COMS E6998-9: Algorithms for Massive Data (Fall'25)	Oct 22, 2025

Lecture 15: Graph Problems in MPC

Instructor: *Alex Andoni* — Scribe: *Patrick Lai*

1 Review

We are given a graph G with n nodes and m edges. N is the input size, which we think of as equal to m since the number of edges is typically larger than the number of nodes. S is the amount of space on each machine ($M = O(N/S)$).

We say that we are in the "dense regime" if $S \geq n^{1+\epsilon}$. We saw in the last lecture that for many problems in this regime, we can perform repeated compression to get $R = O(1/\epsilon)$.

We are in the "sparse regime" if $S << n,\ S = n^\delta$.

2 Problem of Connectivity

Ideally, we can come up with an algorithm that runs in $R = O(log_S N) = O(1/\delta)$ (or at least only dependent on $1/\delta$) rounds. An example of a hard problem where we don't yet have an algorithm that runs in the "ideal" amount of time: distinguishing between 1 large cycle of size n (connected graph) vs. 2 cycles of size $n/2$ (not connected). We will refer to this as the "1 vs 2 - cycle problem". The namesake conjecture states that you need $\Omega(\log N)$ rounds for this (when say $S = \sqrt{n}$).

2.1 Algorithm 1

This algorithm will run in $R = O(D)$, where D is the diameter of G. Note that this runtime will be bad for the hard problem above since a large cycle of size n has diameter $n/2$. The main idea of this algorithm is to run breadth-first search.

Setup

a. Array mark[i] = True if BFS has visited node i

b. We assign node $i \in [n]$ to the fixed machine $\lceil i/S \rceil$

c. $\forall i \in [n]$: we store the list of incident edges L_i such that the edges in Li are stored on consecutive machines. Additionally, on the machine $\lceil i/S \rceil$, we store: $start(i)$, the first machine where the edges in L_i are stored, and $end(i)$, the last machine where the edges in L_i are stored.

Pre-processing

P1. Duplicate all edges: $(i, j) \to (i, j), (j, i)$

P2. Sort edges lexicographically across machines (this gives us the layout described in (c) above)

P3. For each node i, send $start(i)$, $end(i)$ to the machine responsible for i

Algorithm

1. Set mark[s] = 1 and 0 otherwise, where s is the start node (e.g. $s = 1$)
2. While we can (for $\leq D$ iterations):
   - For each node $i \in [n]$: if mark[i] = 1, then send message "push i" to machines between $start(i)$ and $end(i)$, inclusive.
     *Typically, output size is $O(S)$ (when $|end(i) - start(i)| \leq 1$). In the worst-case, we can have $O(M + S)$ output size (there are 2 types of nodes: 1) nodes whose edges span > 1 machine; we need to send O(M) messages for these nodes. 2) nodes whose edges are on 1 machine; we need to send $O(S)$ messages for these nodes. This gives us total output size of $O(M + S)$). Furthermore, each machine receives $\leq n/M = O(S)$ messages. In the worst-case, we can distribute these messages using an s-ary tree in $O(log_S n)$ rounds.
   - For every "push i" message received by a machine:

     For every edge (i, j) that the machine contains: send message "mark j" to the machine responsible for node j

     *To make sure the machine responsible for j can handle the inbound messages, we have a couple of options:
     - Propagate messages with an s-ary tree: collect messages from a subset of machines, remove duplicates, and repeat. This takes $O(log_S n)$ rounds
     - For $O(log_S n)$ iterations: sort by j, then remove duplicates locally
   - For each "mark j" message received: set mark[j] = 1
   - Finish when none of the values of mark[j] change in the previous step

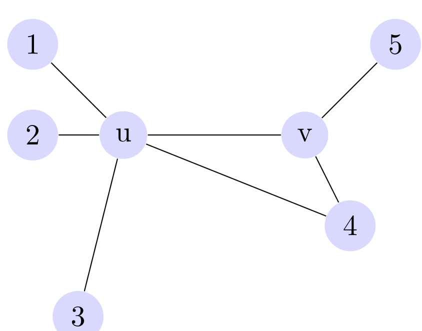
This algorithm runs in $R = O(log_S n * D) = O(D/\delta)$ rounds.

2.2 Algorithm 2

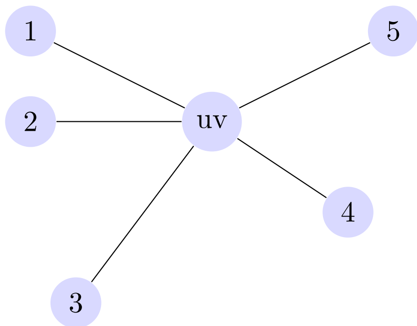
This algorithm will run in $R = O(logn)$ rounds.

Conjecture 1 (1 vs 2 - Cycle Conjecture)**.** *We need $R = \Omega(logn)$ parallel-time to solve the 1 vs 2 - cycle problem, even for $S = \sqrt{n}$*

Idea Pick $\Omega(n)$ edges on $\Omega(n)$ nodes and "contract" them. For example, given:



We can contract the edge (u, v) to get:



The number of nodes will decrease by a constant factor after each round, so after $logn$ rounds the number of nodes will decrease from n to a constant number.

Algorithm
For $T = O(logn)$ iterations:

1. For each vertex, with probability 1/2, we call the vertex a "leader"
2. For each non-leader j, find the incident leader with the smallest index. If such an incident leader l exists, contract j into l. Otherwise, do nothing to j.

Time Analysis
For any node of degree ≥ 1: the probability of being contracted is at least $1/2 \cdot 1/2$, since the node has 1/2 probability of being a non-leader, and each adjacent node has 1/2 probability of being a leader. So, assuming the graph is connected: $E[\# \textit{ contracted nodes}] \geq n \cdot 1/2 \cdot 1/2 = n/4$.

After t rounds: $E[\# \textit{ nodes remaining}] \leq n(1 - 1/4)^t$. So, after $t = O(logn)$ rounds, the expected number of nodes remaining is 1.

Notes on Implementation
For every vertex i that is chosen as leader, we can send a message to the machine handling each incident vertex j to let the machine know that i is an incident leader for j. To make sure we satisfy communication bandwidth constraints, we can propagate these messages in a similar manner as described in the previous algorithm.

Theorem 2. *There is an algorithm that solves this problem in* $R = O(logD + loglog_{m/n}n)$ *rounds.*