Sublinear Algorithms September 10, 2020

Lecture 3

Lecturer: Sofya Raskhodnikova *Scribe(s): Madhav Jha (modified)*

1 Simple Examples of Sublinear-Time Algorithms for Graphs

In the previous lecture notes, we defined additive and multiplicative approximation algorithms and a property tester and saw a few simple examples of property testers. Today we will see several examples of sublinear-time algorithms for graph problems: for testing if a graph is connected, approximating the number of connected components, and approximating the weight of a minimum spanning tree.

1.1 Definitions

For sublinear-time algorithms, it is crucial how the input is represented and accessed. Depending on input representation and queries allowed, we might or might not be able to solve a given problem in sublinear time. In addition, in the property testing model, the correctness guarantee of the algorithm depends on the distance between objects, which is typically normalized by the size of the input. The set of inputs that a test is required to reject (the ones that are ϵ-far) depends on the size of the representation. There are two models for graph representation, one appropriate for working with sparse graphs and the other – with dense graphs.

1.1.1 Adjacency Matrix (for dense graphs)[GGR98]

In this model a graph is represented by its adjacency matrix, of size n^2. We will look at it later in the course.

1.1.2 Adjacency Lists (for sparse graphs or graphs of bounded degree) [GR02]

In this model, for each node in a graph we are given its adjacency list (a list of its neighbors). In the bounded-degree model, we assume that graphs are of degree at most d, and so every adjacency list is of length d. If the degree of a vertex is less than d, the last entries in its adjacency list are empty (represented by $\emptyset$). For each $v \in V(G)$ and $i \in [d]$, a query (v, i) returns the i-th entry of the adjacency list of the vertex v (which might be $\emptyset$).

As always, we define the distance between two instances as a fraction of the entries in their representation on which they differ. Since the representation size in this model is dn, it translates into:

$$\text{DIST}(G_1, G_2) = \frac{\#\text{ of entries in the adjacency lists on which } G_1 \text{ and } G_2 \text{ differ}}{dn}.$$

Note that this definition tells us how to compute distances only betweeen graphs with the same bound on their degree (the same length of the adjacency lists).

Two representations of the same graph might have non-zero distance from each other. However, it is not an issue, since the notion of distance is only used in defining which graphs a property tester has to reject. And all representations of a graph are considered in that definition. Recall that an ϵ-tester for some property $\mathcal{P}$ should accept (with probability $\geq \frac{2}{3}$) all 'YES' instances, and reject (with probability $\geq \frac{2}{3}$) all 'FAR FROM YES' instances. For bounded-degree graph properties, the 'YES' instances are representations of graphs with the property $\mathcal{P}$, and the 'FAR' instances are those with DIST $> \epsilon dn$ from *all* representations of a bounded-degree graph, which is a 'YES' instance.

1.2 Testing Connectedness [GR02]

Input: A graph $G = (V, E)$ of degree at most d, represented by adjacency lists.

Goal: An ϵ-tester for connectedness (that accepts all connected graphs with probability $\geq \frac{2}{3}$, and rejects with probability $\geq \frac{2}{3}$ all graphs that are ϵ-far from connected.)

Idea: Graphs that are far from connected have many connected components, and therefore they also have many "small" connected components.

Algorithm 1: Tester for connectedness$(n, d, \epsilon,$ query access to $G)$.

1 Pick $s = \frac{8}{\epsilon d}$ nodes from V independently and uniformly at random.
2 For every selected node v, determine whether v is in a small connected component:
do a BFS until either the connected component is exhausted or $\frac{4}{\epsilon d}$ new nodes are visited.
3 Reject if at least one small connected component was found, otherwise accept.

Query Complexity For every one of the selected $O(\frac{1}{\epsilon d})$ nodes, we perform a BFS until $O(\frac{1}{\epsilon d})$ new nodes are discovered. For each new node, we query all d of its adjacency list's entries, which gives query complexity $O(\frac{d}{\epsilon^2 d^2}) = O(\frac{1}{\epsilon^2 d})$. Running time is the same.

Analysis. All connected graphs are always accepted. It remains to show that if a graph is ϵ-far from connected, it is rejected with probability at least $\frac{2}{3}$. We will show it using the two claims below.

Claim 1. *If G is ϵ-far from connected, it has at least $\frac{\epsilon dn}{2}$ connected components.*

Claim 2. *If G is ϵ-far from connected, it has at least $\frac{\epsilon dn}{4}$ connected components of size at most $\frac{4}{\epsilon d}$.*

Assuming that Claim 2 holds, at least $\frac{\epsilon dn}{4}$ nodes are in "small" connected components (of size $\leq \frac{4}{\epsilon d}$). This means that at least $\alpha = \frac{\epsilon d}{4}$ fraction of the nodes of the graph "witness" that it is not connected. By the Witness Lemma from the first lecture, it suffices to look at $\frac{2}{\alpha} = \frac{8}{\epsilon d} = s$ random samples to get at least one witness with probability $\geq \frac{2}{3}$. □

Proof of Claim 1: We will show the contrapositive: namely, that if G has fewer than $\frac{\epsilon dn}{2}$ connected components then one can make G connected by modifying less than ϵ fraction of its representation, i.e,. fewer than ϵdn modifications to the adjacency lists suffice in order to make G connected. Suppose G has k connected components, where $K < \frac{\epsilon dn}{2}$. Then one can connect G by adding $k-1$ edges, which requires modifying $2(k-1)$ adjacency list entries. However, we have to be careful not to go beyond the degree bound, d, when we construct our new graph.

First, we show how to free up vertices in components with full adjacency lists. For each component C that has less than two empty entries in its adjacency lists, consider a spanning tree of C, and let v be a leaf of this spanning tree. One of v's edges connects it to its ancestor in the tree. Any of the remaining $d-1$ edges can be safely removed without disconnecting C. Remove one of these edges, thus freeing up two entries in the adjacency lists of vertices in C. Now each connected component has at least 2 empty entries in the adjacency lists. We connect these components in a chain: C_1 to C_2, C_2 to C_3, C_3 to C_4, etc., by adding $k-1$ edges. This requires modifying at most two adjacency entries per component and, if we already erased some entries for a component, we modify the same entries when we are adding new edges. Thus, we can connect k components by changing at most $2k$ entries in the adjacency lists. Since $k < \frac{\epsilon dn}{2}$, we modify less than ϵdn entries in our representation of G. □

Proof of Claim 2: By Claim 1, there are at least $\frac{\epsilon dn}{2}$ connected components, and so their average size is at most $\frac{n}{\epsilon dn/2} = \frac{2}{\epsilon d}$. By an averaging argument, or by Markov's inequality, we deduce that at least half of these components are of size at most twice the average. That is, at least $\frac{\epsilon dn}{4}$ of the components are of size at most $\frac{4}{\epsilon d}$. □

1.3 Approximating the number of connected components [CRT05]

Input: A graph $G = (V, E)$ of degree at most d.

Goal: An ϵn-additive approximation for C, the number of connected components of G.

Intuition: For every $u \in V$, let n_u denote the number of nodes in the connected component containing u. If A is a connected component then $\sum_{u \in A} \frac{1}{n_u} = \frac{|A|}{|A|} = 1$, and therefore

$$\sum_{u \in V} \frac{1}{n_u} = C.$$

The algorithm that employs this formula to compute C (by calculating n_u for all nodes u) is not the best exact algorithm: it takes $O(dn^2)$ time while a BFS can compute C in $O(dn)$ time. However, this formula is an excellent starting point for a sublinear algorithm because it breaks C up into contributions of different nodes. In addition, when n_u is small, it can be calculated the same way as before, by running BFS for a limited number of steps, and when it is large, it does not contribute much to the sum.

Our algorithm will estimate this sum by estimating n_u for a few random nodes u. Define the estimate of n_u, denoted by $\hat{n}_u$, by $\hat{n}_u = \min\{n_u, \frac{2}{\epsilon}\}$, and the corresponding estimate for C by $\hat{C} = \sum_{u \in V} \frac{1}{\hat{n}_v}$. The following claim shows that $\hat{C}$ is close to C.

Claim 3. $|\hat{C} - C| \leq \frac{\epsilon n}{2}$.

Proof. First, observe that for each u,

$$0 \leq \frac{1}{\hat{n}_u} - \frac{1}{n_u} \leq \frac{\epsilon}{2}$$

because either $n_u \leq \frac{2}{\epsilon}$ (and thus $n_u = \hat{n}_u$) or $n_u > \frac{2}{\epsilon} = \hat{n}_u$ (and thus $0 < \frac{1}{\hat{n}_u} - \frac{1}{n_u} < \frac{\epsilon}{2}$). By summing the inequalities over all vertices, we get that $0 \leq \sum_u \frac{1}{\hat{n}_u} - \frac{1}{n_u} = \hat{C} - C \leq \frac{\epsilon n}{2}$. □

Notice that $\hat{C}/n$ is the average of $\frac{1}{\hat{n}_u}$ values over all nodes u. Our algorithm estimates it from a few random samples.

Algorithm 2: *approx_#CCs*$(n, d, \epsilon,$ query access to $G)$

1. Pick $s = O(\frac{1}{\epsilon^2})$ nodes $u_1, u_2, \ldots, u_s \in V$ independently and uniformly at random.
2. For each u_i compute $\hat{n}_{u_i}$ by a BFS, limited to looking at $\frac{2}{\epsilon}$ new vertices.
3. Output $\tilde{C} = n\left(\frac{1}{s}\sum_{i=1}^{s} \frac{1}{\hat{n}_{u_i}}\right)$.

The quantity $\left(\frac{1}{s}\sum_{i=1}^{s} \frac{1}{\hat{n}_{u_i}}\right)$ is the empirical average that we use to approximate $\hat{C}/n$.

Query Complexity. For each of the $O(\frac{1}{\epsilon^2})$ selected nodes, we run a BFS until we see at most $O(\frac{1}{\epsilon})$ new nodes. For each new node, we query at most its entire adjacency list of length d, so we make $O(\frac{d}{\epsilon^3})$ queries. The running time is the same.

Analysis. We have already proven that C and $\hat{C}$ cannot be too far away. Let's prove a similar statement for $\hat{C}$ and $\tilde{C}$.

Claim 4. $\Pr[|\tilde{C} - \hat{C}| \leq \frac{\epsilon n}{2}] \geq \frac{2}{3}$.

The proof of this claim is a simple application of a variant of the Hoeffding bound (see exercise 4.7 in the book by Motwani and Raghavan [MR95] for a proof).

Fact 5 (Hoeffding Bound)**.** *Let $Y_1, Y_2, \ldots, Y_s$ be independent identically distributed random variables with values in $[0,1]$, and let $Y = \frac{1}{s} \cdot \sum_{i \in [s]} Y_i$. Then*

$$\Pr[|Y - E[Y]| > \delta] \leq e^{-\Omega(\delta^2 s)}.$$

Proof of Claim 4: To use the Hoeffding bound, let $Y_i = \frac{1}{\hat{n}_{u_i}}$. Then

$$Y = \frac{1}{s} \cdot \sum_{i \in [s]} Y_i = \frac{\tilde{C}}{n}.$$

By the linearity of expectation and since all Y_i are identically distributed,

$$E[Y] = \frac{1}{s} \sum_{i \in [s]} E[Y_i] = E[Y_1] = \frac{1}{n} \sum_{u \in V} \frac{1}{\hat{n}_u} = \frac{\hat{C}}{n}.$$

Now we apply the Hoeffding bound:

$$\Pr\left[|\tilde{C} - \hat{C}| > \frac{\epsilon n}{2}\right] = \Pr\left[\left|\frac{\tilde{C}}{n} - \frac{\hat{C}}{n}\right| > \frac{\epsilon}{2}\right] = \Pr\left[|Y - E[Y]| > \frac{\epsilon}{2}\right] \leq e^{-\Omega(\epsilon^2 s)}.$$

This is at most $1/3$ for some s that is $O(\frac{1}{\epsilon^2})$. □

Claim 6. $\Pr[|C - \tilde{C}| \leq \epsilon n] \geq \frac{2}{3}$.

Proof. When $|\tilde{C} - \hat{C}| \leq \frac{\epsilon n}{2}$ (which happens with probability at most $\frac{2}{3}$, according to Claim 4), we get

$$|C - \tilde{C}| \leq |C - \hat{C}| + |\hat{C} - \tilde{C}| \leq \frac{\epsilon n}{2} + \frac{\epsilon n}{2} = \epsilon n,$$

since, by Claim 3, $|C - \hat{C}| \leq \frac{\epsilon n}{2}$. □

This completes the argument that, with probability at least $\frac{2}{3}$, the algorithm outputs an estimate of the number of connected components, accurate to within an additive term of ϵn. Next, we employ this algorithm as a subroutine in order to approximate the weight of a minimum spanning tree in a graph in the bounded degree model.

1.4 Approximating the weight of the minimum spanning tree (MST) [CRT05]

Input: A connected graph $G = (V, E)$ of degree at most d, where each edge $(i, j) \in E$ is assigned an integer weight $w_{i,j} \in \{1, \ldots, w\}$. As before, G is represented by the adjacency lists, but now each entry is augmented by the weight of the corresponding edge.

Goal: An ϵn-additive approximation of w_{MST}, the weight of the MST of G.

Later, we show that for this problem additive approximation implies multiplicative approximation.

As a starting point, recall **Kruskal**'s algorithm (Algorithm 3), for computing an MST exactly.

Kruskal's algorithm always finds an MST, but not in sublinear time. We will show how to approximate the weight of an MST, without actually finding an MST.

Algorithm 3: Kruskal(G)

1 Sort edges by weight.
2 Start with $i = 1$ and tree $T = \emptyset$.
3 Add as many edges of weight i to T as possible, making sure there is no cycle.
4 If there are $n - 1$ edges in T, stop. Otherwise, $i = i + 1$ and go to step 3.

Idea: Define $E_i = \{(j,k) \in E : w_{j,k} \leq i\}$. Let $G_i = (V, E_i)$ be the subgraph of G which includes only edges of weight at most i. Define C_i as the number of connected components in G_i. Denote by T some MST of G. According to Kruskal, the number of edges of weight greater than i in T is equal to $C_i - 1$, since at the stage where one adds the last edge of weight at most i, the number of connected components is exactly C_i (otherwise we can add another edge), and each edge that we add next will reduce the number of connected components by 1, until we have just one connected component.

Claim 7. $w_{MST} = n - w + \sum_{i=1}^{w-1} C_i$.

Proof. Let T be any MST of G. Define α_i to be the number of edges of weight i in T, and β_i to be the number of edges of weight greater than i in T. By our previous observation, $\beta_i = C_i - 1$, and so:

$$MST(G) = \sum_{j=1}^{w} j\alpha_j = \sum_{i=1}^{w} \left(\sum_{j=i}^{w} \alpha_j \right) = \sum_{i=1}^{w} \beta_{i-1} = \sum_{i=1}^{w} (C_{i-1} - 1) = -w + \sum_{i=0}^{w-1} C_i = n - w + \sum_{i=1}^{w-1} C_i.$$

□

Algorithm 4: *approx_MST*$(n, d, w, \epsilon;$ query access to $G)$

1 Let $C_0 = |V|$.
2 **for** $i = 1, \ldots, (w-1)$ **do**
compute $\tilde{C}_i = approx\_\sharp CCs(n, d, \epsilon/w; G_i)$.
end
3 Output: $\tilde{w}_{MST} = -w + \sum_{i=0}^{w-1} \tilde{C}_i$.

Analysis. Suppose all estimates of C_i's are good: $|\tilde{C}_i - C_i| \leq \frac{\epsilon}{w} n$. Then

$$\begin{aligned} |\tilde{w}_{MST} - w_{MST}| &= \Big| \sum_{i=1}^{w-1} \Big(\tilde{C}_i - C_i \Big) \Big| \\ &\leq \sum_{i=1}^{w-1} \big| \tilde{C}_i - C_i \big| \leq w \cdot \frac{\epsilon}{w} n = \epsilon n. \end{aligned}$$

The probability that all $w - 1$ estimates are good is at least $(2/3)^{w-1}$. This doesn't seem good enough. In fact, we would like to have error probability at most $\frac{1}{3w}$ for each estimate because then, by a union bound, for the final estimate, $\Pr[\text{error}] \leq w \cdot \frac{1}{3w} = \frac{1}{3}$. The following exercises guide you through changing the subroutine that approximates the number of connected components to reduce the error probability. This completes the analysis.

Exercise 3.1 (Success probability amplification)**.** *Show how to amplify success probability of any randomized algorithm by repeating it and taking the median answer.*

In this case, we can take a more direct approach.

Exercise 3.2. *Prove that it suffices to take* $s = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ *samples in approx_♯CCs to get* ϵn*-additive approximation with probability at least* $1 - \delta$*. What is the resulting running time of approx_MST with* $\delta = \frac{1}{3w}$*?*

Multiplicative approximation for MST cost. For computing the MST cost, additive approximation implies multiplicative approximation. To see this, first observe that the cost of an MST is at least $n-1$ implying $w_{MST} \geq \frac{n}{2}$ for $n \geq 2$. Using this and the fact that $\tilde{w}_{MST}$ is an ϵ-additive approximation, we get $(1 \pm 2\epsilon)$-multiplicative approximation:

$$w_{MST} - \epsilon n \leq \tilde{w}_{MST} \leq w_{MST} + \epsilon n;$$
$$w_{MST}(1 - 2\varepsilon) \leq \tilde{w}_{MST} \leq w_{MST}(1 + 2\varepsilon).$$

References

[CRT05] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998. Preliminary version in 37th FOCS, 1996.

[GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. Preliminary version in 29th STOC, 1997.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.