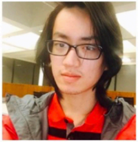
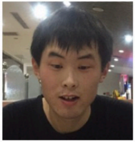
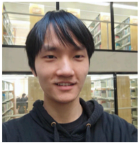
Truly Low-Space Element Distinctness and Subset Sum via Pseudorandom Hash Functions

Lijie Chen[1], Ce Jin[1], R. Ryan Williams[1], and Hongxun Wu[2]

[1] MIT
[2] Tsinghua University



Element Distinctness

Element Distinctness

42	3	23	1	12	30	42	15

- INPUT: n positive integers $a_1, a_2, \ldots, a_n$ with $a_i \in [m], m \leq \text{poly}(n)$[1].

[1] Here $[m] = \{1, 2, \ldots, m\}$.

Element Distinctness

- INPUT: n positive integers $a_1, a_2, \ldots, a_n$ with $a_i \in [m]$, $m \leq \text{poly}(n)$[1].
- Decide whether all a_i's are distinct.

[1] Here $[m] = \{1, 2, \ldots, m\}$.

Element Distinctness

- INPUT: n positive integers $a_1, a_2, \ldots, a_n$ with $a_i \in [m], m \leq \text{poly}(n)$[1].
- Decide whether all a_i's are distinct.
- With linear space, we can simply sort the integers.

[1] Here $[m] = \{1, 2, \ldots, m\}$.

Element Distinctness

- INPUT: n positive integers $a_1, a_2, \ldots, a_n$ with $a_i \in [m], m \leq \text{poly}(n)$[1].
- Decide whether all a_i's are distinct.
- Here we consider the **low-space** regime where $S = O(\text{polylog}\, n)$.

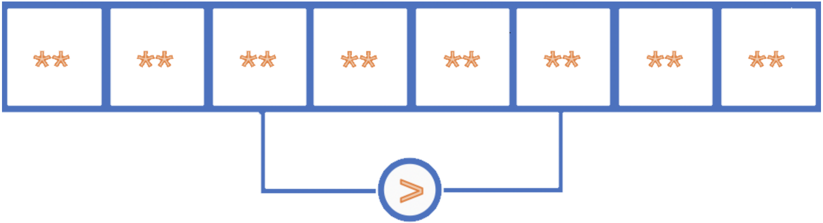
[1] Here $[m] = \{1, 2, \ldots, m\}$.

Element Distinctness

- INPUT: n positive integers $a_1, a_2, \ldots, a_n$ with $a_i \in [m]$, $m \le \text{poly}(n)$[1].
- Decide whether all a_i's are distinct.
- Here we consider the **low-space** regime where $S = O(\text{polylog}\, n)$.
- Brute force takes $T = O(n^2)$ time.

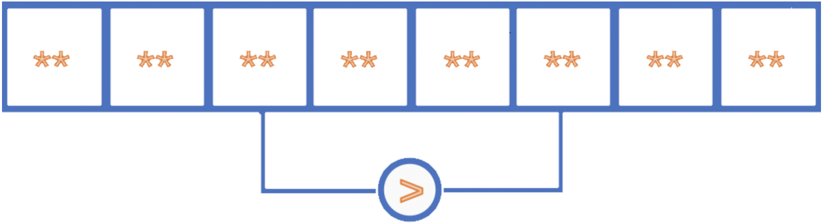
[1] Here $[m] = \{1, 2, \ldots, m\}$.

Comparison Model



- No direct access to the INPUT a.
- Each query (i, j) returns one of "$a_i < a_j$", "$a_i = a_j$", "$a_i > a_j$".

Comparison Model



Theorem (Borodin et al., 1987) (Yao, 1988)

When space $S = O(\text{polylog}\, n)$, Element Distinctness requires $T \geq n^{2-o(1)}$ time in comparison model.

More generally, $TS \geq n^{2-o(1)}$ (Yao, 1988).

RAM model

- Random access to read-only input. Allow arbitrary arithmetic and bit operations.
- Surprisingly, in RAM model, one can bypass the $n^{2-o(1)}$ barrier! (Beame, Clifford, and Machmouchi, 2013)

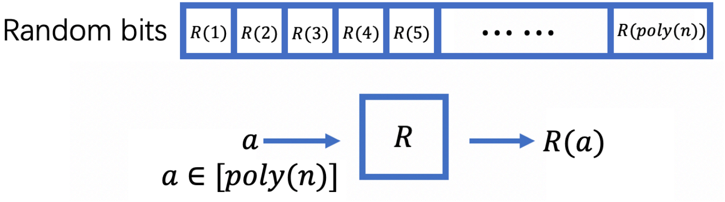
RAM model

Theorem (Beame, Clifford, and Machmouchi, 2013)

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

More generally, $T^2 S = \tilde{O}(n^3)$.

Random Oracle Model



- Random access to poly(n) random bits which do not count into space complexity.

Our Results - 1

Our Result: Element Distinctness

~~Assuming a *Random Oracle*~~, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

Our Results - 1

Our Result: Element Distinctness

~~Assuming a *Random Oracle*~~, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- We construct a pseudorandom hash function family with $O(\text{polylog}\, n)$ seed length to replace the Random Oracle.

Our Results - 1

Our Result: Element Distinctness

~~Assuming a *Random Oracle*~~, Element Distinctness can be solved in $S = O(\text{polylog } n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- We construct a pseudorandom hash function family with $O(\text{polylog } n)$ seed length to replace the Random Oracle.
- In order to explain our result, let's first review BCM algorithm.

Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

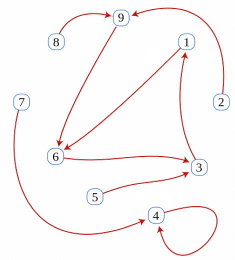
- INPUT: $a_1, a_2, \ldots, a_n \in [m]$.
  Take a random oracle $R : [m] \to [n]$.

Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- INPUT: $a_1, a_2, \ldots, a_n \in [m]$.
  Take a random oracle $R : [m] \to [n]$.
- Implicitly define the directed functional graph $\mathcal{G}_R$ with
  - vertex set $\{1, 2, \ldots, n\}$
  - one outgoing edge $x \mapsto R(a_x)$ for each vertex.

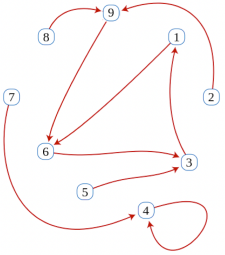


Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog } n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

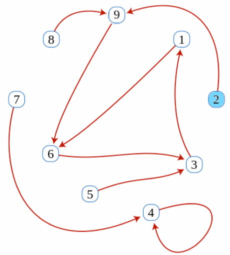
- INPUT: $a_1, a_2, \ldots, a_n \in [m]$.
  Take a random oracle $R : [m] \to [n]$.
- Implicitly define the directed functional graph $\mathcal{G}_R$ with
  - vertex set $\{1, 2, \ldots, n\}$
  - one outgoing edge $x \mapsto R(a_x)$ for each vertex.
- If $a_x = a_y$, x and y must point to the same vertex in $\mathcal{G}_R$.



Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog } n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

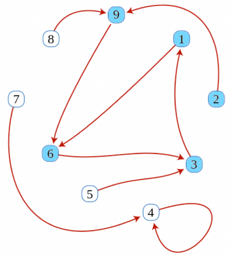
- Pick a random starting point s.



Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog } n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- Pick a random starting point s.
- The vertices reachable from s form a ρ-shape.

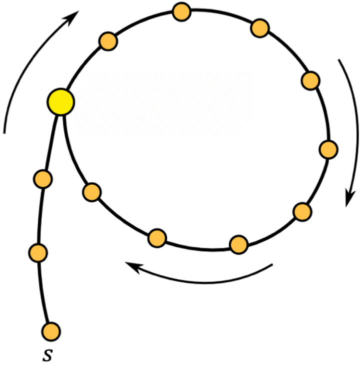


Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog } n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- Pick a random starting point s.
- The vertices reachable from s form a ρ-shape.

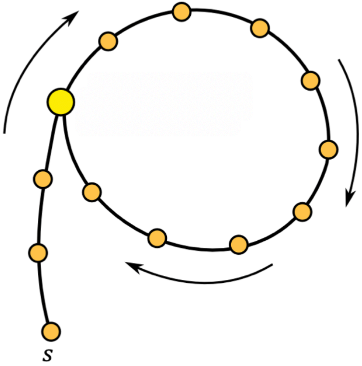


Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- Perform Floyd's cycle finding from s.

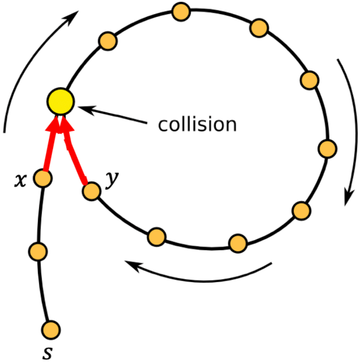


Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- Perform Floyd's cycle finding from s.
- It takes $O(\log n)$ space and returns $x \neq y$ s.t. $R(a_x) = R(a_y)$.

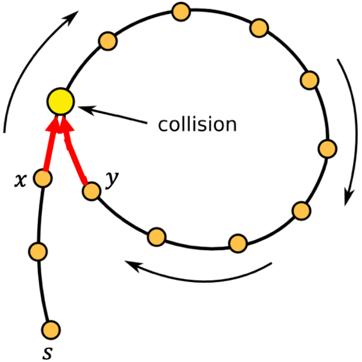


Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog}\, n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- Such (x, y) is either
  - a hash collision : $a_x \neq a_y$ but $R(a_x) = R(a_y)$.

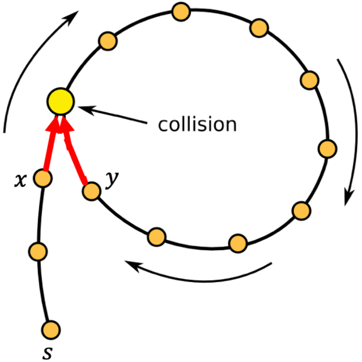


Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog } n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- Such (x, y) is either
  - a hash collision : $a_x \neq a_y$ but $R(a_x) = R(a_y)$.
  - a "real" collision : $a_x = a_y$.

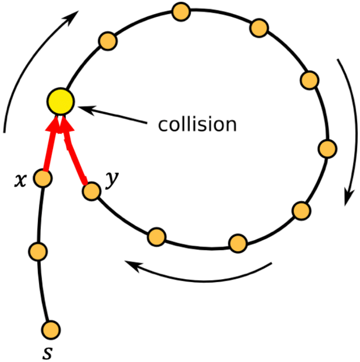


Sketch of BCM - Part 1

Pollard's ρ type algorithm [BCM13]

Assuming a *Random Oracle*, Element Distinctness can be solved in $S = O(\text{polylog } n)$ space and $T = \tilde{O}\left(n^{1.5}\right)$ time in RAM model.

- Such (x, y) is either
  - a hash collision : $a_x \neq a_y$ but $R(a_x) = R(a_y)$.
  - a "real" collision : $a_x = a_y$.
- For any "real" collision (x, y), it is found iff x, y are reachable from s.

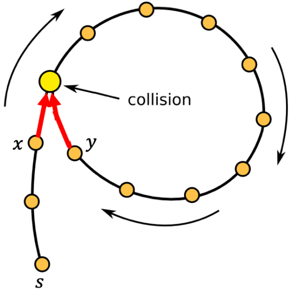


Sketch of BCM - Part 2

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

W.l.o.g. assume that there is only one pair of $x < y$, $a_x = a_y$.



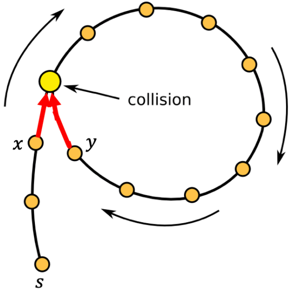
Sketch of BCM - Part 2

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\underset{R,s}{\mathbb{E}}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$

W.l.o.g. assume that there is only one pair of $x < y$, $a_x = a_y$.



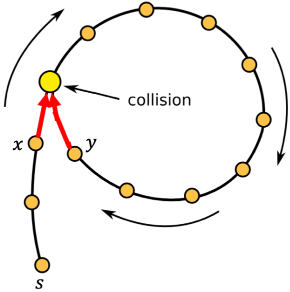
Sketch of BCM - Part 2

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathop{\mathbb{E}}_{R,s}[\#\text{vertices reachable from } s] \le O(\sqrt{n})$
- $\Pr_{R,s}[u, v \text{ are reachable from } s] \ge \Omega(1/n), \ \forall u, v \in [n]$

W.l.o.g. assume that there is only one pair of $x < y$, $a_x = a_y$.



Sketch of BCM - Part 2

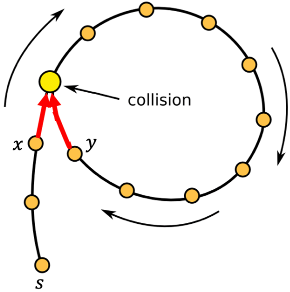
Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathbb{E}_{R,s}[\#\text{vertices reachable from } s] \le O(\sqrt{n})$
- $\Pr_{R,s}[u, v \text{ are reachable from } s] \ge \Omega(1/n), \ \forall u, v \in [n]$

W.l.o.g. assume that there is only one pair of $x < y$, $a_x = a_y$.

- So each cycle-finding takes $O(\sqrt{n})$ time.



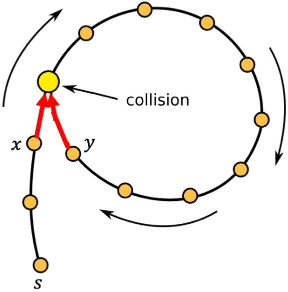
Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathop{\mathbb{E}}_{R,s}[\#\text{vertices reachable from } s] \le O(\sqrt{n})$
- $\Pr_{R,s}[u, v \text{ are reachable from } s] \ge \Omega(1/n),\ \forall u, v \in [n]$

W.l.o.g. assume that there is only one pair of $x < y$, $a_x = a_y$.

- So each cycle-finding takes $O(\sqrt{n})$ time.
- For the "real" collision, we find it with probability $\Omega(1/n)$.



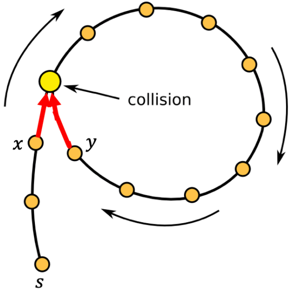
Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathbb{E}_{R,s}[\#\text{vertices reachable from } s] \le O(\sqrt{n})$
- $\Pr_{R,s}[u, v \text{ are reachable from } s] \ge \Omega(1/n), \ \forall u, v \in [n]$

W.l.o.g. assume that there is only one pair of $x < y$, $a_x = a_y$.

- So each cycle-finding takes $O(\sqrt{n})$ time.
- For the "real" collision, we find it with probability $\Omega(1/n)$.
- Repeat $\tilde{O}(n)$ times. In total, $\tilde{O}(n^{1.5})$ time.



Our Results - 2

Our Main Lemma

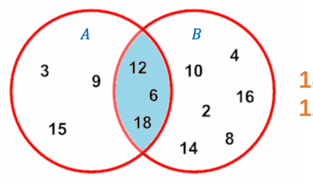
There exsits a family $\{h_{\text{seed}}\}$ of pseudorandom hash functions with **seed length** $O(\log^3 n \log\log n)$, such that functional graph $\mathcal{G}_h : x \mapsto h_{\text{seed}}(a_x)$ satisfies

- $\mathop{\mathbb{E}}_{s,seed}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$
- $\Pr_{s,seed}[u, v \text{ are reachable from } s] \geq \Omega(1/n),\ \forall u, v \in [n]$

W.l.o.g. assume that there is only one pair of $x < y$, $a_x = a_y$.

Out Results - 3

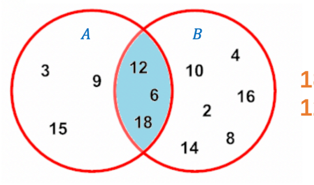
Set Intersection: Given two integer sets A, B, print all the elements in $A \cap B$ in any order. Each element is allowed to be printed multiple times.



18, 6, 12,
12, 6, 18

Out Results - 3

Set Intersection: Given two integer sets A, B, print all the elements in $A \cap B$ in any order. Each element is allowed to be printed multiple times.



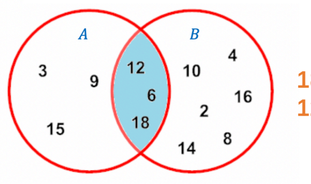
18, 6, 12, 12, 6, 18

Our RAM Upper Bound

Set Intersection can be solved in $O(\text{polylog}\, n)$ space and $\tilde{O}(n^{1.5})$ time.

Out Results - 3

Set Intersection: Given two integer sets A, B, print all the elements in $A \cap B$ in any order. Each element is allowed to be printed multiple times.



18, 6, 12, 12, 6, 18

Our RAM Upper Bound

Set Intersection can be solved in $O(\text{polylog}\, n)$ space and $\tilde{O}(n^{1.5})$ time.

RAM Lower bound (Patt-Shamir and Peleg, 1993) (Dinur, 2020)

$O(\text{polylog}\, n)$ space algorithms for Set Intersection require $\tilde{\Omega}(n^{1.5})$ time.

Our Results - 4

Subset Sum: Given n integers $a_1, a_2, \ldots, a_n$ and target t, decide whether a subset of them sum up to t.

Low-space Subset Sum (Bansal, Garg, Nederlof, and Vyas, 2017)

Assuming a *Random Oracle*, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $2^{0.87n}$ time, with $O(\text{poly}(n))$ space.

Subset Sum: Given n integers $a_1, a_2, \ldots, a_n$ and target t, decide whether a subset of them sum up to t.

Low-space Subset Sum (Bansal, Garg, Nederlof, and Vyas, 2017)

Assuming a *Random Oracle*, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $2^{0.87n}$ time, with $O(\text{poly}(n))$ space.

Our Result

~~Assuming a *Random Oracle*~~, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $2^{0.87n}$ time, with $O(\text{poly}(n))$ space.

Constructing Pseudorandom Hash Function

Analysis for Random Oracle

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

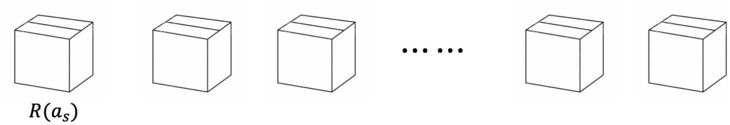
- $\mathop{\mathbb{E}}_{R,s}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$

Analysis for Random Oracle

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

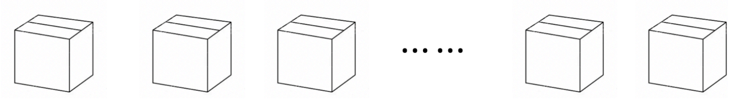
- $\underset{R,s}{\mathbb{E}}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$



Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathop{\mathbb{E}}_{R,s}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$



$R(a_s)$

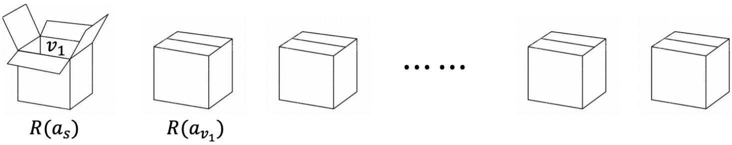
First step, $s \to v_1 = R(a_s)$.
For any $x \in [n]$,

$$\Pr_{s,R}[R(a_s) = x] = \frac{1}{n}.$$

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathop{\mathbb{E}}_{R,s}[\#\text{vertices reachable from } s] \le O(\sqrt{n})$



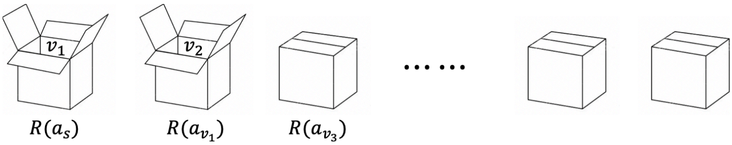
Second step, $v_1 \to v_2 = R(a_{v_1})$.
Given $a_s \neq a_{v_1}$, for any $x \in [n]$,

$$\Pr_{s,R}[R(a_{v_1}) = x \mid R(a_s) = v_1] = \frac{1}{n}.$$

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathbb{E}_{R,s}[\#\text{vertices reachable from } s] \le O(\sqrt{n})$

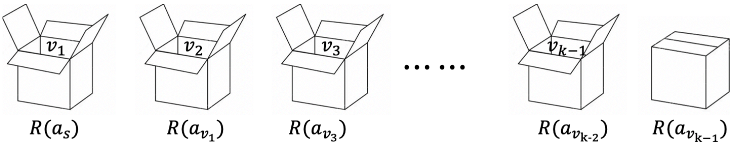


Third step, $v_2 \to v_3 = R(a_{v_2})$.

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathop{\mathbb{E}}_{R,s}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$



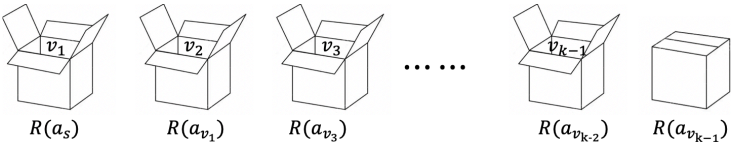
k-th step, $v_{k-1} \to v_k = R(a_{v_{k-1}})$.

Analysis for Random Oracle

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathbb{E}_{R,s}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$

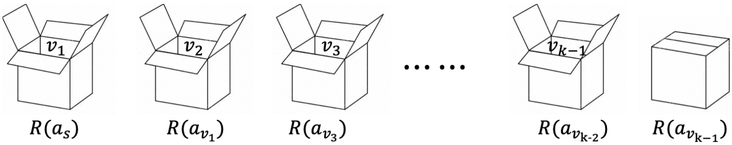


- After opening $k-1$ boxes, the k-th one still has to be random.

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathop{\mathbb{E}}_{R,s}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$



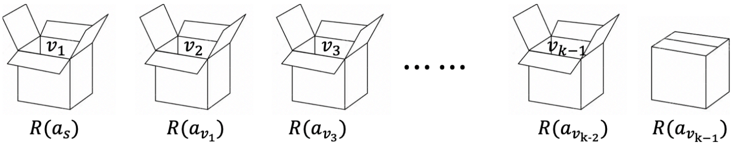
- After opening $k-1$ boxes, the k-th one still has to be random.
- Standard Birthday Paradox.

Analysis for Random Oracle

Birthday Paradox Type Properties [BCM13]

Let $s \in [n]$ be a uniform random starting point. In functional graph $\mathcal{G}_R$,

- $\mathop{\mathbb{E}}_{R,s}[\#\text{vertices reachable from } s] \leq O(\sqrt{n})$



- After opening $k-1$ boxes, the k-th one still has to be random.
- Standard Birthday Paradox.
- Difficulty: $\sqrt{n}$-wise independence.

Our Construction via Iterative Restriction

Our Construction of hash function $h : [m] \to [n]$

Let $\ell = \Theta(\log n)$. Our construction has ℓ independent levels.
For the i-th level, we sample two hash functions r_i, g_i.

Our Construction via Iterative Restriction

Our Construction of hash function $h : [m] \to [n]$

Let $\ell = \Theta(\log n)$. Our construction has ℓ independent levels.
For the i-th level, we sample two hash functions r_i, g_i.

Recall the inputs are $a_1, \ldots, a_n \in [m]$.
$r_i : [m] \to [n]$ and $g_i : [m] \to \{0, 1\}$ are $\Theta(\log n)$-wise independent.

Our Construction via Iterative Restriction

Our Construction of hash function $h : [m] \to [n]$

Let $\ell = \Theta(\log n)$. Our construction has ℓ independent levels.
For the i-th level, we sample two hash functions r_i, g_i.

Recall the inputs are $a_1, \ldots, a_n \in [m]$.
$r_i : [m] \to [n]$ and $g_i : [m] \to \{0, 1\}$ are $\Theta(\log n)$-wise independent.

For any $a \in [m]$, letting $i_a^* = \min\{i \mid g_i(a) = 1\}$,

$$h(a) \triangleq r_{i_a^*}(a)$$

Our Construction via Iterative Restriction

Our Construction of hash function $h : [m] \to [n]$

Let $\ell = \Theta(\log n)$. Our construction has ℓ independent levels.
For the i-th level, we sample two hash functions r_i, g_i.

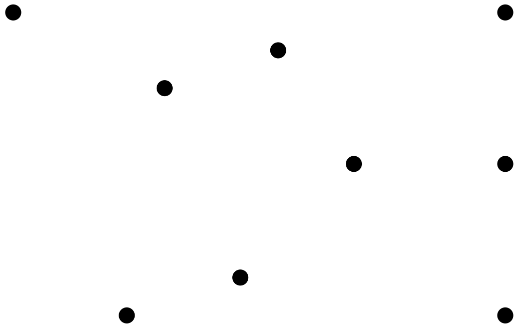
Recall the inputs are $a_1, \ldots, a_n \in [m]$.
$r_i : [m] \to [n]$ and $g_i : [m] \to \{0, 1\}$ are $\Theta(\log n)$-wise independent.

For any $a \in [m]$, letting $i_a^* = \min\{i \mid g_i(a) = 1\}$,

$$h(a) \triangleq r_{i_a^*}(a)$$

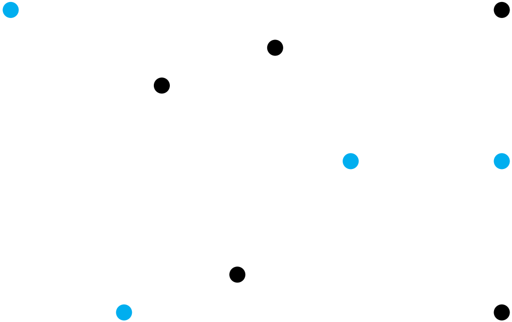
- $g_i(a)$ acts as a "filter".

Our Construction via Iterative Restriction



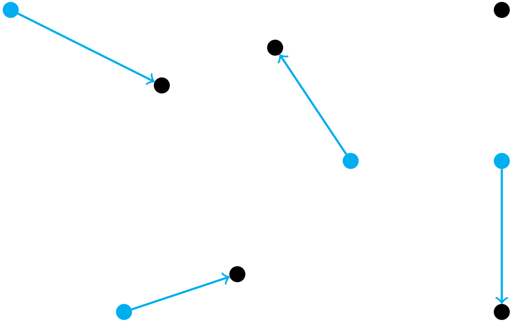
Initially, the functional graph $\mathcal{G}_h$ is empty.

Our Construction via Iterative Restriction

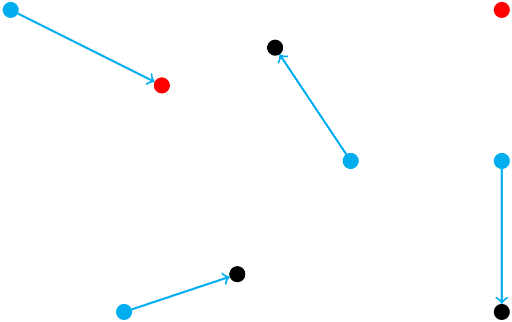


In the 1st level, we select the vertices x with $g_1(a_x) = 1$ ($n/2$ vertices in expectation).

Our Construction via Iterative Restriction

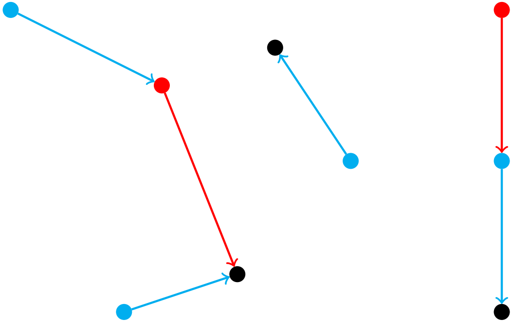


We sample their outgoing edges $x \to r_1(a_x)$ using r_1.



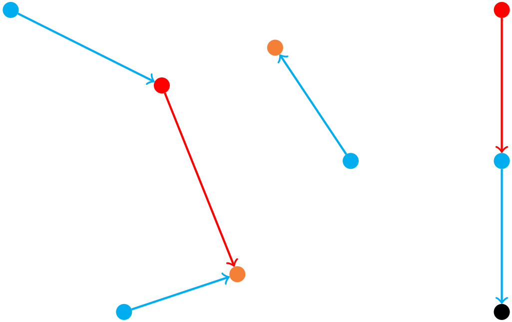
In the 2nd level, we select the remaining vertices x with $g_2(a_x) = 1$ ($n/4$ vertices in expectation).

Our Construction via Iterative Restriction



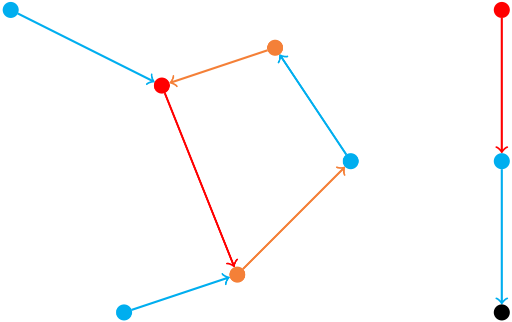
We sample their outgoing edges $x \to r_2(a_x)$ using r_2.

Our Construction via Iterative Restriction



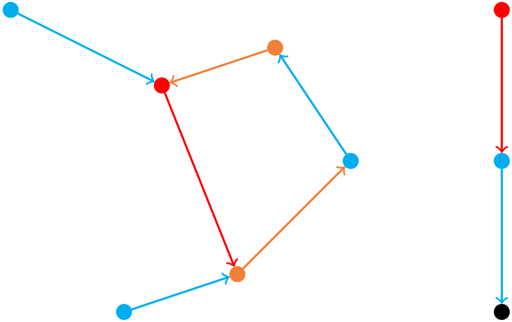
In the 3rd level, we select the remaining vertices x with $g_3(a_x) = 1$ ($n/8$ vertices in expectation).

Our Construction via Iterative Restriction



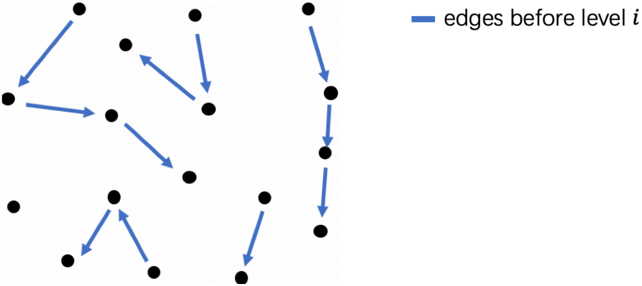
We sample their outgoing edges $x \to r_3(a_x)$ using r_3.

Our Construction via Iterative Restriction



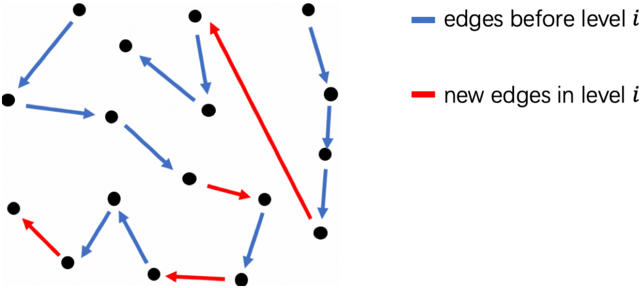
We repeat this for $\ell = \Theta(\log n)$ levels.
Each vertex x got its outgoing edge at level $i^* = \min\{i \mid g_i(a_x) = 1\}$.

Intuition for $\Theta(\log n)$-wise independence



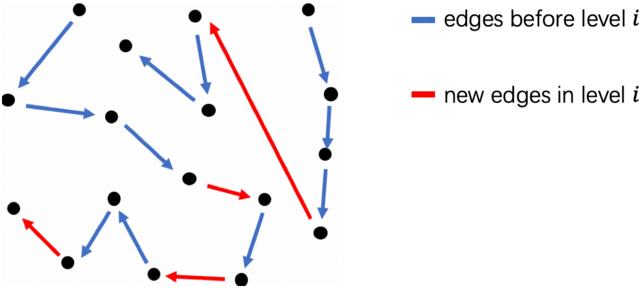
- Remaining vertex are those with no blue outgoing edge.

Intuition for $\Theta(\log n)$-wise independence



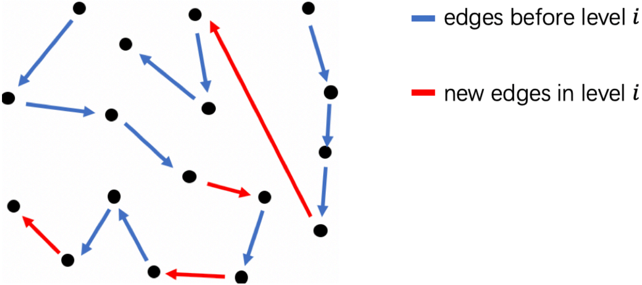
- Remaining vertex are those with no blue outgoing edge.
- Each remaining vertex x has red outgoing edge w.p. $\frac{1}{2}$ $(g_i(a_x) = 1)$.

Intuition for $\Theta(\log n)$-wise independence



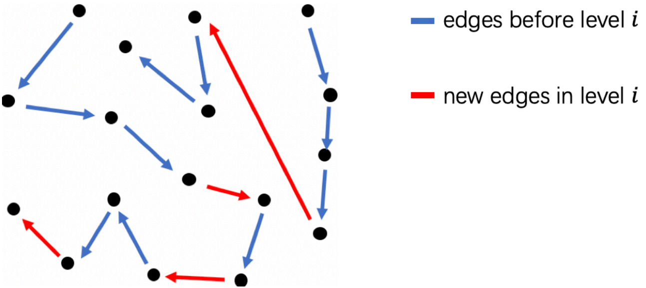
- Remaining vertex are those with no blue outgoing edge.
- Each remaining vertex x has red outgoing edge w.p. $\frac{1}{2}$ $(g_i(a_x) = 1)$.
- W.h.p. a path in this graph contains $O(\log n)$ many red edges.

Intuition for $\Theta(\log n)$-wise independence



- Roughly speaking, this is why we need $\Theta(\log n)$-wise independence per level.

Intuition for $\Theta(\log n)$-wise independence



- Roughly speaking, this is why we need $\Theta(\log n)$-wise independence per level.
- The actual proof is more complicated (40 pages).

Open Problems

Open Problems

- **Time-space Tradeoffs**
  In this work, we only solved the case when $S = O(\text{polylog}\, n)$. Can we extend it to the full tradeoff?
- **Shorter Seed Length**
  In this work, our seed length is $O(\log^3 n \log\log n)$. Can this be improved?
- **Shorter Paper Length**
  Can we obtain a simpler analysis?

Q & A

Questions?

Thank you!

Reference I

Miklos Ajtai and Avi Wigderson.
Deterministic simulation of probabilistic constant depth circuits.
In 26th Annual Symposium on Foundations of Computer Science (sfcs 1985), pages 11–19. IEEE, 1985.

Paul Beame, Raphaël Clifford, and Widad Machmouchi.
Element distinctness, frequency moments, and sliding windows.
In 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, pages 290–299. IEEE, 2013.

Allan Borodin, Faith Fich, F Meyer Auf Der Heide, Eli Upfal, and Avi Wigderson.
A time-space tradeoff for element distinctness.
SIAM Journal on Computing, 16(1):97–99, 1987.

Reference II

Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas.
Faster space-efficient algorithms for subset sum, k-sum, and related problems.
SIAM Journal on Computing, 47(5):1755–1777, 2018.

Andrew Chi-Chih Yao.
Near-optimal time-space tradeoff for element distinctness.
In FOCS, pages 91–97, 1988.