Lecture 3

TTIC 41000: Algorithms for Massive Data

Toyota Technological Institute at Chicago

Spring 2021

Instructor: Sepideh Mahabadi

This Lecture

- ❑ L_0 Samplers
- ❑ Graph Streaming Algorithm

Sketch

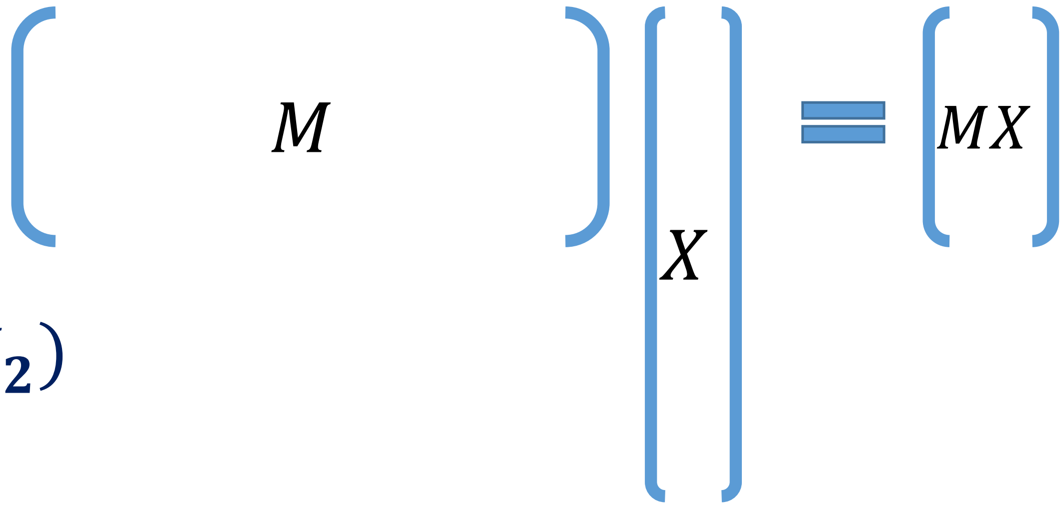
- Large Data set $\boldsymbol{D}$
- An algorithm $\boldsymbol{Alg}$ that produces the output $\boldsymbol{Alg(D)}$

- **Sketch:** $\boldsymbol{f(D)}$ which has much smaller size
- $\boldsymbol{Alg'\big(f(D)\big) \approx Alg(D)}$

Linear Sketch

- ❑ Large Data set $\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{n \times d}}$ which is a **vector** or a **matrix**
- ❑ An algorithm Alg that produces the output $Alg(X)$

- ❑ Sketch: $f(X) = \boldsymbol{M} \cdot \boldsymbol{X}$ where $\boldsymbol{M} \in \mathbb{R}^{\boldsymbol{k \times m}}$ has much smaller size $(\boldsymbol{k} \times \boldsymbol{d})$
- ❑ $Alg'(M \cdot X) \approx Alg(X)$



- ➢ $\boldsymbol{f(X_1 + X_2) = f(X_1) + f(X_2)}$
- ➢ $\boldsymbol{f(aX) = af(X)}$

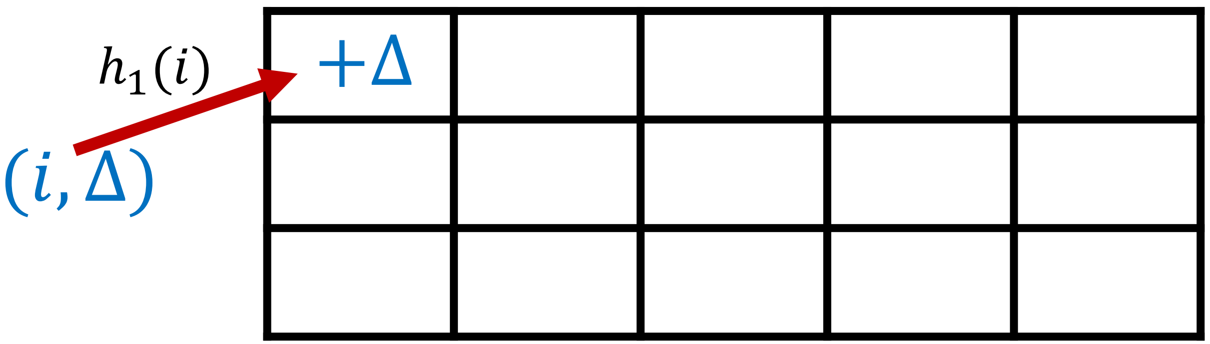
Recap - Count Min

Turnstile Model: input is a stream of updates (i, Δ), where $i \in [m]$

#rows r $= O(\log 1/\delta)$
#buckets/row b = $O(2k)$

- **Hash** $\forall j \leq r$: $h_j : [m] \to [b]$

- **Update:** $C[j, h_j(i)] += \Delta$
-



Count-Min as a Linear Sketch

- Data Set: a vector $X \in \mathbb{R}^m$
- Sketch: $f(X) = M \cdot X$ where $\boldsymbol{M} \in \mathbb{R}^{(\log\frac{1}{\delta}\cdot\frac{1}{\epsilon})\times m}$
- **Each column** of $\boldsymbol{M_i} \in \mathbb{R}^{(\frac{1}{\epsilon})\times m}$ has a **1** in a **random row**
- $\boldsymbol{M}$ **is concatenation** of $\log 1/\delta$ such M_i

$$\begin{pmatrix} 1,0,0,0,1,0,0,0 \\ 0,0,1,1,0,0,0,1 \\ 0,1,0,0,0,1,1,0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_8 \end{pmatrix} = \begin{pmatrix} x_1 + x_5 \\ x_3 + x_4 + x_8 \\ x_2 + x_6 + x_7 \end{pmatrix}$$

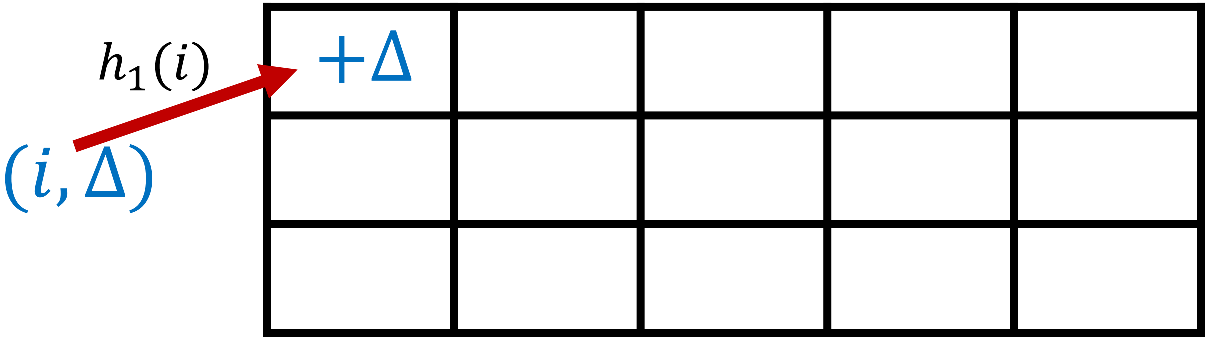
Recap - Count Sketch

Turnstile Model: input is a stream of updates (i, Δ), where $i \in [m]$

#rows r $= O(\log 1/\delta)$

#buckets/row b = $O(9k)$

- **Hash** $h_j : [m] \to [b]$
- **Sign** $\sigma_j : [m] \to \{-1, +1\}$



- **Update:** $C[j, h_j(i)] \mathrel{+}= \sigma_j(i) \cdot \Delta$
-

Count-Sketch as a Linear Sketch

- Data Set: a vector $X \in \mathbb{R}^m$
- Sketch: $f(X) = M \cdot X$ where $M \in \mathbb{R}^{(\log\frac{1}{\delta}\cdot\frac{1}{\epsilon^2})\times m}$
- Each column of $M_i \in \mathbb{R}^{(\frac{1}{\epsilon^2})\times m}$ has a random $\pm\mathbf{1}$ in a random row
- M is concatenation of $\log 1/\delta$ such M_i

$$\begin{pmatrix} 1,0,0,0,-1,0,0,0 \\ 0,0,1,-1,0,0,0,1 \\ 0,-1,0,0,0,1,-1,0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_8 \end{pmatrix} = \begin{pmatrix} x_1 - x_5 \\ x_3 - x_4 + x_8 \\ -x_2 + x_6 - x_7 \end{pmatrix}$$

Recap From Previous Lectures

- ❑ Streaming model of computation
- ❑ Input is a stream of number (insertion-only, insertion-deletion, turnstile)
- ❑ Norm Approximation
- ❑ Coordinate Approximation
- ❑ **Coordinate Sampling**

L_p-Sampler

- Sample one coordinate of a vector $\boldsymbol{x}$ w.p. $\frac{|x_i|^p}{\|x\|_p^p}$
- Approximate variant $(1+\epsilon)\frac{|x_i|^p}{\|x\|_p^p} + n^{-c}$

L_0-Sampler

- Sample one of the non-zero coordinates with uniform probability.
- Sample one coordinate of a vector x w.p. $\frac{\mathbf{1}}{\|\boldsymbol{x}\|_{\mathbf{0}}} + \boldsymbol{n}^{-\boldsymbol{c}}$

- Component in many streaming (and more generally big data) algorithms.
- Works under dynamic updates and turnstile

- Assume there are n coordinates and the coordinates are always integers between $[-W, W]$ for $W = poly(n)$

Algorithm 1

- For $0 \le i \le \log n$ let $\boldsymbol{S_i}$ be a set where each element is picked independently w.p. $1/2^i$
  - $\boldsymbol{A_i} = \sum_{j \in S_i} \boldsymbol{x_j}$
  - $\boldsymbol{B_i} = \sum_{j \in S_i} \boldsymbol{j \cdot x_j}$
  - $\boldsymbol{C_i} = \sum_{j \in S_i} \boldsymbol{x_j r^j} \textbf{ mod } \boldsymbol{p}$

> $\boldsymbol{p} = O(poly\ n)$ is a prime number and

> $\boldsymbol{r}$ is chosen uniformly at random from $\{1, \cdots, p-1\}$

- Assume we can detect some S_i that contains **a single non-zero coordinate**.
  - That coordinate is a random sample.
  - Return $\boldsymbol{B_i/A_i}$ as the index and $\boldsymbol{A_i}$ as the value.

- How to test S_i has a single non-zero coordinate? $\boldsymbol{B_i/A_i \in [n]}$ and $\boldsymbol{C_i = A_i r^{B_i/A_i} \bmod p}$

Algorithm 1

- For $0 \le i \le \log n$ let S_i be a set where each element is picked independently w.p. $1/2^i$
  - $A_i = \sum_{j \in S_i} x_j$
  - $B_i = \sum_{j \in S_i} j \cdot x_j$
  - $C_i = \sum_{j \in S_i} x_j r^j$ mod p
- How to test S_i has a single non-zero coordinate? $\boldsymbol{B_i/A_i \in [n]}$ and $\boldsymbol{C_i = A_i r^{B_i/A_i} \bmod p}$

- Clearly if S_i has one non-zero coordinate, it passes the test.
- If S_i has more than one non-zero coordinate, it fails with high probability
  - Consider the polynomial $\boldsymbol{f(y) = \sum_{j \in S_i} x_j y^j - A_i y^{B_i/A_i} \bmod p}$
  - Its degree is at most n
  - $\boldsymbol{p} = O(poly\ n)$ and $\boldsymbol{r}$ is chosen randomly
  - The probability that for a random r it is 0 is at most $n/(p-1) \le 1/poly(n)$

Algorithm 1

- Show there is at least one S_i with exactly one non-zero coordinate
  - with constant probability

- Let $\boldsymbol{I} \subseteq [\boldsymbol{n}]$ be the set of indices of non-zero coordinates
- let $\boldsymbol{i} \in [\mathbf{log}\, \boldsymbol{n}]$ be s.t. $2^{i-2} \leq |I| \leq 2^{i-1}$,

$$\Pr[|I \cap S_i| = 1] = \sum_{j \in I} \Pr[j \in S_i, \text{ and } \forall k \in I \setminus \{j\}: k \notin S_i] = \sum_{j \in I} \left(\frac{1}{2^i}\right)\left(1 - \frac{1}{2^i}\right)^{|I|-1} \geq$$

$$\frac{|I|}{2^i} \cdot \left(\frac{1}{e}\right)^{\frac{|I|-1}{2^i-1}} \geq \frac{1}{4} \cdot \left(\frac{1}{e}\right)^{\frac{1}{2}} \geq \frac{1}{8}$$

- We can then boost the probability of success **by repetition**
- Again we need to maintain the sets S_i which needs lots of space.

k-wise independent

- $h: [n] \to [D]$ is a ***k*-wise independent** hash function, if for any k distinct indices $i_1, \cdots, i_k \in [n]$ and any k values $t_1, \cdots, t_k \in [D]$

$$\Pr[h(i_1) = t_1, \cdots, h(i_k) = t_k] = \frac{1}{|D|^k}$$

- E.g. function: $h(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_0 \bmod p$ where coefficients are chosen randomly from $\{0, \cdots, p-1\}$ and $p = poly(D)$ is a prime.
- Can be stored using $O(k \log D)$ bits

Algorithm 2 – using hash functions

- To define S_i,
  - Let $h: [n] \rightarrow [O(2^i)]$ be a **2 −wise independent** hash function
  - Let $S_i = \{j: h(j) \text{ is divisible } by\ 2^i\}$,
  - Thus $\Pr[j \in S_i] \approx 1/2^i$
- Again, need to prove that with constant prob. at least one S_i traps **exactly one** non-zero coordinate
- Let I be the set of indices of non-zero coordinates, and let i be s.t. $2^{i-2} \leq |I| \leq 2^{i-1}$,

$$\Pr[|I \cap S_i| = 1] = \sum_{j\in I} \Pr[j \in S_i, \text{and } \forall k \in I \setminus \{j\}: k \notin S_i] = \sum_{j\in I} \Pr[j \in S_i]\Pr[\forall k \in I \setminus \{j\}: k \notin$$

Some references for general L_p sampler

- ❑ Morteza Monemizadeh and David P Woodruff’ 2010. 1-pass relative-error lp-sampling with applications

- ❑ Hossein Jowhari, Mert Sa˘glam, and G´abor Tardos’ 2011. Tight bounds for lp samplers, finding duplicates in streams, and related problems

Streaming Graph Algorithms

Graph Streaming Algorithms

- Input is a graph $G(V, E)$ with n vertices and m edges
- Arrival model, (edge arrival, vertex arrival)
- Semi Streaming, the space usage of the algorithm is $O(n\, polylog\, n)$
- But other regimes, e.g. $O(poly \log n)$ space and $o(n^2)$ space, are also considered.
- Graph might be insertion-only, or dynamic (e.g. edges might get deleted).
- Random order streams are also considered in this model.

Warm up I - Undirected Connectivity Problem

- Given a stream of edges, maintain the connected components.
- Requires $\Omega(n)$ space
- Simple Algorithm, for insertion-only streams
  - Maintain a **spanning forest** (only requires $\boldsymbol{O(n)}$ space)
  - Upon arrival of $e = (u, v)$ see if $\boldsymbol{C(u) = C(v)}$ then do **nothing**. Otherwise **merge** the components $C(u)$ and $C(v)$

➢ This is much better than $O(m)$ which could be as large as $\Omega(n^2)$

➢ What about deletions?

Warm up II - k- Edge Connectivity

- **Goal:** Is the graph k-Edge Connected? (e.g. k-edge disjoint paths between any pair of vertices)
- **Sketch:** Maintain k **forests** $\boldsymbol{F_1, \cdots, F_k}$
  - Upon arrival of $\boldsymbol{e = (u, v)}$, find the first forest F_i where e connects different connected components in F_i and add the edge to it. Otherwise ignore the edge
- **space**: $O(nk)$ which if k is small, is much smaller than n^2
- **Correctness:**
  - If the union of the forests $\mathbf{F} = \boldsymbol{F_1} \cup \cdots \cup \boldsymbol{F_k}$ are k-connected, then so is G
  - If G is k-connected, but F is not, then there should be a cut in F with less than k edges passing it.
  - So one F_i has no edge passing the cut, but there was one edge in G passing it. This is a contradiction

Warm up III – Unweighted Matching

- **Goal:** Find a maximum matching in the graph
- **Sketch:** Maintain a maximal matching
  - Upon arrival of $\boldsymbol{e = (u, v)}$**,** if both u and v are unmatched, keep e in the solution, otherwise ignore the edge
- **space**: $O(n)$
- **Approximation factor: 2**
  - For each edge $e = (u, v)$ in the optimal matching M^* that is not in the reported solution M, charge it to the edge in M which is incident to either u or v
  - Such an edge should exist otherwise we had picked e in M
  - Each edge in M is charged at most twice

Warm up III – Unweighted Matching

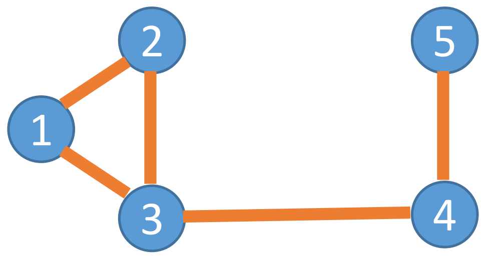
- **Goal:** Find a maximum matching in the graph
- **Sketch:** Maintain a maximal matching
  - Upon arrival of $\boldsymbol{e = (u, v)}$**,** if both u and v are unmatched, keep e in the solution, otherwise ignore the edge
- **space**: $O(n)$
- **Approximation factor: 2**

- Many works on streaming matching (weighted, random order streams, more than one pass, lower bound results …)

Undirected connectivity in dynamic graphs

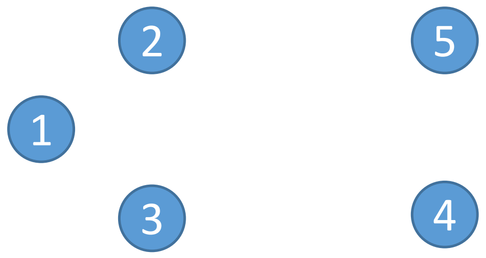
- Given a stream of edge insertion and deletions, maintain the connected components of the graph
- Requires $\Omega(n)$ space
- Insertion only, we get $O(n)$ space algorithm
- Ingredients
  - An offline algorithm
  - Vector representation
  - L_0 sampler

An offline algorithm



An offline algorithm

For all $v \in V$, let $\boldsymbol{CC(v) = v}$

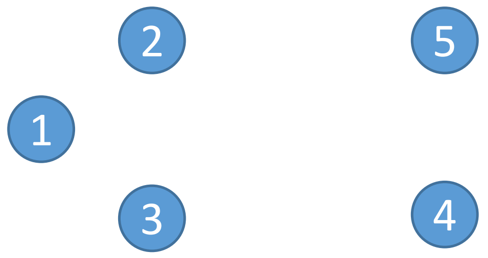


An offline algorithm

For all $v \in V$, let $\boldsymbol{CC(v) = v}$

For $\log n$ iterations

- Pick an incident edge to each connected component
- Merge the components that have an incident edge which we picked

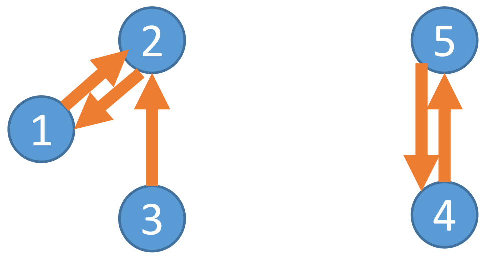


An offline algorithm

For all $v \in V$, let $\boldsymbol{CC(v) = v}$

For $\log n$ iterations

- Pick an incident edge to each connected component
- Merge the components that have an incident edge which we picked

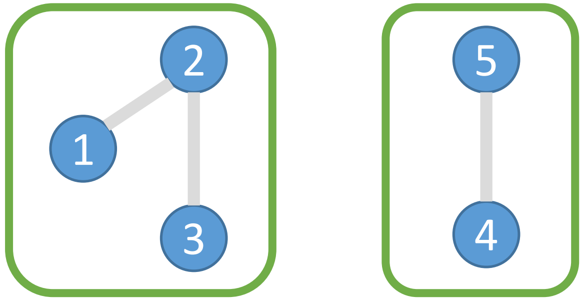


An offline algorithm

For all $v \in V$, let $\boldsymbol{CC(v) = v}$

For $\log n$ iterations

- Pick an incident edge to each connected component
- Merge the components that have an incident edge which we picked

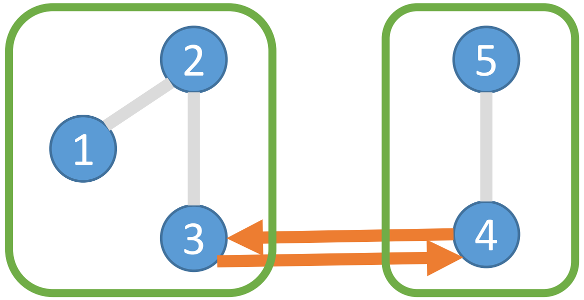


An offline algorithm

For all $v \in V$, let $\boldsymbol{CC(v) = v}$

For $\log n$ iterations

- Pick an incident edge to each connected component
- Merge the components that have an incident edge which we picked

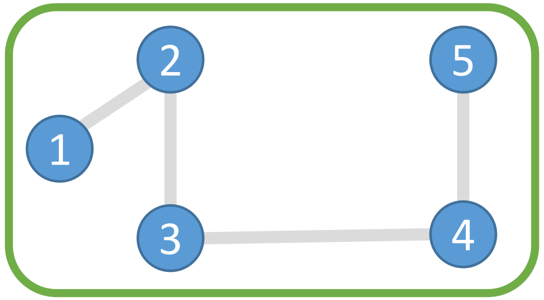


An offline algorithm

For all $v \in V$, let $\boldsymbol{CC(v) = v}$

For $\log n$ iterations

- Pick an incident edge to each connected component
- Merge the components that have an incident edge which we picked



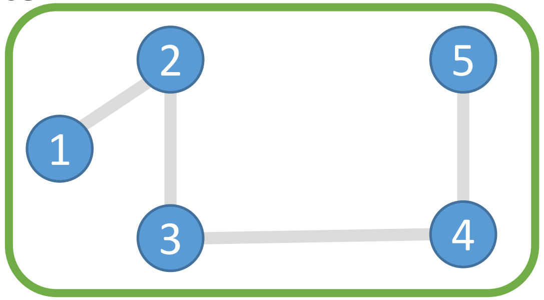
An offline algorithm

For all $v \in V$, let $\boldsymbol{CC(v) = v}$

For $\log n$ iterations

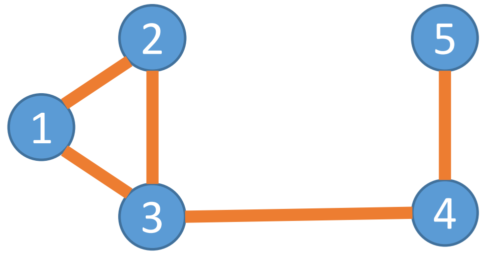
- Pick an incident edge to each connected component
- Merge the components that have an incident edge which we picked

- In the worst case at every iteration every two connected components merge, so it takes $\log n$ iterations at most.



Vector representation

- For each node $v \in V$
  - let $\boldsymbol{x_v} \in \{\boldsymbol{-1, 0, 1}\}^{\binom{n}{2}}$ where
  - for each edge $e = (u, v)$ where $u < v$, we set $x_u(e) = 1$ and $x_v(e) = -1$
  - the rest of the entries are zero.



(1,2)(1,3)(1,4)(1,5)(2,3)(2,4)(2,5)(3,4)(3,5)(4,5)

- $x_1 = (1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)$
- $x_2 = (-1, 0, 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0)$
- $x_{(1,2)} = (0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0)$
- $x_{(4,5)} = (0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0, -1,\ 0,\ 0)$

➢ **Main property:** for a subset of vertices $U \subseteq V$, $\mathbf{Support}(\sum_{v \in U} \boldsymbol{x_v}) = \boldsymbol{E(U, V \setminus U)}$

Sketching Algorithm for Connectivity

Sketch: Pick $\log n$ of L_0 - sampler sketches $\boldsymbol{M_t}$

- Maintain $\boldsymbol{M_t x_v}$ for all $t \le O(\log n)$ and $v \in V$

➢ Total space usage $\boldsymbol{O(n \cdot poly \log n) = \tilde{O}(n)}$

Algorithm (is run after the stream ends):

❑ Initially for all $v \in V$ set $CC(v) = v$

❑ For $\boldsymbol{t = 1}$ $to \log n$, for each remaining component C

- Compute $\boldsymbol{M_t(\sum_{v \in C} x_v) = \sum_{v \in C} M_t x_v}$
- Pick one edge from $\boldsymbol{E(C, V \setminus C)}$ if one exists
- Merge the connected components