Lecture 4

- Estimating the number of connected components in a graph
- Estimating weight of min spanning tree
- Distributed Algorithms vs. sublinear time algorithms

How many Connected Components in G?

Input $G=(V,E)$, ε adjacency list representation $|V|=n$, $|E|=m \le d \cdot n$
max degree d

Output y s.t. if c = # conn comp of G
then $c - \varepsilon n \le y \le c + \varepsilon n$ ← additive approx to w/in εn

A different way to characterize # conn. comp.:

notation $\forall v$, let n_v = # nodes in v's connected component

Observation $\forall$ connected component $A \subseteq V$

$$\sum_{u \in A} \frac{1}{n_u} = \sum_{u \in A} \frac{1}{|A|} = 1$$

$\Rightarrow$ Characterization: $c = \sum_{u \in V} \frac{1}{n_u}$

Why is this better?
Computing $c = \Omega(n^2)$ time? maybe $\Omega(n)$ if careful?
but we will only estimate c

Estimating $c = \sum_{u \in V} \frac{1}{n_u}$:

Two parts:

1) estimating $\frac{1}{n_u}$ quickly

2) estimating $\sum_u \frac{1}{n_u}$ by sampling u's + estimating the average $\frac{1}{n_u}$ value

Estimating a single $\frac{1}{n_u}$ value:

<u>def</u> let $\hat{n}_u = \min\{n_u, 2/\varepsilon\}$

$$\hat{c} = \sum_{u \in V} \frac{1}{\hat{n}_u}$$

<u>Lemma</u> $\forall u, \quad \left| \frac{1}{\hat{n}_u} - \frac{1}{n_u} \right| \leq \varepsilon/2$

<u>pf</u>:

if $n_u \leq 2/\varepsilon$ then $\hat{n}_u = n_u$ so $\frac{1}{\hat{n}_u} - \frac{1}{n_u} = 0$

if $n_u > 2/\varepsilon$ then $\hat{n}_u = 2/\varepsilon$

so $0 \leq \frac{1}{n_u} < \frac{1}{\hat{n}_u} = \frac{\varepsilon}{2}$

so $\frac{1}{\hat{n}_u} - \frac{1}{n_u} < \frac{\varepsilon}{2}$

∎

Corr. $|c - \hat{c}| \leq \frac{\varepsilon n}{2}$

How long does it take to compute $\hat{n}_u$?

Algorithm compute $\hat{n}_u$

Do BFS starting from u until

- visit whole component of u
- or visit $2/\varepsilon$ distinct nodes

Output # visited nodes.

runtime $O(d/\varepsilon)$ ← time per step of BFS

Estimating sum:

Algorithm estimate c

$r \leftarrow 6/\varepsilon^3$

choose $U = \{u_1, \ldots, u_r\}$ random nodes

$\forall u \in U$,

compute $\hat{n}_u$ via above algorithm

Output $\hat{c} = \frac{n}{r} \sum_{u \in U} \frac{1}{\hat{n}_u}$

runtime $O(d/\varepsilon^4)$

why is it a good estimate?

Thm $\Pr[|\tilde{c} - \hat{c}| \le \varepsilon n/2] \ge 3/4$

Pf.

Chernoff Bnd: $X_1 \cdots X_r$ iid $X_i \in [0,1]$

$S = \sum X_i \qquad p = E[X_i] = E[S]/r$

Then: $\Pr\left[\left|\frac{S}{r} - p\right| \ge \delta p\right] \le e^{-\Omega(rp\delta^2)}$

here, $p = E\left[\frac{1}{\hat{n}_{u_i}}\right]$

$S = \sum_{i=1}^{r} \frac{1}{\hat{n}_u}$

$\delta = \varepsilon/2$

so $\Pr\left[\left|\frac{1}{r}\sum_{i=1}^{r}\frac{1}{\hat{n}_{u_i}} - E\left[\frac{1}{\hat{n}_{u_i}}\right]\right| \ge \frac{\varepsilon}{2} E\left[\frac{1}{\hat{n}_{u_i}}\right]\right] \le e^{-\Omega\left(r \cdot \frac{\varepsilon^2}{4} \cdot E\left[\frac{1}{\hat{n}_u}\right]\right)}$

$\frac{1}{r}\sum_{i=1}^{r}\frac{1}{\hat{n}_{u_i}} = \frac{\tilde{c}}{n}$

$E\left[\frac{1}{\hat{n}_{u_i}}\right] = \frac{1}{n}\cdot\sum_{u \in V}\frac{1}{\hat{n}_u} = \frac{\hat{c}}{n}$

since $\frac{\varepsilon}{2} \le \frac{1}{\hat{n}_u} \le 1$

so $\frac{\varepsilon n}{2} \le \hat{c} \le n$

so can make this a large constant

$-\Omega\left(\frac{1}{\varepsilon}\cdot\frac{\hat{c}}{n}\right)$

so $\Pr\left[\left|\frac{\tilde{c}}{n} - \frac{\hat{c}}{n}\right| \ge \frac{\varepsilon}{2}\frac{\hat{c}}{n}\right]$

$= \Pr[|\tilde{c} - \hat{c}| \ge \frac{\varepsilon}{2}\cdot\hat{c}] \le e$

$\le \frac{1}{4}$ by picking good constant in Ω

∎

Corr $\Pr[|c - \tilde{c}| \le \varepsilon n] \ge 3/4$

Pf. $|c - \tilde{c}| \le |c - \hat{c}| + |\hat{c} - \tilde{c}|$

$|c - \hat{c}| \le \varepsilon n/2$ always by corr

$|\hat{c} - \tilde{c}| \le \varepsilon n/2$ by thm with prob $\ge 3/4$

What is the minimum spanning tree weight?

Input: $G=(V,E)$ adj list representation $n=|V|$

max degree d

each edge has weight $w_{uv} \in \{1,\dots,w\} \cup \{\infty\}$ (i.e. $w_{uv} \notin E$)

G is connected (via edges of weight $\le w$)

Output let $M = \min_{T \text{ spans } G} w(T)$, $w(T) = \sum_{(i,j)\in T} w_{ij}$

output $\hat{M}$ s.t.

$$(1-\varepsilon)M \le \hat{M} \le (1+\varepsilon)M$$

Note by assumption of wts, $n-1 \le w(T) \le w(n-1)$

A different characterization of M:

def $G^{(i)} = (V, E^{(i)})$ where $E^{(i)} = \{(u,v) \mid w_{uv} \in \{1..i\}\}$

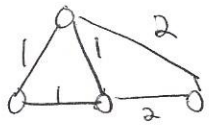
$C^{(i)}$ = # conn comp of $G^{(i)}$

before characterizing, let's do examples:

1) $w=1$ (only one weight + connected by assumption)

then, $M = n-1$

2) $w=2$



G

$G^{(1)}$

$C^{(1)} = 2$

need $= \underbrace{C^{(1)} - 1}$ edges of wt 2 for MST
and this is enough!

so $M = 2(C^{(1)}-1) + 1 \cdot (n-1-(C^{(1)}-1))$

$= n - 2 + C^{(1)}$

In general:

Claim $M = n - w + \sum_{i=1}^{w-1} C^{(i)}$

Pf

let α_i = # edges of wt i in any MST of G

↑ all MST's have same value of α_i!

why?

$$\sum_{i>l} \alpha_i = \#\text{conn comp of } G^{(l)} - 1$$
$$= C^{(l)} - 1$$

where $C^{(0)} = n$

$$M = \sum_{i=1}^{w} i \cdot \alpha_i$$

$$= \sum_{i=1}^{w} \alpha_i + \sum_{i=2}^{w} \alpha_i + \sum_{i=3}^{w} \alpha_i + \dots + \sum_{i=w}^{w} \alpha_w$$

$$= \alpha_w$$

$$= (n-1) + (C^{(1)} - 1) + (C^{(2)} - 1) + \dots + (C^{(w-1)} - 1)$$

$$= n - w + \sum_{i=1}^{w-1} C^{(i)}$$

□

Approximation Algorithm:

For $i = 1$ to $w-1$

$\hat{C}^{(i)}$ = approx # c.c. of $G^{(i)}$ to within $\frac{\varepsilon}{2w}$ (additive error)

Output $\hat{M} = n - w + \sum_{i=1}^{w-1} \hat{C}^{(i)}$

Runtime:

$O(d/(\varepsilon')^4) = O(dw^4/\varepsilon^4)$ for each call to approximate # c.c.

↑ we used $\varepsilon' = \frac{\varepsilon}{2w}$

↑ how do you recompute $G^{(i)}$? ignore edges of wt $> i$

Total: $O(dw^5/\varepsilon^4)$

↗ can improve to $O(\frac{dw}{\varepsilon^2}\log\frac{dw}{\varepsilon})$

+ need $\Omega(\frac{dw}{\varepsilon^2})$

Approximation guarantee:

how? ↓ HW0!

Call approx # cc with error probability $\leq \frac{1}{4\cdot w}$

↑ ie. prob of approx error $> \varepsilon'$

Pr[all calls to approx # cc give output that is ε' additive approx] $\geq 1 - \frac{w}{4w} = 3/4$

If ↑ happens,

$|M - \hat{M}| \leq n \cdot w \cdot \frac{\varepsilon}{2w} = \frac{\varepsilon n}{2}$ ← additive error small

since $M \geq n-1 \geq \frac{n}{2}$ (for $n \geq 2$) ← this is where we use 1.6. on edge wts

$|M - \hat{M}| \leq \varepsilon M$ ← also multiplicative approx error is small

Distributed Algorithms vs. sublinear time algorithms on SPARSE graphs

max deg $\leq d$

Again, Sparse graphs: max degree d
adj list representation

A problem to solve:

Vertex Cover

$V' \subseteq V$ is "Vertex Cover" (VC) if $\forall (u,v) \in E$

either $u \in V'$ or $v \in V'$

VC Question: What is min size of VC?

Note: in deg $\leq d$ graph, $|VC| \geq \frac{m}{d}$ since each node can cover $\leq d$ edges

(VC is NP-complete, but there is a polytime 2-multiplicative approximation)

Can you approximate V.C. in sublinear time?

multiplicative? no! graph with no edges $|VC| = 0$
graph with 1 edge $|VC| = 1$
$\Rightarrow$ can't distinguish these cases in sublinear time but must answer 0 in first case & >0 in second.

additive? hard! need some mult error
computationally hard to approx to better than 1.36 factor (maybe even 2)

Combination?

def. $\hat{y}$ is (α, ε)-estimate of soln value y for minimization problem if

$$y \leq \hat{y} \leq \alpha y + \varepsilon$$

allows mult + additive error

(analogous defn for maximization problems)

Some Background on distributed Algorithms

- Network
  - processors
  - links

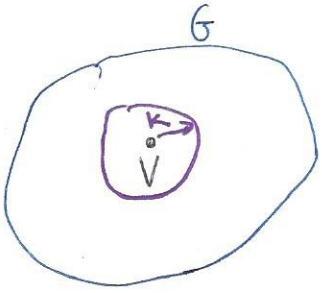
  ⇝ max degree d known to all

- Communication round
  - nodes send messages to neighbors

def. Vertex Cover problem for distributed networks:
(not some other graph)
- Network graph = input graph (i.e. network computes on itself)
- at end, each node knows if in or out of VC

Main insight on why fast distributed ⇔ sub linear time:

in k round algorithm, output of node v only depends on nodes at distance at most k from v. At most d^k of these!



$\Rightarrow$ can simulate V's view of distributed computation in $\leq d^k$ time & figure out if v in or out of VC

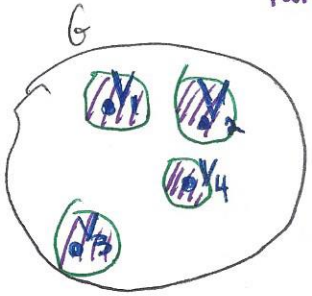
Comment: if algorithm is randomized, v needs to know random bits (or be able to construct) of all d^k nbrs. ← must be consistent

But are there fast VC distributed algorithms?

YES, will see some soon ↑ often called "local distributed algorithms"

How do you use this to approximate VC in sublinear time?

Parnas-Ron framework:



Sample nodes of graph $V_1 \dots V_r$

for each V_i, simulate distributed algorithm to see if $V_i \in VC$

Output $\frac{\#V_i\text{'s in VC}}{r} \cdot n$

Runtime $O(r \cdot d^{k+1}) \approx O(\frac{1}{\varepsilon^2} \cdot d^{k+1})$

Proof of correctness Chernoff/Hoeffding bnds

fast distributed algorithm for VC:

$i \leftarrow 1$

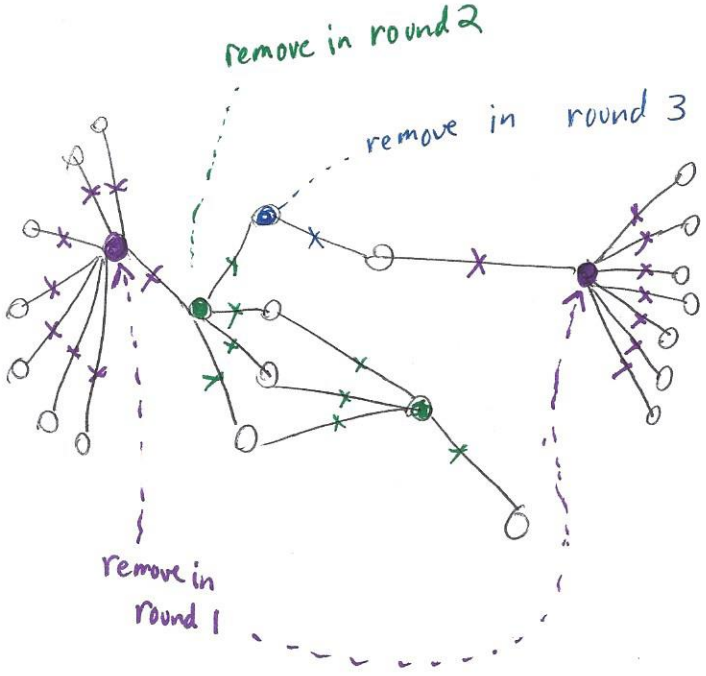
While edges remain:

- remove vertices of degree $> d/2^i$ + adjacent edges
- update degrees of remaining nodes
- increment i

Output all removed nodes as VC

#rounds: $\log d$

example:



is it a VC?

no edges remain at end

all removed along with some adjacent vertex

Is it a good approximation?

Thm let $VC(G)$ = size of min VC of G

Then, $VC(G) \leq$ output $\leq (2 \log d + 1)\, VC(G)$

↑ since output is VC

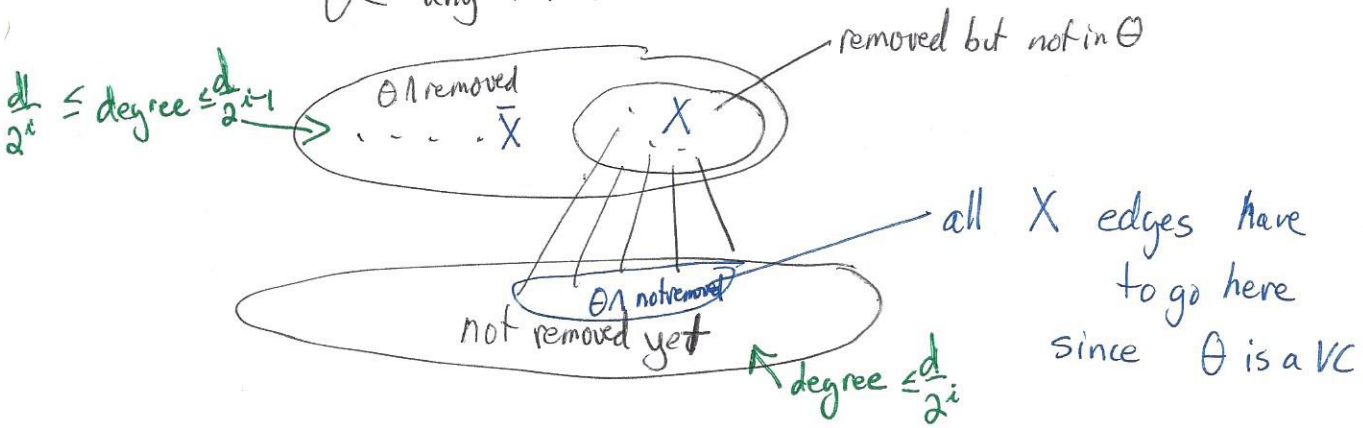
↑ to prove

Proof

Claim: in each iteration, add $\leq 2\, VC_G$ new vertices

why: all nodes removed have deg bet $\frac{d}{2^{i+1}}$ + $\frac{d}{2^i}$

$\theta \leftarrow$ any min VC



so $|X| \cdot \frac{d}{2^i} \leq |\theta| \cdot \frac{d}{2^{i-1}}$

so $|X| \leq 2|\theta|$