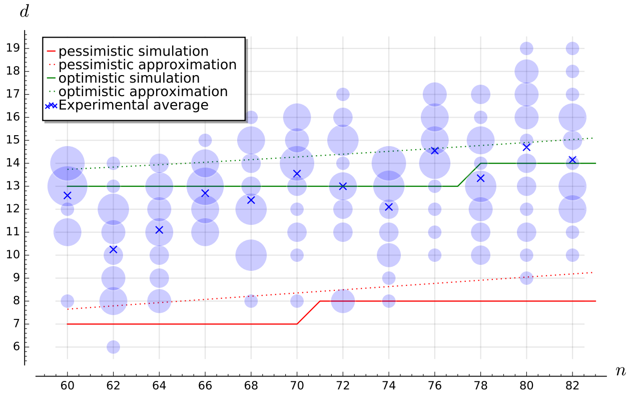
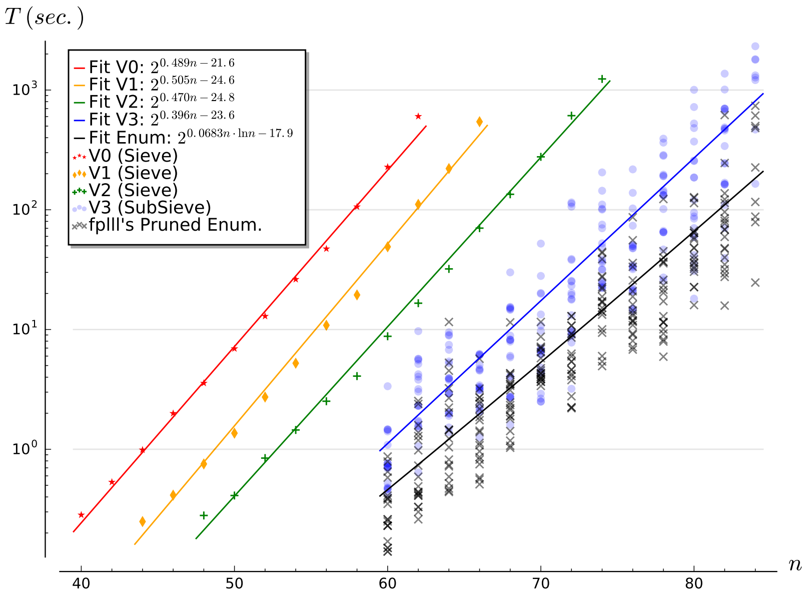


Figure: Predictions experiments for $d_{\min}$.

Performances



Comparison to other Sieving implementation

	Algorithms							
	V0	V1	V2	V3	[MV10]	[FBB+14]	[ML17]	[HK17]
Features								
`XOR-POPCNT` trick		x	x	x		x		
pogressive sieving			x	x				
SubSieve				x				
LSH (more mem.)							x	
tuple (less mem.)								x
Dimension	Running times							
$n = 60$	227s	49s	8s	0.9s	464s	79s	13s	1080s
$n = 70$	-	-	276s	10s	23933s	4500s	250s	33000s
$n = 80$	-	-	-	234s	-	-	4320s	94700s
CPU freq. (GHz)	3.6	3.6	3.6	3.6	4.0	4.0	2.3	2.3

Summary

Sieving vs. Sieving

- Exploit all outputs of Sieve $\Rightarrow$ Dimensions for Free
- Our implementation is 10x faster than all previous Sieving
- It does not use LSH techniques: further speed-up expected

Sieving vs. Enumeration

- Only a factor 4x slower than Enum for dimensions 70–80
- Guesstimates a cross-over at dim $\approx$ 90 with further improvements (LSH/LSF, fine-tuning, vectorization, ...)

Summary

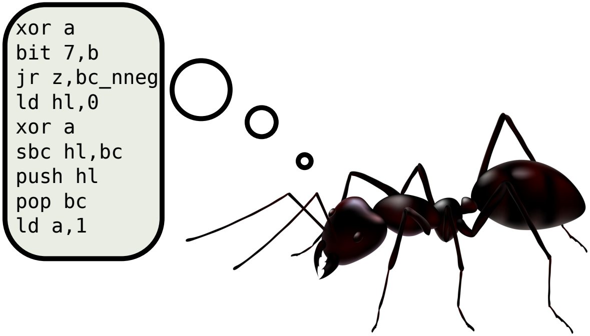
Sieving vs. Sieving

- Exploit all outputs of Sieve $\Rightarrow$ Dimensions for Free
- Our implementation is 10x faster than all previous Sieving
- It does not use LSH techniques: further speed-up expected

Sieving vs. Enumeration

- Only a factor 4x slower than Enum for dimensions 70–80
- Guesstimates a cross-over at dim $\approx$ 90 with further improvements (LSH/LSF, fine-tuning, vectorization, . . .)

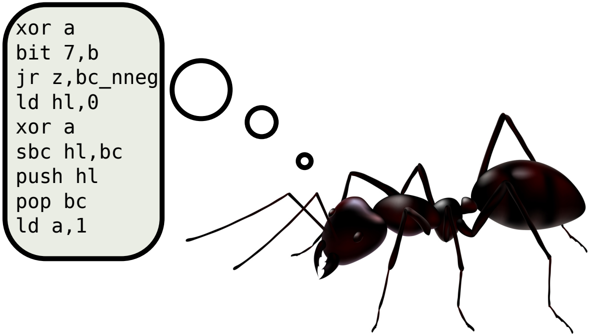
" All that work for a single grain of sand!



```
xor a
bit 7,b
jr z,bc_nneg
ld hl,0
xor a
sbc hl,bc
push hl
pop bc
ld a,1
```

Must I re

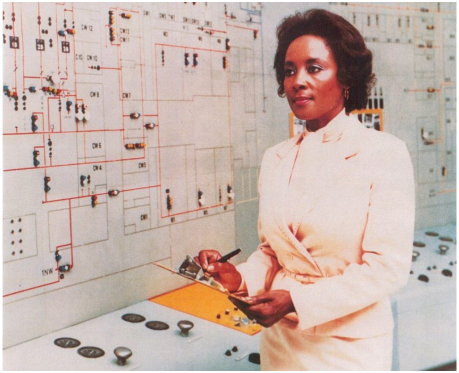
" All that work for a single grain of sand!



```
xor a
bit 7,b
jr z,bc_nneg
ld hl,0
xor a
sbc hl,bc
push hl
pop bc
ld a,1
```

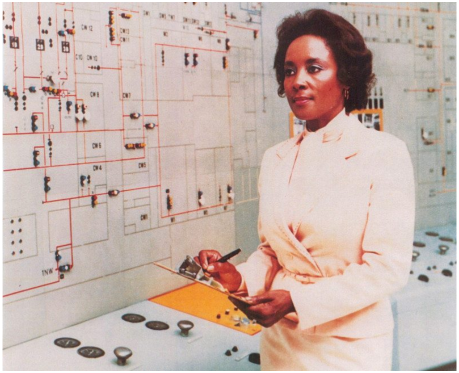
Must I re

“Hum. Let me think.



Maybe we don’t need to re all of this ...”

"Hum. Let me think.



Maybe we don't need to re   all of this …"

The Generalized Sieve Kernel
(G6K, pronounced /ζe.si.ka/) [5]

[5]Work in Progress with M. Albrecht, E. Postlethwaite, G. Herold, E. Kirshanova, M. Stevens

1st design principle: Go Green !

Idea: Recycle vectors between overlapping blocks.

Rather than an function serving as an SVP oracle, design a **stateful machine** that takes advantages of the overlapping instances.

In other words:

An Algorithmic Ant on a Sandpile, carrying a bag of vectors on it

1st design principle: Go Green !

Idea: Recycle vectors between overlapping blocks.

Rather than an function serving as an SVP oracle, design a **stateful machine** that takes advantages of the overlapping instances.

In other words:

An Algorithmic Ant on a Sandpile,
carrying a bag of vectors on it

Moving with a bag of vectors

Relations between the projected sublattices:

$$\begin{tikzcd}
L = & L_{[1:n]} \arrow[d, "\pi"] & \supset & L_{[1:n-1]} \arrow[d, "\pi"] & \supset & \cdots & L_{[1:2]} \arrow[d, "\pi"] & \supset & \ldots L_{[1:1]} \\
& L_{[2:n]} & \supset & L_{[2:n-1]} & \supset & \cdots & L_{[2:2]} & & \\
& \vdots & & & & . & & & \\
& L_{[n-1:n]} \arrow[d, "\pi"] & \supset & L_{[n-1:n-1]} & & & & & \\
& L_{[n:n]} & & & & & & &
\end{tikzcd}$$

- π can be inverted in many ways. Choose π^{-1} to be the Babai lift: the shortest of all possible lifts.
- All maps $\subset$, π^{-1}, π preserve shortness "somewhat"

Moving with a bag of vectors

Relations between the projected sublattices:

$$\begin{tikzcd}
L = & L_{[1:n]} \arrow[d, "\pi"] & \supset & L_{[1:n-1]} \arrow[d, "\pi"] & \supset & \dots & L_{[1:2]} \arrow[d, "\pi"] & \supset & \dots L_{[1:1]} \\
& L_{[2:n]} & \supset & L_{[2:n-1]} & \supset & \dots & L_{[2:2]} & & \\
& \vdots & & & & . & & & \\
& L_{[n-1:n]} \arrow[d, "\pi"] & \supset & L_{[n-1:n-1]} & & & & & \\
& L_{[n:n]} & & & & & & &
\end{tikzcd}$$

- π can be inverted in many ways. Choose π^{-1} to be the Babai lift: the shortest of all possible lifts.
- All maps $\subset$, π^{-1} , π preserve shortness “somewhat”

Moving with a bag of vectors

Relations between the projected sublattices:

$$\begin{tikzcd}
L = & L_{[1:n]} \arrow[d, "\pi"] & \supset & L_{[1:n-1]} \arrow[d, "\pi"] & \supset & \ldots & L_{[1:2]} \arrow[d, "\pi"] & \supset & \ldots L_{[1:1]} \\
& L_{[2:n]} & \supset & L_{[2:n-1]} & \supset & \ldots & L_{[2:2]} & & \\
& \vdots & & & & . & & & \\
& L_{[n-1:n]} \arrow[d, "\pi"] & \supset & L_{[n-1:n-1]} & & & & & \\
& L_{[n:n]} & & & & & & &
\end{tikzcd}$$

- π can be inverted in many ways. Choose π^{-1} to be the Babai lift: the shortest of all possible lifts.
- All maps $\subset$, π^{-1} , π preserve shortness "somewhat"

Moving with a bag of vectors

$$\begin{tikzcd}
L = & L_{[1:n]} \arrow[d, "\pi"] & \supset & L_{[1:n-1]} \arrow[d, "\pi"] & \supset & \cdots & L_{[1:2]} \arrow[d, "\pi"] & \supset & L_{[1:1]} \\
 & L_{[2:n]} & \supset & L_{[2:n-1]} & \supset & \cdots & L_{[2:2]} & & \\
 & \vdots & & \vdots & & \ddots & & & \\
 & L_{[n-1:n]} \arrow[d, "\pi"] & \supset & L_{[n-1:n-1]} & & & & & \\
 & L_{[n:n]} & & & & & & &
\end{tikzcd}$$

Change of context/block `[l:r]` : transform the vectors in the bag

- Extend-Right : $\subset$ (do nothing)
- Shrink-Left : π^{-1} (Babai lift)
- Extend-Left : π (project)

2nd design principle: be flexible

BKZ theory use exact-SVP for each block consecutively, but maybe we're better off making different choices.

- Maintain a cadidate for insertion at **each** position
- Decide where to insert **after** sieving

3rd design principle: seize opportunities

Algorithm 4 $\textsc{Sieve}(\mathcal{L})$

```
$L \leftarrow$ a set of $N$ random vectors from $\mathcal{L}$ where $N \approx (4/3)^{n/2}$.
while $\exists(\mathbf{v}, \mathbf{w}) \in L^2$ such that $\|\mathbf{v} - \mathbf{w}\| < \|\mathbf{v}\|$ do
    $\mathbf{v} \leftarrow \mathbf{v} - \mathbf{w}$
end while
return $L$
```

Even if $\|\mathbf{v} - \mathbf{w}\| \geq \|\mathbf{v}\|$, it could be worth considering the lifts of $\mathbf{v} - \mathbf{w}$.

The abstract machine

State:

- A lattice basis $\mathbf{B}$
- Positions $0 \leq \ell' \leq \ell \leq r \leq d$.
  $[\ell : r]$ the *sieving context*, and $[\ell' : r]$ the *lifting context*.
- A database *db* of N vectors in $\mathcal{L}_{[\ell:r]}$ (preferably short).
- Insertion candidates $\mathbf{c}_{\ell'}, \dots, \mathbf{c}_{\ell}$ where $\mathbf{c}_i \in \mathcal{L}_{[i:r]}$ or $\mathbf{c}_i = \perp$.

Instructions:

- Sieve (`S`): make vector shorter, improve insertion candidates
- Extend Right, Shrink Left, Extend Left (`ER`, `SL`, `EL`): change the sieve-context, updating the database
- Insert (`I`): update the basis and the database

The ideal BKZ with G6K

BKZ can be written very simply:

`Repeat {S; I; ER; }`

When starting the second `Sieve`, vectors are already quite short
$\Rightarrow$ No need to restart progressive sieving from the beginning.

The ER bug.

It turns out that ER is not very compatible with our fastest sieve implementation. Somehow, the Sieve gets stuck in a subspace.

The ideal BKZ with G6K

BKZ can be written very simply:

`Repeat {S; I; ER; }`

When starting the second `Sieve`, vectors are already quite short
$\Rightarrow$ No need to restart progressive sieving from the beginning.

The ER bug.

It turns out that ER is not very compatible with our fastest sieve implementation. Somehow, the Sieve gets stuck in a subspace.

Pump

Before:

$$\texttt{SubSieve}_f : \mathit{Reset}_{0,f,f},\ (\texttt{ER},\ \texttt{S})^{d-f},\ \texttt{I}_0, \texttt{I}_1, \ldots, \texttt{I}_{d-f}.$$

- No issues with EL $\Rightarrow$ Progressive-Sieving toward the left instead.
- Can now Sieve again after insertion
- Can now insert the best candidate rather than a pre-chosen one

$$\texttt{Pump}_{\ell',\ell,r,s} :\ \texttt{Reset}_{\ell',r,r},\ \overbrace{(\texttt{EL},\ \texttt{S})^{r-\ell}}^{\text{pump-up}},\ \overbrace{(\texttt{I},\ \texttt{S})^{r-\ell}}^{\text{pump-down}}.$$

WorkOut

Workout: Pumps of increasing strength

$$\begin{aligned} \texttt{WorkOut}_{\kappa,\beta,f,f^+,s} \ : \ & \texttt{Pump}_{\kappa,\kappa+\beta-f^+,\kappa+\beta,s}, \\ & \texttt{Pump}_{\kappa,\kappa+\beta-2f^+,\kappa+\beta,s}, \\ & \texttt{Pump}_{\kappa,\kappa+\beta-3f^+,\kappa+\beta,s}, \\ & \ldots \\ & \texttt{Pump}_{\kappa,\kappa+f,\kappa+\beta,s}, \end{aligned}$$

- Termination condition can vary (e.g. fixed number of dims for free, or reached satisfying shortest vector)
- steps size of pump strength is not necessarly 1

Pump and Jump

- Block is left somewhat reduced by the pump in the previous block: $\Rightarrow$ no need for a full workout.
- Many short vectors inserted, little improvement left around here: $\Rightarrow$ directly Jump far away.

$$\mathtt{PumpnJumpBKZ}_{\beta',f,j} : \mathtt{Pump}_{0,f,\beta},\ \mathtt{Pump}_{j,j+f,j+\beta},\ \mathtt{Pump}_{2j,2j+f,2j+\beta},\dots$$

Implementation

3 layers

- c++: multi-threaded heavy duty operation (Sieve, *db* updates)
- cython: middleware, basis maintainance
- python: control, tuning, and monitoring

Several Sieve inside:

- Standard Gauss-Sieve (mono-threaded)

$$Mem = 2^{.208n+o(n)}, Time = 2^{.415n+o(n)}$$

- Becker-Gama-Joux with 1 level of filtration (multi-threaded)

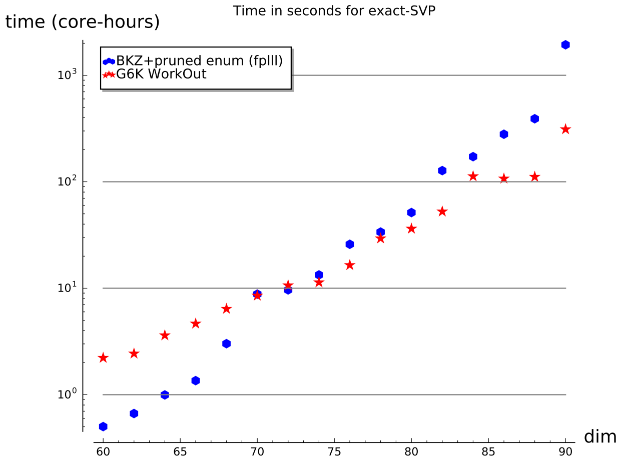
$$Mem = 2^{.208n+o(n)}, Time = 2^{.349n+o(n)}$$

- k-sieve $k = 2, 3$ (multi-threaded)

$$Mem = 2^{.208n+o(n)}, Time = 2^{.349n+o(n)}$$

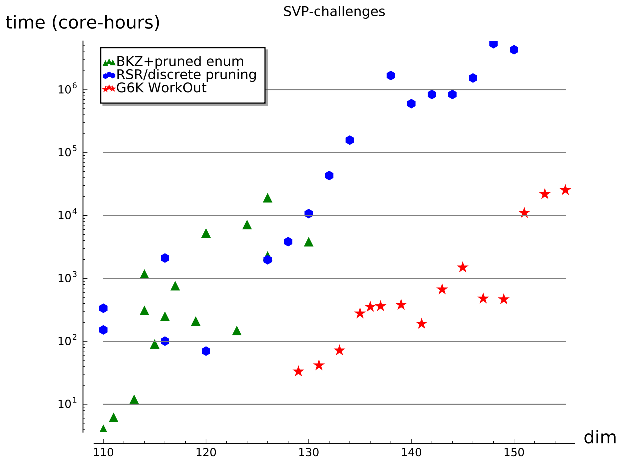
$$Mem = 2^{.189n+o(n)}, Time = 2^{.372n+o(n)}$$

Performances: Exact-SVP



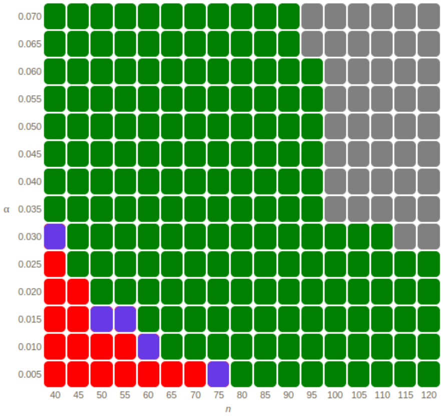
- About 4 extra dims for free
- Cross-over with enum at dim ≈ 70

Records: SVP-challenges



- Solved challenges up to dim 155, with 80 cores in 14 days
- About 400x faster than previous records

Records: LWE-challenges



Red: solved (prior) Blue: solved (ours) Green: unsolved.

- New cost-balancing trick improving upon the prediction of [AGVW17]

Stay tuned

- Paper to be finalized
- Implementation will be made open-source

Thanks!