2017

Distributed Algorithms vs. sublinear time algorithms on SPARSE graphs

$\uparrow$ 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 poly time 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

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

allows mult + additive error

(analogous defn for maximization problems)

Some Backround on distributed Algorithms

- Network
  - processors
  - links

  } max degree d Known to all

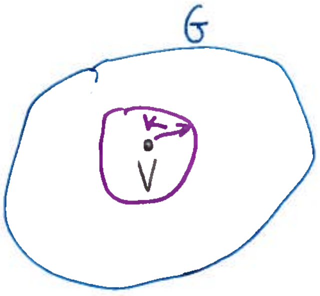
- Communication round
  - nodes send messages to neighbors

def. Vertex Cover problem for distributed networks:

- Network graph = input graph (i.e. network computes on itself) ↙ not some other graph
- at end, each node Knows if in or out of VC (doesn't know about others necessarily)

Main insight on why fast distributed ⟺ sublinear 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 sequentially 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

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

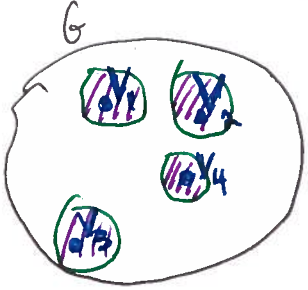
But are there fast VC distributed algorithms?

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 $v_1 \ldots v_r$

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

Output $\dfrac{\# v_i\text{'s in VC}}{r}$ on

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

Proof of correctness Chernoff/Hoeffding bnds

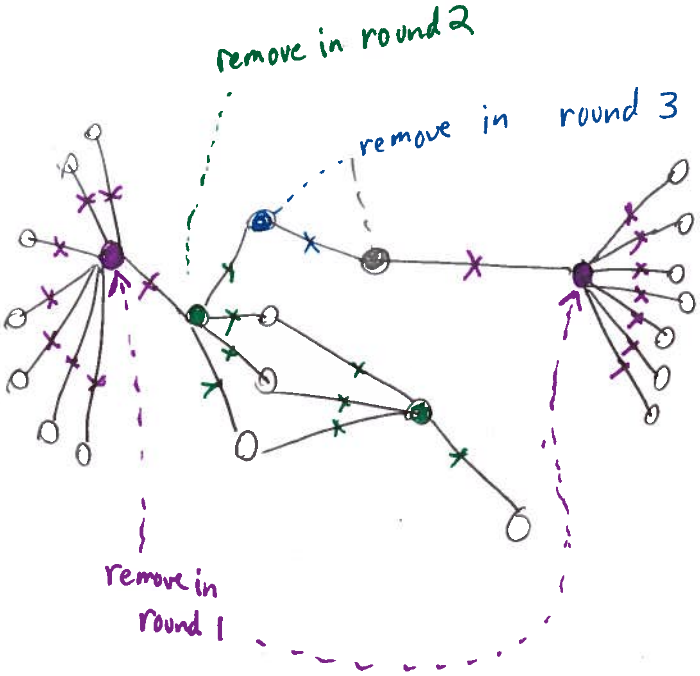
fast distributed algorithm for VC:

```
$i \leftarrow 1$
While edges remain:
    • remove vertices of degree $\geq d/2^i$ + adjacent edges
    • update degrees of remaining nodes
    • increment $i$
Output all removed nodes as VC
```

put these in vertex cover

#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?

<u>Thm</u> let $VC(G)$ = size of min VC of G

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

↑ since output is VC

↑ to prove

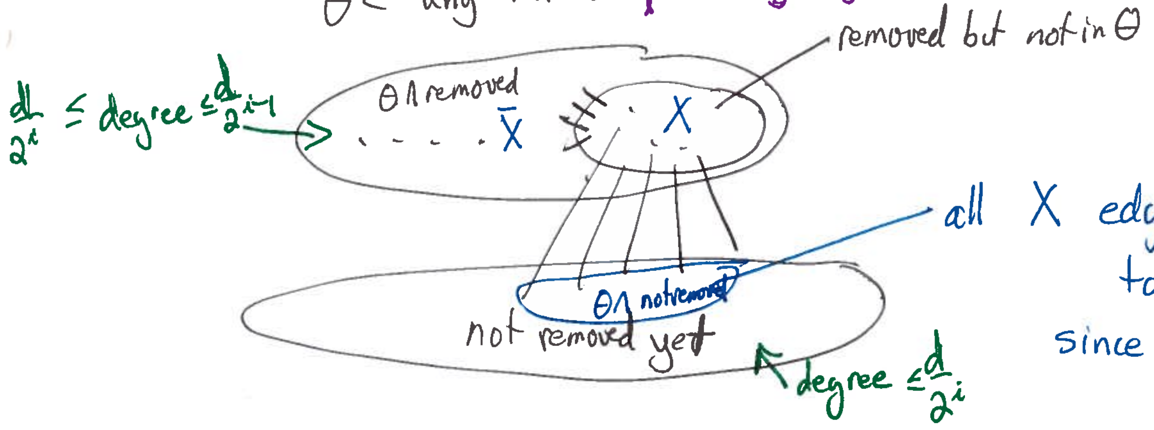
<u>Proof</u>.

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

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

any thing bigger removed earlier

$\Theta \leftarrow$ any min VC (so any edge has to have ≥ 1 endpt in Θ)



all X edges have to go Θ since Θ is a VC

so $|X| \cdot \frac{d}{2^i} \le |\Theta| \cdot \frac{d}{2^{i-1}}$

so $|X| \le 2|\Theta|$