Lecture 4:

Distributed Algorithms vs. Sublinear time Algorithms: the case of vertex cover
Given: "sparse" graph \( \text{max degree } \Delta \) 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?

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 $|V| = 0$
    - mult approx must return 0
  - graph with 1 edge $|V| = 1$
    - mult approx must return $> 0$

- 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:

\[ \hat{y} \text{ is } (\alpha, \varepsilon) \text{- approximation of soln value } y \text{ for a minimization problem if } y \leq \hat{y} \leq \alpha y + \varepsilon \]

(Analogous defn for maximization problems)
Some background on distributed algorithms:

- Network
  - processes \( \leq \max \deg \Delta \)
  - 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 (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 run in 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 $V$.
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, $\Delta$ msg from $\leq \Delta$ nbars.
- 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, $\Delta$ msg from $\leq \Delta$ nbars in first 2 rounds.
- Each node gets msg from each nbr based on their info from rounds 1 & 2.

Fast distributed alg $\Rightarrow$ oracle which tells you if $v$ is in $V$. 
How do you use this to approx V.C. in sublinear time?

**Parnas-Ron Framework:**

Sample nodes $V_1 \ldots V_r$

For each $V_i$,

Simulate distributed algorithm to see if $V_i \in V.C.$

Output $\frac{\#V_i's \in V.C.}{r}$

$\exists$ gives $\epsilon \cdot n$ additive approx of $V.C.$

$f \cdot (\epsilon, \delta)$ gives $\epsilon \cdot n$ additive approx of $V.C.$

$\Rightarrow \epsilon \cdot n$ multiplicative approx of $V.C.$

**Query Complexity:**

$O(r \cdot \Delta^{kn}) \approx O(\frac{1}{\epsilon^2} \cdot \Delta^{kn})$

**Approximation guarantee?**

Chernoff/Hoeffding bounds

$k = 4$ rounds of dist alg

$\Delta = \max$ degree of distributed network
But: Are there fast distributed algorithms for V.C.?

YES!

\[ i = 1 \]

While edges remain:
- remove nodes of \( \text{deg} \geq \frac{\Delta}{2^i} \) \& adjacent edges
- update degrees of remaining nodes
- increment \( i \)

Output all removed nodes as V.C.

\# rounds: \( \log \Delta \)

Example:
$\Delta = 16$

- Remove in round 1
- Remove nothing in round 3
\( i = 1 \)

while edges remain:

• remove nodes of deg \( \geq \Delta / \alpha \) and adjacent edges
  
  \( \text{put these in V.c.} \)

• 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 \( \text{optimal } \Theta \) be any min V.C. of \( G \)

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

\( \uparrow \) since output is V.C. +

\( \Theta \) is min

\( \uparrow \) to prove!
\textbf{Proof.}\ 

While edges remain:
- remove nodes of \( \text{deg} \geq \frac{\Delta}{2^i} \) and adjacent edges
  - put these in V.C.
- update degrees of remaining nodes
- increment \( i \)

Output all removed nodes as V.C.,

\textbf{Claim}\ each round/iteration adds \( \leq 2101 \) new nodes to output V.C.

\textbf{Why?}\ Observation: at \( i \text{th} \) round

1. all nodes remaining in graph have degree \( \leq \frac{\Delta}{2^{i-1}} \)
2. all removed nodes have degree \( \geq \frac{\Delta}{2^i} \) \( \geq \) algorithm design

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

\[ \text{let } x = \text{removed at iteration } i \text{ but not in } \Theta \]

Claim all edges touching \( x \) must touch \( \Theta \) at other end

\textbf{Why?}\ since \( \Theta \) is V.C.
# edges touching X:

\[ \geq \frac{\Delta}{2^i} \cdot |X| \]  

(since \( \deg \geq \frac{\Delta}{2^i} \))

\[ \leq \frac{\Delta}{2^{i-1}} \cdot |\Theta| \]  

(since each edge has other endpt in \( \Theta \), & all nodes have \( \deg \leq \frac{\Delta}{2^{i-1}} \))

\[ \Rightarrow \quad \frac{\Delta}{2^i} |X| \leq \frac{\Delta}{2^{i-1}} \cdot |\Theta| \]

\[ |X| \leq 2 \cdot |\Theta| \]
Round

\( i \leq 1 \)

while edges remain:

- remove nodes of \( \text{deg} \geq \frac{\Delta}{2^i} + \text{adjacent edges} \)
  - put these in V.C.

- update degrees of remaining nodes
  - already covered

- increment \( i \)

Output all removed nodes as V.C.

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

Since \( \leq \log \Delta \) rounds,

\[
\text{output} = |\Theta| + 2|\Theta| \cdot \log \Delta \\
= (1 + 2 \log \Delta) \cdot |\Theta|
\]

Gives \( (O(\log \Delta), \varepsilon) \) approx in \( \Delta \) queries

Can do better...