Lattice sieving via quantum random walks

Johanna Loyer
Joint work with André Chailloux

Overview

1. Preliminaries
   - Lattices
   - Locality sensitive filtering (LSF)
   - Quantum Computing
2. Our algorithm
3. Complexity and space/time trade-offs

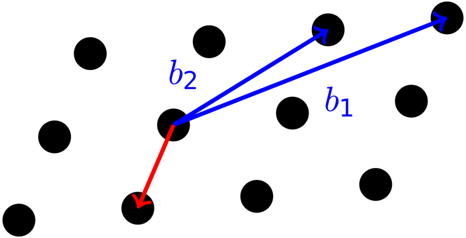
Lattice and SVP

Lattice

The d-dimensional lattice $\mathcal{L} \subset \mathbb{R}^m$ generated by the basis $B = (\vec{b_1}, \ldots, \vec{b_d})$ with $\forall i, \vec{b_i} \in \mathbb{R}^m$ is the set of all integer linear combinations of its basis vectors: $\mathcal{L}(B) = \left\{ \sum_{i=1}^{d} \lambda_i \vec{b_i},\ \lambda_i \in \mathbb{Z} \right\}$.

Shortest Vector Problem (SVP)

Given a lattice $\mathcal{L}$, find the shortest non-zero vector $\vec{v} \in \mathcal{L}$, ie. st. $||\vec{v}|| = \inf\left\{ ||\vec{u}|| \neq 0,\ \vec{u} \in \mathcal{L} \right\}$.



Why do we want to solve SVP?

Cryptography

- NP-hard problem, hard in average.
- Problems derived from SVP: SIS, LWE, NTRU...
- Quantum-resistant cryptosystems based on them: Dilithium, FALCON, NTRU, Kyber, SABER.

Why do we want to solve SVP?

Cryptography

- NP-hard problem, hard in average.
- Problems derived from SVP: SIS, LWE, NTRU...
- Quantum-resistant cryptosystems based on them: Dilithium, FALCON, NTRU, Kyber, SABER.

Cryptanalysis

- Broken if a reduced basis of the lattice can be found.
- BKZ algorithm finds a reduced basis.
- Solving SVP = subroutine of BKZ

$\Rightarrow$ The security of these cryptosystems directly relies on the complexity of solving SVP.

Sieving

SVP-solving methods

- Main practical methods: enumeration and sieving.
- Run in exponential time.

Main heuristic: Lattice vectors acts as random vectors.

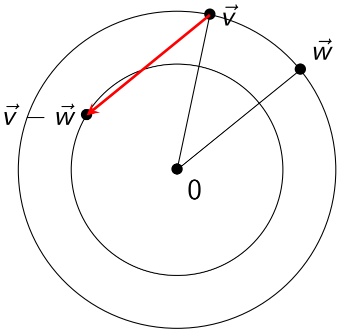
- Implies that vectors of norm at most R are lying on the border of $R \cdot \mathcal{S}^d$, with $R \cdot \mathcal{S}^d := \{\vec{x} \in \mathbb{R}^d : \|\vec{x}\| \leqslant R\}$.
- Validated by experiments.

Sieving

Nguyen-Vidick Sieve (NV-sieve) [NV08]

Input: list L of N lattice vectors of norm at most R ; $\gamma < 1$.
Output: list L' of N lattice vectors of norm at most $\gamma R < R$.
for $(\vec{v}, \vec{w}) \in L$:
    if $\|\vec{v} - \vec{w}\| \leqslant \gamma R$: add $\vec{v} - \vec{w}$ to L'

Sphere of dimension d and radius R.

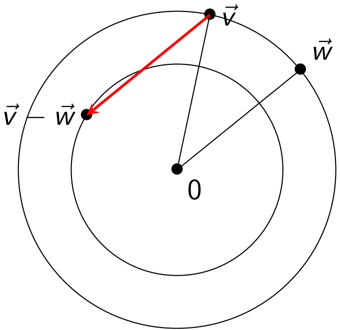


Sieving

Nguyen-Vidick Sieve (NV-sieve) [NV08]

Input: list L of N lattice vectors of norm at most R ; $\gamma < 1$.
Output: list L' of N lattice vectors of norm at most $\gamma R < R$.
for $(\vec{v}, \vec{w}) \in L$:
    if $\|\vec{v} - \vec{w}\| \leqslant \gamma R$: add $\vec{v} - \vec{w}$ to L'

Sphere of dimension d and radius R.



If $\vec{v}, \vec{w} \in \mathcal{L}$ then $\vec{v} - \vec{w} \in \mathcal{L}$.

For $\gamma \to 1$ and $\vec{v}, \vec{w} \in R \cdot \mathcal{S}^d$, $\|\vec{v} - \vec{w}\| \leqslant \gamma R \Leftrightarrow \theta(\vec{v}, \vec{w}) \leqslant \frac{\pi}{3}$.

Sieving - Solving SVP

Solve SVP by sieving

Input: a lattice $\mathcal{L}$ of basis $(\vec{b}_1, ..., \vec{b}_d)$
Output: a shortest vector of $\mathcal{L}$ (probably)
$L \leftarrow$ generate $N = (4/3)^{d/2+o(d)}$ lattice vectors ▷ by Klein's algorithm
while L does not contain a short vector :
  $L \leftarrow$ **NV-sieve step**$(L, \gamma \to 1)$
return min(L)

1st iteration: norm γR
2nd iteration: $\gamma^2 R$
⋮
poly(d)-th iteration: $\gamma^{poly(d)} R$

Complexity: $N^2 = 2^{0.415d+o(d)}$ time and $N = 2^{0.208d+o(d)}$ space.

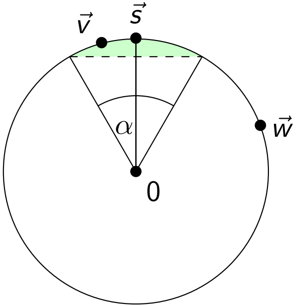
LSF (Locality Sensitive Filtering)

Improvement of the NV-sieve: only check pairs of close vectors.

Filter

A **filter** $f_{\vec{s},\alpha}$ of center $\vec{s} \in \mathbb{R}^d$ and angle $\alpha \in [0, \pi/2]$ maps a vector $\vec{v}$ to a boolean value:

- 1 if $\theta(\vec{v}, \vec{s}) \leqslant \alpha$,
- 0 else.



LSF (Locality Sensitive Filtering)

NV-sieve with LSF

1. Generate filters all over the sphere. $\triangleright$ centers $=$ words from a code
2. Add each vector to its nearest filters of angle at most α. $\triangleright$ list decoding algorithm
3. For each vector : search a reducing one within its filters (instead of in the whole list).
   - Classically or by Grover's search

Complexity ($2^{0.208d+o(d)}$ space):
Original NV-sieve [NV08]: $2^{0.415d+o(d)}$ time.
Classic with LSF [BDGL16]: $2^{0.292d+o(d)}$ time.
Quantum with LSF [Laa16]: $2^{0.265d+o(d)}$ time.

Grover's algorithm

Input: $x_1, ..., x_n \in E^d$ and a function $f : E^d \to \{0, 1\}$.
Output: $i \in [|1, n|]$ such that $f(x_i) = 1$.

Time complexity: $O(\sqrt{n})$.

Quantum Random Walk

Input: a graph $G = (V, E)$,
a function $f : V \to \{0, 1\}$ with $f(v) = 1 \Leftrightarrow v$ is a "**marked**" vertex.
Output: a marked vertex $v \in V$.

(Will be illustrated further with an example.)

Our algorithm

NV-sieve using quantum random walks

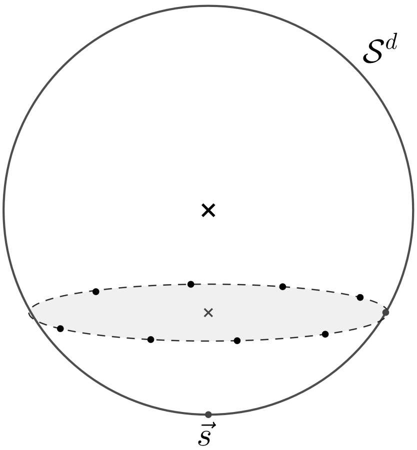
Input: list L of N lattice vectors of norm at most R ; $\gamma < 1$.
Output: list L' of N lattice vectors of norm at most $\gamma R < R$.

Main idea: Replace Grover's search by a quantum random walk.

Step 1

Sample a code C and generate the α-filters.

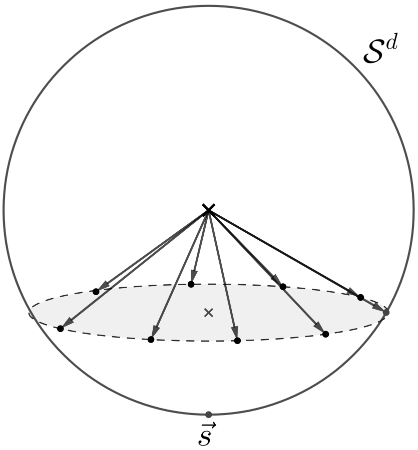
Insert each list vector in its (unique) nearest α-filter. $\triangleright$ N^{c_α} vectors per α-filter. $c_\alpha \in [0, 1]$



Step 1

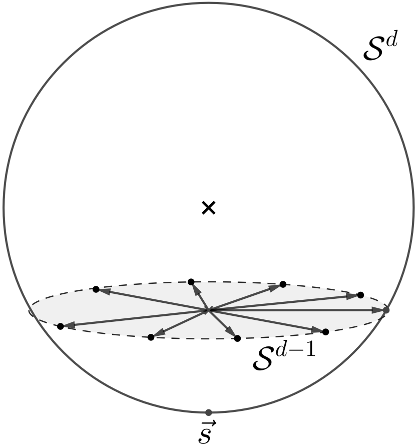
Sample a code C and generate the α-filters.

Insert each list vector in its (unique) nearest α-filter. $\triangleright$ N^{c_α} vectors per α-filter. $c_\alpha \in [0, 1]$



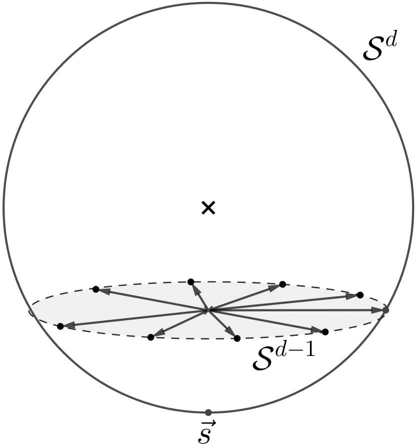
Step 1

Sample a code C and generate the α-filters.
Insert each list vector in its (unique) nearest α-filter. $\triangleright$ N^{c_α} vectors per α-filter. $c_\alpha \in [0, 1]$



Step 1

Sample a code C and generate the α-filters.
Insert each list vector in its (unique) nearest α-filter. $\triangleright$ N^{c_α} vectors per α-filter. $c_\alpha \in [0, 1]$



For $\vec{v}, \vec{w} \in \mathcal{S}^d$ and their residual vectors $\vec{v}_R, \vec{w}_R \in \mathcal{S}^{d-1}$,

$$\theta(\vec{v}, \vec{w}) \leqslant \frac{\pi}{3} \Leftrightarrow \theta(\vec{v}_R, \vec{w}_R) \leqslant \theta^*_\alpha.$$

Step 2

For each α-filter :

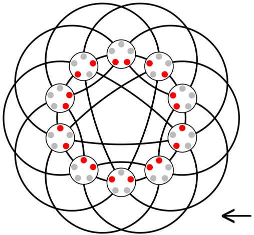
1. Vertex : Choose randomly N^{c_V} vectors from the α-filter.
2. Sample a code C' and generate the β-filters.
   Insert each Vertex's vector in its nearest β-filter.
3. Perform Quantum Random Walks to find all the reducing pairs in the α-filter.

Quantum Random Walk

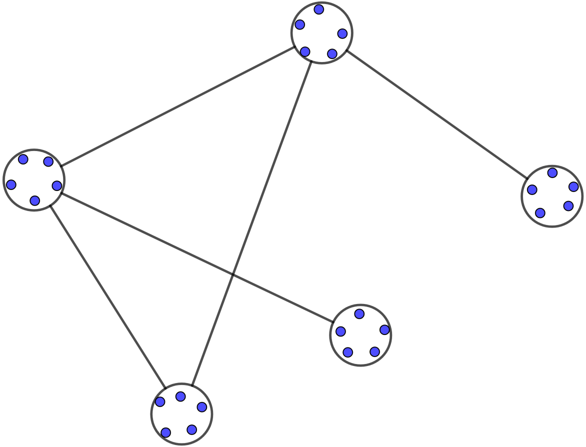
Input: a graph $G = J(N^{c_\alpha}, N^{c_V})$,
a vertex is marked iff. contains a pair of angle at most θ^*_α.
Output: a marked vertex.

Johnson's graph $J(N^{c_\alpha}, N^{c_V})$:

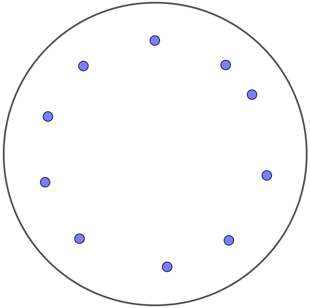
- Vertexes V: set of N^{c_V} from the N^{c_α} of the current α-filter.
- Edges E: 2 vertexes are neighbors iff. they differ by exactly 1 vector.

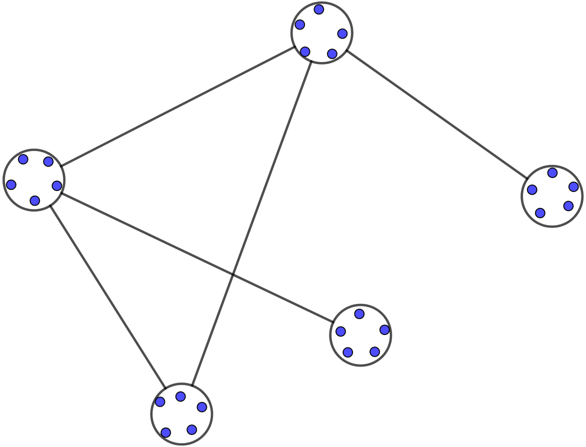


← Johnson's graph $J(5,2)$

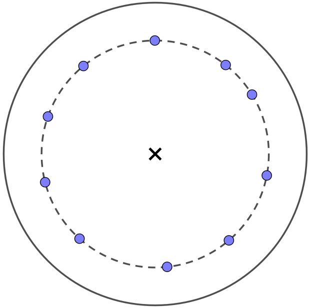


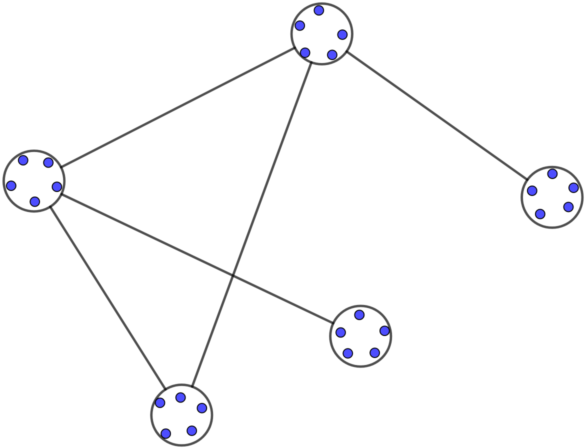
Zoom on the current vertex



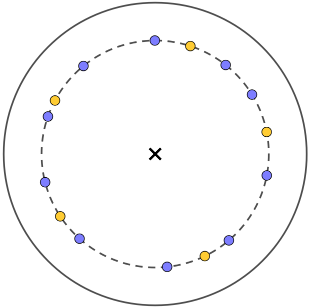


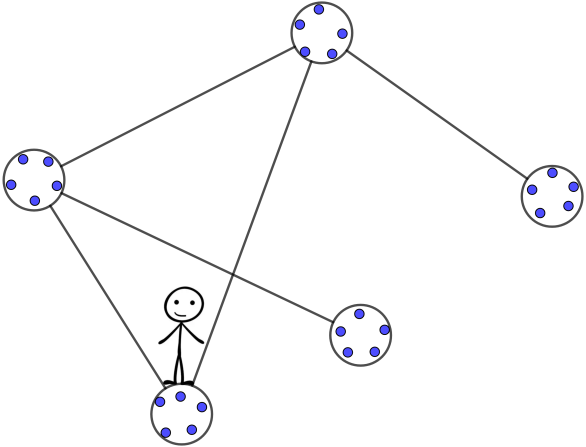
Zoom on the current vertex



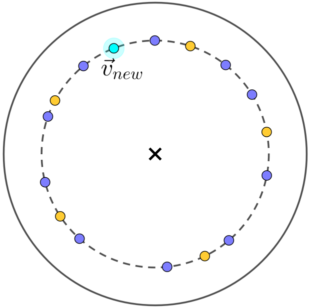


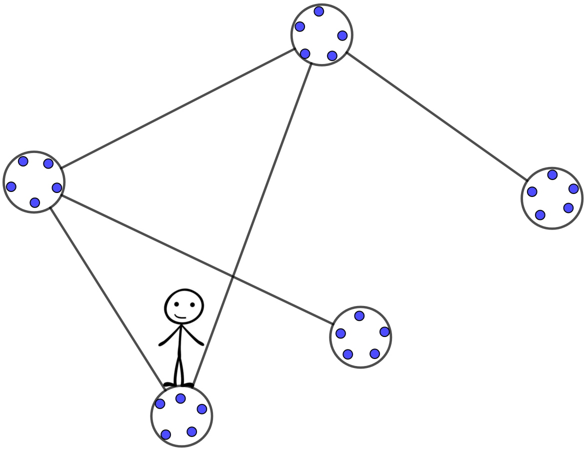
Zoom on the current vertex



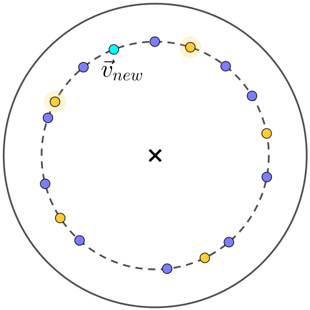


Zoom on the current vertex

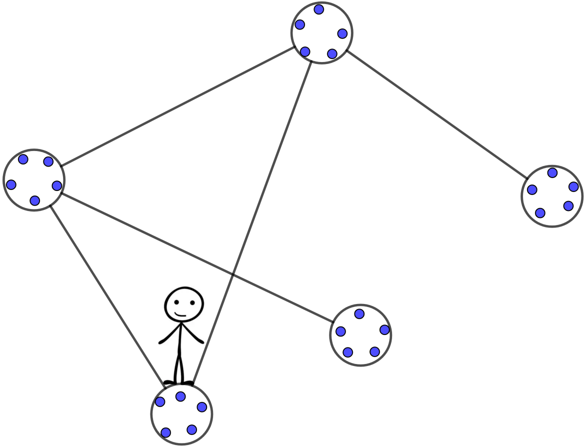


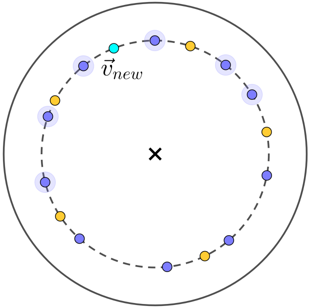


Zoom on the current vertex

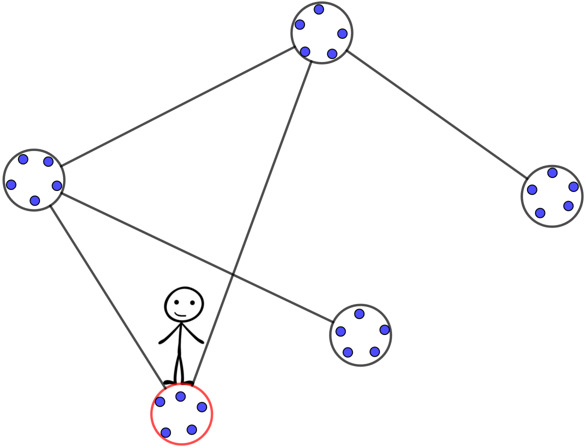


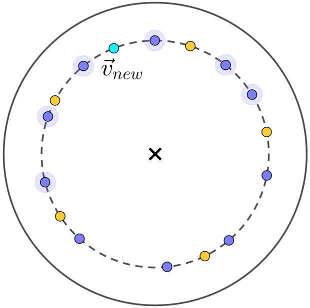
Zoom on the current vertex

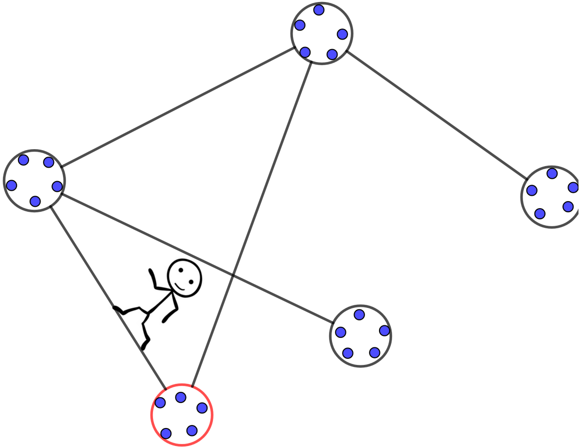




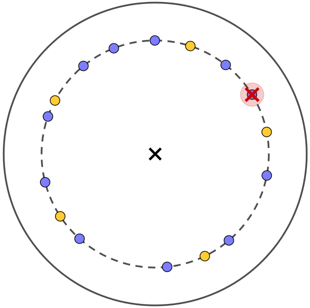
Zoom on the current vertex

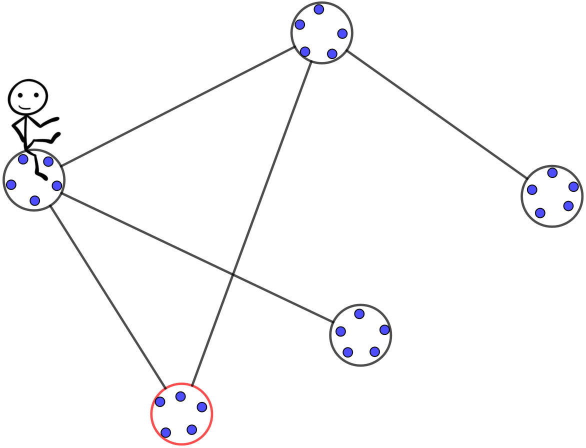




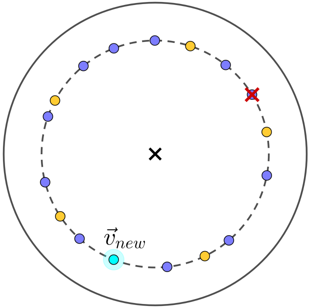


Zoom on the current vertex

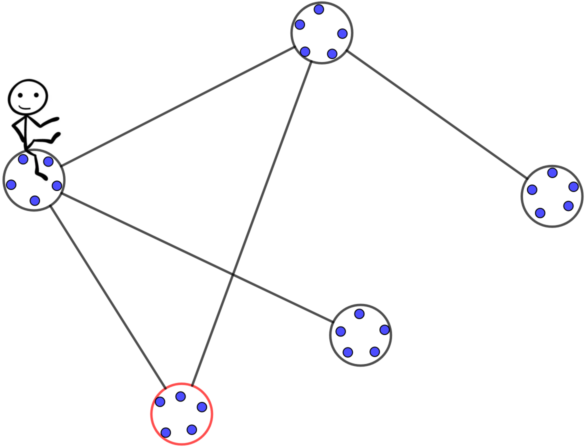


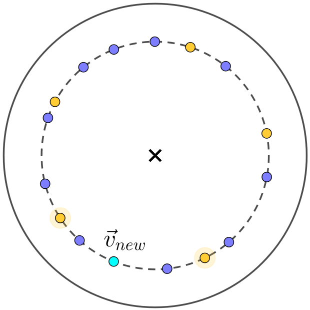


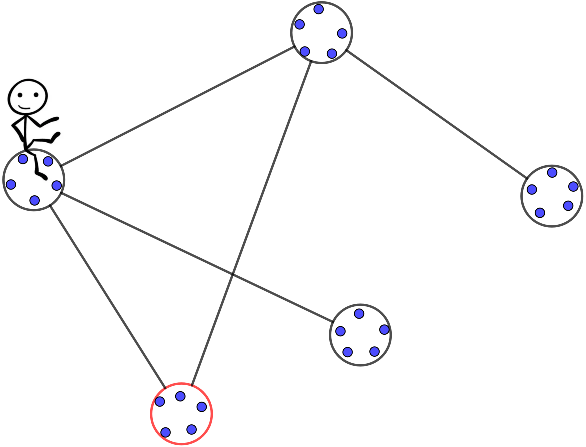
Zoom on the current vertex



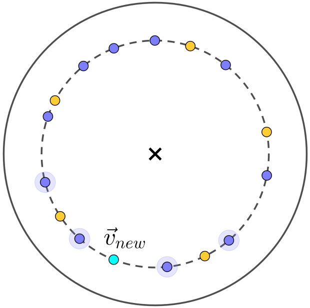
Zoom on the current vertex



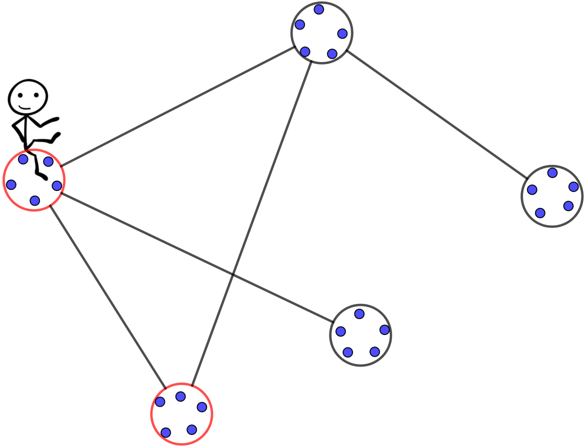


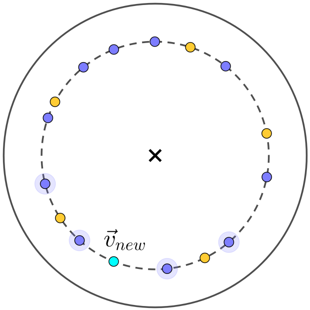


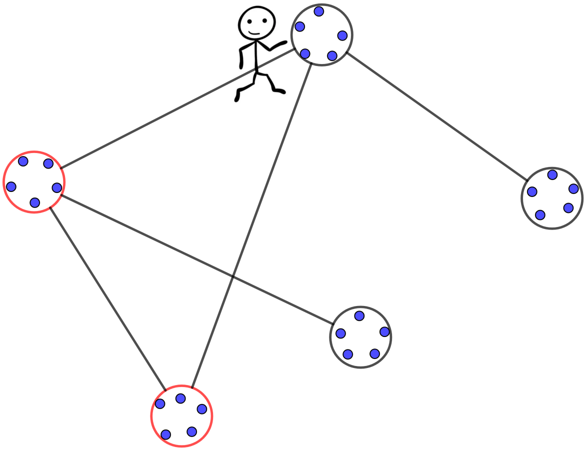
Zoom on the current vertex



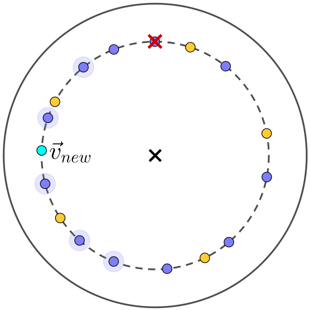
Zoom on the current vertex



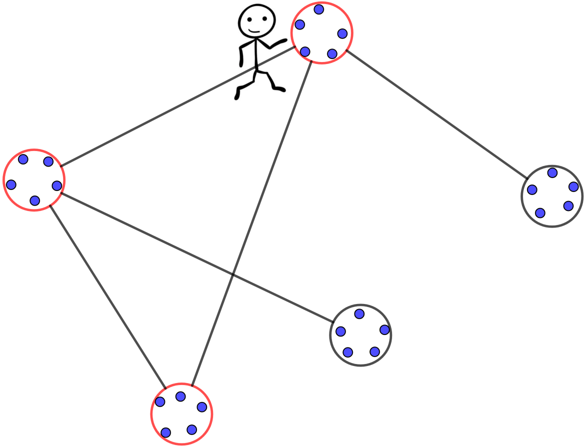


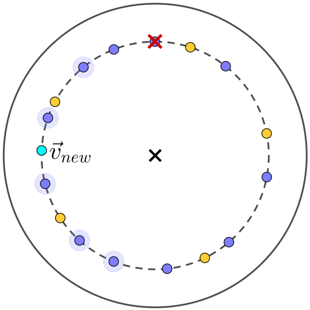


Zoom on the current vertex

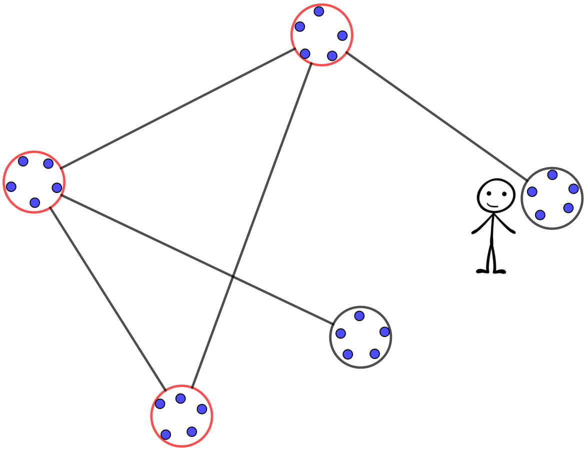


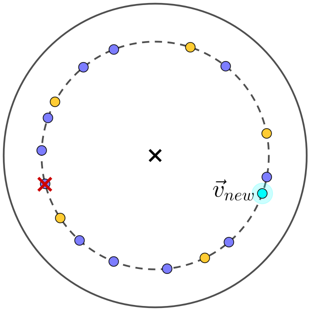
Zoom on the current vertex

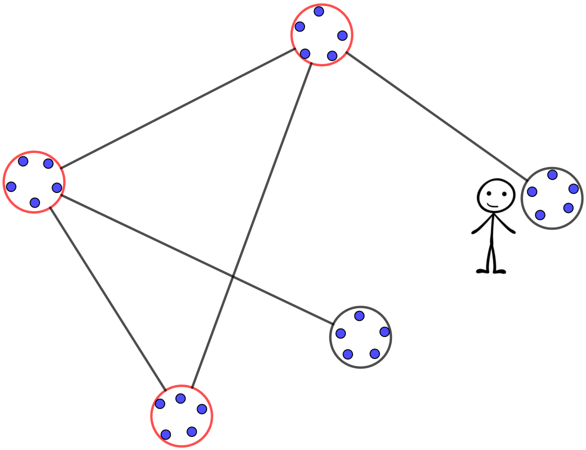




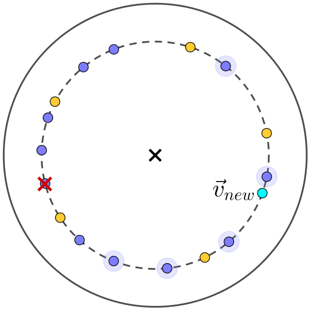
Zoom on the current vertex



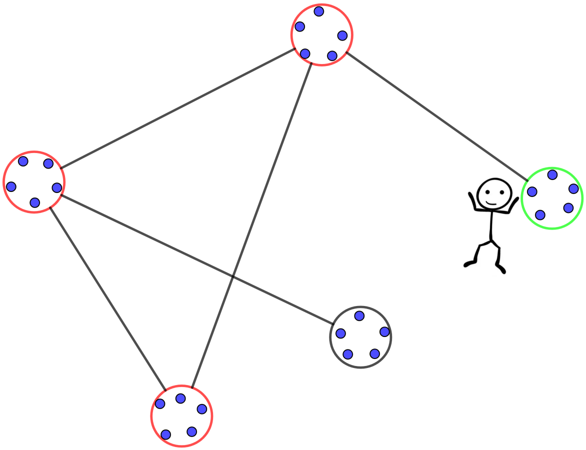


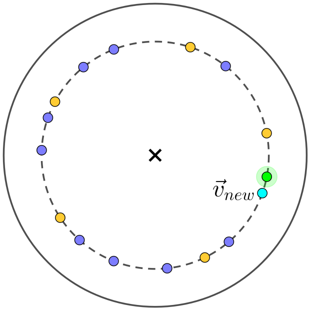


Zoom on the current vertex



Zoom on the current vertex





Classic VS Quantum Random Walk

Classic Random Walk

Randomly choose 1 neighbor vertex.

Quantum Random Walk

Quantum superposition of all the neighbors vertexes.

Step 1

Sample a code C and generate the α-filters.
Insert each list vector in its (unique) nearest α-filter. $\triangleright$ N^{c_α} vectors per α-filter

Step 2

For each α-filter :

1. VERTEX : Choose randomly N^{c_V} vectors from the α-filter. $\triangleright$ N^{c_V} vectors in the vertex
2. Sample a code C' and generate the β-filters.
   Insert each VERTEX's vector in its nearest β-filter.
3. Perform Quantum Random Walks to find all the reducing pairs in the α-filter.

Step 1

Sample a code C and generate the α-filters.
Insert each list vector in its (unique) nearest α-filter. ▷ N^{c_α} vectors per α-filter

Step 2

For each α-filter :

1. VERTEX : Choose randomly N^{c_V} vectors from the α-filter. ▷ N^{c_V} vectors in the vertex
2. Sample a code C' and generate the β-filters.
   Insert each VERTEX's vector in its nearest β-filter.
3. Perform Quantum Random Walks to find all the reducing pairs in the α-filter.

Repetitions

Run the steps 1-2 until we get N reduced vectors.

Complexity

Time complexity of a complete sieve step:

$$N \cdot \left(\mathcal{S} + \frac{1}{\sqrt{\epsilon}} \left(\frac{1}{\sqrt{\delta}} \mathcal{U} + \mathcal{C} \right) \right)$$

Parameters:

- c_α : N^{c_α} vectors per α-filter of C.
- c_V : N^{c_V} vectors per vertex in the graph.

Optimal complexity

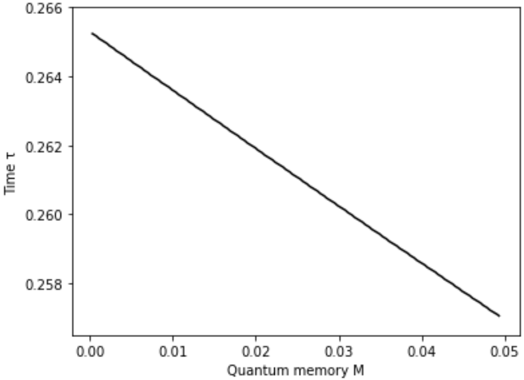
Our algorithm with parameters

$$c_\alpha \approx 0.3696 \quad ; \quad c_V \approx 0.2384$$

heuristically solves SVP on dimension d

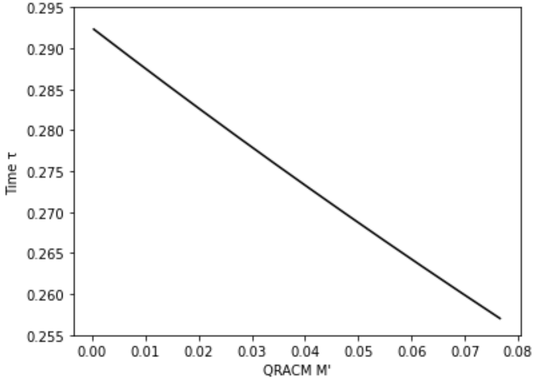
- in time $2^{0.2570d+o(d)}$,
- uses QRAM of maximum size $2^{0.0767d+o(d)}$,
- uses quantum memory of maximum size $2^{0.0495d+o(d)}$
- and uses classical memory of size $2^{0.2075d+o(d)}$.

Trade-off – fixed quantum memory



Quantum memory/time trade-off.

Trade-off – fixed QRAM



QRAM/time trade-off.

Trade-off – Synthesis

Time	**0.2925**	0.2827	0.2733	**0.2653**	0.2621	0.2598	**0.2570**
QRAM	**0**	0.02	0.04	**0.0578**	0.065	0.070	**0.0767**
Qmem	**0**	0	0	**0**	0.0190	0.0324	**0.0495**
Comment	[BDGL16] alg.			[Laa16] alg.			opt.param

Figure: Time, QRAM and quantum memory values for our algorithm.

Conclusion

- Time to break a cryptosystem based on SVP: $2^{0.2653d+o(d)} \to 2^{0.2570d+o(d)}$.
- 128 bits of security $\to$ 124.
- Fix with a slight increase of the parameters.

Thank you for your attention!

References

P.Q. Nguyen and T. Vidick (2008)
Sieve algorithms for the shortest vector problem are practical
J. Math. Crypt. 2, 181 – 207. https://doi.org/10.1515/JMC.2008.009

A. Becker, L. Ducas, N. Gama and T. Laarhoven (2016)
New directions in nearest neighbor searching with applications to lattice sieving
Proc. of the 2016 Annual ACM-SIAM Symposium on Discrete Algorithms.
https://doi.org/10.1137/1.9781611974331.ch2

T. Laarhoven (2016)
(PhD thesis) Search problems in cryptography, From fingerprinting to lattice sieving
Eindhoven University of Technology. https://research.tue.nl/files/14673128/20160216_Laarhoven.pdf

F. Magniez, A. Nayak, J. Roland and M. Santha (2011)
Search via quantum walk
SIAM J. Comput. Vol. 40(1) 142 – 164. https://doi.org/10.1137/090745854

A. Chailloux and J. Loyer (2021)
Lattice sieving via quantum random walks
ASIACRYPT21. https://eprint.iacr.org/2021/570.pdf