COMS E6998-9: Algorithms for Massive Data (Spring'19) Apr 16, 2019

Lecture 23: MPC model: Connectivity

Instructor: *Alex Andoni* Scribes: *Xinyuan Cao, Yanda Chen*

1 Introduction

Today we will continue the topic of Massively Parallel Computing (MPC) model. Suppose we have $O(m) = O(N/S)$ machines, with each machine $O(S)$ space. Here N is the size of input. Suppose $S = N^\epsilon$. (e.g. $S = \sqrt{N}$). We're going to discuss the **graph connectivity** problem.
Input: an undirected graph G on n vertices, N edges.
Output: (i, C_i), $i \in [n]$, where C_i is the name of the component of vertex i.

To check connectivity, a basis idea is through **BFS**: Start with vertex v, push a "wave". Over and over and see whether the entire wave could cover the whole graph. Since the name of the node does not tell us anything, maybe we can only know edges around each node because of the limited space of each machine. Therefore, sometimes it may be hard to distinguish whether a node is too far away but connected, or disconnected from the graph. So BFS is suitable for this setting. In one round of the computation, we can do one step on the wave. BFS runs in $O(D)$ time, where D is the diameter of graph. So the number of rounds is $O(D)$. Next we'll consider the connectivity problems in two regimes: dense regime and sparse regime.

2 Dense Regime

Definition 1. *Dense regime refers to the case where $S >> n$ (i.e. space on each machine is much larger than the number of vertices in the graph).*

Here we assume the space per machine $S \geq n^{1+\epsilon}$, $\epsilon > 0$. This also implies that $N \geq n^{1+\epsilon}$. Assume that $N = n^{1+\delta}$, $\delta > 0$. We can use "filtering", that is to filter inputs until they fit on one machine. Since we only care about connectivity, we can filter edges so as to just keep a spanning forest finally.

Theorem 2. *We can solve graph connectivity in $O(\delta/\epsilon)$ time in dense regime.*

Algorithm: Do for $O(\delta/\epsilon)$ rounds. In round i,

- Each machine computes a spanning forest, throw away the rest of the edges.
- Compute N_i =# remaining edges.
- Reorganize (remaining) edges on $\lceil N_i/S \rceil$ machines.

Finally we have a spanning forest on machine M_1.

Analysis: This is correct for connectivity because we throw out edges only when there is an alternative path. Also we know that finally we can store remaining edges on one machine by Claim 3.

Claim 3. *After $t = O(\delta/\epsilon)$ rounds, $N_t = O(n)$.*

Proof. In each round, each machine starts with $S \geq n^{1+\epsilon}$ edges. Our algorithm computes the spanning forest on this machine. So in the end of the round, it remains with at most $n-1$ edges. So

$$\frac{N_{i+1}}{N_i} \leq \frac{n-1}{n^{1+\epsilon}} < n^{-\epsilon}$$

So in step t,

$$N_t \leq \frac{N_0}{(n^\epsilon)^t} = \frac{N}{n^{\epsilon t}} \leq O(n)$$

□

3 Sparse Regime

Definition 4. *Sparse regime refers to the case where $S << n$ (i.e. space on each machine is much smaller than the number of vertices in the graph).*

Hard Case: To distinguish between the case where the graph is a connected component with N edges and the case where the graph is two connected components each with approximately $N/2$ edges.

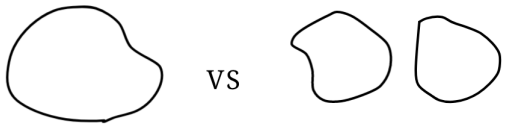

Figure 1: Hard Case Instance

Open Question: Can we do s-t connectivity in MPC with $<< O(\log n)$ if $S = n^{1-\epsilon}$?

Key Tool: BFS runs in O(D) rounds on MPC where D is the diameter of the graph.

Algorithm: Keep a register for each node, with state either unmarked (nodes that are not visited yet), marked (nodes that are at front of the wave) or done (nodes that are completed).

Things We Store:

- Registers stored in distributive fashion (maybe cannot fit onto a single machine): S registers stored on each machine.
- For every vertex, want to store all the edges incident to that vertex consecutively. Denote $E(v) =$ edges adjacent to v, then $|E(v)| = deg(v)$. Want $E(v)$ on a single machine, but this may fail because there may exist vertex with degree larger than size of machine. Instead, we store $E(v)$ on consecutive machines $\lceil E(v)/S \rceil + 1$ machines.

- To know where $E(v)$ for each vertex v is stored, we keep pointers of the start and end of list of incident edges with $start(v)$ and $end(v)$ for each vertex v.

Preprocess Algorithm:
Step 1: Duplicate all edges $(v, u) \to (v, u), (u, v)$
Step 2: Sort all edges
Step 3: Collect $start(v)$ and $end(v)$ information

BFS Algorithm on MPC

- Mark $R(1)$ = marked, $R(\neq 1)$ = unmarked
- While we can (i.e. $\exists$ nodes marked)
  - $\forall$ vertex v, which is marked, generate "push" on its edges, send them to $start(v)$ and $end(v)$
  - every machine M_i if $start(v) = end(v) = M_i$, generate $(u, marked)$, $\forall u \in E(v)$; otherwise communicate to all machines $[start(v), end(v)]$ push request and generate $(u, marked)$, $\forall u \in E(v)$ on these machines.
  - sort $\{(u, marked)\}$ and remove duplicates (to avoid repeatedly send update messages of the same vertex due to the vertex's large degree)
  - send $(u, marked)$ to machine M responsible for u.
  - each machine that receives $(u, marked)$ updates $R(u)$.

4 Randomized Algorithm

Theorem 5. *There is a $O(\log n)$ randomized polynomial time algorithm. (a version of Boruvka's algorithm)*

The idea is to pick $\Omega(n)$ edges and then contract them. The contraction is shown in Figure 2. Then the number of nodes is dropped by a factor $\Omega(1)$ in one iteration. So the number of iterations is $O(\log n)$.

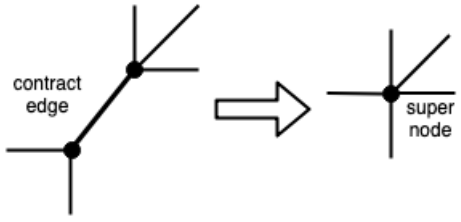

Figure 2: Idea of contracting edges.

Ideally we hope to avoid the situation of "chain" for better collapse. So here $\forall$ vertex, we pick a random incident edge. (The probability of a chain decreases exponentially with length of the chain.)

Figure 3: Situation that the edges we choose construct a chain.

Algorithm: for $t = O(\log n)$ iterations:

- Each vertex is "leader" with probability 1/2.
- Each non-leader chooses the smallest leader.
- Contract all (non-leader $\to$ leader) edges, continue. Here we call the non-leader is "contracted" to the leader.

Analysis: $t = O(\log n)$ iterations is enough to contract a connected component into one vertex with high probability.

Proof. The probability that one node is "contracted" $\geq 1/4$ (probability 1/2 for itself to be a non-leader and probability 1/2 for the vertex it's connected to be a leader), so we have $\mathbb{E}[\#\text{contracted vertices}] \geq n/4$. This means that

$$\mathbb{E}[\#\text{remaining vertices after } t \text{ steps}] \leq n(1 - \frac{1}{4})^t = n(\frac{3}{4})^t$$

Therefore $t \leq \log_{\frac{4}{3}} n = O(\log n)$. □

Contraction: everything labeled "non-leader" is relabeled as "leader".

Storage: (1) labels for each vertex v: $start(v)$, $end(v)$, and whether v is active or passive. Here passive means the node has already been contracted. (2) edges.

So through our iteration, $\forall$ non-leader v, the machine stores $E(v)$. $\forall u \in E(v)$, put all $(u, \text{info generated by } v)$ in one machine, and then sort by u. Finally bring information if u is the smallest leader.