Sublinear Time and Space Algorithms 2020B – Lecture 13
Sublinear-Time Algorithms Planar Vertex Cover (cont’d) and
Communication Complexity of Equality*

Robert Krauthgamer

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

Recall from last week that our 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 $v$.

Algorithm Partition (used later as oracle):

Remark: It uses parameters $k, \varepsilon'$ 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 a $(k, \varepsilon')$-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 $P$

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

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

Lemma 2b: For every $\varepsilon > 0$, Algorithm Partition above with parameters $\varepsilon' = \varepsilon/(12d)$ and $k = k^*(\varepsilon'^2, d)$ computes whp an $(\varepsilon, k)$-partition. Moreover, it can be implemented as a partition oracle (given a query vertex, it returns the part of that vertex), whose running time (and query

---

*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.
complexity into \( G \) to answer \( q \) non-adaptive queries is whp at most \( q \cdot 2^{O(k)} \).

WE SKIPPED THIS PROOF IN CLASS. But here is the rest of the proof.

**Proof of Lemma 2b:** 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 \( \pi_i \), as follows. Let \( S_i = P(\pi_i) \), i.e. the set \( S \in P \) that contains \( \pi_i \) (note it is removed from the graph in iteration \( i \) or earlier), and define \( X_i = \frac{e_{out}'(S_i)}{|S_i|} \), where \( e_{out}'(S_i) \) 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 \( \sum_i X_i = \sum_{S \in P} e_{out}'(S) \) is the number of cross-edges in \( P \) (each edge is counted once, because the graph changes with the iterations).

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

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

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

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 \( \pi \) 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 \((k, \varepsilon')\)-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 \( D^{O(D)} = 2^{D^{O(1)}} = 2^{d^{O(k)}} \).

2 Communication Complexity

**Model:** Two parties, called Alice and Bob, receive inputs \( x, y \) respectively. They can exchange messages, in rounds, until one of them (or both) reports an output \( f(x, y) \).

Main measure is communication complexity, i.e., total communication between the parties (in bits, worst-case).

Variants of randomization: none (deterministic), shared/public, or private.
Number of rounds: zero (simultaneous, i.e., each sends a message to a referee and not directly to each other), one (one-way communication), or more/unbounded.

Many other variants, like more players communicating in series (or broadcast etc.), with different input model (e.g., number on forehead instead of number in hand).

Equality as an Example:

Problem definition: Alice and Bob’s inputs are $x, y \in \{0, 1\}^n$, and their goal is to compute $EQ(x, y) = 1_{\{x=y\}}$.

Public randomness: There is a (simultaneous) protocol with $O(1)$ bits.

Private randomness: There is a (one-way) protocol with $O(\log n)$ bits.

Deterministic one-way: Every protocol requires $\Omega(n)$ communication bits.