COMPSCI 614: Randomized Algorithms with Applications to Data Science

Prof. Cameron Musco

University of Massachusetts Amherst. Spring 2024.

Lecture 8

Logistics

- Problem Set 2 is due next Wednesday.
- One page project proposal due Tuesday 3/12.
- No quiz this week – focus on the problem set/project proposal.

Summary

Last Time:

- Graph connectivity with low communication
- Approach via Boruvka's algorithm and sparse recovery/ℓ_0 sampling.

Summary

Last Time:

- Graph connectivity with low communication
- Approach via Boruvka's algorithm and sparse recovery/ℓ_0 sampling.

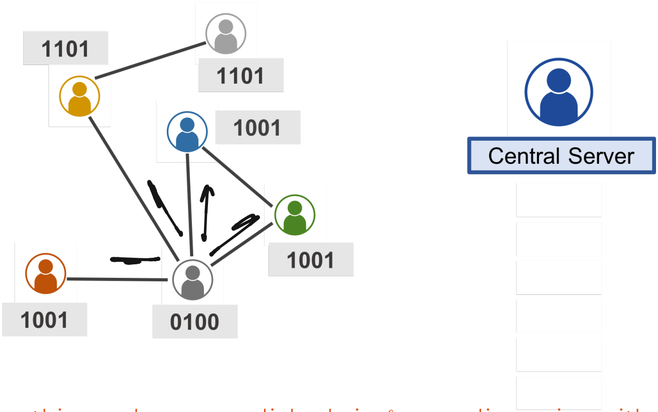
Today:

- Finish up ℓ_0 sampling analysis.
- Other approaches to sparse recovery and applications to data processing in streams.
- The count-sketch algorithm.

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.



Saw how this can be accomplished via ℓ_0 sampling using with messages of size just $O(\log^3 n)$.

$n \log^3 n$

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 $\boxed{(i, x_i)}$ for some $x_i \neq 0$ from $\mathbf{A}x$.

$$
\begin{array}{ccc}
\text{Random sketching matrix } \mathbf{A} & \mathbf{x} & & \mathbf{Ax} \\
\begin{bmatrix} 1 & -1 & 0 & 0 & 1 & -1 & 0 & 1 \\ -1 & 0 & 1 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 1 \\ 0 & -1 & -1 & -1 & 1 & 1 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 0 \\ -2 \\ 0 \\ 0 \\ 3 \\ 0 \end{bmatrix} & = & \begin{bmatrix} 1 \\ -2 \\ 1 \\ 5 \end{bmatrix}
\end{array}
$$

Key Property: 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.

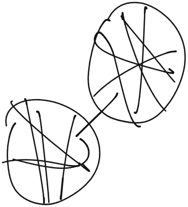
Simulating Boruvka's Algorithm via Sketches

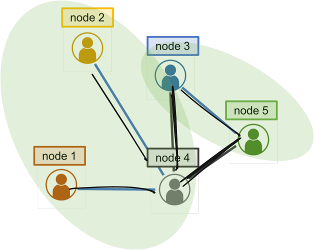
- For independent ℓ_0 sampling matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{\log_2 n}$, each node computes $\mathbf{A}_j v_i$ and sends these sketches to the central server. $O(\log^3 n)$ bits in total.

max degree was Δ : $\log^2 n \cdot \log \Delta$

$\log n \cdot \log^2 \Delta$

- The central server uses $\mathbf{A}_1 v_1, \ldots, \mathbf{A}_1 v_n$ to simulate the first step of Boruvka's – i.e., to identify one outgoing edge from each node.
- 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.





v_3		v_5			
0		0		0	(1,2)
0		0		0	(1,3)
0		0		0	(1,4)
0		0		0	(1,5)
0	+	0	=	0	(2,3)
0		0		0	(2,4)
0		0		0	(2,5)
1		0		1	(3,4)
1		-1		0	(3,5)
0		-1		-1	(4,5)

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. $O(\log^3 n)$ 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 – i.e., to identify one outgoing edge from each node.
- 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.

Implementing ℓ_0 Sampling

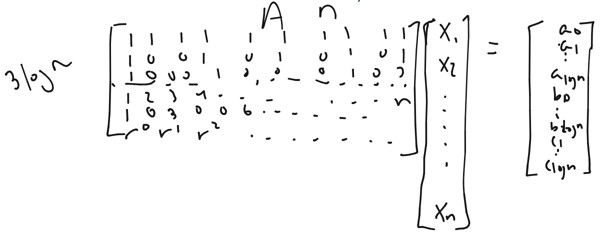
Construction:

$\{1, \ldots n\}$

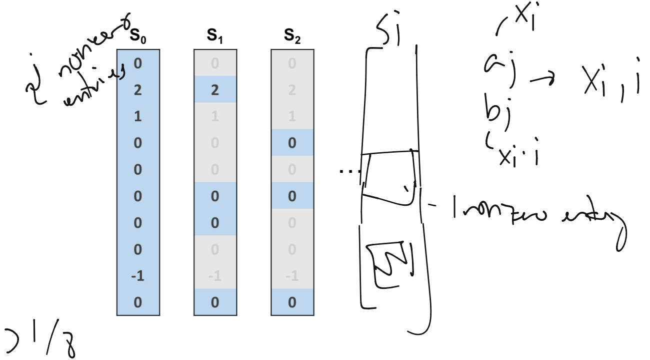
- Let $S_0, S_1, \ldots, S_{\log_2 n}$ be random subsets of $[n]$. Each element is included in S_j independently with probability $1/2^j$.

$\{1, \ldots p\}$

- 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.
- **Observe:** The vector $[a_0, \ldots, a_{\log_2 n}, b_0, \ldots, b_{\log_2 n}, c_0, \ldots, c_{\log_2 n}]$ can be written as $\mathbf{A}x$, where $\mathbf{A} \in \mathbb{Z}^{3\log_2 n \times n}$ is a random matrix.



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

S_j

$S_0, \ldots, S_{\log_2 n}$ 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).

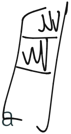
$$a_j = \sum_{i \in S_j} x_i = x_i \qquad b_j = \sum_{i \in S_j} x_i \cdot i = x_i \cdot i$$

$x_1 + x_2$

$$\left[\, A \,\right] \left(\left[\, x \,\right] + \left[\, \varepsilon \,\right] \right)$$

$$[A][D][x]$$

Recovering Unique Nonzeros



$S_0, \ldots, S_{\log_2 n}$ 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.

$x_1 + x_2$ $\qquad b_j = x_1 + 2x_2$

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).

$\frac{b_j}{a_j} = i \in [n]$

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 $c_j = a_j \cdot r^{b_j/a_j} \mod p$.

$c_j = x_1 r + x_2 r^2 \neq x_1 \cdot r^3$

- 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 $\frac{n}{p} = \frac{1}{n^{c-1}}$.

$c_j = a_j \cdot r^i$
$= x_i r^i$
$= \sum_{i \in S_j} x_i \cdot r^i$

Recovering Unique Nonzeros

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 $\frac{n}{p} \leq \frac{1}{n^{c-1}}$.

Recovering Unique Nonzeros

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 $\frac{n}{p} \leq \frac{1}{n^{c-1}}$.

Proof via polynomial identity testing: If $|\{i \in S_j : x_i \neq 0\}| > 1$, then

$$p(r) = c_j - a_j r^{b_j/a_j} \mod p = \sum_{i \in S_j} x_i r^i - a_j r^{b_j/a_j} \mod p$$

= 0 only if test passes

$.5r^3 + 1 \cdot r^4 - 2r^?$

is a non-zero polynomial of degree at most n over $\mathbb{Z}_p$.

Recovering Unique Nonzeros

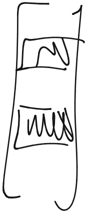
Claim 2: c_j lets us test if there is a unique such i. In particular, we check that $\frac{b_j}{a_j} \in [n]$ and that $c_j = a_j \cdot r^{b_j/a_j} \mod p$.

- 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 $\frac{n}{p} \leq \frac{1}{n^{c-1}}$.

Proof via polynomial identity testing: If $|\{i \in S_j : x_i \neq 0\}| > 1$, then

$$p(r) = c_j - a_j r^{b_j/a_j} \mod p = \sum_{i \in S_j} x_i r^i - a_j r^{b_j/a_j} \mod p$$

is a non-zero polynomial of degree at most n over $\mathbb{Z}_p$.

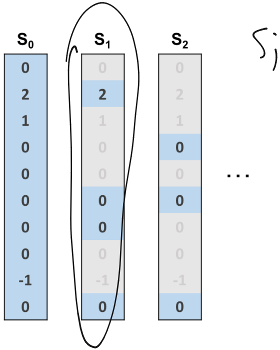


- This polynomial has $\leq n$ roots, so for a random $r \in [p]$, $\Pr[p(r) = 0] \leq \frac{n}{p}$.
- Thus, $c_j = a_j r^{b_j/a_j}$ with probability $\leq \frac{n}{p} \leq \frac{1}{n^{c-1}}$.

Completing The Analysis

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$.

- If any S_j contains a unique i with $x_i \neq 0$, we will recover it.
- It remains to show that with good probability, at least one S_j contains such an i.



Completing The Analysis

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$.

- If any S_j contains a unique i with $x_i \neq 0$, we will recover it.
- It remains to show that with good probability, at least one S_j contains such an i.

nonzero entries in x

Claim: For j with $2^{j-2} \leq \|x\|_0 \leq 2^{j-1}$, $\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] \geq 1/8$.

Completing The Analysis

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$.

- If any S_j contains a unique i with $x_i \neq 0$, we will recover it.
- It remains to show that with good probability, at least one S_j contains such an i.

Claim: For j with $2^{j-2} \leq \|x\|_0 \leq 2^{j-1}$, $\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] \geq 1/8$.

$$\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] = \|x\|_0 \cdot \frac{1}{2^j} \cdot \left(1 - \frac{1}{2^j}\right)^{\|x\|_0 - 1}$$

Completing The Analysis

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$.

- If any S_j contains a unique i with $x_i \neq 0$, we will recover it.
- It remains to show that with good probability, at least one S_j contains such an i.

Claim: For j with $2^{j-2} \leq \|x\|_0 \leq 2^{j-1}$, $\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] \geq 1/8$.

$$\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] = \|x\|_0 \cdot \frac{1}{2^j} \cdot \left(1 - \frac{1}{2^j}\right)^{\|x\|_0 - 1}$$

$$\geq \frac{\|x\|_0}{2^j}\left(1 - \frac{\|x\|_0}{2^j}\right)$$

$$\left(1 - \frac{1}{2^j}\right)^{\|x\|_0 - 1} \geq \left(1 - \frac{1}{2^j}\right)^{\|x\|_0} \geq \left(1 - \frac{\|x\|_0}{2^j}\right)$$

Completing The Analysis

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$.

- If any S_j contains a unique i with $x_i \neq 0$, we will recover it.
- It remains to show that with good probability, at least one S_j contains such an i.

Claim: For j with $\underline{2^{j-2} \leq \|x\|_0 \leq 2^{j-1}}$, $\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] \geq 1/8$.

$$\begin{aligned}
\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] &= \|x\|_0 \cdot \frac{1}{2^j} \cdot \left(1 - \frac{1}{2^j}\right)^{\|x\|_0 - 1} \\
&\geq \frac{\|x\|_0}{2^j}\left(1 - \frac{\|x\|_0}{2^j}\right) \\
&\geq \frac{1}{4} \cdot \left(1 - \frac{1}{2}\right) = \frac{1}{8}.
\end{aligned}$$

$2^{j-2} \leq \|x\|_0$

$\frac{1}{4} = 2^{-2} \leq \frac{\|x\|_0}{2^j}$

Completing The Analysis

Recall: $S_0, \ldots, S_{\log_2 n}$ are random subsets of $[n]$, sampled at rates $1/2^j$.

- If any S_j contains a unique i with $x_i \neq 0$, we will recover it.
- It remains to show that with good probability, at least one S_j contains such an i.

4, 7, 8, always $\exists\, j \in \{0, \log_2 n+1\}$ satisfies this

Claim: For j with $2^{j-2} \leq \|x\|_0 \leq 2^{j-1}$, $\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] \geq 1/8$.

$Ax_1 + Ax_2 = A(x_1+x_2)$

$$\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] = \|x\|_0 \cdot \frac{1}{2^j} \cdot \left(1 - \frac{1}{2^j}\right)^{\|x\|_0 - 1}$$
$$\geq \frac{\|x\|_0}{2^j}\left(1 - \frac{\|x\|_0}{2^j}\right)$$
$$\geq \frac{1}{4} \cdot \left(1 - \frac{1}{2}\right) = \frac{1}{8}.$$

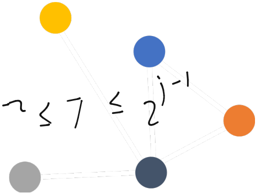
$3\log n$

If we repeat the whole process $t = O(\log n)$ times, with probability $\geq 1 - 1/n^c$ we will recover some nonzero element of x. In total, $\mathbf{A}$ is a random matrix with $t \cdot \log_2 n = O(\log^2 n)$ rows.

Application to Streaming Computation

A Graph Streaming Problem

Consider a setting where an algorithm must process a stream of edge insertions or deletions, which define a graph. At the end of the stream, the algorithm should output whether that graph is connected or not.



X

T non zero

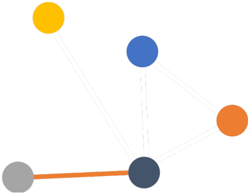
fix j : $2^{j-2} \le T \le 2^{j-1}$

w/ prob $\ge 1/8$ there is single nonzero here

S_0 S_1 S_2 S_3 S_4 S_5

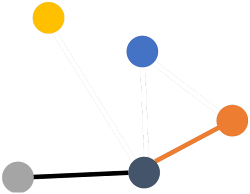
A Graph Streaming Problem

Consider a setting where an algorithm must process a stream of edge insertions or deletions, which define a graph. At the end of the stream, the algorithm should output whether that graph is connected or not.



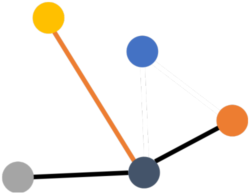
A Graph Streaming Problem

Consider a setting where an algorithm must process a stream of edge insertions or deletions, which define a graph. At the end of the stream, the algorithm should output whether that graph is connected or not.



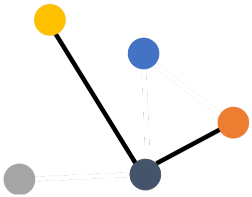
A Graph Streaming Problem

Consider a setting where an algorithm must process a stream of edge insertions or deletions, which define a graph. At the end of the stream, the algorithm should output whether that graph is connected or not.



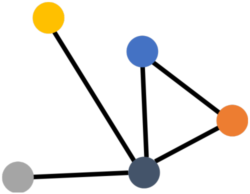
A Graph Streaming Problem

Consider a setting where an algorithm must process a stream of edge insertions or deletions, which define a graph. At the end of the stream, the algorithm should output whether that graph is connected or not.



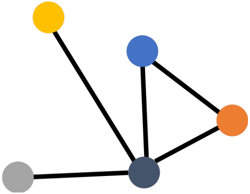
A Graph Streaming Problem

Consider a setting where an algorithm must process a stream of edge insertions or deletions, which define a graph. At the end of the stream, the algorithm should output whether that graph is connected or not.



A Graph Streaming Problem

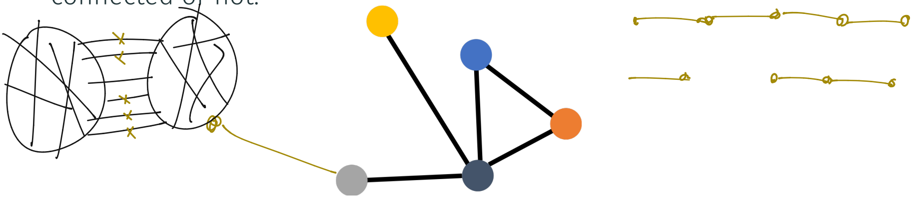
Consider a setting where an algorithm must process a stream of edge insertions or deletions, which define a graph. At the end of the stream, the algorithm should output whether that graph is connected or not.



Algorithmic Question: How much memory must an algorithm use to solve this problem with high probability? $O(n^2)$

A Graph Streaming Problem

Consider a setting where an algorithm must process a stream of edge insertions or deletions, which define a graph. At the end of the stream, the algorithm should output whether that graph is connected or not.



Algorithmic Question: How much memory must an algorithm use to solve this problem with high probability?

$O(n^2)$

What is the worst-case memory required by a naive deterministic algorithm that just stores the current state of the graph? How can you improve on this when there are no edge deletions?

$O(n)$

Randomized Solution via ℓ_0 sampling

- The algorithm samples independent ℓ_0 sampling matrices $\mathsf{A}_1, \ldots, \mathsf{A}_{\log_2 n}$ and maintains $\mathsf{A}_j v_u$ for all j and all $u \in [n]$, where $v_u \in \mathbb{R}^{\binom{n}{2}}$ is the incidence vector for node u.
- $\boxed{O(n \log^3 n)}$ bits of storage in total.

$A_j v_u = O(\log^2 n)$ space

n nodes

$\log_2 n$ ands $A_1 \ldots A_{\log n}$

total space $O(n \log^3 n)$

Randomized Solution via ℓ_0 sampling

- The algorithm samples independent ℓ_0 sampling matrices $\mathsf{A}_1, \ldots, \mathsf{A}_{\log_2 n}$ and maintains $\mathsf{A}_j v_u$ for all j and all $u \in [n]$, where $v_u \in \mathbb{R}^{\binom{n}{2}}$ is the incidence vector for node u.
- $O(n \log^3 n)$ bits of storage in total.
- Key Idea: Linear Updates. When an edge (u, v) is inserted or deleted, one entry is either incremented or decremented in each of v_u, v_v. The algorithm can update $\mathsf{A}_j v_u$ and $\mathsf{A}_j v_v$ in $O(\log^2 n)$ time – simply set $\mathsf{A}_j v_u = \mathsf{A}_j v_u \pm \mathsf{A}_{j,k}$.

Randomized Solution via ℓ_0 sampling

- The algorithm samples independent ℓ_0 sampling matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{\log_2 n}$ and maintains $\mathbf{A}_j v_u$ for all j and all $u \in [n]$, where $v_u \in \mathbb{R}^{\binom{n}{2}}$ is the incidence vector for node u.
- $O(n \log^3 n)$ bits of storage in total.
- **Key Idea: Linear Updates.** When an edge (u, v) is inserted or deleted, one entry is either incremented or decremented in each of v_u, v_v. The algorithm can update $\mathbf{A}_j v_u$ and $\mathbf{A}_j v_v$ in $O(\log^2 n)$ time – simply set $\mathbf{A}_j v_u = \mathbf{A}_j v_u \pm \mathbf{A}_{j,k}$.

$$\underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & 1 & -1 & 0 & 1 \\ -1 & 0 & 1 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 1 \\ 0 & -1 & -1 & -1 & 1 & 1 & 1 & 0 \end{bmatrix}}_{l_0 \text{ sampling matrix } \mathbf{A_j}} \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{v_u} = \underbrace{\begin{bmatrix} 1 \\ -2 \\ 1 \\ 1 \end{bmatrix}}_{\mathbf{A_j} v_u}$$

Randomized Solution via ℓ_0 sampling

- The algorithm samples independent ℓ_0 sampling matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{\log_2 n}$ and maintains $\mathbf{A}_j v_u$ for all j and all $u \in [n]$, where $v_u \in \mathbb{R}^{\binom{n}{2}}$ is the incidence vector for node u.
- $O(n \log^3 n)$ bits of storage in total.
- **Key Idea: Linear Updates.** When an edge (u, v) is inserted or deleted, one entry is either incremented or decremented in each of v_u, v_v. The algorithm can update $\mathbf{A}_j v_u$ and $\mathbf{A}_j v_v$ in $O(\log^2 n)$ time – simply set $\mathbf{A}_j v_u = \mathbf{A}_j v_u \pm \mathbf{A}_{j,k}$.

(u,v)

l_0 sampling matrix $\mathbf{A_j}$ $\quad$ v_u $\quad$ $\mathbf{A_j}v_u$

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

Randomized Solution via ℓ_0 sampling

- The algorithm samples independent ℓ_0 sampling matrices $\mathsf{A}_1, \ldots, \mathsf{A}_{\log_2 n}$ and maintains $\mathsf{A}_j v_u$ for all j and all $u \in [n]$, where $v_u \in \mathbb{R}^{\binom{n}{2}}$ is the incidence vector for node u.
- $O(n \log^3 n)$ bits of storage in total.
- **Key Idea: Linear Updates.** When an edge (u, v) is inserted or deleted, one entry is either incremented or decremented in each of v_u, v_v. The algorithm can update $\mathsf{A}_j v_u$ and $\mathsf{A}_j v_v$ in $O(\log^2 n)$ time – simply set $\mathsf{A}_j v_u = \mathsf{A}_j v_u \pm \mathsf{A}_{j,k}$.

$$\underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & 1 & -1 & 0 & 1 \\ -1 & 0 & 1 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 1 \\ 0 & -1 & -1 & -1 & 1 & 1 & 1 & 0 \end{bmatrix}}_{l_0 \text{ sampling matrix } \mathbf{A_j}} \; \begin{array}{r} \\ \\ \\ \\ \\ \\ 6 \\ \\ \end{array}\!\underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{v_u} = \underbrace{\begin{bmatrix} 1 \\ -3 \\ 1 \\ 2 \end{bmatrix}}_{\mathbf{A_j}v_u}$$

Randomized Solution via ℓ_0 sampling

- The algorithm samples independent ℓ_0 sampling matrices $A_1, \ldots, A_{\log_2 n}$ and maintains $A_j v_u$ for all j and all $u \in [n]$, where $v_u \in \mathbb{R}^{\binom{n}{2}}$ is the incidence vector for node u.
- $O(n \log^3 n)$ bits of storage in total.
- **Key Idea: Linear Updates.** When an edge (u, v) is inserted or deleted, one entry is either incremented or decremented in each of v_u, v_v. The algorithm can update $A_j v_u$ and $A_j v_v$ in $O(\log^2 n)$ time – simply set $A_j v_u = A_j v_u \pm A_{j,k}$.
- At the end of the stream (or at any time during it) can use the sketched neighborhoods to simulate Boruvka's algorithm and determine connectivity with high probability.

Randomized Solution via ℓ_0 sampling

- The algorithm samples independent ℓ_0 sampling matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{\log_2 n}$ and maintains $\mathbf{A}_j v_u$ for all j and all $u \in [n]$, where $v_u \in \mathbb{R}^{\binom{n}{2}}$ is the incidence vector for node u.
- $O(n \log^3 n)$ bits of storage in total.
- **Key Idea: Linear Updates.** When an edge (u, v) is inserted or deleted, one entry is either incremented or decremented in each of v_u, v_v. The algorithm can update $\mathbf{A}_j v_u$ and $\mathbf{A}_j v_v$ in $O(\log^2 n)$ time – simply set $\mathbf{A}_j v_u = \mathbf{A}_j v_u \pm \mathbf{A}_{j,k}$.
- At the end of the stream (or at any time during it) can use the sketched neighborhoods to simulate Boruvka's algorithm and determine connectivity with high probability.
- Can think of the algorithm as computing $\mathbf{AB} \in \mathbb{R}^{\log^3 n \times n}$ where $\mathbf{A} \in \mathbb{R}^{\log^3 n \times \binom{n}{2}}$ is made up of the appended sketching matrices and $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ is the vertex-edge-incidence matrix.

$B \quad \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{bmatrix}$

Other Applications of Linear Sketching

Linear Sketching

- ℓ_0 sampling is an example of a linear sketching algorithm. We compress our data via a random linear function (i.e., the random matrix $\mathbf{A}$), and prove that we can still recover useful information from the compression.

$$
\underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & 1 & -1 & 0 & 1 \\ -1 & 0 & 1 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 1 \\ 0 & -1 & -1 & -1 & 1 & 1 & 1 & 0 \end{bmatrix}}_{\text{Random sketching matrix } \mathbf{A}}
\underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ -2 \\ 0 \\ 0 \\ 3 \\ 0 \end{bmatrix}}_{x}
=
\underbrace{\begin{bmatrix} 1 \\ -2 \\ 1 \\ 5 \end{bmatrix}}_{\mathbf{A}x}
$$

Linear Sketching

- ℓ_0 sampling is an example of a linear sketching algorithm. We compress our data via a random linear function (i.e., the random matrix $\mathbf{A}$), and prove that we can still recover useful information from the compression.

$$
\underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & 1 & -1 & 0 & 1 \\ -1 & 0 & 1 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 1 \\ 0 & -1 & -1 & -1 & 1 & 1 & 1 & 0 \end{bmatrix}}_{\text{Random sketching matrix } \mathbf{A}}
\underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ -2 \\ 0 \\ 0 \\ 3 \\ 0 \end{bmatrix}}_{\mathbf{x}}
=
\underbrace{\begin{bmatrix} 1 \\ -2 \\ 1 \\ 5 \end{bmatrix}}_{\mathbf{A}\mathbf{x}}
$$

- Linearity is useful because it lets us easily aggregate sketches in distributed settings and update sketches in streaming settings.
- Aside from recovering non-zero entries we might want to estimate norms or other aggregate statistics of $\mathbf{x}$, find large magnitude entries, sample entries with probabilities according to their magnitudes.

Linear Sketching for Heavy-Hitters Identification

Goal: For a vector $\mathbf{x} \in \mathbb{R}^n$ we would like to find all entries of $\mathbf{x}$ with magnitude at least $\epsilon\|\mathbf{x}\|_2$ or $\epsilon\|\mathbf{x}\|_1$. $\epsilon > 0$

frequent items estimation

Goal: For a vector $\mathbf{x} \in \mathbb{R}^n$ we would like to find all entries of $\mathbf{x}$ with magnitude at least $\epsilon\|\mathbf{x}\|_2$ or $\epsilon\|\mathbf{x}\|_1$.

count-min sketch

Common Application: count sketch

- $\mathbf{x}$ is a vector of counts (e.g., views of videos, searches for products, visits from IP addresses, etc.) and we would like to identity all items with large counts.
- We often cannot store all of $\mathbf{x}$ in one place but must store a small-space compression of $\mathbf{x}$ as counts are updated over time, or must aggregate information about $\mathbf{x}$ across multiple machines.

$$\begin{array}{cc} & x \\ \begin{array}{c} vid_1 \\ vid_2 \\ \vdots \\ vid_n \end{array} & \begin{bmatrix} 100 \\ 2 \\ 10000 \\ \vdots \\ \; \end{bmatrix} \end{array}$$

Count Sketch

Set up: We would like to estimate all entries of a vector $\mathbf{x} \in \mathbb{R}^n$ up to error $\epsilon\|\mathbf{x}\|_2$ with probability at least $1-\delta$, from a small linear sketch, of size $O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$.

$$O\left(\frac{\log(1/\delta)}{\varepsilon^2}\right)\Big[\quad A \quad\Big]\Big[\; X \;\Big]$$

Count Sketch

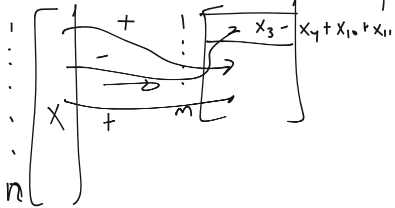
Set up: We would like to estimate all entries of a vector $\mathbf{x} \in \mathbb{R}^n$ up to error $\epsilon\|\mathbf{x}\|_2$ with probability at least $1 - \delta$, from a small linear sketch, of size $O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$.

repetition to get high prob.

- Let $m = O(1/\epsilon^2)$ and $t = O(\log(1/\delta))$.
- Pick t random pairwise independent hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_t : [n] \to [m]$.
- Pick t random pairwise independent hash functions $\mathbf{s}_1, \ldots, \mathbf{s}_t : [n] \to \{-1, 1\}$.

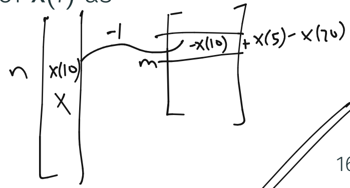
$\Pr(h(i) = h(j)) = K$, $i \neq j$

$= \frac{1}{m^2}$



Set up: We would like to estimate all entries of a vector $\mathbf{x} \in \mathbb{R}^n$ up to error $\epsilon\|\mathbf{x}\|_2$ with probability at least $1-\delta$, from a small linear sketch, of size $O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$.

- Let $m = O(1/\epsilon^2)$ and $t = O(\log(1/\delta))$.
- Pick t random pairwise independent hash functions $\underline{\mathbf{h}_1, \ldots, \mathbf{h}_t} : [n] \to [m]$.
- Pick t random pairwise independent hash functions $\underline{\mathbf{s}_1, \ldots, \mathbf{s}_t} : [n] \to \{-1, 1\}$.
- Compute t independent estimates of $\mathbf{x}(i)$ as $\tilde{\mathbf{x}}_j(i) = \mathbf{s}_j(i) \cdot \sum_{k:\mathbf{h}_j(k)=\mathbf{h}_j(i)} \mathbf{x}(k) \cdot \mathbf{s}(k)$.



$x(10) \approx (-1)\cdot(-x(10) + x(5) - x(20))$
$= x(10) +$ random noise