Lecture 4:

Distributed Algorithms

vs.

Sublinear time Algorithms:

the case of vertex cover

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

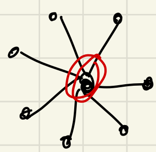
Vertex Cover

$V' \subseteq V$ 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?

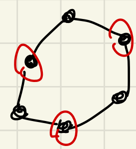


star

$|VC| = 1$

k-clique

$|VC| = k-1$



n-cycle
(n even)

$|VC| = n/2$

Degree $\leq \Delta$ graphs: Can we get a better bound?

$$|VC| \geq \frac{m}{\Delta}$$

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

  distinguishing requires $\Omega(n)$ queries

- sublinear time additive approx?

  hard!

  Computationally hard to estimate to better than 1.36 (maybe even 2)

  $\Rightarrow$ additive even harder

  additive approx $\Rightarrow$ super good mult approx

- Combination?

Additive & Multiplicative approx error:

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

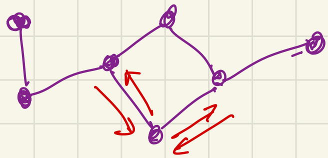
$$y \le \hat{y} \le \underbrace{\alpha}_{\text{mult error}} y + \underbrace{\varepsilon}_{\text{additive error}}$$

(analogous defn for maximization problems)

Some background on distributed algorithms:
local

(LOCAL model)

- Network
  - processes
  - links



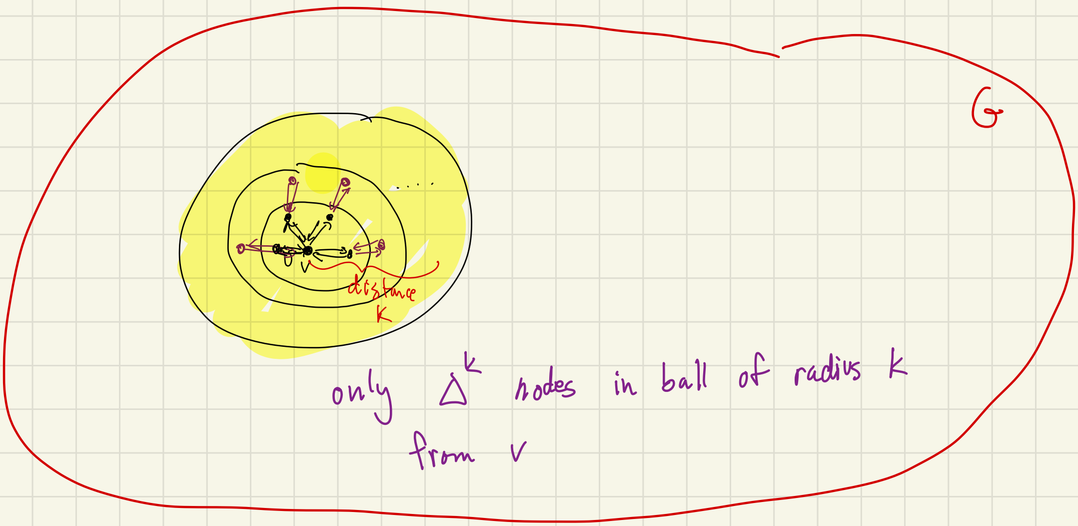
- Communication round:
  - nodes perform computation on
    - input bits
    - random coins
    - node ID
    - history of received messages
  - nodes send msgs to neighbors
  - nodes receive msgs from neighbors

- def Vertex Cover for distributed network:
  - network graph ≡ input graph
  - goal: at end, each node knows if it is in or out of VC

    [don't need to know about other nodes]

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.



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

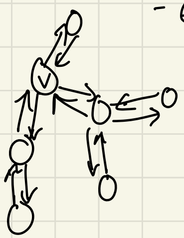
Simulating v's view of k-round distributed computation:

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 Δ nbrs in rnd 1



- each node gets msg from each nbr which is based on nbr's input, random bit + their rcvd. msgs from round 1

⋮

$\therefore$ fast distributed algorithm, we can simulate & get oracle which tells us if v is in V.C.

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

Query Complexity:

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

k = # rounds of dist alg for V.C.

Δ = max degree of network (input graph)

Approximation guarantee?

same approx error of distributed alg

+

Chernoff/Hoeffding bounds $\Rightarrow$ εn additive error

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

YES!!

Here is one (not the best but simple) [Parnas & Ron]

$i \leftarrow 1$ (i = round / iteration #)

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.

rounds: $\log \Delta$

$i \leftarrow 1$

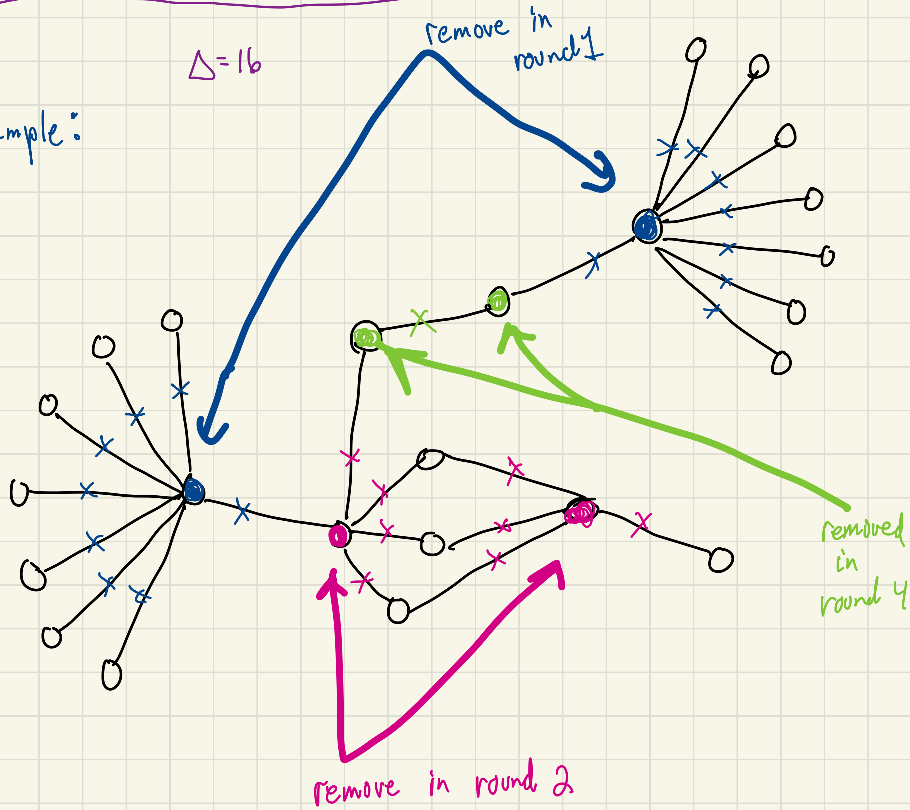
while edges remain:

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

Output all removed nodes as V.C.

$\Delta = 16$

example:



nothing removed in round 3

(blue, pink, green) Removed nodes placed in V.C.

(clear) other nodes are <u>not</u> in output V.C.

$i \leftarrow 1$

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.

Is it a V.C.?

no edges remain at end

all edges were removed when adjacent node was put into V.C.

Is it a good approximation?

Let optimal Θ be any min V.C. of G

Thm $\underbrace{|\Theta| \leq}_{\text{because } \Theta \text{ is min \& a V.C.}}$ output $\underbrace{\leq}_{\text{to prove}}$ $(2 \log \Delta + 1) \cdot |\Theta|$

Pf.

$i \leftarrow 1$

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

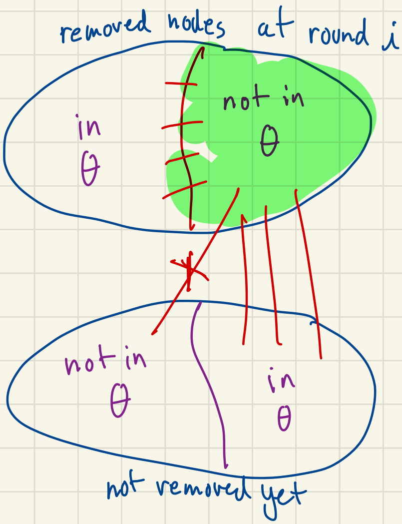
Output all removed nodes as V.C.

Claim each round/iteration adds $\leq 2|\Theta|$ new nodes to output V.C. that are not in Θ

why? observation: at i^{th} round

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

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



for removed nodes:

$$\frac{\Delta}{2^{i-1}} \geq \text{degree} \geq \frac{\Delta}{2^i}$$

Let X = removed nodes at iteration i but not in Θ

Claim all edges touching X must touch Θ at other end

why? because Θ is V.C.

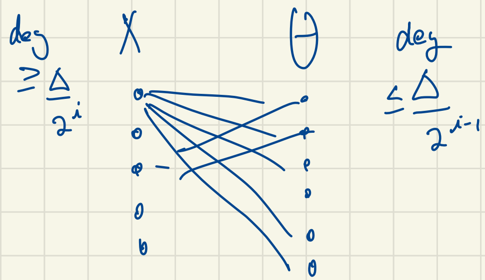
edges touching X:

$\geq \frac{\Delta}{2^i} \cdot |X|$ since deg of any node in X $\geq \frac{\Delta}{2^i}$

$\leq \frac{\Delta}{2^{i-1}} \cdot |\Theta|$ since each edge has other endpt in Θ & all nodes have degree $\leq \frac{\Delta}{2^{i-1}}$

$$\Rightarrow \cancel{\frac{\Delta}{2^{i-1}}} |\Theta| \geq \cancel{\frac{\Delta}{2^i}} |X|$$

$$|X| \leq 2 \cdot |\Theta| \qquad \blacksquare$$



<u>idea</u>

lots of nodes in X $\Rightarrow$ lots of edges in X $\Rightarrow$ (since each node in Θ can't handle too many edges) lots of nodes in Θ.

but Θ isn't that big, so X can't be too big either.

Round

$i \leftarrow 1$

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.

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

since $\leq \log \Delta$ rounds

$$\text{output} \leq |\Theta| + 2|\Theta| \cdot \log \Delta$$

$$= (1 + 2\log\Delta) \cdot |\Theta|$$

size of V.C. that is output

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

no dependence on n

just Δ, ε

Can do better ...