Lecture 20

Sublinear-Time Algorithms

Supplemental reading in CLRS: None

If we settle for approximations, we can sometimes get much more efficient algorithms. In Lecture 18, we saw polynomial-time approximations to a few NP-hard problems. In what follows, we will concern ourselves with *sublinear*-time (that is, $o(n)$-time) approximation algorithms to problems whose exact solutions require at least linear time. We will see two examples:

1. Estimating the number of connected components of a graph G
2. Estimating the size of a minimum spanning tree of a graph G, given that all of G's edge weights are integers between 1 and $\digamma$ for some given constant $\digamma$.[1]

20.1 Estimating the Number of Connected Components

In this section we exhibit a polynomial-time approximation scheme for counting the connected components of a graph. Given a graph G (in adjacency-list format) with n vertices and parameters $\epsilon, \delta > 0$, we exhibit a randomized approximation algorithm which, with probability $1-\delta$, provides an additive ϵn-approximation. That is,

$$\Pr\left[\ \left|\begin{pmatrix}\text{output of}\\ \text{algorithm}\end{pmatrix} - \begin{pmatrix}\text{correct number of}\\ \text{connected components}\end{pmatrix}\right| > \epsilon n\ \right] \quad \le \quad \delta. \tag{20.1}$$

The running time of the algorithm is $\text{Poly}(1/\epsilon,\ \lg(1/\delta))$.[2]

Given a vertex v, let m_v denote the number of vertices in v's connected component. Take a moment to convince yourself of the following lemma:

Lemma 20.1. *The number of connected components in G is*

$$\sum_{v\in V} \frac{1}{m_v}.$$

[1] The archaic Greek letter $\digamma$ (digamma) stood for the sound /w/. Although digamma fell out of use by the time of Classical Greek, it is still occasionally used in Greek numerals (which are used in Greece in approximately the same way that we use Roman numerals) to represent the number 6.

[2] This notation means that the running time is polynomial in $1/\epsilon$ and $\lg(1/\delta)$; i.e., there exist positive constants r_1, r_2 such that the running time of the algorithm is $O\left(\left(\frac{1}{\epsilon}\right)^{r_1}\left(\lg\frac{1}{\delta}\right)^{r_2}\right)$.

The idea of the algorithm is to approximate m_v by a quantity $\widetilde{m}_v$ which can be computed in constant time. Then, for a set K of k randomly chosen vertices, we have

$$\begin{pmatrix}\text{number of connected}\\ \text{components}\end{pmatrix} = \sum_{v\in V}\frac{1}{m_v} \approx \sum_{v\in V}\frac{1}{\widetilde{m}_v} \approx \frac{n}{k}\sum_{v\in K}\frac{1}{\widetilde{m}_v}.$$

```
Algorithm: APPROX-#CC$(G,\epsilon,\delta)$
1  For some $k=\Theta\left(\frac{1}{\epsilon^2}\lg\frac{1}{\delta}\right)$, pick $k$ vertices $v_1,\dots,v_k$ at random
2  for $i \leftarrow 1$ to $k$ do
3      Set
           $\widetilde{m}_{v_i} \leftarrow \min\left\{m_{v_i}, \frac{2}{\epsilon}\right\}$
       ▷ computed using breadth-first search
4  return the value of
           $\frac{n}{k}\sum_{i=1}^{k}\frac{1}{\widetilde{m}_{v_i}}$
```

The key line to examine is line 3.

Exercise 20.1.

(i) The running time of a breadth-first search depends on the size of the graph. So how can there be a bound on the running time of line 3 that doesn't depend on n?

(ii) What is the running time of line 3? Show that the total running time of the algorithm is

$$O\left(\frac{1}{\epsilon^2}k\right) = O\left(\frac{1}{\epsilon^4}\lg\frac{1}{\delta}\right).$$

20.1.1 Correctness

We will prove (20.1) in two steps. First we will prove

$$\left|\sum_{v\in V}\frac{1}{\widetilde{m}_v} - \sum_{v\in V}\frac{1}{m_v}\right| \le \frac{\epsilon n}{2}; \tag{20.2}$$

then we will prove

$$\Pr\left[\ \left|\frac{n}{k}\sum_{i=1}^{k}\frac{1}{\widetilde{m}_{v_i}} - \sum_{v\in V}\frac{1}{\widetilde{m}_v}\right| \ge \frac{\epsilon n}{2}\ \right] \le \delta. \tag{20.3}$$

Combining these two, we obtain

$$\left|\frac{n}{k}\sum_{i=1}^{k}\frac{1}{\widetilde{m}_{v_i}} - \sum_{v\in V}\frac{1}{m_v}\right|$$

$$\le \left|\frac{n}{k}\sum_{i=1}^{k}\frac{1}{\widetilde{m}_{v_i}} - \sum_{v\in V}\frac{1}{\widetilde{m}_v}\right| + \left|\sum_{v\in V}\frac{1}{\widetilde{m}_v} - \sum_{v\in V}\frac{1}{m_v}\right|,$$

and with probability $1-\delta$,

$$\cdots \le \frac{\epsilon n}{2} + \frac{\epsilon n}{2} = \epsilon n,$$

which is (20.1).

Proof of (20.2). This follows from the fact that

$$0 \le \frac{1}{\widetilde{m}_v} - \frac{1}{m_v} < \frac{1}{\widetilde{m}_v} \le \frac{1}{\left(\frac{2}{\epsilon}\right)} = \frac{\epsilon}{2}$$

for each $v \in V$. □

Proof of (20.3). We use *Hoeffding's inequality*, a relative of the Chernoff bound which is stated as follows. Given independent real-valued random variables $X_1,\dots,X_k$, let $Y = \frac{1}{k}\sum_{i=1}^{k} X_i$. Suppose $a,b \in \mathbb{R}$ are constants such that always[3] $a \le X_i \le b$ for each i. Then Hoeffding's inequality states that for any $\eta > 0$, we have

$$\Pr\left[\left|Y - \mathbb{E}[Y]\right| \ge \eta\right] \le 2\exp\left(\frac{-2k\eta^2}{(b-a)^2}\right).$$

We take $X_i = \frac{1}{\widetilde{m}_{v_i}}$, which gives $\mathbb{E}[Y] = \frac{1}{n}\sum_{v\in V}\frac{1}{\widetilde{m}_v}$; we take $a = 0$ and $b = 1$ and $\eta = \epsilon/2$. Then Hoeffding's equality becomes

$$\Pr\left[\ \left|\frac{1}{k}\sum_{i=1}^{k}\frac{1}{\widetilde{m}_{v_i}} - \frac{1}{n}\sum_{v\in V}\frac{1}{\widetilde{m}_v}\right| \ge \frac{\epsilon}{2}\ \right] \le 2\exp\left(-2k\left(\frac{\epsilon^2}{4}\right)\right).$$

Thus, for a suitable $k = \Theta\left(\frac{\lg(1/\delta)}{\epsilon^2}\right)$ (namely, $k = \frac{2\ln(2/\delta)}{\epsilon^2}$), we have

$$\cdots \le 2\exp\left(-\ln\frac{2}{\delta}\right) = \delta.$$

This is equivalent to (20.3). □

20.2 Estimating the Size of a Minimum Spanning Tree

In this section, we exhibit an algorithm which solves the following problem:

Input: An undirected weighted graph $G = (V,E,w)$ with n vertices

all edge weights are integers in the set $\{1,\dots,\digamma\}$, where $\digamma \ge 2$ is a given parameter
all vertices have degree at most d
given in adjacency-list format

Parameters $\epsilon,\delta > 0$

Output: A number t such that, with probability at least $1-\delta$,

$$(1-\epsilon)w^* \le t \le (1+\epsilon)w^*,$$

where w^* is the weight of a minimum spanning tree.

The running time of the algorithm is $\text{Poly}\left(\frac{1}{\epsilon},\ \lg\frac{1}{\delta},\ \digamma,\ d\right)$.

[3] Or at least, with probability 1.

20.2.1 Motivation

Imagine running Kruskal's MST algorithm on a graph $G = (V, E, w)$ whose edge weights are all integers from the set $\{1, \ldots, \digamma\}$. The procedure would look like this:

```
1  $T \leftarrow \emptyset$
2  **for** $i \leftarrow 1$ **to** $\digamma$ **do**
3      **while** there exists an edge of weight $i$ which has its endpoints in different
       connected components **do**
4          add the edge to $T$
5          **if** $|T| = n - 1$ **then**
6              **return** $T$
```

The number of connected components in T starts at n and decreases by 1 every time we add an edge. This leads to the following insight, which will be crucial for us:

Observation 20.2. Let $G^{(i)} = (V, E^{(i)})$, where $E^{(i)} = \{e \in E : w(e) \le i\}$, and let $T^{(i)}$ be the restriction of T to $G^{(i)}$. Let $c^{(i)}$ be the number of connected components in $G^{(i)}$. Lines 3–6 (plus induction) guarantee that the number of connected components in $T^{(i)}$ is also $c^{(i)}$. Moreover, the number of connected components in $T^{(i)}$ is equal to

$$n - \left(\text{\# edges in } T^{(i)}\right)$$

(this is true for any forest with n vertices). Thus, we have

$$\begin{pmatrix}\text{\# edges in } T \text{ with} \\ \text{weight at most } i\end{pmatrix} = \left(\text{\# edges in } T^{(i)}\right) = n - c^{(i)}.$$

Observation 20.2 plus some clever algebra lead to the following lemma:

Lemma 20.3. *With $G^{(i)}$, $T^{(i)}$ and $c^{(i)}$ as above, we have*

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

Proof. Let

$$A_i = \begin{pmatrix}\text{\# edges it } T \text{ with} \\ \text{weight exactly } i\end{pmatrix} \quad \text{and} \quad B_i = \begin{pmatrix}\text{\# edges in } T \text{ with} \\ \text{weight at least } i\end{pmatrix}.$$

Then

$$\begin{aligned} B_i &= |T| - \begin{pmatrix}\text{\# edges in } T \text{ with} \\ \text{weight at most } i-1\end{pmatrix} \\ &= (n-1) - \left(n - c^{(i-1)}\right) \\ &= c^{(i-1)} - 1. \end{aligned}$$

Now, the clever algebra trick is to notice that

$$w(T) = \sum_{i=1}^{\digamma} i \cdot A_i = \sum_{i=1}^{\digamma} B_i.$$

(Make sure you see how this works.) Completing the computation,

$$\begin{aligned} w(T) &= \sum_{i=1}^{F} B_i \\ &= \sum_{i=1}^{F} \left(c^{(i-1)} - 1\right) \\ &= -F + \sum_{i=0}^{F-1} c^{(i)} \\ &= n - F + \sum_{i=1}^{F-1} c^{(i)}. \end{aligned}$$ □

Algorithm: MST-APPROX($G, \epsilon, \delta, F, d$)

```
1 for $i \leftarrow 1$ to $F - 1$ do
2     ▷ Let $G^{(i)}$ denote the subgraph of $G$ consisting of those edges whose weight is at
      most $i$
3     $\hat{c}^{(i)} \leftarrow$ APPROX-#CC$\left(G^{(i)}, \frac{\epsilon}{2F}, \frac{\delta}{F}\right)$
4 return the value of
      $|G.V| - F + \sum_{i=1}^{F-1} \hat{c}^{(i)}$
```

The tricky part here is that we cannot compute $G^{(i)}$ and store it in memory, as that would take $\Omega(n + G.E)$ time. So how does line 3 work? We must modify APPROX-#CC to so as to ignore any edges of weight greater than i. However, this modification forces us to reconsider the running time of APPROX-#CC, and is the reason for the dependence on d.

Exercise 20.2.

(i) *Suppose we modify* APPROX-#CC *so as to ignore any edges of weight greater than i. Use an example to show that, if we treat ϵ, δ and F as constants but do not allow any dependence on d, then the breadth-first search on line 3 of* APPROX-#CC *has worst-case running time $\Omega(n)$. (Hint: Every time we ignore an edge, it takes one step.)*

(ii) *Despite part (i), we can put a good bound on the running time of the modified version of* APPROX-#CC *if we allow the bound to depend on d. Show that the modified breadth-first search in line 3 of* APPROX-#CC *(with $\frac{\epsilon}{2F}$ and $\frac{\delta}{F}$ standing in for what in* APPROX-#CC *are denoted ϵ and δ) takes $O\left(\frac{dF}{\epsilon}\right)$ time, and thus that the total running time of* APPROX-MST *is*

$$O\left(\frac{dF^4}{\epsilon^3} \lg \frac{F}{\delta}\right).$$

20.2.2 Correctness

The return value of APPROX-MST is

$$n - F + \sum_{i=1}^{F-1} \hat{c}^{(i)},$$

while in §20.2.1 we showed that

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

Thus, to show correctness, we need to show that

$$\Pr\left[\left| \sum_{i=1}^{\digamma-1} c^{(i)} - \sum_{i=1}^{\digamma-1} \hat{c}^{(i)} \right| > \epsilon w^* \right] \le \delta.$$

By the union bound, it suffices to show that

$$\Pr\left[\left| c^{(i)} - \hat{c}^{(i)} \right| > \frac{\epsilon}{\digamma-1} w^* \right] \le \frac{\delta}{\digamma-1}$$

for each i. Now, because each edge has weight at least 1, we know that $w^* \ge n-1$. So it suffices to show that

$$\Pr\left[\left| c^{(i)} - \hat{c}^{(i)} \right| > \frac{\epsilon}{\digamma-1}(n-1) \right] \le \frac{\delta}{\digamma-1}.$$

The correctness of APPROX-#CC guarantees that

$$\Pr\left[\left| c^{(i)} - \hat{c}^{(i)} \right| > \frac{\epsilon}{2\digamma} n \right] \le \frac{\delta}{\digamma},$$

so we are fine as long as

$$\frac{\epsilon}{\digamma-1}(n-1) \ge \frac{\epsilon}{2\digamma} n,$$

which holds whenever $n > 1$. That is good enough.