Shortest Vector from Lattice Sieving: a Few Dimensions for Free

Léo Ducas[1]

Cryptology Group, CWI, Amsterdam, The Netherlands

EUROCRYPT 2018
Tel Aviv, April 30th

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

Two class 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[2]	$[2^{.292n+o(n)}, 2^{.415n+o(n)}]$	$[2^{.2075n+o(n)}, 2^{.292n+o(n)}]$

The paradox

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

[2] Given complexity are heuristic, heavily supported by experiments.

Two class 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[2]	$[2^{.292n+o(n)}, 2^{.415n+o(n)}]$	$[2^{.2075n+o(n)}, 2^{.292n+o(n)}]$

The paradox

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

[2] Given complexity are heuristic, heavily supported by experiments.

Two class 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 $\mathbf{v} \in \mathcal{L}$

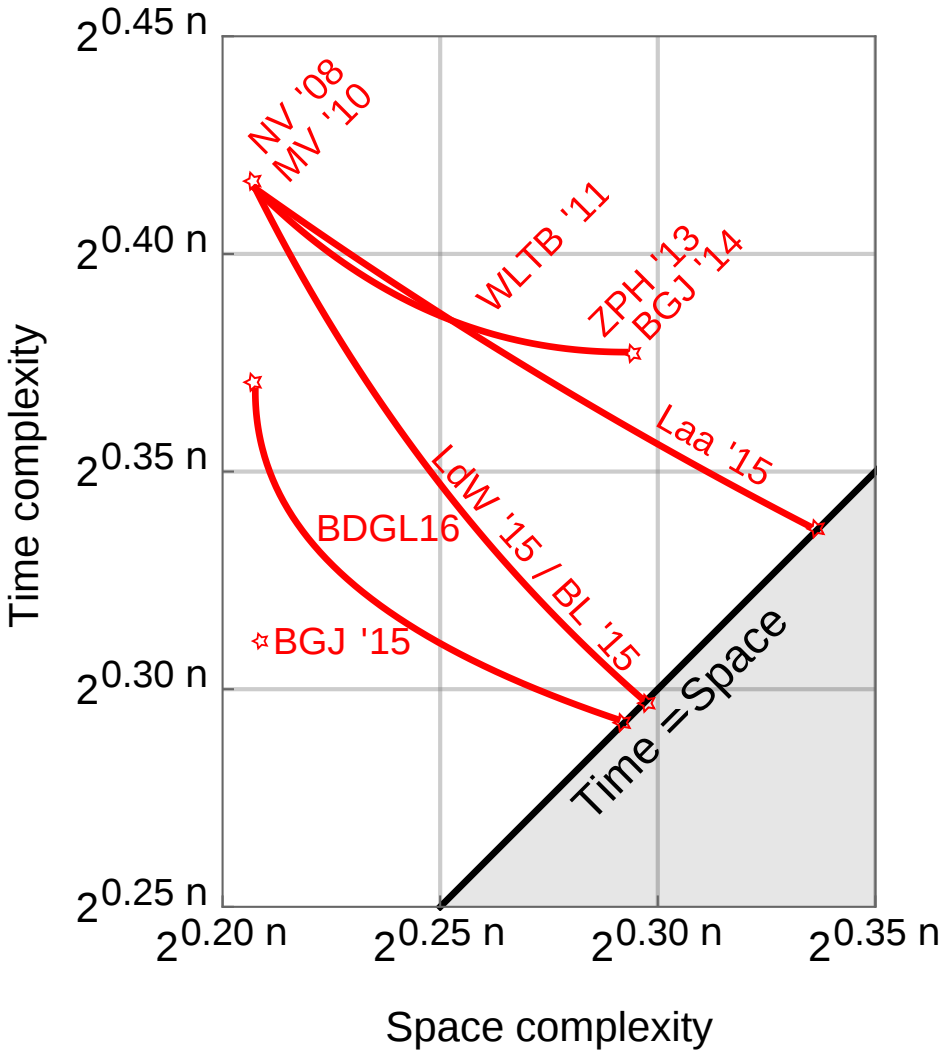
Algorithm	Running time	Memory
Enumeration	$n^{n/2e} \cdot 2^{O(n)}$	$\mathrm{poly}(n)$
Sieving[2]	$[2^{.292n+o(n)}, 2^{.415n+o(n)}]$	$[2^{.2075n+o(n)}, 2^{.292n+o(n)}]$

The paradox

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

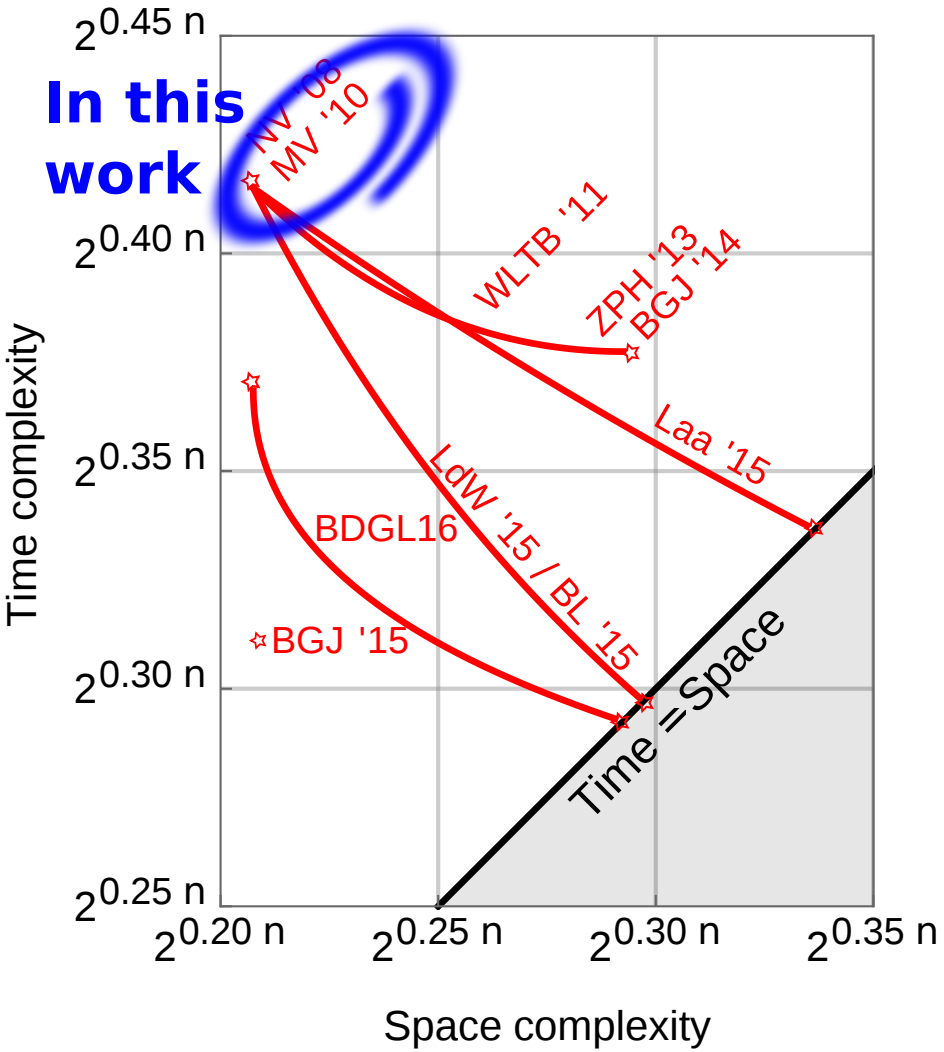
[2] Given complexity are heuristic, heavily supported by experiments.

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 $\textsc{Sieve}$ 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 $\textsc{Sieve}$ in dimension i for each $i = 1 \dots n - d$ 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 $\textsc{Sieve}$ implem. almost on par with enumeration (within a factor 4 in dims 70–80), with still much room for improvements.

Results

Heuristic claim, asymptotic

One can solve SVP in dimension n with a call to $\textsc{Sieve}$ 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 $\textsc{Sieve}$ in dimension i for each $i = 1 \dots n - d$ 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 $\textsc{Sieve}$ implem. almost on par with enumeration (within a factor 4 in dims 70–80), with still much room for improvements.

Results

Heuristic claim, asymptotic

One can solve SVP in dimension n with a call to $\textsc{Sieve}$ 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 $\textsc{Sieve}$ in dimension i for each $i = 1 \dots n - d$ 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 $\textsc{Sieve}$ implem. almost on par with enumeration (within a factor 4 in dims 70–80), with still much room for improvements.

Table of Contents

Table of Contents

Sieving

Algorithm 1 $\textsc{Sieve}(\mathcal{L})$

$L \leftarrow$ 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, . . .].

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 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 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.

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

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

Pessimistic prediction for (1)

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

Optimistic prediction for (1)

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

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)

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

Optimistic prediction for (1)

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

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

Pessimistic prediction for (1)

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

Optimistic prediction for (1)

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

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

Pessimistic prediction for (1)

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

Optimistic prediction for (1)

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

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[3] 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 $2^{\Theta(n/\log n)}$

[3] 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[3] 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 $2^{\Theta(n/\log n)}$

[3] 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 $d = n-1, n-2, \dots, d_{\text{min}}$
- Hope that iteration $d_{\text{min}} + 1$ 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)

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

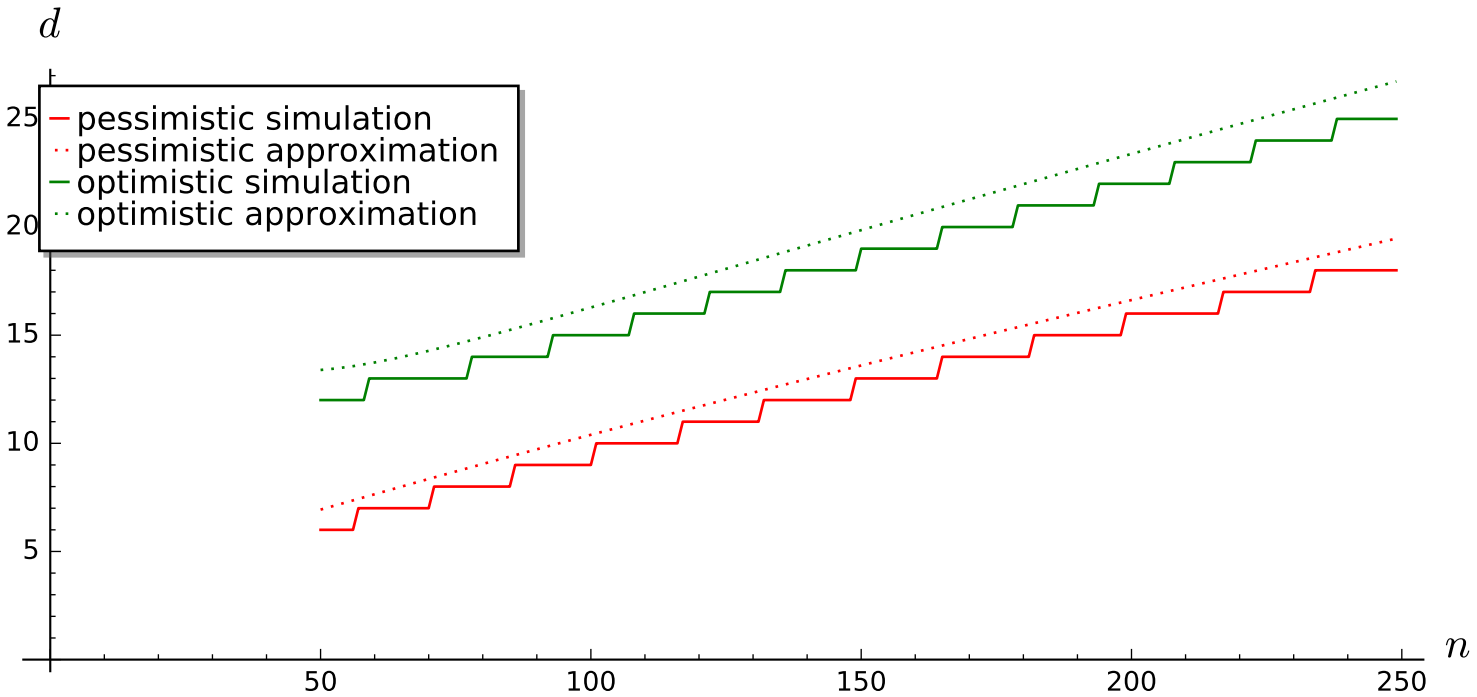


Figure: Predictions of the maximal successful choice of d_{min}.

Table of Contents

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 $\tilde{\mathbf{v}} = \textsc{Sign}(\mathbf{v}) \in \{0,1\}^k$
- 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 $\mathcal{L}(\mathbf{b}_1, \dots \mathbf{b}_{n-1})$
- Start the sieve in dimension n with many short-ish vectors
- Fresh vector 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 → V3)

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

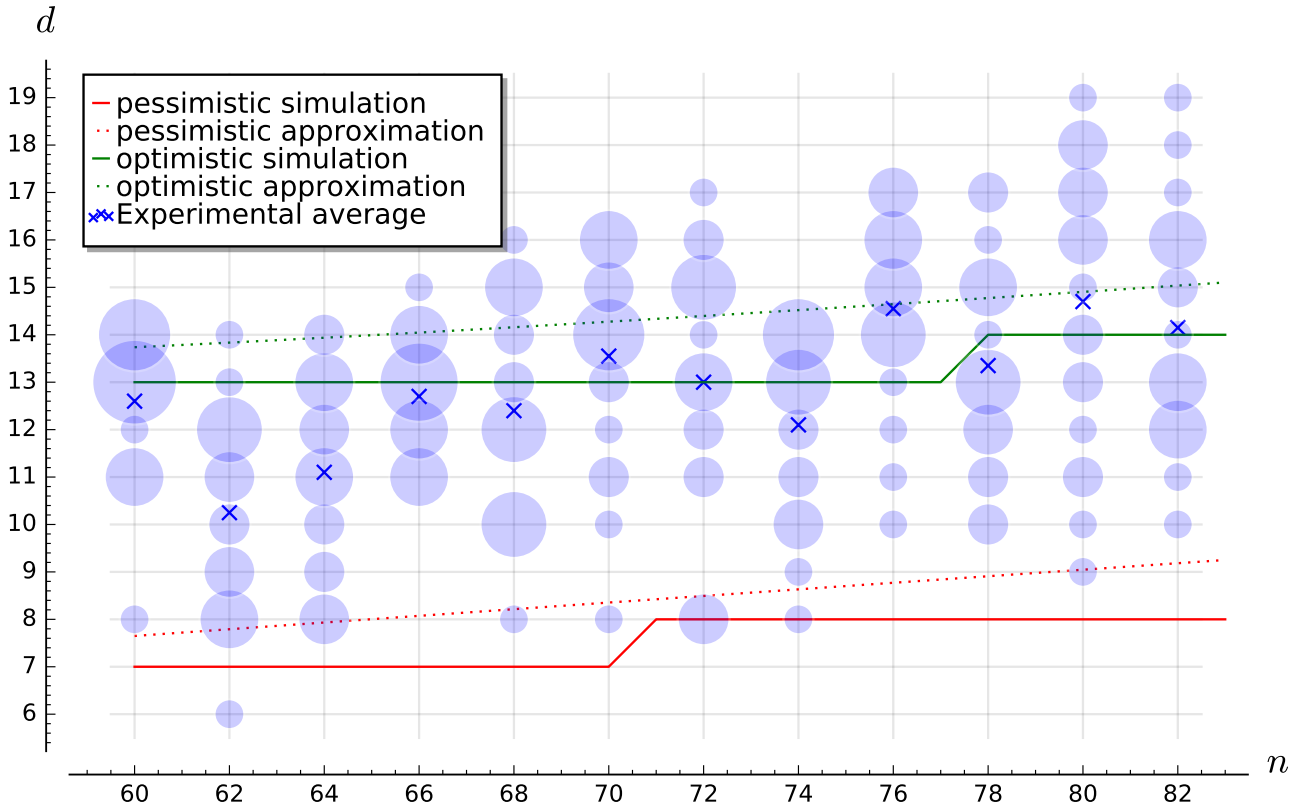
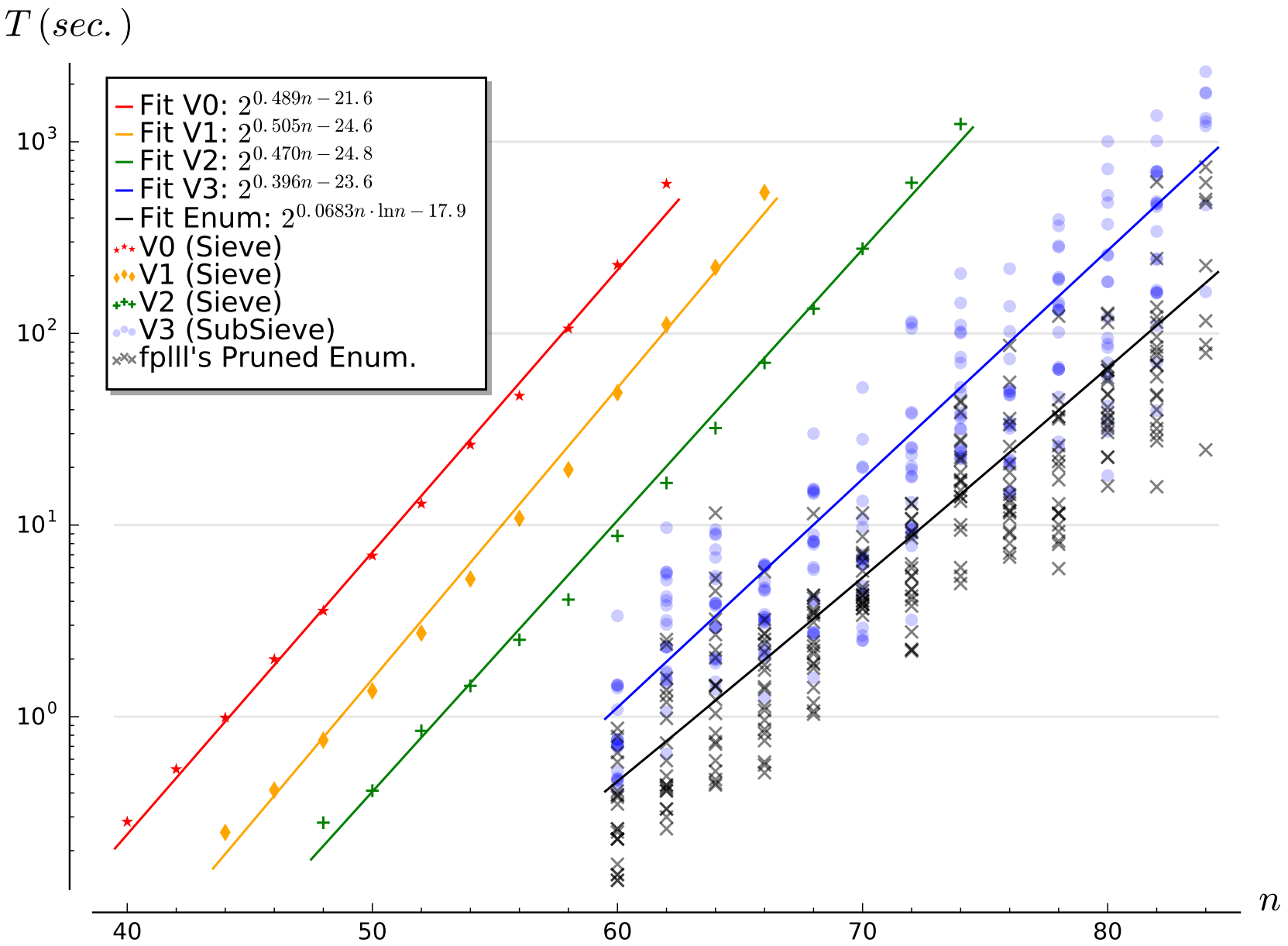


Figure: Predictions experiments for d_{min}.

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 ≈ 90 with further improvements (LSH/LSF, fine-tuning, vectorization, . . .)

To be continued. . .

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 ≈ 90 with further improvements (LSH/LSF, fine-tuning, vectorization, . . .)

To be continued. . .

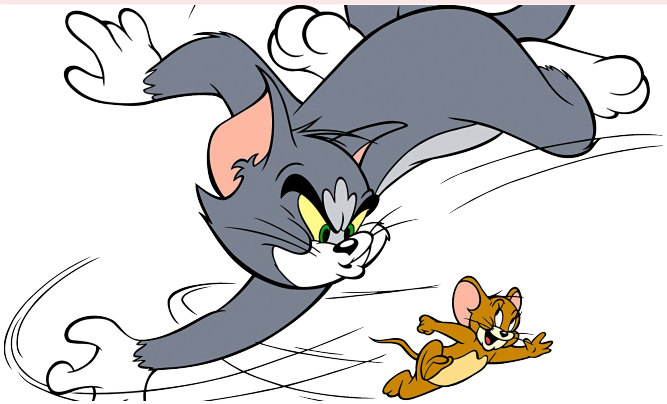
Summary

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 ≈ 90 with further improvements (LSH/LSF, fine-tuning, vectorization, . . .)



To be continued. . .

Thanks !

Question ?

Simple Sieving in the Area-Time Model

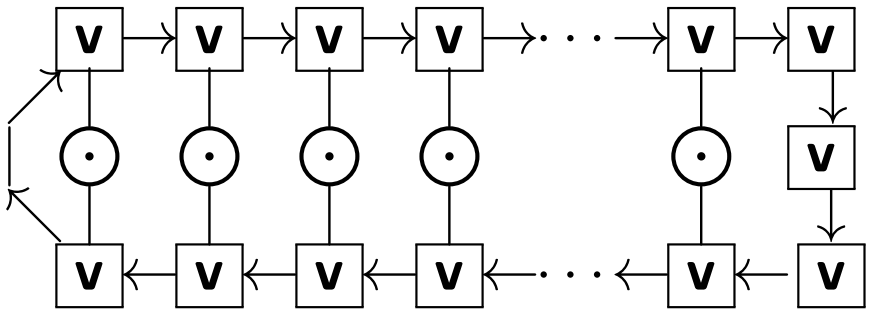
A claim of P. Kirchner[4]

A simple Sieving algorithm (e.g. NV) can be implemented with

$$A = 2^{.2075n+o(n)} \text{ and } T = 2^{.2075n+o(n)}.$$

The circuit is easy to re-invent, using shift registers:

- No long wires: no speed-of-light delays !
- Essentially 1-dimensional



[4] https://groups.google.com/forum/#!msg/cryptanalytic-algorithms/BoSRLOuHIjM/wAkZQlwRAgAJ

Advanced Sieving in the Area-Time Model

Conjecture / Open Question

There exist a sieving circuit with:

$$A = 2^{.2075n+o(n)} \text{ and } T \leq 2^{.142n+o(n)}.$$

Hint

- **[Becker Gama Joux 2015]** with only on level of filtration
- 3 or 4 layers of 2-dimensions should suffice.
- Keep shift-registers not fully saturated, for easier on-the-fly insertion.

Advanced Sieving in the Area-Time Model

Conjecture / Open Question

There exist a sieving circuit with:

$$A = 2^{.2075n+o(n)} \text{ and } T \leq 2^{.142n+o(n)}.$$

Hint

- **[Becker Gama Joux 2015]** with only on level of filtration
- 3 or 4 layers of 2-dimensions should suffice.
- Keep shift-registers not fully saturated, for easier on-the-fly insertion.