The General Sieve Kernel

or, The Algorithmic Ant and the Sandpile

Eamonn W. Postlethwaite, with
M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, M. Stevens

February 27, 2023

The Algorithmic Ant and the Sandpile

Once upon a time...

[1] Thanks to Léo Ducas for the ants and images in the story!

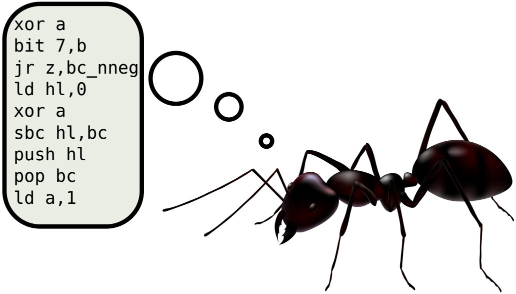
Once upon a time...

there was an ant.

[1]Thanks to Léo Ducas for the ants and images in the story!

Once upon a time…

there was an ant.



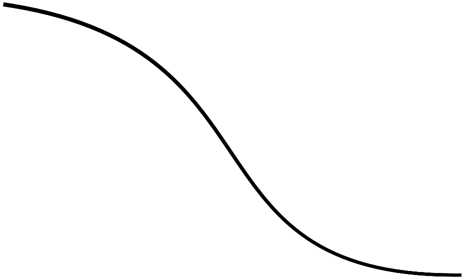
An algorithmic ant.[1]

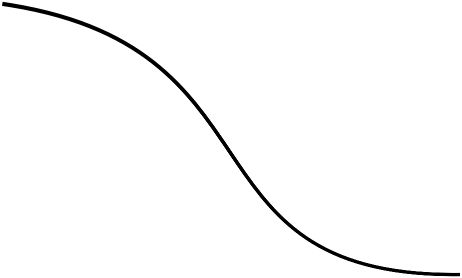
[1] Thanks to Léo Ducas for the ants and images in the story!

The Queen of ants summoned the algorithmic ant,

The Queen of ants summoned the algorithmic ant,

"See this sandpile?"





"I want it flat!"

Looking closer at the sandpile,

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."

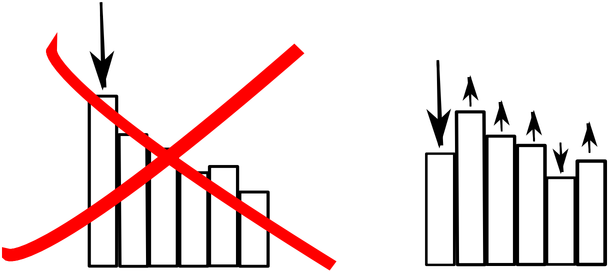
(. . . so hoped the ant.)

But the sandpile is whimsical,

each excavation is a puzzle of its own.

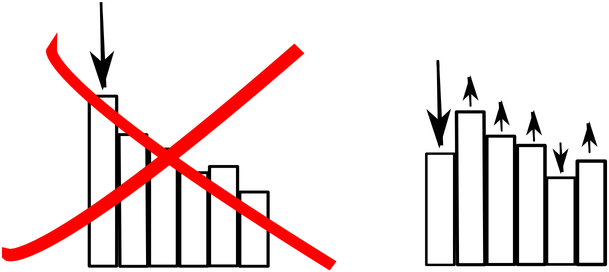
But the sandpile is whimsical,

each excavation is a puzzle of its own.



But the sandpile is whimsical,

each excavation is a puzzle of its own.

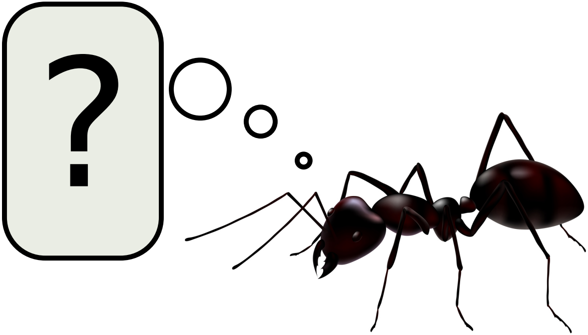


Columns are tied in mysterious ways;

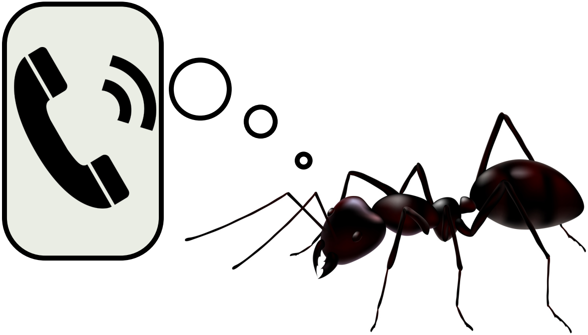
to push one down, one must find

the right combination.

Unsure how to proceed,



Unsure how to proceed,



the ant calls a computer scientist.

"Let's start from the top!"

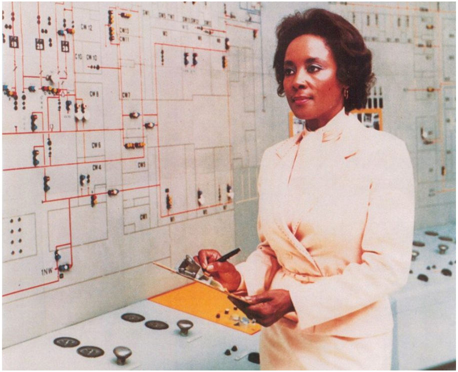
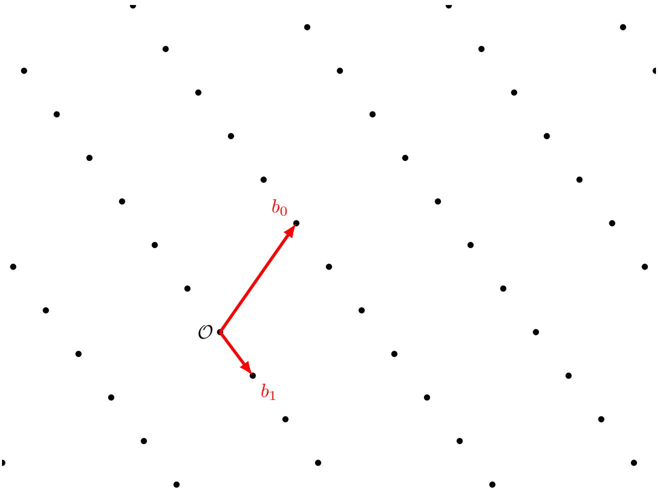


Figure 1: Annie Easley (NASA/NACA)

- Lattices $\leftrightarrow$ Sandpiles
- Sieving for a grain of sand
- Beat the Oracle! G6K, a stateful machine

From Lattices to Sandpiles



$\Lambda = \mathrm{Span}_{\mathbb{Z}}(b_0, \ldots, b_{d-1})$, $B = (b_0, \ldots, b_{d-1}) \subset \mathbb{R}^d$ basis

Define the orthogonal projection of v on u,

$$\pi_u(v) = \frac{\langle v, u \rangle}{\langle u, u \rangle} u.$$

Define the orthogonal projection of v on u,

$$\pi_u(v) = \frac{\langle v, u\rangle}{\langle u, u\rangle} u.$$

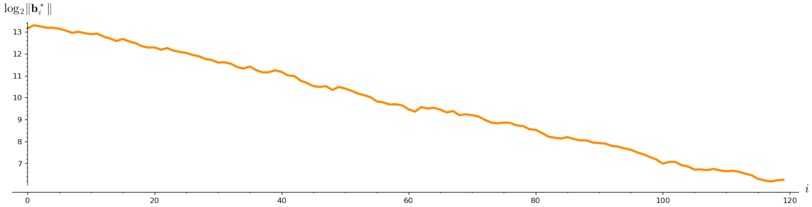
Given B we define the Gram–Schmidt vectors iteratively

$$\begin{aligned}
b_0^* &= \pi_0^\perp(b_0) & &= b_0 \\
b_1^* &= \pi_1^\perp(b_1) & &= b_1 - \pi_{b_0^*}(b_1) \\
b_2^* &= \pi_2^\perp(b_2) & &= b_2 - \pi_{b_0^*}(b_2) - \pi_{b_1^*}(b_2) \\
&\vdots \\
b_i^* &= \pi_i^\perp(b_i) & &= b_i - \sum_{j<i} \pi_{b_j^*}(b_i).
\end{aligned}$$

Let $B^* = (b_0^*, \ldots, b_{d-1}^*)$, the Gram–Schmidt basis of B.

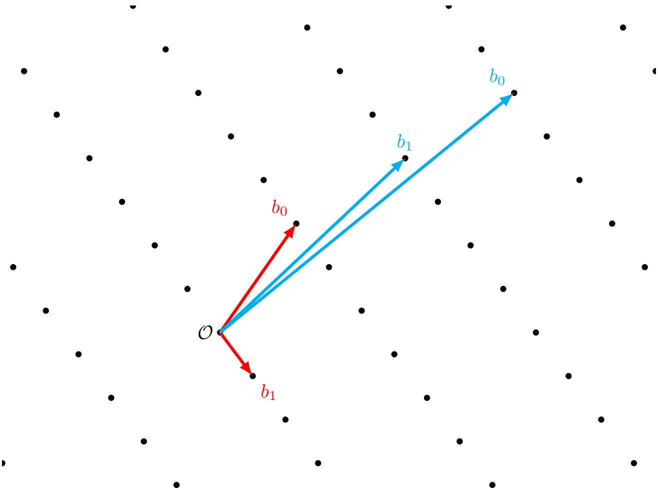
WARNING: $\mathsf{Span}_{\mathbb{Z}}(B^*) \neq \Lambda$!

We get our sandpile from B^*.[2]

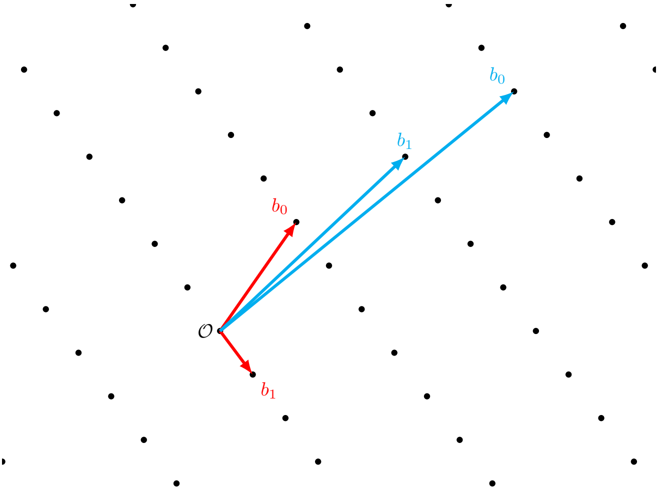


But how to flatten it, and why is it so whimsical?

[2] Thanks to Martin Albrecht for the LLL/BKZ experiments throughout!



Good basis B, bad basis B



B is also a basis of $\Lambda \iff \exists$ unimodular U such that $BU = B$

We have an invariant quantity

$$\mathrm{vol}(\Lambda) = \det(B) = \det(B) = \prod_{i=0}^{d-1} \|b_i^*\| = \prod_{i=0}^{d-1} \|b_i^*\|.$$

We have an invariant quantity

$$\mathrm{vol}(\Lambda) = \det(B) = \det(B) = \prod_{i=0}^{d-1} \|b_i^*\| = \prod_{i=0}^{d-1} \|b_i^*\|.$$

So our desire to “flatten” is constrained by

- constant product of $\|b_i^*\|$
- strong dependence between b_i^*, b_j^*.

Lattice reduction is any procedure P

$$B \xrightarrow{P} B \text{ such that } B^* \text{ is flatter than } B^*$$

- Search for global U that minimises slope?

Lattice reduction is any procedure P

$$B \xrightarrow{P} B \text{ such that } B^* \text{ is flatter than } B^*$$

- Search for global U that minimises slope? Too hard!

Lattice reduction is any procedure P

$$B \xrightarrow{P} B \text{ such that } B^* \text{ is flatter than } B^*$$

- Search for global U that minimises slope? Too hard!
- Search for local $U_{\ell,r}$ to make iterative improvements?

Lattice reduction is any procedure P

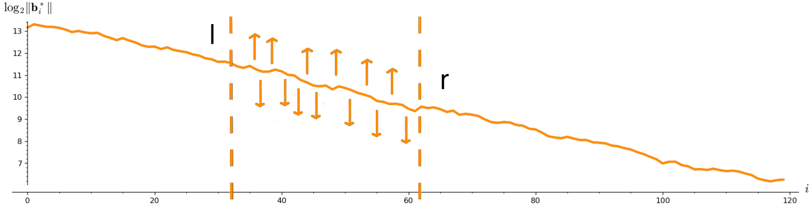
$$B \xrightarrow{P} B \text{ such that } B^* \text{ is flatter than } B^*$$

- Search for global U that minimises slope? Too hard!
- Search for local $U_{\ell,r}$ to make iterative improvements?

BKZ!

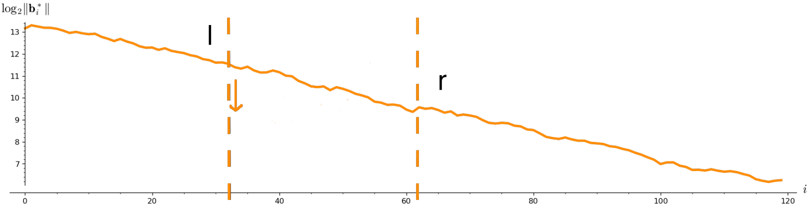
If $U_{\ell,r} = \begin{pmatrix} I_\ell & & \\ & U'_{\ell,r} & \\ & & I_{d-r} \end{pmatrix}$, then $BU_{\ell,r}$ gives

$$\text{If } U_{\ell,r} = \begin{pmatrix} I_\ell & & \\ & U'_{\ell,r} & \\ & & I_{d-r} \end{pmatrix}, \text{ then } BU_{\ell,r} \text{ gives}$$



Even this is too hard, so instead find $U_{\ell,r}$ that minimises $\|b_\ell^*\|$

Even this is too hard, so instead find $U_{\ell,r}$ that minimises $\|b_\ell^*\|$



More specifically let

$$B_{[\ell:\,r]} = (\pi_\ell^\perp(b_\ell), \ldots, \pi_\ell^\perp(b_{r-1})),$$

More specifically let

$$B_{[\ell:\,r]} = (\pi_\ell^\perp(b_\ell), \ldots, \pi_\ell^\perp(b_{r-1})),$$

$$\Lambda_{[\ell:r]} = \mathrm{Span}_{\mathbb{Z}}(B_{[\ell:\,r]}),$$

More specifically let

$$B_{[\ell:\,r]} = (\pi_\ell^\perp(b_\ell), \ldots, \pi_\ell^\perp(b_{r-1})),$$
$$\Lambda_{[\ell:r]} = \mathsf{Span}_{\mathbb{Z}}(B_{[\ell:\,r]}),$$

then $B_{[\ell:\,r]} = (b_\ell^*, \ldots)$, so let us solve SVP in $\Lambda_{[\ell:\,r]}$!

$$\left(\begin{array}{ccccccccc} \overbrace{b_0 \quad b_1 \quad b_2 \quad b_3 \quad b_4}^{\beta = 5} & b_5 & b_6 & b_7 & \dots \end{array}\right)$$

$$\begin{pmatrix} \underbrace{\overbrace{b_0 \quad b_1 \quad b_2 \quad b_3 \quad b_4}^{\beta = 5}} \quad b_5 \quad b_6 \quad b_7 \quad \dots \end{pmatrix}$$

$$\begin{pmatrix} \overbrace{\color{green}{b_0} \quad b_1 \quad b_2 \quad b_3 \quad b_4}^{\beta = 5} \quad b_5 \quad b_6 \quad b_7 \quad \ldots \end{pmatrix}$$

$$\left(\quad b_0 \quad \overbrace{b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5}^{\beta = 5} \quad b_6 \quad b_7 \quad \dots \quad \right)$$

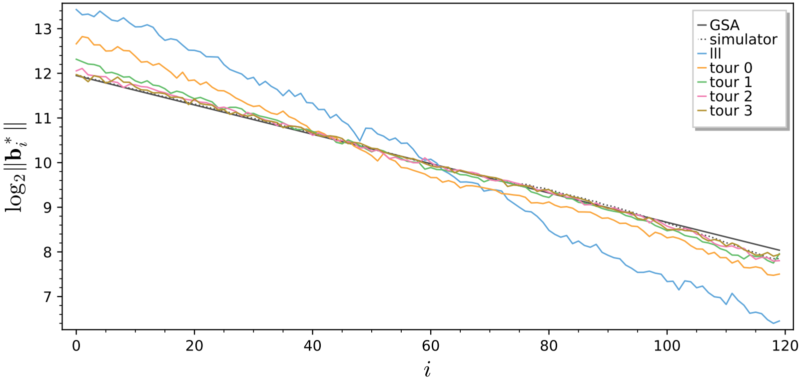
$$\left(\color{green}{b_0} \quad \underbrace{\overbrace{b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5}^{\beta = 5}} \quad b_6 \quad b_7 \quad \dots \right)$$

$$\left(b_0 \quad \overbrace{b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5}^{\beta = 5} \quad b_6 \quad b_7 \quad \dots \right)$$

$$\left(\color{green}{b_0} \quad \color{green}{b_1} \quad \overbrace{b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6}^{\beta = 5} \quad b_7 \quad \dots \right)$$

$$\begin{pmatrix} b_0 \quad b_1 \quad \overbrace{\underbrace{b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6}}^{\beta = 5} \quad b_7 \quad \dots \end{pmatrix}$$

$$\begin{pmatrix} b_0 & b_1 & \overbrace{b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6}^{\beta = 5} & b_7 & \dots \end{pmatrix}$$

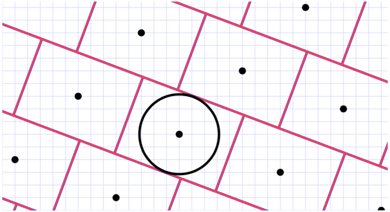


GSA
simulator
lll
tour 0
tour 1
tour 2
tour 3
$\log_2 \|\mathbf{b}_i^*\|$
i
13
12
11
10
9
8
7
0
20
40
60
80
100
120

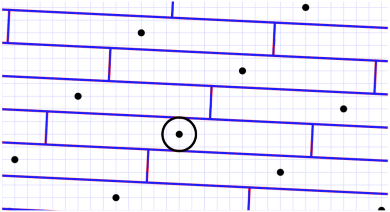
A flat basis helps with the

- *dual* and *primal* LWE attacks (practical)
- shortest independent vectors problem (fundamental)
- approximate closest vector problem (pertinent here!)

We solve the "approx CVP" with Babai's Nearest Plane.



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



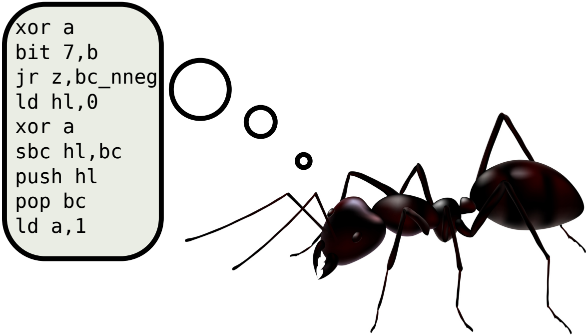
Decoding radius with $\mathbf{B}^\star$

- Worst case distance $\frac{1}{2}\sqrt{\sum \|b_i^*\|^2}$ (approx CVP)
- Correct decoding of $t = v + e$ where $v \in \Lambda$ if (BDD)

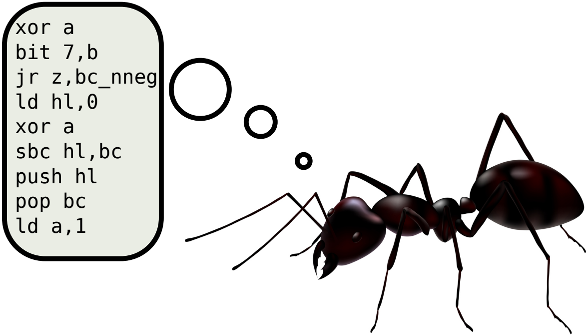
$$\|e\| \leq \frac{1}{2} \min \|b_i^*\|$$

Liberté, Égalité, Fraternité!

“Thanks for the lecture, but...

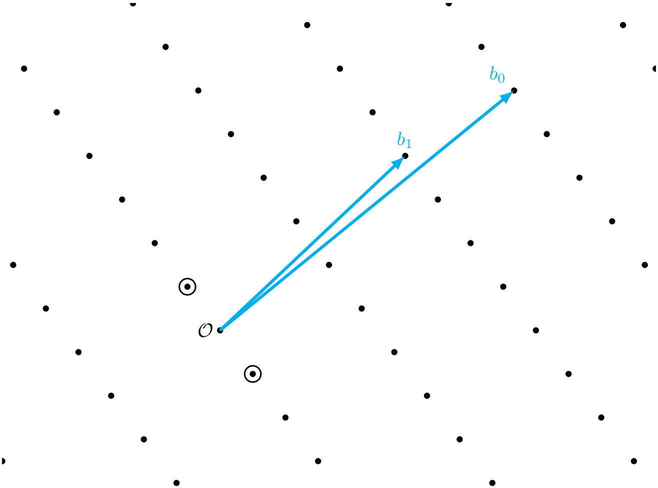


"Thanks for the lecture, but...

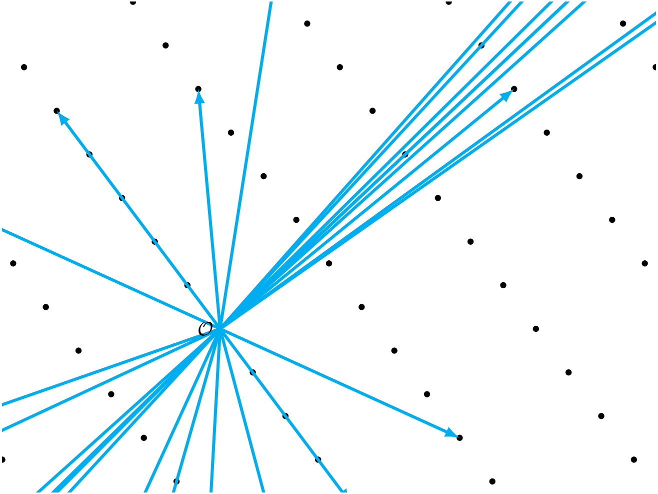


how should I solve these SVP puzzles?"

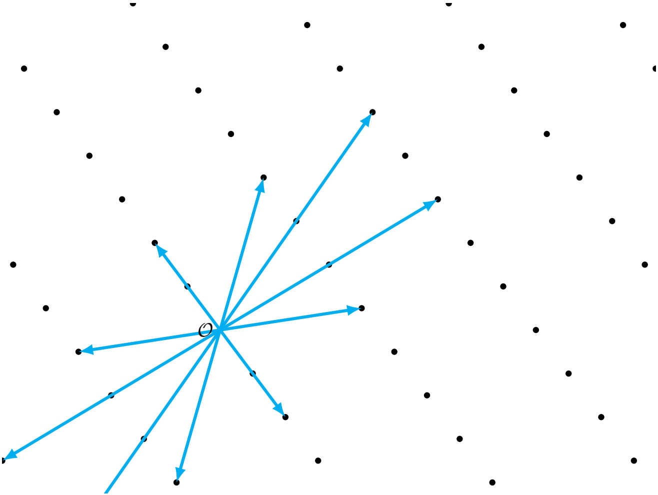
Sieving for a Grain of Sand



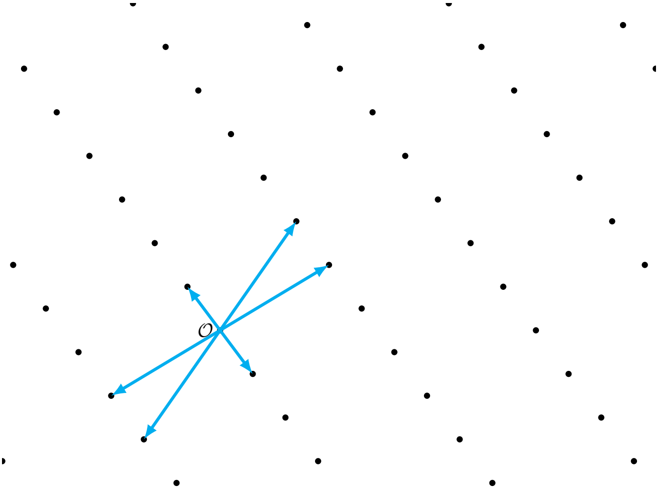
SVP: find $v \in \Lambda \setminus \{0\}$ such that $\|v\| \leq \|w\|$ for all $w \in \Lambda \setminus \{0\}$



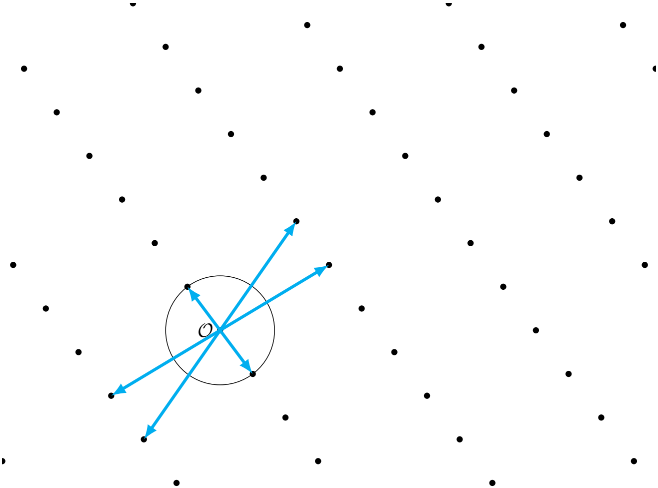
Sieve: $v, w \in L \subset \Lambda$, if $\|v \pm w\| \leq \|v\|$, $v \leftarrow v \pm w$



Sieve: $v, w \in L \subset \Lambda$, if $\|v \pm w\| \leq \|v\|$, $v \leftarrow v \pm w$



Sieve: $v, w \in L \subset \Lambda$, if $\|v \pm w\| \leq \|v\|$, $v \leftarrow v \pm w$



Sieve: $v, w \in L \subset \Lambda$, if $\|v \pm w\| \leq \|v\|$, $v \leftarrow v \pm w$

So $\mathrm{BKZ} = (\mathrm{Sieve}(\Lambda_{[0:\,\beta]}), \mathrm{Sieve}(\Lambda_{[1:\,\beta+1]}), \ldots)^{\mathrm{tours}}$.

So BKZ $= (\mathrm{Sieve}(\Lambda_{[0:\,\beta]}), \mathrm{Sieve}(\Lambda_{[1:\,\beta+1]}), \ldots)^{\mathrm{tours}}$.

Some sieving facts

- dimension d lattice (time, space) $= (\exp(\Theta(d)), \exp(\Theta(d)))$
- a Sieve outputs most of the shortest vectors in a lattice…

Define the Gaussian heuristic (expected length of SVP solution)

$$\mathrm{gh}(\Lambda) = \sqrt{\frac{d}{2\pi e}}\mathrm{vol}(\Lambda)^{1/d}.$$

[3] Can parameterise termination by ensuring some constant fraction of L.

Define the Gaussian heuristic (expected length of SVP solution)

$$\mathrm{gh}(\Lambda) = \sqrt{\frac{d}{2\pi e}} \mathrm{vol}(\Lambda)^{1/d}.$$

The output of Sieve is most[3] of

$$L = \mathrm{Sieve}(\Lambda) = \left\{ v \in \Lambda \text{ s.t. } \|v\| \leq \sqrt{\frac{4}{3}}\, \mathrm{gh}(\Lambda) \right\}.$$

[3]Can parameterise termination by ensuring some constant fraction of L.

Can we use more of L than simply the shortest vector?

---

[4] Thijs Laarhoven and Artur Mariano, Progressive lattice sieving, Post-Quantum Cryptography – 9th International Conference, PQCrypto 2018 (Tanja Lange and Rainer Steinwandt, eds.), Springer, Heidelberg, 2018, pp. 292–311.

[5] L. Babai, On lovász' lattice reduction and the nearest lattice point problem, Combinatorica 6 (1986), no. 1, 1–13.

[6] Léo Ducas, Shortest vector from lattice sieving: A few dimensions for free, EUROCRYPT 2018, Part I (Jesper Buus Nielsen and Vincent Rijmen, eds.), LNCS, vol. 10820, Springer, Heidelberg, April / May 2018, pp. 125–145.

Can we use more of L than simply the shortest vector? Yes!

[4] Thijs Laarhoven and Artur Mariano, Progressive lattice sieving, Post-Quantum Cryptography – 9th International Conference, PQCrypto 2018 (Tanja Lange and Rainer Steinwandt, eds.), Springer, Heidelberg, 2018, pp. 292–311.

[5] L. Babai, On lovász' lattice reduction and the nearest lattice point problem, Combinatorica 6 (1986), no. 1, 1–13.

[6] Léo Ducas, Shortest vector from lattice sieving: A few dimensions for free, EUROCRYPT 2018, Part I (Jesper Buus Nielsen and Vincent Rijmen, eds.), LNCS, vol. 10820, Springer, Heidelberg, April / May 2018, pp. 125–145.

Can we use more of L than simply the shortest vector? Yes!

- sieve in sublattices to “seed” higher dimensional sieves[4]
- sieve in (projected) sublattices and lift[5] to the full lattice[6]

---

[4] Thijs Laarhoven and Artur Mariano, Progressive lattice sieving, Post-Quantum Cryptography – 9th International Conference, PQCrypto 2018 (Tanja Lange and Rainer Steinwandt, eds.), Springer, Heidelberg, 2018, pp. 292–311.

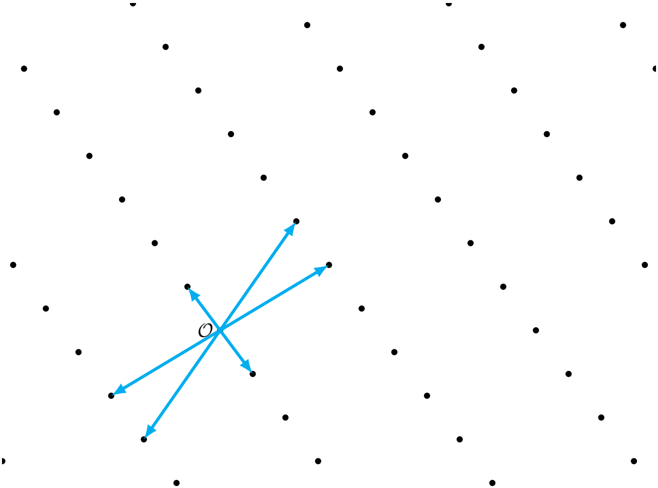
[5] L. Babai, On lovász’ lattice reduction and the nearest lattice point problem, Combinatorica 6 (1986), no. 1, 1–13.

[6] Léo Ducas, Shortest vector from lattice sieving: A few dimensions for free, EUROCRYPT 2018, Part I (Jesper Buus Nielsen and Vincent Rijmen, eds.), LNCS, vol. 10820, Springer, Heidelberg, April / May 2018, pp. 125–145.

Imagine we want to sieve in $\Lambda = \mathsf{Span}_{\mathbb{Z}}(b_0, \ldots, b_{d-1})$.

Imagine we want to sieve in $\Lambda = \mathsf{Span}_{\mathbb{Z}}(b_0, \ldots, b_{d-1})$.

function Progressive Sieve($b_0, \ldots, b_{d-1}$)
 $L \leftarrow \text{Sieve}((b_0, b_1), \emptyset)$
 for $i \in \{2, \ldots, d-1\}$ **do**
  $L \leftarrow \text{Sieve}((b_0, \ldots, b_i), L)$
 end for
 Return L
end function



Some $v \in \mathsf{Span}_{\mathbb{Z}}(b_0, b_1, b_2)$ will shorten quicker!

Let $s \in \Lambda$ be a shortest vector. Rather than (eventually)

$$s \in L = \text{Sieve}(b_0, \ldots, b_{d-1}),$$

Let $s \in \Lambda$ be a shortest vector. Rather than (eventually)

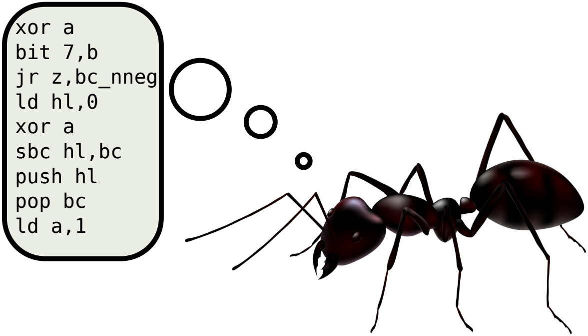
$$s \in L = \mathrm{Sieve}(b_0, \ldots, b_{d-1}),$$

hope that

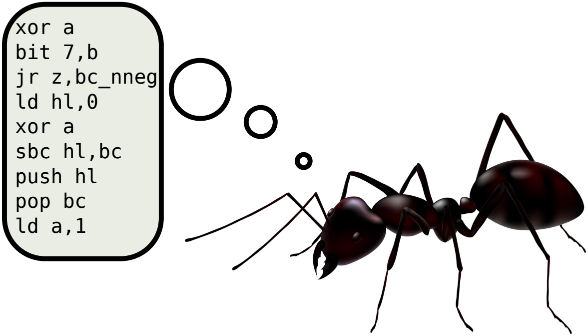
$$\pi_f^{\perp}(s) \in L = \mathrm{Sieve}(\Lambda_{[f\,:\,d]}), \text{ and } s \in \mathrm{Lift}(L).$$

Both conditions are determined by how flat the sandpile is!

“All that work for a single grain of sand!

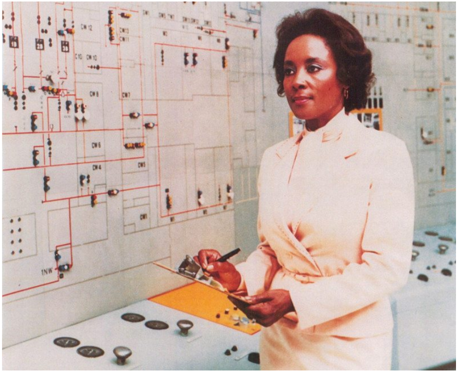


"All that work for a single grain of sand!

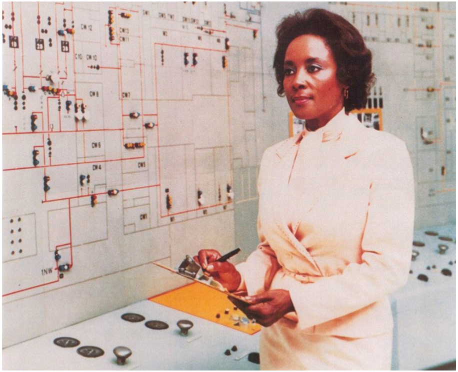


Must I repeat it all for each grain?"

“Hum. Let me think.



“Hum. Let me think.



Maybe we don’t need to repeat all of it...”

The Generalised Sieve Kernel[7]
(G6K, pronounced /ζe.si.ka/)

[7] Albrecht M. R., Ducas L., Herold G., Kirshanova E., Postlethwaite E. W., Stevens M. (2019) The General Sieve Kernel and New Records in Lattice Reduction. In: Ishai Y., Rijmen V. (eds) Advances in Cryptology – EUROCRYPT 2019. EUROCRYPT 2019. Lecture Notes in Computer Science, vol 11477. Springer, Cham

Idea: Recycle vectors between overlapping SVP instances.

Rather than an SVP oracle, Sieve is a stateful machine!

Idea: Recycle vectors between overlapping SVP instances.

Rather than an SVP oracle, Sieve is a stateful machine!

An algorithmic ant on a sandpile,
carrying a bag of vectors on its back.

$$
\begin{array}{ccccccccc}
\Lambda = & \Lambda_{[0:d]} & \supset & \Lambda_{[0:d-1]} & \supset & \cdots & \Lambda_{[0:2]} & \supset & \Lambda_{[0:1]} \\
& \downarrow \pi_1^{\perp} & & \downarrow \pi_1^{\perp} & & & \downarrow \pi_1^{\perp} & & \\
& \Lambda_{[1:d]} & \supset & \Lambda_{[1:d-1]} & \supset & \cdots & \Lambda_{[1:2]} & & \\
& \vdots & & \vdots & & \iddots & & & \\
& \Lambda_{[d-2:d]} & \supset & \Lambda_{[d-1:d]} & \iddots & & & & \\
& \downarrow \pi_{d-1}^{\perp} & & & & & & & \\
& \Lambda_{[d-1:d]} & & & & & & &
\end{array}
$$

The $\pi_i^{\perp}$ are “inverted” via Nearest Plane.

If $L = \text{Sieve}\left(\Lambda_{[\ell:r]}\right)$ then we take our vectors with us.

$$\begin{tikzcd}
\Lambda_{[\ell-1:r]} \arrow[d, shift left, "\pi_\ell^\perp"] & \supset & \Lambda_{[\ell-1:r-1]} \\
\Lambda_{[\ell:r]} \arrow[u, shift left, "\left(\pi_\ell^\perp\right)^{-1}"]
\end{tikzcd}$$

Extend right: $\subset$ (Inclusion – ER)
Shrink left: $\pi_\ell^\perp$ (Project – SL)
Extend left: $\left(\pi_\ell^\perp\right)^{-1}$ (Lift – EL)

$$
\begin{pmatrix}
 & & | & & & & | & & \\
b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & \dots \\
 & & | & & & & | & &
\end{pmatrix}
$$

$$\left(b_0 \quad b_1 \quad \overbrace{\underbrace{b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6}}^{\beta = 5} \quad b_7 \quad \dots \right)$$

$$\begin{array}{c} \begin{matrix} i_0 & i_1 & i_2 & & & & & & \end{matrix} \\ \begin{pmatrix} & & \vert & & & & \vert & & \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & \dots \\ & & \vert & & & & \vert & & \end{pmatrix} \end{array}$$

$$
\begin{array}{c}
\begin{array}{ccccccccc} i_0 & i_1 & i_2 & & & & & & \end{array} \\
\begin{pmatrix}
 & & & \vert & & & & \vert & \\
b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & \ldots \\
 & & & \vert & & & & \vert &
\end{pmatrix}
\end{array}
$$

$$\begin{pmatrix} i_0 & i_1 & i_2 & \overbrace{\hphantom{b_3 \quad b_4 \quad b_5 \quad b_6 \quad b_7}}^{\beta = 5} & \\ b_0 & b_1 & b_2 & \underbrace{b_3 \quad b_4 \quad b_5 \quad b_6 \quad b_7} & \dots \end{pmatrix}$$

$$
\begin{array}{c}
\begin{array}{cccccccc} i_0 & i_1 & i_2 & i_3 & & & & \end{array} \\
\begin{pmatrix}
 & & & | & & & & | & \\
b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & \dots \\
 & & & | & & & & | &
\end{pmatrix}
\end{array}
$$

The Full Abstract Machine

State:

- Basis B,
- Indices $0 \leq \kappa \leq \ell \leq r \leq d$, $[\kappa : r]$ "lifting context", $[\ell : r]$ "sieving context",
- Database, L, of (short) vectors in $\Lambda_{[\ell:r]}$,
- Insertion candidates $i_\kappa, \ldots, i_\ell$, $i_j \in \Lambda_{[j:r]}$, or $i_j = \perp$.

Instructions:

- Reset (R): empty L, set (κ, ℓ, r),
- Sieve (S): sieve in $\Lambda_{[\ell:r]}$, shorten $v \in L$, better i_j,
- $\{$EL, ER, SL$\}$: change $[\ell : r]$ and apply to L,
- Insert (I): update B and L, $[\ell : r] \to [\ell + 1 : r]$.

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

Key: sieve, lift, c_i = insert

A Pump

$\text{Pump}_{f,r}$: $\text{Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$

$$(b_0, b_1, \ldots, b_{f-1}, b_f, b_{f+1}, \ldots, b_{d-2}, b_{d-1})$$

Reset!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$(b_0, b_1, \ldots, b_{f-1}, b_f, b_{f+1}, \ldots, \color{red}{b_{d-2}}, \color{red}{b_{d-1}})$$

Extend Left!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$(b_0, b_1, \ldots, b_{f-1}, b_f, b_{f+1}, \ldots, b_{d-2}, b_{d-1})$$

Sieve!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$({\color{blue}b_0}, {\color{blue}b_1}, \ldots, {\color{blue}b_{f-1}}, {\color{blue}b_f}, {\color{red}b_{f+1}}, \ldots, {\color{red}b_{d-2}}, {\color{red}b_{d-1}})$$

Extend Left!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$(b_0, b_1, \ldots, b_{f-1}, b_f, b_{f+1}, \ldots, b_{d-2}, b_{d-1})$$

Sieve!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$(b_0, b_1, \ldots, b_{f-1}, b_f, b_{f+1}, \ldots, b_{d-2}, b_{d-1})$$

Extend Left!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$({\color{blue}b_0}, {\color{blue}b_1}, \ldots, {\color{blue}b_{f-1}}, {\color{red}b_f}, {\color{red}b_{f+1}}, \ldots, {\color{red}b_{d-2}}, {\color{red}b_{d-1}})$$

Sieve!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$({c_0}, {\color{blue}b_1}, \ldots, {\color{blue}b_{f-1}}, b_f, {\color{red}b_{f+1}}, \ldots, {\color{red}b_{d-2}}, {\color{red}b_{d-1}})$$

Insert!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$(b_0, b_1, \ldots, b_{f-1}, b_f, b_{f+1}, \ldots, b_{d-2}, b_{d-1})$$

Sieve!

Key: sieve, lift, c_i = insert

Pump$_{f,r}$: Reset$_{r-1,r}$ $\overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$

$$(b_0, c_1, \ldots, b_{f-1}, b_f, b_{f+1}, \ldots, b_{d-2}, b_{d-1})$$

Insert!

Key: sieve, lift, c_i = insert

A Pump

$$\text{Pump}_{f,r}\text{: Reset}_{r-1,r}\ \overbrace{(\text{EL S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\text{I S})^{r-f-1}}^{\text{pump-down}}$$

$$(b_0, b_1, \ldots, b_{f-1}, b_f, b_{f+1}, \ldots, b_{d-2}, b_{d-1})$$

Sieve!

Key: sieve, lift, c_i = insert

A Pump

$$\mathrm{Pump}_{f,r}\text{: } \mathrm{Reset}_{r-1,r}\ \overbrace{(\mathrm{EL\ S})^{r-f-1}}^{\text{pump-up}}\ \overbrace{(\mathrm{I\ S})^{r-f-1}}^{\text{pump-down}}$$

$$(c_0, c_1, \dots, c_{f-1}, c_f, c_{f+1}, \dots, c_{d-2}, b_{d-1})$$

Insert!

Key: sieve, lift, c_i = insert

A Workout

$\text{Workout}_{f,f^+,r}$: $\text{Pump}_{0,r-f^+,r}$ $\text{Pump}_{0,r-2f^+,r}$ $\ldots\text{Pump}_{0,r-f,r}$

A Workout

$\text{Workout}_{f,f^+,r}$: $\text{Pump}_{0,r-f^+,r}$ $\text{Pump}_{0,r-2f^+,r}$ ...$\text{Pump}_{0,r-f,r}$

$$(\qquad\qquad\qquad | \qquad\qquad\qquad)$$

A Workout

$\mathrm{Workout}_{f,f^+,r}$: $\mathrm{Pump}_{0,r-f^+,r}$ $\mathrm{Pump}_{0,r-2f^+,r}$ ...$\mathrm{Pump}_{0,r-f,r}$

(|)

A Workout

$\text{Workout}_{f,f^+,r}$: $\text{Pump}_{0,r-f^+,r}$ $\text{Pump}_{0,r-2f^+,r}$ ...$\text{Pump}_{0,r-f,r}$

$$(\qquad\qquad | \qquad\qquad\qquad\qquad\qquad\qquad)$$

Workout$_{f,f^+,r}$: Pump$_{0,r-f^+,r}$ Pump$_{0,r-2f^+,r}$ ...Pump$_{0,r-f,r}$

$$(\qquad \mid \qquad\qquad\qquad)$$

$$(\quad \mid \qquad\qquad\qquad\quad)$$

A `Workout`

$\text{Workout}_{f,f^+,r}$: $\text{Pump}_{0,r-f^+,r}$ $\text{Pump}_{0,r-2f^+,r}$ ...$\text{Pump}_{0,r-f,r}$

$$(\quad | \quad)$$

$$(\quad | \quad)$$

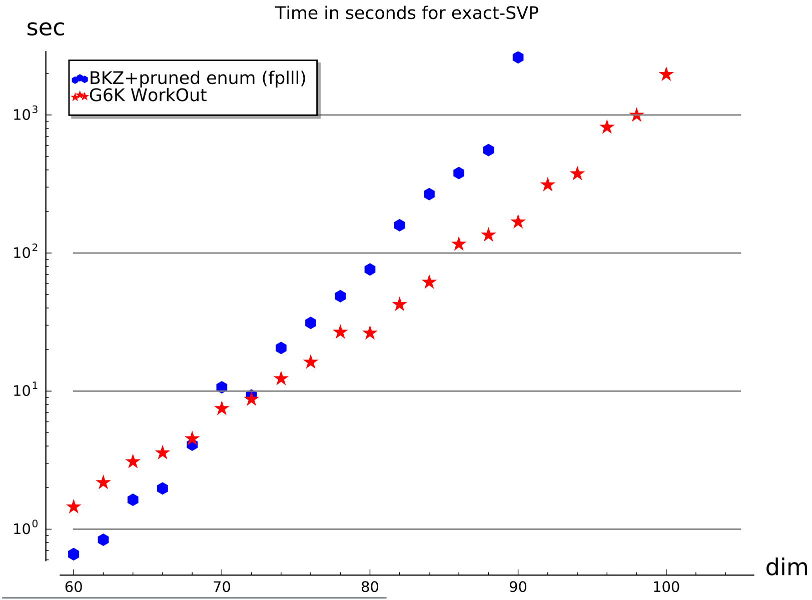
In short, more, smaller `Pumps` still faster than one larger `Pump`.

G6K has three high level design principles, to

- recycle (short) vectors between lattices,
- lift vectors, on the fly, to higher dimensional lattices,
- decide the insertion position only *after* sieving.

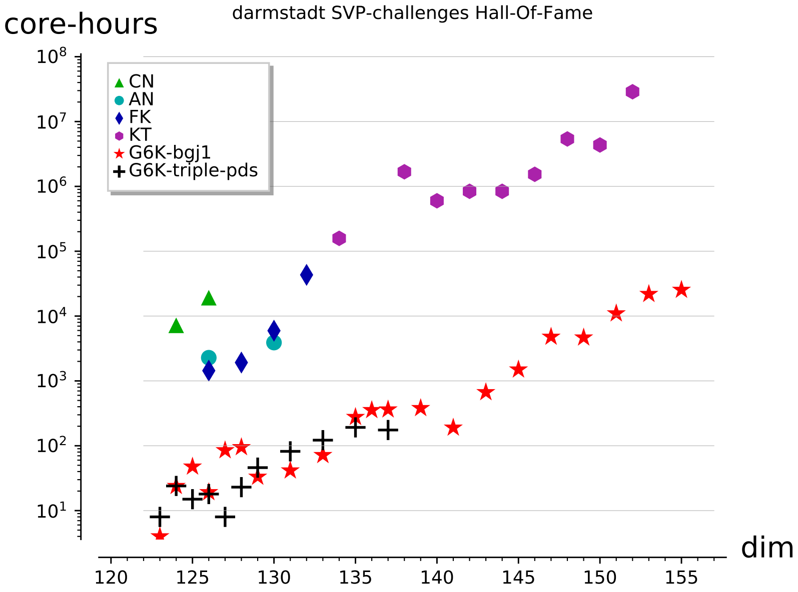
Records

Exact SVP, `Workout` vs. Enumeration (FPLLL)[8]

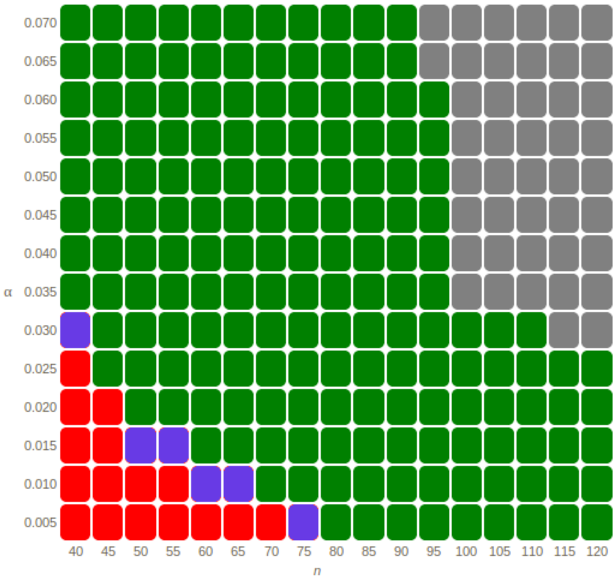


[8] The FPLLL development team, fplll, a lattice reduction library, 2019, available at https://github.com/fplll/fplll

Hermite SVP (Darmstadt Challenges), with `Workout`



LWE (Darmstadt Challenges), with Pump&JumpBKZ



Please hack around with the open source implementation!

`https://www.github.com/fplll/g6k`