0368.416701 Sublinear Time Algorithms 17th November 2014

Lecture 4

Lecturer: Ronitt Rubinfeld *Scribe: Coral F. Grichener, Omer A. Anson, Tomer Galanti*

Lesson Plan

1. Estimating the number of connected components in a graph
2. Estimating the weight of a Minimum Spanning Tree (MST) in a graph
3. Comparing distributed algorithms and sub-linear time algorithms

Part 1: Estimating the number of connected components in a graph

Given an undirected graph $G(V,E)$. Let $n \equiv |V|$ be the number of vertices, $m \equiv |E|$ be the number of edges, and $d = \max_{v \in V} \deg(v)$ be the maximum degree in the graph. We assume the graph is given in adjacency list representation. **Note:** This notation is standard, and will in appear in this manner throughout.

In general, sub-linear algorithms over graphs are considered sub-linear time in m. However, for this particular problem, it is possible that $m = 0$, so we the time complexity is considered sub-linear in time in $n + m$. Eventually, the time complexity will depend on d, and not on n or m.

Definition 1 *A connected component of an undirected graph is a sub-graph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the super-graph*

Algorithm: Estimate the number of connected components in undirected graph G
Input: $G(V,E)$, ε
Output: y s.t. $c - \varepsilon n \leq y \leq c + \varepsilon n$, where c is the number of connected components of G.

Note that there is a lower bound on the time complexity for this algorithm, in terms of ε.
In order to show and prove the algorithm, we begin with a few observations.

Definition 2 *Let v be a vertex in V. We define n_v as the number of vertices in the connected component to which v belongs.*

Observation 3 *Let A be a connected component in V. Then*

$$\sum_{v \in A} \frac{1}{n_v} = \sum_{v \in A} \frac{1}{|A|} = 1 \tag{1}$$

therefore

$$|VC| = \sum_{v \in V} \frac{1}{n_v} \tag{2}$$

is the number of connected components. This is what we would like to estimate.

Since we want a sub-linear time complexity, we cannot use search algorithms such as BFS or DFS. The time complexity for these algorithms is $O(|V| + |E|) = O(n^2)$.

We will show that we can estimate the sum in equation (2) quickly using standard Chernoff bounds.

Recall that d is the maximum degree in G. Let's consider d as a constant in the input. Since the graph is represented as an adjacency list, iterating all neighbours of a given vertex takes at most d steps. We will separate the estimation of equation (2) into two parts: 1) Part 1 will estimate $1/n_v$ and 2) part 2 will estimate the sum of the estimation using Chernoff bounds.

Part 1 - Estimation of $1/n_v$

Let $\hat{n_v}$ be defined as

$$\hat{n_v} \equiv \min\left\{n_v, \frac{2}{\varepsilon}\right\}$$

Therefore

$$\frac{1}{\hat{n_v}} = \max\left\{\frac{1}{n_v}, \frac{\varepsilon}{2}\right\}$$

It is considered accepted notation to treat ε s.t. $\varepsilon \ll 1$. Therefore $\frac{2}{\varepsilon} \gg 1$. We will get good estimate on small connected components. We are less concerned with large connected components, since it that case $\frac{1}{n_v}$ is small, and an error in its estimation will be small as well.

Let $\hat{c}$ be defined as

$$\hat{c} \equiv \sum_{v \in V} \frac{1}{\hat{n_v}}$$

Lemma 4 $\forall u. \left|\frac{1}{n_u} - \frac{1}{n_v}\right| \leq \frac{\varepsilon}{2}$

Proof There are two cases: 1) $n_u < \frac{2}{\varepsilon}$, and 2) $n_u \geq \frac{2}{\varepsilon}$. In the first case,

$$\begin{aligned} n_u &< \frac{2}{\varepsilon} \\ \implies \hat{n_u} &= n_u \\ \implies |n_u - \hat{n_u}| &= 0 < \frac{\varepsilon}{2} \end{aligned}$$

in the second case,

$$\begin{aligned} n_u &\geq \frac{2}{\varepsilon} \\ \implies \hat{n_u} &= \frac{2}{\varepsilon} \\ \frac{1}{n_u} &\leq \frac{\varepsilon}{2} \\ \implies 0 &\leq \frac{1}{n_u} \leq \frac{1}{\hat{n_u}} = \frac{\varepsilon}{2} \\ \implies \frac{1}{\hat{n_u}} - \frac{1}{n_u} &\leq \frac{\varepsilon}{2} \\ \implies \left|\frac{1}{\hat{n_u}} - \frac{1}{n_u}\right| &\leq \frac{\varepsilon}{2} \end{aligned}$$

■

Corollary 5 $|c - \hat{c}| \leq \frac{\varepsilon n}{2}$

Proof

$$\begin{aligned}
c &\equiv \sum_{v \in V} \frac{1}{n_v} \\
\hat{c} &\equiv \sum_{v \in V} \frac{1}{\hat{n_v}} \\
|c - \hat{c}| &= \left| \sum_{v \in V} \frac{1}{n_v} - \sum_{v \in V} \frac{1}{\hat{n_v}} \right| \\
&= \left| \sum_{v \in V} \left(\frac{1}{n_v} - \frac{1}{\hat{n_v}} \right) \right| \\
&\leq \sum_{v \in V} \left| \frac{1}{n_v} - \frac{1}{\hat{n_v}} \right| \\
&\leq \sum_{v \in V} \frac{\varepsilon}{2} \\
&= \frac{\varepsilon n}{2}
\end{aligned}$$

■

Note that we have constructed the estimate of $\hat{n_v}$ s.t. the estimate of $|c - \hat{c}|$ has a factor of half. This allows us room for the additive error in part 2 as well.

Algorithm: Find a lower bound on the size of this connected component

Outline: We will use a search algorithm (e.g. BFS) from vertex v until it has scanned the entire connected component to which v belongs, or until it has seen at most $\frac{2}{\varepsilon}$ nodes. The time complexity is $O\left(d \cdot \frac{2}{\varepsilon}\right) = O\left(\frac{d}{\varepsilon}\right)$, since every step of BFS has time complexity of at most d, and there are at most $\frac{2}{\varepsilon}$ such steps.

Output: Number of visited nodes

We have seen that we can estimate $\frac{1}{n_v}$ in $O\left(\frac{d}{\varepsilon}\right)$ time for vertex v.

Part 2: Estimating the sum

We will find $\tilde{c}$, an estimation of $\hat{c}$, which itself is an estimation of c.

Algorithm: Estimate $\hat{c}$, and estimation of c, the number of connected components.

Outline: Let $r \equiv \frac{b}{\varepsilon^3}$, where b is some constant we will define later. Choose $U = \{u_1, \ldots, u_r\}$ random vertices. For all $u_i \in U$, compute $\hat{n_{u_i}}$ using the algorithm in part 1 above.

Output: $\tilde{c} = \frac{n}{r} \sum_{u_i \in U} \frac{1}{\hat{n_{u_i}}}$. We use $\frac{n}{r}$ as the scale factor, since we want the sum over n, but we sum only $r$elements.

Runtime: $O\left(\frac{d}{\varepsilon} \cdot r\right) = O\left(\frac{d}{\varepsilon^4}\right)$. Recall that b is constant, even though we have not defined it yet.

Reminder: Chernoff Bound:

Let $x_1, \ldots, x_r$ be independant random variables, in the range $x_i \in [0, 1]$. Let $S = \sum_i x_i$ be the sum of these variables, $p = E(x_i) = \frac{E(S)}{r}$ the expected value of these variables. Then

$$Pr\left(\left| \frac{S}{r} - p \right| \geq \delta p \right) \leq e^{-\Omega\left(rp\delta^2\right)}$$

Theorem 6 $Pr\left(|\tilde{c}-\hat{c}| \leq \frac{\varepsilon^2}{2}\right) \geq \frac{3}{4}$

Let $p = E\left(\frac{1}{\hat{n_u}}\right)$, $S = \sum_{u \in V} \frac{1}{\hat{n_u}}$, $\delta = \frac{\varepsilon}{2}$. *Using Chernoff bounds, we get*

$$Pr\left(\left|\frac{1}{r}\sum_{i=1}^{r}\frac{1}{\hat{n_{u_i}}} - E\left(\frac{1}{\hat{n_u}}\right)\right| \geq \frac{\varepsilon}{2}\cdot E\left(\frac{1}{\hat{n_u}}\right)\right) \leq e^{-\Omega\left(r\cdot\frac{\varepsilon^2}{4}\cdot E\left(\frac{1}{\hat{n_u}}\right)\right)}$$

Note the following equalities:

$$\begin{aligned}
\Omega\left(r\cdot\frac{\varepsilon^2}{4}\cdot E\left[\frac{1}{\hat{n_u}}\right]\right) &= \Omega\left(r\cdot\frac{\varepsilon^2}{4}\cdot\frac{\hat{c}}{n}\right) = \Theta\left(\frac{b}{\varepsilon}\cdot\frac{\hat{c}}{n}\right)\\
\frac{1}{r}\sum_{i=1}^{r}\frac{1}{\hat{n_{u_i}}} &= \frac{\tilde{c}}{n}\\
E\left(\frac{1}{\hat{n_u}}\right) &= \frac{1}{n}\sum\frac{1}{\hat{n_u}} = \frac{\hat{c}}{n}
\end{aligned}$$

Therefore,

$$\begin{aligned}
Pr\left(\left|\frac{1}{r}\sum_{i=1}^{r}\frac{1}{\hat{n_{u_i}}} - E\left(\frac{1}{\hat{n_u}}\right)\right| \geq \frac{\varepsilon}{2}\cdot E\left(\frac{1}{\hat{n_u}}\right)\right) &\leq e^{-\Omega\left(r\cdot\frac{\varepsilon^2}{4}\cdot E\left(\frac{1}{\hat{n_u}}\right)\right)}\\
\implies Pr\left(\left|\frac{\tilde{c}}{n}-\frac{\hat{c}}{n}\right| \geq \frac{\varepsilon}{2}\cdot\frac{\hat{c}}{n}\right) &\leq e^{-\Theta\left(\frac{b}{\varepsilon}\cdot\frac{\hat{c}}{n}\right)}\\
\implies Pr\left(|\tilde{c}-\hat{c}| \geq \frac{\varepsilon}{2}\cdot\hat{c}\right) &\leq e^{-\Theta\left(\frac{b}{\varepsilon}\cdot\frac{\hat{c}}{n}\right)}
\end{aligned}$$

Note that $\frac{\varepsilon n}{2} \leq \hat{c} \leq n$, *immediate from the defintiion of* $\hat{c}$. *Therefore,*

$$\begin{aligned}
\frac{\varepsilon n}{2} &\leq \hat{c} \leq n\\
\implies \hat{c} \leq n \quad & \text{and } \frac{\hat{c}}{n\varepsilon} \geq \frac{1}{2}\\
\implies Pr\left(|\tilde{c}-\hat{c}| > \frac{\varepsilon}{2}n\right) &\leq Pr\left(|\tilde{c}-\hat{c}| > \frac{\varepsilon}{2}\hat{c}\right)\\
&\leq e^{-\Theta\left(\frac{b}{\varepsilon}\cdot\frac{\hat{c}}{n}\right)}\\
&\leq e^{-\Theta\left(\frac{b}{2}\right)}
\end{aligned}$$

So for proper constant b, *we get*

$$Pr\left(|\tilde{c}-\hat{c}| \geq \frac{\varepsilon}{2}\cdot n\right) \leq \frac{1}{4} \tag{3}$$

Corollary 7 $Pr\left(|c-\tilde{c}| \leq \varepsilon n\right) \geq \frac{3}{4}$, *since* $|c-\tilde{c}| \leq |c-\hat{c}| + |\hat{c}-\tilde{c}|$

Minimum Spanning Tree

Definition 8 *Given a connected, undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together.*

Definition 9 *Let* T *be a spanning tree in* G. *Let* $w(T)$ *be the sum of the weights of all edges in* T. *We say that* T *is the Minimum Spanning Tree (MST) if* $M = w(T)$ *is minimal.*

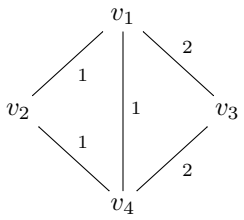

Figure 1: This is an example of a graph with $M = w(T) = 4$. It can be seen that $C^{(1)} = 2$, and $\alpha_1 = 2$, $\alpha_2 = C^{(1)} - 1 = 1$.

Assumption 10 *G is connected.*

G is connected implies that $w(T) \leq (n-1) \cdot \omega$ and $w(T) \geq (n-1)$.

We present an algorithm in sub-linear time that estimates the weight of the MST, $M = w(T)$. Note that ω will be used to bound the time complexity.

Algorithm: Estimate the weight of the MST, $M = w(T)$

Input: Undirected graph $G(V, E)$, with maximum degree d, in adjacency list format. Each edge $\langle u, v \rangle$ has weight w_{uv}. In this case, we assume $w_{uv} \in \{1 \dots \omega\} \cup \{\infty\}$, where $w_{uv} = \infty \iff \langle u, v \rangle \notin E$.

Aside: It is possible for w_{uv} to be continuous, and not discrete, but this will cost in time complexity. The continuous case can also be done sub-linearly in time.

Output: $\hat{M}$, an estimation of $M = w(T)$, s.t. $(1-\varepsilon) M \leq \hat{M} \leq (1+\varepsilon) M$.

We will see that the algorithm generates an additive, and not a multiplicative, error. Once we have an additive estimate, we will see how to get the multiplicative estimate. The estimation itself of $M = w(T)$ will be done using the number of connected components.

Definition 11 *Let $G^{(i)} = \left(V, E^{(i)}\right)$, where $E^{(i)} = \{\langle u, v \rangle \in E | w_{uv} \in \{1 \dots i\}\}$ is E with all the edges with weights greater than i deleted, and define $C^{(i)}$ to be the number of connected components in $G^{(i)}$*

Observation 12 *If for all weights in G, $w_{uv} = 1$, then $M = n - 1$, and the number of connected components in $G^{(1)}$ is $C^{(1)} = 1$*

See figure 1 for an example graph.

Claim 13 *In figrure 1, $M = 2 \cdot \left(C^{(2)} - 1\right) + 1 \cdot \left(n - 1 - \left(C^{(1)} - 1\right)\right)$. Recall that $C^{(2)}$ is the number of connected components in G, after deleting all edges with weights greater than 2*

Claim 14 *Generalisation of claim 13: $M = n - \omega + \sum_{i=1}^{\omega-1} C^{(i)}$*

Proof Let α_i be the number of edges with weights i in any MST. Note that all MSTs have the same value of α_1. $\sum_{i>l} \alpha_i$ is one less the number of connected components of $G^{(l)}$, $C^{(l)} - 1$.

$$\begin{aligned} \sum_{i>l} \alpha_i &= C^{(l)} - 1 \\ C^{(0)} &\equiv n \end{aligned} \tag{4}$$

In figure 1, it can be seen that $\alpha_2 = C^{(1)} - 1 = 1$, and $\alpha_1 = 2 = C^{(0)} - 1$. This is what we get from formula (4).

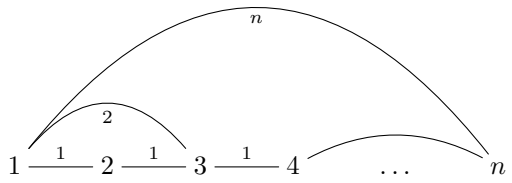

Figure 2: A graph with a line metric. It can be seen that in this case, the complexity becomes $O\left(dn^5/\varepsilon^4\right)$, which is even worse than linear.

$$\begin{aligned} M &= \sum_{i=1}^{\omega} i \cdot \alpha_i \\ &= \sum_{i=1}^{\omega} \alpha_i + \sum_{i=2}^{\omega} \alpha_i + \cdots + \sum_{i=\omega}^{\omega} \alpha_i \\ &= \left(C^{(0)} - 1\right) + \left(C^{(1)} - 1\right) + \cdots + \left(C^{(\omega-1)} - 1\right) \\ &= n - \omega + \sum_{i=1}^{\omega-1} C\left(i\right) \end{aligned}$$

■

Note that the first term is the number of all edges in the MST. The second term is the number of all edges in the MST with weights greater or equal to 2. We have shown above that this is $\left(C^{(1)} - 1\right)$.

This means that to estimate M, we need just to estimate $C^{(i)}$, or more precisely, to estimate $\sum_{i=1}^{\omega-1} C^{(i)}$.

Algorithm: Estimate the weight of the MST, $M = w\left(T\right)$, using connected components

Outline:

1. For $i = 1$ to $\omega - 1$,
   (a) $\hat{C^{(i)}} \leftarrow$estimation of the number of connected components of $G^{(i)}$ to within $\frac{\varepsilon}{2\omega}$ additive error.

Output: $\hat{M} = n - \omega + \sum_{i=1}^{\omega-1} \hat{C^{(i)}}$

By summing over the additive error, we'll get the additive error $\frac{\varepsilon}{2\omega} \cdot (\omega - 1) \leq \frac{\varepsilon}{2\omega} \cdot \omega = \frac{\varepsilon}{2}$.

The weights are bounded between 1 and ω. This comes in not only in the error term, but also in the running time complexity.

Runtime: $O\left(\omega \cdot \frac{d}{\left(\frac{\varepsilon}{2\omega}\right)^4}\right) = O\left(\frac{d\omega^5}{\varepsilon^4}\right)$.

Other algorithms can get the running time complexity down to $O\left(\frac{d\omega}{\varepsilon^2} \log \frac{d\omega}{\varepsilon^2}\right)$, but the lower bound was found to be $\Omega\left(\frac{d\omega}{\varepsilon^2}\right)$.

This isn't good for groups with big ω. See figure 2 for an example.

It is possible to call the connected components algorithm shown earlier to be with error probability at most $\frac{1}{4\omega}$, since its error probability is less than $1/2$.

Claim 15 *Approximation guarantee: If the call to the approximation algorithm of connected components is with error probability less than or equal to $\frac{1}{4\omega}$, then all calls give approximation with error probability less than or equal to $\frac{\omega}{4\omega} = \frac{1}{4}$.*

Proof $\left|M - \hat{M}\right| \leq \omega \cdot \frac{\varepsilon}{2\omega} \cdot n = \frac{\varepsilon n}{2}$ additive error, which comes only from the error on $C^{(i)}$. We use a half factor in the error to approximate $n-1$ to n. Since $M \geq n-1$, and we can assume $n > 3$, we get $M \geq n/2$, since $n - 1 > \frac{n}{2}$. Therefore, $\left|M - \hat{M}\right| \leq \varepsilon M$. ■

Aside: We can get $G^{(i)}$ from G by ignoring any edge with weight greater than i. This can therefore be done in constant time.

A sense of the relationship between distributed algorithms and sub-linear time algorithms

In this section, we aim to show a correspondence between distributed and sub-linear algorithms. We will mainly consider tasks dealing with sparse graphs (i.e. graphs with maximum degree $\deg(G) \leq d$). The representation of the graphs will be held by adjacency lists.

Preliminaries and Motivations

We have two desired motivations:

- Explain the correspondence between fast distributed algorithms and sub-linear time algorithms.
- We will demonstrate this correspondence using the Vertex Cover (VC) problem. We introduce a sub-linear time version of this task and a distributed type solution for it.

The Vertex Cover Problem

Definition 16 *Given an undirected graph $G(V, E)$, a set of vertices $V' \subseteq V$ is called a Vertex Cover (VC) if $\forall (u, v) \in E$, either u or v in V'.*

Question: What is the minimum size VC in a given graph G?

Note: For an undirected graph $G(V, E)$ s.t. $\deg(G) \leq d$, the minimal VC must contain at least m/d vertices, since each vertex can cover at most d edges.

This problems is NP-complete, but there is a poly-time 2-mult approximation for this problem.

Question: Can we approximate the vertex cover problem in sub-linear time?

Multiplicative: No.

Explanation: For instance, given two graphs of the same size, one with only one edge and another with no edges. In the first $|VC| > 0$ and for the second $|VC| = 0$. Such an algorithm must distinguish between both cases (to answer $|VC| > 0$ for the first and $|VC| = 0$ for the second) in sub-linear time (intractable).

Additive: Hard.

Explanation: Need some multiplicative error - computationally hard to do.

In order to solve this situation, we will take a combination of both.

Definition 17 *We refer $\hat{y}$ as (α, ϵ)-estimate of solution y for a minimization problem if $y \leq \hat{y} \leq \alpha \cdot y + \epsilon$ i.e, we allow both multiplicative and additive errors on the same time.*

Distributed algorithms

Background on distributed algorithms

In our context we take the following abstraction to distributed algorithms. We view distributed algorithms as networks with the following.

1. Nodes: each node denotes a processor in the network.
2. Links: each link connects between two processor units.
3. Communication rounds: in each communication round, each of the processors sends messages to it's neighbors.

Definition 18 *The VC problem for distributed networks:*

- *Network graph: Input graph. The network computes on itself!*
- *Termination time: Each node knows if it is part of the VC or not.*

Corresponding distributed and sub-linear time algorithms

We examine the VC problem. In a k-round algorithm, the output of each node v depends only on nodes at distance at most k from v. i.e, at most d^k nodes! In other words, it simulates v's view of the distributed computation in less than d^k steps, and figures out if v is contained (or not contained) in the VC.

Note: if the algorithm is randomized, v receives random bits (or constructs them by itself). These bits must be consistent with those of the d^k neighbours.

In advance, we will introduce some VC distributed algorithms (often called "local distributed algorithms"). We will use this to approximate VC in sub-linear time. For this purpose we will first introduce the following framework.

Parnas-Ron framework:

- Sample nodes from the graph $v_1, ..., v_r$.
- For each v_i (in the sample set) simulate the distributed algorithm to see if $v_i \in VC$ or not.
- **Output:** $\frac{\#v_i \in VC}{r} \cdot n$.

Runtime: $O(r \cdot d^k)$.
Sketch of Proof We don't introduce the entire proof of correctness. Just notice it is based on Chernoff/Hoeffding bounds. ■

Fast distributed algorithm for solving the VC problem:

1. $i \leftarrow 1$.
2. While edges remain:
   (a) Remove vertices v of degree $\deg(v) > d/2^i$ and adjacent edges.
   (b) Update degrees of remaining nodes.
   (c) Increment i.
3. **Output**: all removed vertices are the VC.

We restrict the number of rounds to $log(d)$.

Lemma 19 *The output is a VC.*

Proof By the definition of the algorithm, at the end there are no remaining edges. In addition, all edges are removed along to some adjacent vertex. ■

Theorem 20 *Let* $VC(G) = \min|VC|$. *Then:*

$$VC(G) \leq output \leq (2logd + 1) \cdot VC(G)$$

Proof The first inequality holds by lemma 19. In order to prove the second inequality, we claim that in each iteration we add at most $2VC(G)$ vertices.

Explanation: in each iteration, all nodes v removed have degree $d/2^i \leq \deg(v) \leq d/2^{i-1}$ (by 2a in the algorithm). So, if θ is some $\min(VC)$ and X is the set of removed vertices of some iteration which aren't in θ. ■