6.889 Sublinear Time Algorithms September 8, 2020

Lecture 3

Lecturer: Ronitt Rubinfeld *Scribe: Justin Chen*

1 Outline

In this lecture, we continue the discussion of the approximate average degree problem described in lecture 2. The lecture is split into three parts.

1. Recap
2. A $(2+\epsilon)$-multiplicative approximation algorithm
3. A $(1+\epsilon)$-multiplicative approximation algorithm

2 Estimating the average degree of a graph

2.1 Recap

In lecture 2, we introduced the problem of estimating the average degree of a graph in sublinear time.

2.1.1 Problem statement

Recall that we are given a graph $G = (V, E)$ with $d(u)$ for $u \in V$ being the degree of node u. Let $n = |V|$ and $m = |E|$. Our goal is to estimate the average degree of G, defined as follows.

Definition 1 (Average Degree) *The average degree of a graph $G = (V, E)$ is defined as*

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

For this problem, we assume that G is stored as an adjacency list with an additional degree array mapping nodes $u \in V$ to their degrees $d(u)$. Because of this representation, we have access to both neighbor and degree queries in constant time.

Definition 2 (Neighbor Query) *A neighbor query $nbr(u, j)$ returns the jth neighbor of u for a node $u \in V$.*

(Note that in order for these queries to run in constant time, the entries of the adjacency list would need to be implemented as arrays rather than linked lists.)

Definition 3 (Degree Query) *A degree query $d(u)$ returns the degree of u for a node $u \in V$.*

We will also make two key assumptions in order to simplify the problem. We assume (1) that G is simple (has no parallel edges or self loops) and (2) that $m = \Omega(n)$ (G is not ultra-sparse). Recall from last lecture that if we do not make the second assumption, than we have no hope of providing a constant multiplicative approximation in sublinear time as linear time is needed to distinguish between the cases where $m = 0$ and $m = 1$.

2.1.2 First attempt at an algorithm

At the end of lecture 2, we introduced a first idea for an algorithm for estimating average degree. Recall that the basic idea of taking a sample of nodes and using the sample mean degree as our estimate did not work as, in order to apply Chernoff bounds, we would need to normalize our sample degrees which could have high variance (they can take on any values in the range $[0, n]$). Therefore, we introduced the idea of using *bucketing*: separately estimating nodes with different degrees.

Specifically, let $\beta = \frac{\epsilon}{c}$ and $t = O(\frac{\log n}{\epsilon})$. Let the ith bucket be defined as

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

for $i \in \{0, 1, ..., t-1\}$. (Note that we could make a separate bucket for degree zero nodes or just assume there are none.)

Before we reiterate the algorithm, note a few useful observations Let $d_{B_i} = \sum_{v \in B_i} d(v)$ be the sum of the degrees of all nodes in the ith bucket, and let $d_{total} = \sum_{v \in V}$ be the sum of the degrees of all nodes in the graph. Following directly from the definition of B_i above,

$$(1+\beta)^{i-1}|B_i| < d_{B_i} \leq (1+\beta)^i |B_i|.$$

Therefore,

$$\sum_{i=0}^{t-1} (1+\beta)^{i-1}|B_i| < d_{total} \leq \sum_{i=0}^{t-1} (1+\beta)^i |B_i|.$$

Our first attempt at an algorithm uses this fact to estimate $\bar{d}$ by separately estimating the sizes of each of the buckets.

Algorithm 1 Bucketing

Take a sample $S \subset V$ of nodes
for $i \in \{0, ..., t-1\}$ **do**
  $S_i \leftarrow S \cap B_i$
  $\rho_i \leftarrow \frac{|S_i|}{|S|}$ ▷ Estimating fraction of nodes in $|B_i|$
return $\sum_{i=0}^{t-1} \rho_i (1+\beta)^{i-1}$

Note that each ρ_i is an unbiased estimate of B_i. Let $\sigma_j^{(i)} = 1$ if sample j falls in bucket i and $\sigma_j^{(i)} = 0$ otherwise.

$$\text{Exp}[\rho_i] = \text{Exp}\left[\frac{|S_i|}{|S|}\right] = \text{Exp}\left[\sum_{j=1}^{|S|} \frac{\sigma_j^{(i)}}{|S|}\right] = \frac{|S| * |B_i|/n}{|S|} = \frac{|B_i|}{n}$$

Also note that assuming our estimates of the fractions ρ_i are good, then the estimate returned by this algorithm is an undercounting as we use the lower bound of the bucket degrees $(1+\beta)^{i-1}$.

Observation 4 *The key problem with this algorithm is that for i where $|S_i|$ is small, our estimates of ρ_i will be bad approximations of the fraction of nodes that fall in bucket i.*

We continue by exploring this problem and adapting our algorithm to mitigate it.

2.2 A $(2+\epsilon)$-multiplicative approximation algorithm

Consider the bipartite graph with three nodes on one side of the graph connected to the $n-3$ nodes on the other side of the graph. Let a be the index of the bucket that contains the $n-3$ nodes with degree 3 and let b be the index of the bucket that contains the three nodes with degree $n-3$. Though $|B_a| = n-3$ and $|B_b| = 3$, both buckets contribute $(n-3)*3$ edges to d_{total}.

However, under Algorithm 1 we will likely never see any nodes in B_b in our sample if we have $o(n)$ samples, leading to an underestimate of the average degree by a factor of two. In the unlikely case where we do see a node in B_b, we will likely greatly overestimate the average degree. More generally, when there are small buckets (small in terms of their cardinality) with large degrees, our estimate will have high variance because our estimate of ρ_i for a small bucket B_i will have high variance and changes in ρ_i will have a large impact on our estimate of $\bar{d}$.

The key idea for our second attempt at an algorithm will be to ignore small buckets in our estimate in order to reduce the variance of our estimate. (Importantly, note that whether a bucket is considered small or large depends on our sample and therefore these classifications may change on different runs of the algorithm.)

Algorithm 2 Ignore small buckets

Take a sample $S \subset V$ of nodes
for $i \in \{0, ..., t-1\}$ **do**
  $S_i \leftarrow S \cap B_i$
  if $|S_i| \geq \sqrt{\frac{\epsilon}{n}} * \frac{|S|}{c*t}$ **then** ▷ S_i big
    $\rho_i \leftarrow \frac{|S_i|}{|S|}$ ▷ Estimating fraction of nodes in $|B_i|$ (same as before)
  else ▷ S_i small
    $\rho_i \leftarrow 0$ ▷ Ignoring small buckets
return $\sum_{i=0}^{t-1} \rho_i (1+\beta)^{i-1}$

Intuitively, the reason we set this threshold of $\sqrt{\frac{\epsilon}{n}} * \frac{|S|}{c*t}$ is related to the $\sqrt{n}$ clique example from last lecture. Recall that such cliques can affect the average degree by an additive constant. Therefore, any buckets that appear approximately $\frac{1}{\sqrt{n}}$ fraction of the time cannot be ignored if we wish to get a constant multiplicative approximation. Additionally, it is important that small buckets can be safely ignored without incurring much additional error. As we will show later, the choice of this threshold means that small buckets contribute $O(\epsilon n)$ edges, so we can ignore them and maintain reasonable approximation error.

2.2.1 Picking $|S|$

So far, we have not addressed $|S|$, i.e. the size of our sample. Consider

$$|S| = \Theta(\sqrt{n} * polylog(n) * poly(\frac{1}{\epsilon})).$$

(Note that one of the $\log(n)$ factors and one of the $\frac{1}{\epsilon}$ factors comes from our choice of $t = O(\frac{\log n}{\epsilon})$ buckets.)

This choice of $|S|$ implies that if our algorithm classifies S_i as big, then

$$|S_i| \geq \sqrt{\frac{\epsilon}{n}} * \frac{|S|}{c*t} \geq \Omega(polylog(n) * poly(\frac{1}{\epsilon})).$$

From this, we will assume that with high probability for all i where S_i is classified as big, our approximation of the fraction of nodes in bucket B_i is good:

$$(1-\gamma)\frac{|B_i|}{n} \leq \rho_i \leq (1+\gamma)\frac{|B_i|}{n}.$$

(This follows from applying Chernoff and union bounds.)

Intuitively, we choose this setting of $|S|$ in order to collect enough samples to accurately estimate the size of big buckets. The $\sqrt{n}$ dependence is necessary to guarantee that we will see buckets that appear

a $\frac{1}{\sqrt{n}}$ fraction of the time. In order to apply the union bound for the claim above, we need to show that for each big bucket, the probability of having a "bad" estimate is at most $o(\frac{1}{t}) = o(\frac{\epsilon}{\log n})$. The choice of $|S|$ gives us the $\Omega(polylog(n) * poly(\frac{1}{\epsilon}))$ samples per large bucket which allows us to comfortably apply Chernoff bounds to limit the failure probability to at most $o(\frac{\epsilon}{\log n})$ (an additional $O(\frac{1}{\epsilon})$ factor also comes from the Chernoff bound as we will later need $\gamma < \epsilon$).

2.2.2 Analysis

We will split our analysis into two steps: first showing that the output cannot be too large (overestimation) and then that the output cannot be too small (underestimation).

1. **Can our estimate be too large?**

   First, consider the idealistic case where all of our estimates ρ_i are perfect, i.e. $\rho_i = \frac{|B_i|}{n}$. Then, our estimate cannot be too large:

   $$\sum_{i=0}^{t-1} \rho_i (1+\beta)^{i-1} = \frac{1}{n} \sum_{i=0}^{t-1} |B_i| (1+\beta)^{i-1} \leq \bar{d}.$$

   Realistically, we can assume that $\rho_i \leq (1+\gamma)\frac{|B_i|}{n}$. For large buckets, this assumption comes from our choice of $|S|$ and for small buckets this is true as $\rho_i = 0$. Then, our estimate still cannot be that large:

   $$\sum_{i=0}^{t-1} \rho_i (1+\beta)^{i-1} \leq \frac{1}{n} \sum_{i=0}^{t-1} (1+\gamma)|B_i| (1+\beta)^{i-1} \leq (1+\gamma)\bar{d}.$$

2. **Can our estimate be too small?**

   In the idealistic case where all of our estimates ρ_i are perfect, then

   $$\sum_{i=0}^{t-1} \rho_i (1+\beta)^{i-1} = \frac{1}{n} \sum_{i=0}^{t-1} |B_i| (1+\beta)^{i-1} \geq \frac{1}{n} \sum_{i=0}^{t-1} |B_i| (1+\beta)^{i} (1-\beta) \geq (1-\beta)\bar{d}.$$

   (As $(1-\beta)(1+\beta) \leq 1$)

   In the realistic case, we can assume that for big buckets, $\rho_i \geq (1-\gamma)\frac{|B_i|}{n}$ from our choice of $|S|$. However, for small buckets, we do not count them at all in our estimate and therefore they could raise an issue in terms of undercounting.

In order to quantify how much error we can suffer by ignoring small buckets, it will be useful to classify all edges of the graph into three categories.

1. **big-big**

   Big-big edges are edges where both endpoints belong to large buckets. In our algorithm, these edges are counted from both sides.

2. **big-small**

   Big-small edges are edges where one endpoint belongs to a large bucket and the other to a small bucket. Our algorithm only counts these edges once in the average degree estimate.

3. **small-small**

   Small-small edges are edges where both endpoints belong to small buckets. Our algorithm never counts these edges.

As shown above, for large buckets and therefore for big-big edges, error in our estimates of ρ_i will add only small multiplicative error to our approximation. Undercounting of the big-small edges can add a factor of two multiplicative error to our approximation. We will proceed by showing that undercounting of the small-small edges will only add small multiplicative error.

Consider a bucket where $|B_i| > 2\frac{\sqrt{\epsilon n}}{ct}$. Then, $\text{Exp}\left[|S_i|\right] = |S| * \frac{|B_i|}{n} > 2 * \sqrt{\frac{\epsilon}{n}} * \frac{|S|}{c*t}$. As the expected size of S_i is twice the threshold for being classified as a big bucket, with high probability, S_i will be big (by Chernoff bound). Therefore, by union bound we can assume that for all small buckets B_i, $|B_i| \leq 2\frac{\sqrt{\epsilon n}}{ct}$. So, at most there can be $t * 2\frac{\sqrt{\epsilon n}}{ct}$ nodes in small buckets and the number of small-small edges is at most

$$\left(t * 2\frac{\sqrt{\epsilon n}}{ct}\right)^2 = O(\epsilon n).$$

As we assume that G is not ultra-sparse ($\hat{d} \geq 1$), this ϵn additive error corresponds to a multiplicative error of at most $(1+\epsilon)$.

Combining the cases of our error analysis, we get the following claim on our algorithm.

Claim 5 *Algorithm 2 gives a $(2+\epsilon)$-multiplicative approximation algorithm.*

2.3 A $(1+\epsilon)$-multiplicative approximation algorithm

In this section we will try to reduce the error in estimating big-small edges to get a better approximation factor. As we are already counting both sides of the big-big edges and never see either endpoint of the small-small edges, this is the obvious area for improvement. Specifically, as we never see the small side of big-small edges, we will double count them from the big side.

Observation 6 *Note that in both algorithms presented so far, we have only made use of degree queries and have not made use of neighbor queries.*

First, let's introduce a new type of query (that can be implemented simply using existing queries).

Definition 7 Random Neighbor Query *A random neighbor query $randnbr(v)$ returns a random neighbor of v for any node $v \in V$.*

We can implement random neighbor queries by the following.

1. Run the degree query $d(v)$
2. Pick a random index $j \in \{1, ..., d(v)\}$
3. Return the neighbor query $nbr(v, j)$

We will use this new query to sample an (almost) random edge from a given bucket B_i.

1. Sample nodes until we fall into B_i, call this node v
2. Return the random neighbor query $randnbr(v)$

If all nodes in B_i had the same degree, this would correspond to sampling a random edge from B_i. However, as the nodes in B_i can have slightly different degrees (by a factor of $(1+\beta)$), the sampling distribution is not quite uniform over all edges in B_i (edges of lower degree nodes are more likely to be sampled). As this difference is slight, we will assume that edges are randomly sampled.

Now, we will use this protocol to estimate the fraction of edges in a big bucket B_i that are big-small edges.

- For $j = \{1, ..., O(\frac{1}{\delta})\}$

1. Pick a random edge e in B_i
2. Set $a_j = 1$ if e is a big-small edge and $a_j = 0$ otherwise

- Output $\alpha_i =$ average a_j

In terms of implementation, let's assume that this protocol is run after enough samples are taken to determine which buckets are big and which are small. Then, determining if a given edge $e = (u, v)$ where $u \in B_i$ is a big-small edge can simply be done by taking the degree query $d(v)$ to figure out which bucket v belongs to.

To analyze how good of an estimate α_i is for the fraction of big-small edges in B_i, first consider the idealized case where all nodes in B_i have the same degree d. Let T_i be the number of big-small edges in B_i. From our protocol for picking random edges,

$$\Pr[\text{A specific edge } e \text{ in } B_i \text{ is chosen}] = \frac{1}{|B_i|} * \frac{1}{d}.$$

Then,

$$\text{Exp}[a_j] = \frac{T_i}{|B_i| * d}.$$

In the general case where the nodes in B_i can have slightly different degrees, we can limit the expected value of a_j to a small range:

$$\frac{T_i}{|B_i| * (1+\beta)^i} \leq \text{Exp}[a_j] \leq \frac{T_i}{|B_i| * (1+\beta)^{i-1}}.$$

Now, we have the tools to present our final algorithm, where we use these estimates for the fraction of big-small edges in a big bucket B_i to double count those edges.

Algorithm 3 Double count big-small edges

```
Take a sample S ⊂ V of nodes
for i ∈ {0, ..., t − 1} do
    S_i ← S ∩ B_i
    if |S_i| ≥ sqrt(ε/n) * |S|/(c*t) then                    ▷ S_i big
        ρ_i ← |S_i|/|S|
        for all v ∈ S_i do
            Pick a random neighbor of v, u ← randnbr(v)
            X(v) = 1 if u belongs to a small bucket, X(v) = 0 otherwise
        α_i ← |{v ∈ S_i | X(v) = 1}|/|S_i|     ▷ Estimate of the number of edges in B_i that are big-small edges
    else                                                     ▷ S_i small
        ρ_i ← 0
return Σ_{i=0}^{t−1} ρ_i(1 + α_i)(1 + β)^{i−1}     ▷ (1 + α_i) factor double counting the big-small edges
```

The approximation error of this algorithm comes from three factors: error in estimating the ρ_i's, error in estimating the α_i's, and ignoring the small-small edges. The errors in estimating the ρ_i's and α_i's result in multiplicative error while ignoring the small-small edges results in additive error on the order of ϵn (which can be rewritten as multiplicative error as G is not ultra-sparse).

In combination, this algorithm gives an $(1 + \epsilon)$-multiplicative approximation for average degree.