0368.416701 Sublinear Time Algorithms	October 26, 2009

Lecture 2

Lecturer: Ronitt Rubinfeld — *Scribe: Svetlana Olonetsky & Iftah Gamzu*

1 Lecture Outline

- Chernoff bound
- Estimating the number of connected components
- Estimating the weight of the minimum spanning tree
- Distributed algorithms vs. sublinear time algorithms

2 Chernoff Bound

Let $X_1, X_2, \ldots, X_m$ be m independent identically distributed random variables such that $X_i \in [0,1]$. Let $S = \sum_{i=1}^m X_i$ and $p = \mathrm{E}[X_i] = \mathrm{E}[S]/m$. Then,

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

3 Estimating the Number of Connected Components

Given an undirected graph $G(V,E)$ having n nodes and maximal degree d (in an adjacency list representation), and ϵ, we want to find an ϵn-additive estimate of the number of connected components. Specifically, if c denotes the number of connected components in G then the estimated number of connected components y should satisfy

$$c - \epsilon n \leq y \leq c + \epsilon n$$

Definition 1 *Let n_u be the number of nodes in u's connected component.*

Observation 2 *For any connected component $A \subseteq V$, we have*

$$\sum_{u \in A} \frac{1}{n_u} = \sum_{u \in A} \frac{1}{|A|} = 1 \ .$$

Furthermore, this implies that the number of connected components c is equal to

$$c = \sum_{u \in V} \frac{1}{n_u} \ .$$

Definition 3 *Let $\hat{n}_u = \min\{n_u, 2/\epsilon\}$, and let $\hat{c} = \sum_{u \in V} 1/\hat{n}_u$.*

The following lemma bounds the amount by which the estimates can be off.

Lemma 4 *For any node u, it holds that*

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

Proof. We know that $\hat{n}_u \leq n_u$ by the definition of $\hat{n}_u$. If $n_u \leq 2/\epsilon$ then $\hat{n}_u = n_u$, and therefore, the left hand side in the above inequality is equal to 0. If $n_u > 2/\epsilon$ then $\epsilon/2 = 1/\hat{n}_u \geq 1/n_u \geq 0$ and the lemma follows. ∎

Corollary 5 $|c - \hat{c}| \le \epsilon n/2$.

Lemma 6 *We can compute* $\hat{n}_u$ *in* $O(d/\epsilon)$ *time.*

Proof. We begin by presenting the algorithm that computes $\hat{n}_u$.

estimate_cc(u)
  run BFS from u until:
    • visited the whole connected component
    • or visited $2/\epsilon$ distinct nodes of the connected component
  output the number of visited nodes

It is clear that during execution of the algorithm at most $2/\epsilon$ nodes are visited. Since the degree of each node is at most d, the running time of the algorithm is $O(d/\epsilon)$. ∎

We now present algorithm approx_num_cc(G, ϵ), which calculates an ϵn-additive estimation of the number of connected components.

approx_num_cc(G, ϵ):
  choose a set $U = \{u_1, u_2, \ldots, u_r\}$ of $r = \Theta(1/\epsilon^3)$ random nodes
  for each $u \in U$ compute $\hat{n}_u$ using estimate_cc(u)
  output $\tilde{c} = \frac{n}{r} \sum_{u \in U} \frac{1}{\hat{n}_u}$

One can easily verify that the running time of the algorithm is $O(1/\epsilon^3 \cdot d/\epsilon) = O(d/\epsilon^4)$. We turn to prove that $\tilde{c}$ is an ϵn-additive estimation of c with constant probability.

Theorem 7 $\Pr(|\tilde{c} - \hat{c}| \le \epsilon n/2) \ge 3/4$.

Proof. We apply the Chernoff bound from Section 2 with $p = \mathrm{E}[1/\hat{n}_{u_i}]$, $S = \sum_{i=1}^{r} 1/\hat{n}_{u_i}$, $m = r$, and $\delta = \epsilon/2$, and get that

$$\Pr\left(\left|\frac{1}{r}\sum_{i=1}^{r}\frac{1}{\hat{n}_{u_i}} - \mathrm{E}\left[\frac{1}{\hat{n}_{u_i}}\right]\right| \ge \frac{\epsilon}{2}\mathrm{E}\left[\frac{1}{\hat{n}_{u_i}}\right]\right) \le \exp\left(-\Omega\left(r\mathrm{E}\left[\frac{1}{\hat{n}_{u_i}}\right]\left(\frac{\epsilon}{2}\right)^2\right)\right) .$$

Notice that $\tilde{c} = n/r \cdot \sum_{i=1}^{r} 1/\hat{n}_{u_i}$, $\mathrm{E}[1/\hat{n}_{u_i}] = 1/n \cdot \sum_{i=1}^{n} 1/\hat{n}_{u_i} = \hat{c}/n$, and $r = \Theta(1/\epsilon^3)$, and thus,

$$\Pr\left(\left|\frac{\tilde{c}}{n} - \frac{\hat{c}}{n}\right| \ge \frac{\epsilon}{2}\frac{\hat{c}}{n}\right) = \Pr\left(|\tilde{c} - \hat{c}| \ge \frac{\epsilon}{2}\hat{c}\right) \le \exp\left(-\Omega\left(r\frac{\hat{c}}{n}\left(\frac{\epsilon}{2}\right)^2\right)\right) = \exp\left(-\Omega\left(\frac{1}{\epsilon}\frac{\hat{c}}{n}\right)\right)$$

By the definition of $\hat{n}_u$, we know that $\epsilon/2 \le 1/\hat{n}_u \le 1$, and therefore, $\epsilon n/2 \le \hat{c} \le n$. Consequently, we attain that

$$\Pr\left(|\tilde{c} - \hat{c}| \ge \frac{\epsilon}{2}n\right) \le \Pr\left(|\tilde{c} - \hat{c}| \ge \frac{\epsilon}{2}\hat{c}\right) \le e^{-\Omega(1)} < \frac{1}{4} .$$

∎

Corollary 8 $\Pr(|c - \tilde{c}| \le \epsilon n) \ge 3/4$.

Proof. By Corollary 5 and the triangle inequality $|c - \tilde{c}| \le |c - \hat{c}| + |\hat{c} - \tilde{c}|$, one can obtain that $\Pr(|c - \tilde{c}| \le \epsilon n) = \Pr(|\tilde{c} - \hat{c}| \le \epsilon n/2)$. ∎

4 Estimating the Weight of the Minimum Spanning Tree

4.1 Problem statement

The input for the problem is a connected undirected graph $G = (V, E)$ in which the degree of each node is at most d. Furthermore, each edge (i, j) has an integer weight $w_{ij} \in [w] \cup \{\infty\}$. Note that the graph is given in an adjacency list format, and edges of weight ∞ do not appear in it. The goal is to find the weight of a minimum spanning tree (MST) of G. Specifically, if we let $w(T) = \sum_{(ij)\in T} w_{ij}$ for $T \subseteq E$, then our objective is to find

$$M = \min_{T \text{ spans } G} w(T) \ .$$

Since we are interested in sublinear time algorithms for this problem, and therefore, cannot hope to find M, we focus on finding an ϵ-multiplicative estimate of M, that is, a weight $\hat{M}$ which satisfies

$$(1-\epsilon)M \le \hat{M} \le (1+\epsilon)M \ .$$

We note that $n-1 \le M \le w \cdot (n-1)$, where $n = |V|$. This follows since G is connected, and thus, any spanning tree of it consists of $n-1$ edges, and by the assumption on the input weights.

4.2 From motivation to characterization

In what follows, we relate the weight of a MST of G to the number of connected components in certain subgraphs of G. We begin by introducing the following notation for a graph G:

- Let $G^{(i)} = (V, E^{(i)})$ be the subgraph of G that consists of the edges having a weight of at most i.
- Let $C^{(i)}$ be the number of connected components in $G^{(i)}$.

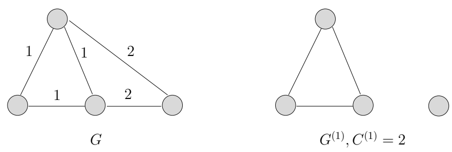

Figure 1: A graph G having $w = 2$, and its induced subgraph $G^{(1)}$.

A motivation. Let us consider two simple cases. The first case is when $w = 1$, namely, all the edges of G have a weight of 1. In this case, it is clear that the weight of a MST is $n-1$. Now, let us consider the case that $w = 2$, and let us focus on $G^{(1)}$. Clearly, one has to use $C^{(1)} - 1$ edges (of weight 2) to connect the connected components in $G^{(1)}$. This implies that the weight of a MST in this case is

$$2 \cdot (C^{(1)} - 1) + 1 \cdot (n - 1 - (C^{(1)} - 1)) = n - 2 + C^{(1)} \ .$$

The characterization. We extend and formalize the intuition presented above. Specifically, we characterize the weight of a MST of G using the $C^{(i)}$'s, for any integer w.

Claim 9 $M = n - w + \sum_{i=1}^{w-1} C^{(i)}$

Proof. Let α_i be the number of edges of weight i in any MST of G. Remark that it is well-known that all minimum spanning trees of G have the same number of edges of weight i, and hence, the α_i's are well defined. It is easy to validate that the number of edges having weight greater than ℓ is equal to the number of connected components in $G^{(\ell)}$ minus 1. That is, $\sum_{i=\ell+1}^{w} \alpha_i = C^{(\ell)} - 1$, where $C^{(0)}$ is set to be n. Now, notice that

$$\begin{aligned} M &= \sum_{i=1}^{w} i \cdot \alpha_i \\ &= \sum_{i=1}^{w} \alpha_i + \sum_{i=2}^{w} \alpha_i + \sum_{i=3}^{w} \alpha_i + \ldots + \alpha_w \\ &= (n-1) + (C^{(1)} - 1) + (C^{(2)} - 1) + \ldots + (C^{(w-1)} - 1) \\ &= n - w + \sum_{i=1}^{w-1} C^{(i)} \end{aligned}$$

■

4.3 Approximation algorithm

Algorithm MST_approx, formally defined below, estimates the weight of the MST.

MST_approx(G, ϵ, w)
  for $i = 1$ to $w - 1$
    $\hat{C}^{(i)} =$ approx_num_cc$(G^{(i)}, \epsilon/(2w))$
  output $\hat{M} = n - w + \sum_{i=1}^{w-1} C^{(i)}$

Running time. One can easily see that there are w calls to approx_num_cc. Recall that the running time of this procedure is $O(d/(\epsilon/(2w))^4) = O(dw^4/\epsilon^4)$, and hence, the running time of MST_approx is $O(dw^5/\epsilon^4)$. It is worth noting that rather than extracting $G^{(i)}$ from G for each call of approx_num_cc (which would make the algorithm have non-sublinear time), we simply modify approx_num_cc so it ignores edges with weight greater than i.

Approximation guarantee. We establish that $(1-\epsilon)M \leq \hat{M} \leq (1+\epsilon)M$ with high probability (whp). For this purpose, recall that approx_num_cc outputs an estimation $\hat{C}^{(i)}$ of the number of connected components which satisfies $|\hat{C}^{(i)} - C^{(i)}| \leq n\epsilon/(2w)$ whp. Consequently, we get that $|M - \hat{M}| \leq n\epsilon/2$ whp. Notice that $M \geq n-1 \geq n/2$, where the last inequality is valid for any "interesting" n, i.e., $n \geq 2$. Therefore, $|M - \hat{M}| \leq M\epsilon$, which completes the proof.

Concluding remark. The current state of the art algorithm for finding an ϵ-multiplicative estimate of M has a running time of $O(dw/\epsilon^2 \cdot \log dw/\epsilon)$. On the lower bound side, it is known that the running time of any algorithm must be $\Omega(dw/\epsilon^2)$.

5 Distributed Algorithms vs. Sublinear Time Algorithms

We introduce a definition and a theorem, which will be used in the next lesson.

Definition 10 *$\hat{y}$ is an* (α, ϵ)-estimate *of a solution value y for a minimization problem of size n if*

$$y \leq \hat{y} \leq \alpha y + \epsilon n \ .$$

Theorem 11 (Vizing's Theorem) *Every graph is edge-colorable*[1] *with at most $d + 1$ colors, where d is the maximum degree of the graph.*

Corollary 12 *Every graph whose maximum degree is d has a matching of size at least $|E|/(d + 1)$.*

Corollary 13 *The vertex cover size of every graph whose maximum degree is d is at least $|E|/(d + 1)$.*

[1] An *edge coloring* of a graph is an assignment of colors to the edges of the graph so that no two adjacent edges have the same color.