Sublinear Time and Space Algorithms 2022B – Lecture 10
Sublinear-Time Algorithms for Maximum Matching and Vertex Cover*

Robert Krauthgamer

1 Maximum Matching

Problem definition:

Input: An n-vertex graph $G = (V, E)$ of maximum degree D, represented such that one has direct access to the j-th neighbor of a vertex.

Definition: A matching is a set of edges that are incident to distinct vertices.

Goal: Compute the maximum size of a matching in G.

Note: The matching itself is too large to report in sublinear time, we only estimate its cost using (α, β)-approximation, i.e., $OPT \leq ALG \leq \alpha\ OPT + \beta$.

Theorem 1 [Nguyen and Onak, 2008]: There is an algorithm that gives $(2, \varepsilon n)$ approximation to the maximum matching size in time $D^{O(D)}/\varepsilon^2$.

Main idea: It is well-known that *maximal matching* (note: maximal means with respect to containment) is a 2-approximation for *maximum matching*. We will pick one such matching almost implicitly, and then estimate its size by sampling.

Algorithm GreedyMatching:

1. Start with an empty matching M.

2. Scan the edges (in arbitrary order), and add each edge to M unless it is adjacent to an edge already in M.

Lemma 1a: The size of a maximal matching is at least half that of a maximum matching.

Proof: Exercise

Algorithm ApproxGreedyMatching:

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.

1. choose random edge priorities $p(e) \in [0,1]$, implicitly defining a permutation of the edges

2. choose $s = O(D/\varepsilon^2)$ edges $e_1, \ldots, e_s$ uniformly at random from the Dn possibilities (note that each edge has two "chances" to be chosen, and some choices may lead to no edge, if the actual degree is smaller than D)

3. for each edge e_i, compute an indicator X_i for whether e_i belongs to the maximal matching corresponding to p, by exploring the neighborhood of e_i incrementally

[stop if the algorithm took too many steps altogether]

4. report $X = \frac{Dn}{2s} \sum_i X_i$

Running time: Let M be a greedy matching constructed according to the priorities p. As seen in class, to determine whether a single $e_i \in M$, we only "expect" to explore paths of length up to $k = O(D)$. Thus, the expected running time is $O(s \cdot D^{cD}) \leq D^{O(D)}/\varepsilon^2$, and by Markov's inequality there is small probability to exceed it by much.

Correctness: As seen in class, it follows by applying Chebychev's inequality to $X = \frac{Dn}{2s} \sum_i X_i$.

2 Vertex Cover in Planar Graphs via Local Partitioning

Problem definition:

Input: A graph $G = (V, E)$ on n vertices. We shall assume G is planar, has maximum degree $\leq d$, and is represented using adjacency lists.

Definition: A vertex-cover is a susbet $V' \subset V$ that is incident to every edge.

Goal: Estimate $\mathrm{VC}(G)$ = the minimum size of a vertex-cover of G.

Theorem 2 [Hassidim, Kelner, Nguyen and Onak, 2009]: There is a randomized algorithm that, given $\varepsilon > 0$ and a planar graph G with maximum degree $\leq d$, estimates whp $\mathrm{VC}(G)$ within additive εn and runs in time $T(\varepsilon, d)$ (independent of n).

Main idea: Fix "implicitly" some near-optimal solution. Then estimate it's size by sampling $s = O(1/\varepsilon^2)$ random vertices and checking whether they belong to that solution.

Initial analysis: Let SOL be the implicit solution computed by the algorithm, let X_i for $i = 1, \ldots, s = O(1/\varepsilon^2)$ be an indicator for whether the i-th chosen vertex belongs to SOL. The algorithm outputs $\frac{n}{s} \sum_i X_i$. We will need to prove:

$$|\mathrm{SOL} - \mathrm{VC}(G)| \leq \varepsilon n$$
$$\Pr[|\tfrac{n}{s} \textstyle\sum_i X_i - \mathrm{SOL}| \leq \varepsilon n] \geq 0.9$$

The last inequality follows immediately from Chebychev's inequality, since each $X_i = 1$ independently with probability SOL/n.

Definition: We represent a partition of the graph vertices as $P : V \to 2^V$. It is called an (ε, k)-*partition* if every part $P(v)$ has size at most k, and at most $\varepsilon|V|$ edges go across between different parts.

Theorem 3: For every $\varepsilon, d > 0$ there is $k^* = k^*(\varepsilon, d)$ such that every planar G with max-degree $\leq d$ admits an (ε, k^*)-partition.

We will discuss its proof in the next class.

Our sublinear algorithm will actually not compute this partition directly, and instead will use local computation to compute another partition (with somewhat worse parameters).

Proof Plan for Theorem 2: Given an (ε, k)-partition P of G, we define the solution SOL by taking some optimal solution in each part of P, and adding one endpoint for each cross-edge. The following lemma is immediate.

Lemma2a: $\text{VC}(G) \leq \text{SOL} \leq \text{VC}(G) + \varepsilon n$.

The remaining (and main) challenge is to implement a partition oracle, i.e., an "algorithm" that can compute $P(v)$ for a queried vertex $v \in V$ in constant time. Note: P could be random, but should be "globally consistent" for the different queries v.

To be continued in the next class.