0368.416701 Sublinear Time Algorithms November 24, 2014

Lecture 5

Lecturer: Ronitt Rubenfeld *Scribe: Evgeny Gorokhovsky*

1 Fast Distributed Vertex Cover

A distributed algorithm for finding a Vertex Cover.

Algorithm 1

1: $i \leftarrow 1$
2: **while** edges remain **do**
3: Remove nodes of degree higher than $\frac{d}{2^i}$ and adjacent edges
4: Update degrees of remaining nodes
5: $i \leftarrow i + 1$
6: **end while**
7: **return** Removed nodes

Execution example

Figure 1.1 below shows a result of an execution. Assuming $d = 16$, each node that will be output by the algorithm is shown with the round it was selected (i) and the degree ($d' = \frac{d}{2^i}$) used in that round. Nodes without a label are not in the output.

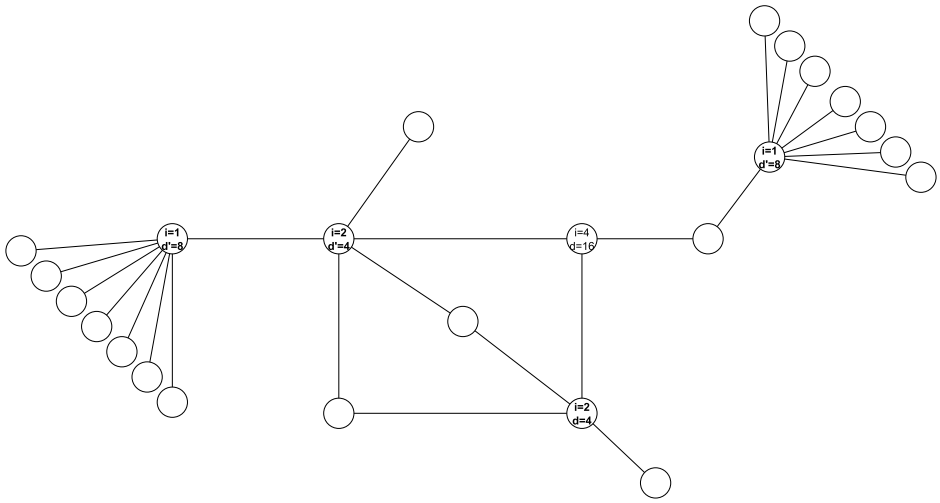

Figure 1.1: Example execution for Algorithm 1

Claim. The number of rounds for the algorithm is $\log d$.

Proof. The highest value for i is $\log d$ since after removing all nodes with deg $\geq \frac{d}{2^{\log d}} = 1$ no edges remain □

Claim. The algorithm output is a Vertex Cover of G

Proof. When the algorithm terminates, no edges remain. An edge (u, v) is removed only when the nodes u or v are removed. We return all removed nodes so all edges are covered. □

Theorem. *Let $VC(G)$ be the minimum size of any vertex cover of G*
then $VC(G) \leq |OUTPUT| \leq (2 \log d + 1) VC(G)$

Claim. Let Θ be some minimal size vertex cover of G. On iteration i the algorithm removes at most $2VC(G)$ nodes not in Θ

Proof. (of claim)
Define the following sets: □

X_i Nodes removed in round i that are not in Θ

$\bar{X}_i$ Nodes removed in round i that are in Θ

Y_i Remaining nodes in round i

Nodes removed in iteration i have a degree in the range $\left[\frac{d}{2^i}, \frac{d}{2^{i-1}}\right)$ since they were not removed in the previous round. So we have that the lower bound on the number of edges exiting X_i is $|X_i| \cdot \frac{d}{2^i}$.
The degree for any node in the graph on round i is less than $\frac{d}{2^{i-1}}$ so the upper bound on the number of edges exiting Θ is $|\Theta| \cdot \frac{d}{2^{i-1}}$.
Since Θ is a vertex cover, for every edge (u, v) s.t $u \in X_i$ and $v \in Y_i$ we have $v \in \Theta$ and therefore $|X_i| \cdot \frac{d}{2^i} \leq |\Theta| \cdot \frac{d}{2^{i-1}}$ and $X_i \leq 2VC(G)$. The total output size is

$$|OUTPUT| \leq \sum_{i=1}^{\log d} |X_i| + \sum_{i=1}^{\log d} \left|\bar{X}_i\right| \leq \sum_{i=1}^{\log d} 2VC(G) + |\Theta| = (2 \log d + 1) VC(G)$$

Since $OUTPUT$ is a vertex cover and $VC(G)$ is the minimal vertex cover we have $VC(G) \leq |OUTPUT|$

Proof. (of theorem)
The first part, follows directly from the above claim. Since the output is a vertex cover, and $VC(G)$is the *minimal* vertex cover $VC(G) \leq OUTPUT$. □

2 Sub linear matching via a greedy algorithm

Definition. A *matching* for graph $G = \langle V, E \rangle$ is a set of edges $(u, v) \in E$ s.t that no node appears more than once in the set.

Definition. A *maximal matching* is a matching that cannot be increased

A greedy algorithm for maximal matching

Algorithm 2 Greedy Sequential Matching

```
1: M ← ∅
2: for all (u, v) ∈ E do
3:     if u and v are not in the matching then
4:         M ← M ⋃ {(u, v)}
5:     end if
6: end for
7: return M
```

Output depends on the order in which we considered the edges.

Claim. The output of the algorithm is a maximal matching

Proof. The algorithm checks all of the edges. If an edge (u, v) was not added to M it means either u or v are already in the matching, therefore the matching is maximal. □

Oracle reduction framework

Assume $\mathcal{O}(e)$ is an oracle that returns YES if $e \in M$ after a run of the greedy algorithm and NO otherwise.

Algorithm 3

$S \leftarrow O\left(\frac{1}{\epsilon^2}\right)$ nodes chosen i.i.d
for all $v \in S$ **do**
    if for any $w \in V$,$\mathcal{O}(e)$ outputs YES **then**
        $X_v \leftarrow 1$
    else
        $X_v \leftarrow 0$
    end if
end for
return $\frac{n}{2|S|}\sum_{v\in s} X_v + \frac{\epsilon n}{2}$

The $\frac{\epsilon n}{2}$ is to counter for the algorithm's possible underestimation. It is also possible to accept an underestimation of up to $\frac{\epsilon n}{2}$ (w.h.p)
This works since $E[OUTPUT] = |M| + \frac{\epsilon n}{2}$ and using an additive Chernoff bound we can get that

$$\Pr\left[|OUTPUT - E[OUTPUT]| \leq \frac{\epsilon}{2}\right] \geq \frac{2}{3}$$

In contrast to a regular Parnas framework where a distributed algorithm is simulated, we will use a sub-linear oracle for the Greedy algorithm

Implementing $\mathcal{O}(e)$

To decide if an edge is in the matching, we only need to know what Greedy did with *lower ordered* edges. An edge e is matched if and only if no lower order adjacent edge was matched.
Assume random distinct ranks r_e assigned to edges $e \in E$.

Algorithm 4 Checking if $e \in M$

for all $e' \in E$ adjacent to E **do**
    if $r_{e'} < r_e$ **then**
        recursively check e'
    else
        ignore e'
    end if
end for
return NO if any e' was matched, YES otherwise

Claim. The number of queries per node made by $\mathcal{O}$ is $2^{O(d)}$

Proof. Query calls in an execution of $\mathcal{O}$ for a single node are in descending order. For some path p of length k $\Pr[p \text{ is explored}] = \frac{1}{(k+1)!}$. The number of paths of length k is at most d^k so

$$E[\#\text{edges explored}] \leq \sum_{k=0}^{n} \frac{d^k}{(k+1)!} \leq \frac{e^d}{d} = 2^{O(d)}$$

□

Corollary. *The total query complexity of using $\mathcal{O}$ with the framework is* $\frac{2^{O(d)}}{\epsilon^2}$

3 Testing H-minor freeness

Definition. A graph H is a *minor* of graph G if it is possible to transform G into H using node removals, edge removals and node contractions

Definition. A graph G is H *minor free* if H is not a minor of G .

Definition. A graph G is ϵ-*close* to being H minor free if it is possible to transform it into an H minor free graph by removing at most ϵdn edges

Definition. A property P is *minor-closed* if for any $G \in P$, H is a minor of G $\Rightarrow H \in P$

Theorem. *(Robertson–Seymour)*
Every minor closed property can be expressed by a constant number of excluded minors.

For example, planarity for graphs can be reduced to verifying that a graph is $K_{3,3}$and K_5 minor

Definition. A graph G is (ϵ, k) *hyperfinite* if it is possible to remove at most ϵn edges to transform G into a graph with components of size at most k

Definition. A graph G is ρ *hyper finite* if for any $\epsilon > 0$, G is $(\epsilon, \rho(\epsilon))$ hyperfinite

All minor free families are hyperfinite.

Theorem. *Given H,* there is a constant C_H s.t for any $0 < \epsilon < 1$ every H minor free graph with maximal degree at most d is$\left(\epsilon d, \frac{C_H^2}{\epsilon^2}\right)$ hyperfinite

In other words, for such graphs, there is no way to remove at most ϵdn edges to get components of size $O\left(\frac{1}{\epsilon^2}\right)$

Algorithm overview

First partition G into G'.

- G' will have only constant size connected components
- We will try to remove at most $\frac{\epsilon}{2} dn$ edges connecting these components
- If we can't partition, it means that G is not H minor free

If G' is close to having the property than so is G , since we removed relatively few edges. We can pick random components and see if the have the property.