6.5240 Sublinear Time Algorithms Sep 14, 2022

Lecture 2

Lecturer: Ronitt Rubinfeld *Scribe: Lichen Zhang*

In this lecture, we study two fundamental problems in graph theory: finding the minimum spanning tree (MST) and computing the average degree of the graph.

1 Approximating MST in sublinear time

We recall the minimum spanning tree problem on a graph: given a graph $G = (V, E)$ that is connected and a cost function $f : E \to \mathbb{R}_{\geq 0} \cup \{\infty\}$, the goal is to output a spanning tree with minimum total weight $f(T) = \sum_{\{u,v\} \in T} f(\{u, v\})$ where T denotes a spanning tree of G. In addition to the standard formulation of the MST problem, we introduce several extra assumptions.

Assumption 1 *Let $G = (V, E)$ be a connected graph. We assume that*

- *The graph is given in the adjacency list representation.*
- *The maximum degree of G is d.*
- *The cost function f is restricted to positive integer output, i.e., $f : E \to \{1, 2, \ldots, w\} \cup \{\infty\}$.*

We also let $\epsilon \in (0, 1)$ be a parameter.

Further, let $M = \min_{T \text{is a spanning tree of } G} f(T)$, i.e, the minimum weight of the tree. We will output $\widehat{M}$ that is a $(1 \pm \epsilon)$ multiplicative approximation to M, that is,

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

Remark We note that due to the assumption on the edge weights, any spanning tree has its total weights being lower bounded by $n - 1$ and upper bounded by $w \cdot (n - 1)$, hence,

$$n - 1 \leq M \leq w \cdot (n - 1).$$

As we will show, we will use an additive approximation algorithm for approximating the number of connected components in our algorithm for approximating minimum weight of a spanning tree. The reason we can get a multiplicative approximation is due to this assumption on edge weights.

1.1 Approximating the number of connected components in sublinear time

Before describing our MST algorithm, we first recall a tool that was studied last time.

Theorem 2 *Let $G = (V, E)$ be a graph with max degree d and in adjacency list representation. Let $\epsilon, \beta \in (0, 1)$ be parameters. Let $C(G)$ denote the number of connected components of G. There exists a randomized algorithm that succeeds with probability at least $1 - \beta$ that outputs a number $\widehat{C}(G)$, such that*

$$C(G) - \epsilon \cdot n \leq \widehat{C}(G) \leq C(G) + \epsilon \cdot n.$$

The algorithm runs in time $O(\epsilon^{-4} d \log(1/\beta))$.

1.2 A new characterization of MST

To see how to use oracles for approximating connected components for MST, we give a new characterization of this problem in terms of the number of connected components.

Definition 3 *We define $G^{(i)} = (V, E^{(i)})$, where $E^{(i)} = \{\{u, v\} \in E : f(\{u, v\}) \in \{1, \ldots, i\}\}$. Further, we define $C^{(i)}$ to be the number of connected components of $G^{(i)}$.*

Before proceeding to prove any interesting lemmas, let us first play with some simple examples, and make some observations regarding $G^{(i)}$ and $C^{(i)}$.

Example 1

- *Suppose $w = 1$, then it is obvious that $M = n - 1$.*
- *Suppose $w = 2$, then the MST on such a cost function becomes interesting. Recall a classic algorithm by Kruskal for MST: we first sort all edges according to their weights, then greedily add edges from small weights to large while ensuring the added edge does not form a cycle. Such a sourting-then-grouping process is precisely captured by $G^{(i)}$: take the $w = 2$ case as an example, $G^{(1)}$ denote the graph that has only unit weight edges. To connect all components of $G^{(1)}$, we will have to use edges with weight 2, since we've consumed all unit weight edges. The total number of weight 2 edges we'll be using is $C^{(1)} - 1$, for connecting all connected components.*

  This also provides a new way to count the total weight of a MST:

$$\begin{aligned} M &= \textit{number of unit weight edges in the tree} + 2 \cdot \textit{number of weight 2 edges in the tree} \\ &= 1 \cdot \textit{number of edges in the tree} + 1 \cdot \textit{number of weight 2 edges in the tree} \\ &= n - 1 + C^{(1)} - 1 \\ &= n + C^{(1)} - 2, \end{aligned}$$

  where the first step is by definition, second step is by splitting the weight of weight 2 edges, by putting one copy for all edges, the other copy for themselves. For the third step, note that $C^{(1)} - 1$ captures the number of weight 2 edges in the tree, since we only need $C^{(1)} - 1$ edges to connect the graph.

The core idea of our algorithm is a generalization of the $w = 2$ example. We formalize it as follows.

Claim 4

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

Proof Let α_i denote the number of edges of weight i in any MST of G. Due to Kruskal's algorithm, we know that for any MST, α_i remains the same. We make some observations of α_i.

- $\sum_{i \in [w]} \alpha_i = n - 1$, this is the total number of edges in an MST.
- For any $\ell \in [w]$, $\sum_{i > \ell} \alpha_i = C^{(\ell)} - 1$, to interpret this quantity, we note that in this sum, we are only allowed to use edges with weights greater than ℓ, the number of such edges is $C^{(\ell)} - 1$ for connecting all the connected components of $G^{(\ell)}$.
- $C^{(0)} = n$, since there's no edge with weight 0, the graph is has no edges.

Let us express M in terms of α_i.

$$\begin{aligned} M &= \sum_{i=1}^{w} i \cdot \alpha_i \\ &= \sum_{j=1}^{w} (\sum_{i=j}^{w} \alpha_i) \\ &= \sum_{j=1}^{w} C^{(j-1)} - 1 \\ &= n - w + \sum_{j=1}^{w-1} C^{(j-1)}, \end{aligned}$$

where the second step is by splitting $i \cdot \alpha_i$ into i copies of α_i, and put them into different bucket of the sum. The third step is by $\sum_{i=j}^{w} \alpha_i = \sum_{i>j-1} \alpha_i = C^{(j-1)} - 1$. ■

Given this formula of M, we describe our algorithm for approximating MST:

Algorithm 1 Approximating MST

```
1: procedure ApproxMST(G, d, ε, β)
2:     for i = 1 → w − 1 do
3:         ε' ← ε/(2w)
4:         β' ← β/w
5:         Ĉ^(i) ← ApproxCC(G^(i), d, ε', β')
6:     end for
7:     M̂ ← n − w + Σ_{i=1}^{w−1} Ĉ^(i)
8:     return M̂
9: end procedure
```

Let us analyze the runtime and approximation guarantee of Algorithm 1.

Lemma 5 *Algorithm 1 runs in time $O(\epsilon^{-4} d w^5 \log(w/\beta))$.*

Proof Note that our algorithm consists of w iterations, for each iteration, we call ApproxCC with parameter $\epsilon' = \epsilon/(2w)$ and $\beta' = \beta/w$, each call runs in time $O(\epsilon'^{-4} d \log(1/\beta')) = O(\epsilon^{-4} d w^4 \log(w/\beta))$. Summing over w iterations yields our desired runtime $O(\epsilon^{-4} d w^5 \log(w/\beta))$. ■

Lemma 6 *Algorithm 1 outputs a number $\widehat{M}$ such that, with probability at least $1 - \beta$, $\widehat{M}$ satisfies*

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

Proof For each $\widehat{C}^{(i)}$, it is a $\epsilon' \cdot n$ additive approximation to $C^{(i)}$ with probability at least $11 - \beta'$, therefore, the sum is

$$\begin{aligned} \sum_{i=1}^{w} \widehat{C}^{(i)} &\leq (\sum_{i=1}^{w} C^{(i)}) + w \cdot \epsilon' n \\ &= (\sum_{i=1}^{w} C^{(i)}) + \epsilon \cdot n/2, \end{aligned}$$

and the lower bound follows similarly. This merely gives the following additive bound:

$$M - \epsilon \cdot n/2 \leq \widehat{M} \leq M + \epsilon \cdot n/2,$$

recall that by the assumption on edge weights, M can be lower bounded by $n-1$, therefore,

$$\begin{aligned} |M - \widehat{M}| &\leq \epsilon \cdot n/2 \\ &\leq \epsilon \cdot M, \end{aligned}$$

which gives the desired approximation factor.

Regarding the success probability, each individual call fails with probability at most β', via a union bound, the overall failure probability is at most $w \cdot \beta' = \beta$, which finishes the proof. ■

Remark We make two remarks about Algorithm 1. The runtime of the algorithm seems unsatisfactory — in fact, it can be improved to $O(\epsilon^{-2} dw \log(dw/\epsilon))$. This runtime is nearly optimal, since this problem has a lower bound of $\Omega(\epsilon^{-2} dw)$.

The assumption that the graph has max degree is crucial to the runtime of Algorithm 1, this is because for each call, we have to run BFS for sampled vertices, which will potentially go through all their neighbors (even though the edges are marked not presented in the graph, we still have to go through-then-skip due to the input format).

2 Approximating average degree of a graph

In this section, we study algorithms for approximating the average degree.

Definition 7 *Let $G = (V, E)$, the average degree of G, denoted by $\overline{d}$, is $\frac{1}{n}\sum_{v \in V} \deg(v)$.*

We also make some assumptions that concern certain type of query oracles.

Assumption 8 *We make the following assumptions:*

- *The graph G is simple, i.e, no parallel edges and self-loops.*
- *The graph is given in adjacency list representation.*
- *We have access to the* degree query*: on $v \in V$, return $\deg(v)$ in unit time.*
- *We have access to the* neighbor query*: on $(v, j) \in V \times [\deg(v)]$, return j-th neighbor of v in unit time.*

2.1 The first (but failed) idea: simple sampling

A natural idea is to sample s vertices $v_1, \ldots, v_s$ and output $\frac{1}{s}\sum_{i=1}^{s} \deg(v_i)$. However, this idea does not work well. We give some lower bound results.

Example 2 (Ultra sparse graph) *Consider the graph G_1 with 0 edges and $\overline{d} = 0$, and the graph G_2 with 1 edge, hence $\overline{d} = \frac{2}{n}$. Since there's only one edge present, one has to use $\Omega(n)$ queries to distinguish between these two cases.*

Example 3 (Cycle vs cycle & clique) *Consider the graph G_1 that is an n-cycle with $\overline{d} = 2$. The other graph G_2 is an $n - c\cdot\sqrt{n}$-cycle and a $c\cdot\sqrt{n}$-clique, the total degree on the cycle is $2\cdot(n - c\cdot\sqrt{n})$, and the total degree on the clique is $(c\cdot\sqrt{n}-1)^2 \approx c^2 n$. In total, this adds up to $2n - 2c\sqrt{n} + c^2 n \approx (2 + c^2)n$, and $\overline{d} = 2 + c^2$. To distinguish these two cases, it is imperative for the algorithm to detect the clique, since there are only $O(\sqrt{n})$ vertices in the clique, a lower bound of $\Omega(\sqrt{n})$ queries is needed.*

2.2 Overview of approach

We sketch the approach that gives good approximation: the idea is to divide the vertices into $\log n$ groups, and for each group, vertices have average degree within a $(1 \pm \beta)$ multiplicative factor from each other.

The algorithm is then sample vertices, query degrees and throw vertices into corresponding group. The obvious problem is that some groups might not have enough samples. For these groups, we simply set the degrees of the empty spots to 0. Notice this will end up with under counting the total degrees. In next lecture, we will show that the aforementioned approach leads to a 2-approximation.