6.5240 Sublinear Time Algorithms September 4, 2024

Lecture 1

Lecturer: Ronitt A. Rubinfeld *Scribe: Alek Westover*

The content of this class is follows:

- A brief discussion in Section 1 of a very simple sublinear time algorithm for estimating the diameter of a point set under some metric.
- A more involved discussion in Section 2 of sublinear time algorithms for estimating the average degree of a graph.

1 A Simple Example: Estimating the Diameter

In this section we give a simple sublinear time algorithm for the following problem.

2-Approximate Diameter Problem

- **Input:** $\sqrt{n} \times \sqrt{n}$ matrix D encoding a metric with D_{ij} storing the distance between points i and j.
- **Output:** A value $D_{i_*j_*}$ satisfying $D_{i_*j_*} \geq \max_{ij} D_{ij}/2$.

Remark "D encodes a metric" means that D satisfies the triangle inequality (i.e., $D_{ij}+D_{jk} \geq D_{ik}$ for all i, j, k) and symmetry (i.e., $D_{ij} = D_{ji}$ for all i, j). Also, note that we cannot hope to exactly compute the diameter in sublinear time. As a specific counter example, the following two distance metrics cannot be reliably distinguished in sublinear time:

- $D_{ij} = 100$ for all i, j.
- $D_{ij} = 100$ for all i, j, except that for one random pair of distinct points i_*, j_* we have $D_{i_*j_*} = D_{j_*i_*} = 101$.

This motivates settling for an approximation to the diameter.

Now we give an algorithm to solve the 2-Approximate Diameter Problem.

Algorithm to 2-Approximate the Diameter:
Output $\max_k D_{1k}$.

Proposition 1 *The proposed algorithm outputs a 2-approximation of the diameter in time $O(\sqrt{n})$.*

Proof The algorithm scans one row of the matrix; this requires time $O(\sqrt{n})$. Now, fix i, j maximizing $D_{i,j}$. By the triangle inequality and symmetry we have:

$$D_{ij} \leq D_{i1} + D_{j1} \leq 2 \max_k D_{1k}.$$

That is, our algorithm outputs a value which is a 2-approximation of the diameter. ■

2 A More Complicated Example: Estimating the Average Degree of a Graph

Now we turn to a harder problem: estimating the average degree of a graph. More formally, the problem is described as follows:

Average Degree Estimation Problem

- **Parameters**: We are given a ***confidence parameter*** δ and an ***approximation parameter*** ε, which specify our allowed failure probability and allowed multiplicative approximation error.
- **Input:** an n vertex m edge graph, encoded as an adjacency list (i.e., we can look up the j-th neighbor of a vertex v in constant time), along a list mapping vertices to their degrees (i.e., we can look up $\deg(v)$ for any v in constant time).
- **Output:** A value $\tilde{d}$ satisfying

$$\Pr[|\tilde{d} - \overline{d}| \leq \varepsilon \overline{d}] \geq 1 - \delta,$$

where

$$\overline{d} = \frac{1}{n} \sum_{u \in V} \deg(u) = 2m/n.$$

Remark We generally won't optimize δ too much, because for most problems we can boost the success probability by independent repetitions of our algorithm.

Obstacles and Lower Bounds.
One natural approach to estimating the degree is to sample random vertices and average their degrees. This gives an unbiased estimator for the average degree (i.e., this estimator has the correct expectation). However, the variance of this estimator can be extremely bad. For instance, suppose your graph was a star. Then, the vast majority of the time our average degree estimate would be 1, and very rarely we would give an extremely large estimate of the average degree. So the naive sampling approach doesn't work very well. This shows that sampling might not be a good estimation algorithm.

Now we show some lower bounds against *any* algorithm. These lower bounds will motivate us to refine our problem statement to make the problem more tractable (by avoiding these cases where we can't do anything interesting).

The Ultra-Sparse Lower Bound: $\Omega(n)$ queries are necessary to distinguish between the empty graph and a graph with a single edge.

Average Degree 2 Example: Let G_1 be an n cycle. Let G_2 be an $n - \sqrt{2\varepsilon n}$ cycle union a $\sqrt{2\varepsilon n}$ clique (assume that these quantities are integers). The average degree in G_1 is 2, whereas the average degree in G_2 is approximately (up to low order terms):

$$\frac{1}{n}(2(n - \sqrt{2\varepsilon n}) + \sqrt{2\varepsilon n} \cdot (\sqrt{2\varepsilon n} - 1)) \approx 2(1 + \varepsilon).$$

However, distinguishing between G_1, G_2 is difficult. Random sampling requires $\Omega(\sqrt{n/\varepsilon})$ tries (in expectation) to obtain a high degree vertex in G_2, or to be somewhat confident in G_1 that

there are no high degree vertices. One might hope to find (or rule out the existence of) a high degree vertex faster than random sampling using the connectivity structure of the graph. Unfortunately, traversing $o(n)$ edges in the cycle will not help find high degree vertices, because cycles of distinct lengths locally look identical.

Handling the Lower Bounds We will assume our way out of these difficulties. Specifically we will assume $\overline{d} > 1$; this gets rid of the ultra-sparse lower bound. We still should be mindful of the second lower bound however.

The average degree 2 obstacle presented above, along with the example of a star, highlight why the naive sampling based approach doesn't work: the variance is too high. However, if we assume that the degrees lie in a range of bounded multiplicative width then we can control the variance. Specifically, we will show:

Proposition 2 *Fix $\varepsilon, \delta > 0, \Delta \in \mathbb{N}$. Assume each vertex v in G has degree in the range $[\Delta, 10\Delta]$. There is a randomized algorithm with running time $O(\varepsilon^{-2}\ln(1/\delta))$ that, with probability at least $1-\delta$, outputs a value $\tilde{d} \in [(1-\varepsilon)\overline{d}, (1+\varepsilon)\overline{d}]$, where $\overline{d}$ denotes the average degree of G.*

Proof Set $k = \frac{50}{\varepsilon^2}\ln(2/\delta)$.
Our algorithm is as follows:

1. Sample vertices $w_1, \ldots, w_k$ uniformly and independently at random from V.
2. Output $\tilde{d} = \frac{1}{k}\sum_{i=1}^{k} \deg(w_i)$.

This algorithm clearly has the advertised running time; Note that there is no dependence on Δ or n in the running time — this is great! Now, we analyze the correctness of this procedure. First we analyze the expectation of our estimate.

Claim 3 $\mathbb{E}[\tilde{d}] = \overline{d}$.

Proof We use linearity of expectation.

$$\mathbb{E}[\tilde{d}] = \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k}\deg(w_i)\right] = \mathbb{E}[\deg(w_1)] = \overline{d}.$$

■

Now we analyze the variance of our algorithm. We will use the following fact.

Fact 4 (Chernoff Bound) *Let $Y_1, \ldots, Y_k$ be independent random variables with $0 \le Y_i \le 1$ deterministically. Define $Y = \sum_{i=1}^{k} Y_i$. Then, for any $b \ge 1$ we have*

$$\Pr[|Y - \mathbb{E}[Y]| > b] \le 2e^{-2b^2/k}.$$

To apply the Chernoff bound in our setting, we need to first normalize the degrees. Define

$$Z_i = \frac{\deg(v_i)}{10\Delta}, \qquad Z = \sum_{i=1}^{k} Z_i = \frac{k}{10\Delta}\tilde{d}.$$

By our assumed bound on the degrees we have $Z_i \in [0,1]$ for all i. Now, using the Chernoff bound (Theorem 4) we have:

$$\Pr[|Z - \mathbb{E}[Z]| > \frac{k}{10\Delta}\varepsilon\overline{d}] \le 2e^{-2\frac{k^2}{100\Delta^2}\varepsilon^2(\overline{d})^2/k} \tag{1}$$

Applying the bound $\overline{d} \geq \Delta$ — which holds because the average degree is at least the lower bound on degree — in Equation (1) gives:

$$\Pr[|Z - \mathbb{E}[Z]| > \frac{k}{10\Delta}\varepsilon\overline{d}] \leq 2e^{-k\varepsilon^2/50} \leq 2e^{-((50/\varepsilon^2)\ln(2/\delta))\varepsilon^2/50} \leq 2e^{-\ln(2/\delta)} = \delta. \tag{2}$$

Finally, observe that by Theorem 3 we have

$$|Z - \mathbb{E}[Z]| > \frac{k}{10\Delta}\varepsilon\overline{d} \iff |\tilde{d} - \overline{d}| > \varepsilon\overline{d}.$$

The moral of the story is that bounded random variables have bounded variance. ■