COMS E6998: Algorithms for Massive Data (Fall'25) Nov 10, 2025

Lecture 19: Sublinear Time Algorithms: MST, Vertex Cover

Instructor: *Prof. Alex Andoni* Scribes: *Thea Zhu*

1 Estimating MST in Graphs

We aim to estimate the minimum spanning tree (MST) in an undirected graph G with edge weights in $[1, M]$, where G is connected.

Let G_i be the graph obtained from G by keeping only edges with weights $\leq i$. Let CC_i be the number of connected components of G_i.

$$\text{Fact: } \text{MST}(G) = n - 1 + \sum_{i=1}^{M-1} (CC_i - 1)$$

and hence

$$\text{MST}(G) \geq n - 1.$$

Our goal is to estimate this sum up to a $(1 + \varepsilon)$ factor.

Estimator Construction

To estimate C_i, use the estimator:

$$\hat{CC}_i = \frac{n}{k} \sum_{v_j=1} \frac{1}{\alpha_{v_j}}, j = 1...k$$

where $\alpha_{v_j} = \min\{\text{size of cc}(v_j), 1/\delta\}$ and δ is a sampling parameter.

Claim 1. $\#\ samples = O\big(Mk\frac{1}{\delta}d\big)$

Claim 2. $\mathbb{E}[|\hat{CC}_i - CC_i|] \leq \delta n$

We set $\delta = \varepsilon/M$.

Claim 3.

$$\text{Var}[\hat{CC}_i] \leq O\Big(\frac{n}{k} \cdot (CC_i + \delta^2 n)\Big)$$

$$\begin{aligned}
\mathrm{Var}[\hat{CC}_i] &= \frac{n^2}{k^2}\, k \cdot \mathrm{Var}\left[\frac{1}{\hat{\alpha}_{v_j}}\right] \\
&= \frac{n^2}{k} \cdot \mathrm{Var}\big[\max\{\delta, \alpha_{v_j}\}\big] \\
&\le \frac{n^2}{k} \cdot \mathbb{E}_{v_j}\big[\max\{\delta, 1/\alpha_{v_j}\}^2\big] \\
&\le \frac{n^2}{k} \cdot \mathbb{E}_{v_j}\left[\delta^2 + \frac{1}{\alpha_{v_j}^2}\right] \\
&\le \frac{n}{k} \cdot \left(\delta^2 n + \mathbb{E}_{v_j}\left[\frac{1}{\alpha_{v_j}}\right] \cdot n\right) \\
&= \frac{n}{k} \cdot (\delta^2 n + CC_i) \quad \square
\end{aligned}$$

Claim 4. *With $\delta = \Theta(\varepsilon/M)$ and $k = \Theta(1/\delta^2)$, we have*

$$\Pr[MST(G) \in (1 \pm \varepsilon)\, MST(G)] \ge 0.9.$$

Overall number of samples:

$$O\left(\frac{M}{\varepsilon^2} \cdot d\right),$$

where d is the maximum degree.

Theorem 5 (Chakrabarti, Chuzhoy, Saranurak, and Sohler)**.** *Fix a matrix M with query access. One can estimate the cost of* $\mathrm{MST}(G)$ *up to a $(1 \pm \varepsilon)$ factor using*

$$O\left(n \cdot \left(\frac{\log n}{\varepsilon^2}\right)^{O(1)}\right)$$

queries and time.

2 Estimating Size of Vertex Cover

Definition 6. *A set S is a vertex cover (VC) of an undirected graph $G = (V, E)$ if and only if for every edge $(i, j) \in E$, we have $i \in S$ or $j \in S$.*

The problem is NP-hard.

Fact 7. *We can obtain a 2-approximation efficiently by taking a maximal matching M and setting S to include all endpoints of edges in M. Then $|S| = 2|M|$.*

Proof. M is maximal iff we cannot add another edge to M. If there exists an edge e not touching M, it implies $M \cup \{e\}$ is also a matching, a contradiction. Thus S is a valid vertex cover. $\square$

Algorithm for 2-Approximation of Vertex Cover

1. Compute a maximal matching M.
2. Set S to be all nodes that are endpoints of edges in M.
   Hence, $|S| = 2|M|$.

Claim 8. *S is a vertex cover.*

If not, then there exists an edge e that does not touch any edge in M. That means $M \cup \{e\}$ is also a matching, which contradicts the maximality of M. Hence, S must be a vertex cover.

Claim 9. *$|M| \leq |VC|$, and therefore $|S| \leq 2|VC|$.*

Proof. For every edge $e \in M$, at least one of its endpoints must belong to the vertex cover VC. □

3 Estimating Maximum Independent Set Size

We now consider estimating the size of the maximum independent set (IS) in a graph G of maximum degree d.

We will construct a local oracle to approximate the size of IS.

$$G \xrightarrow{\text{Local Oracle}} \text{LO}(v) = \begin{cases} 1 & \text{if } v \in I(R, G) \\ 0 & \text{otherwise} \end{cases}$$

where $I(R, G)$ is a maximal independent set determined by randomness R.

For all v, $\text{LO}(v)$ has expected runtime $O(e^d \cdot d)$ and returns whether $v \in I$.

It suffices to estimate $|I|$ by random sampling.

In $O(1/\varepsilon^2)$ time, we can obtain a $(\pm\varepsilon n)$ additive approximation.

$$|IS| \geq \frac{n}{2d}.$$

Idealized Algorithm for MaxIS

Enumerate vertices v in some order:

1. For each v: if no neighbor of v is yet in I, add v to I.
2. Otherwise, skip v.

To break long dependency chains, randomize the order

Randomized Ordering in the Independent Set Algorithm

Algorithm:

1. Set the order to be random.
2. For each vertex v, draw a random value $r_v \in [0, 1]$.

3. The processing order of vertices is determined by the increasing order of r_v values.

Example:

$$v\ (r_v = 0.5) \longrightarrow \begin{cases} 0.1 & \text{added to } I, \\ 0.7 & \text{skipped}, \\ 0.9 & \text{skipped}. \end{cases}$$

In other words, vertices with smaller random priorities are processed first, reducing long dependency chains.