Almost Improving Quantum Lattice Sieving

Journées Informatique Quantique 2026

Mathias BOUCHER

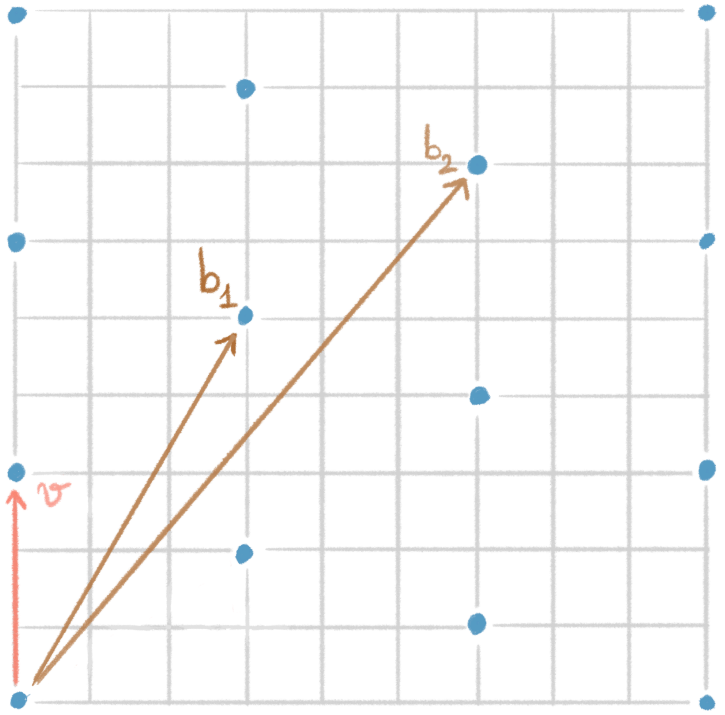
UMR IRISA

Université de Rennes

Introduction

Lattices, Shortest Vector Problem

A Lattice is $\mathcal{L} = \left\{ \sum_{i=1}^{d} s_i \boldsymbol{b_i} : s_i \in \mathbb{Z} \right\}$ where $\boldsymbol{b_1}, ..., \boldsymbol{b_d}$ is a basis of $\mathbb{R}^d$.



Shortest Vector Problem (SVP)

Given a basis of $\mathcal{L}$, find a non zero shortest vector.

Motivation to solve SVP

Post-Quantum Cryptography

- NP-hard problem, believed to be quantum-resistant.
- Other problems like LWE, SIS, NTRU are derieved from SVP.
- Cryptosystems based on them: Kyber, Dilithium, Falcon.

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

Sieving approach

Sieving Approach

Generate a list L of vectors and combine them to reduce there length.

Let $\mathcal{A}$ be an algorithm that, starting from a list L_0 of unit vectors, constructs a list L_1 of vectors of norm $\gamma < 1$.

1. $L_0 = L$ with $\|x\| = 1, \forall x \in L_0$
2. $L_1 = \mathcal{A}(L_0)$ with $\|x\| = \gamma, \forall x \in L_1$

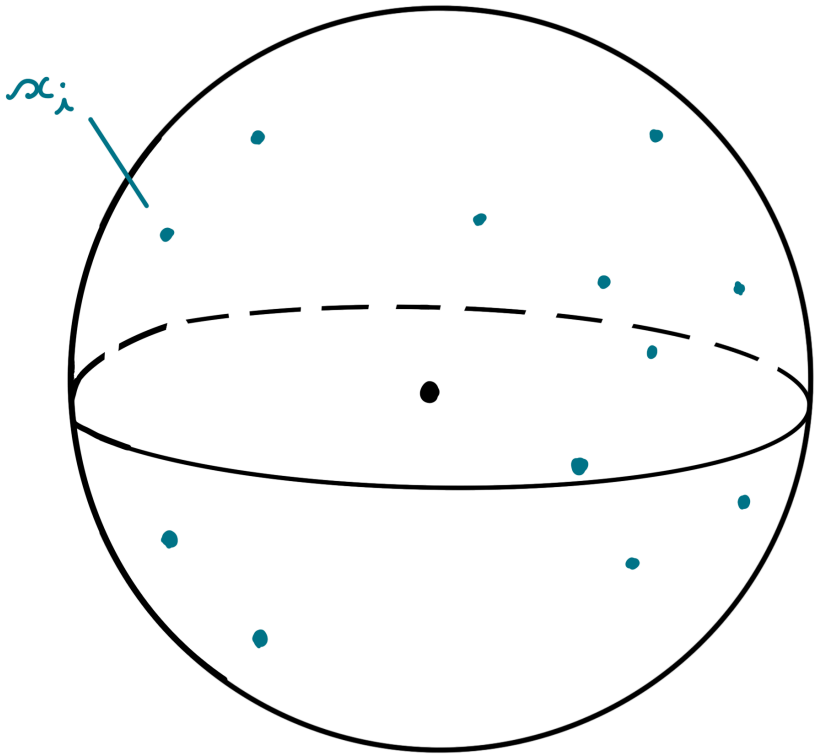
$$\vdots$$

3. $L_k = \mathcal{A}(L_{k-1})$ with $\|x\| = \gamma^k, \forall x \in L_k$

Given a list L of vectors, preprocess L such that one can efficiently find pairs of vectors close to each other.

Locality Sensitive Filtering

Sieving Algorithm, Filters

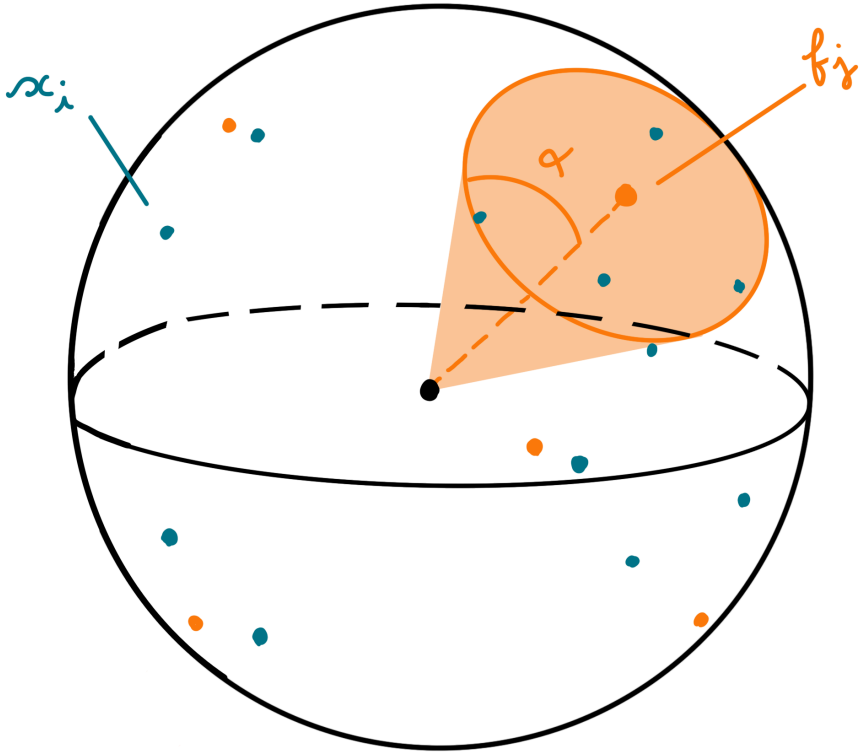


Hypothesis

Let $x_i \in L$,

$$x_i \xleftarrow{\$} \mathbb{S}^d$$

Sieving Algorithm, Filters



Let $x, x' \in L$,

$$\|x - x'\| \leq 1 \Leftrightarrow \theta(x, x') \leq \frac{\pi}{3}$$

Filters, buckets

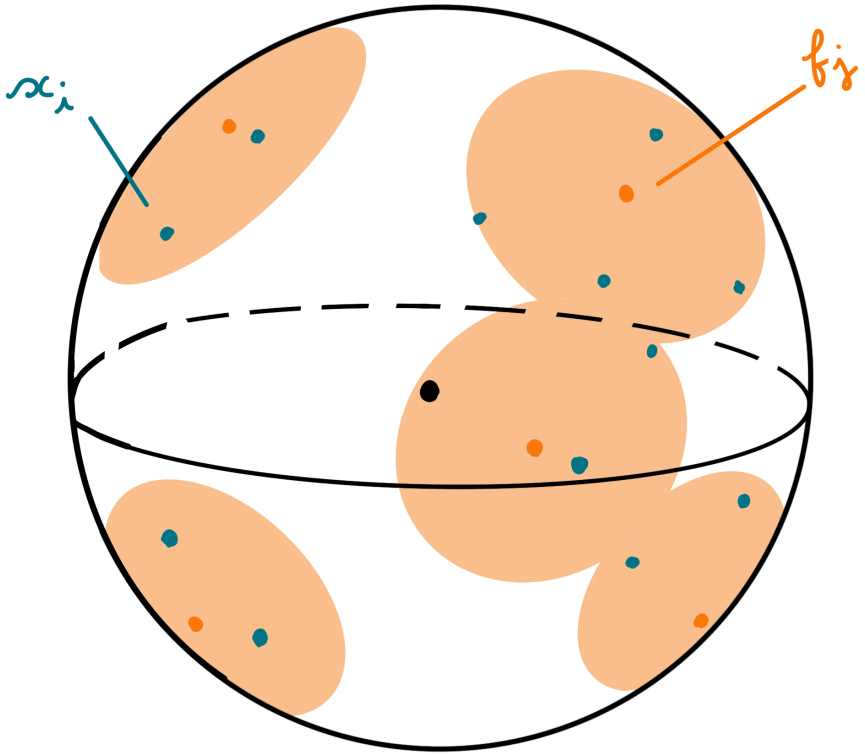
Let $f_j \in \mathbb{S}^d$ be a filter. We define the α-hypercone,

$$\mathcal{H}_{\alpha, f_j} = \{x \in \mathbb{S}^d : \theta(f, x) \leq \alpha\}.$$

The bucket $B_\alpha(f_j)$ is defined by

$$B_\alpha(f_j) = \mathcal{H}_{\alpha, f_j} \cap L.$$

Sieving Algorithm, Filters



1. Sample filters $\{f_i\}_i$ and initialize the buckets.
2. Fill the buckets $B_\alpha(f_j)$ with the lattice vectors.
3. Find the solution among each buckets.
4. Repeat steps 1, 2, and 3 until you find enough solutions.

Time Complexity

$$N_{\text{REP}}(N_{\text{INIT}} + N_{\text{FindSol}})$$

Quantum Random Walk

Basis of Quantum Computing

Grover's Search

Let L a list of size N and a function $f : L \to \{0, 1\}$, there exists a quantum algorithm which finds $x \in L$ such that $f(x) = 1$ using $O\left(\sqrt{N}\right)$ evaluations of the function.

$$\frac{1}{\sqrt{N}} \sum_{x \in L} |x\rangle$$

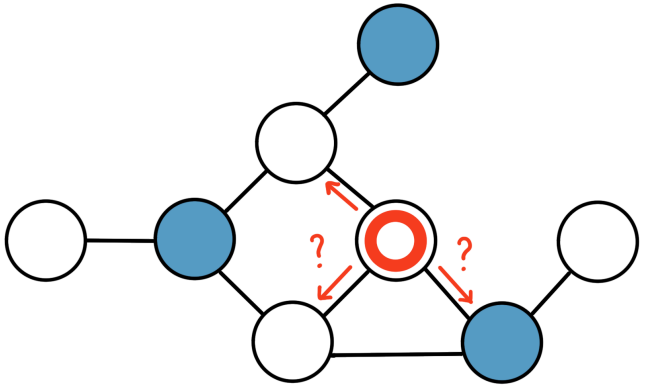
Quantum Random Walk

- Setup cost: $\mathcal{S}$
- Checking cost: $\mathcal{C}$
- Update cost: $\mathcal{U}$

The classical random walk finds a marked element in time

$$T = \mathcal{S} + \frac{1}{\varepsilon}\left(\mathcal{C} + \frac{1}{\delta}\mathcal{U}\right)$$

where ε is the ratio of marked vertices and δ the spectral gap.



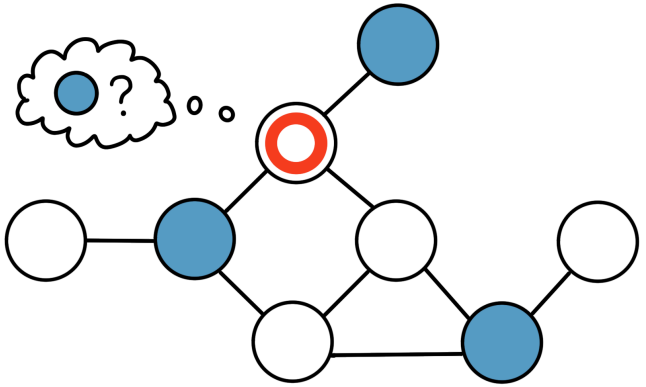
Quantum Random Walk

- Setup cost: $\mathcal{S}$
- Checking cost: $\mathcal{C}$
- Update cost: $\mathcal{U}$

The classical random walk finds a marked element in time

$$T = \mathcal{S} + \frac{1}{\varepsilon}\left(\mathcal{C} + \frac{1}{\delta}\mathcal{U}\right)$$

where ε is the ratio of marked vertices and δ the spectral gap.



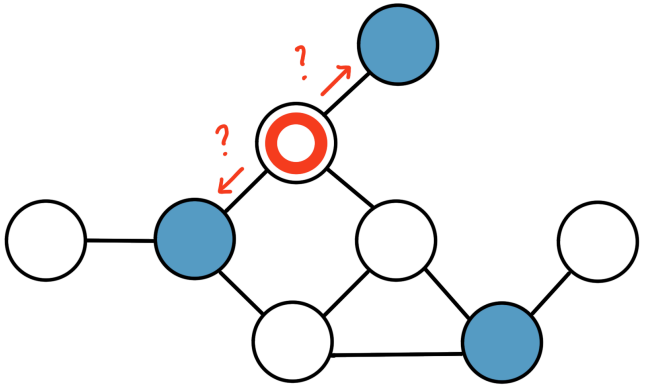
Quantum Random Walk

- Setup cost: $\mathcal{S}$
- Checking cost: $\mathcal{C}$
- Update cost: $\mathcal{U}$

The classical random walk finds a marked element in time p

$$T = \mathcal{S} + \frac{1}{\varepsilon}\left(\mathcal{C} + \frac{1}{\delta}\mathcal{U}\right)$$

where ε is the ratio of marked vertices and δ the spectral gap.



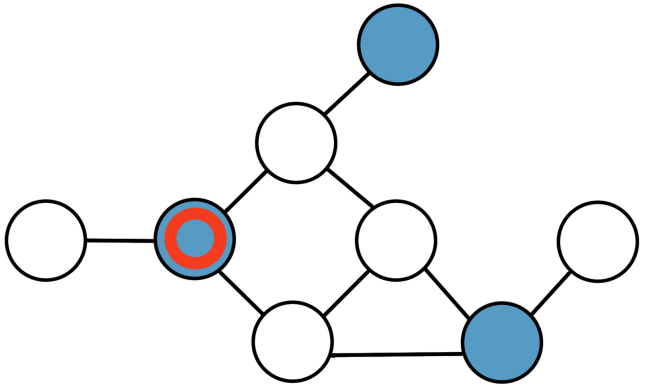
Quantum Random Walk

- Setup cost: $\mathcal{S}$
- Checking cost: $\mathcal{C}$
- Update cost: $\mathcal{U}$

The classical random walk finds a marked element in time

$$T = \mathcal{S} + \frac{1}{\varepsilon}\left(\mathcal{C} + \frac{1}{\delta}\mathcal{U}\right)$$

where ε is the ratio of marked vertices and δ the spectral gap.



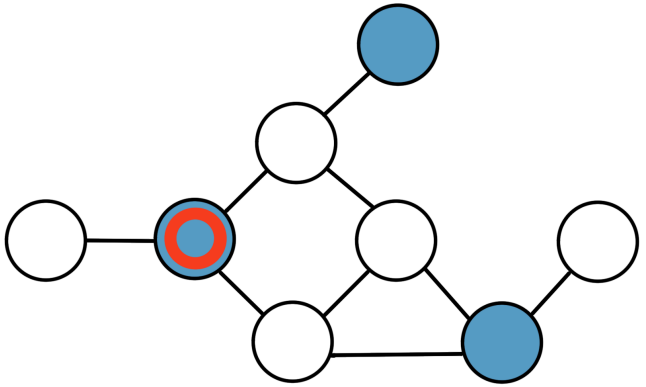
Quantum Random Walk

- Setup cost: $\mathcal{S}$
- Checking cost: $\mathcal{C}$
- Update cost: $\mathcal{U}$

The quantum random walk find a marked element in time

$$T = \mathcal{S} + \frac{1}{\sqrt{\varepsilon}}\left(\mathcal{C} + \frac{1}{\sqrt{\delta}}\mathcal{U}\right)$$

where ε is the ratio of marked vertices and δ the spectral gap.



Johnson Graph

$B_\alpha = \{y_1, ..., y_n\}$, fix $r < n$,

The graph $J(n, r)$:

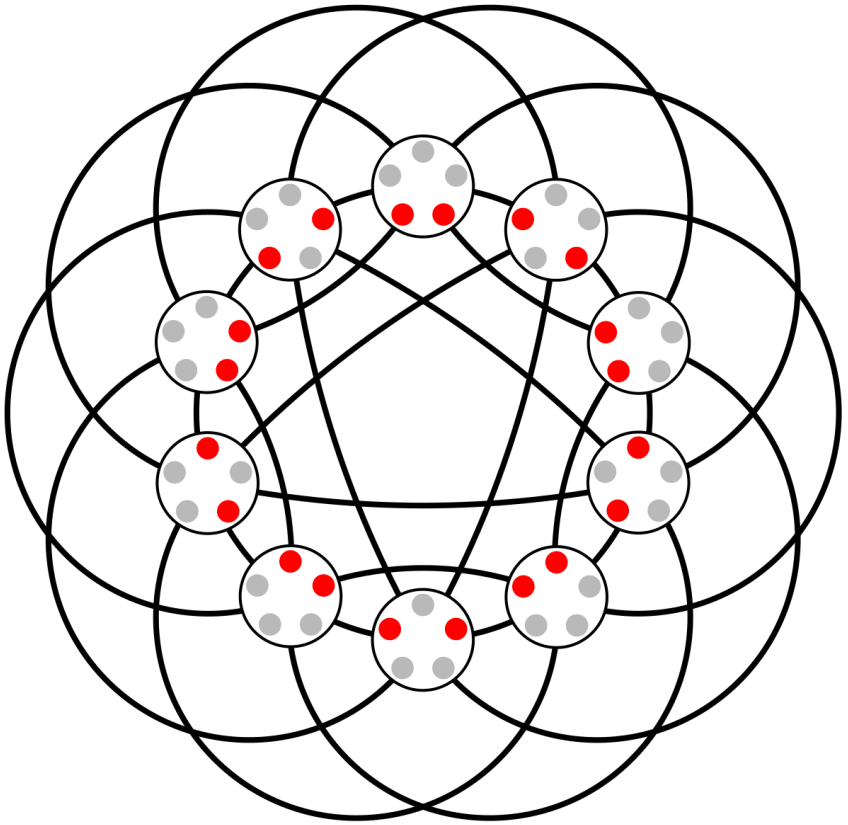
- Nodes are subsets of r vectors among B_α (with $r \ll n$), and data stuff

$$v = (L^v, D(v), b^v).$$

- Two nodes v, w are connected if there exists y_{old} and y_{new} such that

$$L^w = (L^v \setminus \{y_{\text{old}}\}) \cup \{y_{\text{new}}\}.$$

- v is marked if there exists a neighboring pair in L^v



Example : $J(5, 2)$

How to update the last bit ?

Let $v = (L^v, b^v)$ and $w = ((L^v \setminus \{y_{\text{old}}\}) \cup \{y_{\text{new}}\}, b^w)$ be two connected nodes.

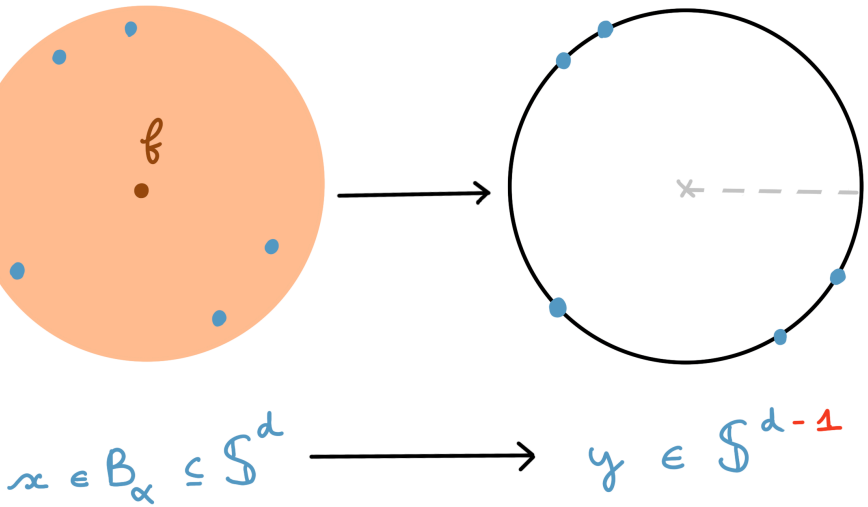
Naive Idea: Check among all y_i's in L^v if removing y_{old} would change b^v, idem with y_{new} and update b^w accordingly. $\rightsquigarrow$ too slow !

How to update the last bit ?

Let $v = (L^v, b^v)$ and $w = ((L^v \setminus \{y_{\text{old}}\}) \cup \{y_{\text{new}}\}, b^w)$ be two connected nodes.

Naive Idea: Check among <u>all</u> y_i's in L^v if removing y_{old} would change b^v, idem with y_{new} and update b^w accordingly. $\rightsquigarrow$ too slow !

How to improve it? Add more data !

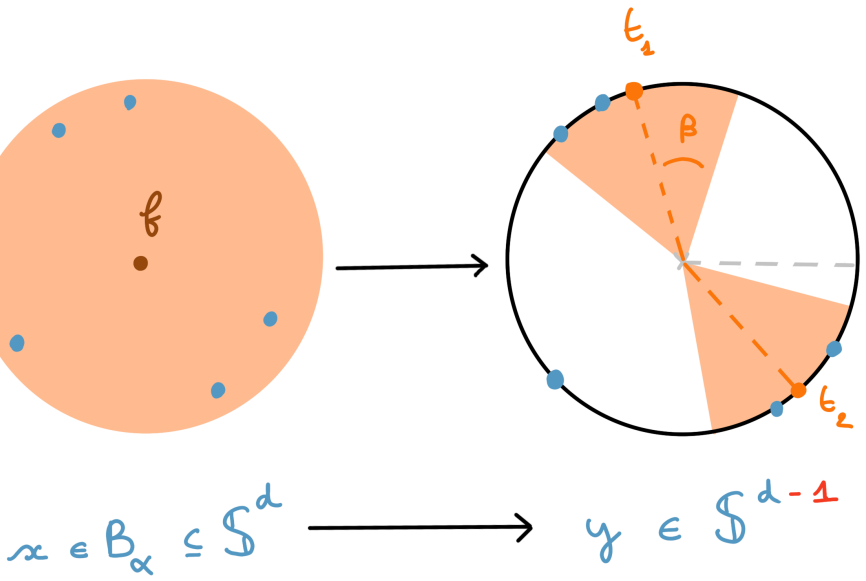


How to update the last bit ?

Let $v = (L^v, b^v)$ and $w = ((L^v \setminus \{y_{\text{old}}\}) \cup \{y_{\text{new}}\}, b^w)$ be two connected nodes.

Naive Idea: Check among <u>all</u> y_i's in L^v if removing y_{old} would change b^v, idem with y_{new} and update b^w accordingly. $\rightsquigarrow$ too slow !

How to improve it? Add more data !

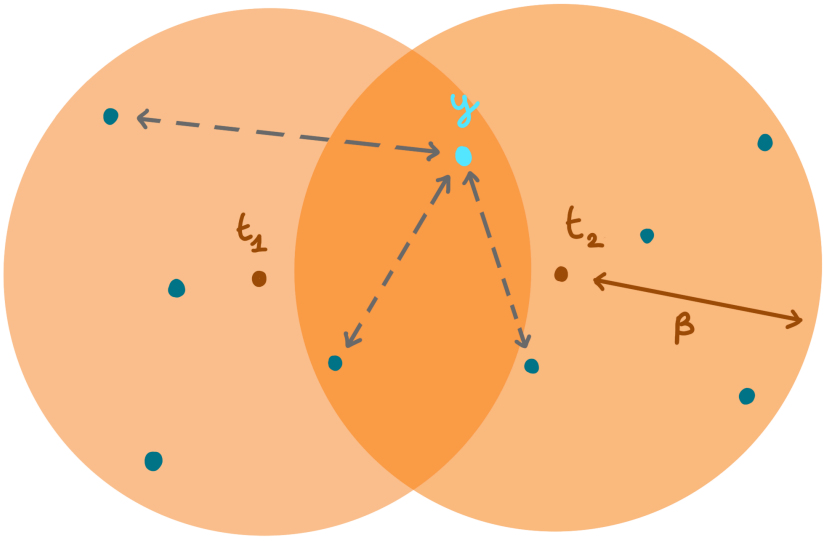


Sample $t_1, \ldots, t_r \in \mathcal{F}$ filters

$$B^v_\beta(t_j) = \mathcal{H}_{\beta,t_j} \cap L^v$$

$$D(v) = \left\{B^v_\beta(t_1), \ldots, B^v_\beta(t_r)\right\}$$

Update phase



UPDATE_PHASE(y, v):

1 Compute $K_\beta = \mathcal{H}_{\beta,y} \cap \mathcal{F}$
2 **For** $t_i \in K_\beta$:
3 update(y) **in** $B^v_\beta(t_i)$
4 **For** $y_i \neq y$ **in** $\cup_{t_i \in K_\beta} B^v_\beta(t_i)$:
5 **If** (y_i, y) is a neighboring pair:
6 update the last register
7 **Return** the new vertex

[Chailloux Loyer21]

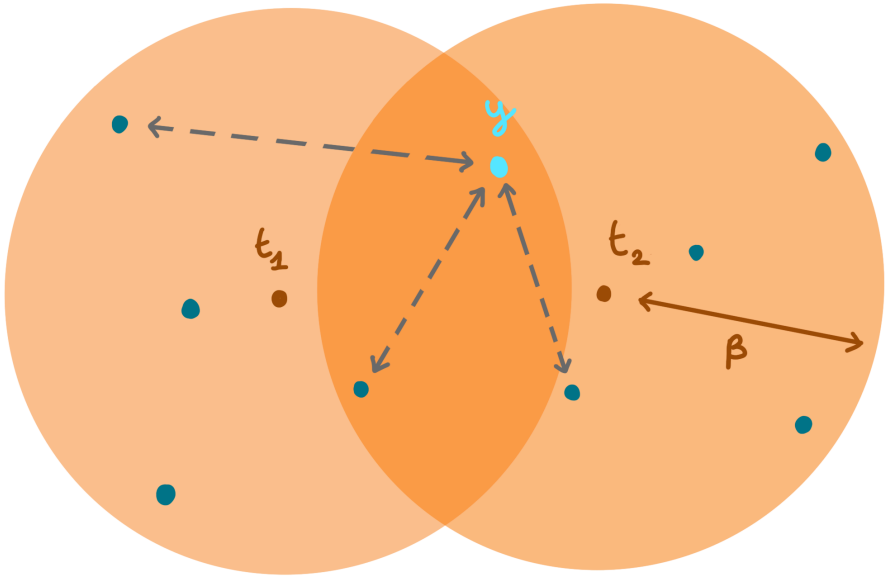
Do it twice in order to remove y_{old} and add y_{new}.

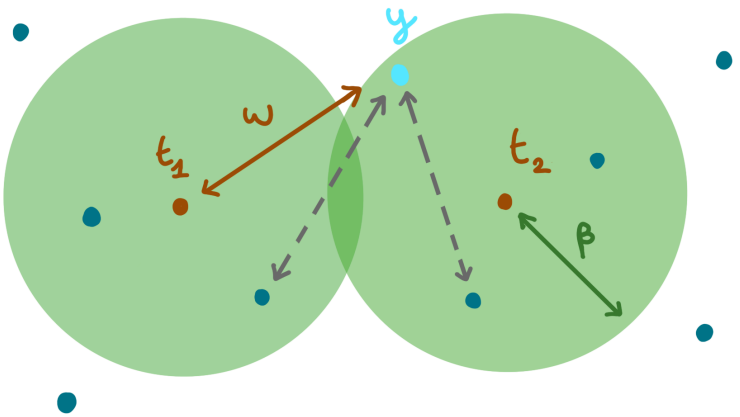
Current Complexity :

$$T_{\text{update}} = 2 \cdot \left(|K_\beta| + |K_\beta| + \sqrt{|K_\beta| \cdot |B_\beta|} \right)$$

Improvement

How to improve the update phase ?





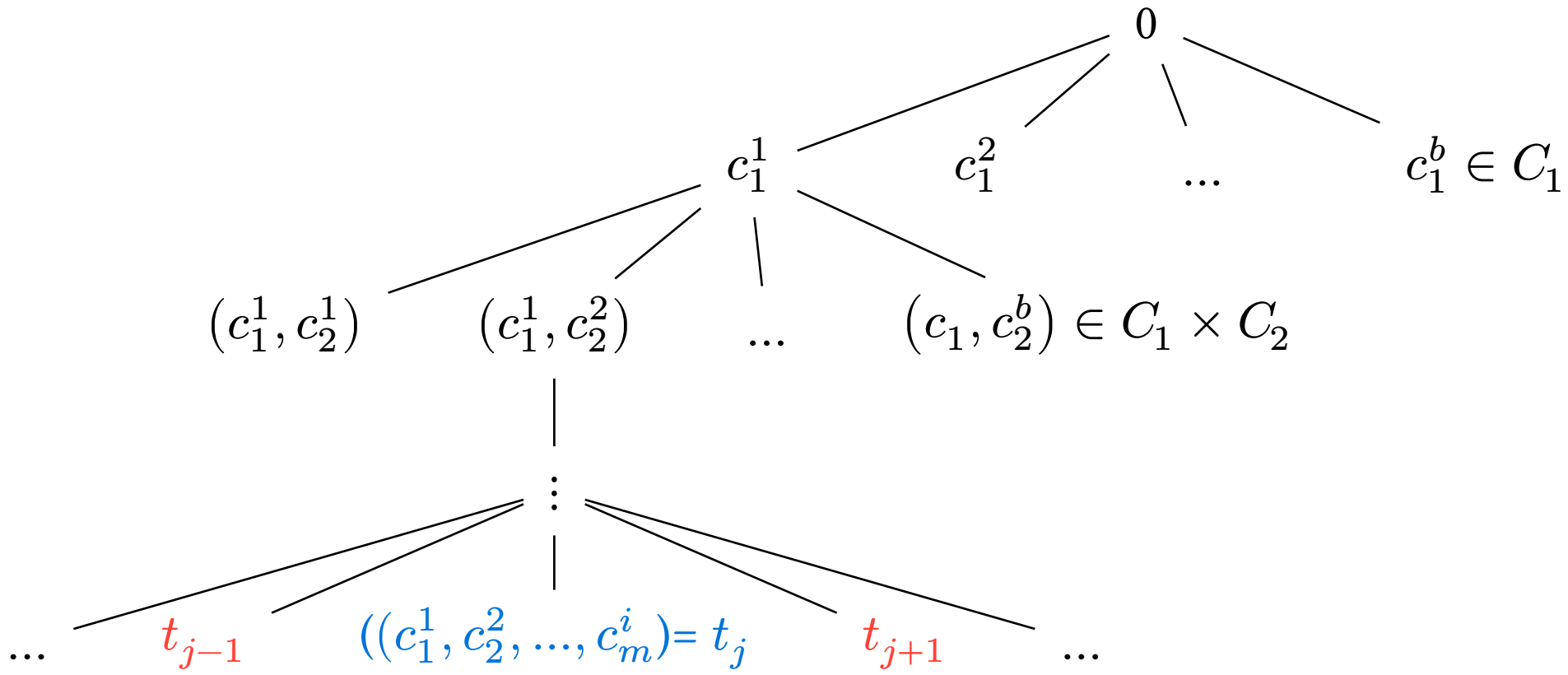
- Bigger β-buckets
- y compared with y_i in β-close filters

- Smaller β-buckets
- y compared with y_i in ω-close filters

$\rightsquigarrow$ less time to fill the buckets

Tree Structure on RPC

Random Product codes : $\mathcal{F} = C_1 \times ... \times C_m$, where $|C_i| = b$.
For $t \in \mathcal{C}_2$, we can write $t = (c_1, ..., c_m)$ with $c_i \in C_i$.



■ ω-close filters - ■ not ω-close filters

Good Leaf Sampler

Filter Sampling [Heiser21]

Let $y \in B_\alpha(f)$, for any $\omega \in \left[\frac{\pi}{3}, \frac{\pi}{2}\right]$, there exists a sampling routine wich samples a filter t_j which is ω-close to y in time $N^{o(1)}$. $(N := |L|)$

Good Leaf Sampler

Filter Sampling [Heiser21]

Let $y \in B_\alpha(f)$, for any $\omega \in \left[\frac{\pi}{3}, \frac{\pi}{2}\right]$, there exists a sampling routine wich samples a filter t_j which is ω-close to y in time $N^{o(1)}$. $(N := |L|)$

$$\sum_{t_i\ \omega\text{-close}} |t_i\rangle \longrightarrow \sum_{t_i\ \omega\text{-close}} \sum_{y_j \in B_\beta(t_i)} |t_i, y_j^i\rangle$$

1. Construct the list K_ω of t_i's ω-close to y,
2. Do a Grover's search on the list $\cup_{t\in K_\omega} B_\beta(t)$.

Time: $|K_\omega| + \sqrt{|K_\omega| \cdot |B_\beta|}$

1. Preprocess the ω-close filter sampler,
2. Do a Grover's search on the above quantum state.

Time: $N^{o(1)} + \sqrt{|K_\omega| \cdot |B_\beta|}$

Complexity

$$T_{\text{update}} = |K_\beta| + |K_\beta| + \sqrt{|K_\beta| \cdot |B_\beta|}$$

$$T_{\text{improved\_update}} = |K_\beta| + N^{o(1)} + \sqrt{|K_\omega| \cdot |B_\beta|}$$

Complexity

$$T_{\text{update}} = |K_\beta| + |K_\beta| + \sqrt{|K_\beta| \cdot |B_\beta|}$$

$$T_{\text{improved\_update}} = |K_\beta| + N^{o(1)} + \sqrt{|K_\omega| \cdot |B_\beta|}$$

$$T_{\text{LSF}} = N^{c-\zeta}\left(N + N^{1-c}\max\{N^{\zeta}, 1\}\left(N^{c_1+\rho_0} + \frac{1}{\max\left(N^{c_1}\sqrt{\mathcal{V}_d(\theta^*_\alpha)}\right)}\left(N^{\max\{\rho_0, \frac{\rho_0+c_3}{2}\}+\frac{c_1}{2}}\right)\right)\right)$$

Complexity

- Total time: c_T where $T_{\mathrm{LSF}} = 2^{c_T d + o(d)}$,
- Update cost: $c_{\mathcal{U}}$ where $\mathcal{U} = 2^{c_{\mathcal{U}} d + o(d)}$

Algorithms	Total Time	Update cost
[CL21]–1st algo	0.2605	0.0530
↳ Our version	0.2594	0.0109

Complexity

- Total time: c_T where $T_{\mathrm{LSF}} = 2^{c_T d + o(d)}$,
- Update cost: $c_{\mathcal{U}}$ where $\mathcal{U} = 2^{c_{\mathcal{U}} d + o(d)}$

Algorithms	Total Time	Update cost
[CL21]–1st algo	0.2605	0.0530
↳ Our version	0.2594	0.0109
[Heiser21]	0.2571	no quantum walk
[CL21]–2nd algo	0.2570	0
↳ Our version	0.2570	0
[BCSS23]	0.2563	0
↳ Our version	0.2563	0

Conclusion

In this work:

- We integrate the algorithm of **[Heiser21]** into the update phase of the quantum walk of **[ChaillouxLoyer21]**.
- No further asymptotic gains through this approach.

Further research:

- Look if it works for 3-sieve algorithms (or k-sieve algorithms)

Conclusion

In this work:

- We integrate the algorithm of **[Heiser21]** into the update phase of the quantum walk of **[ChaillouxLoyer21]**.
- No further asymptotic gains through this approach.

Further research:

- Look if it works for 3-sieve algorithms (or k-sieve algorithms)

Thank you for listening !