6.5240 Sublinear Time Algorithms September 12, 2022

Lecture 1

Lecturer: Ronitt Rubinfeld *Scribe: Jeffery Li*

1 Outline

Today we had our first discussion of sublinear time algorithms! The general outline of the lecture was as follows:

- Overview of Course and Sublinear Algorithms
- Diameter of a Point Set
- Number of Connected Components in a Graph

We refer the reader to the slides on the course homepage for the first item.

2 Diameter of a Point Set

Suppose we are given m points, numbered from 1 to m, described by an $m \times m$ distance matrix $\mathcal{D}$, such that each entry $\mathcal{D}_{i,j}$ is the distance from i to j. We are given the following guarantees:

- *Symmetry*: $\mathcal{D}_{i,j} = \mathcal{D}_{j,i}$ for all $i, j \in \{1, \ldots, m\}$.
- *Triangle inequality*: $\mathcal{D}_{i,j} \leq \mathcal{D}_{i,k} + \mathcal{D}_{k,j}$ for all $i, j, k \in \{1, \ldots, m\}$.

Note that the input size is $n = \Theta(m^2)$, as we are given m^2 distances.

We define the *diameter* to be the largest element $\mathcal{D}_{i,j}$ of the matrix $\mathcal{D}$. We wish to output k, l such that $\mathcal{D}_{k,l} \geq \mathcal{D}_{i,j}/2$; we call this a **2-multiplicative approximation**, as our answer is within a factor of two of the exact answer.

2.1 Algorithm

Here, we present a basic algorithm. This algorithm will be the only *deterministic* algorithm we present in this class.

Algorithm 1: 2-Multiplicative Diameter Estimator

Input : An $m \times m$ distance matrix $\mathcal{D}$, such that each entry $\mathcal{D}_{i,j}$ is the distance from i to j, and $\mathcal{D}$ satisfies the triangle inequality and symmetry.

Output: A 2-multiplicative approximation of the diameter of the point set.

1 Pick a point $k \in \{1, \ldots, m\}$ arbitrarily.

2 Compute ℓ that maximizes $\mathcal{D}_{k,\ell}$, by looking through the row of the matrix $\mathcal{D}$ corresponding to k.

3 Output $k, \ell, \mathcal{D}_{k,\ell}$.

Runtime - The most computationally expensive step is step two, which involves looking through one row of the matrix $\mathcal{D}$, consisting of m entries. This requires $O(m)$ time, meaning that the total runtime is $O(m) = O(n^{1/2})$, i.e., the square root of the size of the input!

Correctness - Suppose the real diameter is $\mathcal{D}_{i,j}$, for some (possibly other) nodes i, j. Then:

$$\begin{aligned} \mathcal{D}_{i,j} &\leq \mathcal{D}_{i,k} + \mathcal{D}_{k,j} && \text{by Triangle inequality} \\ &\leq \mathcal{D}_{k,i} + \mathcal{D}_{k,j} && \text{by Symmetry} \\ &\leq \mathcal{D}_{k,\ell} + \mathcal{D}_{k,\ell} && \text{by maximality of } \mathcal{D}_{k,\ell} \\ &= 2\mathcal{D}_{k,\ell}, \end{aligned}$$

or that $\mathcal{D}_{k,\ell} \geq D_{i,j}/2$. This means that $\mathcal{D}_{k,\ell}$ is at least half the diameter!

Note that the fact that we are using a metric space is important — we needed to use the triangle inequality and symmetry to show that our output is indeed a 2-multiplicative approximation. We will see many more examples later where we make use of specific properties of the input to achieve better algorithmic results than the naive approach!

3 Number of Connected Components in a Graph

Suppose we are given a graph G, with $|V| = n$, $|E| = m$, and maximum degree d (i.e., no vertex has more than $d + 1$ edges adjacent to it), and another parameter ε. Suppose G is given in adjacency list format.

We wish to output an εn**-additive approximation** of the number of connected components c of G. In other words, we want to output y such that $c - \varepsilon n \leq y \leq c + \varepsilon n$.

3.1 Notation

We will define more notation for this problem, to give an alternate characterization of the number of connected components of G. For all $v \in V$, define n_v to be the number of nodes in v's component. Observe that for all connected components $A \subseteq V$,

$$\sum_{u \in A} \frac{1}{n_u} = \sum_{u \in A} \frac{1}{|A|} = 1.$$

This means that, summing over all connected components $A \subseteq V$,

$$\sum_{u \in V} \frac{1}{n_u} = \sum_{A \subseteq V} \sum_{u \in A} \frac{1}{n_u} = \sum_{A \subseteq V} 1 = c.$$

Thus, the number of connected components c is exactly the sum $\sum_{u \in V} \frac{1}{n_u}$.

At first glance, this does not look like an improvement. It seems like we would need to spend more time to compute this sum because the n_u are hard to compute, especially for linear-sized components!

We will now show how to estimate the n_u quickly, and then estimate the number of connected components using sampling bounds. Intuitively, we will set a "cutoff," independent of n, for our estimate of n_u, so that if we ever find out that a component has too many vertices, we will report this cutoff value.

3.2 Subroutine

Here, to estimate $\frac{1}{n_u}$, we define

$$\widehat{n_u} := \min\left\{n_u, \frac{2}{\varepsilon}\right\}, \text{ and } \quad \widehat{c} := \sum_{u \in V} \frac{1}{\widehat{n_u}}.$$

First, we note the following:

Lemma 1 *For all u, $\left|\frac{1}{\widehat{n_u}} - \frac{1}{n_u}\right| \leq \frac{\varepsilon}{2}$. As a result,*

$$|\widehat{c} - c| \leq \frac{\varepsilon n}{2}.$$

Proof Note that n_u is always positive, so $\frac{1}{n_u} > 0$. We now perform casework:

- If $n_u < \frac{2}{\varepsilon}$, then $\widehat{n_u} = n_u$, so the difference $\frac{1}{\widehat{n_u}} - \frac{1}{n_u} = 0$.
- Otherwise, if $n_u \geq \frac{2}{\varepsilon}$, then $\widehat{n_u} = \frac{2}{\varepsilon}$. Reciprocating the inequalities, we get

  $$0 < \frac{1}{n_u} \leq \frac{1}{\widehat{n_u}} = \frac{\varepsilon}{2};$$

  since both quantities are in the interval $(0, \varepsilon/2]$, their absolute difference must be at most $\frac{\varepsilon}{2} - 0 = \frac{\varepsilon}{2}$.

Thus, in all cases, $\left|\frac{1}{\widehat{n_u}} - \frac{1}{n_u}\right| \leq \frac{\varepsilon}{2}$. Summing over all u and using the triangle inequality, we see that

$$|\widehat{c} - c| = \left|\sum_{u \in V} \frac{1}{\widehat{n_u}} - \frac{1}{n_u}\right| \leq \sum_{u \in V} \left|\frac{1}{\widehat{n_u}} - \frac{1}{n_u}\right| \leq \sum_{u \in V} \frac{\varepsilon}{2} = \frac{\varepsilon n}{2},$$

as desired. ■

Now, we note that computing $\frac{1}{\widehat{n_u}}$ is less time-consuming!

Algorithm 2: Algorithm to Compute $\widehat{n_u}$

Input : A graph $G = (V, E)$, in adjacency list format, and a vertex u.
Output: The value $\widehat{n_u} = \min\left\{n_u, \frac{2}{\varepsilon}\right\}$.

1 Perform BFS from u until one of the following two conditions:
2 We visit $\frac{2}{\varepsilon}$ distinct nodes (i.e., $n_u \geq \frac{2}{\varepsilon}$), in which case we output $\frac{2}{\varepsilon}$.
3 We visit all $n_u < \frac{2}{\varepsilon}$ vertices of u's component, in which case we output n_u.

Runtime - Note that we take $O(d)$ time per BFS step, as each vertex has degree at most d, and we visit $O(\frac{1}{\varepsilon})$ vertices in our BFS. Thus, our overall runtime is $O(d \cdot \frac{1}{\varepsilon})$. Note that we technically do not need the factor of d (and we could instead replace it with a factor of $O(\frac{1}{\varepsilon})$, say), but we will keep it since we need it later on (particularly for other graph representations).

3.3 Algorithm

Now, how do we estimate $\sum_u \frac{1}{\widehat{n_u}}$? The algorithm is as follows:

Algorithm 3: εn-Additive Connected-Component Estimator

Input : A graph $G = (V, E)$, in adjacency list format.
Output: An εn-additive approximation of the number of connected components of G, with good constant probability (say with probability at least $\frac{3}{4}$).

1 $r \leftarrow b/\varepsilon^3$ for some constant b.
2 Choose r nodes $U = \{u_1, \ldots, u_r\}$ independently and uniformly at random from V.
3 For all $u_i \in U$, compute $\widehat{n_{u_i}}$ via Algorithm 2.
4 Output $\tilde{c} = n \cdot \left(\frac{1}{r} \sum_{u_i \in U} \frac{1}{\widehat{n_{u_i}}}\right)$.

(Note that in our last step, we are estimating the average value of $\frac{1}{\widehat{n_u}}$ based on our sample, and renormalizing based on the number of vertices.)

Runtime: Since we chose $O(\frac{1}{\varepsilon^3})$ nodes and ran $O(\frac{d}{\varepsilon})$-time BFS on each of the nodes in step three, and none of the other steps are computationally expensive, this algorithm takes time $O(\frac{d}{\varepsilon^4})$ overall.

Accuracy/Correctness: We now have to recall the **Chernoff bounds** — super powerful but has a lot of moving parts (and similar to Hoeffding's inequality and binomial coefficient bounds). Intuitively, what this says is that, as the range of values gets smaller and smaller and our error range gets larger and larger, we get exponentially more powerful guarantees. More formally, we have the following:

Theorem 2 (Chernoff Bounds) *Let $\delta \in [0, 1]$. Suppose we have $X_1, X_2, \ldots, X_r$, independent, identically distributed (i.i.d.) random variables in the range $[0, 1]$ with $\mathbb{E}[X_i] = p$. Let $S = \sum_{i=1}^{r} X_i$. Then*

$$\mathbb{P}\left[\left|\frac{S}{r} - p\right| \geq \delta p\right] \leq \exp(-\Omega(rp\delta^2)).$$

Note that some versions restrict the X_i to be $0 - 1$ random variables; here we use the more general version.

If we apply the Chernoff bound here, note that

$$p = \mathbb{E}[X_i] = \frac{1}{n} \sum_{u \in V} \frac{1}{\widehat{n_u}} = \frac{\widehat{c}}{n},$$

$\delta = \frac{\varepsilon}{2}$, and $\frac{S}{r} = \frac{\tilde{c}}{n}$. Since $\frac{\varepsilon}{2} \leq \frac{1}{\widehat{n_u}} \leq 1$ for all $u \in V$,

$$\frac{\varepsilon n}{2} \leq \widehat{c} \leq n \implies \frac{\varepsilon}{2} \leq \frac{\widehat{c}}{n} = p \leq 1,$$

so that

$$\begin{aligned}
\mathbb{P}\left[|\tilde{c} - \widehat{c}| \geq \frac{\varepsilon \hat{c}}{2}\right] &= \mathbb{P}\left[\left|\frac{\tilde{c}}{n} - \frac{\widehat{c}}{n}\right| \geq \frac{\varepsilon \hat{c}}{2n}\right] \\
&= \mathbb{P}\left[\left|\frac{\tilde{c}}{n} - \frac{\widehat{c}}{n}\right| \geq \delta p\right] \\
&\leq \exp\left(-\Omega\left(\frac{b}{\varepsilon^3} p \delta^2\right)\right) \\
&= \exp\left(-\Omega\left(\frac{b}{\varepsilon^3} \cdot \frac{\widehat{c}}{n} \cdot \left(\frac{\varepsilon}{2}\right)^2\right)\right) \\
&\leq \exp\left(-\Omega\left(\frac{b}{\varepsilon^3} \cdot \frac{\varepsilon}{2} \cdot \left(\frac{\varepsilon}{2}\right)^2\right)\right) = \exp(-\Omega(b/8)).
\end{aligned}$$

Setting b large enough will give us a good constant probability (i.e., at least $\frac{3}{4}$) that $|\tilde{c} - \widehat{c}| \leq \frac{\varepsilon \hat{c}}{2}$, or that

$$|\tilde{c} - c| \leq |\tilde{c} - \widehat{c}| - |\widehat{c} - c| \leq \frac{\varepsilon \hat{c}}{2} + \frac{\varepsilon n}{2} \leq \varepsilon n.$$

Combining all of our work, this means that our algorithm will return an εn-additive approximation of the number of connected components of G with good constant probability, as desired.

This is part of the reason why we chose $\frac{2}{\varepsilon}$ as the cutoff for our estimates for the sizes of connected components and $r = \frac{b}{\varepsilon^3}$ as the size of our sample. Not only do these values help make our runtime independent of n while still maintaining εn additive error, they also help lower bound r and p in the Chernoff bounds, which in turn help us get a small constant probability of error in our last few steps when applying the Chernoff bounds.

Note that one can turn this into an algorithm with "with high probability" guarantees (i.e., error probability is at most $\frac{1}{\text{poly}(n)}$) with slight increase in runtime (in particular, the runtime becomes dependent on n); see Homework 0 for more details.

Also, it turns out that the runtime can be driven down to $\tilde{O}(d\frac{1}{\varepsilon^2})$.