# Finding many Collisions via Quantum Walks

## Application to Lattice Sieving

Xavier Bonnetain André Chailloux André Schrottenloher
**Yixin Shen**

June 22, 2023

# Classical Collisions Finding

# Classical collisions algorithms

## Problem

Given random $f : \{0,1\}^n \to \{0,1\}^m, n \le m \le 2n$, find $2^k$ collision pairs,

# Classical collisions algorithms

**Problem**

Given random $f : \{0,1\}^n \to \{0,1\}^m, n \leq m \leq 2n$, find $2^k$ collision pairs, where $k \leq 2n - m$.

Birthday paradox: expect $2^{2p-m}$ collisions among $2^p$ random $x$'s

# Classical collisions algorithms

## Problem

Given random $f : \{0,1\}^n \to \{0,1\}^m, n \leq m \leq 2n$, find $2^k$ collision pairs, where $k \leq 2n - m$.

Birthday paradox: expect $2^{2p-m}$ collisions among $2^p$ random $x$'s

## Algorithm

- Query $f$ on $2^{k/2+m/2}$ different $x$.
- Sort the list of $(x, f(x))$ according to $f(x)$.
- The list contains $2^k$ collisions on average (birthday paradox).

# Classical collisions algorithms

## Problem

Given random $f : \{0,1\}^n \to \{0,1\}^m, n \leq m \leq 2n$, find $2^k$ collision pairs, where $k \leq 2n - m$.

Birthday paradox: expect $2^{2p-m}$ collisions among $2^p$ random $x$'s

## Algorithm

- Query $f$ on $2^{k/2+m/2}$ different $x$.
- Sort the list of $(x, f(x))$ according to $f(x)$.
- The list contains $2^k$ collisions on average (birthday paradox).

## Lower bounds

Matching query lower bound in all cases

# Quantum Collisions Finding

# Quantum Collisions Finding

## BHT algorithm for finding 1 collision when $m = n$

- Take a list $L = (f(y_0), \ldots, f(y_{2^r}))$
- (Grover) Search for an $x$ with $f(x) = f(y_i)$ and $x \neq y_i$
- Cost $2^r$ memory, $2^r + \sqrt{\frac{2^n}{2^r}}$ time $\rightsquigarrow$ optimal for $r = n/3$

## Finding $2^k$ collisions when $m = n$

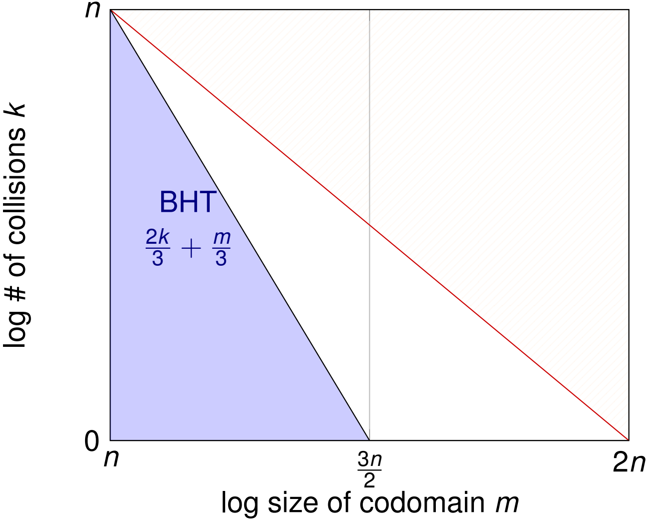
- Use one larger list of size $2^{n/3+2k/3}$
- Do $2^k$ quantum searches $\left(\text{cost } 2^k \times \sqrt{\frac{2^n}{2^{n/3+2k/3}}} = 2^{n/3+2k/3}\right)$

## Lower bound [LZ19]

General query lower bound $\Omega\left(2^{m/3+2k/3}\right)$

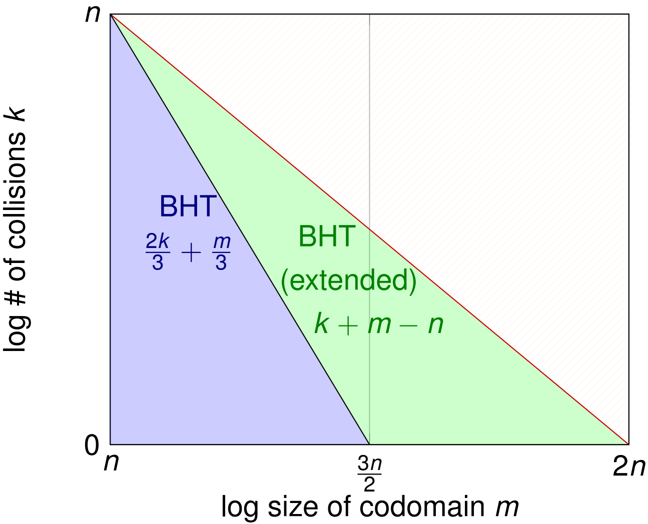
# Summary

Find $2^k$ collision pairs of $f : \{0,1\}^n \to \{0,1\}^m$, with $k \leq 2n - m$



# Summary

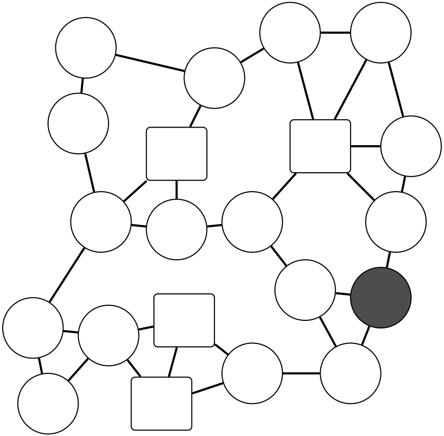
Find $2^k$ collision pairs of $f : \{0,1\}^n \to \{0,1\}^m$, with $k \leq 2n - m$



# Collision Finding
## via Random Walks

# Classical random walk

We move to random neighbors until we find a marked vertex.

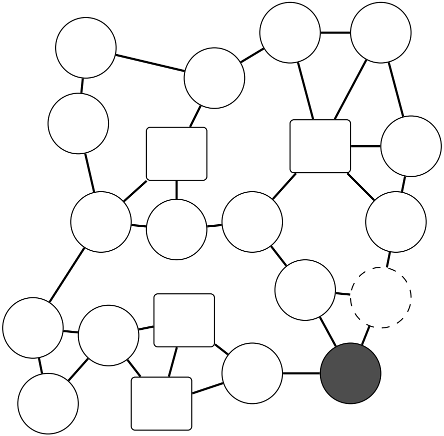


**Applications**

- graph: search space
- marked nodes: solutions

# Classical random walk

We move to random neighbors until we find a marked vertex.

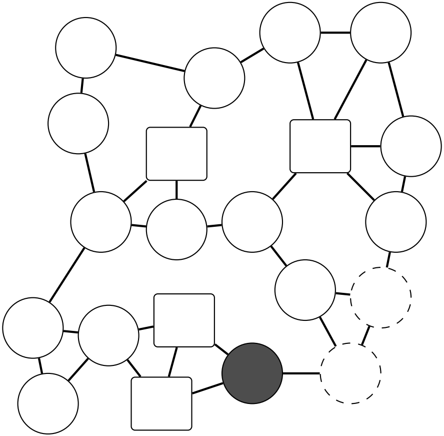


**Applications**

- graph: search space
- marked nodes: solutions

# Classical random walk

We move to random neighbors until we find a marked vertex.

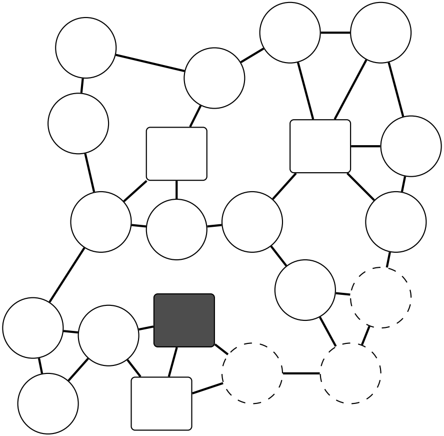


**Applications**

- graph: search space
- marked nodes: solutions

# Classical random walk

We move to random neighbors until we find a marked vertex.



Applications

- graph: search space
- marked nodes: solutions

# Cost of a classical random walk

We need procedures:

- To **setup** a starting arbitrary vertex (S)
- To **move** from one vertex to one of its neighbors (U)
- To **check** if a vertex is marked (C)

We will find a marked vertex in time:

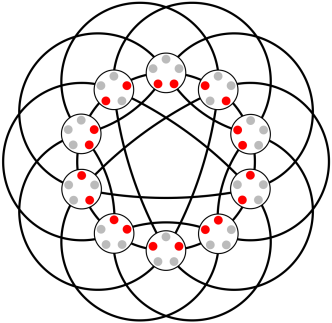
$$\mathsf{S} + \underbrace{\frac{1}{\epsilon}}_{\substack{\epsilon \text{ proportion of} \\ \text{marked vertices}}} \left( \underbrace{\frac{1}{\delta}}_{\substack{\delta \text{ spectral gap} \\ \text{of the graph}}} \mathsf{U} + \mathsf{C} \right)$$

where $\frac{1}{\delta}$ is the number of updates before we reach a new uniformly random vertex.

# Example: Walk-based collision finding

## Definition (Johnson graph)

- Nodes are sets of $2^r$ elements among $2^n$
- $N_1$ and $N_2$ are adjacents if $|N_1 \cap N_2| = 2^r - 1$
- $\frac{1}{\delta} = \frac{2^r(2^n - 2^r)}{2^n} \simeq 2^r$ (We need to replace all elements.)



# Example: Walk-based collision finding

## Definition (Johnson graph)

- Nodes are sets of $2^r$ elements among $2^n$
- $N_1$ and $N_2$ are adjacents if $|N_1 \cap N_2| = 2^r - 1$
- $\frac{1}{\delta} = \frac{2^r(2^n - 2^r)}{2^n} \simeq 2^r$ (We need to replace all elements.)

## Collision finding with Johnson graph

- Create a random list of elements of size $2^r$
- Repeat until a collision is found:
  - Walk $2^r$ times
  - Check whether the node contains a collision

# Example: Walk-based collision finding

## Definition (Johnson graph)

- Nodes are sets of $2^r$ elements among $2^n$
- $N_1$ and $N_2$ are adjacents if $|N_1 \cap N_2| = 2^r - 1$
- $\frac{1}{\delta} = \frac{2^r(2^n - 2^r)}{2^n} \simeq 2^r$ (We need to replace all elements.)

## Collision finding with Johnson graph

- Create a random list of elements of size $2^r$
- Repeat until a collision is found:
  - Walk $2^r$ times
  - Check whether the node contains a collision

## Complexity

$$2^r + \frac{1}{2^{2r-m}}(2^r \times 1 + 2^r) \approx \max(2^r, 2^{m-r})$$

# Example: Walk-based collision finding

## Definition (Johnson graph)

- Nodes are sets of $2^r$ elements among $2^n$
- $N_1$ and $N_2$ are adjacents if $|N_1 \cap N_2| = 2^r - 1$
- $\frac{1}{\delta} = \frac{2^r(2^n - 2^r)}{2^n} \simeq 2^r$ (We need to replace all elements.)

## Collision finding with Johnson graph

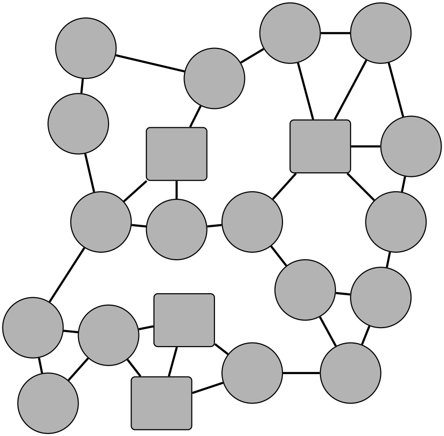
- Create a random list of elements of size $2^r$
- Repeat until a collision is found:
  - Walk $2^r$ times
  - Check whether the node contains a collision

## Complexity

$$2^r + \frac{1}{2^{2r-m}}(2^r \times 1 + 2^r) \approx \max(2^r, 2^{m-r}) \quad \rightsquigarrow \quad \text{optimal for } r = m/2$$

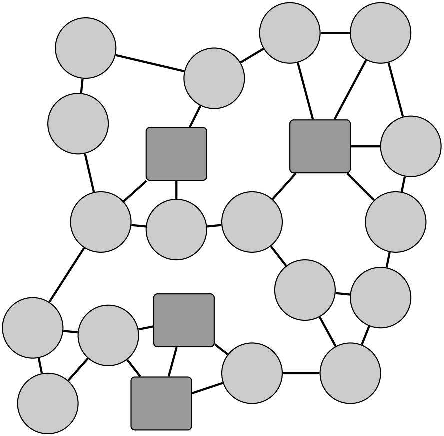
# Quantum walk

The quantum walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.



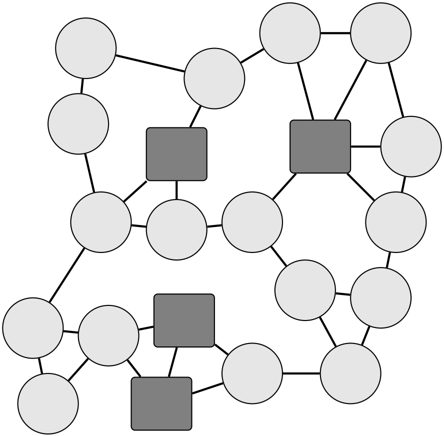
# Quantum walk

The quantum walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.

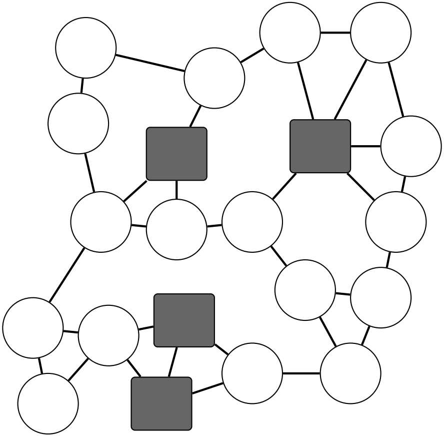


# Quantum walk

The quantum walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.



The quantum walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.



# Time of a quantum walk (MNRS framework)

- The **setup** now requires to create a superposition over **all** vertices
- As in quantum search, we perform $\sqrt{\frac{1}{\epsilon}}$ steps instead of $\frac{1}{\epsilon}$
- But the mixing is also accelerated!

$$\mathsf{S} + \underbrace{\sqrt{\frac{1}{\epsilon}}}_{\text{Walk steps}} \left( \underbrace{\sqrt{\frac{1}{\delta}}\mathsf{U}}_{\text{Mixing time}} + \mathsf{C} \right)$$

- The **Update** handles all vertices and all edges in superposition

# Ambainis's algorithm for Collision Finding

## Problem

$f : \{0,1\}^n \to \{0,1\}^m$, $n \leq m \leq 2n$, find a collision

## MNRS walk in a Johnson graph

- Setup : Create the uniform superposition of all lists of $2^r$ elements
- Fraction of marked nodes : $\epsilon = 2^{2r-m}$
- Mixing time: $\sqrt{\frac{1}{\delta}} = 2^{r/2}$
- **Assume Update and Check polynomial time**
- Cost $2^r + 2^{m/2-r} \times 2^{r/2} \simeq \max(2^r, 2^{m/2-r/2})$

# Ambainis's algorithm for Collision Finding

## Problem

$f : \{0,1\}^n \to \{0,1\}^m$, $n \leq m \leq 2n$, find a collision

## MNRS walk in a Johnson graph

- Setup : Create the uniform superposition of all lists of $2^r$ elements
- Fraction of marked nodes : $\epsilon = 2^{2r-m}$
- Mixing time: $\sqrt{\frac{1}{\delta}} = 2^{r/2}$
- **Assume Update and Check polynomial time**
- Cost $2^r + 2^{m/2-r} \times 2^{r/2} \simeq \max(2^r, 2^{m/2-r/2})$
  $\rightsquigarrow$ optimal for $r = m/3$

# Finding $2^k$ collisions

## Idea

Repeat the quantum walk $2^k$ times.

# Finding $2^k$ collisions

## Idea

Repeat the quantum walk $2^k$ times.

## Issue

Repeat Ambainis's walk $2^k$ times is expensive.

# Finding $2^k$ collisions

## Idea

Repeat the quantum walk $2^k$ times.

## Issue

Repeat Ambainis's walk $2^k$ times is expensive.

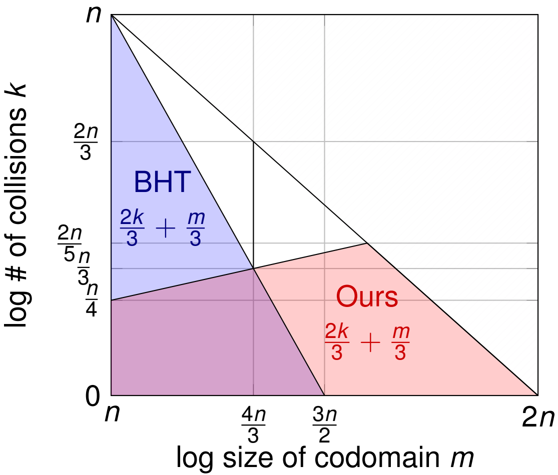
## Aim

At the end of each walk, extract collisions and preserve a useful quantum data structure $\rightsquigarrow$ new starting state of the next quantum walk.

$$2^k \cdot \left(\mathsf{S} + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)\right) \to \mathsf{S} + \frac{2^k}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)$$

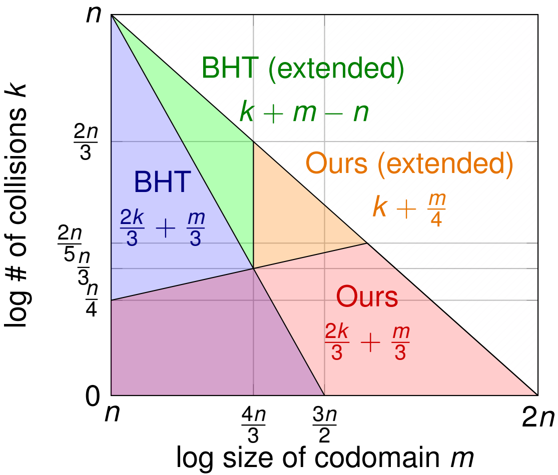
# Results

Find $2^k$ collision pairs of $f : \{0,1\}^n \to \{0,1\}^m$, with $k \le 2n - m$



# Results

Find $2^k$ collision pairs of $f : \{0,1\}^n \to \{0,1\}^m$, with $k \leq 2n - m$

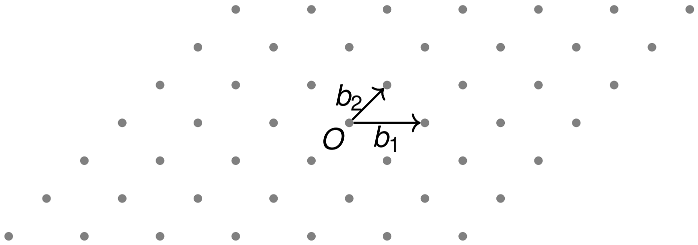


# Application to Lattice Sieving

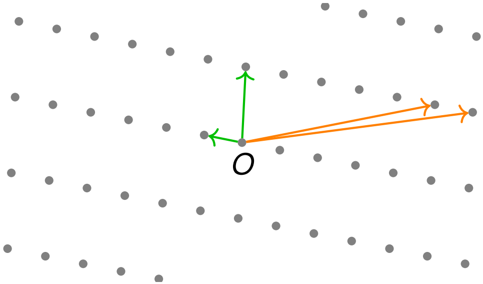
# What is a (Euclidean) lattice?

**Definition**

$\mathcal{L}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) = \left\{\sum_{i=1}^{n} x_i \boldsymbol{b}_i : x_i \in \mathbb{Z}\right\}$ where $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ is a basis of $\mathbb{R}^n$.

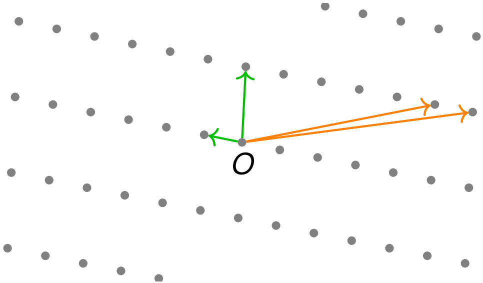


# Lattice-based cryptography: fundamental idea




- good basis: private information, makes problem easy
- bad basis: public information, makes problem hard

# Lattice-based cryptography: fundamental idea

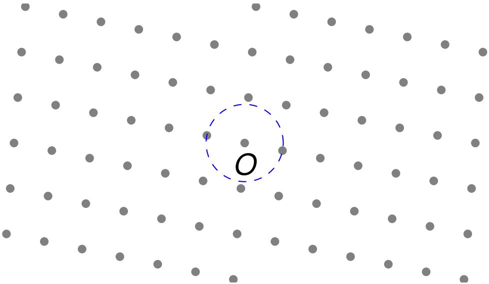



- good basis: private information, makes problem easy
- bad basis: public information, makes problem hard

Basis reduction: transform a bad basis into a good one
Main tool: BKZ algorithm and its variants

Requires to solve the (approx-)SVP problem in smaller dimensions.

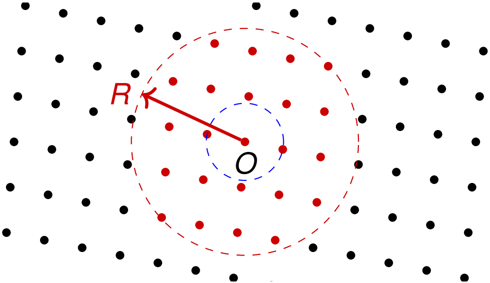
# The Shortest Vector Problem




The Shortest Vector Problem (SVP):

Given a basis, find a shortest nonzero vector in the lattice.

# The Shortest Vector Problem




The Shortest Vector Problem (SVP):

Given a basis, find a shortest nonzero vector in the lattice.

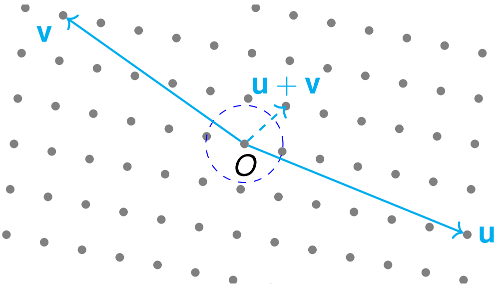
Two main approaches:

- enumeration:
  super-exponentiel time, polynomial space
- sieving:
  exponential time and space

**Enumeration**

1. choose a radius $R$
2. enumerate all vectors of length smaller than $R$
3. keep the shortest

# The Shortest Vector Problem




The Shortest Vector Problem (SVP):
Given a basis, find a shortest nonzero vector in the lattice.

Two main approaches:

- enumeration:
  super-exponentiel time, polynomial space
- sieving:
  exponential time and space

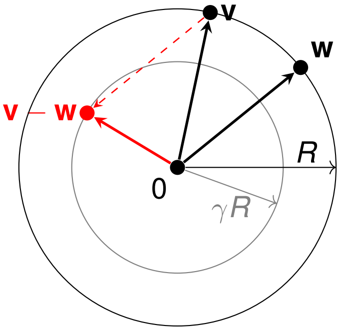
**Sieving**

1. generate a lot of random vectors
2. combine them recursively to reduce the length

# Heuristic Sieving Algorithms

## 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 $(\mathbf{v}, \mathbf{w}) \in L$ :
  if $\|\mathbf{v} - \mathbf{w}\| \leqslant \gamma R$ : add $\mathbf{v} - \mathbf{w}$ to $L'$ $\triangleright$ $(\mathbf{v}, \mathbf{w})$ is called a reducing vector pair.




### Fact

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

# Heuristic Sieving Algorithms

**Main heuristic**

At each sieving step, lattice vectors acts as random independent vectors of similar norm.

- We can pretend that they are randomly lying on the border of $R \cdot \mathcal{S}^d := \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| \leqslant R\}$.
- Validated by experiments.

# Sieving - Solving SVP [NV08]

## Solve SVP by sieving

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

Choose $\gamma = 1 - \frac{1}{\mathrm{poly}(d)}$

- initially: norm $\approx R$ (Gaussian vectors)
- 1st iteration: norm $\approx \gamma R$ (by heuristic)
- $K$-th iteration: norm $\approx \gamma^K R$

Only need $K = \mathrm{poly}(d)$ iterations.

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

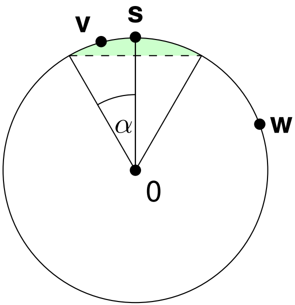
[1] Number chosen to get enough reducing vector pairs.

# Locality Sensitive Filtering [BDGL16]

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

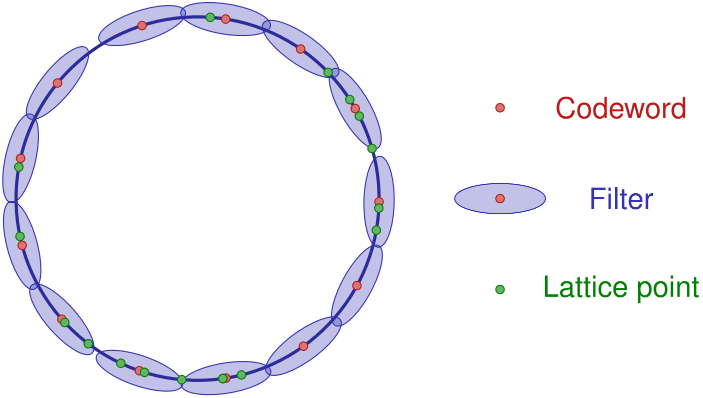
## $\alpha$-filter

A **filter** $f_{\mathbf{s},\alpha}$ of center $\mathbf{s} \in \mathbb{R}^d$ and angle $\alpha \in [0, \pi/2]$ only accepts vectors $\mathbf{v}$ such that $\theta(\mathbf{v}, \mathbf{s}) \leqslant \alpha$.



# Use of filtering for sieving [BDGL16]

- Choose an efficiently decodable code $C$
- Define $f$ : lattice points $\to C$




Look for pairs of vectors in the same filter, i.e $f(x) = f(y)$, to speed up search of reducing vector pairs.

# NV-sieve with LSF

## NV-sieve with LSF

1. Generate filters all over the sphere. $\triangleright$ centers = codewords
2. Add each vector to its nearest filters of angle at most $\alpha$.
3. Search reducing vector pairs inside each 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.
- Classical with LSF [BDGL16]: $2^{0.292d+o(d)}$ time.
- Quantum search with LSF [Laa16]: $2^{0.265d+o(d)}$ time.

# Double filtering [CL21]

Previous approach: Search reducing vector pairs inside each $\alpha$-filter.

New approach: first $\alpha$-filter, then $\beta$-filter in lower dimension inside each $\alpha$-filter

- vectors in the same $\alpha$-filter not necessary close enough
- exponentially many vectors in each list
- vectors in same ($\alpha$,$\beta$) filters lead to short vectors

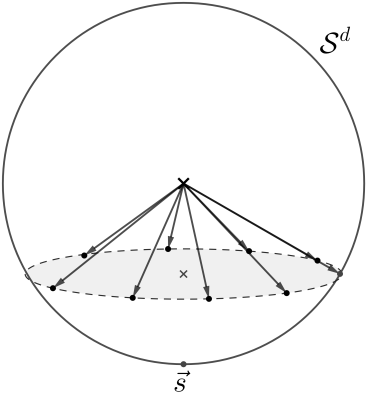
Leads to better quantum algorithm:

- Quantum search with LSF [Laa16]: $2^{0.265d+o(d)}$ time.
- Quantum wallk with LSF [CL21]: $2^{0.2570d+o(d)}$ time.

## Step 1 [CL21]

- Sample a code $C$ and generate the $\alpha$-filters.
- Insert each vector into its nearest $\alpha$-filters.

Each $\alpha$-filter contains $N^{c_\alpha}$ vectors, $c_\alpha \in [0, 1]$.




**Fact**

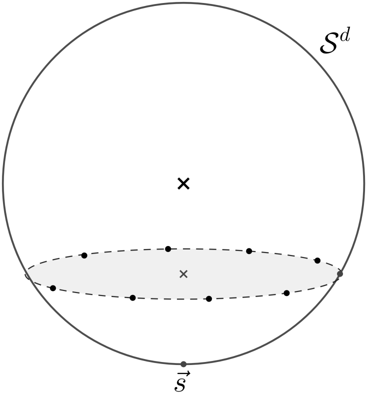
With high probability[1], lattice vectors fall on the border of the filter.

[1] This only works in high dimension, the picture in a bit counter-intuitive.

## Step 1 [CL21]

- Sample a code $C$ and generate the $\alpha$-filters.
- Insert each vector into its nearest $\alpha$-filters.

Each $\alpha$-filter contains $N^{c_\alpha}$ vectors, $c_\alpha \in [0, 1]$.




### Fact

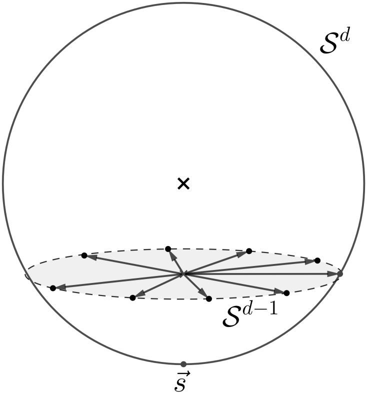
With high probability[1], lattice vectors fall on the border of the filter.

[1] This only works in high dimension, the picture in a bit counter-intuitive.

## Step 1 [CL21]

- Sample a code $C$ and generate the $\alpha$-filters.
- Insert each vector into its nearest $\alpha$-filters.

Each $\alpha$-filter contains $N^{c_\alpha}$ vectors, $c_\alpha \in [0, 1]$.



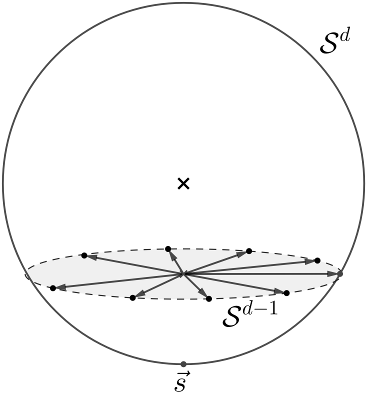

Consider the projection of the vectors on the the $(d-1)$-dimensional sphere defining the filter, called residuals.

[1] This only works in high dimension, the picture in a bit counter-intuitive.

## Step 1 [CL21]

- Sample a code $C$ and generate the $\alpha$-filters.
- Insert each vector into its nearest $\alpha$-filters.

Each $\alpha$-filter contains $N^{c_\alpha}$ vectors, $c_\alpha \in [0, 1]$.




Consider the projection of the vectors on the $(d-1)$-dimensional sphere defining the filter, called residuals.

### Fact

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

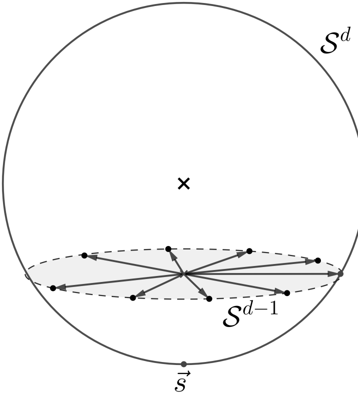
$$\theta(\mathbf{v}, \mathbf{w}) \leqslant \frac{\pi}{3} \Leftrightarrow \theta(\mathbf{v}_R, \mathbf{w}_R) \leqslant \theta^*_{\frac{\pi}{3}}.$$

[1] This only works in high dimension, the picture in a bit counter-intuitive.

## Step 1 [CL21]

- Sample a code $C$ and generate the $\alpha$-filters.
- Insert each vector into its nearest $\alpha$-filters.

Each $\alpha$-filter contains $N^{c_\alpha}$ vectors, $c_\alpha \in [0, 1]$.




### Fact

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

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

Same situation as before in dimension $d-1$: we can build $\beta = \frac{\theta^*_{\frac{\pi}{3}}}{2}$-filters on the residuals.

[1] This only works in high dimension, the picture in a bit counter-intuitive.

## Step 2 [CL21]

For each $\alpha$-filter : Walk on a Johnson graph of the list of the vectors.

1. VERTEX : A subset of $N^{c_V}$ vectors in the list.
2. Sample a code $C'$ and generate the $\beta$-filters.
   Insert each VERTEX's vector in its nearest $\beta$-filter.
3. Mark vertices that contain two vectors in the same $\beta$-filter (collisions).
4. Perform Quantum Random Walks to find all the marked vertices (reducing vector pairs) in the $\alpha$-filter.

With our improvement on quantum collision finding:

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

Quantum lattice sieving $2^{0.2570d} \to 2^{0.2563d}$

# Conclusion

- searching for collision is an important problem in cryptography
- quantum walks are very versatile
- with clever data structures, we can reuse previous searches' quantum states $\rightsquigarrow$ probably applies to other algorithms!
- Best quantum lattice sieving algorithm: $2^{0.2570d} \to 2^{0.2563d}$.

# Backup slides

# New algorithm

- Do a normal quantum walk

# New algorithm

- Do a normal quantum walk
- Measure the collisions

# New algorithm

- Do a normal quantum walk
- Measure the collisions
- Remove the collisions from the state

# New algorithm

- Do a normal quantum walk
- Measure the collisions
- Remove the collisions from the state
- Remaining state is now the uniform superposition of all nodes:
  - Without collision
  - Without any of the extracted inputs

# New algorithm

- Do a normal quantum walk
- Measure the collisions
- Remove the collisions from the state
- Remaining state is now the uniform superposition of all nodes:
  - Without collision
  - Without any of the extracted inputs
- Do a new march on a smaller Johnson graph:
  - With smaller sets
  - In a smaller ambient set (avoid the extracted preimages)

# New algorithm

- Do a normal quantum walk
- Measure the collisions
- Remove the collisions from the state
- Remaining state is now the uniform superposition of all nodes:
  - Without collision
  - Without any of the extracted inputs
- Do a new march on a smaller Johnson graph:
  - With smaller sets
  - In a smaller ambient set (avoid the extracted preimages)

Complexity:

$$\widetilde{\mathcal{O}}\left(2^r + 2^k 2^{m/2-r/2}\right) = \widetilde{\mathcal{O}}\left(2^{k+m/2-r/2}\right)$$

where $r \leq \min(2k/3 + m/3, m/2)$.

# With larger $m$

## BHT algorithm

- Take a list $L = (f(y_0), \ldots, f(y_{2^r}))$
- (Grover) Search for an $x$ with $f(x) = f(y_i)$ and $x \neq y_i$

- Only $2^{2n-m}$ inputs are part of a collision
- Each element has probability $2^{n-m}$ to be part of a collision.
- In the initial list, $\mathcal{O}\left(2^{r-m+n}\right)$ elements are part of a collision.
- Output $2^k$ collision pairs, need $r - m + n \geq k$, otherwise the list might contain no relevant input
- Cost $2^r$ memory, $2^r + 2^k \times \sqrt{\frac{2^n}{2^{r-m+n}}}$ time $\rightsquigarrow 2^{m/3+2k/3}$ time for $k \leq 3n - 2m$.

# Quantum data structures

- Need efficient data structures to allow poly time Update and Test

# Quantum data structures

- Need efficient data structures to allow poly time Update and Test
- To have a proper interference, one node must be represented by a single quantum state

# Quantum data structures

- Need efficient data structures to allow poly time Update and Test
- To have a proper interference, one node must be represented by a single quantum state
- This state must not depend on the path in the graph

# Quantum data structures

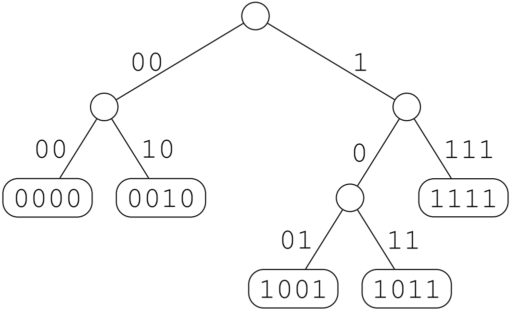
- Need efficient data structures to allow poly time Update and Test
- To have a proper interference, one node must be represented by a single quantum state
- This state must not depend on the path in the graph

# Quantum data structures

- Need efficient data structures to allow poly time Update and Test
- To have a proper interference, one node must be represented by a single quantum state
- This state must not depend on the path in the graph

## History-free quantum data structures: step 1

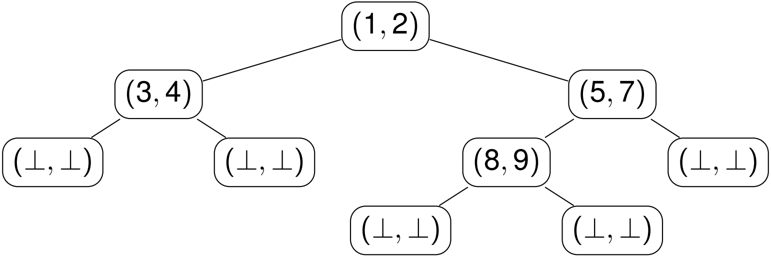
Take an efficient classical data structure (ex. radix tree)




Tree representing $\{0000, 0010, 1001, 1011, 1111\}$.

# Representing a tree

Node: (addr left, addr right)



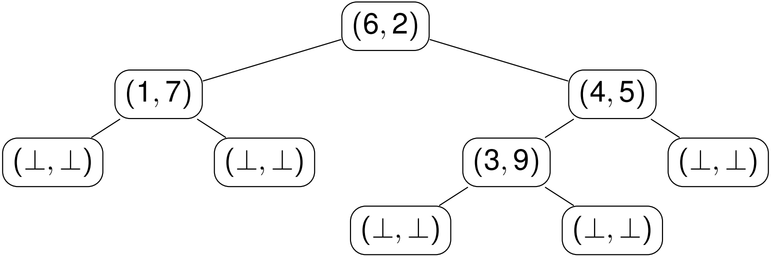

Actual memory content:

| | | | | | |
|---|---|---|---|---|---|
| 0 | $(1,2)$ | 1 | $(3,4)$ | 2 | $(5,7)$ |
| 3 | $(\perp,\perp)$ | 4 | $(\perp,\perp)$ | 5 | $(8,9)$ |
| 6 | Empty cell | 7 | $(\perp,\perp)$ | 8 | $(\perp,\perp)$ |
| 9 | $(\perp,\perp)$ | | | | |

Address sequence: $(0,1,3,4,2,5,8,9,7)$

# Representing a tree

Node: (addr left, addr right)




Actual memory content:

| | | | | | |
|---|---|---|---|---|---|
| 0 | $(6,2)$ | 1 | $(\perp,\perp)$ | 2 | $(4,5)$ |
| 3 | $(\perp,\perp)$ | 4 | $(3,9)$ | 5 | $(\perp,\perp)$ |
| 6 | $(1,7)$ | 7 | $(\perp,\perp)$ | 8 | Empty cell |
| 9 | $(\perp,\perp)$ | | | | |

Address sequence: $(0,6,1,7,2,4,3,9,5)$

# Quantum tree: problems and solutions

- a tree can have several representations
- the representation depends on the memory allocations
- the memory allocations are history dependent

# Quantum tree: problems and solutions

- a tree can have several representations
- the representation depends on the memory allocations
- the memory allocations are history dependent

Solution: superposition of all possible representations

$$|T(S)\rangle = \sum_{\text{address sequences } \boldsymbol{I}} |T_I(S)\rangle \quad .$$

Lots of details:

- allocate memory uniformly at random
- need data structure for free cells
- need extra data in nodes for efficient operations/collision extraction

# Quantum tree: problems and solutions

- a tree can have several representations
- the representation depends on the memory allocations
- the memory allocations are history dependent

Solution: superposition of all possible representations

$$|T(S)\rangle = \sum_{\text{address sequences } I} |T_I(S)\rangle \quad .$$

Lots of details:

- allocate memory uniformly at random
- need data structure for free cells
- need extra data in nodes for efficient operations/collision extraction

**Summary**

Make update operations **polynomial time** and **history independent**