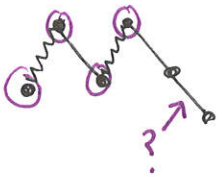
# Sublinear Time Approximation Algorithms:

## Estimating size of maximal matching in degree bounded graph

Why?

- relation to Vertex Cover
  - $VC \geq MM$ ← for each edge in matching, ≥1 endpt must be in VC (these are disjoint)
  - $VC \leq 2MM$ ← 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 $e \notin M$ (i.e. $(u,v)$) 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 $\frac{n}{2s} \sum_{v \in S} X_v + \frac{\varepsilon}{2} \cdot n$

$\frac{n}{2s}$: since 2 nodes matched for each edge in $M$

$\frac{\varepsilon}{2} \cdot n$: makes an underestimate unlikely

### Behavior of output: Why does it work?

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

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

(fraction of matched nodes)

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

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

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

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

by additive Chernoff-Hoeffding

## Implementing the oracle:

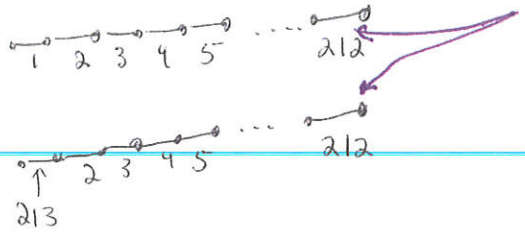
**Main idea:** figure out "what would **greedy** do on $(\pi, w)$?"

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,

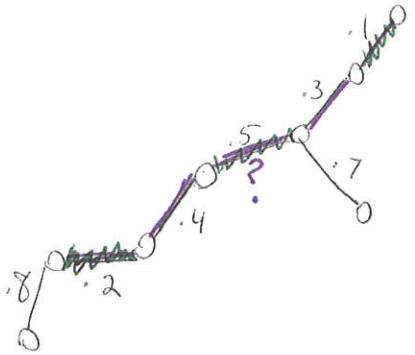
- need to know decisions for **adjacent** edges that came **before** $e$ in ordering
- do not need to know anything about any edge that comes **after** $e$ in ordering since not considered by greedy algorithm before $e$

So, if any adjacent edge **before** $e$ in ordering **matched**,
$e$ **is not** matched
otherwise $e$ **is** 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

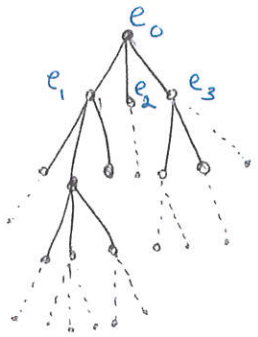
Query complexity:

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 Query Tree 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 $\leq k$ in tree $\leq d^k$

- $E[\#\text{ edges explored at dist } \leq k] \leq \frac{d^k}{(k+1)!}$

- $E[\text{total \# edges explored}] \leq \sum_{k=0}^{\infty} \frac{d^k}{(k+1)!}$

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

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

▨

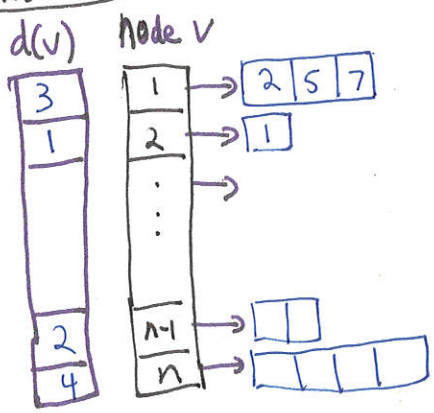
# Approximating the Average degree

def Average degree $\bar{d} = \frac{\sum_{u \in V} d(u)}{|V|}$

$G$: simple (no parallel edges, self-loops)

$\Omega(n)$ edges (not "ultra-sparse")

Representation of $G$:




Adjacency list + degrees:

- degree queries: on $v$ return $d(v)$
- neighbor queries: for $(v, j)$ return $j$th nbr of $v$

## Naive sampling:

Pick ?? random nodes $v_1 \dots v_s$

Output $\frac{1}{s}\sum_i d(v_i)$ (ave degree of sample)

straightforward use of Chernoff/Hoeffding needs $\Omega(\frac{1}{n})$ samples

## Degree sequences are special?

$(n-1, 0, 0, \dots, 0)$ not possible

$(n-1, 1, 1, \dots, 1)$ is possible

## Some lower bounds for approximation:

"Ultra sparse" case: need linear time to get any multiplicative approx:

graph with 0 edges ave deg = 0 vs. graph with 1 edge ave deg = 1/n

requires $\Omega(n)$ queries to distinguish

Ave deg ≥ 2 case :

$n$-cycle $\quad \bar{d} = 2$

$n - cn^{1/2}$ cycle + $cn^{1/2}$-clique $\quad \bar{d} \approx 2 + c^2$

need $\Omega(n^{1/2})$ queries to find a clique node

## Algorithm

idea: group nodes of similar degrees } doesn't work for estimating ave of arbitrary numbers, why should it work here?
estimate average w/in each group }

buckets!

set $\beta = \varepsilon/c$

$t = O\left(\frac{\log n}{\varepsilon}\right)$ # buckets

$$B_i = \left\{ v \mid (1+\beta)^{i-1} < d(v) \leq (1+\beta)^i \right\} \quad \text{for } i \in \{0 \ldots t-1\}$$

Note that total degree of nodes in $B_i$

$$(1+\beta)^{i-1} |B_i| \leq d_{B_i} \leq (1+\beta)^i |B_i|$$

+ total degree of graph $\quad \sum_i (1+\beta)^{i-1} |B_i| \leq d_{\text{total}} \leq \sum_i (1+\beta)^i |B_i|$

First idea:

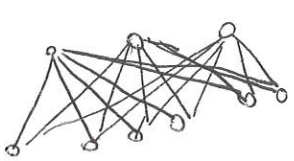
- Take sample $S$
- $S_i \leftarrow S \cap B_i$ (samples that fell in $i^{th}$ bucket)
- estimate average degree of $B_i$ using $S_i$
  i.e. $\rho_i \leftarrow \frac{|S_i|}{S}$

note: $\forall i$, $E[\rho_i] = E\left[\frac{|S_i|}{|S|}\right] = \frac{|B_i|}{n}$

- output $\sum_i \rho_i (1+\beta)^{i-1}$

Problem: $i$ s.t. $|S_i|$ is small ( " " $|B_i|$ " " ) } for these, estimate of $\rho_i$ will be "off"

example



← 3 nodes, degree $n-3$

← $n-3$ nodes, degree 3

$f \leftarrow$ s.t. $(1+\beta)^{i-1} \le 3 \le (1+\beta)^{i}$

$g \leftarrow j$ s.t. $(1+\beta)^{j-1} \le n-3 \le (1+\beta)^{j}$

$|B_f| = n-3$ ← contributes $m$

$|B_g| = 3$ ↖ will never be hit, but contributes $m$

+ all other $|B_i| = 0$

is it good enough for 2-approximation?