0368.416701 Sublinear Time Algorithms December 1st, 2014

Lecture 6

Lecturer: Ronitt Rubinfeld *Scribe: Eyal Itkin, Yotam Frank, Tomer Ezra*

Lesson Overview

1. Testing H-minor freeness (Continue from last lecture)
2. Approximating the average degree

1 Testing H-minor freeness (Continue)

1.1 Last Lecture

In the last Lecture we saw that testing for planarity of a graph G implies that G should be:

$$K_{3,3} - free$$

$$K_5 - free$$

This is a private case for the general testing if a graph G is H-m.f. (H is a constant size subgraph). Meaning that:

$$Test\ H - m.f. \rightarrow Testing\ Planarity$$

In addition, we also saw the following:

Definition: G is $(\epsilon, k)-$*Hyperfinite* if can remove $\leq \epsilon n$ edges and remain with C.C of size $\leq k$

Claim: All H-m.f. graphs are hyperfinite

Note: A subgraph G' of G also maintains the hyperfinite property, i.e.

$$G\ hyperfinite \Rightarrow \forall G' \subset G,\ G'\ is\ hyperfinite$$

1.2 Tentative Plan

The tentative plan for our tester will be based on the shown properties:

1. H-m.f. can be used to test planarity
2. An H-m.f. graph is hyperfinite
3. We will try to remove $\leq \epsilon dn$ edges and leave components of size $\leq O\left(\frac{1}{\epsilon^2}\right)$ (Theorem 18 from last lesson)

Meaning, we will check if the hyperfinite property holds.

1.3 Partition Oracle P :

Let P be a *Partition Oracle* , i.e. an oracle that returns the connected component of a given vertex v in the partitioned graph. The partitioned graph is what we get after "breaking" the graph to small connected components.

Input: v

Output: $P[v]$ - name of the connected component

$\forall v \in V$ the following holds:

1. $k := |P[v]| \leq \theta\left(\frac{1}{\epsilon^2}\right)$
2. $\{w|P[w] = P[v]\}$ is a connected component

1.4 Algorithm - 1st Attempt

repeat for '?' times:

1. Pick $v \in V$ randomly. (denoted as $\underset{R}{\in}$ from now on)
2. BFS to find v's connected component
3. check that the component is H-free

Notes:

1. The BFS scan uses the oracle P to scan only v's connected component
2. We return 'reject' if the scan found more than $\theta\left(\frac{1}{\epsilon^2}\right)$ vertices
3. Checking for H-freeness is np-complete, however we can afford a running time of $2^{\theta\left(\frac{1}{\epsilon^2}\right)}$

Claim: if G is H-m.f. with probability ≥ 0.9 then
$C := \left|\{(u,v) \in E|P[u] \neq P[v]\}\right| \leq \frac{\epsilon dn}{4}$

meaning that there are relatively few edges crossing between different connected components. The proof wasn't given, and remained to be completed at home.

1.5 Algorithm - 2nd Attempt

We will add an extra step after building the oracle and before the running of the algorithm itself. The algorithm will now be:

Algorithm:

repeat for '?' times:

1. Pick $v \underset{R}{\in} V$ randomly
2. Estimate the number of edges (u,v) s.t. $P[v] \neq P[u]$
   (a) If the number $\geq \frac{3}{8}\epsilon dn$ 'reject' and halt
3. BFS to find v's connected component

(a) If size of connected component too big - 'reject' and halt

4. check that the component is H-free

   (a) If found an instance of H - 'reject' and halt

return 'accept'

Explanation: The 1st attmept of the algorithm assumed a perfect partition oracle, so there was no need to test it's credibility. This assumption is not practical, and so we overcome it by adding an additional step that tests the oracle - new step (2).

1.5.1 Algorithm Correctness:

If G is H-m.f. than we have two mistake options:

1. $Pr\left[oracle\ had\ a\ mistake\right] \leq 0.1$
2. Else, $Pr\left[estimate\ of\ C \geq \frac{3}{8}\epsilon dn\right] \ll 0.1$

Case 1: The oracle's output was that $C > \frac{\epsilon dn}{2}$, and did not match our claim. If G is H-m.f. than we have two mistake options:

1. $Pr\left[oracle\ had\ a\ mistake\right] \leq 0.1$
2. Else, $Pr\left[estimate\ of\ C \geq \frac{3}{8}\epsilon dn\right] \ll 0.1$

The second point is due to the fact that we can use chernoff to bound the probability of deviating from the expectation of $E\left[C\right] \leq \frac{\epsilon dn}{4} \leq \frac{3}{8}\epsilon dn$. The partition should be made after removing $\leq \epsilon dn$ edges, and so step (2) was added to check the amount of removed edges. If the oracle is correct, than the property does not hold, and if it does holds than this case has a low probability.

Case 2: The oracle's output was that $C \leq \frac{\epsilon dn}{2}$. We will use some more definitions:

Definition: $G' := G\ without\ crossing\ edges$

Definition: we say that G is ϵ-*far from H-m.f* if there needs to be removed $\geq \epsilon dn$ edges

According to the definition of G', any instance of H must be inside a connected component. In addition, G' is also $\frac{\epsilon}{2} - close\ to\ G$ and so:

$$G\ is\ \epsilon - far \Rightarrow G'\ is\ \frac{\epsilon}{2} - far \Rightarrow must\ remove\ \geq \frac{\epsilon dn}{2}\ edges$$

G' is already a set of connected components, and so the removal of these edges will only be needed in order to remove the remainders of H from any connected component. A closer look on the algorithm will show that after passing step (2) we are actually running on G'. We will now use d, maximal degree in the graph, to see that the minimal number of vertices that connected to an instance of H is $\frac{\frac{\epsilon dn}{2}}{d} = \frac{\epsilon n}{2}$. This ensures high probability for the BFS search, step (3), to find an interesting connected component - one that has H in it.

2 Approximate Average Degree

We would like to get an estimate of $\bar{d}$ which is the average degree of vertex v in graph G.

$\bar{d} := \frac{\sum_{u \in V} d(u)}{n}$

G is simple , $\Omega(n)$ edges

degree queries: on $v \in V$ return $d(v)$.

neighbor queries: on (v_{ij}) return j^{th} neighbor of v.

The representation of G is an array of neighbors for every vertex of V.

Observation: The rank of a vertex necessitates edges to other vertices. Therefore we will not have degenerate case like $(n-1, 0, 0, ..., 0)$

Definition:

We define $\beta := \frac{\epsilon}{c}$ and $t := O(\frac{logn}{\epsilon})$ s.t:

$B_i := \{v | (1+\beta)^{i-1} |B_i| \leq d(v) \leq (1+\beta)^i\}$

$B_0 := \{v | d(v) \in \{0\}\}$

These will be the buckets into which we put in the vertices of the graph. From the definition of B_i we will try to estimate the size of buckets...

Total degree of B_i: $(1+\beta)^{i-1}|B_i| \leq d_{B_i} \leq (1+\beta)^i |B_i|$

Total degree of G: $\sum_{i=0}^{t} d_{B_i} \Rightarrow \sum_{i=0}^{t} (1+\beta)^{i-1}|B_i| \leq d_{total} \leq \sum_{i=0}^{t} (1+\beta)^i |B_i|$

Intuition: For each bucket B_i we would like to estimate its size $|B_i|$. Later we will show this approach give rise to some problems.

2.1 Algorithm - 1st Attempt

1. Take sample S.
2. $S_i \leftarrow S \cap B_i$
3. $\alpha_i \leftarrow \frac{|S_i|}{|S|}$ (note $E[\frac{|S_i|}{|S|}] = \frac{|B_i|}{n}$)

Remark: A 'big' bucket is considered a bucket that contributes a sizeable amount to the total degree and does not necessarily have many representatives (a highly populated bucket).

2.2 Algorithm - 2nd Attempt

We will use 0 to estimate the contribution for 'small' buckets.

1. Take sample S.
2. for all i
   (a) if $|S_i| \leq \sqrt{\frac{\epsilon}{n}} \frac{|S|}{ct}$ then $\alpha_i \leftarrow 0$
   (b) else $\alpha_i \leftarrow \frac{|S_i|}{|S|}$
3. Output $\sum_i \alpha_i (1+\beta)^{i-1}$

Remark: notice the difference from the first attempt implies $|S| \gg t * poly(\frac{1}{\gamma}) * \sqrt{\frac{\epsilon}{n}}$.

For overpopulated sample i:

$Output \geq \frac{1-\gamma}{n} * \sum_i (1+\beta)^{i-1} |B_i| \geq \frac{1-\gamma}{1+\beta} * \frac{d_{total}}{n}$

Correct choice of β gives an expression of the form:

$\geq (1-\gamma)(1-\beta)\frac{d_{total}}{n}$

We classify the types of edges in G:

1. large-large := an edge between vertices both in populated buckets
2. large-small := an edge between a vertex in an underpopulated bucket and a vertex in a populated bucket
3. small-small:= an edge between vertices both in underpopulated buckets

How many 'small-small' edges do we have?

Let's assume that for each bucket the following holds:

$|B_i| \leq \sqrt{\frac{\epsilon}{n}} * \frac{2n}{ct}$

It follows that even if all these are connected in one clique, there total number of edges is still bounded by:

$\#edges$ (t buckets) $\leq (\sqrt{\frac{\epsilon}{n}} \frac{2n}{c})^2 \leq \frac{4}{c}\epsilon n = O(\epsilon n)$

lets try and improve the algorithm by estimating better the "small-large" edges

2.3 estimating "large small" edges

We will give a better estimation of the number of "large small" edges from B_i

Algorithm:

Random Neighbor Query(v)

1. get deg(v)
2. pick i $\in$ [1,deg(v)] randomly
3. return neighbor query(v,i)

Almost Random Edge

1. pick v $\in B_i$
2. return (v,Random Neighbor Query(v))

Estime Fraction of "large small" in B_i

1. repeat $O(\frac{1}{\delta})$ times
2. e= Almost Random Edge
3. a_j =1 $\iff$ e is "large small" o.w. 0

4. p_i = averege of a_j

correctness

let T_i be the number of "large small" edges in B_i
let assume that all the nodes have degree d
so the probability that a specific "large small" edge e will be chosen is $\frac{1}{d|B_i|}$
so the probability that any of the "large small" edges will be chosen is $\frac{T_i}{d|B_i|}$
in general:
we can notice that if Pr["large small" edge e in B_i chosen]=p for e spesific edge
then $\frac{1}{|B_i|(1+\beta)^i} \leq p \leq \frac{1}{|B_i|(1+\beta)^{i-1}}$
therefore Pr[any of "large small" edges in B_i chosen]= p' = p#("large small" edges)
so $\frac{T_i}{|B_i|(1+\beta)^i} \leq E[a_j] \leq \frac{T_i}{|B_i|(1+\beta)^i}$
so $E[a_j]|B_i|(1+\beta)^{i-1} \leq T_i \leq E[a_j]|B_i|(1+\beta)^i$
and our algorithm estimates $E[a_j]$ to $(1+\epsilon)$ multiplicative factor
giving $(1+\epsilon)(1+\beta)$ estimate to $\frac{T_i}{n}$ via the next algorithm

final algorithm

1. sample S s.t. $|S| > \sqrt{\frac{n}{\epsilon}}/t$
2. $S_i = S \cap B_i$
3. for all i
   (a) if $|S_i| \geq \sqrt{\frac{\epsilon}{n}}\frac{|S|}{ct}$ then $p_i = \frac{|S_i|}{|S|}$ o.w. $p_i = 0$
   (b) for all $v \in S_i$
      i. pick random neighbor u of v
      ii. if u in underpopulated bucket then $\chi(v) = 1$ o.w. $\chi(v) = 0$
   (c) $\alpha_i = \frac{\sum_{v \in S_i} \chi(v)}{|S_i|}$
4. output $\sum_{largei} p_i(1+\alpha_i)(1+\beta)^{i-1}$