Distributed Algorithms vs. Sublinear Time Algorithms on Sparse Graphs

Max degree \( d \)

Again, Sparse graphs: max degree \( d \), adjacency list representation

A problem to solve:

**Vertex Cover**

\( V' \subseteq V \) is "Vertex Cover" (VC) if \( \forall (u,v) \in E \)

either \( u \notin V' \) or \( v \in V' \)

**VC Question:** What is min size of VC?

Note: in degree \( d \) graph, \( |VC| \geq \frac{m}{d} \) since each node can cover \( d \) edges

VC is NP-complete, but there is a polytime 2-multiplicative approximation

Can you approximate VC in sublinear time?

- Multiplication graph with \( m \) edges \( |VC| = 0 \) \( \Rightarrow \) Can't distinguish these cases in sublinear time
- Multiplication graph with 1 edge \( |VC| = 1 \)

Additive: hard. Need some multiplicative approximation

- Computationally hard to approximate better than 1.36 factor (maybe even 2)

Combination?
\[ y \text{ is } (a, \varepsilon) \text{-estimate of soln value } y \text{ for minimization problem if } \]
\[ y \leq \hat{y} \leq ay + \varepsilon \]
\[ \text{(allows mult + additive error)} \]

(analogous defn for maximization problems)

---

**Some Background on Distributed Algorithms**

- **Network**
  - processors \( \leq \) max degree \( d \) known to all
  - links

- **Communication round**
  - nodes send messages to neighbors

**Def. Vertex Cover problem for distributed networks:**

- Network graph = Input graph (i.e., network computes on itself)
- At end, each node knows if in or out of VC (doesn't know about others necessarily)

**Main insight on why fast distributed \( \iff \) 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!
Can simulate \( V \)'s view of distributed computation in \( \leq d^k \) time

+ figure out if \( v \) is in or out of \( VC \)

Comment: if algorithm is randomized, \( v \) needs to know random bits (or be able to construct) of all \( d^k \) nbdrs. \( k \) must be consistent

"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

"local distributed algorithms"

How do you use this to approximate VC in sublinear time?

Parnas- Ron Framework:

- Sample nodes of graph \( V_1 ... V_r \)
- For each \( V_i \), run simulated distributed algorithm to see if \( V_i \in VC \)

Output \( \# V_i \)'s in \( VC \)

Runtime \( O(r \cdot d^k) \approx O(\varepsilon^2 \cdot d^k) \)

Proof of correctness: Chernoff/holding bounds
fast distributed algorithm for VC:

\[ i = 1 \]

While edges remain:
- remove vertices of degree \( \geq \frac{d}{2^i} \) and adjacent edges
- update degrees of remaining nodes
- increment \( i \)

Output all removed nodes as VC

# rounds: \( \log d \)

Example:

Is it a VC?
- no edges remain at end
- all removed along with some adjacent vertices
Is it a good approximation?

Then let \( VC(G) = \text{size of min VC of } G \)

Then, \( VC(G) \leq \text{output} \leq (2 \log d + 1) VC(G) \)

since output is \( VC \)

to prove

Proof.

Claim: in each iteration, add \( \leq 2 \cdot VC(G) \) new vertices

why: all nodes removed have deg bet \( \frac{d}{2^i} + \frac{d}{2^i} \)

\( \Theta \subseteq \text{any min VC (so any edge has to have z|vertex in } \Theta) \)

\( \frac{d}{2^i} \leq \text{degree} \leq \frac{d}{2^{i-1}} \)

\( \Theta \text{remained removed but not in } \Theta \)

not removed yet

all \( X \) edges have to go \( \Theta \)

since \( \Theta \) is a VC

so \( |x| \cdot \frac{d}{2^i} \leq |\Theta| \cdot \frac{d}{2^{i-1}} \)

so \( |x| \leq 2|\Theta| \)