Tuple Lattice Sieving

Damien Stehlé

ENS de Lyon

Based on joint work with S. Bai and T. Laarhoven,
and on follow-up works of E. Kirshanova and G. Herold

lots of slides borrowed from G. Herold

The topic of this talk

The Shortest Vector Problem (SVP)

Input: $\mathbf{B} \in \mathbb{Z}^{n \times n}$ full rank.
Output: $\mathbf{s} \in \mathbf{B} \cdot \mathbb{Z}^n \setminus \mathbf{0}$ shortest.

Why do we consider this problem?

- Solving SVP is the costly component in cryptanalysis of lattice-based cryptosystems.
- Practical limitations of SVP solvers should drive the choice of concrete cryptographic parameters.
- And solving SVP is useful in plenty of other contexts!

The topic of this talk

The Shortest Vector Problem (SVP)

Input: $\mathbf{B} \in \mathbb{Z}^{n \times n}$ full rank.
Output: $\mathbf{s} \in \mathbf{B} \cdot \mathbb{Z}^n \setminus \mathbf{0}$ shortest.

Why do we consider this problem?

- Solving SVP is the costly component in cryptanalysis of lattice-based cryptosystems.
- Practical limitations of SVP solvers should drive the choice of concrete cryptographic parameters.
- And solving SVP is useful in plenty of other contexts!

The topic of this talk

The Shortest Vector Problem (SVP)

Input: $\mathbf{B} \in \mathbb{Z}^{n \times n}$ full rank.

Output: $\mathbf{s} \in \mathbf{B} \cdot \mathbb{Z}^n \setminus \mathbf{0}$ shortest.

Why do we consider this problem?

- Solving SVP is the costly component in cryptanalysis of lattice-based cryptosystems.
- Practical limitations of SVP solvers should drive the choice of concrete cryptographic parameters.
- And solving SVP is useful in plenty of other contexts!

The end goal...

Find which SVP solver is fastest for huge computational effort.

Which costs are we interested in?

- 2^{80} to 2^{160} bit operations.
- How much memory? Quantum resources?

Reminder: Proofs are over-rated!

Cryptanalysts are fine with heuristics

- Heuristic correctness
- Heuristic run-time
- Approximate solutions

But it should work in practice!

The end goal...

Find which SVP solver is fastest for huge computational effort.

Which costs are we interested in?

- 2^{80} to 2^{160} bit operations.
- How much memory? Quantum resources?

Reminder: Proofs are over-rated!

Cryptanalysts are fine with heuristics

- Heuristic correctness
- Heuristic run-time
- Approximate solutions

But it should work in practice!

The end goal...

Find which SVP solver is fastest for huge computational effort.

Which costs are we interested in?

- 2^{80} to 2^{160} bit operations.
- How much memory? Quantum resources?

Reminder: Proofs are over-rated!

Cryptanalysts are fine with heuristics

- Heuristic correctness
- Heuristic run-time
- Approximate solutions

But it should work in practice!

… and where we are today

It is not even clear which family of algorithms is the best.

Personal belief: sieving algorithms may be starting to win.

tuple sieving helps closing the gap

Talk based on:

S. Bai, T. Laarhoven, D. Stehlé: Tuple lattice sieving. ANTS'16.

G. Herold, E. Kirshanova: Improved algorithms for the approximate *k*-list problem in Euclidean norm. PKC'17.

G. Herold, E. Kirshanova, T. Laarhoven: Speed-ups and time-memory trade-offs for tuple lattice sieving. PKC'18.

... and where we are today

It is not even clear which family of algorithms is the best.

Personal belief: sieving algorithms may be starting to win.

tuple sieving helps closing the gap

Talk based on:

S. Bai, T. Laarhoven, D. Stehlé: Tuple lattice sieving. ANTS'16.

G. Herold, E. Kirshanova: Improved algorithms for the approximate k-list problem in Euclidean norm. PKC'17.

G. Herold, E. Kirshanova, T. Laarhoven: Speed-ups and time-memory trade-offs for tuple lattice sieving. PKC'18.

Roadmap

1. **Background**
2. Solving SVP by sieving
3. Tuple sieving
4. Fast tuple sieving

Best known fully analyzed algorithms

SVP

Input: $\mathbf{B} \in \mathbb{Z}^{n \times n}$ a basis matrix of $\Lambda = \mathbf{B} \in \mathbb{Z}^n$.
Output: $\mathbf{s} \in \Lambda \setminus \mathbf{0}$ shortest.

	Time upper bound	Space upper bound	Deterministic or Probabilistic
via enumeration [FiPo'83,Kan'83,HaSt'07]	$n^{n/(2e)+o(n)}$	$\mathcal{P}oly(n)$	Deterministic
via sieving [AjKuSi'01, MiVo'10, PuSt'09]	$2^{2.247n+o(n)}$	$2^{1.325n+o(n)}$	Probabilistic
via Voronoi cell [MiVo'10]	$2^{2n+o(n)}$	$2^{n+o(n)}$	Deterministic
Gaussians [ADRS'16,AS'17]	$2^{n+o(n)}$	$2^{n+o(n)}$	Probabilistic

Heuristic algorithms, prior to tuple sieving

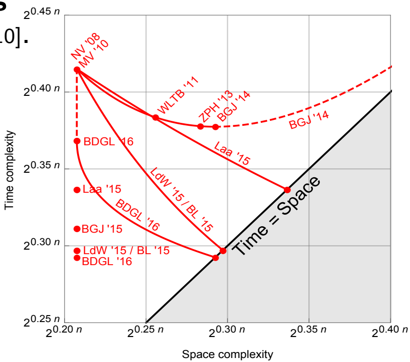
Enumeration with **pre-processing** [Kan'83] and **extreme pruning** [GNR'10].

Heuristic algorithms, prior to tuple sieving

Enumeration with **pre-processing** [Kan'83] and **extreme pruning** [GNR'10].

Sieving without **perturbations** and with **locality sensitive hashing**.

[Figure courtesy of T. Laarhoven]



Tuple sieving

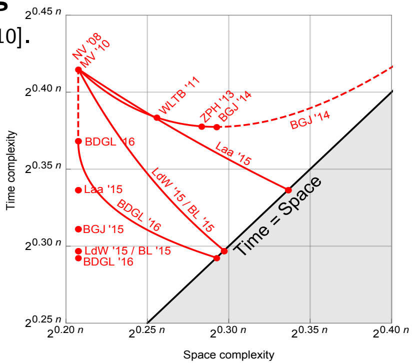
Beats the "Space = $2^{0.207n}$" boundary.
While keeping a $2^{O(n)}$ time complexity.
And this increases practical performance!

Heuristic algorithms, prior to tuple sieving

Enumeration with **pre-processing** [Kan'83] and **extreme pruning** [GNR'10].

Sieving without **perturbations** and with **locality sensitive hashing**.

[Figure courtesy of T. Laarhoven]



Tuple sieving

Beats the "Space $= 2^{0.207n}$" boundary.
While keeping a $2^{O(n)}$ time complexity.
And this increases practical performance!

In practice

Enumeration with **extreme pruning** and **pre-processing**.

SVP-challenge webpage (Darmstadt Crypto Group)

- K. Kashiwabara, M. Fukase and T. Teruya, up to $n = 150$ in ≈ 500 core years
- Y. Aono and P. Nguyen, up to $n = 130$ in ≈ 160 core days
- lots of others, based on enumeration
- T. Kleinjung, up to $n = 116$... using "sieving"

In practice

Enumeration with **extreme pruning** and **pre-processing**.

SVP-challenge webpage (Darmstadt Crypto Group)

- K. Kashiwabara, M. Fukase and T. Teruya,
  up to $n = 150$ in ≈ 500 core years
- Y. Aono and P. Nguyen,
  up to $n = 130$ in ≈ 160 core days
- lots of others, based on enumeration
- T. Kleinjung, up to $n = 116$... using "sieving"

In practice

Enumeration with **extreme pruning** and **pre-processing**.

SVP-challenge webpage (Darmstadt Crypto Group)

- K. Kashiwabara, M. Fukase and T. Teruya,
  up to $n = 150$ in ≈ 500 core years
- Y. Aono and P. Nguyen,
  up to $n = 130$ in ≈ 160 core days
- lots of others, based on enumeration
- T. Kleinjung, up to $n = 116$... using "sieving"

In practice

Enumeration with **extreme pruning** and **pre-processing**.

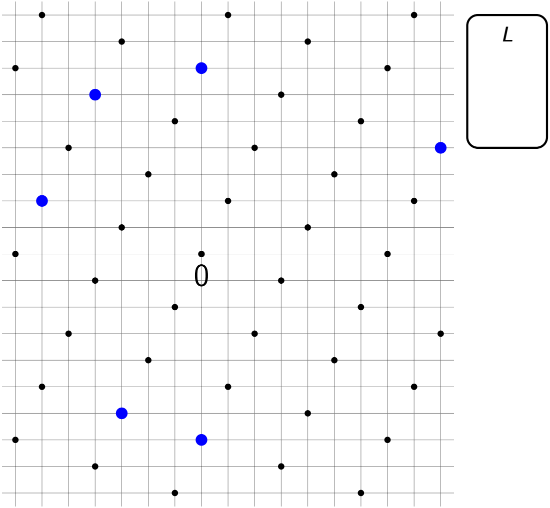
SVP-challenge webpage (Darmstadt Crypto Group)

- K. Kashiwabara, M. Fukase and T. Teruya,
  up to $n = 150$ in ≈ 500 core years
- Y. Aono and P. Nguyen,
  up to $n = 130$ in ≈ 160 core days
- lots of others, based on enumeration
- T. Kleinjung, up to $n = 116$... using "sieving"

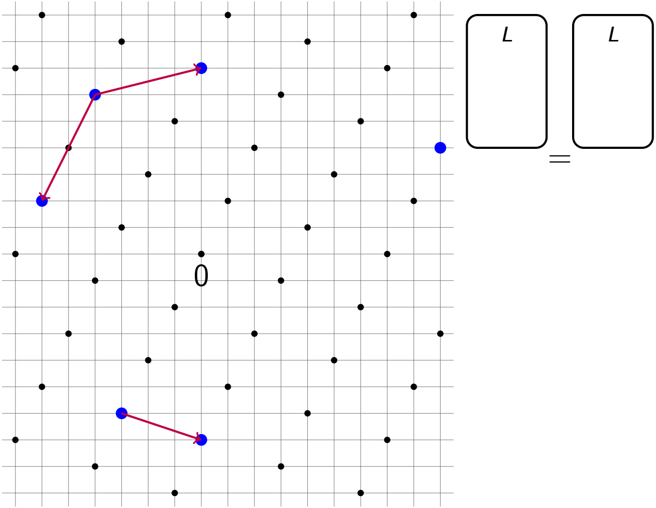
Roadmap

1. Background
2. **Solving SVP by sieving**
3. Tuple sieving
4. Fast tuple sieve

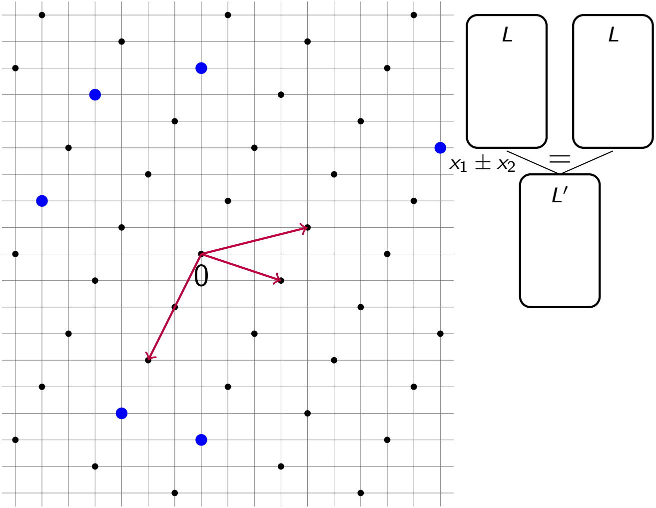
The sieving algorithm [Figure courtesy of G. Herold]



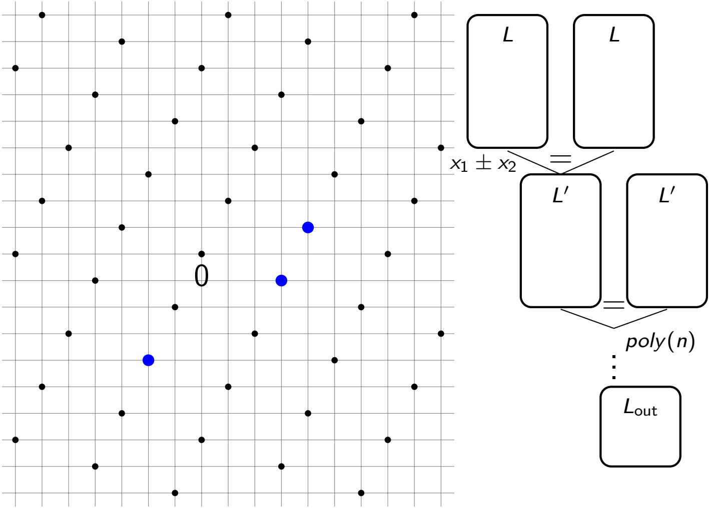
The sieving algorithm [Figure courtesy of G. Herold]



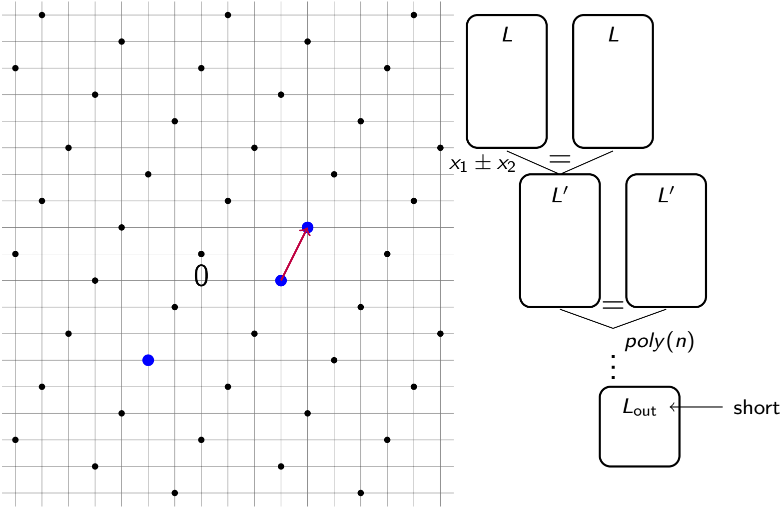
The sieving algorithm [Figure courtesy of G. Herold]



The sieving algorithm [Figure courtesy of G. Herold]



The sieving algorithm [Figure courtesy of G. Herold]



Analysis of sieving

Correctness: fingers crossed!

For the cost, it suffices to bound the list size:

$$\text{Time} \leq |L|^2 \cdot \mathcal{P}oly(n).$$

It suffices to bound how many points there can be

- with angle $\geq \pi/3$ between each other
  (else the point is passed to the next list)
- with essentially the same Euclidean norm
  (consider $\mathcal{P}oly$ coronas)

Analysis of sieving

Correctness: fingers crossed!

For the cost, it suffices to bound the list size:

$$\text{Time} \quad \leq \quad |L|^2 \cdot \mathcal{P}oly(n).$$

It suffices to bound how many points there can be

- with angle $\geq \pi/3$ between each other
  (else the point is passed to the next list)
- with essentially the same Euclidean norm
  (consider $\mathcal{P}oly$ coronas)

Cost of sieving

It suffices to bound how many points there can be

- with angle $\geq \pi/3$ between each other
- with essentially the same Euclidean norm

The fraction of the n-sphere $\mathcal{S}_n$ at angle $\leq \pi/3$ from a given point is $\approx (\sin(\pi/3))^{-n}$.

Assuming that caps do not intersect much:

$$\begin{aligned} \text{Memory} &\leq \sqrt{4/3}^n &\leq 2^{0.208n} \\ \text{Time} &\leq (4/3)^n &\leq 2^{0.416n} \end{aligned}$$

Cost of sieving

It suffices to bound how many points there can be

- with angle $\geq \pi/3$ between each other
- with essentially the same Euclidean norm

The fraction of the n-sphere $\mathcal{S}_n$ at angle $\leq \pi/3$ from a given point is $\approx (\sin(\pi/3))^{-n}$.

Assuming that caps do not intersect much:

$$\begin{aligned} \text{Memory} &\leq \sqrt{4/3}^{\,n} &&\leq 2^{0.208n} \\ \text{Time} &\leq (4/3)^n &&\leq 2^{0.416n} \end{aligned}$$

Roadmap

1. Background
2. Solving SVP by sieving
3. **Tuple sieving**
4. Fast tuple sieve

2-Sieve vs. k-Sieve

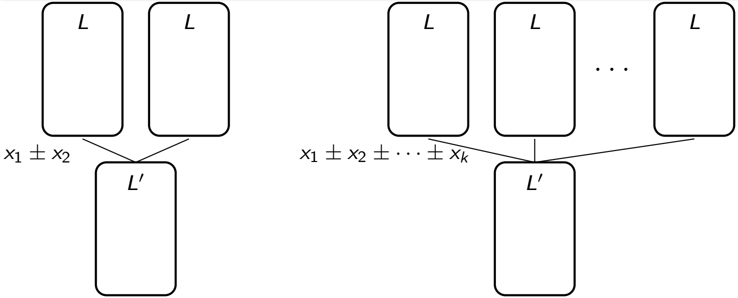
k-Sieve [BLS16]

Consider sums of $k > 2$ vectors at once

2-Sieve vs. k-Sieve

k-Sieve [BLS16]

Consider sums of $k > 2$ vectors at once



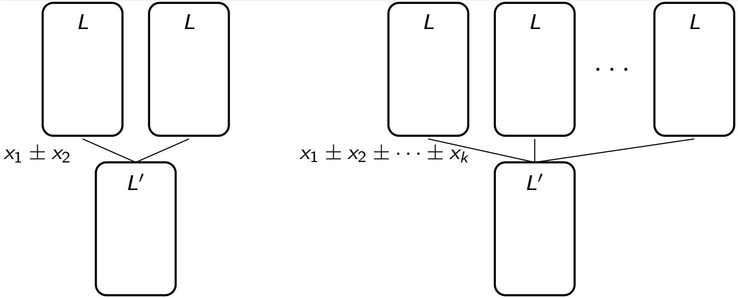
Aim

- Each point is more useful $\Rightarrow$ memory decreases
- Finding useful tuples is more expensive $\Rightarrow$ time increases

2-Sieve vs. k-Sieve

k-Sieve [BLS16]

Consider sums of $k > 2$ vectors at once



Aim

- Each point is more useful $\Rightarrow$ memory decreases
- Finding useful tuples is more expensive $\Rightarrow$ time increases

The k-list problem

k-List problem (informal)

Input. k lists $L_1, \ldots, L_k$, whose entries are iid. uniformly chosen vectors from the n-sphere $\mathcal{S}_n$.
Task. Output all k-tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in L_1 \times \ldots \times L_k$ st

$$\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\| \leq 1.$$

(in our case: $L_1 = L_2 = \ldots = L_k = L$)

- List size (heuristically) determined by

$$|L| = |L|^k \cdot \Pr\left[\|\mathbf{x}_1 \pm \ldots \pm \mathbf{x}_k\| \leq 1\right]$$

- Cost of naive algorithm: $|L|^k$.

The $k = 2$ analysis can be extended [BLS16], but it's not very insightful.

The k-list problem

k-List problem (informal)

Input. k lists $L_1, \ldots, L_k$, whose entries are iid. uniformly chosen vectors from the n-sphere $\mathcal{S}_n$.
Task. Output all k-tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in L_1 \times \ldots \times L_k$ st

$$\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\| \leq 1.$$

(in our case: $L_1 = L_2 = \ldots = L_k = L$)

- List size (heuristically) determined by

$$|L| = |L|^k \cdot \Pr\left[\|\mathbf{x}_1 \pm \ldots \pm \mathbf{x}_k\| \leq 1\right]$$

- Cost of naive algorithm: $|L|^k$.

The $k = 2$ analysis can be extended [BLS16], but it's not very insightful.

Configurations [HK17]

Task. Find $\mathbf{x}_1, \ldots, \mathbf{x}_k \in L1 \times \ldots \times L_k$ st. $\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\| \leq 1$.

We only care about the positions of the $\mathbf{x}_1, \ldots, \mathbf{x}_k$ relative to each other.

Definition (Configuration)

The configuration $C = C(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ of $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathcal{S}_n$ is defined as the Gram matrix $C = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle)_{i,j}$.

Configurations [HK17]

Task. Find $\mathbf{x}_1, \ldots, \mathbf{x}_k \in L1 \times \ldots \times L_k$ st. $\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\| \leq 1$.

We only care about the positions of the $\mathbf{x}_1, \ldots, \mathbf{x}_k$ relative to each other.

Definition (Configuration)

The configuration $C = C(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ of $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathcal{S}_n$ is defined as the Gram matrix $C = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle)_{i,j}$.

Configuration C is positive semi-definite, with $C_{ii} = 1$, and:

$$\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\|^2 = \sum_{i,j} C_{i,j}.$$

Distribution of Configurations

Wishart'28

Let $\mathbf{x}_1, \ldots, \mathbf{x}_k$ be iid uniform on $\mathcal{S}_n$. Then the Gram matrix $C = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle)_{i,j}$ follows a distribution with pdf

$$W_{n,k} \cdot \det(C)^{\frac{1}{2}(n-k)} \, \mathrm{d}C = \widetilde{O}\left(\det(C)^{\frac{n}{2}}\right) \, \mathrm{d}C \ ,$$

where $W_{n,k}$ is a normalization constant.

The distribution of C is very concentrated.

Only one configuration matters!

The distribution of C is very concentrated.
$\Rightarrow$ Essentially all solutions come from a single C.

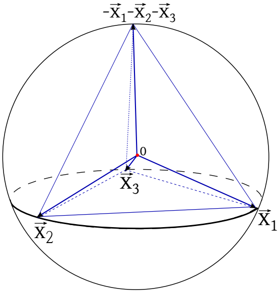


Figure: The configuration of solutions is concentrated on the configuration with maximal symmetry.

Memory cost

For iid uniform $\mathbf{x}_1, \ldots, \mathbf{x}_k$ on $\mathcal{S}_n$, we have

$$\begin{aligned} \Pr[\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\| \leq 1] &= \Pr[\forall i \neq j : \langle \mathbf{x}_i, \mathbf{x}_j \rangle \approx -1/k] \\ &= \left(\frac{(k+1)^{k-1}}{k^k} \right)^{\frac{n}{2}}. \end{aligned}$$

List size

For the balanced configuration, we need lists of size

$$|L| = \widetilde{O}\left(\left(\frac{k^{\frac{k}{k-1}}}{k+1} \right)^{\frac{n}{2}} \right).$$

$k = 2 \; : \; 2^{0.207n}$ $\qquad$ $k = 3 \; : \; 2^{0.189n}$

Memory cost

For iid uniform $\mathbf{x}_1, \ldots, \mathbf{x}_k$ on $\mathcal{S}_n$, we have

$$\begin{aligned} \Pr[\|\mathbf{x}_1 + \ldots + \mathbf{x}_k\| \leq 1] &= \Pr[\forall i \neq j : \langle \mathbf{x}_i, \mathbf{x}_j \rangle \approx -1/k] \\ &= \left(\frac{(k+1)^{k-1}}{k^k} \right)^{\frac{n}{2}}. \end{aligned}$$

List size

For the balanced configuration, we need lists of size

$$|L| = \widetilde{O}\left(\left(\frac{k^{\frac{k}{k-1}}}{k+1} \right)^{\frac{n}{2}} \right).$$

$k = 2 \ : \ 2^{0.207n} \qquad k = 3 \ : \ 2^{0.189n}$

How to find the solutions?

The distribution of C is very concentrated.

Finding almost all solutions to the k-list problem is equivalent to finding all $\mathbf{x}_1, \ldots, \mathbf{x}_k \in L_1 \times \ldots \times L_k$ st. $C(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ is close to the target concentration:

$$\forall i \neq j \; : \; \langle \mathbf{x}_i, \mathbf{x}_j \rangle = -1/k.$$

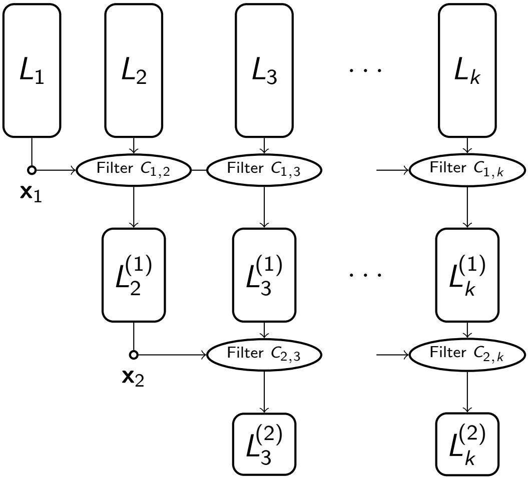
How to find the solutions?

The distribution of C is very concentrated.

Finding almost all solutions to the k-list problem is equivalent to finding all $\mathbf{x}_1, \ldots, \mathbf{x}_k \in L_1 \times \ldots \times L_k$ st. $C(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ is close to the target concentration:

$$\forall i \neq j \ : \ \langle \mathbf{x}_i, \mathbf{x}_j \rangle = -1/k.$$

The Herold-Kirshanova algorithm



Triple sieve beats double sieve!

Double sieve

Memory: $2^{0.207n}$ Time: $2^{0.415n}$

Triple sieve

Memory: $2^{0.189n}$ Time: $2^{0.397n}$

$k = 4$ is slower

Memory: $2^{0.173n}$ Time: $2^{0.424n}$

Triple sieve beats double sieve!

Double sieve

Memory: $2^{0.207n}$ Time: $2^{0.415n}$

Triple sieve

Memory: $2^{0.189n}$ Time: $2^{0.397n}$

$k = 4$ is slower

Memory: $2^{0.173n}$ Time: $2^{0.424n}$

Roadmap

1. Background
2. Solving SVP by sieving
3. Tuple sieving
4. **Faster tuple sieve** [HKL18]

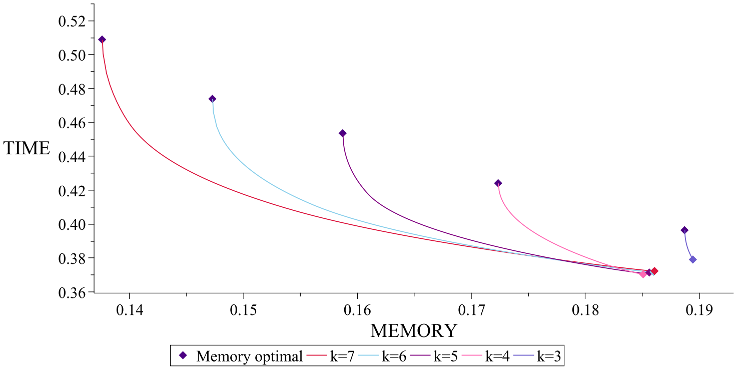
Alternative target configurations

Increase the list size.

⇒ exponentially more good k-tuples.

⇒ We only need to find an exponential fraction of solutions.

Consider unbalanced configurations C that are easier to find.



Locality sensitive hashing and filtering

Clever lists

Pre-process L, such that it becomes easier to find all $\mathbf{x}_2 \in L$ with $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle \approx c$, for a given $\mathbf{x}_1$.

Locality sensitive hash functions and filters

Hash functions $h \in \mathcal{H}$ st:

- Close points are likely to collide
- Far away points are unlikely to collide

Filters: a point may end up in more than one bucket

Locality sensitive hashing and filtering

Clever lists

Pre-process L, such that it becomes easier to find all $\mathbf{x}_2 \in L$ with $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle \approx c$, for a given $\mathbf{x}_1$.

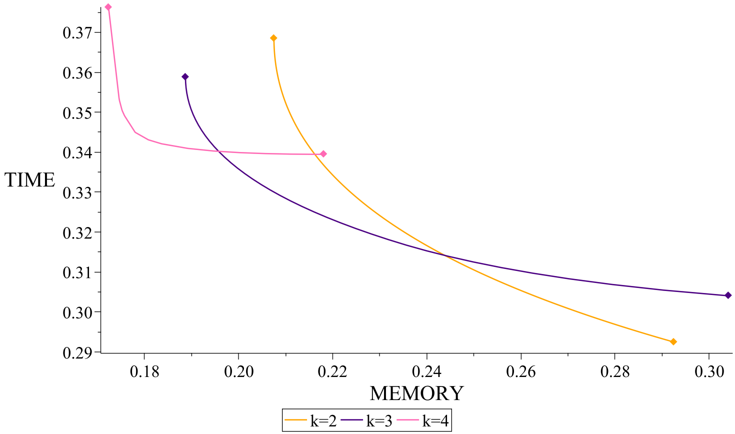
Locality sensitive hash functions and filters

Hash functions $h \in \mathcal{H}$ st:

- Close points are likely to collide
- Far away points are unlikely to collide

Filters: a point may end up in more than one bucket

Time-memory trade-offs with both techniques



And in practice?

- ListSieve vs GaussSieve
- Variable configurations clearly help
- Locality-sensitive filtering does not (yet?)

Roadmap

1. Background
2. Solving SVP by sieving
3. Tuple sieving
4. Faster tuple sieve

Take-home message

The $\sqrt{4/3}^n$ memory barrier is broken.

Frodo's paranoia [ADPS16]

"Because all those algorithms require classically building lists of size $\sqrt{4/3}^n$, it is very plausible that the best quantum SVP algorithm would run in time $\geq 2^{0.2075n}$."

One of the SVP scenarios considered for setting parameters in lattice-based cryptography.

The lower bound may be correct, but the underlying justification is invalidated by tuple sieve.

Sounder approaches:

- Asymptotic cost of the best known algorithm
- Extrapolation of well-understood practice

Take-home message

The $\sqrt{4/3}^n$ memory barrier is broken.

Frodo's paranoia [ADPS16]

"Because all those algorithms require classically building lists of size $\sqrt{4/3}^n$, it is very plausible that the best quantum SVP algorithm would run in time $\geq 2^{0.2075n}$."

One of the SVP scenarios considered for setting parameters in lattice-based cryptography.

The lower bound may be correct, but the underlying justification is invalidated by tuple sieve.

Sounder approaches:

- Asymptotic cost of the best known algorithm
- Extrapolation of well-understood practice

Take-home message

The $\sqrt{4/3}^n$ memory barrier is broken.

Frodo's paranoia [ADPS16]

"Because all those algorithms require classically building lists of size $\sqrt{4/3}^n$, it is very plausible that the best quantum SVP algorithm would run in time $\geq 2^{0.2075n}$."

One of the SVP scenarios considered for setting parameters in lattice-based cryptography.

The lower bound may be correct, but the underlying justification is invalidated by tuple sieve.

Sounder approaches:

- Asymptotic cost of the best known algorithm
- Extrapolation of well-understood practice

Take-home message

The $\sqrt{4/3}^n$ memory barrier is broken.

Frodo's paranoia [ADPS16]

"Because all those algorithms require classically building lists of size $\sqrt{4/3}^n$, it is very plausible that the best quantum SVP algorithm would run in time $\geq 2^{0.2075n}$."

One of the SVP scenarios considered for setting parameters in lattice-based cryptography.

The lower bound may be correct, but the underlying justification is invalidated by tuple sieve.

Sounder approaches:

- Asymptotic cost of the best known algorithm
- Extrapolation of well-understood practice

(When) will sieving outperform enumeration?

- Sieving is trickier, at least with our current comprehension
  - Practice and asymptotics do not match
  - Locality-sensitive hashing is too costly
  - Parallelism
- For cryptanalysis, sieving is important via BKZ
  - One interested in projected sublattices of an already quite reduced basis
  - [Duc17]: Sieving can handle these much faster

Or is enumeration just the best for cryptanalytic costs?

(When) will sieving outperform enumeration?

- Sieving is trickier, at least with our current comprehension
  - Practice and asymptotics do not match
  - Locality-sensitive hashing is too costly
  - Parallelism
- For cryptanalysis, sieving is important via BKZ
  - One interested in projected sublattices of an already quite reduced basis
  - [Duc17]: Sieving can handle these much faster

Or is enumeration just the best for cryptanalytic costs?

(When) will sieving outperform enumeration?

- Sieving is trickier, at least with our current comprehension
  - Practice and asymptotics do not match
  - Locality-sensitive hashing is too costly
  - Parallelism
- For cryptanalysis, sieving is important via BKZ
  - One interested in projected sublattices of an already quite reduced basis
  - [Duc17]: Sieving can handle these much faster

Or is enumeration just the best for cryptanalytic costs?

THANK YOU!