6.889 Sublinear Time Algorithms	September 3, 2020
Lecture 2	
Lecturer: Ronitt Rubinfeld	*Scribe: Ashwin Sah*

1 Outline

The following topics were discussed during the second lecture.

- Estimating Minimum Spanning Tree (MST) Weight
- Estimating average degree of a graph

The second problem was not completed and will be continued next time.

2 Approximate Minimum Spanning Tree Weight

Our next sublinear algorithm will be estimation of the minimum weight of a spanning tree of a graph. We will use in a key way the estimation of the number of connected components of a bounded degree graph, which was covered last time. Specifically, we had the following result.

Theorem 1 (Connected components) *There is an algorithm which takes as input a graph G of maximum degree at most d, in adjacency list format, and outputs a number within ϵn of the number of connected components of G within time $O(d\epsilon^{-4})$ with probability at least $3/4$.*

Furthermore, as a result of the Chernoff bound (see Homework 0), we can boost the success probability to at least $1-\beta$ with a running time of $O(d\epsilon^{-4}\log(1/\beta))$.

2.1 Stating the problem

Now we state the minimum weight spanning tree problem in a precise format.

Input - We are given a parameter $\epsilon > 0$ and a graph $G = (V, E)$ in adjacency list representation. It has $n = |V|$ vertices and maximum degree at most d. We are also given, associated to each of these edges, a weight $w_{uv} \in \{1, \ldots, w\} \cup \{\infty\}$. One can think of ∞ as corresponding to the edge (u, v) not being in E.

Remark - One can handle the more general case where w_{uv} are real numbers in $[1, w]$ (or are ∞). This is on Homework 1.

Promise - We are guaranteed that G is connected.

Output - For a tree spanning G, let

$$w(T) = \sum_{(i,j)\in T} w_{ij}.$$

Let $M = \min_{T \text{ spans } G} w(T)$, where the minimum is taken over trees T. We wish to output some $\widehat{M}$ such that $(1-\epsilon)M \leq \widehat{M} \leq (1+\epsilon)M$, with probability at least $3/4$.

Observation - By the condition on the weights, we see that $n - 1 \leq M \leq w(n-1)$. This will be useful later in converting additive error to multiplicative error.

2.2 Kruskal's algorithm

We now recall the classical approach to computing (exactly) the minimum spanning tree. The idea is the following: choose as many weight 1 edges as possible, then as many weight 2 edges as possible, and so on.

Kruskal's algorithm, informally, is as follows. Order the edges in increasing order by weight (breaking ties arbitrarily). Then run through the edges in increasing order, and add in an edge to our partially-constructed tree if it introduces no cycles.

We will not revisit the proof of correctness, but Kruskal's algorithm is intuitively telling us that "choosing edges as greedily as possible" gives a minimum spanning tree.

As a special case, imagine that there are only weight 1 and weight 2 edges. Maximizing the number of weight 1 edges is the same as minimizing the number of weight 2 edges. At this point, we need a bit of notation. Let $E^{(1)}$ be the set of edges with weight 1, and let $G^{(1)}$ be the graph spanned by these edges. Let $c^{(1)}$ be the number of its connected components.

Note that we need at least $c^{(1)} - 1$ edges of weight 2 in our spanning tree, in order to connect up the components of $G^{(1)}$ (which cannot be done using weight 1 edges by definition). Furthermore, it turns out it is possible to use exactly this many edge. Then, the minimum spanning weight satisfies

$$M = 1\#\{\text{weight } 1\} + 2\#\{\text{weight } 2\} = n - 1 + \#\{\text{weight } 2\} = (n-1) + (c^{(1)} - 1) = n - 2 + c^{(1)}.$$

Here we used that the total number of edges in a spanning tree is $n - 1$.

Now we generalize this idea. For $i \geq 1$, define $E^{(i)} = \{(u, v) : w_{uv} \leq i\}$ (edges of weight *at most* i) and $G^{(i)} = (V, E^{(i)})$ (the graph spanned by those edges). Let $c^{(i)}$ be the number of components of $G^{(i)}$.

Claim 2 $M = n - w + \sum_{i=1}^{w-1} c^{(i)}$.

Proof Let

$$\alpha_i = \#\{\text{edges of weight } i \text{ in any minimum spanning tree of } G\}.$$

Implicitly, we are using the fact that *all* minimum spanning trees of G have the same value of α_i, which follows from an analysis of Kruskal's algorithm. Furthermore, we have from Kruskal that

$$\sum_{i>\ell} \alpha_i = c^{(\ell)} - 1$$

for all $\ell \in \{0, 1, \ldots, w-1\}$. Here we let $c^{(0)} = n$, corresponding to $G^{(0)}$ being the empty graph. This is precisely the generalization of the previous analysis: $G^{(\ell)}$ has $c^{(\ell)}$ connected components, which therefore must be connected in a spanning tree by at least $c^{(\ell)} - 1$ edges of weight bigger than ℓ. Furthermore, this is optimal.

Thus

$$\begin{aligned} M &= \sum_{i=1}^{w} i\alpha_i = \sum_{i=1}^{w} \alpha_i + \sum_{i=2}^{w} \alpha_i + \sum_{i=3}^{w} \alpha_i + \cdots + \sum_{i=w}^{w} \alpha_i \\ &= (n-1) + (c^{(1)} - 1) + (c^{(2)} - 1) + \cdots + (c^{(w-1)} - 1) \\ &= n - w + \sum_{i=1}^{w-1} c^{(i)}, \end{aligned}$$

as desired. ■

Claim 2 tells us how we should proceed. For each $i \in \{1, \ldots, w-1\}$, we have a graph $G^{(i)}$ for which we wish to approximate $c^{(i)}$. At this point, Theorem 1 tells us that this is possible. With this in mind, we are ready to propose a sublinear algorithm for this problem.

2.3 Estimation of minimum spanning tree weight and analysis

Algorithm - For $i \in \{1, \ldots, w-1\}$, run the algorithm from Theorem 1 to compute $\widehat{c}^{(i)}$, the approximate number of connected components of $G^{(i)}$ within an additive error of $\epsilon' n$, where

$$\epsilon' = \frac{\epsilon}{2w}.$$

We use the remark after the statement of Theorem 1 to boost the success probability to at least $1 - \beta$, where $\beta = 1/(4w)$. Then we output

$$\widehat{M} = n - w + \sum_{i=1}^{w-1} \widehat{c}^{(i)}.$$

There is one technicality: we cannot actually construct the graph $G^{(i)}$ to run the connected components algorithm. Instead, we perform the algorithm on the graph G, but merely ignore edges with weight greater than i during the breadth-first search (BFS) phase of the algorithm (see the previous lecture notes for the implementation details of the connected components algorithm).

Running Time - Each call to the (boosted) connected component algorithm takes time which is $O(d(\epsilon')^{-4} \log(1/\beta))$. Using our values of ϵ' and β, this is time

$$O(d\epsilon^{-4} w^4 \log(1/w)) = \text{wt}O(d\epsilon^{-4} w^4).$$

Furthermore, there are $w - 1$ calls to said algorithm, at which point we merely compute the sum $\widehat{M}$. The total running time is

$$O(d\epsilon^{-4} w^5 \log(1/w)) = \text{wt}O(d\epsilon^{-4} w^5).$$

Remark - This running time can be improved to $O(dw\epsilon^{-2} \log(dw\epsilon^{-1}))$ with a more efficient algorithm, which is the current best known. There is a lower bound of $\Omega(dw\epsilon^{-2})$, but the logarithmic gap is open. Also, d being the maximum degree can be replaced with the average degree (since the same can be done in the connected components algorithm, which was mentioned last time).

Correctness - We call each approximate connected component algorithm with failure probability at most $\beta = 1/(4w)$. Therefore, the failure probability over the $w - 1$ different calls is at most $1/4$, so with probability at least $3/4$ we know that

$$|c^{(i)} - \widehat{c}^{(i)}| \leq \epsilon' n = \frac{\epsilon n}{2w}$$

for each $i \in \{1, \ldots, w-1\}$. Thus by Claim 2, we have

$$|M - \widehat{M}| \leq w \cdot \frac{\epsilon n}{2w} = \frac{\epsilon n}{2}.$$

Now, our earlier observation that $M \geq n - 1$ allows us to turn this additive error into multiplicative error. Indeed, this tells us $M \geq n/2$, hence

$$|M - \widehat{M}| \leq \epsilon M \implies (1 - \epsilon)M \leq \widehat{M} \leq (1 + \epsilon)M.$$

As noted above, this occurs with probability at least $3/4$, so the algorithm is valid.

3 Approximate average degree of a graph

Our next sublinear time algorithm will be estimating the average degree of a graph. We will set up the problem and explain a preliminary approach, discussing why it fails. A correct algorithm will be given next time.

3.1 Stating the problem

Given a graph $G = (V, E)$, let $d(u)$ for $u \in V$ be the degree of u, i.e., the number of neighbors.

Definition 3 (Average degree) *We let the* average degree *of G be*

$$\overline{d} = \frac{\sum_{u \in V} d(u)}{n}.$$

We will use a slightly modified input format for graphs than usual. We will have the usual adjacency list representation, but it will be supplemented by a degree list. In particular, we have the following two query types.

- Degree query: given $v \in V$, output $d(v)$.
- Neighbor query: given (v, j) for $v \in V$, return the jth neighbor of V.

We now set up the problem.

Input - We are given a parameter $\epsilon > 0$ and a graph $G = (V, E)$ in adjacency list representation with degree queries. It has $n = |V|$ vertices.

Promise - We are guaranteed that G is simple and has $\Omega(n)$, i.e., G is not ultra-sparse.

Output - We wish to output some $\widehat{d}$ such that $(1-\epsilon)\overline{d} \leq \widehat{d} \leq (1+\epsilon)\overline{d}$, with probability at least $3/4$.

3.2 Initial investigations

First we consider a naive sampling approach. We pick $O(??)$ sample nodes $v_1, \ldots, v_s$ independently, and output

$$\frac{1}{s}\sum_{i=1}^{s} d(v_i).$$

Unfortunately, straightforward Chernoff/Hoeffding needs $\Omega(n)$ samples for this to work. The problem with Chernoff is that nodes have degree 0 to n, so there is potentially high variance. We will discuss this in more detail later.

Lower bound? - Based on the above intuition, we construct a potentially problematic example. Consider a list of degrees $(n-1, 0, 0, 0, \ldots, 0)$. If this occurs, we need $\Omega(n)$ samples to find the "needle in a haystack". This would immediately crush all hopes of obtaining a sublinear algorithm. Luckily, there is no such graph: the vertex of degree $n-1$ must connect to all the other vertices, which all have degree 0. Instead, the degrees must look something like $(n-1, 1, 1, \ldots, 1)$, which may be possible.

However, here are some valid lower bounds. First, we consider why the ultra-sparse regime is to be avoided. Consider the following two graphs:

- The graph with 0 edges. $\overline{d} = 0$.
- A graph with 1 edge. $\overline{d} = 2/n$.

Approximation within multiplicative error ϵ will distinguish these two graphs. However, using either degree queries or adjacency queries, we need at least $\Omega(n)$ queries before we expect to hit either endpoint of the single edge.

Now consider the case where $\overline{d} \geq 2$, i.e., we are not in the ultrasparse case. Consider the following two graphs:

- The cycle C_n on n vertices. $\overline{d} = 2$.
- The disjoint union of a cycle $C_{n-cn^{1/2}}$ on $n - cn^{1/2}$ vertices and a clique $K_{cn^{1/2}}$ on $cn^{1/2}$ vertices. $\overline{d} \approx 2 + c^2$.

To explain the latter calculation, note that the cycle portion of the graph has $n - cn^{1/2} \approx n$ edges and the clique has $\binom{cn^{1/2}}{2} \approx c^2 n/2$ edges. Thus the total sum of degrees, which is twice the number of edges, is approximately $(2 + c^2)n$, so indeed $\overline{d} \approx 2 + c^2$.

Distinguishing within a multiplicative error of ϵ for small ϵ requires distinguishing these two graphs. However, again, degree queries and adjacency queries both will not find any vertex in the clique unless we use $\Omega(n/\sqrt{n}) = \Omega(\sqrt{n})$ queries. Therefore, we have a lower bound of $\Omega(\sqrt{n})$.

3.3 An (incorrect) algorithm idea

Let's try grouping nodes of similar degree, and estimate the average degree within each group. Then combine the information together.

Why might this help? Recall the Chernoff bound.

Theorem 4 (Chernoff bound) *Fix $\delta \in [0, 1]$. Let X_i be iid random variables with values in $[0, 1]$ and mean $p = \mathbb{E}[X_i]$. Let $X = \sum_{i=1}^{r} X_i$ and $\mu = \mathbb{E}[X] = rp$. Then*

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

To get a reasonable bound on the deviation probability, we need to pick $r = \Omega(p^{-1}\delta^{-2})$. For simplicity, assume $\delta = \Theta(1)$. Then we need $r = \Omega(p^{-1})$. However, X_i needs to be in $[0, 1]$. Therefore, let $X_i = \deg(v_i)/n$ where v_i is a randomly chosen vertex. The issue is that p can be as small as size $1/n$, which would require $r = \Omega(n)$, hence $\Omega(n)$ samples. This is too much.

On the other hand, if $b \leq \deg(v_i) \leq (1 + \epsilon)v$, we can set

$$X_i = \frac{\deg(v_i)}{(1 + \epsilon)b} \in \left[\frac{1}{1 + \epsilon}, 1\right].$$

Thus

$$p \in \left[\frac{1}{1 + \epsilon}, 1\right],$$

which is good.

Bucketing - Based on this, we consider the following *bucketing* process. Choose a parameter $\beta = \epsilon/C$, where C will be some absolute constant. We will have $t = O(\epsilon^{-1} \log n)$ buckets, namely,

$$B_i = \{v | (1 + \beta)^{i-1} < d(v) \leq (1 + \beta)^i\}$$

for $i \in \{0, 1, \ldots, t - 1\}$. We can also introduce a separate bucket for degree 0 nodes, or just assume there are no isolated vertices (it does not matter too much). Now the total degree of the nodes in B_i, call it d_{B_i}, satisfies

$$(1 + \beta)^{i-1}|B_i| \leq d_{B_i} \leq (1 + \beta)^i |B_i|.$$

Thus the average degree of the graph satisfies

$$\sum_i (1 + \beta)^{i-1} \frac{|B_i|}{n} \leq \overline{d} \leq \sum_i (1 + \beta)^i \frac{|B_i|}{n}.$$

It thus suffices to estimate each $|B_i|$ up to a multiplicative error of around ϵ/C.

First algorithm idea - Take a sample S of nodes. Let $S_i = S \cap B_i$ for each $i \in \{0, \ldots, t - 1\}$. (Note t is logarithmic in size.) Then use the fraction

$$\rho_i = \frac{|S_i|}{|S|}$$

as an estimate for $|B_i|/n$. Thus we output

$$\sum_i \rho_i (1+\beta)^{i-1}.$$

Issue - This fails. The problem is roughly that some buckets are actually hard to estimate (the above process not doing well enough). For example, suppose we have the complete bipartite graph $K_{3,n-3}$. That is, we have 3 nodes on one side and $n-3$ on the other, and all cross-edges are filled in. The bucket B_i containing degree 3 nodes and the bucket B_j containing degree $n-3$ nodes are both contributing roughly the same amount to the total degree

$$d_{\text{total}} = n\overline{d},$$

but it takes too long to use naive sampling to detect only 3 nodes of degree $n-3$.

In fact, this initial attempt will lead to a 2-approximation algorithm to the maximum degree, and then we will pursue further refinements to fully resolve the average degree approximation problem next lecture.