Sublinear Time and Space Algorithms 2024A – Lecture 11
Planar Vertex Cover (cont'd) and Sampling an Edge Uniformly[*]

Robert Krauthgamer

1 Vertex Cover in Planar Graphs via Local Partitioning (cont'd)

After last week, it remained to prove that the algorithm below provides a partition oracle.

Algorithm Partition (used later as oracle):

Remark: It uses parameters ε', k' that will be set later (in the proof)

```
1. $P = \emptyset$
2. iterate over the vertices in a random order $\pi_1, \ldots, \pi_n$
3.   if $\pi_i$ is still in the graph then
4.     if $\pi_i$ has an $(\varepsilon', k')$-isolated neighborhood in the current graph
5.       then $S =$ this neighborhood
6.       else $S = \{\pi_i\}$
7.     add $\{S\}$ to $P$ and remove $S$ from the graph
8. output $P$
```

Lemma 1c: For every $\varepsilon > 0$, Algorithm Partition above with parameters $\varepsilon' = \varepsilon/(12d)$ and $k = k^*(\varepsilon'^2, d)$ computes whp an (ε, k)-partition. Moreover, it can be implemented as a partition oracle (given a query vertex, it returns the part containing that vertex), whose running time (and query complexity into G) to answer q non-adaptive queries is whp at most $q \cdot 2^{d^{O(k)}}$.

Proof of Lemma 2c: By construction, the output P is a partition, and every part in it has size at most k.

Analyzing the number of cross-edges in P: for each $i = 1, \ldots, n$ define two random variables related to π_i, as follows. Let $S_i = P(\pi_i)$, i.e. the set $S \in P$ that contains π_i (note it is removed from the graph in iteration i or earlier), and define $X_i = \mathrm{e_{out}}'(S_i)/|S_i|$, where $\mathrm{e_{out}}'(S_i)$ is the number of edges at the time of removing S_i. Notice that a nontrivial iteration create one part $S \in P$, and thus "sets" $|S|$ variables X_i all to the same value, thus $\sum_i X_i = \sum_{S \in P} \mathrm{e_{out}}'(S)$ is the number of cross-edges in P (each edge is counted once, because the graph changes with the iterations).

[*]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.

Now fix i. Since π_i is a random vertex, by Lemma 1a, with probability $\geq 1-2\varepsilon'$, it has an (ε', k)-isolated neighborhood in the input G, and then $X_i \leq \varepsilon'$ (this is certainly true if π_i was removed at an earlier iteration). This argument applies also to the current graph, because a subgraph of G is still planar with degrees bounded by d (formally, we need to condition on the first $i-1$ iterations). With the remaining probability $\leq 2\varepsilon'$, we bound $X_i \leq d$ which always holds. Altogether,

$$\mathbb{E}[X_i] \leq 1 \cdot \varepsilon' + 2\varepsilon' \cdot d \leq 3\varepsilon' d.$$

$$\mathbb{E}[\sum_i X_i] \leq 3\varepsilon' dn.$$

By Markov's inequality, with probability $\geq 3/4$, the number of cross-edges in P is at most $4(3\varepsilon' dn) = \varepsilon n$.

Implementation as an oracle: We generate the permutation π on the fly by assigning each vertex v a priority $r(v) \in [0,1]$ (and remember previously used values). Before computing $P(v)$, we first compute (recursively) $P(w)$ for all vertices w within distance at most $2k$ from v that satisfy $r(w) < r(v)$. (Note that a vertex w at distance $2k-2$ might affect v by causing removal of a vertex mid-way between v and w.) If $v \in P(w)$ for one of them, then $P(v) = P(w)$. Otherwise, search by brute-force for a (ε', k)-isolated neighborhood of v, keeping in mind that vertices in any $P(w)$ as above are no longer in the graph.

Running time: We effectively work in an auxiliary graph H, where we connect two vertices if their distance in G is at most $2k$. Thus, the maximum degree in H is at most $D = d^{2k}$. As seen earlier, this means the expected number of vertices inspected recursively is at most $D^{O(D)} = 2^{D^{O(1)}} = 2^{d^{O(k)}}$.

QED

Concluding Theorem 1: Lemma 1c designs a partition oracle, and thus completes (the proof of) the algorithm for vertex cover.

2 Sampling an Edge Uniformly

Problem definition: Input: A graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges represented as the adjacency list for each vertex (thus can access the degree and outgoing edges of any vertex).

Goal: report an edge sampled uniformly from E.

Easy case: If G is regular (all vertices have the same degree), one can sample uniformly a vertex and then sample uniformly an outgoing edge.

The problem is harder when the graph is not regular, not even approximately.

Theorem [Eden, Narayanan and Tetek, 2023]: There is an algorithm that, given also m, samples an edge uniformly in expected time $O(n/\sqrt{m})$. (In fact, an $O(1)$-approximation to m suffices.)

Remark: If G is connected, we saw that $O(1)$-approximation to m can be computed in time $O(\sqrt{n})$, which can be absorbed in $O(n/\sqrt{m})$.

We will see today a weaker result of [Eden and Rosenbaum, 2018], where the output distribution is within $(1+\varepsilon)$-approximation (pointwise):

$$\forall e \in E, \qquad \Pr[\mathit{output} = e] \in (1 \pm \varepsilon)\tfrac{1}{|E|}.$$

This algorithm always outputs an edge, and its expected running time is $O(n/\sqrt{\varepsilon m})$. One can of course terminate it after slightly more time and output FAIL, and argue that it happens with small probability by Markov's inequality.

Exer (example application): Show how to estimate within additive $\pm\delta$ (whp) the fraction of edges inside a subset $S \subset V$ by using $O(1/\delta^2)$ independent samples of of a uniformly random edge.

Proof plan: Think of every edge as two directed anti-parallel edges (as in the regular case). Our goal is to report every directed edge uv with the same probability, and report FAIL with the remaining (small) probability. The final algorithm repeats this until the output is an edge (and not FAIL).

Fix a threshold $\theta = \sqrt{2m/\varepsilon}$, and partition V based on vertex degrees into *light and heavy*, namely $L = \{v \in V : \deg(v) \leq \theta\}$ and $H = V \setminus L$.

A (directed) edge uv is called *light* if its tail u is light. Otherwise the edge is called *heavy.*

Sampling a light edge: Sample uniformly a vertex u' and check it is light, then sample uniformly $j \in [\theta]$ and check if the j-th edge from u' exists. If both checks pass, report that j-th edge. Otherwise report FAIL. Clearly

$$\forall \text{ light edge } uv, \qquad \Pr[\mathit{output} = uv] = \tfrac{1}{n\theta}. \tag{1}$$

Sampling a heavy edge: First sample a light edge $u'v'$ as described above. Now check that the head v' is heavy. and sample uniformly an incident edge $v'w'$. If the check passes report the edge $v'w'$. Otherwise report FAIL. Let $\deg_L(v)$ denote the number of light edge from v'; then

$$\forall \text{ heavy edge } vw, \qquad \Pr[\mathit{output} = vw] = \tfrac{\deg_L(v)}{n\theta} \cdot \tfrac{1}{\deg(v)}. \tag{2}$$

Lemma: If $v \in H$ then $\frac{\deg_L(v)}{\deg(v)} \in [1-\varepsilon, 1]$.

Proof: $\deg_H(v) \leq |H| \leq \frac{2m}{\theta} = \varepsilon\theta < \varepsilon \deg(v)$.

Caveat: Which of the two cases/algorithms should we run? Here it's easy, we just pick between the two options with equal probabilities.

Rejection sampling: The above uses a technique called rejection sampling, where the algorithm may decide to reject the sample, i.e., report FAIL instead. It could be because the sample turns out to be "bad" (e.g., the vertex is not light), or with some probability (e.g., fixed or that depends on the sample-at-hand) to effectively reduce the probability of reporting this sample.

Algorithm ApproxSample:

Input: G, m, ε

```
1. $u' \leftarrow$ uniformly random vertex
2. $j \leftarrow U([\theta])$, where $\theta = \sqrt{m/\varepsilon}$
3. if $\deg(u') > \theta$ or $j > \deg(u')$ then return FAIL
4. $v' \leftarrow j$-th neighbor of $u'$
5. $b \leftarrow$ Bernoulli(1/2)
6. if $b = 1$ then
7.    return $u'v'$
8. else if $v'$ is heavy then
9.    $w' \leftarrow$ random neighbor of $v'$
10.   return $v'w'$
11. return FAIL
```

Analysis: Based on (1),(2) and the Bernoulli variable b:

$$\forall \text{ light edge } uv, \qquad \Pr[output = uv] = \tfrac{1}{2n\theta};$$
$$\forall \text{ heavy edge } vw, \qquad \Pr[output = vw] = \tfrac{1}{2n\theta} \cdot p(v), \text{ where } p(v) = \tfrac{\deg_L(v)}{\deg(v)} \in [1-\varepsilon, 1].$$

The probability to report an edge (and not FAIL) is at least $2m \cdot \frac{1-\varepsilon}{n\theta} = \Omega(\frac{\sqrt{\varepsilon m}}{n})$, hence the expected number of attempts until sampling an edge is indeed $O(\frac{n}{\sqrt{\varepsilon m}})$.

QED

First Improvement: The output has a "bias" of factor $p(v) = \frac{\deg_L(v)}{\deg(v)}$.

The algorithm can estimate $p(v)$ within additive δ, by picking $O(1/\delta^2)$ random neighbors of v and counting how many of them are light. This takes only $O(1/\delta^2)$ time.

If the algorithm knows $p(v)$ exactly, it could just add another round of rejection sampling: Keep a light edge with probability $1-\varepsilon$, and a heavy edge with probability $\frac{1-\varepsilon}{p(v)} \leq 1$. We could use fixed ε (say $\varepsilon = 1/3$).

If $p(v) \in [\frac{2}{3}, 1]$ then having additive error δ is equivalent to multiplicative error $1 + O(\delta)$. The output will still be $(1+\delta)$-approximation of the distribution, but the running time improves from multicplicative dependence (on ε) to additive dependence (on δ, recall $\varepsilon = 1/3$).

Second Improvement: Apply a method (called Bernoulli factory) that uses in expectation $O(1)$ Bernoulli variables (coins) with unknown parameter $p \in [\frac{2}{3}, 1]$ to generate a Bernoulli variable with parameter $\frac{2/3}{p}$ (more precisely, a quantity proportional to $\frac{1}{p}$) to "fix" the heavy case.

This crux is that the method is independent of $p(v)$.

Remark: The method may use also coins of its own.

Exer: Give an example for such a method, for instance using random samples from $B(p)$ to generate $B(1/2)$ (assuming $p \geq 2/3$).