Finding many collisions via quantum walks

Xavier Bonnetain André Chailloux André Schrottenloher
Yixin Shen

November 2, 2022

Classical Collision Finding

Classical collision algorithms

$$f : \{0,1\}^n \to \{0,1\}^n$$

A first algorithm

- Create a sorted list of size $2^{n/2}$
- Look for collisions

Classical collision algorithms

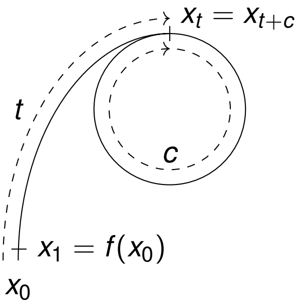
$$f : \{0,1\}^n \to \{0,1\}^n$$

A first algorithm

- Create a sorted list of size $2^{n/2}$
- Look for collisions

Pollard-Rho

- The cyclic part of the functional graph of f is of size $\Theta(2^{n/2})$
- Seek a collision in the graph
- Memory-less, parallelizable



Other cases

$f : \{0,1\}^n \to \{0,1\}^m$, $n \le m \le 2n$

Same algorithms, complexity $2^{m/2}$.

Other cases

$f : \{0,1\}^n \to \{0,1\}^m$, $n \le m \le 2n$

Same algorithms, complexity $2^{m/2}$.

Finding 2^k collisions instead of 1

List of size $2^{k/2+m/2}$ contains 2^k collisions on average

Other cases

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

Same algorithms, complexity $2^{m/2}$.

Finding 2^k collisions instead of 1

List of size $2^{k/2+m/2}$ contains 2^k collisions on average

Lower bounds

Matching query lower bound in all cases

Quantum Collision Finding

Quantum Search

Operators

- Test: $|x\rangle \mapsto -|x\rangle$ if x marked, identity otherwise
- Diffusion: $\sum_{i=0}^{2^n-1}|i\rangle \mapsto -\sum_{i=0}^{2^n-1}|i\rangle$, identity otherwise

Principle

Use the test operator to differentiate marked elements from the others, and use the diffusion operator to make marked elements interfere constructively.

Quantum Search

Operators

- Test: $|x\rangle \mapsto -|x\rangle$ if x marked, identity otherwise
- Diffusion: $\sum_{i=0}^{2^n-1} |i\rangle \mapsto -\sum_{i=0}^{2^n-1} |i\rangle$, identity otherwise

Principle

Use the test operator to differentiate marked elements from the others, and use the diffusion operator to make marked elements interfere constructively.

Algorithm

- Test and Diffusion are reflexions in the same plane
- Composition is a rotation of angle $2 \arcsin \sqrt{\epsilon}$
- Need to repeat until angle is $\sim \pi/2$

Quantum Collision Finding via Quantum Search

Pollard-Rho

- Still need to compute $f \circ \cdots \circ f$
- No quantum improvement!

Quantum Collision Finding via Quantum Search

Pollard-Rho

- Still need to compute $f \circ \cdots \circ f$
- No quantum improvement!

What works: a list + an enumeration (BHT algorithm)

- Take a list $L = (f(y_0), \ldots, f(y_{2^u}))$
- Search for an x such that there exists i with $f(x) = f(y_i)$ and $x \neq y_i$
- Cost 2^u memory, $2^u + \sqrt{\frac{2^n}{2^u}}$ time

BHT

BHT algorithm

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

Finding 2^k collisions

- 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 $\Omega\left(2^{m/3+2k/3}\right)$

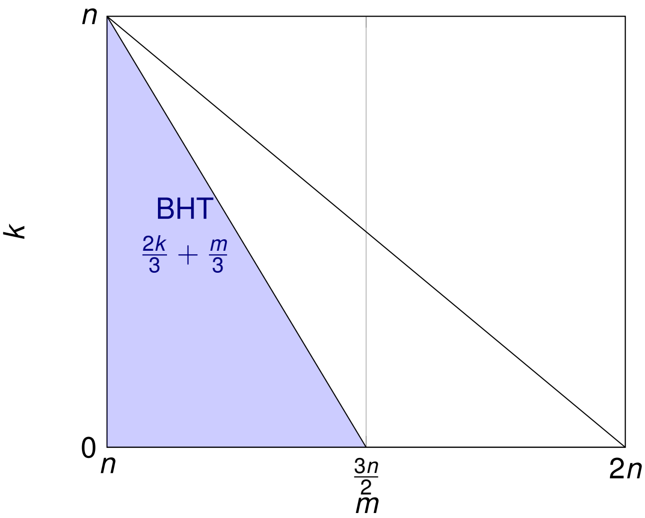
With larger m

BHT algorithm

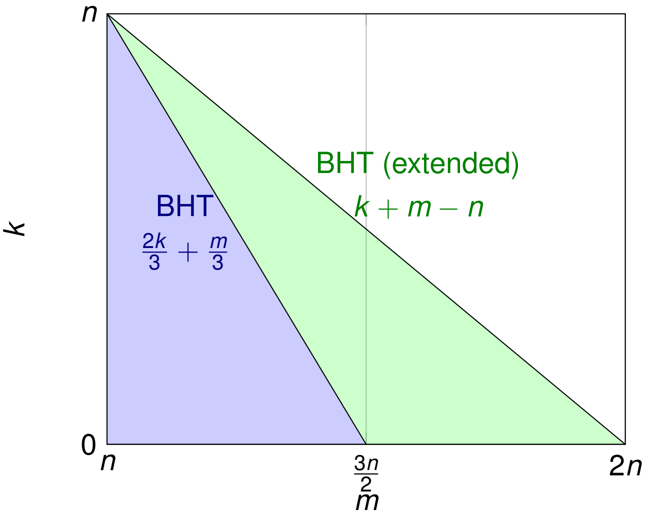
Query a list of size 2^{ℓ}

- Only 2^{2n-m} inputs are part of a collision
- Each element has probability 2^{n-m} to be in a collision pair
- $\mathcal{O}\left(2^{\ell-m+n}\right)$ collisions pairs for the initial list.
- Output 2^k collision pairs, need $\ell - m + n \geq k$, otherwise the list might contain no relevant input $\rightsquigarrow m \leq \frac{3}{2}n$

Summary



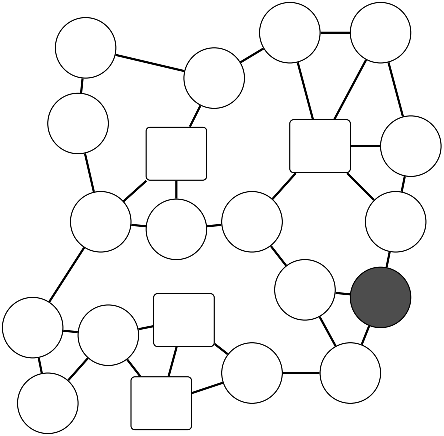
Summary



Collision Finding
via Random Walks

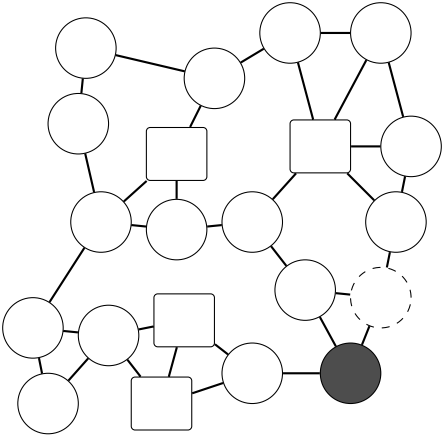
Classical random walk

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



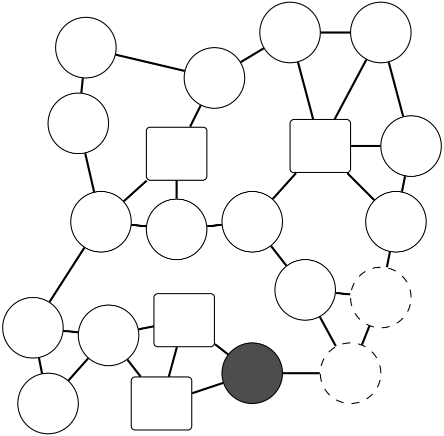
Classical random walk

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



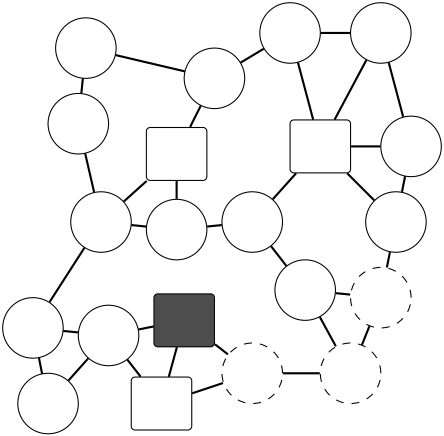
Classical random walk

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



Classical random walk

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



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 (trivial) (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 is $|N_1 \cap N_2| = 2^r - 1$
- $\frac{1}{\delta} = \frac{2^r(2^n - 2^r)}{2^n} \simeq 2^r$ (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 is $|N_1 \cap N_2| = 2^r - 1$
- $\frac{1}{\delta} = \frac{2^r(2^n - 2^r)}{2^n} \simeq 2^r$ (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 is $|N_1 \cap N_2| = 2^r - 1$
- $\frac{1}{\delta} = \frac{2^r(2^n - 2^r)}{2^n} \simeq 2^r$ (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

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

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 is $|N_1 \cap N_2| = 2^r - 1$
- $\frac{1}{\delta} = \frac{2^r(2^n - 2^r)}{2^n} \simeq 2^r$ (We need to replace all elements.)

Collision finding with Johnson graph

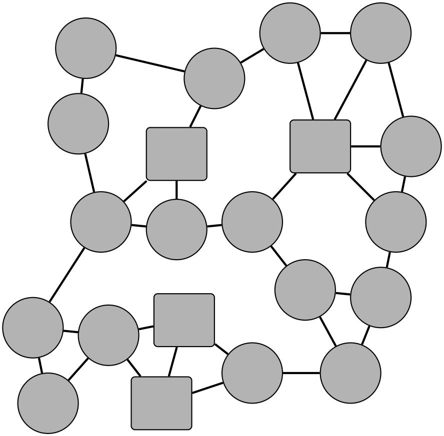
- Create a random list of elements of size 2^r
- Repeat until a collision is found:
  - Walk 2^r times
  - Check whether the node contains a collision

Complexity

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

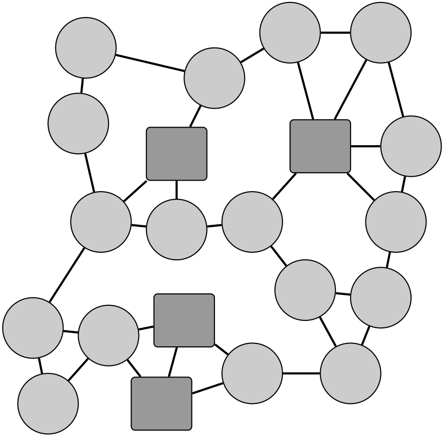
Quantum walk

As in quantum search, the walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.



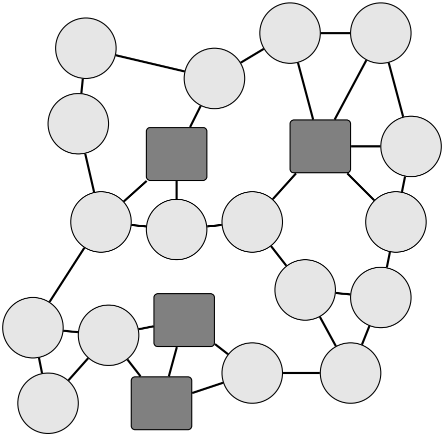
Quantum walk

As in quantum search, the walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.



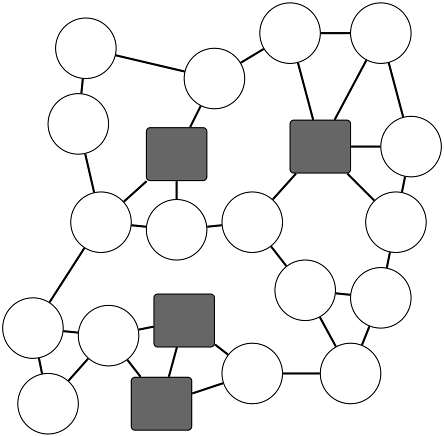
Quantum walk

As in quantum search, the walk transforms a uniform superposition over the whole graph into a superposition over marked vertices.



Quantum walk

As in quantum search, the 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

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

MNRS walk in a Johnson graph

- Setup : Creat 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 Test 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 : Creat 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 Test 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 $r = m/3$

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

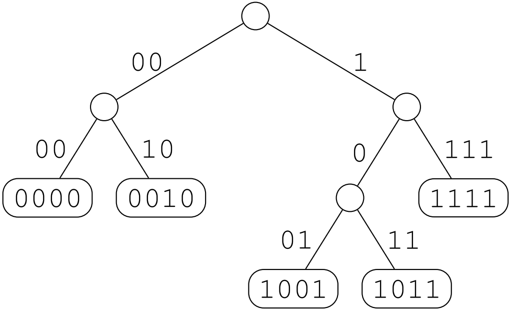
- Need efficient data structures to allow poly time Update and Test
- To have a proper interference, one node must be represented by a single quantum state
- This state must not depend on the path in the graph

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

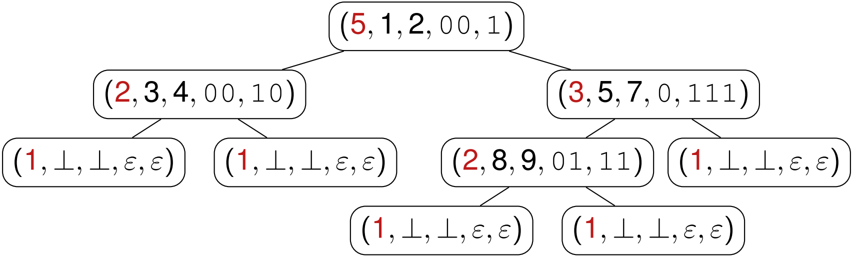
Take an efficient classical data structure (ex. radix tree)



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

Quantum data structures

Example for set $S = \{\texttt{0000}, \texttt{0010}, \texttt{1001}, \texttt{1011}, \texttt{1111}\}$:



Format: (# leaves, addr left, addr right, `label left`, `label right`)

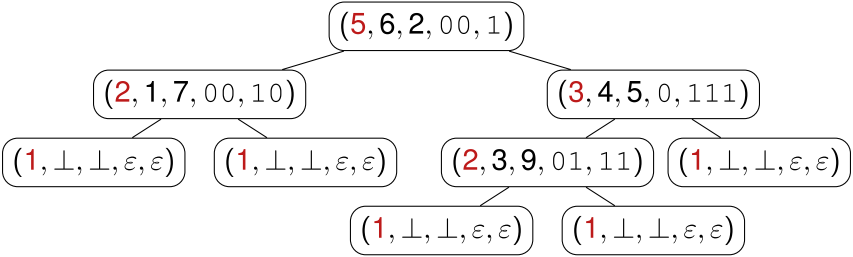
Actual memory content:

0	$(5, 1, 2, \texttt{00}, \texttt{1})$	1	$(2, 3, 4, \texttt{00}, \texttt{10})$	2	$(3, 5, 7, \texttt{10}, \texttt{11})$
3	$(1, \perp, \perp, \varepsilon, \varepsilon)$	4	$(1, \perp, \perp, \varepsilon, \varepsilon)$	5	$(2, 8, 9, \texttt{01}, \texttt{11})$
6	Empty cell	7	$(1, \perp, \perp, \varepsilon, \varepsilon)$	8	$(1, \perp, \perp, \varepsilon, \varepsilon)$
9	$(1, \perp, \perp, \varepsilon, \varepsilon)$				

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

Quantum data structures

Example for set $S = \{\texttt{0000}, \texttt{0010}, \texttt{1001}, \texttt{1011}, \texttt{1111}\}$:



Format: (# leaves, addr left, addr right, `label left`, `label right`)

Actual memory content:

0	$(5, 6, 2, \texttt{00}, \texttt{1})$	1	$(1, \perp, \perp, \varepsilon, \varepsilon)$	2	$(3, 4, 5, \texttt{10}, \texttt{11})$
3	$(1, \perp, \perp, \varepsilon, \varepsilon)$	4	$(2, 3, 9, \texttt{01}, \texttt{11})$	5	$(1, \perp, \perp, \varepsilon, \varepsilon)$
6	$(2, 1, 7, \texttt{00}, \texttt{10})$	7	$(1, \perp, \perp, \varepsilon, \varepsilon)$	8	Empty cell
9	$(1, \perp, \perp, \varepsilon, \varepsilon)$				

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

Quantum memory allocation

We work with

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

Problem when inserting a new node

Compute the uniform superposition of all free memory cells

Quantum memory allocation

We work with

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

Problem when inserting a new node

Compute the uniform superposition of all free memory cells

Approach 1: Random guess

Store in 1 qubit per cell its availability. Do a quantum search to compute the superposition of available cells.

Quantum memory allocation

We work with

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

Problem when inserting a new node

Compute the uniform superposition of all free memory cells

Approach 1: Random guess

Store in 1 qubit per cell its availability. Do a quantum search to compute the superposition of available cells.

Approach 2 : Memory tree

Maintain a tree with each memory cell as a leaf. Count for each node the number of free cells.

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.

New algorithm

- Begin with a normal quantum walk

New algorithm

- Begin with a normal quantum walk
- Measure the number of collisions we have in the end

New algorithm

- Begin with a normal quantum walk
- Measure the number of collisions we have in the end
- Remove reversibly this number of collision from the tree

New algorithm

- Begin with a normal quantum walk
- Measure the number of collisions we have in the end
- Remove reversibly this number of collision from the tree
- Measure the extracted collisions

New algorithm

- Begin with a normal quantum walk
- Measure the number of collisions we have in the end
- Remove reversibly this number of collision from the tree
- Measure the extracted collisions
- Final state is now the uniform superposition of all nodes:
  - Without collision
  - Without any of the extracted inputs

New algorithm

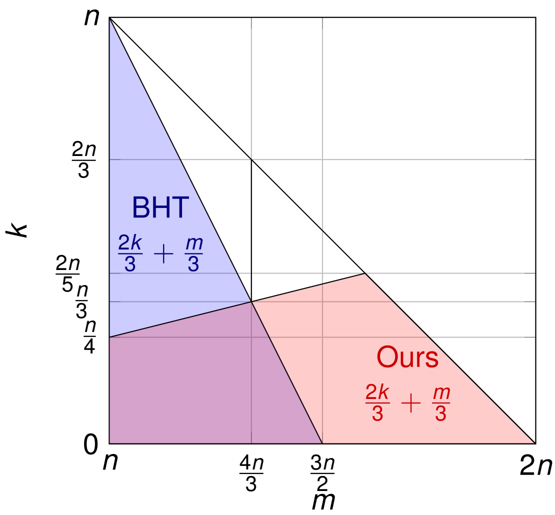
- Begin with a normal quantum walk
- Measure the number of collisions we have in the end
- Remove reversibly this number of collision from the tree
- Measure the extracted collisions
- Final state is now the uniform superposition of all nodes:
  - Without collision
  - Without any of the extracted inputs
- Do a new march on a smaller Johnson graph:
  - With smaller sets (-collisions)
  - In a smaller ambient set (avoid the extracted preimages)

New algorithm

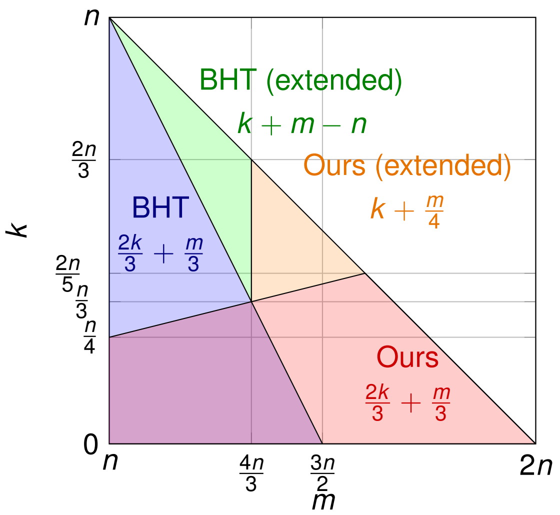
- Begin with a normal quantum walk
- Measure the number of collisions we have in the end
- Remove reversibly this number of collision from the tree
- Measure the extracted collisions
- Final state is now the uniform superposition of all nodes:
  - Without collision
  - Without any of the extracted inputs
- Do a new march on a smaller Johnson graph:
  - With smaller sets (-collisions)
  - In a smaller ambient set (avoid the extracted preimages)

Complexity: $\widetilde{\mathcal{O}}\left(2^{\ell} + 2^{k}2^{m/2-\ell/2}\right) = \widetilde{\mathcal{O}}\left(2^{k+m/2-\ell/2}\right)$ where $\ell \leq \min(2k/3 + m/3, m/2)$.

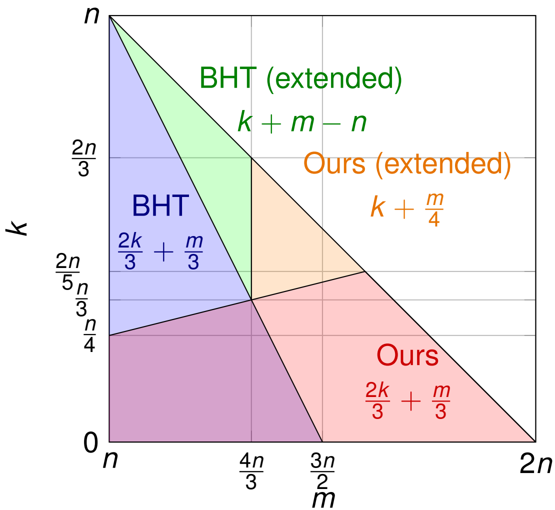
Results


n
$\frac{2n}{3}$
k
BHT
$\frac{2k}{3} + \frac{m}{3}$
$\frac{2n}{5}$
$\frac{n}{3}$
$\frac{n}{4}$
Ours
$\frac{2k}{3} + \frac{m}{3}$
0
n
$\frac{4n}{3}$
$\frac{3n}{2}$
$2n$
m

Results


BHT (extended)
$k + m - n$
Ours (extended)
$k + \frac{m}{4}$
BHT
$\frac{2k}{3} + \frac{m}{3}$
Ours
$\frac{2k}{3} + \frac{m}{3}$
n
$\frac{2n}{3}$
$\frac{2n}{5}$
$\frac{n}{3}$
$\frac{n}{4}$
0
k
n
$\frac{4n}{3}$
$\frac{3n}{2}$
$2n$
m

Results



Many applications:

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