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.

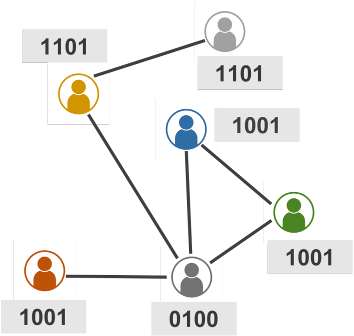
Today:

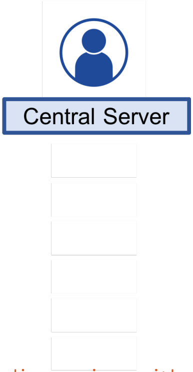
- Finish up ℓ_0 sampling analysis.
- Other approaches to sparse recovery and applications to data processing in streams.
- The count-sketch algorithm.

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 $O(\log^3 n)$.

Key Ingredient 1: ℓ_0 Sampling

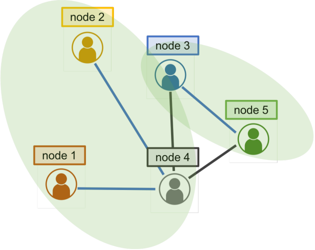
Theorem: There exists a distribution over random matrices $\mathbf{A} \in \mathbb{Z}^{O(\log^2 n) \times n}$ such that for any fixed $x \in \mathbb{Z}^n$, with probability at least $1 - 1/n^c$, we can learn (i, x_i) for some $x_i \neq 0$ from $\mathbf{A}x$.

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

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.
- 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)

- 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

Implementing ℓ_0 Sampling

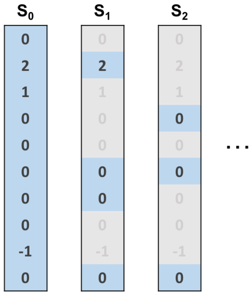
ℓ_0 Sampling Construction

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.
- **Observe:** 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

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

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

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

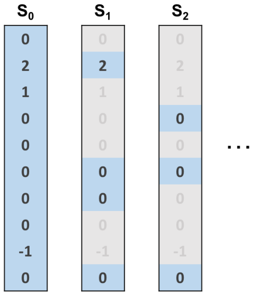
is a non-zero polynomial of degree at most n over $\mathbb{Z}_p$.

- 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.



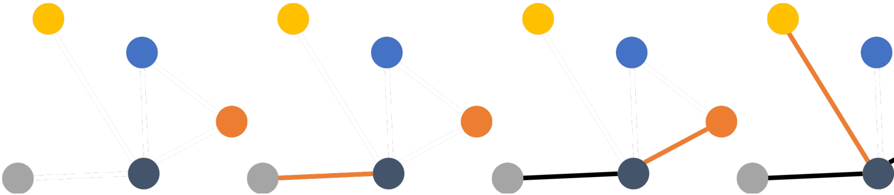
Claim: For j with $2^{j-2} \leq \|x\|_0 \leq 2^{j-1}$, $\Pr[|\{i \in S_j : x_i \neq 0\}| = 1] \geq 1/8$.

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

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.



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

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?

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 $\mathbf{A}_j v_u = \mathbf{A}_j v_u \pm \mathbf{A}_{j,k}$.

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

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.

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

- 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 $\epsilon\|\mathbf{x}\|_1$.

Common Application:

- $\mathbf{x}$ 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.

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 $O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$.

- 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 $\mathbf{s}_1, \ldots, \mathbf{s}_t : [n] \to \{-1, 1\}$.
- Compute t independent estimates of $\mathbf{x}(i)$ as $\tilde{\mathbf{x}}_j(i) = \mathbf{s}(i) \cdot \sum_{k:\mathbf{h}_j(k)=\mathbf{h}_j(i)} \mathbf{x}(k) \cdot \mathbf{s}(k)$.
- Output the median of $\{\tilde{\mathbf{x}}_1(i), \ldots, \tilde{\mathbf{x}}_t(i)\}$ as our estimate.