CSCI 1951-W Sublinear Algorithms for Big Data **Fall 2020**

Lecture 5: Bounded Degree Graph Algorithms

Lecturer: Jasper Lee *Scribe: Louis Kilfoyle*

1 Representing Graphs

- To consider how far from some property a graph is, we need a distance metric.
- Algorithms and complexities for testing depend significantly on how we represent graphs. In particular, sparse and dense graphs are often represented differently.
  - **Uniform Sparsity:** All vertices have degree $\leq d$. Well represented by adjacency lists.
  - **Average Sparsity:** Average vertex degree $\leq d$
  - **Dense:** One idea of "dense" is the property that there exists at least one vertex with high degree. Well represented by an adjacency matrix.

2 Bounded Degree Graph Model

Overview

- Assume graph is represented such that we can query (v, i) to get the i-th neighbor of vertex v, for $i \in [d]$.
- No kind of consistency between ordering of vertices. v_1 could be the i-th neighbor of v_2, and v_2 the j-th neighbor of v_1.
- Query returns NULL if v has fewer than i neighbors. (Notice binary search can determine vertex degree).
- Input size? What does "linear in the input" mean here? Here size $= d \cdot n$, since this is the total number of distinct queries possible.

Distance & Representation

- A natural distance metric in this model is:

$$d(G_1, G_2) = \frac{1}{dn} \cdot \sum_{j=1}^{n} \sum_{i=1}^{d} \mathbb{1}_{G_1(j,i) \neq G_2(j,i)}$$

- Notice that if G_1 and G_2 are the same graph but with different representations (so all vertices have the same neighbors, but ordered differently), they will have non-zero distance between one another.
  - This is odd, but not problematic for property testing.
  - Recall that, to measure property distance, we find the element in the property set with the least distance to the element being tested.

– In this case, we additionally take the representation (permutations of adjacency lists) of that element (graph) with the least distance to the element being tested.

3 Connectedness Testing: $O(\frac{1}{\epsilon^2 d})$ Algorithm

ϵ-far from connected just means we need to change at most $\epsilon \cdot dn$ entries to make the graph connected. In order to leverage the "ϵ-gap" algorithmically, we first relate this number of necessary changes to the number of components in the graph.

Lemma 5.1 *If G is ϵ-far from connected, then it has $\geq \frac{\epsilon \cdot dn}{2}$ components.*

Proof. Contrapositive: If there are $< \frac{\epsilon \cdot dn}{2}$ components in the graph, then we only need to change fewer than $\epsilon \cdot dn$ entries to connect the graph.

Idea: For k components, we can add $k-1$ edges by making $2k-2$ changes (2 changes per edge, one in each of two components) to connect the graph. However, if a component is d-regular, it cannot gain an edge and we must redirect an existing one. This can still be done in two changes. So each component requires at most two changes, and therefore we can connect the graph in $2k$ changes.

Thus we can conclude that a graph with fewer than $\frac{\epsilon \cdot dn}{2}$ components can be connected in fewer than $\epsilon \cdot dn$ changes. □

Furthermore, we can show by an averaging argument that, not only does the graph have a lot of components, it must also have a lot of *small* components.

Lemma 5.2 *If G is ϵ-far from connected, then at least $\frac{\epsilon \cdot dn}{4}$ components have size $\leq \frac{4}{\epsilon d}$.*

Proof. By contradiction. Assume $< \frac{\epsilon \cdot dn}{4}$ components of size $\leq \frac{4}{\epsilon d}$. Because G is ϵ-far from connected, we know by Lemma 5.1 it has at least $\frac{\epsilon \cdot dn}{2}$ components. So there must exist $> \frac{\epsilon \cdot dn}{4}$ components of size $> \frac{4}{\epsilon d}$, but this implies $> \frac{\epsilon \cdot dn}{4} \times \frac{4}{\epsilon d} = n$ vertices in these components alone. □

Algorithm 5.3: ϵ-far Connectedness Testing

Input: A graph G represented by adjacency lists
Output: An estimate for whether or not G is ϵ-far from connected
for *i = 1 ... $O(\frac{1}{\epsilon d})$* **do**
  Pick random vertex u
  Breadth-first search (BFS) from u for at most $\frac{4}{\epsilon d}$ new vertices
end

So we explore at most $\frac{1}{\epsilon d}$ vertices in each BFS, and do this for $O\left(\frac{1}{\epsilon d}\right)$ iterations. We might however "visit" a vertex $O(d)$ many times because it has degree at most d. Hence the query complexity is

$$\left(\frac{1}{\epsilon d} \cdot \frac{d}{\epsilon d}\right) = O\left(\frac{1}{\epsilon^2 d}\right)$$

A more sophisticated variation of the above algorithm can improve the query complexity to

$$O\left(\frac{1}{\epsilon}\text{polylog}\left(\frac{1}{\epsilon^2 d}\right)\right)$$

but it is beyond the scope of this class. Notice that this algorithm is adaptive (BFS is inherently adaptive). Non-adaptive algorithms cannot do better than $\Omega(\sqrt{n})$. Again, proof of this is outside of scope.

4 Approximating # of Connected Components: $O(\frac{d}{\epsilon^3})$ Algorithm

Known Complexity Bounds: $O\left(\frac{d}{\epsilon^2}\log\frac{d}{\epsilon}\right)$ and $\Omega\left(\frac{d}{\epsilon^2}\right)$

Goal: Approximate true number of connected components in graph to within $\pm\epsilon n$, with probability $\geq 1-\delta$.

General idea is to sample random vertices and BFS from them (possibly stopping early), like algorithm 5.3. As we BFS, we get an idea of the size of a component. This raises the question: how do we relate the number of components to the size of components?

Notation: n_u is the size of the component to which vertex u belongs. CC refers to Connected Components.

Proposition 5.4

$$\#\ of\ CC = \sum_v \frac{1}{n_v}$$

Proof. For a component H

$$\sum_{v\in H}\frac{1}{n_v} = 1$$

□

Idea: Subsample to approximate total sum. Use Hoeffding's Inequality for concentration (applicable because $\frac{1}{n_v}\in[0,1]$).

Algorithm 5.5: Estimating # of CC in a Graph

Input: A graph G represented by adjacency lists
Output: A high probability estimate for the # of CC in G
for $i = 1...k \in \Theta(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ **do**
  Pick random vertex u
  Compute n_u via BFS from u
end
return $n \times \frac{1}{k}\sum_{i=1}^{k}\frac{1}{n_u}$

The query complexity here is unfortunately $\Omega(n)$ because there might exists only a component with $\Theta(n)$ vertices. This algorithm is therefore not sublinear in n. We need to fix the algorithm.

Observation: Step 2 (BFS) has high complexity if and only if n_u is big (i.e. n belongs to a big component). Equivalently, if and only if $\frac{1}{n_u}$ is small. Refer back to Prop 5.4, and notice now that the highest complexity vertices also contribute the least to the sum.

Solution: End the BFS early if running too long. That is, compute $\hat{n}_u$ instead of n_u:

$$\hat{n}_u = \min\left(n_u, \frac{2}{\epsilon}\right)$$

This is an acceptable introduction of error, because

$$\left|\frac{1}{\hat{n}_u} - \frac{1}{n_u}\right| \leq \frac{\epsilon}{2} \Rightarrow \sum_u \left|\frac{1}{\hat{n}_u} - \frac{1}{n_u}\right| \leq \frac{\epsilon n}{2}$$

Algorithm 5.6: Run Algorithm 5.5 but compute $\hat{n}_u$ instead by ending the BFS at $\frac{2}{\epsilon}$ vertices explored. The query complexity here is now $O\left(\frac{d}{\epsilon^3}\right)$.

Error: We have bias (introduced by $\hat{n}_u$), and sampling error. The former we've shown to be bounded by $\frac{\epsilon n}{2}$, and the latter can be bounded the same way by tuning sample size. So

$$\text{Total Error} = \text{Bias} + \text{Sampling} \leq \frac{\epsilon n}{2} + \frac{\epsilon n}{2} = \epsilon n$$

5 Approximating MST Weight: $O\left(\frac{dw^4 \log w}{\epsilon^3}\right)$ Algorithm

Known Complexity Bounds: $O\left(\frac{dw}{\epsilon^3} \log \frac{dw}{\epsilon}\right)$ and $\Omega\left(\frac{dw}{\epsilon^2}\right)$

Assumptions:

- d is max degree of graph
- Edges have weights in $[w] = \{1, ..., w\}$. For a graph with weights in $\mathbb{R}$, divide the range of weights into w equal width regions and assign labels from $[w]$. w is a "granularity parameter" that affects the quality of our estimate and complexity of our algorithm (higher granularity = higher complexity = higher accuracy).

Goal: Approximate the weight of the Minimum Spanning Tree (denoted $w(\text{MST})$) to $\pm\,\epsilon n$. Note this is additive error, but $w(\text{MST})$ is $\geq n$ and therefore additive error implies multiplicative error as well. That is, our estimate will also be at most $(1 + \epsilon) \cdot w(\text{MST})$.

Idea: Reduce approximating MST to counting number of components in a family of subgraphs of G.

Notation:

- $B_i = \#$ of edges in the MST with weight $> i$
- $C_i = \#$ of components in subgraph of edges with weights $\leq i$

Observations:

- If we apply Kruskal's Algorithm on a graph with weights from $[w]$, we essentially run the following process. Take all edges with weight 1 that connect disjoint components (i.e. don't form cycles). Repeat for weights 2...w.

- After the first pass of Kruskal's Algorithm, we've created a subgraph by taking edges with weight 1. This subgraph now has C_1 components, by definition. We know that a graph with k components needs $k-1$ more edges to become connected, so it must be the case that the remaining passes of Kruskal's Algorithm will add $C_1 - 1$ edges. Since these will end up being exactly the MST edges with weight > 1, the number of such edges is B_1 by definition. So $B_1 = C_1 - 1$, and this generalizes to any $i \in [w]$. Therefore
$$B_i = C_i - 1$$

- Now we can simply write the weight of the MST as a sum over weighted edges. There are $B_i - B_{i+1}$ edges in the MST with weight $i+1$, for $i \in [0, w-1]$. This is a telescoping sum, so we can rearrange to write the weight only in terms of n, w, and the C_i values.
$$\begin{aligned} w(\text{MST}) &= \sum_{i=0}^{w-1} (i+1)(B_i - B_{i+1}) \\ &= \sum_{i=0}^{w-1} B_i \\ &= \sum_{i=0}^{w-1} C_i - 1 \\ &= n - w + \sum_{i=1}^{w-1} C_i \end{aligned}$$

- With Algorithm 5.6, we know how to estimate the number of components in a graph. So we can estimate C_i for each $i \in [1, w-1]$ to get an estimate for $w(MST)$:

Algorithm 5.7: Estimating weight of MST

Input: A graph G represented by adjacency lists
Output: An estimate for the weight of the MST of G
for $i = 1 \dots w-1$ **do**
  $\widetilde{C}_i \leftarrow$ Alg 5.6 with error $\frac{\epsilon}{w}$ and $\delta = \frac{1}{3w}$
end
return $n - w + \sum \widetilde{C}_i$

Analysis:

- Total Error $\leq \sum_{i=1}^{w} \frac{\epsilon}{w} n = \epsilon n$
- Failure Probability $\leq \delta w = \frac{1}{3w} w = \frac{1}{3}$

Why not subsample the sum, as in the previous application? Not obvious how to control variance in this simple algorithm. Hoeffding's bound wouldn't be useful here because $C_i \in [0, n]$, making the query complexity upper bound quadratic in n.