Sublinear Time and Space Algorithms 2022B – Lecture 11
Sublinear-Time Algorithms for Planar Vertex Cover (cont'd)*

Robert Krauthgamer

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

Last week we stated the following theorem.

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.

It is proved using the famous Planar Separator Theorem (which we will not prove).

Planar Separator Theorem [Lipton and Tarjan, 1979]: In every planar graph $G = (V, E)$ there is a set S of $O(\sqrt{|V|})$ vertices such that in $G \setminus S$, every connected component has size at most $n/2$.

Remark: It extends to excluded-minor families.

Exer: Prove Theorem 3 by using the planar separator theorem recursively. What k^* do you get?

Our sublinear algorithm will not compute this partition directly, and instead will use local computation to compute another partition P (with somewhat worse parameters). The remaining (and main) challenge is to design an algorithm that can compute $P(v)$ for a queried vertex $v \in V$ in constant time. This is called a *partition oracle.* Note: P could be random, but should be "globally consistent" for the different queries v.

Algorithm Partition (used later as oracle):

Remark: It uses parameters k, ε' 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 π_i is still in the graph then
4.   if π_i has a (k, ε')-isolated neighborhood in the current graph
5.   then $S =$ this neighborhood
6.   else $S = \{\pi_i\}$

*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.

7. add $\{S\}$ to P and remove S from the graph
8. output P

Definition: A (k,ε')-*isolated neighborhood* of $v \in V$ is a set $S \subset V$ that contains v, has size $|S| \le k$, the subgraph induced on S is connected, and the number of edges leaving S is $\mathrm{e_{out}}(S) \le \varepsilon'|S|$.

Lemma 2b: Fix $\varepsilon' > 0$. Then a random vertex in G has probability at least $1 - 2\varepsilon'$ to have a $(k^*(\varepsilon'^2, d), \varepsilon')$-isolated neighborhood.

Proof of Lemma 2b: Was seen in class, by considering the $(\varepsilon'^2, k^*(\varepsilon'^2, d))$-partition guaranteed by Theorem 3.

Lemma 2c: 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, where every part has size at most k.

To analyze the number of cross-edges in P, we define for each $i = 1, \ldots, n$ 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 each $S \in P$ "sets" $|S|$ variables X_i 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).

Now fix i. Since π_i is a random vertex, by Lemma 2a, with probability $\ge 1 - 2\varepsilon'$, it has a (k,ε')-isolated neighborhood in the input G, and also in later iterations (as that subgraph of G is planar too), in which case $X_i \le \varepsilon'$ (both if π_i is removed in iteration i and if in an earlier iteration). With the remaining probability $\le 2\varepsilon'$, we can bound $X_i \le d$ which always holds. Altogether,

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

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

By Markov's inequality, with probability $\ge 3/4$, 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 (k,ε')-isolated neighborhood of v, keeping in mind that vertices in any $P(w)$ as above are no longer in the graph. Searching for an optimal vertex cover inside a part is done exhaustively.

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 $D^{O(D)} = 2^{D^{O(1)}} = 2^{d^{O(k)}}$.