6.5240 Sublinear Time Algorithms September 21, 2022

Lecture 4

Lecturer: Ronitt Rubinfeld *Scribe: Jonathan Rodríguez Figueroa*

In this lecture, we finished the algorithm for estimating the average degree of a graph. Then, we started discussing the similarities between distributed and sublinear algorithms, and how one can use techniques from distributed algorithms to get interesting results on sublinear algorithms.

1 Estimating Average Degree

We start with a few refreshers from last time for reference.

1.1 Assumptions

We make the following assumptions on the input graph:

- The average degree $\overline{d}$ is at least 1.
- We have access to degree queries, which take a vertex x and output its degree $d(x)$.
- We have access to neighbor queries, which take a pair (v, j) and output the j^{th} neighbor of vertex v.
- Every vertex has a unique integer ID assigned to it. Vertices are ordered first by degree, then by label (for vertices with the same degree). We call this ordering "$\prec$", and say that an edge uv is directed from u to v if $u \prec v$.
- We let $\deg^+(v)$ denote the number of out-edges from v. That is, $\deg^+(v)$ gives the number of neighbors larger than v following $\prec$.

1.2 The Algorithm

The following algorithm is used to estimate the average degree of a graph.

Algorithm 1 Approximate Average Degree

```
$k \leftarrow \frac{16}{\varepsilon^2}\sqrt{n}$
**for** $i = 1$ to $k$ **do**
    Pick $v_i \in V$                                   $\triangleright$ (1)
    Pick $u_i \in_u N(v_i)$                            $\triangleright$ (2)
    **if** $v_i \prec u_i$ **then**
        $X_i \leftarrow 2\deg(v_i)$
    **else**
        $X_i \leftarrow 0$
    **end if**
**end for**
**return** $\tilde{d} = \frac{1}{k}\sum_{i=1}^{k} X_i$
```

The analysis follows.

1.3 Analysis

We show here that with probability at least $\frac{3}{4}$, this algorithm gives an output that is within a multiplicative factor of $1+\varepsilon$ from the true average degree.

In the previous lecture, we proved the following three useful claims:

Claim 1 $\sum_u \deg^+(u) = \frac{n}{2}\bar{d}$.

Claim 2 $\forall v$, $\deg^+(v) \le \sqrt{2m}$.

Claim 3 $\mathbb{E}[X_i] = \bar{d}$.

Moving on, we bound the variance to use Chebyshev's inequality to get a bound on the probability that this algorithm does very poorly. Like Chernoff bounds, this is a standard technique in the field. You can often use either of the two techniques to bound the error probability (in fact, you could use Chernoff bounds to bound the error probability of this algorithm), but oftentimes Chebyshev's inequality is cleaner if you can bound the variance.

Claim 4 $\mathrm{Var}[X_i] \le 4\bar{d}\sqrt{2m}$

Proof We know that the variance of a random variable can be expressed as $\mathrm{Var}[X_i] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2$. This quantity is bounded above by $\mathbb{E}[X_i^2]$ since the last term is a square, so we only need to bound this term. By expanding the expected value term and using the definition of X_i, we get the following (note that (1) and (2) refer to the steps in the algorithm):

$$\begin{aligned}
\mathrm{Var}[X_i] &\le \mathbb{E}[X_i^2] \\
&= \sum_{v\in V} \frac{1}{n}\mathbb{E}[X_i^2 \mid v \text{ picked in (1)}] \\
&= \frac{1}{n}\sum_{v\in V}\sum_{u\in N(v)} \Pr[u \text{ picked in (2)} \mid v \text{ picked in (1)}] \cdot \mathbb{E}[X_i^2 \mid u \text{ and } v \text{ picked in (1) and (2)}] \\
&= \frac{1}{n}\sum_{v\in V}\sum_{\substack{u\in N(v)\\ v\prec u}} \frac{1}{\deg(v)} \cdot (2\deg(v))^2 \\
&= \frac{4}{n}\sum_{v\in V} \deg^+(v)\cdot\deg(v) \\
&\le \frac{4}{n}\sqrt{2m}\sum_{v\in V}\deg(v) \\
&= 4\bar{d}\sqrt{2m}
\end{aligned}$$

where the last inequality follows from claim 2. This proves our claim. ■

Observation 5 *$m = \frac{n}{2}\cdot\bar{d}$, so $4\bar{d}\sqrt{2m} = \Theta(n^{1/2}\bar{d}^{3/2})$. This tells us that the variance of X_i is sublinear in n.*

For the rest of the analysis, we will need the following two useful facts about variance

Fact 6 *(Chebyshev's Inequality) For $b > 0$,* $\Pr[|X - \mathbb{E}[X]| \ge b] \le \frac{\mathrm{Var}[X]}{b^2}$

To use this effectively, we need the variance to be small, hence why we try to find an upper bound. We can make the variance smaller bytaking several samples and averaging, thanks to the next fact.

Fact 7 *Let $Y = \frac{1}{k}\sum_{i=1}^{k} X_i$ where the X_i's are independently and identically distributed. Then* $\mathrm{Var}[Y] = \frac{1}{k}\mathrm{Var}[X]$. *(Fun fact, you don't need the variables to be iid; having pairwise independent variables is enough.)*

Now we have everything we need to finish bounding the error probability of our algorithm.

Lemma 8 $\Pr[|\tilde{d} - \overline{d}| \le \varepsilon\overline{d}] \ge \frac{3}{4}$

Proof First, note that $\mathbb{E}[\tilde{d}] = \overline{d}$ by linearity of expectation, so by Chebyshev's inequality (with $b = \varepsilon\overline{d}$), $\Pr[|\tilde{d} - \overline{d}| \ge \varepsilon\overline{d}] \le \frac{\mathrm{Var}[\tilde{d}]}{(\varepsilon\overline{d})^2}$. With Fact 7 and Claim 4, we can bound the variance of $\tilde{d}$ by $\mathrm{Var}[\tilde{d}] = \frac{1}{k}\mathrm{Var}[X_i] \le \frac{1}{k}(4\overline{d}\sqrt{2m})$. Thus the probability is bounded by $\frac{4\sqrt{2m}}{k\varepsilon^2\overline{d}}$. Recall that $n\overline{d} = 2m$, so $\sqrt{n} = \sqrt{\frac{2m}{\overline{d}}}$, so

$$\Pr[|\tilde{d} - \overline{d}| \ge \varepsilon\overline{d}] \le \frac{4\sqrt{n}}{k\varepsilon^2\sqrt{\overline{d}}} \le \frac{4\sqrt{n}}{k\varepsilon^2}$$

where this last inequality is due to the fact that we assume $\overline{d} \ge 1$. Since we chose $k = \frac{16}{\varepsilon^2}\sqrt{n}$, the right hand side of this inequality is at most $\frac{1}{4}$, so in the end $\Pr[|\tilde{d} - \overline{d}| \le \varepsilon\overline{d}] \ge \frac{3}{4}$. ∎

1.4 The Other Algorithm

The algorithm described here is cleaner and more recent than the one generally taught in this course. Here is a broad overview of the other algorithm for estimating average degree.

We first split the nodes into $O_\varepsilon(\log n)$ buckets based on degree, where the min-max ratio of degrees in each bucket is similar. We take $O(\sqrt{n})$ samples of vertices to check their degrees. Use the samples to estimate how many nodes are in each bucket, and take a weighted average using a representative value for each bucket. This will give an approximation for the average degree.

2 Sublinear vs Distributed Algorithms

In this section, we switch gears to look at how general techniques used in distributed algorithms can lead to efficient sublinear time algorithms. We assume the following about our input graphs:

- The graphs are sparse.
- The maximum degree is d.
- We use an adjacency list representation for the graph.

2.1 Overview on Distributed Algorithms

As a guiding example, we look at the *Vertex Cover (VC)* problem. A vertex cover is a subset $V' \subseteq V$ such that for every edge (u, v), either $u \in V'$ or $v \in V'$. The goal is to find a minimum vertex cover of a graph. This problem is NP-hard.

Observation 9 *In any graph with max degree d, any vertex cover must have size at least $\frac{m}{d}$.*

To see this, suppose we have a vertex cover V'. Each vertex of V' covers at most d edges. Thus, $|V'|d$ must cover all edges, so $|V'|d \ge m$.

Side note: there is a polynomial-time 2-approximation for Vertex Cover, where you compute a maximal matching and take the vertices that are endpoints of edges in the matching to be the vertex cover. We know this is a vertex cover because any edge not covered would have two points not in the matching, contradicting its maximality. We can also see that this gives no more than twice the amount of nodes in the minimum vertex cover by looking at the maximal matching. If a minimum vertex cover used less than half of the nodes given by this algorithm, then there would be an edge in the matching not covered by a vertex, so this would not be a vertex cover. Thus, this algorithm gives a 2-approximation. It is also hard to do better than a 1.36-approximation.

To get a sublinear time algorithm, we need to include an additive error, since we cannot distinguish between a graph with 0 edges and one with 1 edge without checking all nodes.

Definition 10 *$\hat{y}$ is an* (α, ε)-approximation *of value y for a minimization problem if $y \leq \hat{y} \leq \alpha y + \varepsilon$. We call α the* multiplicative error *and ε the* additive error*.*

Now we give some background on the LOCAL model of distributed algorithms. A distributed algorithm looks at a network (graph) of processors (nodes) and links (edges). Each processor only knows its neighbors and the maximum degree d. Computation is split into several communication rounds, each one consisting of the following processes:

- Processors can perform any arbitrary computation with the information they have.
- Processors can send messages to their neighbors (the messages can be different for different neighbors).
- Processors can receive messages from their neighbors.

When solving the vertex cover problem for distributed networks, the input graph is the network itself. That is, the processors are figuring out a property of the network that they are a part of. At the end of the computation, each processor should know whether it is part of a specific vertex cover.

2.2 The Parnas-Ron Reduction

This is the main insight in the connection between distributed algorithms and sublinear algorithms. Consider a vertex v during a k-round distributed algorithm. After the first round, v can only use information from its neighbors. After the second round, v can use information from its neighbors and their neighbors. After k rounds, v can only use information from nodes at a distance at most k away, so we can imagine a radius-k ball centered at v.

We simulate the distributed computation in the radius-k ball. Since the maximum degree is d, there are at most d^k nodes in the ball. Thus, if there is a k-round LOCAL algorithm, there is an $O(d^k)$ algorithm (in terms of query complexity).

Given a LOCAL algorithm for Vertex Cover, we can get a sublinear-time approximation. An overview of the algorithm is given here. First, we pick a random vertex v and simulate the distributed algorithm on the radius-k ball centered at v. We repeat this over several rounds for different random v, then return the fraction of vertices that were placed in the vertex cover to get an approximate minimum vertex cover. The details of this algorithm will be given in a later lecture.