Sublinear Time Approximation Algorithms:

Estimating size of maximal matching in degree bounded graph.

Why?

- Relation to Vertex Cover
  - $VC \geq MM$ \iff for each edge in matching, $\exists 1$ endpt in VC, these are disjoint
  - $VC \leq 2MM$ \iff put all HA nodes in VC, if an edge not covered, then violates maximality

- A step towards approx maximum matching

Note: if $\deg \leq d$, maximal matching \( \leq \frac{n}{d} \) \iff to see this, run greedy algorithm

Greedy Sequential Matching Algorithm:

\[
M \leftarrow \emptyset \\
\forall e = (u, v) \in E, \\
\text{if neither } u \text{ or } v \text{ matched, add } e \text{ to } M
\]

Output $M$

Observe: $M$ maximal, since if $e \notin M$, either $u$ or $v$ already matched earlier
**Oracle reduction Framework**

- Assume given deterministic "oracle" \( O(e) \)
  - which tells you if \( e \in M \) or not in one step

\[ S \subseteq S = \frac{n}{\epsilon_2} \text{ nodes chosen iid.} \]

\[ \forall v \in S \quad X_v = \begin{cases} 1 \quad \text{if any call to } O((v, w)) \text{ for } w \in N(v) \\ 0 \quad \text{o.w.} \end{cases} \]

- Output \( \frac{n}{2S} \sum_{v \in S} X_v + \frac{\epsilon}{2} \cdot n \)

Since 2 nodes matched for each edge in \( M \)

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

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

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

\[ = \frac{n}{2S} \sum_{v \in S} \mathbb{E}[X_v] + \frac{\epsilon}{2} \cdot n \quad \text{but } \mathbb{E}[X_v] = \frac{2|E|}{n} = \frac{2|M|}{n} \]

\[ = \frac{n}{2S} \cdot S \cdot \frac{2|M|}{n} + \frac{\epsilon}{2} \cdot n = \frac{2|M|}{n} + \frac{\epsilon}{2} \cdot n \]

\[ \Pr \left[ \left| \frac{n}{2S} \sum_{v \in S} X_v + \frac{\epsilon}{2} \cdot n \right| - \mathbb{E}[\text{output}] \right| \geq \frac{\epsilon}{2} \cdot n \right] \leq 1 \]

\[ \Pr \left[ \left| \frac{n}{2S} \sum_{v \in S} X_v - |M| \right| = \frac{\epsilon}{2} \cdot n \right] \leq \frac{1}{3} \quad \text{by additive Chernoff-Hoeffding} \]
Implementing the oracle:

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

Problem: Greedy is "sequential"

Can have long dependency chains

Example: 1 2 3 4 5 \ldots 212

\underline{1} 2 3 4 5 \ldots 212

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 to 0
    - 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 re assigned to each edge e to check if e ∈ M:

∀ e' neighboring e,

• if r_e' < r_e, recursively check e'+
  • if e' ∈ M, return "e ∈ M" and halt
    else continue
  return "e ∉ 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 $O(d^2)$

Claim ⇒ total query complexity is $\frac{2^{O(d)}}{e^2}$
pf of Claim

- Consider a 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.

- \( \Pr[\text{given path of length } k \text{ explored}] = \frac{1}{(k+1)!} \)

- \( \# \text{ edges in original graph at dist } \leq k \text{ in tree} \leq d^k \)

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

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

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