6.889 Sublinear Time Algorithms February 25, 2019

Lecture 6

Lecturer: Ronitt Rubinfeld *Scribe: Michal Shlapentokh-Rothman*

1 Outline

Today, we will discuss a general framework for testing minor-free properties by looking at a way to test planarity.

- Problem Setup and Definitions
- Algorithm Given Partition Oracle P
- Implementing Partition Oracle P

2 Problem Setup and Definitions

Our overall goal is to be able to distinguish graphs with a property P from all other graphs in sublinear time. However, there is no (known) method to do this exactly without exploring the whole graph. So, we will make a compromise: can we distinguish graphs with property P from graphs that are **ϵ-far** away from having property P? We will show how to distinguish such graphs by looking at the property of

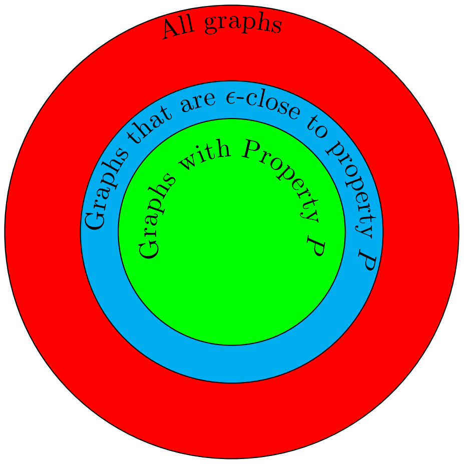

Figure 1: We want to distinguish the line between the blue circle and the red circle

planarity. First, we will start with several definitions. Note that all graphs have max degree of d.

Definition 1 *H is a **minor** of a graph G if we can obtain H from G via vertex removals, edge removals, or edge contractions.*

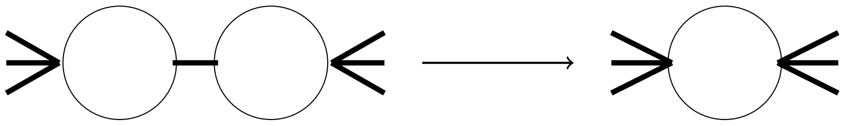

Figure 2: The graph on the left, is a minor of the graph on the right

Definition 2 *G is $H-minor-free$ if H is not minor of G*

Definition 3 *G is ϵ-close to $H-minor-free$ if can remove $\leq \epsilon dn$ edges to make it $H-minor-free$. (Otherwise, G is ϵ-far)*

Definition 4 *For a minor close property P, if $G \in P$, then all minors of G are in P*

Our goal is to test $H - minor$ freeness which means we want to pass $H - minor$ free graphs and fail graphs that are ϵ-far from $H - minor$ free. The following theorem and definitions will be helpful.

Theorem 5 (Robertson and Seymour) *Every minor-closed property is expressible as a constant number of excluded minors.*

Examples of minor-closed properties include $K_{3,3}$ or K_5, non-planar graphs and bounded tree width.

Definition 6 *G is (ϵ, k)-hyperfinite if we can remove $\leq \epsilon n$ edges and remain with connected components of size $\leq k$.*

In other words, we have a (ϵ, k)-hyperfinite graph if we can remove a few edges and break up graph into small components. We can connect the notion of hyperfiniteness to $H - minor$ freeness.

Theorem 7 *Given H, $\exists$ C_H such that $\forall$ $0 < \epsilon < 1$, every $H - minor$ free of graph of degree $\leq d$ is $(\epsilon d, \frac{C_H^2}{\epsilon^2})$-hyperfinite.*

All of this means that subgraphs of $H - minor$ free graphs are also $H - minor$ free and also hyperfinite.

Why is hyperfinitiness useful? We can use it to prove that a graph is $H - minor$ free. We can partition a graph G into G' by removing only a few edges. If non-constant size components remain, then we know that G is **not** $H - minor$ free.

What if G' is ϵ-close to having the property? Then we know that G is $2 \cdot \epsilon$-close to having the property. Furthermore, it is particularly easy to test whether G' has the property since we can just pick random components and in constant time test if they have the property.

What we need now is is a 'local' (sublinear) way to determine G'. First, we will assume that we have a 'partition oracle' P and then we will construct the partition oracle.

3 Algorithm Given Partition Oracle

Our partition oracle will work as follows:

- Input: vertex v
- Output: $p[v]$ (where v is the partition) name such that $\forall$ $v \in V$
    1. $|P[v]| \leq k$
    2. $P(v)$ is connected

If G is $H - minor$ free, then with probability $\frac{9}{10}$, $|u, v \in E|P[u] + P[v]| \leq \frac{\epsilon d}{4}n$. In the first half of the algorithm, we are determining if the partition oracle 'looks right' meaning are there few crossing edges and in the second part we are testing random partitions.

3.1 Algorithm

Algorithm 1 Algorithm with Oracle

```
$\hat{f}$ number of edges $(u, v)$ such that $P[u] \neq P[v]$ to additive error $\leq \frac{\epsilon dn}{8}$
if $\hat{f} > \frac{3}{8}\epsilon dn$ then output fail and halt
end if
Choose $S = \mathcal{O}(\frac{1}{\epsilon})$ random nodes
for $s \in S$ do
    if $P[s] \geq K$ or $P[s]$ is not $H - minor$ free then
        reject and halt.
    end if
end for
Accept
```

The first part of the algorithm (the part before the choose statement), makes $\mathcal{O}(\frac{1}{\epsilon^2})$ calls to the oracle. The second part, makes $\mathcal{O}(\frac{d}{\epsilon^2})$ calls to the oracle, making the total number of calls $\mathcal{O}(\frac{d}{\epsilon^3})$

3.2 Analysis

If G is $H - minor$ free:

- We know that if G is $H - minor$ free, then we will not halt on the first half of the algorithm because $E[\hat{f}] \leq \frac{\epsilon dn}{4}$ and with sampling bounds $\hat{f} \leq \frac{\epsilon dn}{4} + \frac{\epsilon dn}{8} = \frac{3}{8}\epsilon dn$.
- Since, our oracle produces $H - minor$ free graphs we can say $\forall s \in V$, $P[s]$ is $H - minor$ free.

We see that our algorithm works when G is $H - minor$ free. Let us now analyze what happens when G is ϵ-far from $H - minor$ free.

- Case 1:P's output does not satisfy $|\{(u, v) \in E : P(u) \neq P(v)\}| < \frac{\epsilon dn}{2}$. Using out sampling bounds, we can say that $\hat{f} > \frac{\epsilon dn}{2} - \frac{\epsilon dn}{8} = \frac{3}{8}\epsilon dn$ which means the algorithm will out put 'fail' with probability $\geq \frac{9}{10}$.
- Case 2: P's output does satisfy $|\{(u, v) \in E : P(u) \neq P(v)\}| < \frac{\epsilon dn}{2}$. This means that G' can be created from G with edges in C removed. It is important to note that if G' is $\frac{\epsilon}{2}$-close to G, This means that if G is ϵ-far from aving a property, then G' is $\frac{\epsilon}{2}$-far from having the property. Since G' is $\frac{\epsilon}{2}$-far from $H - minor$ free, we much change $\geq \frac{\epsilon dn}{2}$ edges to make it $H - minor$ free, which touch at least $\frac{\epsilon n}{2}$ nodes. So with probability $\geq \frac{\epsilon}{2}$, we will pick a node in a component which is not $H - minor$ free.

We now need to implement the partition oracle P.

4 Implementing Partion Oracle

We will implement our partion oracle by first defining a global partitioning stragety that does not work in sublinear time and then figuring out how to implement the strategy locally.

4.1 Global Partitioning Strategy

We will first define a useful concept known as an isolated neighborhood.

Definition 8 *S is (δ, k)-isolated neighborhood of node v if:*

- $v \in S$
- *S is connected*
- $|S| \leq k$
- *Number edges connecting S and $\overline{S}$ is $\leq \delta|S|$.*

In hyperfinite graphs, most nodes have (δ, k)-isolated nodes. Why? Because if G is hyperfinite, then we know that there exists some partitioning. However, we need there to be a partitioning for the remaining graph since an algorithm may find a different partition. Luckily, we know that no matter what was removed, due to $H - minor$ free properties, we still have an $H - minor$ free graph that is still hyperfinite.

Algorithm 2 Global Partitioning Algorithm

```
$\pi_1, ..., \pi_n$ be nodes in a random order $p = \phi$
for $i = 1..n$ do
    if $\pi_i$ still in graph then,
        if $\exists(\delta, k)$-isolated neighborhood of $\pi_i$ in remaining graph. then
            $S$=this neighborhood
        else
            $S = \{\pi_i\}$
        end if
        $P = P \cup \{S\}$
        Remove $S$ and adjacent edges from graph
    end if
end for
```

We know that if S is (δ, k)-isolated, then it will contribute $\leq \delta|s|$ edges, which overall means $\leq \delta n$. However, we cannot say the same if $S = \{\pi_i\}$ (one node). But we can show that we will not have too many of these singleton cases.

Lemma 9 *If G' is a subgraph of a (hyperfinite) graph G such that G' has $\geq \delta n$ nodes, then $\leq \frac{\epsilon}{30}$ fraction of nodes in G' do not have (ϵ, k)-isolated neighborhoods.*

The idea for this proof is that since G is $H - minor$ free, then G' is $H - minor$ free, meaning G' is hyperfinite as well. This means there exists a partition such that most nodes in G' are in (δ, k)-isolated neighborhoods. This means that with a high probability, a random π_i will be in a (δ, k)-isolated neighborhood.

4.2 Local Simulation of Partitioning Oracle

We will input v, assume access to a random function $\pi(v)$ such that $\pi : v \to [n]$ and then we will output $P[v]$.

Algorithm 3 Local Simulation

```
for all $w$ such that $\pi(w) < \pi(v)$ and $w$ is distance $\leq 2k$ from $v$ do
    recursively compute $P[w]$
    if $\exists w$ such that $v \in P[w]$ then
        P[v]=P[w]
    else
        look for $(\delta, k)$-isolated neighborhood of $v$ (ignoring nodes which are in $P[w]$ for smaller ranked $w$s)
        if find one then
            $P[v]$= neighborhood
        else
            $P[v] = \{v\}$
        end if
    end if
end for
```

This algorithm makes $d^{\mathcal{O}(k)}$ recursive calls. However, with the bounds we learned in the previous lecture, the runtime is $2^{d^{O(k)}}$. The best possible runtime right now is $d^{\mathcal{O}(\log^2(\frac{1}{\epsilon}))}$.