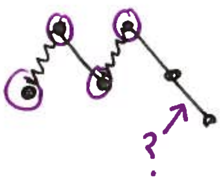
Sublinear Time Approximation Algorithms:

Estimating size of maximal matching in degree bounded graph

Why?

- relation to Vertex Cover



  - $VC \geq MM$ ← for each edge in matching, ≥ 1 $\underbrace{\text{endpt}}_{\text{these are disjoint}}$ must be in VC
  - $VC \leq 2MM$ ← put all MM nodes in VC; if an edge not covered, then violates maximality

- a step towards approx maximum matching

Note: if $\deg \leq d$, Maximal matching $\geq \frac{n}{d}$ ← to see this, run greedy algorithm

Greedy Sequential Matching Algorithm:

$M \leftarrow \phi$

$\forall\ e = (u,v) \in E,$

if neither u or v matched,

add e to M

Output M

} output depends only on ordering of input edges

Observe:
M maximal, since if $\underset{(u,v)}{\overset{\shortparallel}{e}} \notin M$ either u or v already matched earlier

Oracle reduction Framework

assume given deterministic "oracle" $\mathcal{O}(e)$ which tells you if $e \in M$ or not in one step

- $S' \leftarrow s = \frac{8}{\varepsilon^2}$ nodes chosen iid.

- $\forall v \in S'$

$$X_v = \begin{cases} 1 & \text{if any call to } \mathcal{O}((v,w)) \text{ for } w \in N(v) \text{ returns "yes"} \\ 0 & \text{o.w.} \end{cases}$$

- Output $\underbrace{\frac{n}{2s}}_{\text{since 2 nodes matched for each edge in } M} \sum_{v \in S'} X_v + \underbrace{\frac{\varepsilon}{2} \cdot n}_{\text{makes an underestimate unlikely}}$

Behavior of output: Why does it work?

$$|M| = \frac{1}{2} \sum_{v \in V} X_v$$

$$E[|\text{output}|] = E\left[\frac{n}{2s} \sum_{v \in S'} X_v\right] + \frac{\varepsilon}{2} \cdot n$$

$$= \frac{n}{2s} \sum_{v \in S'} E[X_v] + \frac{\varepsilon}{2} \cdot n \quad \leftarrow \text{but } E[X_v] = \frac{2|M|}{|V|} = \frac{2|M|}{n}$$

fraction of matched nodes

$$= \frac{n}{2s} \cdot s \cdot \frac{2|M|}{n} + \frac{\varepsilon}{2} n = |M| + \frac{\varepsilon}{2} n$$

$$\Pr\left[\left|\left(\frac{n}{2s} \sum_{v \in S'} X_v + \frac{\varepsilon}{2} n\right) - E[\text{output}]\right| \geq \frac{\varepsilon}{2} n\right]$$

$$\|$$

$$\Pr\left[\left|\frac{n}{2s} \sum_{v \in S} X_v - |M|\right| \geq \frac{\varepsilon}{2} n\right] \leq \frac{1}{3} \quad \text{by additive Chernoff-Hoeffding}$$

Implementing the oracle:

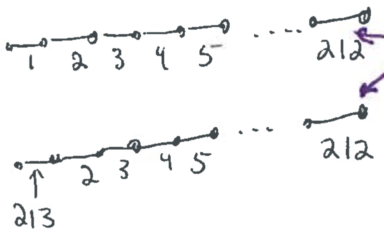
Main idea: figure out "what would $\underbrace{\textbf{greedy}}_{}$ do on (v, w)?"

how? according to which input order?

Problem: Greedy is "sequential"

Can have long dependency chains

example:



even if you know the graph is a line, how do you know if edge is odd or even in the order?

How to implement oracle based on greedy?

To decide if e in matching,

- need to know decisions for **adjacent** edges that came **before** e in ordering
- do <u>not</u> need to know <u>anything</u> about any edge that comes **after** e in ordering since not considered by greedy algorithm before e

So, if any adjacent edge **before** e in ordering matched,

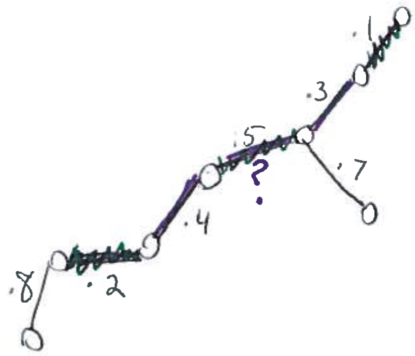
e <u>is not</u> matched

otherwise e <u>is</u> matched

How to break length of dependency chains?

assign random ordering to edges

example



is edge .5 in M?

- recurse on .3
  - recurse on .1
    - no other adjacent edges so .1 is matched
  - therefore .3 is not matched
  - no need to recurse on .7
- don't know yet about .5 so recurse on .4
  - recurse on .2
    - .8 comes after .2 in order so doesn't affect Greedy's behavior
    - same for .4
    - so .2 is matched
  - .4 is not matched
- .5 is matched

Implementation of oracle: assume ranks r_e assign to each edge e

to check if $e \in M$:

$\forall$ e' neighboring e,

- if $r_{e'} < r_e$, recursively check e' +
- if $e' \in M$, return "$e \notin M$" + halt

  else continue

return "$e \in M$"

↑ since no e' of lower rank than e is in M

Correctness: follows from correctness of greedy

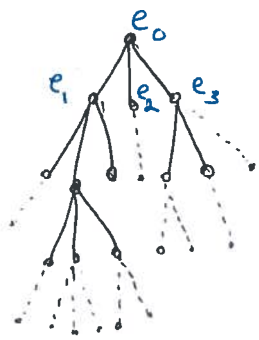
Query complexity:

Claim expected # queries to graph per oracle query is $2^{O(d)}$

Claim $\Rightarrow$ total query complexity is $\frac{2^{O(d)}}{\varepsilon^2}$

Pf of Claim

- Consider QueryTree where root node labelled by original query edge, Children of each node are edges adjacent to it.



- will only query paths that are monotone decreasing in rank
- $\Pr[\text{given path of length } k \text{ explored}] = \frac{1}{(k+1)!}$
- # edges in original graph at dist $= k$ in tree $\leq d^k$
- $E[\#\text{ edges explored at dist } = k] \leq \frac{d^k}{(k+1)!}$
- $E[\text{ total } \#\text{edges explored}] \leq \sum_{k=0}^{\infty} \frac{d^k}{(k+1)!}$

$$\leq \frac{e^d}{d}$$

- $E[\text{query complexity}] \leq d \cdot \frac{e^d}{d} = e^d = 2^{O(d)}$

$\blacksquare$