CIS 700: “algorithms for Big Data”

Lecture 6: Graph Sketching

Slides at http://grigory.us/big-data-class.html

Grigory Yaroslavtsev

http://grigory.us

Penn
UNIVERSITY of PENNSYLVANIA

Sketching Graphs?

- We know how to sketch vectors: $v \to Mv$
- How about sketching graphs?
- $G(V, E) \equiv A_G$ (adjacency matrix): $A_G \to MA_G$
- Sketch columns of A_G
- $n = |V|, m = |E|$
- $O(poly(\log n))$ sketch per vertex / $\tilde{O}(n)$ total
  - Check connectivity
  - Check bipartiteness
- As always, space rather than dimension. Why?

Graph Streams

- Semi-streaming model: [Muthukrishnan '05; Feigenbaum, Kannan, McGregor, Suri, Zhang'05]
  - Graph defined by the stream of edges $e_1, \dots, e_m$
  - Space $\tilde{O}(n)$, edges processed in order
  - Connectivity is easy on $\tilde{O}(n)$ space for insertion-only
- Dynamic graphs:
  - Stream of insertion/deletion updates $+e_{i_1}, -e_{i_2}, \dots, -e_{i_t}$ (assume sequence is correct)
  - Resulting graph has edge e_i if it wasn't deleted after the last insertion
- Linear sketching dynamic graphs:

$$MA_{G \setminus e} = MA_G - \mathrm{MA_e}$$

Distributed Computing

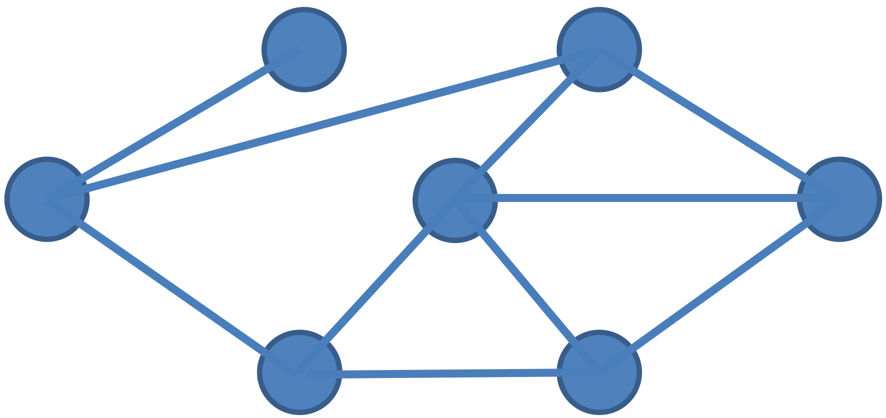
- Linear sketches for distributed processing
- S servers with o(m) memory:
  - Send m/S edges $(E_1, \dots, E_s)$ to each server
  - Compute sketches $ME_1, \dots, ME_s$ locally
  - Send sketches to a central server
  - Compute $MA_G = \sum_i^s ME_i$
- M has to have a small representation (same issue as in streaming)

Connectivity

- Thm. Connectivity is sketchable in $\tilde{O}(n)$ space
- Framework:
  - Take existing connectivity algorithm (Boruvka)
  - Sketch $A_G \to MA_G$
  - Run Boruvka on MA_G
- Important that the sketch is homomorphic w.r.t the algorithm

Part 1: Parallel Connectivity (Boruvka)

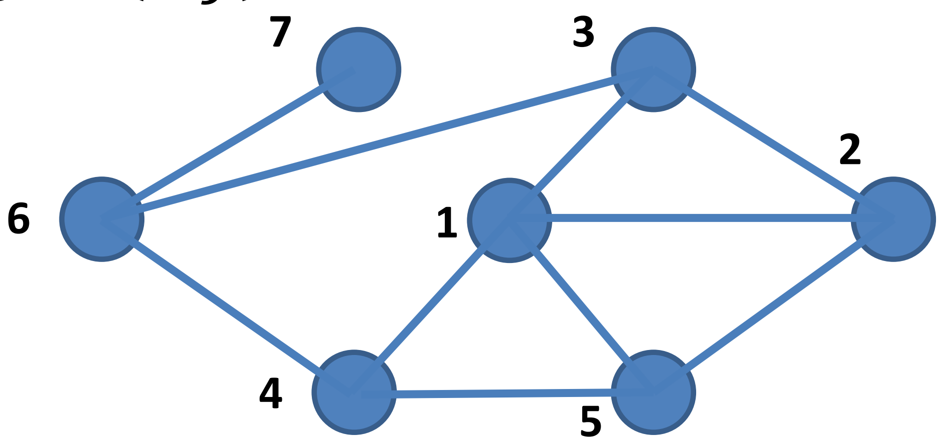
- Repeat until no edges left:
  - For each vertex, select any incident edge
  - Contract selected edges



- Lemma: process converges in $O(\log n)$ steps

Part 2: Graph Representation

- For a vertex i let a_i be a vector in $\mathbb{R}^{\binom{n}{2}}$
- Non-zero entries for edges (i,j)
  - $a_i[i,j] = 1$ if $j > i$
  - $a_i[i,j] = -1$ if j $< i$
- Example:



$a_1 = (\quad 1, 1, 1, 1, 0, \dots, 0)$

$a_2 = (-1, 0, 0, 0, 0, 0, 1, 0, 1, \dots, 0)$

{1,2}, {1,3}, {1,4}, {1,5}, {1,6}, {1,7}, {2,3}, {2.4}. {2,5}, …

- Lem: For any $S \subseteq V$ supp$(\sum_{i \in S} a_i) = E(S, V \setminus S)$

Part 3: L_0-Sampling

- There is a distribution over $M \in \mathbb{R}^{d \times m}$ with $d = O(\log^2 m)$ such w.p. 9/10 that $\forall a \in \mathbb{R}^m$:
$$Ca \rightarrow e \in supp(a)$$

[Cormode, Muthukrishnan, Rozenbaum'05; Jowhari, Saglam, Tardos '11]

- Constant probability suffices – still $O(\log n)$ Boruvka iterations

Final Algorithm

- Construct $\log n$ ℓ_0-samplers for each a_i
- Run Boruvka on sketches:
  - Use $C_1 a_j$ to get an edge incident on a node j
  - For $i = 2$ to t:
    - To get incident edge on a component $S \subseteq V$ use:

$$\sum_{j \in S} C_i a_j = C_i \left(\sum_{j \in S} a_j \right) \rightarrow$$

$$\rightarrow e \in supp \left(\sum_{j \in S} a_j \right) = E(S, V \setminus S)$$

K-Connectivity

- Graph is k-connected is every cut has size $\geq k$
- Thm: There is a $O(nk \log^3 n)$-size linear sketch for k-connectivity
- Generalization: There is an $O(n \log^5 n / \epsilon^2)$-size linear sketch which allows to approximate all cuts in a graph up to error $(1 \pm \epsilon)$

K-connectivity Algorithm

- Algorithm for k-connectivity:
  - Let F_1 be a spanning forest of $G(V,E)$
  - For $i = 2, \dots, k$
    - Let F_i be a spanning forest of $G(V, E \setminus F_1 \setminus \cdots \setminus F_{i-1})$
- Lem: $G(V, F_1 + \cdots + F_k)$ is k-connected iff G(V,E) is.
- $\Rightarrow$ Trivial
- $\Leftarrow$ Consider a cut in $G(V, \sum_{i=1}^{k} F_i)$ of size $< k$

$\Rightarrow \exists i^*$: this cut didn't grow in step i^*

$\Rightarrow$ there is a cut in $G(V,E)$ of size $< k$

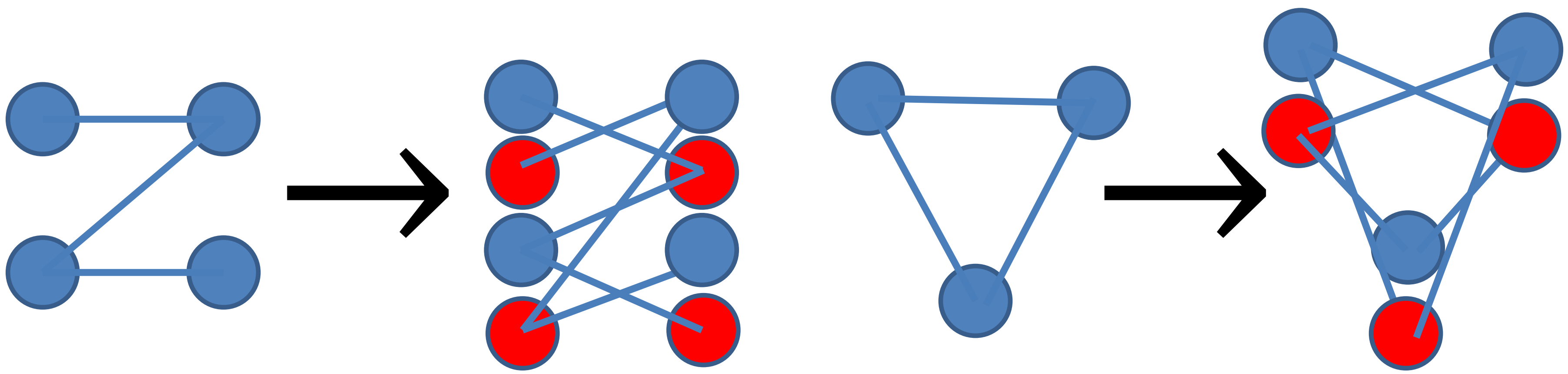
$\Rightarrow$ contradiction

K-connectivity Algorithm

- Construct k independent linear sketches $\{M_1 A_G, M_2 A_G \ldots, M_k A_G\}$ for connectivity
- Run k-connectivity algorithm on sketches:
  - Use $M_1 A_G$ to get a spanning forest F_1 of G
  - Use $M_2 A_G - M_2 A_{F_1} = M_2(A_{G-F_1})$ to find F_2
  - Use $M_3 A_G - M_3 A_{F_1} - M_3 A_{F_2} = M_3(A_{G-F_1-F_2})$ to find F_3
  - ...

Bipartiteness

- Reduction: Given G define G' where vertices $v \to (v_1, v_2)$; edges $(u, v) \to (u_1, v_2)$ & (u_2, v_1)



- Lem: # connected components doubles iff the graph is bipartite.
- Thm: $O(n \log^3 n)$-size linear sketch for k-connectivity (sketch G' (implicitly).)

Minimum Spanning Tree

- If n_i = # connected components in a subgraph induced by edges of weight $\leq (1+\epsilon)^i$:

$$w(MST) \leq n - (1+\epsilon)^r + \sum_{i=0\ldots r-1} \lambda_i n_i \leq (1+\epsilon) w(MST)$$

where $\lambda_i = ((1+\epsilon)^{i+1} - (1+\epsilon)^i$

- cc(G) = #connected components of G
- Round weights up to the nearest power of $1+\epsilon$
- $G_i \equiv$ subgraph with edges of weight $\leq (1+\epsilon)^i$
- Edges taken by the Kruskal's algorithm:
  - n – cc(G_0) edges of weight 1
  - $cc(G_0) - cc(G_1)$ edges of weight $(1+\epsilon)$
  - …
  - cc(G_{i-1}) – cc(G_i) edges of weight $(1+\epsilon)^i$

Minimum Spanning Tree

- Let $r = \log_{1+\epsilon} W$ where W = max edge weight
- Overall weight:

$$n - cc(G_0) + \sum_1^r (1+\epsilon)^i \left(cc(G_{i-1}) - cc(G_i)\right)$$

$$= n - (1+\epsilon)^r + \sum_0^{r-1} \left((1+\epsilon)^{i+1} - (1+\epsilon)^i\right) cc(G_i)$$

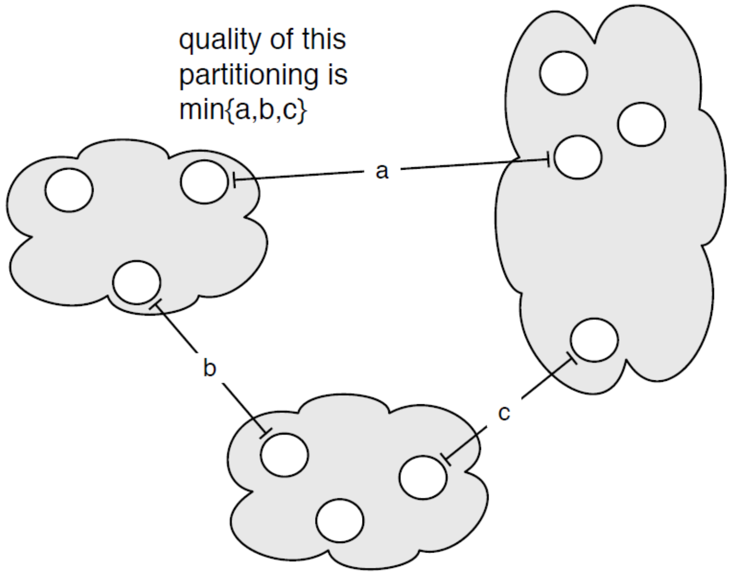
- Thm: $(1+\epsilon)$-approx. MST weight can be computed with $\tilde{O}(n)$ linear sketch for $W = poly(n)$

MST: Single Linkage Clustering

- [Zahn'71] **Clustering** via MST (Single-linkage):

k clusters: remove $k - 1$ longest edges from MST

- Maximizes **minimum** intercluster distance



[Kleinberg, Tardos]

Cut Sparsification

- Two problems:
  - Approximating Min-Cut in the graph (up to $1 \pm \epsilon$)
  - Preserving all cuts in the graph (up to $1 \pm \epsilon$)
- General cut sparsification framework:
  - Sample each edge e with probability p_e
  - Assign sampled edges weights $1/p_e$
- Expected weight of each cut is preserved, but too many cuts – can't take union bound

Cut Sparsification

- For an edge e let λ_e = weight of the minimum cut that contains e
- $\lambda =$ size of the Min-Cut in G
- Thm [Fung et al.]: If G is an undirected weighted graph the if $p_e \geq \min\left(\frac{C \log^2 n}{\lambda_e\, \epsilon^2}, 1\right)$ then the cut sparsification alg. Preserves weights of all cuts up to $(1 \pm \epsilon)$
- Thm [Karger]: $p_e \geq \min\left(\frac{C \log n}{\lambda\, \epsilon^2}, 1\right)$ preserves Min-Cut up to $(1 \pm \epsilon)$

Minimum Cut

Algorithm:

- For $i = \{0,1, \dots, 2 \log n\}$:
  - Let G_i be the subgraph of G where each edge is sampled with probability $1/2^i$
  - Let $H_i = \mathrm{F}_1, \dots, F_k$ where $k = O\left(\frac{1}{\epsilon^2} \cdot \log n\right)$ and F_i are forests constructed by the k-connectivity alg.
- Return $2^j \lambda(H_j)$ where $j = \min\{i : \lambda(H_i) < k\}$

Space: $O\left(\frac{n \log^4 n}{\epsilon^2}\right)$, works for dynamic graph streams

Minimum Cut: Analysis

- Key property: If G_i has $\leq k$ edges across a cut then H_i contains all such edges
- $i^* = \left\lfloor \log \max\left\{1, \frac{\lambda\epsilon^2}{6\log n}\right\} \right\rfloor$
- $i \leq i^* \Rightarrow p_e \geq \min\left(\frac{6\log n}{\lambda\epsilon^2}, 1\right) \Rightarrow$ min cut in G_i is approximating min-cut in G up to $(1 \pm \epsilon)$
- $i = i^*$: By Chernoff bound # edges in G_{i^*} that crosses min-cut in G is $O\left(\frac{1}{\epsilon^2}\log n\right) \leq k$ w.h.p.

Cut Sparsification

Algorithm:

- For $i = \{0,1,\ldots,2\log n\}$:
  - Let G_i be the subgraph of G where each edge is sampled with probability $1/2^i$
  - Let $H_i = \mathrm{F}_1,\ldots,F_k$ where $k = O\left(\frac{1}{\epsilon^2}\cdot\log^2 n\right)$ and F_i are forests constructed by the k-connectivity alg.
- For each edge e let $j_e = \min\{\mathrm{i}: \lambda_e(H_i) < k\}$.
- If $e \in H_{j_e}$ then add e to the sparsifier with weight 2^{j_e}

- Space: $O\left(\frac{n\log^5 n}{\epsilon^2}\right)$, works for dynamic graph streams
- Analysis similar to the Min-Cut using [Fung et al.]