Lecture 4:

Distributed Algorithms

vs.

Sublinear time Algorithms:

the case of vertex cover
Given: "sparse" graph 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?

$\text{star}$
$\|V_c\| = 1$

$\text{k-clique}$
$\|V_c\| = k - 1$

$\text{n-cycle (n even)}$
$\|V_c\| = \frac{n}{2}$

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

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

  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 \( \Rightarrow \) super good mult approx

- Combination?
Additive + Multiplicative approx error:

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

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

- Network
  - processes
  - links

- Communication round:
  - nodes perform computation on
    - input bits
    - random coins
    - node IDs
    - 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 run 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 $\leq d^k$ queries to input, and figure out if $v$ is in or out of $V$.

Only $\Delta^k$ nodes in ball of radius $k$ from $v$. 

Diagram: 
- $G$ as a graph with nodes and edges illustrating the concept of distance $k$ from $v$. 
  - Nodes connected by edges showing dependencies. 
  - A set of nodes marked as within the ball of radius $k$ from $v$. 
  - Markers indicating $\Delta^k$ nodes in the ball.
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
- Each node gets msg from each nbr which is based on nbr's input, random bit & their round msgs from round 1

*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 $V_1 \ldots V_r$

for each $V_i$, simulate distributed algorithm to see if $V_i \in V.C.$

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

Query Complexity:

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

Approximation guarantee?

Same approx error of distributed alg +
Chernoff/Hoeffding bounds $\Rightarrow \varepsilon n$ additive error
But: Are there fast distributed algorithms for V.C.?

YES!!

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

\[ i = \text{round(iteration #)} \]

\( i \leftarrow 1 \)

While edges remain:

- remove nodes of deg \( \geq \frac{\Delta}{2^i} \) + adjacent edges

- update degrees of remaining nodes

- increment \( i \)

Output all removed nodes as V.C.

# rounds: \( \log \Delta \)
\[ i = 1 \]

While edges remain:

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

Output all removed nodes as \( \text{V.C.} \).

Example:

\[ \Delta = 16 \]

Removed nodes placed in \( \text{V.C.} \),

(\( \text{blue}, \text{purple}, \text{green} \)) Removed nodes placed in \( \text{V.C.} \),

(\( \text{clear} \)) other nodes are not in output \( \text{V.C.} \).
\( i = 1 \)

\begin{itemize}
  \item while edges remain:
    \begin{itemize}
      \item remove nodes of degree \( \Delta \) + adjacent edges
        \begin{itemize}
          \item put these in \( V.C. \)
        \end{itemize}
      \item update degrees of remaining nodes
      \item increment \( i \)
    \end{itemize}

Output all removed nodes as \( V.C. \).
\end{itemize}

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 \( \Theta \) be any min \( V.C. \) of \( G \)

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

\( \uparrow \)
- because \( \Theta \) is min \( + V.C. \)
\( \uparrow \)
- to prove
Theorem

Let $\gamma = \theta + 1 \log_2 \theta$. Then, $\gamma$-edge coloring for $\theta$-edge coloring can be approximated in $O(\theta \log \theta)$ rounds.

Proof

1. Set $\gamma = \theta + 1 \log_2 \theta$ and $\epsilon = 1/\log_2 \theta$.
2. Let $\theta' = \theta - \epsilon$.
3. Run $L$-edge coloring algorithm on graph $\theta'$.
4. Let $\theta'' = \theta' + \epsilon$.
5. Run $\gamma$-edge coloring algorithm on graph $\theta''$.

Correctness

Let $\Gamma^*$ be an optimal $(\gamma + \epsilon)$-coloring. The algorithm produces a $(\gamma + \epsilon)$-coloring $\Gamma$. We define $\Gamma'_0 = \Gamma$ and $\Gamma'_i = \Gamma_i$ for $i \geq 1$. Then, $\Gamma'_i$ is a $(\gamma + \epsilon)$-coloring that differs from $\Gamma^*$ in at most $\epsilon \log_2 \theta$ colors.

Analysis

Let $\theta' = \theta - \epsilon$. The algorithm produces a $(\gamma + \epsilon)$-coloring $\Gamma'$ of the graph $\theta'$. We define $\Gamma'_0 = \Gamma'$ and $\Gamma'_i = \Gamma_i$ for $i \geq 1$. Then, $\Gamma'_i$ is a $(\gamma + \epsilon)$-coloring that differs from $\Gamma^*$ in at most $\epsilon \log_2 \theta$ colors.

Running time

The algorithm runs in $O(\theta \log \theta)$ rounds.
# edges touching $X$:

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

since deg of any node in $X$ is at least $\frac{\Delta}{2^i}$.

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

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

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

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

\[\square\]

\[\text{idea}\]

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

lots of nodes in $\Theta$.

but $\Theta$ isn't that big, so $X$ can't be too big either.
Round

\[ i \leq 1 \]

while edges remain:

• remove nodes of \( \text{deg} \geq \frac{\Delta}{2^i} + \text{adjacent edges} \)

• put these in \( V.C. \) 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 \( \Theta \)) to output \( V.C. \)

since \( \leq \log \Delta \) rounds

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 \( O(\log \Delta) \) queries

Can do better...