Element Distinctness,
Birthday Paradox,
and
1-out Pseudorandom Graphs

Hongxun Wu
IIIS, Tsinghua University
Authors of this work

Lijie Chen, Ce Jin, and R. Ryan Williams are from MIT.
Element Distinctness
Element Distinctness

- **INPUT:** \(n\) positive integers \(a_1, a_2, \ldots, a_n\) with \(a_i \leq \text{poly}(n)\).
- Decide whether all \(a\)'s are distinct.

\[
\begin{array}{cccccccc}
42 & 3 & 23 & 1 & 12 & 30 & 42 & 15 \\
\end{array}
\]
1 3 12 15 23 30 42 42

Just sort it
You wish...

Just sort it
Comparision model

- No direct access to the INPUT $a$.
- Each query $(i, j)$ returns one of $a_i < a_j$, $a_i = a_j$, $a_i > a_j$. 
Time-Space tradeoff [BFMADH$^+$87, Yao88]

Element distinctness requires $TS = \Omega(n^{2-o(1)})$ in Comparision model.
Time-Space tradeoff [BFMADH$^+87$, Yao88]

Element distinctness requires $TS = \Omega \left( n^{2-o(1)} \right)$ in Comparison model.

- When $S = O(\text{polylog } n)$, $T = \Omega \left( n^{2-o(1)} \right)$. 
**RAM model**

- Random access to read-only input.
- Working memory has a (relatively small) size $S$. 

![Image of memory elements with values 42, 3, 23, 1, 12, 30, 42, 15]
Time-Space tradeoff [BCM13]

- Assuming the existence of Random Oracle, there is an algorithm with $T^2S = \tilde{O}(n^3)$. 
Time-Space tradeoff [BCM13]

- Assuming the existence of Random Oracle, there is an algorithm with $T^2S = \tilde{O}(n^3)$.

- When $S = \tilde{O}(1)$, $T = \tilde{O}(n^{1.5})$. 
RAM model

Time-Space tradeoff [BCM13]

- Assuming the existence of Random Oracle, there is an algorithm with 
  \( T^2 S = \tilde{O}(n^3) \).

- When \( S = \tilde{O}(1) \), \( T = \tilde{O}(n^{1.5}) \).
- In the rest of this talk, we always assume there is only one collision \((a_i = a_j)\).
Pollard’s $\rho$ method [BCM13]

Assuming the existence of Random Oracle, when $S = \tilde{O}(1)$, there is an algorithm with $T = \tilde{O}(n^{1.5})$.

- For random oracle $R$, define graph $x \mapsto R(a_x)$ with $x \in [n]$. 

\[ 
\begin{array}{c}
\text{0} \\
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{5} \\
\text{6} \\
\text{7} \\
\text{8}
\end{array}
\]
Pollard’s \( \rho \) method [BCM13]

Assuming the existence of Random Oracle, when \( S = \tilde{O}(1) \), there is an algorithm with \( T = \tilde{O}(n^{1.5}) \).

- For random oracle \( R \), define graph \( x \mapsto R(a_x) \) with \( x \in [n] \).
- Pick a random starting point \( s \).
Pollard’s $\rho$ method [BCM13]

Assuming the existence of Random Oracle, when $S = \tilde{O}(1)$, there is an algorithm with $T = \tilde{O}(n^{1.5})$.

- For random oracle $R$, define graph $x \mapsto R(a_x)$ with $x \in [n]$.
- Pick a random starting point $s$.
- Run Floyd’s cycle finding.
Birthday Paradox Type Properties [BCM13]

Suppose $f^*(s)$ is the set of vertices reachable from $s$.

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$

- So each cycle-finding takes $O(\sqrt{n})$ time and finds any collision $u, v$ with probability $\Omega(1/n)$.

- Repeat $O(n)$ times. It takes $O(n^{1.5})$ time in total.
Birthday Paradox Type Properties [BCM13]

Suppose $f^*(s)$ is the set of vertices reachable from $s$.

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \geq \Omega(1/n)$, $\forall u, v \in [n]$

*collision*
Birthday Paradox Type Properties [BCM13]

Suppose $f^*(s)$ is the set of vertices reachable from $s$.

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \geq \Omega(1/n), \forall u, v \in [n]$

So each cycle-finding takes $O(\sqrt{n})$ time and finds any collision $u, v$ with probability $\Omega(1/n)$. 
Birthday Paradox Type Properties [BCM13]

Suppose $f^*(s)$ is the set of vertices reachable from $s$.

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \geq \Omega(1/n), \forall u, v \in [n]$

- So each cycle-finding takes $O(\sqrt{n})$ time and finds any collision $u, v$ with probability $\Omega(1/n)$.
- Repeat $O(n)$ times. It takes $O(n^{1.5})$ time in total.
Our Results

Our Main Lemma

There exists a family \( \{ r_{\text{seed}} \} \) of hash functions efficiently samplable with seed length \( O(\text{polylog } n) \), and the graph defined by \( \{ r_{\text{seed}} \} \) (instead of Random Oracle \( R \)) satisfy

- \( \mathbb{E}[|f^*(s)|] \leq O(\sqrt{n}) \)

Our Result

Assuming the existence of Random Oracle, when \( S = O(\text{polylog } n) \), there is a RAM algorithm for Element Distinctness with \( T = \tilde{O}(n^{1.5}) \).
Our Main Lemma

There exists a family \( \{ r_{seed} \} \) of hash functions efficiently samplable with seed length \( O(\text{polylog } n) \), and the graph defined by \( \{ r_{seed} \} \) (instead of Random Oracle \( R \)) satisfy

- \( \mathbb{E}[|f^*(s)|] \leq O(\sqrt{n}) \)
- \( \Pr[u, v \in f^*(s)] \geq \Omega(1/n), \forall u, v \in [n] \)
Our Results

Our Main Lemma

There exists a family \( \{ r_{\text{seed}} \} \) of hash functions efficiently samplable with seed length \( O(\text{polylog } n) \), and the graph defined by \( \{ r_{\text{seed}} \} \) (instead of Random Oracle \( R \)) satisfy

- \( \mathbb{E}[|f^*(s)|] \leq O(\sqrt{n}) \)
- \( \Pr[u,v \in f^*(s)] \geq \Omega(1/n), \forall u,v \in [n] \)

Our Result

Assuming the existence of Random Oracle, when \( S = O(\text{polylog } n) \), there is a RAM algorithm for Element Distinctness with \( T = \tilde{O}(n^{1.5}) \).
Low-space Algorithm for Subset Sum [BGNV18]

Assuming the existence of Random Oracle, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $O^*(2^{0.86n})$ time, with $O(poly(n))$ space.
Low-space Algorithm for Subset Sum [BGNV18]

Assuming the existence of Random Oracle, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $O^*(2^{0.86n})$ time, with $O(poly(n))$ space.

Our Result

Assuming the existence of Random Oracle, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $O^*(2^{0.86n})$ time, with $O(poly(n))$ space.
Our Results

Our Main Lemma

There exists a family \( \{ r_{\text{seed}} \} \) of hash functions efficiently samplable with seed length \( O(\text{polylog } n) \), and the graph defined by \( \{ r_{\text{seed}} \} \) (instead of Random Oracle \( R \)) satisfy

- \( \mathbb{E}[|f^*(s)|] \leq O(\sqrt{n}) \)
- \( \Pr[u, v \in f^*(s)] \geq \Omega(1/n), \forall u, v \in [n] \)
Our Results

Our Main Lemma

There exists a family \( \{ r_{\text{seed}} \} \) of hash functions efficiently samplable with seed length \( O(\text{polylog } n) \), and the graph defined by \( \{ r_{\text{seed}} \} \) (instead of Random Oracle \( R \)) satisfy

- \( \mathbb{E}[|f^*(s)|] \leq O(\sqrt{n}) \)
- \( \Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n] \)
Construction
This is Ryan O’Donnell’s Youtube lecture which is a masterpiece.
This is the Ajtai-Wigderson Paradigm [AW85] for building PRG.
This is the Ajtai-Wigderson Paradigm [AW85] for building PRG.
This is the Ajtai-Wigderson Paradigm [AW85] for building PRG.
Recall the input \( a_1, a_2, \ldots, a_n \in [m] \).

Two Level Example

Suppose we have the following:

- \( O(\text{polylog } n) \)-wise independent functions \( g : [m] \rightarrow \{0, 1\} \) and \( r : [m] \rightarrow [n] \).

- Random Oracle \( R \).
Recall the input $a_1, a_2, \ldots, a_n \in [m]$.

Two Level Example

Suppose we have the following:

- $O(\text{polylog } n)$-wise independent functions $g : [m] \rightarrow \{0, 1\}$ and $r : [m] \rightarrow [n]$.
- Random Oracle $R$.

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$
We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} \text{R}(a_x) & g(a_x) = 0 \\ \text{r}(a_x) & g(a_x) = 1 \end{cases}$$

level 2
$g(a_x) = 0$
level 1
$g(a_x) = 1$

$s$
Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

- **level 1**
  - $g(a_x) = 0$
  - $g(a_x) = 1$

- **level 2**
  - $s$
We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

level 2
$g(a_x) = 0$

level 1
$g(a_x) = 1$

$s$
Toy Example: Two levels

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

**Two Level Example**

level 1

\[
\begin{align*}
g(a_x) &= 0 \\
g(a_x) &= 1
\end{align*}
\]

level 2

\[
\begin{align*}
g(a_x) &= 0 \\
g(a_x) &= 1
\end{align*}
\]

$s$
We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

- **level 1**
  - $g(a_x) = 0$
  - $g(a_x) = 1$

- **level 2**
Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} 
R(a_x) & g(a_x) = 0 \\
r(a_x) & g(a_x) = 1 
\end{cases}$$

level 1

$g(a_x) = 0$

$g(a_x) = 1$

level 2

$s$
We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

level 2
$g(a_x) = 0$

level 1
$g(a_x) = 1$
Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} 
R(a_x) & g(a_x) = 0 \\
r(a_x) & g(a_x) = 1 
\end{cases}$$

level 2
$g(a_x) = 0$

level 1
$g(a_x) = 1$

$s$
Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

level 2
$g(a_x) = 0$

level 1
$g(a_x) = 1$

$s$
Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

level 2
$g(a_x) = 0$

level 1
$g(a_x) = 1$

s

w.h.p. $O(\log n)$ edges
• Why this might be a good idea?
Sanity Check

• Why this might be a good idea?

• Each subpath has length $O(\log n)$. 

Sanity Check
Sanity Check

- Why this might be a good idea?

- Each subpath has length $O(\log n)$.
- Every level 2 edge is an independent sample of a subpath.
Recall our goal.

Our Main Lemma

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n]$
• “Memory” of a random walk: The current vertex it is at.
“Memory” of a random walk: The current vertex it is at.
• “Memory” of a random walk: The current vertex it is at.
• “Memory” of a random walk: The current vertex it is at.
Intuition: Memory Eraser

- "Memory" of a random walk: The current vertex it is at.

$h(h(h(s)))$

I know 3 of them
• “Memory” of a random walk: The current vertex it is at.
• “Memory” of a random walk: The current vertex it is at.
• “Memory” of a random walk: The current vertex it is at.
- “Memory” of a random walk: The current vertex it is at.

\[ h(h(s)) \]
• “Memory” of a random walk: The current vertex it is at.
• “Memory” of a random walk: The current vertex it is at.
Now we sample $O(\log n)$ many hash functions $\{r_i, g_i\}_{i \in [\ell]}$. Each $r_i : [m] \rightarrow [n]$ and $g_i : [m] \rightarrow [2]$ are $O(\log n)$-wise independent.
Our Construction via Iterative Restriction

Our Construction

Now we sample $O(\log n)$ many hash functions $\{r_i, g_i\}_{i \in [\ell]}$. Each $r_i : [m] \to [n]$ and $g_i : [m] \to [2]$ are $O(\log n)$-wise independent. Then we set $h_{\ell+1}(a_x) = \perp$ and

$$h_i(a_x) = \begin{cases} 
    h_{i+1}(a_x) & g_i(a_x) = 0 \\
    r_i(a_x) & g_i(a_x) = 1 
\end{cases}$$

Finally, we set $h = h_1$. 
Our Construction via Iterative Restriction

The Random Walk

$g_1(a_x) = 1$

$s$

The Random Walk

$g_1(a_x) = 1$

$s$

The Random Walk

$g_1(a_x) = 1$

$s$

The Random Walk

$g_1(a_x) = 1$

$s$

The Random Walk

$g_1(a_x) = 1$

$s$
Our Construction via Iterative Restriction

level 5

level 4

level 3

level 2

level 1

g_1(a_x) = 0

\( s \)

The Random Walk
Our Construction via Iterative Restriction

The Random Walk

$g_1(a_x) = 0$
$g_2(a_x) = 0$

level 1
level 2
level 3
level 4
level 5
Our Construction via Iterative Restriction

level 5

level 4

level 3
\( g_3(a_x) = 0 \)

level 2
\( g_2(a_x) = 0 \)

level 1
\( g_1(a_x) = 0 \)

The Random Walk
Our Construction via Iterative Restriction

<table>
<thead>
<tr>
<th>Level</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$g_4(a_x) = 1$</td>
</tr>
<tr>
<td>4</td>
<td>$g_3(a_x) = 0$</td>
</tr>
<tr>
<td>3</td>
<td>$g_2(a_x) = 0$</td>
</tr>
<tr>
<td>2</td>
<td>$g_1(a_x) = 0$</td>
</tr>
<tr>
<td>1</td>
<td>$s$</td>
</tr>
</tbody>
</table>

The Random Walk
Our Construction via Iterative Restriction

level 5

level 4

level 3

level 2

level 1

$g_2(a_x) = 1$

$g_1(a_x) = 0$

$s$

The Random Walk
Our Construction via Iterative Restriction

level 5

level 4

level 3
\(g_3(a_x) = 1\)

level 2
\(g_2(a_x) = 0\)

level 1
\(g_1(a_x) = 0\)

The Random Walk
Our Construction via Iterative Restriction

The Random Walk

The Random Walk

$g_1(a_x) = 1$

$s$

level 1

level 2

level 3

level 4

level 5
Our Construction via Iterative Restriction

The Random Walk

The Random Walk

level 5

level 4

level 3

level 2

$g_2(a_x) = 1$

level 1

$g_1(a_x) = 0$

s

The Random Walk
Our Construction via Iterative Restriction

The Random Walk

The Random Walk
Our Construction via Iterative Restriction

level 1

level 2
\[ g_2(a_x) = 1 \]

level 3

level 4

level 5

\[ g_1(a_x) = 0 \]

The Random Walk
Our Construction via Iterative Restriction

level 5
level 4
\( g_4(a_x) = 1 \)
level 3
\( g_3(a_x) = 0 \)
level 2
\( g_2(a_x) = 0 \)
level 1
\( g_1(a_x) = 0 \)

The Random Walk

The Random Walk
Our Construction via Iterative Restriction

The Random Walk

The Random Walk

The Random Walk
Our Construction via Iterative Restriction

The Random Walk

level 1

level 2

level 3

level 4

level 5

\[ g_1(a_x) = 0 \]

\[ g_2(a_x) = 0 \]

\[ g_3(a_x) = 1 \]

\( s \)
Our Construction via Iterative Restriction

The Random Walk

The Random Walk

level 5

\[ g_4(a_x) = 0 \]

\[ g_3(a_x) = 0 \]

\[ g_2(a_x) = 0 \]

\[ g_1(a_x) = 0 \]
Key Ideas in Our Analysis
The Random Walk
The Random Walk

Hongxun Wu (IIIS, Tsinghua)
Dependency Tree

The Random Walk
The Random Walk

Hongxun Wu (IIIS, Tsinghua)
We index a node by the shape of its path, e.g. $\vec{k}_{10} = (0, 0, 1, 2)$. Consider $\vec{k}_x$. Fix $x$, $\vec{k}$ is a random variable. Fix $\vec{k}$, $x$ is a random variable. We fix index $\vec{k}$ and let $x$ be the random variable (which may not exist).
We index a node by the shape of its path, e.g. $\vec{k}_{10} = (0, 0, 1, 2)$. 
- We index a node by the shape of its path, e.g. $\vec{k}_{11} = (0, 0, 2, 2)$. 
- We index a node by the shape of its path, e.g. \( \vec{k}_{11} = (0, 0, 2, 2) \).
- Consider \( \vec{k}_x \). Fix \( x \), \( \vec{k} \) is a random variable. Fix \( \vec{k} \), \( x \) is a random variable.
- We index a node by the shape of its path, e.g. \( \vec{k}_{11} = (0, 0, 2, 2) \).
- Consider \( \vec{k}_x \). Fix \( x, \vec{k} \) is a random variable. Fix \( \vec{k}, x \) is a random variable.
- We fix index \( \vec{k} \) and let \( x \) be the random variable (which may not exist).
Memory Eraser on Dependency Tree

- Fix $\vec{k} = (0, 0, 2, 2)$.
• Fix $\vec{k} = (0, 0, 2, 2)$. 
Fix $\vec{k} = (0, 0, 2, 2)$. 

- Fix $k = (0, 0, 2, 2)$. 

---

**Memory Eraser on Dependency Tree**

- **Level 5** ($\ell = 4$)
  - Node 0
  - Node 2
    - Node 4
      - Node 3
      - Node 5
    - Node 6
    - Node 7
  - Node 9
    - Node 10
    - Node 11
    - Node 12
- Fix $\vec{k} = (0, 0, 2, 2)$.
- Blue part is a random variable. But it will finally end up with a node with level $\geq 4$. 
Memory Eraser on Dependency Tree

- Fix $\vec{k} = (0, 0, 2, 2)$.
- Blue part is a random variable. But it will finally end up with a node with level $\geq 4$. 
Fix $\vec{k} = (0, 0, 2, 2)$.

Blue part is a random variable. But it will finally end up with a node with level $\geq 4$. 
Fix $\vec{k} = (0, 0, 2, 2)$.

Blue part is a random variable. But it will finally end up with a node with level $\geq 4$. 

For another issue: What if $a_w^2 = a_w^9$?
Fix $k = (0, 0, 2, 2)$.

Blue part is a random variable. But it will finally end up with a node with level $\geq 4$.

One issue: What if $a_{w_2} = a_{w_9}$?
• Instead of original walk $w$, we look at extended walk $w^*$. 

(Locally Simulatable) Extended Walk
Instead of original walk \( w \), we look at extended walk \( w^* \).

Once a position in our memory is queried twice, we replace it with true randomness.
Instead of original walk \( w \), we look at extended walk \( w^* \).

Once a position in our memory is queried twice, we replace it with true randomness.

\( w^* \) is \textit{locally simulatable} in the sense that each query position can be uniquely determined by memory.
Instead of original walk $w$, we look at extended walk $w^\ast$.

Once a position in our memory is queried twice, we replace it with true randomness.

$w^\ast$ is *locally simulatable* in the sense that each query position can be uniquely determined by memory.

$w$ and $w^\ast$ agree if $w^\ast$ has no collision $a_{w_i^\ast} = a_{w_j^\ast}$.
Good = All - Bad

Recall our goal.

Our Main Lemma

- \( \mathbb{E}[|f^*(s)|] \leq O(\sqrt{n}) \)
- \( \Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n] \)

- \( w \) and \( w^* \) agree if \( w^* \) has no collision \( a_{w_i^*} = a_{w_j^*} \).
Recall our goal.

Our Main Lemma

- \( E[|f^*(s)|] \leq O(\sqrt{n}) \)
- \( \Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n] \)

- \( w \) and \( w^* \) agree if \( w^* \) has no collision \( a_{w_i^*} = a_{w_j^*} \).
- Good: \( E[\#\{ t \mid w_t^* = u, \ w^* \ has \ no \ collision \}] \geq \Pr[\exists t, w_t = u] \).
Recall our goal.

**Our Main Lemma**

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \forall u \in [n]$

- $w$ and $w^*$ agree if $w^*$ has no collision $a_{w_i^*} = a_{w_j^*}$.
- Good: $E[\#\{t|w_t^* = u, w^* \text{ has no collision}\}] \geq \Pr[\exists t, w_t = u]$.
- All: $E[\#\{t|w_t^* = u\}]$
Recall our goal.

**Our Main Lemma**

- \( \mathbb{E}[|f^*(s)|] \leq O(\sqrt{n}) \)
- \( \Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n] \)

- \( w \) and \( w^* \) agree if \( w^* \) has no collision \( a_{w_i^*} = a_{w_j^*} \).
- Good: \( \mathbb{E}[\#\{t|w_t^* = u, w^* \text{ has no collision}\}] \geq \Pr[\exists t, w_t = u] \).
- All: \( \mathbb{E}[\#\{t|w_t^* = u\}] \)
- Bad:

\[
\mathbb{E}[\#\{t|w_t^* = u, \exists t' \neq t'', a_{w_t'} = a_{w_{t''}}\}] \\
\leq \mathbb{E}[\#\{t, t' \neq t''|w_t^* = u, a_{w_t^*} = a_{w_{t''}}\}]
\]
Good = All - Bad

$E[\#\{ t| w_t^* = u\}] = \sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_{\ell})}}{n}$
Good = All - Bad

$$E[\#\{t, t' \neq t''|w_t^* = u, a_{w_t^*} = a_{w_{t''}}\}] = \sum_{\vec{k}, \vec{k}', \vec{k}''} \frac{2^{-\|\vec{k}\|_1 - \|\vec{k}'\|_1 - \|\vec{k}''\|_1}}{n^2}$$
Good = All - Bad

\[
\text{Good} = \sum_{\vec{k}} 2^{-\left(k_1 + k_2 + \cdots + k_{\ell}\right)} n - \sum_{\vec{k}, \vec{k}', \vec{k}''} 2^{-\|\vec{k}\|_1 - \|\vec{k}'\|_1 - \|\vec{k}''\|_1} n^2
\]
Good = All - Bad

\[
\text{Good} = \sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n} - \sum_{\vec{k},\vec{k}',\vec{k}''} \frac{2^{-\|\vec{k}\|_1 - \|\vec{k}'\|_1 - \|\vec{k}''\|_1}}{n^2}
\]

\[
\sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n} = \frac{1}{n} \prod_{i=1}^{\ell} \sum_{k_i=0}^{\infty} 2^{-k_i} = \frac{2^\ell}{n}
\]
Good = All - Bad

\[
\begin{align*}
\text{Good} &= \sum_{\vec{k}} \frac{2^{-(k_1 + k_2 + \cdots + k_{\ell})}}{n} - \sum_{\vec{k}, \vec{k}', \vec{k}''} \frac{2^{-\|\vec{k}\|_1 - \|\vec{k}'\|_1 - \|\vec{k}''\|_1}}{n^2} \\
&= \sum_{\vec{k}, \vec{k}', \vec{k}''} \frac{2^{-\|\vec{k}\|_1 - \|\vec{k}'\|_1 - \|\vec{k}''\|_1}}{n^2} = \frac{8^\ell}{n^2}
\end{align*}
\]
Good = All - Bad

\[
\text{Good} = \sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n} - \sum_{\vec{k},\vec{k}',\vec{k}''} \frac{2^{-\|\vec{k}\|_1 - \|\vec{k}'\|_1 - \|\vec{k}''\|_1}}{n^2} = \frac{2^\ell}{n} - \frac{8^\ell}{n^2}
\]

Let \( \ell \leftarrow \frac{1}{2} \log n - 100. \) \( \frac{2^\ell}{n} - \frac{8^\ell}{n^2} = \frac{2^{-100}}{\sqrt{n}} - \frac{2^{-300}}{\sqrt{n}} = \Omega \left( \frac{1}{\sqrt{n}} \right). \)
Our Main Lemma

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \forall u \in [n]$

- Even for this simple case, there is so much more technical challenges that is hidden in this talk.
Open Problems
Open Problems

- **Time-space Tradeoffs**
  In this work, we only solved the case when $S = \tilde{O}(1)$. 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?

