Sublinear time algorithms based on simulating greedy algorithms

Lecture 14
Ronitt Rubinfeld
Sparse Graphs

- **Representation:**
  - Max degree $d$
  - Adjacency list format

- **Problems of interest:**
  - Vertex cover
  - Matching
  - Dominating set
  - Set cover (on sparse set systems)
Maximal Matching

• \( M \subseteq E \) is a matching if \( \forall (u,v), (w,x) \in M \), 
  \( \{u,v\} \cap \{w,x\} = \emptyset \).

• \( M \) is a maximal matching if adding any edge violates the matching property.

• Note:
  – Size of any vertex cover \( \geq \) size of any maximal matching.
Greedy algorithm for maximal matching

• Algorithm:
  – \( M \leftarrow \emptyset \)
  – \( \forall e=(u,v) \in E \)
    – If neither of \( u,v \) matched
      – Add \( e \) to \( M \)
    – Output \( M \)

• Why is \( M \) maximal?
  – If \( e \) not in \( M \) then either \( u \) or \( v \) already matched

• How big is \( M \) for graphs with max degree \( d \)?
  – Any edge in \( M \) removes \( < 2d \) others from consideration
  – Still have possible edges to add for \( \geq n/2d \) rounds
Idea for sublinear time algorithm:

- Run [Parnas-Ron] reduction algorithm:
  - i.e.,
    - Sample $O(1/\epsilon^2)$ nodes
    - For each sampled node, call “oracle” on neighboring edges to decide if it is in the matching
    - Output
      $\left(\frac{\text{fraction of sampled nodes in matching}}{2}\right) \cdot n + (\epsilon/2)n$

- How do you implement oracle?
  - Idea: figure out what greedy would do
Problems with greedy

• Can have long dependency chains!
  – (see board for example)

• How can you implement the oracle?
  – Must know if adjacent edges that come before e in the ordering are in the matching
  – Do not need to know anything about edges coming after
Breaking long dependency chains

• Assign \textit{random} ordering to edges
  – Greedy works under any ordering
  – To show: random order has short dependency chains
Implementing oracle $\mathcal{O}$

[Nguyen Onak]

- **Preprocessing:**
  - assign random number $r_e \in [0,1]$ to each $e \in E$

- **Oracle implementation:**
  - Input: edge $e \in E$,
  - Output: is $e$ in $M$?
  - Algorithm:
    - Find all the adjacent edges of $e$, $e' \in E$, such that $r_{e'} < r_q$
    - Recursively check if any in $M$
      - If any in the matching, output NO
      - If none are in the matching, output YES
Example Run $\emptyset$ (cont.)
Example Run $\emptyset$ (cont.)
Example Run $\Theta$ (cont.)
Example Run $\emptyset$ (cont.)
Example Run $\emptyset$ (cont.)
Example Run $\varnothing$ (cont.)
Example Run $\emptyset$ (cont.)
Example Run $\emptyset$ (cont.)
Example Run $\emptyset$ (cont.)
Example Run $\theta$ (cont.)
Example Run θ (cont.)
Example Run $\emptyset$ (cont.)
Correctness

• This algorithm simulates run of classical greedy algorithm
  – Greedy works under any ordering of edges

• Outputs estimate $t$ such that

\[ \text{MM}(G) \leq t \leq \text{MM}(G) + \epsilon n \]

where $\text{MM}(G)$ is size of some maximal matching
Complexity: Heuristic, nonoptimal argument

Consider ranks along chain of recursive calls:

- \( r_e' \) uniformly distributed
- \( \Pr[r_e' \text{ less than median of ranks } < r_e \mid r_e', < r_e ] \) is roughly \( \frac{1}{2} \)
  - So ranks go down by factor of roughly 2 at each recursion
- When rank \( < 1/2d \) unlikely that any neighbor ranked smaller
- \( O(\log d) \) levels of recursion suffice
Complexity

• Claim: Expected number queries to graph per oracle query is $2^{O(d)}$
  
  – Total complexity is $2^{O(d)}/\varepsilon^2$

– Main idea:
  
  • Bound probability a path of length k explored:
    – Ranks must decrease along the path
    – So probability $\leq 1/(k)!$
Complexity

• Claim: Expected number of queries to graph per oracle query is $2^{O(d)}$

• Proof:
  – $\Pr[\text{given path of length } k \text{ explored}] \leq 1/(k)!$
  – Number of neighbors at distance $k \leq d^k$
  – $E[\text{Number of nbrs explored at dist } k] \leq d^k/(k)!$
  – $E[\text{number of explored nodes}] \leq \sum_{k=0}^{\infty} d^k/(k)! \leq e^d/d$
  – $E[\text{query complexity}] = O(d) \cdot e^d/d$
    \[= 2^{O(d)}\]
Further work

• Always recurse on least ranked edge first gives better runtime [Yoshida Yamamoto Ito]
• More complicated argument for Maximum matching, set cover,…
• Even better results for certain classes of graphs [Hassidim Kelner Nguyen Onak]
  – Minor-free (e.g., planar)