Lecture 4:

Distributed Algorithms

vs.

Sublinear time Algorithms:

the case of vertex cover

Given: "sparse" graph — max degree Δ, adjacency list representation

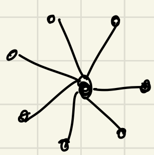
Vertex Cover

$V' \subseteq V$ is a "vertex cover" (VC)

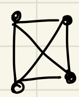
if $\forall (u,v) \in E$

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

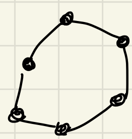
What is min size of VC?



star



k-clique



n-cycle (n even)

Degree $\leq \Delta$ graphs:

$|VC| \geq \frac{m}{\Delta}$ since each node can cover $\leq \Delta$ edges

Complexity of V.C.:

- NP-complete to solve exactly
- poly time to get 2-approx
- sublinear time multiplicative approx?
  - graph with no edges $|VC|=0$
    mult approx must return 0
  - graph with 1 edge $|VC|=1$
    mult approx must return >0

  need $\Omega(n)$ queries to distinguish

- sublinear time additive approx?
  hard! need mult error
  computationally hard to est to better
  than 1.36 mult (maybe even 2)

- Combination?

Additive + Multiplicative approx error:

<u>def</u> $\hat{y}$ is (α, ε)-approximation of soln value y for a minimization problem if

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

mult error

additive error

(analogous defn for maximization problems)

Some background on distributed algorithms:

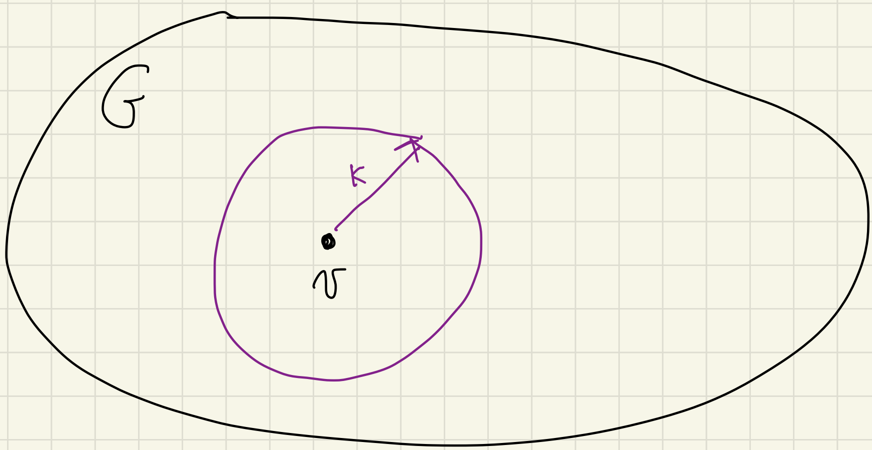
- Network
  - processes ⇐ max deg Δ
  - links

- Communication round:
  - nodes perform computation on
    - input bits
    - random coins
    - node ID
    - history of received messages
  - nodes send msgs to neighbors
  - nodes receive msgs from neighbors

- def Vertex Cover for distributed network:
  - network graph $\equiv$ input graph (network computes on ITSELF)
  - goal: at end, each node knows if it is in or out of VC (doesn't necessarily know anything else)

Main insight on why fast distributed algorithms
$\Downarrow$
sublinear time

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



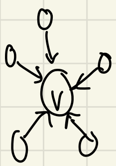
- Can sequentially simulate v's view of distributed computation with $\leq d^k$ queries to input, & figure out if v is in or out of V.C.

Simulating v's view of k-round distributed computation:

round 1:

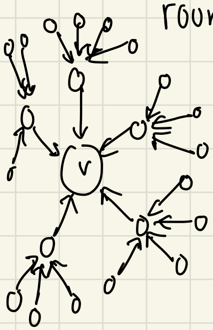
- each node sends msg based on input & random bits
- each node gets msg from each nbr which is based on nbr's input, random bit

round 2:



- each node sends msg based on input, random bits, + msgs from $\leq \Delta$ nbrs
- each node gets msg from each nbr based on their info from round 1

round 3:



- each node sends msg based on input, random bits, + msgs from $\leq \Delta$ nbrs in first 2 rounds
- each node gets msg from each nbr based on their info from rounds 1 + 2

$\therefore$ fast distributed alg $\Rightarrow$ oracle which tells you if v is in VC

How do you use this to approx V.C. in sublinear time?

Parnas-Ron framework:

Sample nodes $v_1 \dots v_r$

for each v_i,

simulate distributed algorithm to see if $v_i \in VC$.

Output $\frac{\#\, v_i\text{'s in } VC}{r} \cdot n$ } gives $\varepsilon \cdot n$ additive approx of V.C.

$\Rightarrow$ c-multiplicative approx of V.C.

Query Complexity:

$$O(r \cdot \Delta^{k+1}) \approx O\left(\frac{1}{\varepsilon^2} \cdot \Delta^{k+1}\right)$$

k = # rounds of dist alg

Δ = max degree of distributed network

Approximation guarantee?

Chernoff / Hoeffding bnds

BUT: Are there fast distributed algorithms for V.C.?

YES!

$i \leftarrow 1$

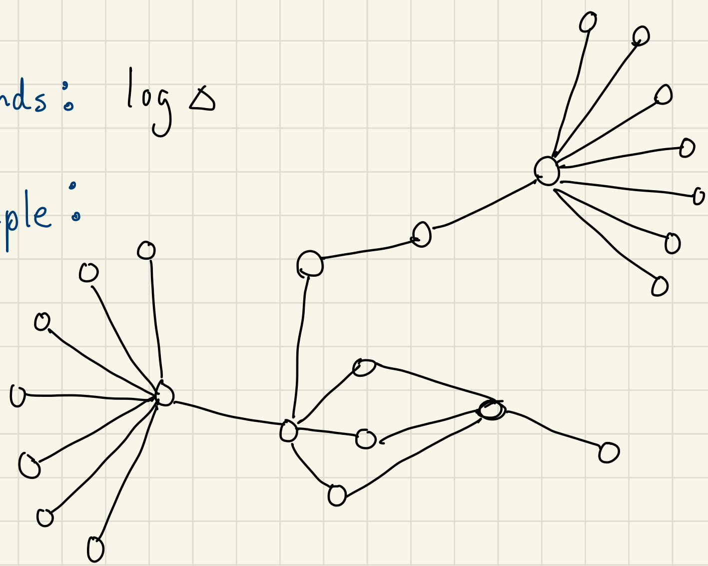
while edges remain:

- remove nodes of $\underbrace{\deg \geq \frac{\Delta}{2^i}}_{\text{put these in V.C.}}$ + $\underbrace{\text{adjacent edges}}_{\text{already covered}}$
- update degrees of remaining nodes
- increment i

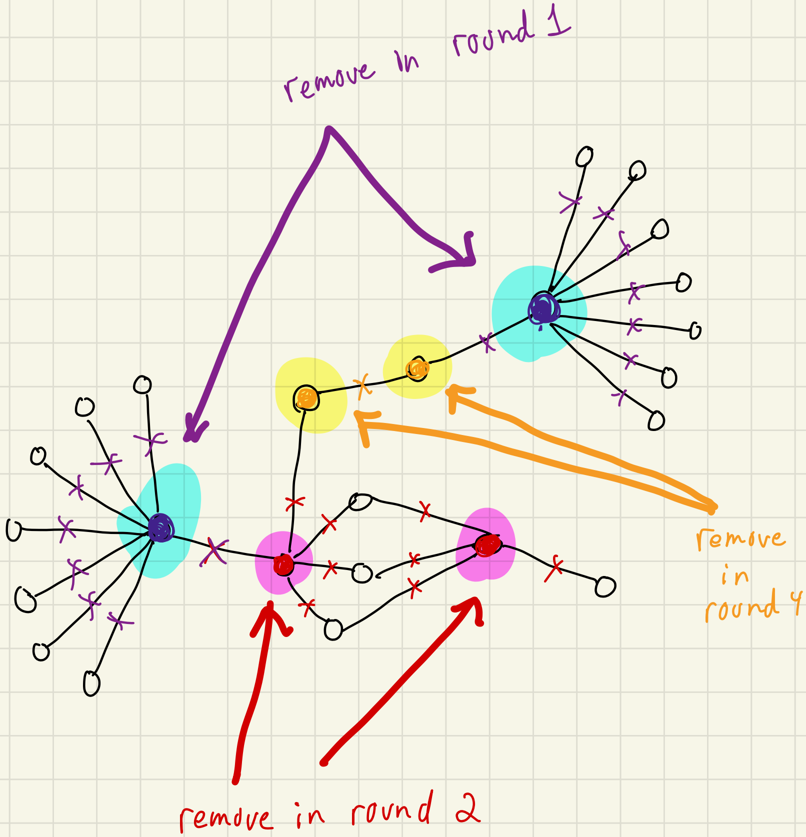
Output all removed nodes as V.C.

rounds: $\log \Delta$

example:



$\Delta = 16$



remove nothing in round 3

$i \leftarrow 1$

while edges remain:

- remove nodes $\underbrace{\text{of deg} \geq \frac{\Delta}{2^i}}_{\text{put these in V.C.}}$ + $\underbrace{\text{adjacent edges}}_{\text{already covered}}$
- update degrees of remaining nodes
- increment i

Output all removed nodes as V.C.

Is it a V.C.?

no edges remain at end

all removed along with adjacent node

Is it a good approximation?

Let optimal Θ be any min V.C. of G

Thm. $|\Theta| \leq \text{output} \leq (2 \log \Delta + 1) |\Theta|$

since output is V.C. + Θ is min

to prove!

Pf.

$i \leftarrow 1$

while edges remain:

- remove $\underbrace{\text{nodes of deg} \geq \frac{\Delta}{2^i}}_{\text{put these in V.C.}}$ & $\underbrace{\text{adjacent edges}}_{\text{already covered}}$
- update degrees of remaining nodes
- increment i

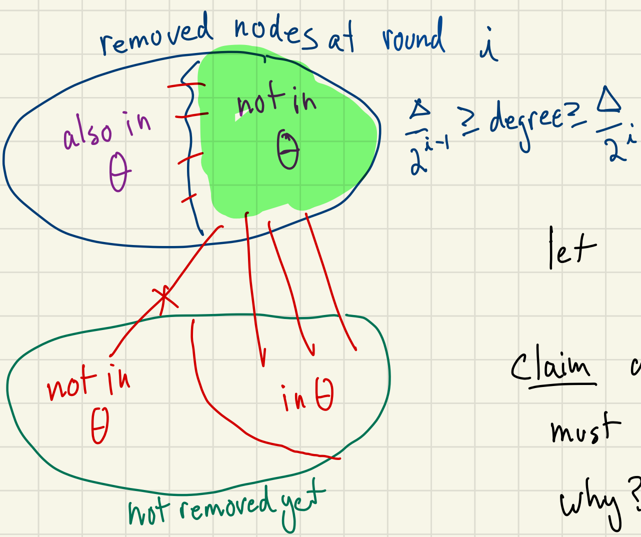
Output all removed nodes as V.C.

Claim each round/iteration adds $\leq 2|\theta|$ new nodes to output V.C.

Why? Observation: at i^{th} round

(1) all nodes remaining in graph have degree $\leq \frac{\Delta}{2^{i-1}}$ } others removed earlier

(2) all removed nodes have degree $\geq \frac{\Delta}{2^i}$ } algorithm design



let X = removed at iteration i but not in θ

Claim all edges touching X must touch θ at other end

why? since θ is V.C.

edges touching X:

$$\geq \frac{\Delta}{2^i} \cdot |X| \qquad \left(\text{since } \deg \geq \frac{\Delta}{2^i}\right)$$

$$\leq \frac{\Delta}{2^{i-1}} |\theta| \qquad \left(\text{since each edge has \underline{other} endpt in } \theta, \text{ \& all nodes have } \deg \leq \frac{\Delta}{2^{i-1}}\right)$$

$$\Rightarrow \quad \frac{\cancel{\Delta}}{2^{\cancel{i}}} \cdot |X| \leq \frac{\cancel{\Delta}}{\cancel{2^{i-1}}} \cdot |\theta|$$

$$|X| \leq 2 \cdot |\theta| \quad \blacksquare$$

Round

$i \leftarrow 1$

while edges remain:

- remove nodes of $\underbrace{\text{deg} \geq \frac{\Delta}{2^i}}_{\text{put these in V.C.}}$ + $\underbrace{\text{adjacent edges}}_{\text{already covered}}$
- update degrees of remaining nodes
- increment i

Output all removed nodes as V.C.

Claim each round adds $\leq 2|\Theta|$ new nodes (not in Θ) to output V.C.

Since $\leq \log \Delta$ rounds,

$$\text{output} \leq |\Theta| + 2|\Theta| \cdot \log \Delta$$
$$= (1 + 2\log \Delta) \cdot |\Theta| \qquad \blacksquare$$

Gives $(O(\log \Delta), \varepsilon)$ approx in $\Delta^{O(\log \Delta)}$ queries

Can do better...