CS 514: Advanced Algorithms II – Sublinear Algorithms **Rutgers: Fall 2021**

Lecture

September 14, 2021

Instructor: Sepehr Assadi *Scribe: Sepehr Assadi*

Topics of this Lecture

1 Sublinear Time Algorithms for Graphs **1**

1.1 Query Model for Graph Problems 1

2 Estimating Number of Connected Components **2**

2.1 Proof of Correctness 4

2.2 Runtime Analysis 5

2.3 Concluding Remarks 5

3 Estimating Average Degree **6**

3.1 Warm Up: Almost-Regular Graphs 6

3.2 General Case 7

3.3 Amplifying the Probability of Success 10

1 Sublinear Time Algorithms for Graphs

We are going to study sublinear time algorithms in this and the next couple of lectures. In this lecture, we will focus on sublinear time *graph* algorithms. Before we start, a quick notation is in order.

Notation. For any vertex $G = (V, E)$, we use $n = |V|$ and $m = |E|$ to denote the number of vertices and edges, respectively. For any vertex $v \in V$, $N(v)$ denote the set of neighbors of v in G and $\deg(v) = |N(v)|$ is the degree of v. We also recall the following basic equation: $\sum_{v \in V} \deg(v) = 2m$ (the 'handshaking lemma').

1.1 Query Model for Graph Problems

When designing sublinear time algorithms, specifying the exact data model, or rather the *query model*, is crucial as the algorithm cannot even read the entire input once[1]. A query model then specifies what type of queries can be made to the input or in other words, how one should expect to receive the input to the algorithms (often times, we assume a query takes $O(1)$ time).

In the context of graph problems, we typically work with one of the following models: *adjacency list* model, *adjacency matrix* model, or the *general query* model. In each model, we assume that the graph $G = (V, E)$

[1] In the classical setting also specifying the input access is important; however, one can typically change di erent types of access in time *line* r in the input size and so this does not form a barrier for classical algorithms.

has *known* vertices $V = \{1, \ldots, n\}$ (so $\text{ID}(v) \in \{1, \ldots, n\}$ for any $v \in V$) but the edges are *unknown*. Each model then specifies how one can access the edges of the graph.

Adjacency list query model: The following queries can be answered in $O(1)$ time in this model:

Degree queries: Given a vertex $v \in V$, output $\deg(v)$, namely, the degree of v.

Neighbor queries: Given a vertex $v \in V$ and $i \in [n]$, output the i-th neighbor of v or $\perp$ if $i > \deg(v)$.

By storing the graph in the adjacency list format, we can implement the above query model for algorithms.

Adjacency matrix query model: The following queries can be answered in $O(1)$ time in this model:

Pair queries: Given two vertices $u, v \in V$, output whether (u, v) is an edge in G or not.

By storing the graph in the adjacency matrix format, we can implement the above query model for algorithms.

General query model for graphs: This model is simply a combination of both models above that allows all the three queries mentioned above. This query model can be implemented by storing both the adjacency list and the adjacency matrix of the graph separately.

Remark. The three models above are the most standard models for graph problems. However, sometimes one can consider extensions of these models, for instance, by allowing an extra *edge-sample* query that returns an edge uniformly at random from the graph.

Additionally, the query models we discussed are considered *local* queries as they answer "local" information about the graph (typically functions of local neighborhood of a single vertex). Researchers have also studied *global* query models that answer much more global information: for instance, given a set of vertices, return the number of edges with both endpoints in the set. We will talk about global queries later in the course and for now only mention that power of local and global queries are vastly different; there are various problems that can be solved much faster when one has access to these global queries.

2 Estimating Number of Connected Components

We start with one of the most classical problems in the area of sublinear time graph algorithms, namely, estimating the number of connected components, studied first by Chazelle, Rubinfeld, and Trevisan [1], in the earliest stages of the field of sublinear time algorithms. The problem is as follows:

Problem 1 (**Estimating number of connected components**)**.** Given a graph $G = (V, E)$ in the adjacency list query model, approximation parameter $\varepsilon \in (0, 1)$, and confidence parameter $\in (0, 1)$, output an approximate number of connected components $\widetilde{C}$ such that:

$$\Pr\left(|\widetilde{C} \quad C| \leq \varepsilon n\right) \geq 1 \quad ,$$

where C is the actual number of connected components in G.

Remark. The reason why we settled for this additive approximation (with respect to n) as opposed to multiplicative approximation (having $|\widetilde{C} \quad C| \leq \varepsilon \cdot C$) or just aiming for the exact answer is as follows: distinguishing whether a graph is connected or has two connected components, thus a better-than-2-approximation, requires (n^2) time (we will prove this result later in the course).

Before we get to describe the algorithm, we need a definition.

Definition 1. *For any vertex $v \in V$, we de ne s_v as the* **size of the connected component of v in G,** *i.e., the number of vertices (including v) that are in the same connected component as v.*

The following claim reduces the task of estimating the number of connected components to computing a simple function of s_v's for all $v \in V$.

Claim 2. $C = \sum_{v \in V} 1/s_v$.

Proof. Let $D_1, \ldots, D_C$ denote the connected components of G. Note that $V = D_1 \cup \ldots \cup D_C$ and D_i's are disjoint. This way, vertices of each connected component D_i contribute $1/|D_i|$ to the sum, which adds up to 1 in the component. Hence, the total sum is C. Formally,

$$\sum_{v \in V} \frac{1}{s_v} = \sum_{i=1}^{C} \sum_{v \in D} \frac{1}{s_v} = \sum_{i=1}^{C} \sum_{v \in D} \frac{1}{|D_i|} = \sum_{i=1}^{C} |D_i| \cdot \frac{1}{D_i} = \sum_{i=1}^{C} 1 = C.$$

□

Our general strategy is now to calculate the sum in Claim 2 to estimate C by sampling a small number of vertices v and computing s_v, which can be done by doing any form of graph search, say, DFS or BFS, starting from v and counting number of visited vertices. This strategy at this point however is problematic because when s_v is very large, computing all vertices connected to v can take a long time. An important observation is that having a "large" s_v makes the contribution of v to the summation above, i.e., $\frac{1}{s_v}$, "small" and thus almost negligible. We formalize this in the following.

Claim 3. *De ne $s'_v := \min(s_v, 2/\varepsilon)$ for all $v \in V'$ and $C' := \sum_{v \in V} 1/s_v$. Then, $|C \quad C'| \leq (\varepsilon/2) \cdot n$.*

Proof. First, observe that for each $v \in V$:

$$0 \leq \frac{1}{s'_v} \quad \frac{1}{s_v} \leq \frac{\varepsilon}{2}.$$

This holds because $s'_v \leq s_v$, and $s_v > 0$, and whenever $s'_v \neq s_v$, we have that $s'_v = 2/\varepsilon$, and $s_v > 0$. By summing the inequality over all vertices:

$$C' \quad C = \sum_{v \in V} \frac{1}{s'_u} \quad \frac{1}{s_u} \leq \frac{\varepsilon}{2} \cdot n,$$

$$C' \quad C = \sum_{v \in V} \frac{1}{s'_u} \quad \frac{1}{s_u} \geq 0$$

concluding the proof. □

Claim 3 ensures that if instead of computing s_v, we compute s'_v, we can still get a good estimate of C. However, computing s'_v is easier now since we only need to do a graph search starting from the vertex v and terminate the search whenever more than $2/\varepsilon$ vertices are found.

Remark. Claim 3 gives a straightforward *deterministic* algorithm for this problem – simply compute s'_v for every vertex which takes $O(1/\varepsilon^2)$ time per vertex (see Section 2.2 for details). This gives an $O(n/\varepsilon^2)$ time deterministic algorithm which is sublinear in the size of input (which can be (n^2)) but not sublinear in the number of vertices. In the rest of this part, we are going to show that using randomization, one can get a much faster algorithm for this problem.

We are now ready to present the algorithm.

Algorithm: An algorithm for Problem 1 on any given graph $G = (V, E)$.

1. Let $k := {}^{2}/_{\varepsilon} \cdot \ln({}^{2}/_{\quad})$.
2. For $i = 1$ to k do the following:

   Sample a vertex v_i uniformly at random from V (with replacement).
   For the vertex v_i, compute $X_i := \frac{1}{s_v}$ by doing a graph search, say, DFS or BFS, from v_i and truncating the search once ${}^{2}/_{\varepsilon}$ vertices are visited.
3. Output $\widetilde{C} = {}^{n}/_{k} \cdot \sum_{i=1}^{k} X_i$.

In order to analyze this algorithm, we use the following additive variant of Chernoff bound[2].

Proposition 4 (Additive Cherno Bound). *Let $Y_1, Y_2, \ldots, Y_k$ be k independent random variables with values in $[0, 1]$ and $Y = \sum_i Y_i$. Then, for any $b \geq 1$,*

$$\Pr[|Y \quad [Y]| > b] \leq 2 \cdot \exp\left(\quad \frac{2b^2}{k}\right).$$

We now present the proof of correctness and runtime analysis of this algorithm.

2.1 Proof of Correctness

As in the previous lecture, we first compute the expected value of the output $\widetilde{C}$, and show that it is close to the desired answer and then bound the probability of deviation of this random variable from its expectation.

Claim 5. $\left[\widetilde{C}\right] = C'$.

Proof. By linearity of expectation, we have,

$$\begin{aligned}\left[\widetilde{C}\right] &= \frac{n}{k} \cdot \sum_{i=1}^{k} \quad [X_i] = \frac{n}{k} \cdot k \cdot \quad [X_1] \qquad \text{(as } X_1, \ldots, X_k \text{ are identically distributed)} \\ &= n \cdot \quad [X_1].\end{aligned}$$

We can compute $[X_1]$ as follows:

$$[X_1] = \sum_{v \in V} \Pr(v \text{ is chosen as } v_1) \cdot \quad [X_1 \mid v \text{ is chosen as } v_1] = \frac{1}{n} \cdot \sum_{v \in V} \frac{1}{s'_v} = \frac{1}{n} \cdot C',$$

where the second to last equality is because when we choose v in the algorithm as v_1, we set $X_1 = {}^{1}/_{s_v}$, and the last equality is by the definition in Claim 3. The claim now follows from the above two equations. □

By Claim 5 (and Claim 3), the output is within the desired range in expectation. We now use Chernoff bound to bound the probability that it also deviates from its expectation by much.

Claim 6. $\Pr\left(|\widetilde{C} \quad C'| \leq ({}^{\varepsilon}/_{2}) \cdot n\right) \geq 1 \quad .$

[2]It is worth mentioning that the bounds one get from multiplicative Cherno bound is always at least as good as the additive version – we thus only use additive Cherno for simplifying the calculations when possible.

Proof. Define $X := \sum_{i=1}^{k} X_i$. Note that this way $\widetilde{C} = {}^{n}/_{k} \cdot X$ and

$$[X] = \frac{k}{n} \cdot \quad \left[\widetilde{C}\right] = \frac{k}{n} \cdot C',$$

by Claim 5. Moreover,

$$|\widetilde{C} \quad C'| \geq \frac{\varepsilon}{2} \cdot n \iff |\frac{n}{k} \cdot X \quad \frac{n}{k} \cdot \quad [X]\,| \geq \frac{\varepsilon}{2} \cdot n \iff |X \qquad [X]\,| \geq \frac{\varepsilon}{2} \cdot k.$$

Finally, X is a sum of k independent random variables X_i's which are in $[0,1]$. Hence, we can apply the additive Chernoff bound in Proposition 4 with parameter $b = {}^{\varepsilon}/_{2} \cdot k$ and obtain that,

$$\begin{aligned}\Pr\left(|X \qquad [X]\,| \geq \frac{\varepsilon}{2} \cdot k\right) &\leq 2 \cdot \exp\left(\quad \frac{2 \cdot (\varepsilon/2)^2 \cdot k^2}{k}\right) = 2 \cdot \exp\left(\quad \frac{\varepsilon^2}{2} \cdot k\right) \\ &= 2 \cdot \exp\left(\quad \frac{\varepsilon^2}{2} \cdot \frac{2}{\varepsilon^2} \cdot \ln\left(2/\ \right)\right) \leq 2 \cdot \quad /2 = \quad . \qquad \text{(by the choice of } k\text{)}\end{aligned}$$

This proves the desired claim. □

By Claim 3 we know that C' is close to C (deterministically) and by Claim 6, we get that $\widetilde{C}$ is close to C' with probability $1 \quad$. We can combine these two together to conclude the correctness of the algorithm.

Lemma 7. *The output $\widetilde{C}$ of the algorithm satis es* $\Pr\left(|\widetilde{C} \quad C| \leq \varepsilon \cdot n\right) \geq 1 \quad .$

Proof. By Claim 6, with probability at least $1 \quad$, we have, $|\widetilde{C} \quad C'| \leq ({}^{\varepsilon}/_{2}) \cdot n$. Moreover, by Claim 3, we have $|C' \quad C| \leq ({}^{\varepsilon}/_{2}) \cdot n$ (deterministically). Hence, by triangle inequality, with probability at least $1 \quad$,

$$|\widetilde{C} \quad C| \leq |\widetilde{C} \quad C'| + |C' \quad C| \leq \frac{\varepsilon}{2} \cdot n + \frac{\varepsilon}{2} \cdot n = \varepsilon \cdot n,$$

finalizing the proof of correctness of the algorithm. □

2.2 Runtime Analysis

Given v_i, computing s'_v in the algorithm takes $O({}^{1}/_{\varepsilon}\ )$ time because we are going to visit only ${}^{2}/_{\varepsilon}$ vertices from v_i and thus DFS or BFS will time proportional to the number of these vertices plus all edges between them which is at most $O({}^{1}/_{\varepsilon}\ )$. As such, the total runtime of the algorithm is:

$$k \cdot O(\frac{1}{\varepsilon^2}) = \frac{1}{\varepsilon^2} \cdot \ln\left(2/\ \right) \cdot O(\frac{1}{\varepsilon^2}) = O(\frac{1}{\varepsilon^4} \cdot \ln\left(1/\ \right)).$$

Remark. Notice that this algorithm runs in constant time (independent of the size of the input graph) whenever ε and are fixed constants.

2.3 Concluding Remarks

We saw an algorithm for estimating the number of connected components to within an $\varepsilon \cdot n$ additive approximation in time $O(\frac{1}{\varepsilon^4} \cdot \ln\left(1/\ \right))$. This result was first proved by Chazelle, Rubinfeld, and Trevisan in [1] who used it as a subroutine to estimate the weight of a minimum spanning tree in a graph in sublinear time.

Open question? The algorithm we discussed does not seem to obtain optimal bounds as a function of ε, . It would be interesting to investigate if these bounds can be improved further and/or prove a matching lower bound for this problem[3].

[3]Important Note: This problem may have already been solved and a literature search is the first step.