0368.416701 Sublinear Time Algorithms November 21, 2014

Lecture 5

Lecturer: Ronitt Rubinfeld *Scribe: Itay Polack, Talya Eden, Guy Braude*

Lesson Overview

Continuing last lecture: comparing distributed algorithms and sub-linear time algorithms

1. Present a distributed algorithm for approximating the size of the graph vertex cover in constant time.
2. Using oracle reduction framework for approximating the size of a graph maximal matching in sublinear number of queries.
3. Testing H-minor freeness.

1 Distributed algorithm for approximating VC in constant time

Last week we saw a connection between distributed algorithms and sublinear time algorithms. The main idea is to run a sublinear time simulation of a distributed algorithm. Notice that for each individual node, distributed algorithm that runs k rounds, is bounded to affect no more than d^k nodes (where d is the graph max degree), as message can pass only from one neighbor to another. We will later use this bound for our advantage. This time we will see an actual distributed algorithm for approximating VC in a given graph with constant runtime that mostly depends on the degree of the graph.
Remark In distributed algorithms, there is no tradional input - the graph itself, containing the nodes (processors) and the edges between them is the input. The runtime is measured in "rounds". During a round, each node can send and receive unlimited sized messages from all its neighbors.

1.1 Algorithm

The following algorithm approximates minimum VC. Let $|VC|$ be the minimum possible VC, and $|VC'|$ the algorithm results, $|VC'| \leq \log(d)|VC| + \epsilon n$ (where ϵ is a parameter, and n is the number of nodes in the graph).

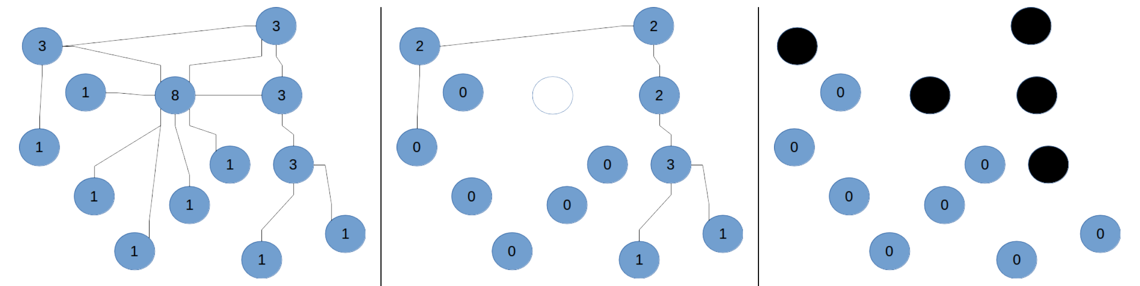
Remark There is a better algorithm giving $|VC'| \leq 2|VC| + \epsilon n$, but we will not discuss it now.

```
Fast distributed VC algorithm
Input: Undirected graph G of max degree d
Output: Approximation of min VC

1. Let $i \leftarrow 1$
2. While any edges remain in the graph:
    (a) Remove vertices where vertex degree $\geq \frac{d}{2^i}$ and all adjacent edges
    (b) Update degrees of all remaining vertices
    (c) Let $i \leftarrow i + 1$
3. Output all removed vertices as VC
```

Observation 1 *The algorithm requires up to $\log d$ rounds: it can only run until there are no edges left, and after $\log d$ rounds we remove all vertices of degree $\frac{d}{2^{\log d}} = 1$, meaning any vertex with edges will be removed with its adjacent edges. The only vertices that will not be part of the output are the ones which degree becomes zero during the algorithm run. It means that at least one adjacent vertex of such vertices was already added to the VC.*

We will see an example for better understanding the algorithm run:
The graph shown has max degree of 8, so we start with removing the vertices of degree 4 and above. Next step would be removing the ones with degree 2 and above, and at this point - we are done. The vertices highlighted in black are the output. Looking carefully at the output, you can see it is not the optimal solution, as the minimum VC at this case is 3.



$i = 1$, remove $d \geq 4$	$i = 2$, remove $d \geq 2$	Done!

Remeber this is a distributed algorithm, meaning - each node is performing the algorithm on itself, updating its neighbors and getting updates from the neighbors. The processing is done in parallel.

Claim 2 *The algoritm outputs a vertex cover.*

Proof By the way the algoritm operates, we remove all edges. Any edge (u, v) is removed either if u or v becomes part of the VC. ■

We will now refer to $|VC|$ as the minimum possible vertex cover for graph G, and $|VC'|$ as the output of the algorithm on the same graph.

Theorem 3 $|VC| \leq |VC'| \leq (2\log d + 1)|VC|$

Notice that $|VC| \leq |VC'|$ is trivial, since $|VC'|$ is a VC, and $|VC|$ is the minimum possible VC. The hard part is upper bound to $|VC'|$.

Claim 4 *Let θ be any minimal vertex cover. In each iteration, the algorithm removes no more than* $2|VC|$ *nodes that are not in θ.*

Motivation: If we can prove this claim, then the size of the algorithm's solution cannot go beyond *the number of rounds* (bounded to $\log d$) times *the number of removed nodes.*
Proof By the algorithm definition, the degree of all vertices removed in round i is between $\frac{d}{2^i}$ and $\frac{d}{2^{i-1}}$ (as vertices with degree $\geq \frac{d}{2^{i-1}}$ were removed in previous rounds, and only vertices with degree $\frac{d}{2^i}$ or higher are removed in the current round). Now, let's take a look on the vertices removed in iteration i of the algorithm.
Define $X \equiv$ All vertices that are not part of θ and removed in iteration i, and $\bar{X} \equiv$ All vertices that are part of θ and removed in iteration i. For each edge (v, u) coming out of X (e.g. $v \in X$), it must be either connected with an edge to another vertex $u \in \bar{X}$, therefore θ, or connected to a vertex that is removed in another iteration and belong to θ (otherwise θ would not be a valid VC). Notice that it's not possible for any vertex in X to be connected to another vertex in X, as the meaning is that we have

a vertex that is not covered by θ which must be a valid vertex cover.
The lower bound on the number of edges coming out of X is $|X|\frac{d}{2^i}$ (number of edges times the minimum degree of each vertex). The upper bound on the number of edges going to θ is $|\theta|\frac{d}{2^{i-1}}$ (number of vertices times the maximum degree).
Now we know the number of edges coming out of X is at most the number of edges going in to θ, therefore: $|X|\frac{d}{2^i} \le |\theta|\frac{d}{2^{i-1}} \Rightarrow |X| \le 2|\theta|$. ■

Corollary 5 *The number of additional vertices is* $\le (2\log d)|VC|$. *Therefore,* $|VC'| \le (2\log d+1)|VC|$.

2 Maximal matchings

Definition 6 *Given a graph $G = (V, E)$, a* ***matching*** *M in G is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex.*

Definition 7 *Maximal matching is a matching in a graph, where it is not possible to add any more edges to the matching.*

Remark Notice that **maximal** matching is not always **maximum** matching.

2.1 A greedy algorithm for maximal matching

A GREEDY SEQUENTIAL ALGORITHM FOR MAXIMAL MATCHING
Input: An undirected graph G of max degree d
Output: A maximal matching $\mathcal{M}$

```
1. Initialize $\mathcal{M} \leftarrow \emptyset$
2. For every edge $e = (u, v) \in E$
   (a) If neither $u$ or $v$ are matched
       $\mathcal{M} \leftarrow \mathcal{M} \cup \{e\}$
3. Output $\mathcal{M}$
```

Remark The greedy algorithm is deterministic, but there could be many outputs depending on the order of edges. Also note that it is *not* sublinear.

2.2 Oracle reduction framework

Let $\mathcal{O}_{\mathcal{M}}$ be an oracle such that $\mathcal{O}_{\mathcal{M}}(e)$ returns whether $e \in \mathcal{M}$ or not, for some arbitrary fixed matching $\mathcal{M}$. Consider the following algorithm for estimating the size of $\mathcal{M}$:
AN ALGORITHM FOR ESTIMATING THE MAXIMAL MATCHING ASSUMING ORACLE ACCESS
Input: An undirected graph G, Oracle access to $\mathcal{O}_{\mathcal{M}}$
Output: An estimate $\widehat{m}$ for the size of the maximal matching $\mathcal{M}$

```
1. $S \leftarrow O(\frac{1}{\epsilon^2})$ nodes chosen independently identically at random
2. $\forall v \in S$ let $\chi_v$
```

$$\chi_v = \begin{cases} 1 & \text{if any call to } \mathcal{O}((v,w)) \text{ for } w \text{ neighbor of } v \text{ returns yes} \\ 0 & \text{otherwise} \end{cases}$$

3. Output $\widehat{m} = \frac{n}{2|S|}\sum_{v \in S} \chi_v + \epsilon n/2$

Claim 8 *With probability $> 2/3$ the algorithm returns $\widehat{m}$ such that $|\mathcal{M}| \leq \widehat{m} \leq |\mathcal{M}| + \epsilon n$.*

Proof Note that the χ_v's are identically distributed Bernoulli random variables with $p = \frac{2|\mathcal{M}|}{n}$. Therefore, by Hoeffding's inequality, with probability $> 1 - 2e^{-2\epsilon^2|S|}$

$$(p-\epsilon)|S| \leq \sum_{v \in S} \chi_v \leq (p+\epsilon)|S|$$
$$(\frac{2|\mathcal{M}|}{n} - \epsilon)|S| \leq \sum_{v \in S} \chi_v \leq (\frac{2|\mathcal{M}|}{n} + \epsilon)|S|$$

Taking $|S|$ to be $\frac{c}{\epsilon^2}$ for $c = 10$ we get that with probability $> 2/3$

$$\widehat{m} - \epsilon n/2 \leq \frac{n}{2|S|}\sum_{v \in S} \chi_v \leq \widehat{m} + \epsilon n/2$$
$$|\mathcal{M}| - \epsilon n/2 \leq \frac{n}{2|S|}\sum_{v \in S} \chi_v \leq |\mathcal{M}| + \epsilon n/2$$

Hence with probability $> 2/3$

$$|\mathcal{M}| \leq \widehat{m} \leq |\mathcal{M}| + \epsilon n.$$

This completes the proof. ■

Fact 9 *If any edge e' adjacent to e and before e in the order is matched, then e is not matched. Otherwise e is matched.*

Therefore in order to determine whether an edge e is matched, we can recursively check if any of its adjacent edges that are before it in the order of edges are matched, and decide accordingly. We call these set of edges the *dependency chain* of e. Hence to implement an oracle $\mathcal{O}_{\mathcal{M}}$ we can only consider the dependency chain of the queried edge e. The problem with this approach is that the dependency chains can be very long with respect to m.

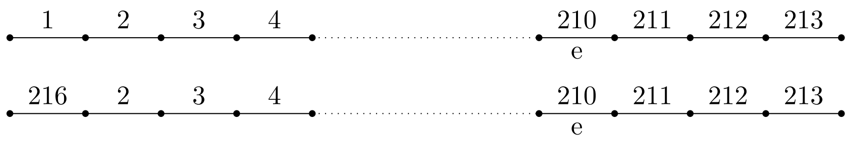


Figure 1: Example for two different rankings that result in long dependency chains. Note that in both cases in order to determine whether the edge marked e belongs to $\mathcal{M}$ the algorithm must recursively check all edges preceding it in the ranking

.

To overcome the difficulty of long dependency chains we assign a random ordering to the edges. We later show that with a random order the expected size of the dependency chains is not too long. Therefore our implementation of the matching oracle $\mathcal{O}_{\mathcal{M}}$ is as follows:

Algorithm for determining whether $e \in \mathcal{M}$
Input: An edge e, and access to a ranking on the edges.
Output: "Yes" if $e \in \mathcal{M}$. "No" otherwise.

```
1. Assume a random ranking on the edges, such that each edge $e$ is assigned a rank $r_e$. Further
   assume that all the ranks are distinct.
2. For all edges $e'$ adjacent to $e$:
   (a) If $r_{e'} < r_e$ recursively check $e'$. If $e' \in \mathcal{M}$ return "No" and halt.
   (b) If $r_{e'} \geq r_e$ continue.
3. Return "Yes".
```

The correctness follows the correctness of the greedy algorithm, since if we would have run the greedy algorithm on edges ordered according to r_e it would output the same maximal matching.

Claim 10 *The expected number of queries preformed by the oracle algorithm is $2^{O(d)}$.*

If the claim is correct then the expected query complexity of algorithm 2.2 is $\frac{2^{O(d)}}{\epsilon^2}$.
Proof Note that the algorithm only looks at paths decreasing in edge ranks.
Therefore, Pr[path p of length k is explored] $= \frac{1}{k!}$, since of all possible $k!$ permutation on the k edges, only one is decreasing in order. Also note that there are at most d^k paths of length k. Therefore

$$E[\text{number of edges explored}] \leq \sum_{k=0}^{\inf} \frac{d^k}{(k)!} \leq e^d \leq 2^{O(d)}.$$

■

3 Testing H-minor freeness

Definition 11 *We say that **H is a minor of G** if you can get H from G by removing nodes or edges, or contract edges.*

Definition 12 *We say that **G is H minor free** (or H-m.f.) if H is not a minor of G.*

Definition 13 *We say that **G is ϵ-close to H-m.f.** if you can remove $\epsilon \cdot n$ edges and get an H-m.f. graph.*

Definition 14 *We say that the property P is **a minor closed property** if $\forall G \in P. \forall H$ such that H is a minor of $G. H \in P$.*
in other words, if G has the property, every minor of G has the property.

Theorem 15 ***(a really cool one)*** *every minor closed property can be expressed as a constent number of excluded minors.*

For example, the property of planer graphs, which is a minor closed property, can be expressed as all the graphs that are 5-click minor free. (see RobertsonSeymour theorem)

Definition 16 ***G is (ϵ,k)-hyper finite*** *if you can remove at most $\epsilon \cdot n$ edges and remain with a graph G' which all its connected component are with size at most k.*

Definition 17 ***G is "ρ-hyper finite"*** *(or ρ-h.f) if $\forall\epsilon > 0$. G is $(\epsilon, \rho(\epsilon))$-hyper finite.*

Theorem 18 *Given a graph H, there exists a constant C_h s.t. $\forall\epsilon . 0 < \epsilon < 1$. every H-m.f graph, which max degree is at most d, is $(\epsilon \cdot d, \frac{C_h}{\epsilon^2})$-h.f.*

This means that removing at most $n \cdot d \cdot \epsilon$ edges will get a graph with connected component with maximum size of $O(\frac{1}{\epsilon^2})$.

Idea for "H-minor free" testing:

- Partition G into G' which has only constant size componants by removing n$\cdot$d$\cdot\epsilon$ edges.
- If it can't be done then G is not H-m.f.
- If G is close to having a property, so is G'.
  So test G' for the property, by picking a random component and search for the property there.