

Chapter 10

Massively Parallel

Computations

Advanced Algorithms

SS 2019

Fabian Kuhn

Massively Parallel Computations

Challenges

- Moore's law does not hold for ever
- We can only increase computational power by increasing the parallelism
- We need algorithmic techniques to deal with immense amounts of data

Massively Parallel Graph Computations

- Many important applications require solving standard graph problems in very large graphs (e.g., search engines, shortest path computations, etc.)
- We need ways to perform graph computations in highly parallel settings:
 - Graph data is shared among many servers / machines
 - No machine can only store a small part of the graph
 - Need techniques to split and parallelize computations among machines
 - Use communication to coordinate between the machines
- Related to (standard) distributed graph computations

MPC Model

- An abstract formal model to study large-scale parallel computations
 - Aims to study parallelism at a more coarse-grained level than classic fine-grained parallel models like PRAM
(models settings where communication is much more expensive than computation)

Formal Model

- Input of size N words (1 word = $O(\log N)$ bits, for graphs, $N = O(|E|)$)
- There are $M \ll N$ machines
- Each machine ~~has~~^{has} a memory of S words, i.e., we need $S \geq N/M$
 - We typically assume that $S = N^c$ for a constant $c < 1$
- Time progresses in synchronous rounds, in each round, every machine can send & receive S words to & from other machines
- Initially, the data is partitioned in an arbitrary way among the M machines
 - Such that every machine has a roughly equal part of the data
 - W.l.o.g., data is partitioned in a random way among the machines

MPC Model for Graph Computations

Assumption: Input is a graph $G = (V, E)$

- Number of nodes $n = |V|$, number of edges $m = |E|$, nodes have IDs
- Input can be specified by the set E of edges
 - each edge might have some other information, e.g., a weight
 - for simplicity, assume that every node has degree ≥ 1 ←
- Initially, each edge is given to a uniformly random machine
- We typically assume that $S = \tilde{O}(N/M) = \tilde{O}(m/M)$

$N=m$

Strongly superlinear memory regime

$$S = n^{1+\varepsilon} \text{ for a constant } \varepsilon > 0$$

Strongly sublinear memory regime

$$S = n^\alpha \text{ for a constant } 0 < \alpha < 1$$

Near-linear memory regime

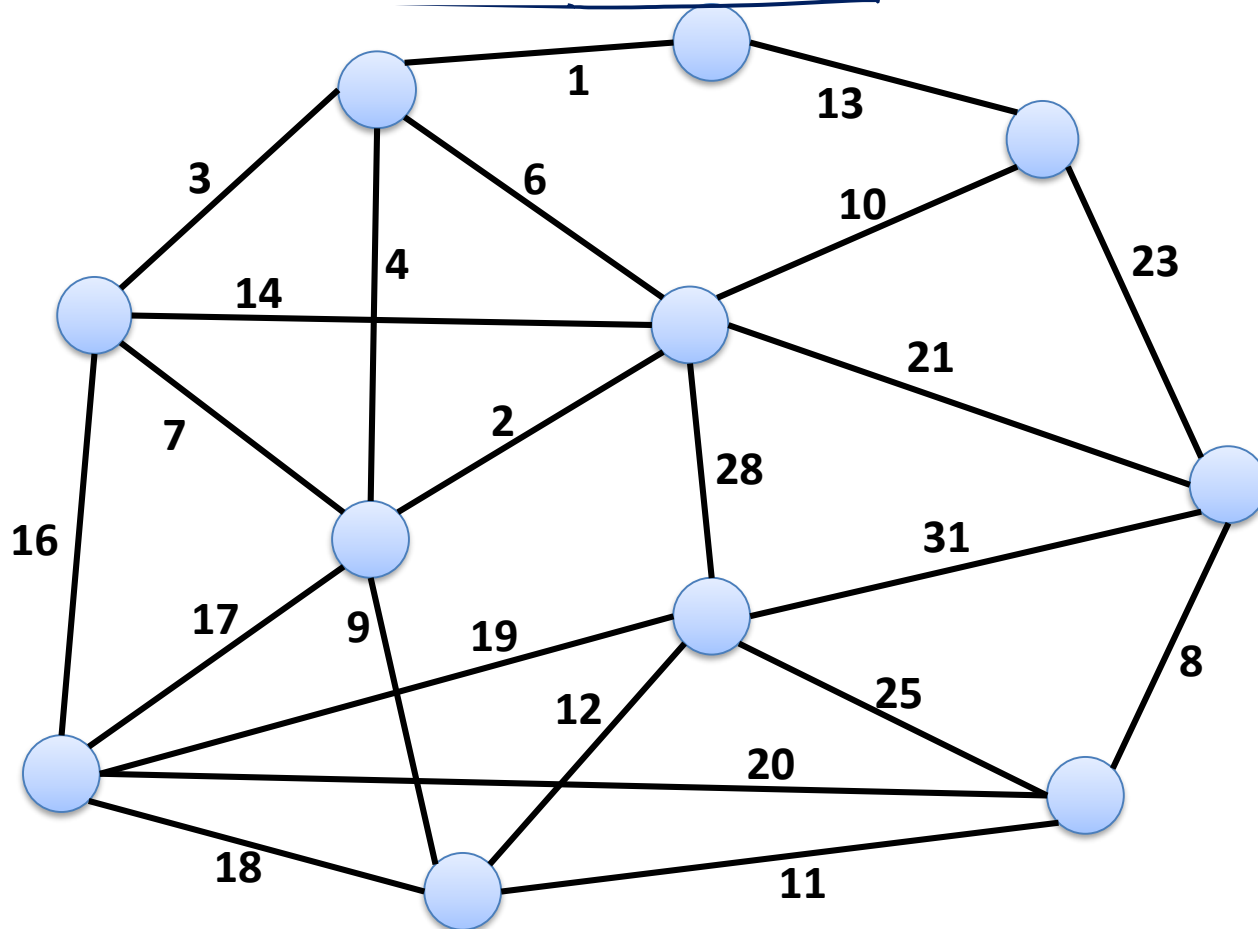
$$S = \tilde{O}(n)$$

Minimum Spanning Tree (MST) Problem

Given: connected graph $G = (V, E)$ with edge weights w_e

Goal: find a spanning tree $T = (V, E_T)$ of minimum total weight

- For simplicity, assume that the edge weights w_e are unique (makes MST unique)

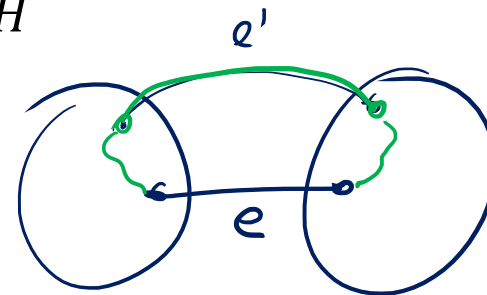


Properties of the MST

Minimum Spanning Forest (MSF) of G :

- A forest consisting of the MST of each of the connected components of G
 - Maximal forest of minimum total weight

Claim: Let $G = (V, E, w)$ be a weighted graph and let $H = (V', E', w)$ be a subgraph of G . If $e \in E'$ is an edge of the MST (or MSF) of G , then e is also an edge of the minimum spanning forest (MSF) of H



MST With Strongly Superlinear Memory

Initially:

- Each machine has $O(n^{1+\varepsilon})$ edges
 - There are $M = O(m/n^{1+\varepsilon})$ machines
- Let H_M be the subgraph induced by the edges of machine M

MPC Algorithm:

1. Each machine M computes minimum spanning forest F_M of H_M
 2. Discard all edges that are not part of some MSF F_M
 3. Remaining number of edges:
$$m' \leq M \cdot n = O(m/n^\varepsilon)$$
 4. Redistribute remaining edges to $M' = O(m'/n^{1+\varepsilon})$ machines
 - Randomly reassign each edge
- Algorithm reduces number of edges by factor $\Theta(n^\varepsilon)$ in 1 rounds.
 - $O(1/\varepsilon)$ repetitions suffice to solve the problem

Borůvka's MST Algorithm

MST Fragment:

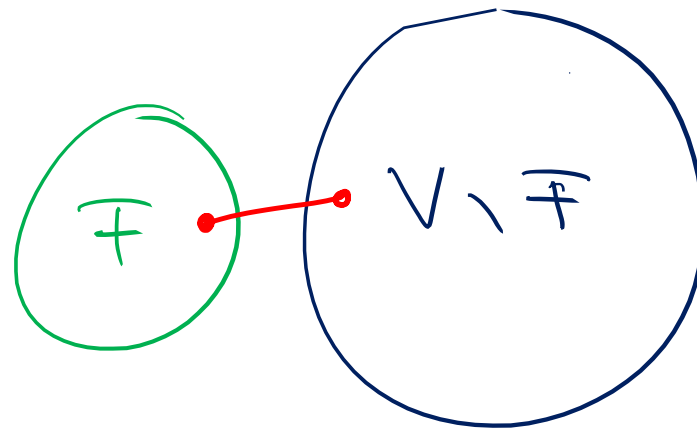
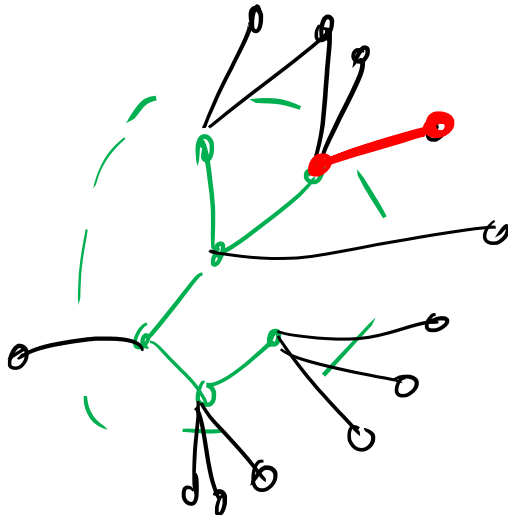
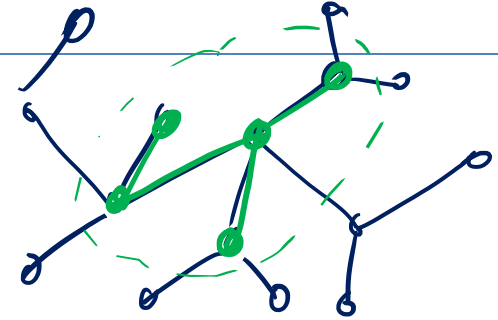
- A connected subtree $F = (V_F, E_F)$ of the MST

Minimum edge of MST fragment $F = (V_F, E_F)$:

- Minimum weight edge connecting a node in V_F with a node in $V \setminus V_F$

minimum

Lemma: For every MST fragment F , the ~~blue~~ edge of F is in the MST



Borůvka's MST Algorithm

Algorithm description:

- Develops the MST in parallel phases
- Initially, each node is an MST fragment of size 1 (and with no edges)
- In each phase:
add the minimum edge of each fragment to the MST
- Terminate when there is only one fragment
 - or when there are no edges between different fragments

Theorem: The above alg. computes the MST in $O(\log n)$ phases.

min. fragment size doubles in each phase

fragments decr. by factor ≥ 2

MST With Strongly Sublinear Memory: Ideas

$\alpha < 1$

Assume: $G = (V, E)$ with n nodes, m edges, memory $S = n^\alpha$ for const. $\alpha > 0$

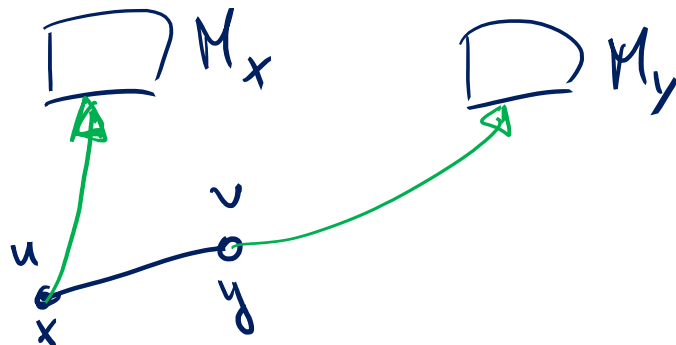
- Also assume that we have $M \geq m/S \cdot c \log n$ machines for suff. large $c \geq 1$

Representation of algorithm state:

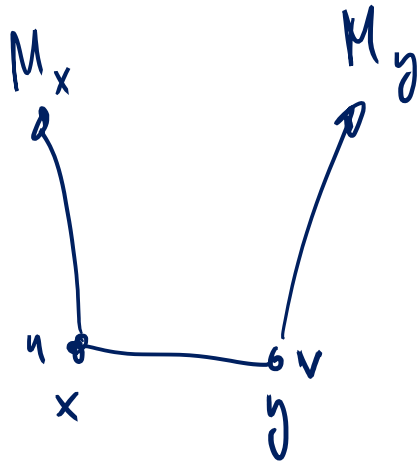
- Each fragment has a unique ID, fragment ID of node u : $\text{FID}(u)$
- The machine storing an edge $\{u, v\}$ knows the fragment IDs of u and v

Goal: implement one phase in time $O(1)$:

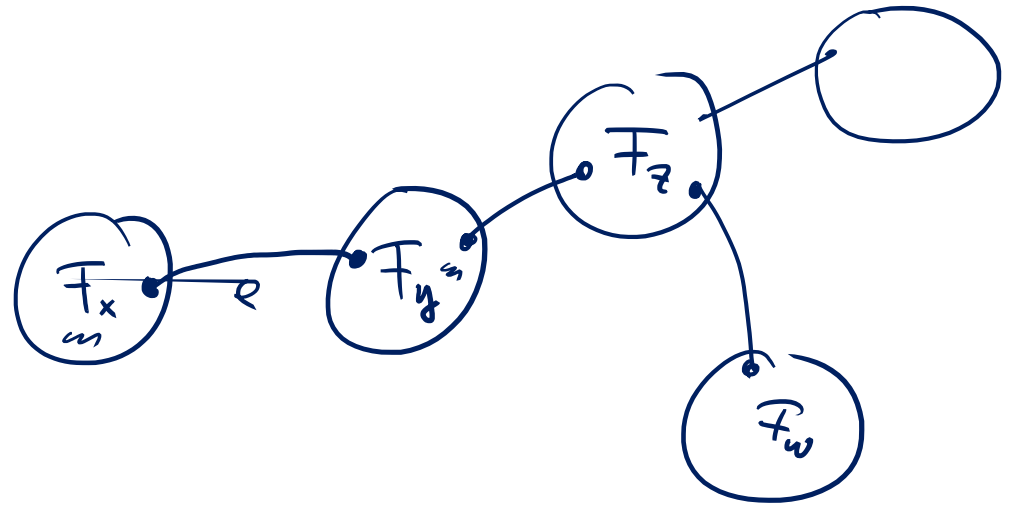
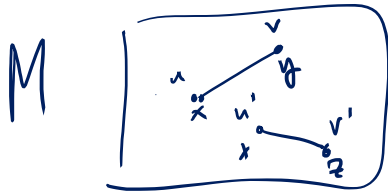
- Assume that for each fragment ID x , there is some responsible machine M_x
 - Additional empty machines that are randomly assigned (e.g. by a hash function)
- For now, assume that each node u directly interacts with machine $M_{\text{FID}(u)}$



Implementing One Phase (First Attempt)

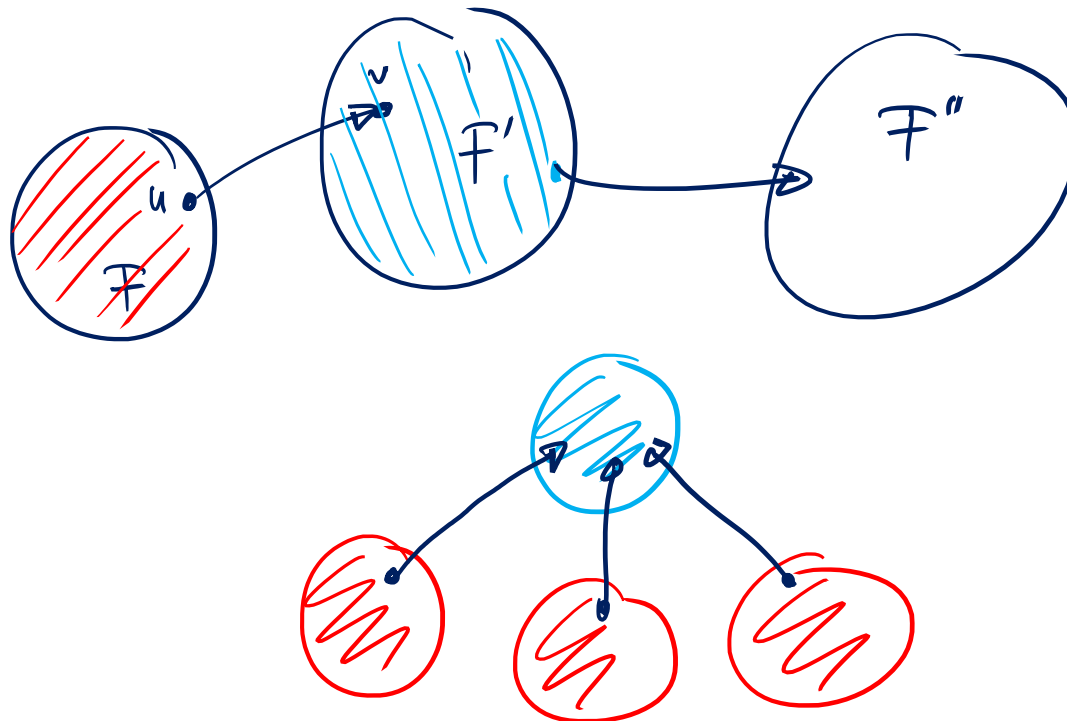


M_x can compute min edge of fragment x

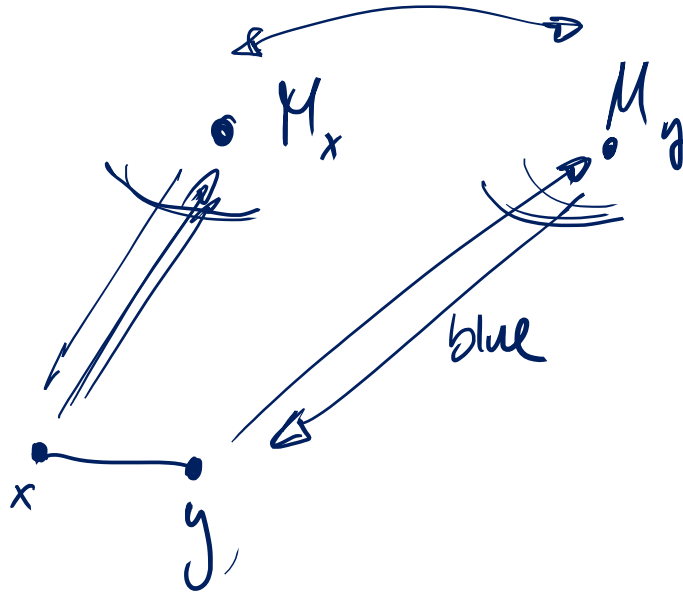


Small Change to the Basic Algorithm

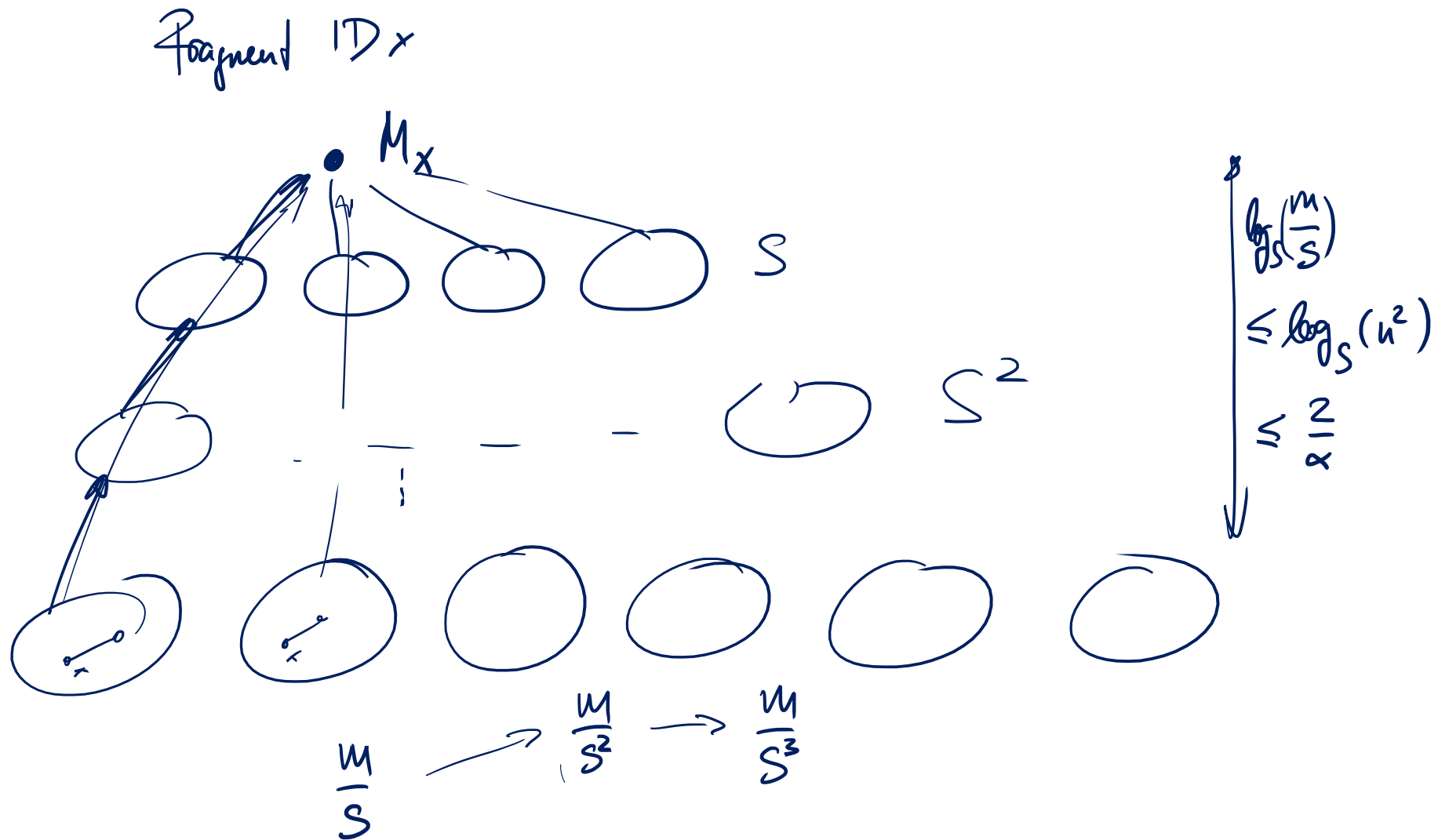
- In each phase, each fragment initially picks a random color in {red, blue}
- Let $\{u, v\}$ be the minimum edge of a fragment F
- Only add $\{u, v\}$ to MST in current phase if F is a red fragment and $\{u, v\}$ connects to a blue fragment.



Implementing One Phase (Second Attempt)



Implementation with Aggregation Trees



MST with Strongly Sublinear Memory

Theorem: In the strongly sublinear memory regime (i.e., when $S = n^\alpha$ for a constant $\alpha \in (0,1)$), an MST can be computed in time $O(\log n)$.

MST in the Near-Linear Memory Regime

- Assume that $S = \underline{n \cdot (\log n)^c}$ for a sufficiently large constant $c > 0$.
- Instead of MST, we consider a simpler, closely related problem

Connectivity / Component Identification

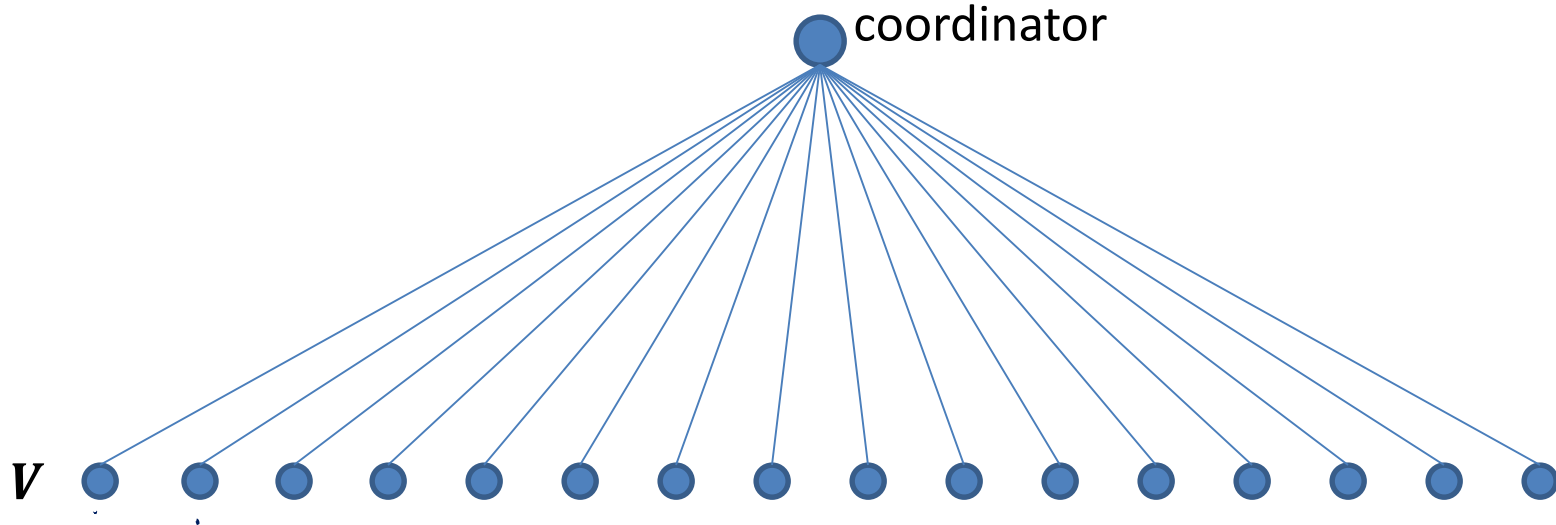
- At the end, algorithm needs to output a number $C(u)$ for each node $u \in V$ such that $C(u) = C(v)$ iff u and v are in the same connected component of G .

Observations

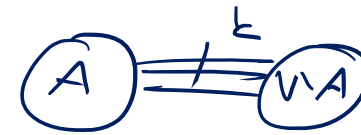
- Algorithm in particular allows to compute whether G is connected
- The MST algorithm from before can be used to solve component identification
 - The algorithm terminates when there are no more edges connecting different fragments. The fragment IDs at the end can be used as outputs
- In combination with some binary search over the edge weights, component identification can be used to also compute an MST
 - Everything we will do can be extended to the MST problem
(at the cost of maybe a couple of log-factors in the required memory per machine)

The Single-Round Coordinator Model

- We will study the problem in a different communication model



- There is a coordinator and one node for each $v \in V$
- Node v initially knows the set of its neighbors (i.e., all incident edges)
- Each node $v \in V$ is allowed to send one message to the coordinator
- Afterwards the coordinator needs to be able to compute the output
- We will assume that the nodes have access to shared randomness
- We will use the **graph sketching** technique



Single Cut Problem:

- Fix $A \subseteq V$. Assume that there are $k \geq 1$ edges across the cut $(A, V \setminus A)$.
- Goal:** Coordinator needs to return one of the k edges across the cut

Assume first that $k = 1$:

- Define a unique ID for each edge $e = \{u, v\} \in E$: $\text{ID}(e) = \text{ID}(u) \circ \text{ID}(v)$
- Each node $u \in A$ computes XOR_u as

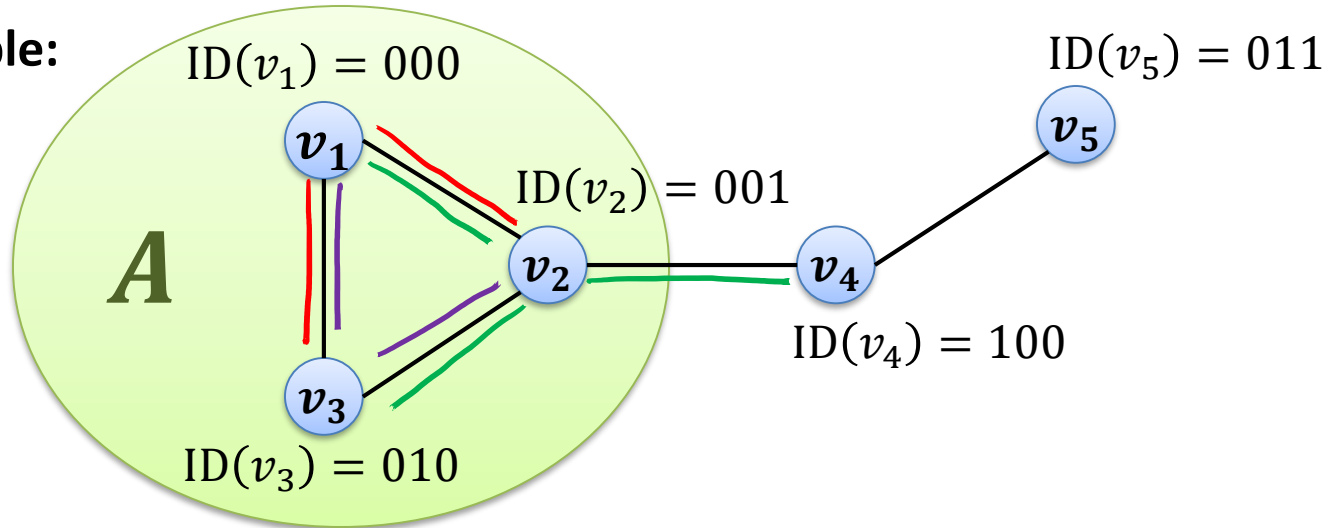
$$\text{XOR}_u := \bigoplus_{e \in E: u \in e} \text{ID}(e)$$

- Each node $u \in V$ sends XOR_u to coordinator
- Coordinator computes

$$\underline{\text{XOR}_A} := \bigoplus_{u \in A} \text{XOR}_u$$

Graph Sketching: Warm Up 1

Example:



XOR_{v_1}

XOR_{v_2}

XOR_{v_3}

Graph Sketching: Warm Up 2

Assume that k is an arbitrary value

- Let E_A be the set of edges across the cut $(A, V \setminus A)$ ($|E_A| = k$)

Claim: If we use the same algorithm, $\text{XOR}_A = \bigoplus_{e \in E_A} \text{ID}(e)$.

Assume that we are given an estimate $\hat{k}$ s.t. $\frac{\hat{k}}{2} \leq k \leq \hat{k}$:

- Sample each edge with probability $1/\hat{k}$ and apply alg. with sampled edges

Graph Sketching: Warm Up 2

Assume that $k > 1$ and an estimate $\hat{k}$ s.t. $\frac{\hat{k}}{2} \leq k \leq \hat{k}$ is given

- Sample each edge with probability $1/\hat{k}$
- Let E'_A be the sampled edges of E_A (across the cut)

Claim: $\mathbb{P}(|E'_A| = 1) \geq 1/10$.

$$\mathbb{P}(|E'_A| = 1) = k \cdot \frac{1}{\hat{k}} \cdot \left(1 - \frac{1}{\hat{k}}\right)^{k-1}$$

$$\geq \frac{\hat{k}}{2} \cdot \frac{1}{\hat{k}} \cdot \left(1 - \frac{1}{\hat{k}}\right)^{\hat{k}}$$

$$\geq \frac{1}{2} \cdot 4^{-\frac{1}{\hat{k}} \cdot \hat{k}}$$

$$\geq \frac{1}{10}$$

$$1 - x \geq 4^{-x} \quad (|x| \leq \frac{1}{2})$$

Graph Sketching: Warm Up 2

Discussion:

- How can we sample each edge with probability $1/\hat{k}$?
 - Use shared randomness
- If we use the same algorithm, XOR_A is equal to an edge of E_A if $|E'_A| = 1$

How can we distinguish $|E'_A| = 1$ from $|E'_A| \neq 1$?

- We need to make sure that
 - a) The bit-wise XOR of 0 or > 1 edge IDs is not equal to an edge ID
 - b) Edge IDs can be distinguished from the XORs of 0 or > 1 edge IDs

Random Edge IDs

Edge ID of edge $e = \{u, v\} \in E$ (assume $ID(u) < ID(v)$)

$$ID(e) = ID(u) \circ ID(v) \circ \underline{R_e}$$

- R_e is a random bit string of length $80 \ln n$ where each bit is 1 with prob. $1/8$
- Let R'_A be the bitwise XOR of R_e for $e \in E'_A$

Claim: Let X be the number of 1s in R'_A . If $|E'_A| = 0$, then $X = 0$, otherwise

- If $|E'_A| = 1$, then $1 < X < \underline{14 \ln n}$ with high probability
- If $|E'_A| > 1$, then $X > 14 \ln n$ with high probability

Proof Sketch:

Random Edge IDs

Claim: Let X be the number of 1s in R'_A . If $|E'_A| = 0$, then $X = 0$, otherwise

- If $|E'_A| = 1$, then $1 < X < 14 \ln n$ with high probability
- If $|E'_A| > 1$, then $X > 14 \ln n$ with high probability

Proof Sketch:

- If $|E'_A| \geq 2$, each of the $80 \ln n$ bits of R'_A is 1 with prob. $\geq 2 \cdot \frac{1}{8} \cdot \frac{7}{8} > \frac{1}{5}$

One phase of the Borůvka algorithm

- We need to find one outgoing edge for each fragment
 - Then the coordinator can add a subset of these edges and reduce the number of fragments by a factor 2
- We do not know the number of out-going edges of the different fragments
 - And different fragments might have different numbers
- Use different sampling probabilities: $\frac{1}{n}, \frac{2}{n}, \frac{4}{n}, \dots, \frac{1}{2}$ and send sketches for all probabilities to coordinator
 - For each instance, each $v \in V$ sends XOR of sampled edges to coordinator
- For each fragment, one of the probabilities succeeds with probability $\geq 1/10$
- When having $\Theta(\log n)$ instances for each of the probabilities, we get an outgoing edge for each fragment with high probability
- Each node can send $O(\log^3 n)$ bits to coordinator for one phase

Observation: The protocol does not depend on the fragments

- We can therefore send the information for all phases in parallel

Theorem: In the coordinator model, there is a protocol where every node $v \in V$ send $O(\log^4 n)$ bits to the coordinator s.t. the coordinator can solve the connectivity & connected components problem.

Remarks:

- The number of bits can be reduced to $O(\log^3 n)$
 - It is sufficient to succeed with constant prob. for each fragment in each phase
- $\Omega(\log^3 n)$ bits are necessary [Nelson, Yu; 2019]
- Graph sketching has been introduced by [Ahn, Guha, McGregor; 2012]

$$S = n \cdot \text{poly}(\log(n))$$

1. For every node $v \in V$, create a responsible machine M_v
 - Send each edge $\{u, v\}$ to both M_u and M_v
 - Make sure that each machine gets $\tilde{O}(n)$ edges

1. The randomness for each edge can be generated initially by the machine that holds the edge
 - Also send the randomness for the edge $\{u, v\}$ to M_u and M_v

2. Use one additional machine for the coordinator

Theorem: In the MPC model with $S = \tilde{O}(n)$, the connectivity & connected components problem can be solve in $O(1)$ rounds.

- Graph sketching can help in many different contexts, e.g.,
 - also in the strongly-sublinear memory regime to save communication
 - in the streaming model
 - in the standard distributed model to save message
- In the strongly sublinear memory regime, it is not known whether it is possible to be faster than $O(\log n)$ rounds
 - It is widely believed that there should be an $\Omega(\log n)$ lower bound
 - Even the following simple version of the problem seems to require $\Omega(\log n)$ time to distinguish 2 cycles of length $n/2$ from one cycle of length n