CS 7880: Algorithms for Big Data (Fall'22) **Northeastern University**

Lecture 15

November 1, 2020

Instructor: Soheil Behnezhad *Scribe: Ethan Mook*

Disclaimer: *These notes have not been edited by the instructor.*

In this lecture we will change gears a bit from the previous lectures and introduce a new format for designing algorithms for big data called *massively parallel computation.* This is a model of computation that seeks to provide a clean abstraction for studying algorithms designed for the immensely popular practical frameworks of distributed computation like MapReduce, Hadoop, Spark, etc.

1 Massively Parallel Computation

Massively parallel computation is a model of computation where a large input of size N is distributed arbitrarily across M machines that each have limited space S. These parameters are not fixed by the model and will vary from problem to problem, but we typically consider S to be much smaller than N, say $S = O(N^\delta)$ for some $0 < \delta < 1$, and we assume that there are enough machines for them to collectively hold the entire input, so $M = \Omega(N/S) = \Omega(N^{1-\delta})$.

Computation proceeds in synchronous rounds, in between which the machines are able to communicate with each other. Our main goal in designing MPC algorithms will be to minimize the number of rounds it takes. Each round consists of the following two steps:

1. Each machine can perform any arbitrary computation on the data it holds locally.
2. Each machine can send messages to other machines so long as the total messages sent and received by each machine is at most S. All messages are delivered *simultaneously* at the end of this round.

This sequence of steps repeats for each round until the machines are ready to output their computation. If the output of a problem is small enough to fit on one machine we usually consider sending the entire output to, say, the first machine. However if the output is large, we may consider spreading the output across many machines or having machines each be responsible for some small part of the output. For instance, if the problem is to compute a coloring of a graph, each machine may be responsible for holding the coloring of its edges.

1.1 MPC Parameters for Graph Problems

In the setting of graph problems, a graph with n vertices has up to $m = \binom{n}{2}$ edges edges and hence the total input size is $N = O(n^2)$. Therefore the first non-trivial setting of parameters for MPC algorithms for graphs would be to set $S = n^{2-\Omega(1)}$. However because the input size of a graph problem varies with the density of the graph, we often consider three distinct regimes of parameters in the MPC model:

- **Super-linear regime:** $S = O(n^{1+\epsilon})$ where $0 < \epsilon < 1$ is constant.
- **Near-linear regime:** $S = \tilde{O}(n)$.
- **Sub-linear regime:** $S = O(n^\epsilon)$ where $0 < \epsilon < 1$ is constant.

In all regimes we generally set $M = N/S$. In this lecture we will only work in the super-linear regime.

1.2 Maximal Matching in the Super-linear MPC Model

Recall that in the maximal matching problem, the goal is to output a matching in the graph that is maximal in the sense that there are no edges that can be added to it.

Theorem 1. *There is a randomized MPC algorithm that finds a maximal matching with $S = n^{1+\epsilon}$ space and $M = m/S$ machines in $O(1/\epsilon)$ rounds with high probability.*

Proof. We will follow a technique called *filtering.* Define the following algorithm.

`MPC-Matching`:

1. Randomly sample $n^{1+\epsilon}$ edges E' and send them to machine 1.
2. Machine 1 finds a maximal matching M of E' locally and adds M to the final output.
3. Remove the matched vertices in M from the graph and recurse on the remaining edges until the graph is empty.

Observe that correctness of this algorithm follows from the fact that we repeat until the graph is empty. This implies that there are no edges in the original graph with two unmatched endpoints, hence the final matching is maximal. Next to show the claimed efficiency we argue the following two claims.

Claim 2. *Each iteration of the algorithm can be implemented in $O(1)$ rounds of MPC.*

Proof sketch. The act of sampling $n^{1+\epsilon}$ edges can be done by having each machine choose each of its edges with probability $n^{1+\epsilon}/m$. By the Chernoff bound this produces a set E' of size close to $n^{1+\epsilon}$ with high probability. ☐

Claim 3. *After the ith recursive iteration, the remaining graph has at most $O(n^{2-i\epsilon})$ edges with high probability.*

Proof. Suppose that we have m edges after iteration $(i-1)$. We prove that after one more iteration, the number of edges drops to m/n^ϵ with high probability.

Observe that every edge in E' is sampled with probability $n^{1+\epsilon}/m$. Suppose that there more than m/n^ϵ edges that remain after finding some maximal matching of the sampled edges and let S^* denote the set of remaining vertices. Notice that any edge $e \in G[S^*]$ must not have been sampled because otherwise, e could be added to the matching found by machine 1, contradicting its maximality.

Call a subset of vertices $S \subseteq V$ "critical" if the induced graph $G[S]$ has at least m/n^ϵ edges. Because a critical set S induces a graph with so many edges, the probability that none of its edges were sampled is very low:

$$\begin{aligned}\Pr[\text{no edge in } G[S] \text{ is sampled}] &= \left(1 - \frac{n^{1+\epsilon}}{m}\right)^{\frac{m}{n^\epsilon}} \\ &= \left(1 - \frac{n^{1+\epsilon}}{m}\right)^{\frac{m \cdot n}{n^{1+\epsilon}}} \\ &\leq e^{-n},\end{aligned}$$

where here we used the inequality that $(1+1/x)^x \leq 1/e$ for all x. Then, by a union bound over the at most 2^n critical subsets S, we get that with high probability, for all critical subsets S at least one edge in $G[S]$ is sampled. Therefore there are at least m/n^ϵ edges left unmatched. ☐

Combining the previous two claims immediately completes the proof because they show that $O(1/\epsilon)$ rounds are sufficient for the algorithm to terminate with high probability. ☐

1.3 MST in the Super-linear MPC Model

Theorem 4. *There is a randomized MPC algorithm that finds the MST of a graph with $S = n^{1+\epsilon}$ space and $M = m/S$ machines in $O(1/\epsilon)$ rounds with high probability.*

Proof. Define the algorithm as follows:

```
MPC-MST:
  1. If $|E| \leq n^{1+\epsilon}$, simply compute $T = \texttt{MST}(E)$ locally on one machine and return it.
  2. Divide the edge set $E$ arbitrarily into $E_1, \ldots, E_\ell$ such that $|E_i| = n^{1+\epsilon}$ and send $E_i$ to machine $i$.
  3. Each machine locally computes $T_i = \texttt{MST}(E_i)$.
  4. Recursively compute $\texttt{MPC-MST}(T_1 \cup \cdots \cup T_\ell)$ and return it.
```

We first argue the running time of the algorithm. Suppose $|E| = n^{1+c}$ for some c, then $\ell = \frac{n^{1+c}}{n^{1+\epsilon}} = n^{c-\epsilon}$. Observe that each of the local trees T_i has at most $n-1$ edges because it's a tree, so the total number of edges after one iteration is $\ell \cdot (n-1) = O(n^{1+c-\epsilon})$. From this we can see that the algorithm always terminates within $O(1/\epsilon)$ rounds. To prove correctness, we first prove the following claim.

Claim 5. *For any graph $G = (V, E)$, let $E' \subseteq E$. Then any edge $e \in E'$ that is not in $T' = \texttt{MST}(E')$ is also not in $T = \texttt{MST}(E)$ either.*

Proof. Let $e = (u, v) \in E' \setminus T'$. Since T' is a minimum spanning tree, that means there is some path in T' that connects u and v such that each edge along the path has weight less than that of e. Moreover, for any edge $e' = (u', v')$ in that path that is not in T, there must be a path in T that connects u' and v' all of whose edges have lower weight than e' and thus by extension they have lower weight than e as well. Thus we get there is a path from u to v in T entirely consisting of edges with lower weight than e, so $e \notin T$. □

Now correctness of the algorithm follows immediately from the claim by noticing that the only edges that are ever pruned along the computation are those that aren't in the MST of a subset of the edges. Thus the final tree output is the MST of the graph.

□