COMPSCI 614: Randomized Algorithms with Applications to Data Science

Prof. Cameron Musco

University of Massachusetts Amherst. Spring 2024.
Lecture 7

Logistics

- Problem Set 2 is posted and due next Wednesday.
- One page project proposal due Tuesday 3/12.
- If you have emailed me about project ideas and I haven't replied I will shortly.

Summary

0 1 0 0 1

$X_1 \ldots X_n$ $\quad h(x) = \sum x_i 2^i \bmod p$

$O(tn)$

collision prob $1/t$

Last Time:

- Random hashing and the Rabin fingerprint.
- Applications to low communication protocol for equality testing (testing equality of n-bit strings using $O(\log n)$ bits), and to pattern matching (Rabin-Karp algorithm).

$O(n+m)$

n bit string

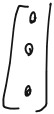
m bit pattern

Summary

Last Time:

- Random hashing and the Rabin fingerprint.
- Applications to low communication protocol for equality testing (testing equality of n-bit strings using $O(\log n)$ bits), and to pattern matching (Rabin-Karp algorithm).

Today:



- Sparse recovery/ℓ_0 sampling via linear sketching.
- Application to a low-communication protocol for graph connectivity.

Quiz Review

Question 1

Not complete

Points out of 1.00

Flag question

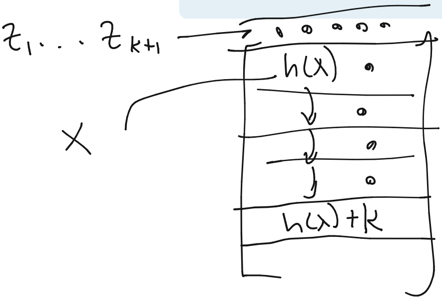
Edit question

Consider a hash table that uses linear probing to resolve collisions. Assume that item x is stored in position $h(x) + k$. True or False: The interval $[h(x), h(x)+1, \ldots, h(x)+k]$ is always a length-(k+1) full interval.

(at least k+1 items hash to this interval

- True
- False

Check



Quiz Review

Question 2

Not complete

Points out of 1.00

Flag question

Edit question

Alice and Bob both have n-bit strings, $a, b \in \{0,1\}^n$. For any $\delta > 0$, how many bits of communication do they need to determine, with probability at least $1 - \delta$ whether or not $a = b$?

- a. $O(\log(n) \cdot \log(1/\delta))$
- b. $O(\log(n/\delta))$
- c. $O\left(\frac{\log(n)}{\delta}\right)$
- d. $O(1/\delta)$
- e. $O(\log(n))$

Check

— apply rabin fingerprint

$p = O(n/\delta) = O(nt)$

$t = 1/\delta$

$Pr(h(a) = h(b)) = \frac{1}{t} = \delta$

assume $a \neq b$

you need to send p, $h(a)$

$O(\log p)$ bits $O(\log(n/\delta))$

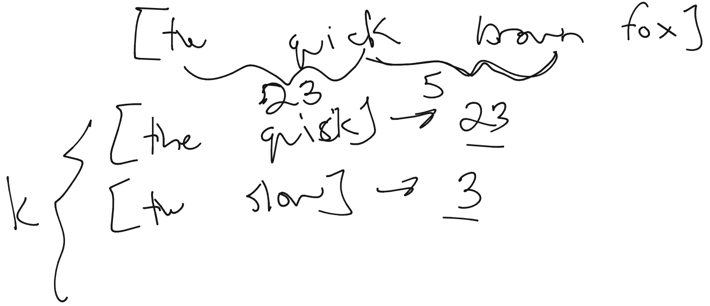
$= O(\log n + \log(1/\delta))$

What is also achieved a:

- set $t = O(1)$ so we succeed w.p 2/3
- repeat protocol $\log(1/\delta)$ times
- $O(\log n) \cdot \log(1/\delta)$

The Rabin-Karp algorithm can be extended to search for k patterns in just $O(n + km)$ expected time.

- Significantly better than the naive $O((n + m)k)$ that would follow from repeating single pattern matching k times.



Rabin-Karp for Multiple Pattern Matching

The Rabin-Karp algorithm can be extended to search for k patterns in just $O(n + km)$ expected time.

- Significantly better than the naive $O((n + m)k)$ that would follow from repeating single pattern matching k times.
- **Key Idea:** Compute fingerprints for all k patterns in $O(mk)$ time and store them in a hash table.
- Compute the fingerprints of $X_1, X_2, \ldots, X_{n-m+1}$ iteratively in $O(n)$ time via the rolling hash trick.
- At each iteration, check X_j against all patterns by doing a hash table look-up in $O(1)$ expected time.

Other Topics in Hashing

There are a ton of interesting topics related to random hashing that I am not covering.

map to hash table w/ n buckets collision prob btw 2 items is ≤1/n

- Constructions of universal hash functions.
- Constructions of k-wise independent hash functions.
- Concentration bounds and hash table analysis using k-wise independent hash functions. See Lectures 3-4 of Jelani Nelson's course notes for some material on this (link on schedule page).
- Connections to pseudorandom number generators (PRGs).

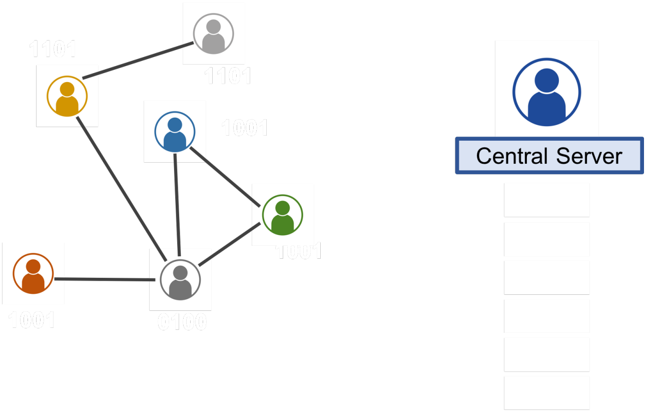
ℓ_0 Sampling and Graph Sketching

A Graph Communication Problem

Consider n nodes, each only knows its own neighborhood. They want to send messages to a central server, who will then determine if the graph is connected.

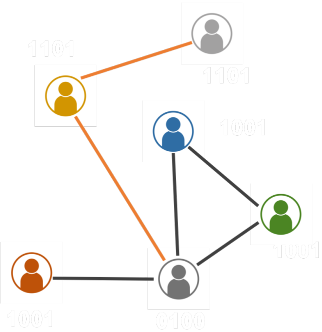
A Graph Communication Problem

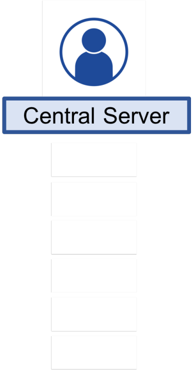
Consider n nodes, each only knows its own neighborhood. They want to send messages to a central server, who will then determine if the graph is connected.



A Graph Communication Problem

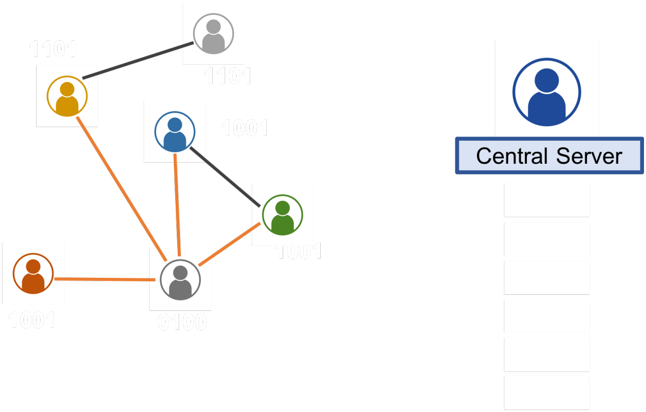
Consider n nodes, each only knows its own neighborhood. They want to send messages to a central server, who will then determine if the graph is connected.





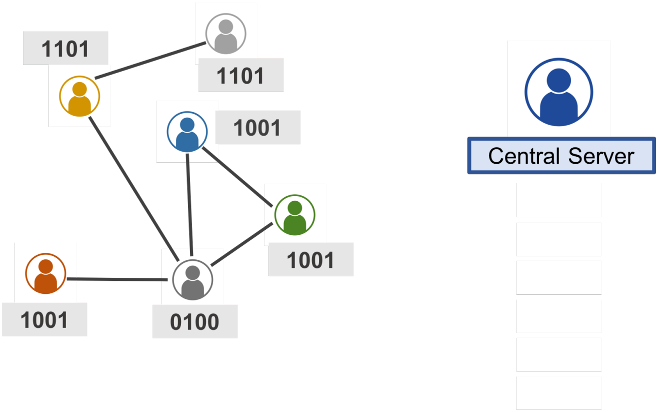
A Graph Communication Problem

Consider n nodes, each only knows its own neighborhood. They want to send messages to a central server, who will then determine if the graph is connected.



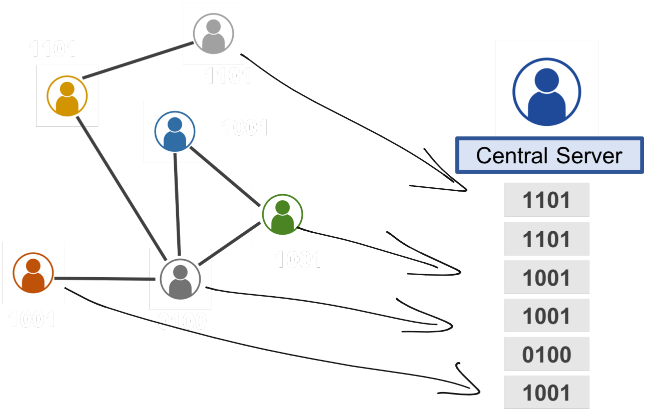
A Graph Communication Problem

Consider n nodes, each only knows its own neighborhood. They want to send messages to a central server, who will then determine if the graph is connected.



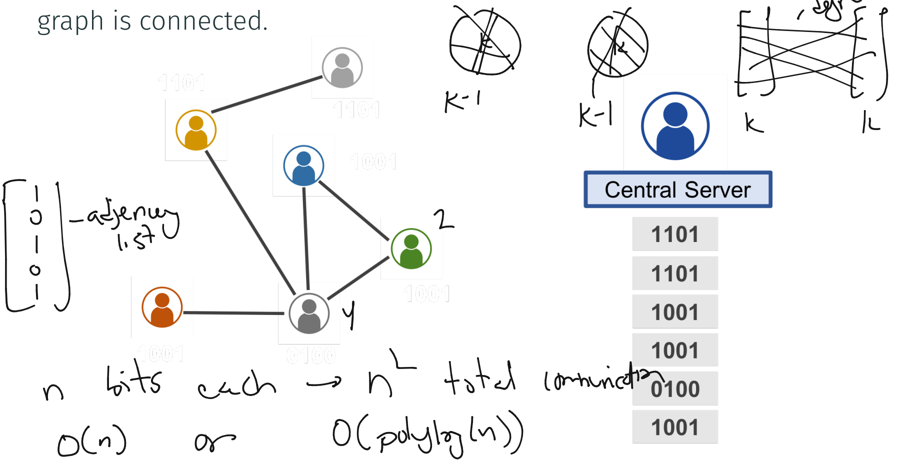
A Graph Communication Problem

Consider n nodes, each only knows its own neighborhood. They want to send messages to a central server, who will then determine if the graph is connected.



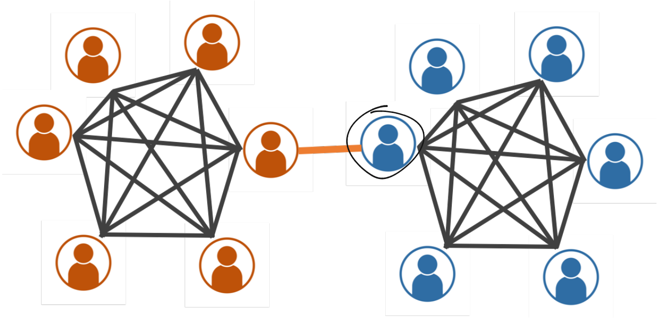
A Graph Communication Problem

Consider n nodes, each only knows its own neighborhood. They want to send messages to a central server, who will then determine if the graph is connected.

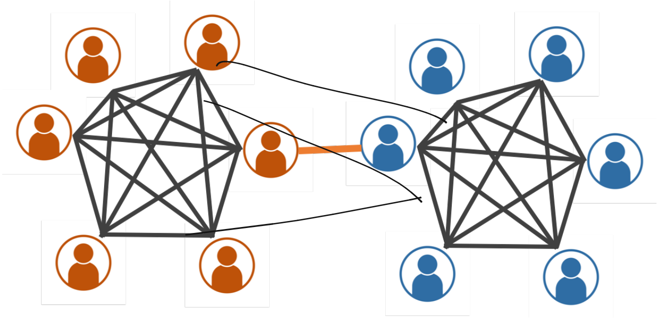


How large of messages (# bits) are needed to determine connectivity with high probability?

A Hard Case



A Hard Case

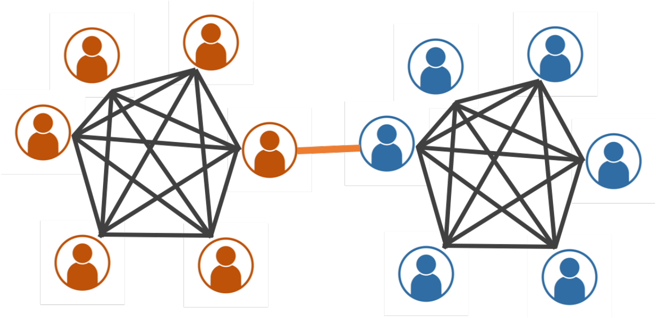


$1 - \frac{1}{n^c}$

- Surprisingly, for any input graph, the problem can be solved with high probability using just $O(\log^c n)$ bits per message!

$O(\log^3 n)$

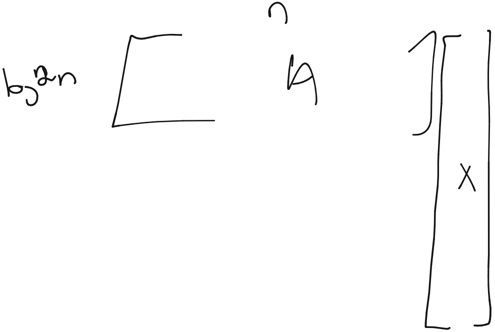
A Hard Case



- Surprisingly, for any input graph, the problem can be solved with high probability using just $O(\log^c n)$ bits per message!
- Solution will be based on a random linear sketch.

Key Ingredient 1: ℓ_0 Sampling

Theorem: There exists a distribution over random matrices $A \in \mathbb{Z}^{O(\log^2 n) \times n}$ such that for any fixed $x \in \mathbb{Z}^n$, with probability at least $1 - 1/n^c$, we can learn (i, x_i) for some $x_i \neq 0$ from Ax.



Key Ingredient 1: ℓ_0 Sampling

Theorem: There exists a distribution over random matrices $\mathbf{A} \in \mathbb{Z}^{O(\log^2 n)\times n}$ such that for any fixed $x \in \mathbb{Z}^n$, with probability at least $1 - 1/n^c$, we can learn (i, x_i) for some $x_i \neq 0$ from $\mathbf{A}x$.

Random sketching matrix A

1	-1	0	0	1	-1	0	1
-1	0	1	1	0	0	-1	0
1	1	-1	0	-1	-1	0	1
0	-1	-1	-1	1	1	1	0

x

x
1
0
0
-2
0
0
3
0

=

Ax

Ax
1
-2
1
5

$\log^2 n$

output (4, -2)

sparse recovery sketch

compressed sensing

Key Ingredient 1: ℓ_0 Sampling

Theorem: There exists a distribution over random matrices $\mathbf{A} \in \mathbb{Z}^{O(\log^2 n)\times n}$ such that for any fixed $x \in \mathbb{Z}^n$, with probability at least $1 - 1/n^c$, we can learn (i, x_i) for some $x_i \neq 0$ from $\mathbf{A}x$.

Random sketching matrix **A** · **x** = **Ax**

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 & -1 & 0 & 1 \\ -1 & 0 & 1 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 1 \\ 0 & -1 & -1 & -1 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ -2 \\ 0 \\ 0 \\ 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \\ 5 \end{bmatrix}$$

Useful Property 1: Given t vectors $x_1, \ldots, x_t \in \mathbb{Z}^n$, can recover a nonzero entry from each with probability $\geq 1 - t/n^c$.

Key Ingredient 1: ℓ_0 Sampling

Theorem: There exists a distribution over random matrices $\mathbf{A} \in \mathbb{Z}^{O(\log^2 n) \times n}$ such that for any fixed $x \in \mathbb{Z}^n$, with probability at least $1 - 1/n^c$, we can learn (i, x_i) for some $x_i \neq 0$ from $\mathbf{A}x$.

Random sketching matrix A

1	-1	0	0	1	-1	0	1
-1	0	1	1	0	0	-1	0
1	1	-1	0	-1	-1	0	1
0	-1	-1	-1	1	1	1	0

x
1
0
0
-2
0
0
3
0

=

Ax
1
-2
1
5

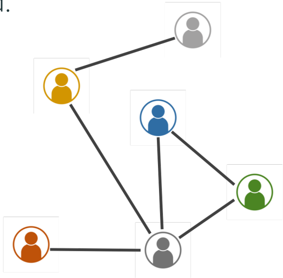
Useful Property 1: Given t vectors $x_1, \ldots, x_t \in \mathbb{Z}^n$, can recover a nonzero entry from each with probability $\geq 1 - t/n^c$.

Useful Property 2: Given sketches $\mathbf{A}x_1$ and $\mathbf{A}x_2$, can easily compute $\mathbf{A}(x_1 + x_2)$ and recover a nonzero entry from $x_1 + x_2$ with high probability.

$= Ax_1 + Ax_2$

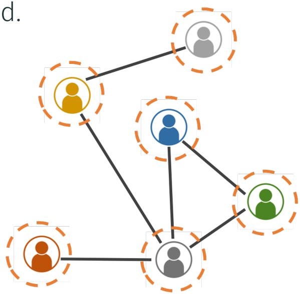
Key Ingredient 2: Boruvka's Algorithm

1. Initialize each node as its own connected component.
2. For each connected component, select an outgoing edge. Merge any newly connected components.
3. Repeat until no connected component has an outgoing edge. If at this point, all nodes are in the same component, then the graph is connected.



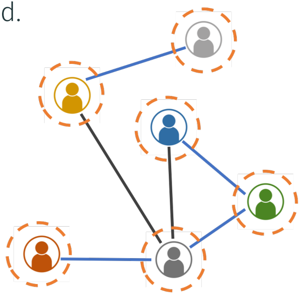
Key Ingredient 2: Boruvka's Algorithm

1. Initialize each node as its own connected component.
2. For each connected component, select an outgoing edge. Merge any newly connected components.
3. Repeat until no connected component has an outgoing edge. If at this point, all nodes are in the same component, then the graph is connected.



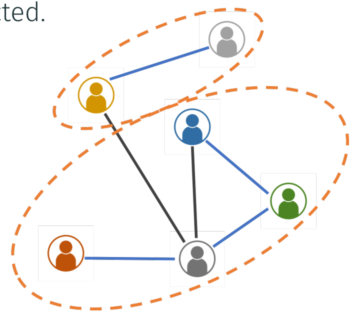
Key Ingredient 2: Boruvka's Algorithm

1. Initialize each node as its own connected component.
2. For each connected component, select an outgoing edge. Merge any newly connected components.
3. Repeat until no connected component has an outgoing edge. If at this point, all nodes are in the same component, then the graph is connected.



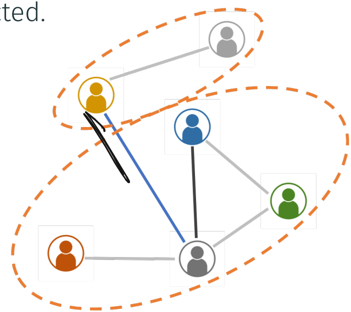
Key Ingredient 2: Boruvka's Algorithm

1. Initialize each node as its own connected component.
2. For each connected component, select an outgoing edge. Merge any newly connected components.
3. Repeat until no connected component has an outgoing edge. If at this point, all nodes are in the same component, then the graph is connected.



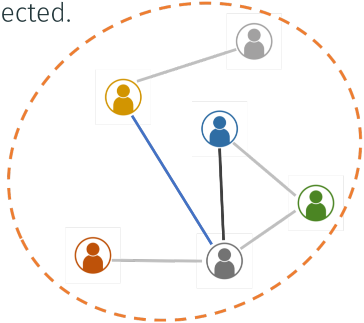
Key Ingredient 2: Boruvka's Algorithm

1. Initialize each node as its own connected component.
2. For each connected component, select an outgoing edge. Merge any newly connected components.
3. Repeat until no connected component has an outgoing edge. If at this point, all nodes are in the same component, then the graph is connected.



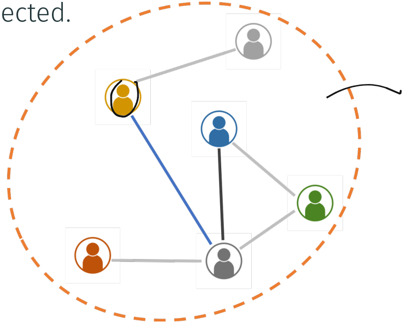
Key Ingredient 2: Boruvka's Algorithm

1. Initialize each node as its own connected component.
2. For each connected component, select an outgoing edge. Merge any newly connected components.
3. Repeat until no connected component has an outgoing edge. If at this point, all nodes are in the same component, then the graph is connected.



Key Ingredient 2: Boruvka's Algorithm

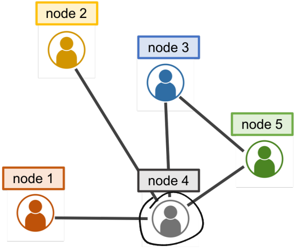
1. Initialize each node as its own connected component.
2. For each connected component, select an outgoing edge. Merge any newly connected components.
3. Repeat until no connected component has an outgoing edge. If at this point, all nodes are in the same component, then the graph is connected.



Converges in $\leq \log_2 n$ rounds.

Key Ingredient 3: Neighborhood Sketches

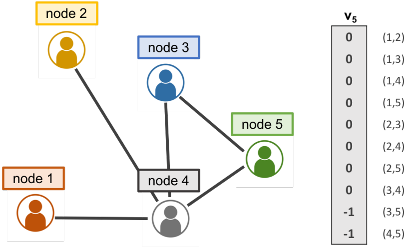
Each node i, can compute a vector $\mathbf{v}_i \in \mathbb{Z}^{\binom{n}{2}}$. v_i has a ± 1 for every edge in the graph and incident to node i. $+1$ is used for edges (i, j) and -1 for edges (j, i).



$\mathbf{v}_4$	
0	(1,2)
0	(1,3)
-1	(1,4)
0	(1,5)
0	(2,3)
-1	(2,4)
0	(2,5)
-1	(3,4)
0	(3,5)
1	(4,5)

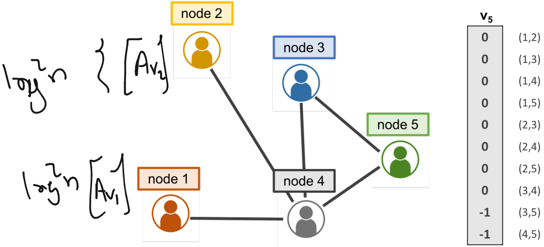
Key Ingredient 3: Neighborhood Sketches

Each node i, can compute a vector $\mathbf{v}_i \in \mathbb{Z}^{\binom{n}{2}}$. v_i has a ± 1 for every edge in the graph and incident to node i. $+1$ is used for edges (i, j) and -1 for edges (j, i).



Key Ingredient 3: Neighborhood Sketches

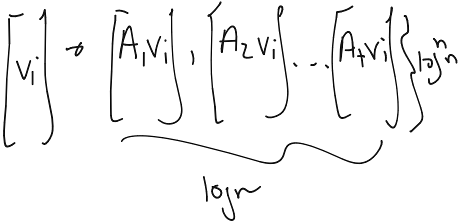
Each node i, can compute a vector $\mathbf{v}_i \in \mathbb{Z}^{\binom{n}{2}}$. v_i has a ± 1 for every edge in the graph and incident to node i. $+1$ is used for edges (i,j) and -1 for edges (j,i).



- Given an ℓ_0 sampling matrix $\mathbf{A} \in \mathbb{Z}^{O(\log^2 n) \times \binom{n}{2}}$, each node can compute $\mathbf{A}v_i \in \mathbb{Z}^{O(\log^2 n)}$ and send it to the central server.
- Using these sketches, with probability $\geq 1 - 1/n^c$, the central server can identify one edge incident to each node – i.e., they can simulate the first iteration of Boruvka's algorithm.

Simulating Boruvka's Algorithm via Sketches

- For independent ℓ_0 sampling matrices $A_1, \dots, A_{\log_2 n}$, each node computes $A_j v_i$ and sends these sketches to the central server. $O(\log^2 n)$ bits in total.

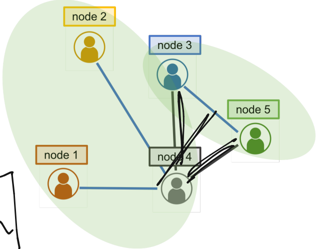


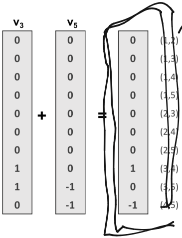
Simulating Boruvka's Algorithm via Sketches

- For independent ℓ_0 sampling matrices $\mathsf{A}_1, \ldots, \mathsf{A}_{\log_2 n}$, each node computes $\mathsf{A}_j v_i$ and sends these sketches to the central server. $O(\log^c n)$ bits in total.
- The central server uses $\mathsf{A}_1 v_1, \ldots, \mathsf{A}_1 v_n$ to simulate the first step of Boruvka's algorithm.

Simulating Boruvka's Algorithm via Sketches

- For independent ℓ_0 sampling matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{\log_2 n}$, each node computes $\mathbf{A}_j v_i$ and sends these sketches to the central server. $O(\log^c n)$ bits in total.
- The central server uses $\mathbf{A}_1 v_1, \ldots, \mathbf{A}_1 v_n$ to simulate the first step of Boruvka's algorithm.
- For each subsequent step j, let $S_1, S_2, \ldots S_c$ be the current connected components. Observe that $\sum_{i \in S_k} v_i$ has non-zero entries corresponding exactly to the outgoing edges of S_k.



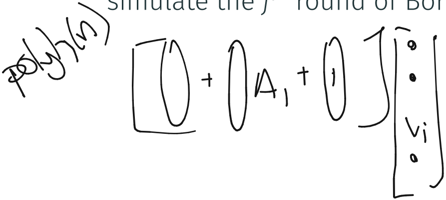


$\log^2 n$? $\log^3 n$

dependent on A_1 therefore can't reapply A_1 to it

Simulating Boruvka's Algorithm via Sketches

- For independent ℓ_0 sampling matrices $A_1, \ldots, A_{\log_2 n}$, each node computes $A_j v_i$ and sends these sketches to the central server. $O(\log^c n)$ bits in total.
- The central server uses $A_1 v_1, \ldots, A_1 v_n$ to simulate the first step of Boruvka's algorithm.
- For each subsequent step j, let $S_1, S_2, \ldots S_c$ be the current connected components. Observe that $\sum_{i \in S_k} v_i$ has non-zero entries corresponding exactly to the outgoing edges of S_k.
- So, from $A_j \sum_{i \in S_k} v_i = \sum_{i \in S_k} A_j v_i$, the server can find an outgoing edge from each connected component S_k. Thus, the server can simulate the j^{th} round of Boruvka's algorithm.



Simulating Boruvka's Algorithm via Sketches

- For independent ℓ_0 sampling matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{\log_2 n}$, each node computes $\mathbf{A}_j v_i$ and sends these sketches to the central server. $O(\log^c n)$ bits in total.

$1 - \frac{1}{n^{10}}$

- The central server uses $\mathbf{A}_1 v_1, \ldots, \mathbf{A}_1 v_n$ to simulate the first step of Boruvka's algorithm.
- For each subsequent step j, let $S_1, S_2, \ldots S_c$ be the current connected components. Observe that $\sum_{i \in S_k} v_i$ has non-zero entries corresponding exactly to the outgoing edges of S_k.
- So, from $\mathbf{A}_j \sum_{i \in S_k} v_i = \sum_{i \in S_k} \mathbf{A}_j v_i$, the server can find an outgoing edge from each connected component S_k. Thus, the server can simulate the j^{th} round of Boruvka's algorithm.
- Overall, using the $\log_2 n$ different sketches from each node, the server can simulate the full algorithm and determine with high probability if the graph is connected or not.

Prof. McGregor

Implementing ℓ_0 Sampling

ℓ_0 Sampling Construction

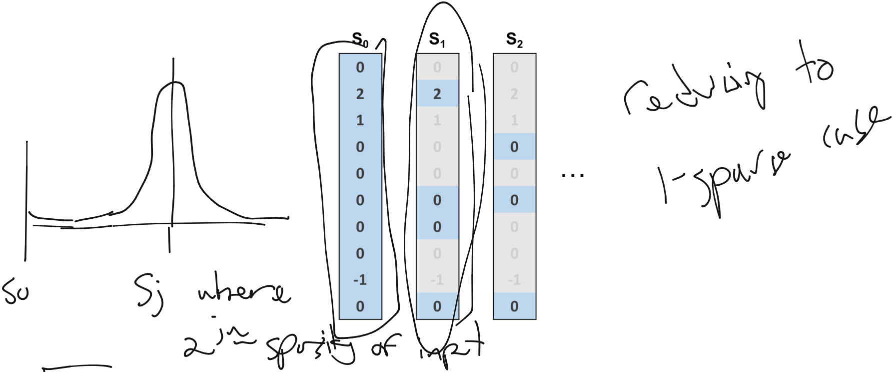
Theorem: There exists a distribution over random matrices $\mathbf{A} \in \mathbb{Z}^{O(\log^2 n)\times n}$ such that for any fixed $x \in \mathbb{Z}^n$, with probability at least $1 - 1/n^c$, we can learn (i, x_i) for some $x_i \neq 0$ from $\mathbf{A}x$.

Construction:

- Let $S_0, S_1, \ldots, S_{\log_2 n}$ be random subsets of $[n]$. Each element is included in S_j independently with probability $1/2^j$.
- For each S_j, compute $a_j = \sum_{i\in S_j} x_i$, $b_j = \sum_{i\in S_j} x_i \cdot i$ and $c_j = \sum_{i\in S_j} x_i \cdot r^i \mod p$, where r is a random value in $[p]$ and p is a prime with $p \geq n^c$ for some large constant c.
- **Exercise:** Show that the vector $[a_1, \ldots, a_{\log_2 n}, b_1, \ldots, b_{\log_2 n}, c_1, \ldots, c_{\log_2 n}]$ can be written as $\mathbf{A}x$, where $\mathbf{A} \in \mathbb{Z}^{3\log_2 n\times n}$ is a random matrix.

Construction Intuition

We will recover a nonzero element from a sampling level when there is exactly one nonzero element at that level.



With good probability, there is will exactly one element at some level. Can improve success probability via repetition.

Recovering Unique Nonzeros

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$. $a_j = \sum_{i \in S_j} x_i$, $b_j = \sum_{i \in S_j} x_i \cdot i$ and $c_j = \sum_{i \in S_j} x_i \cdot r^i \mod p$, where r is a random value in $[p]$ and $p = n^c$ for large enough constant c.

Recovering Unique Nonzeros

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$. $a_j = \sum_{i \in S_j} x_i$, $b_j = \sum_{i \in S_j} x_i \cdot i$ and $c_j = \sum_{i \in S_j} x_i \cdot r^i \mod p$, where r is a random value in $[p]$ and $p = n^c$ for large enough constant c.

Claim 1: If there is a unique $i \in S_j$ with $x_i \neq 0$, then $a_j = x_i$ and $b_j = x_i \cdot i$. So, from these quantities we can exactly determine (i, x_i).

$$\frac{b_j}{a_j} = i$$

$$a_j = x_i$$

Recovering Unique Nonzeros

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$. $a_j = \sum_{i \in S_j} x_i$, $b_j = \sum_{i \in S_j} x_i \cdot i$ and $c_j = \sum_{i \in S_j} x_i \cdot r^i \mod p$, where r is a random value in $[p]$ and $p = n^c$ for large enough constant c.

Claim 1: If there is a unique $i \in S_j$ with $x_i \neq 0$, then $a_j = x_i$ and $b_j = x_i \cdot i$. So, from these quantities we can exactly determine (i, x_j).

Claim 2: c_j lets us test if there is a unique such i. In particular, we check that $\frac{b_j}{a_j} \in [n]$ and that $c_j = a_j \cdot r^{b_j/a_j} \mod p$.

- If there is a unique $i \in S_j$ with $x_i \neq 0$, the test passes.
- If not, it fails with probability at most $\frac{n}{p} = \frac{1}{n^{c-1}}$.

Recovering Unique Nonzeros

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$. $a_j = \sum_{i \in S_j} x_i$, $b_j = \sum_{i \in S_j} x_i \cdot i$ and $c_j = \sum_{i \in S_j} x_i \cdot r^i \mod p$, where r is a random value in $[p]$ and $p = n^c$ for large enough constant c.

Claim 1: If there is a unique $i \in S_j$ with $x_i \neq 0$, then $a_j = x_i$ and $b_j = x_i \cdot i$. So, from these quantities we can exactly determine (i, x_j).

Claim 2: c_j lets us test if there is a unique such i. In particular, we check that $\frac{b_j}{a_j} \in [n]$ and that $c_j = a_j \cdot r^{b_j/a_j} \mod p$.

- If there is a unique $i \in S_j$ with $x_i \neq 0$, the test passes.
- If not, it fails with probability at most $\frac{n}{p} = \frac{1}{n^{c-1}}$.

The problem of recovering a unique $i \in S_j$ with $x_i \neq 0$ is called 1-sparse recovery.