Distributed Algorithms
Minimum Spanning Trees

Maruth Goyal

UT Austin

Spring 2021

Table of Contents

Minimum Spanning Trees

- **Motivation:** In a scenario where communication links have different delays, a node broadcasting a message may want to do it in a manner which minimizes total communication time.

Minimum Spanning Trees

- **Motivation:** In a scenario where communication links have different delays, a node broadcasting a message may want to do it in a manner which minimizes total communication time.
- We assume every node knows the weight w_i for each of its neighbors with UID i.

Minimum Spanning Trees

- **Motivation:** In a scenario where communication links have different delays, a node broadcasting a message may want to do it in a manner which minimizes total communication time.
- We assume every node knows the weight w_i for each of its neighbors with UID i.
- We further assume the graph is undirected.

Minimum Spanning Trees

- **Motivation:** In a scenario where communication links have different delays, a node broadcasting a message may want to do it in a manner which minimizes total communication time.
- We assume every node knows the weight w_i for each of its neighbors with UID i.
- We further assume the graph is undirected.
- **Goal:** At the end, each node should know, for each of its edges, whether or not it is in the minimum spanning tree.

Minimum Spanning Trees

- Recall the following theorem:

Theorem

Suppose $\{F_1, \ldots, F_k\}$ *is a minimum spanning forest of a graph* G*, and* e_i *the minimum weight outgoing edge from* F_i*, then there exists a minimum spanning tree* T *containing all* e_i*.*

Idea

Replicate classical MST algorithms: grow components by merging along minimum-weight outgoing edge.

Minimum Spanning Trees

Idea

Replicate classical MST algorithms: grow components by merging along minimum-weight outgoing edge.

Minimum Spanning Trees

Idea

Replicate classical MST algorithms: grow components by merging along minimum-weight outgoing edge.

Issue

Consider triangle graph with unit-weight edges. MST has cycle!

Minimum Spanning Trees

Idea

Replicate classical MST algorithms: grow components by merging along minimum-weight outgoing edge.

Issue

Consider triangle graph with unit-weight edges. MST has cycle!

Theorem

A connected graph with unique edge weights has a unique MST.

Minimum Spanning Trees

Idea

Replicate classical MST algorithms: grow components by merging along minimum-weight outgoing edge.

Issue

Consider triangle graph with unit-weight edges. MST has cycle!

Theorem

A connected graph with unique edge weights has a unique MST.

Idea

New Assumption: graph has unique edge weights

Minimum Spanning Trees

Idea

New Assumption: graph has unique edge weights

Minimum Spanning Trees

Idea

New Assumption: graph has unique edge weights

Corollary

Safe to merge components by picking Minimum-Weight Outgoing Edge (MWOE)

Minimum Spanning Trees

Idea

New Assumption: graph has unique edge weights

Corollary

Safe to merge components by picking Minimum-Weight Outgoing Edge (MWOE)

Algorithm Sketch (synchronized version of [Gallager et al., 1983])

1. Keep track of components

Minimum Spanning Trees

Idea

New Assumption: graph has unique edge weights

Corollary

Safe to merge components by picking Minimum-Weight Outgoing Edge (MWOE)

Algorithm Sketch (synchronized version of [Gallager et al., 1983])

1. Keep track of components
2. Find MWOE for each component

Minimum Spanning Trees

Idea

New Assumption: graph has unique edge weights

Corollary

Safe to merge components by picking Minimum-Weight Outgoing Edge (MWOE)

Algorithm Sketch (synchronized version of [Gallager et al., 1983])

1. Keep track of components
2. Find MWOE for each component
3. Merge components
    1. Elect a new leader

Minimum Spanning Trees

Idea

New Assumption: graph has unique edge weights

Corollary

Safe to merge components by picking Minimum-Weight Outgoing Edge (MWOE)

Algorithm Sketch (synchronized version of [Gallager et al., 1983])

1. Keep track of components
2. Find MWOE for each component
3. Merge components
    1. Elect a new leader
4. Terminate when only one component.

Minimum Spanning Trees

Issue #1

How do we maintain/represent components?

Minimum Spanning Trees

Issue #1

How do we maintain/represent components?

Representing Components

1. Represent each component using a representative member, the leader.
2. Each node in the component stores the UID of the leader, L.

Minimum Spanning Trees

Algorithm Sketch

1. Keep track of components ✓
2. Find MWOE for each component
3. Merge components
   1. Elect a new leader
4. Terminate when only one component.

Minimum Spanning Trees

Issue #2

How do we find the Minimum-Weight Outgoing Edge for a component?

Minimum Spanning Trees

Issue #2

How do we find the Minimum-Weight Outgoing Edge for a component?

Finding MWOEs

- Recall **Convergecast** from BFS lecture. Computing minimums is assosciative, commutative.
- Each node keeps track of which edges lead to a node already in the component.

Minimum Spanning Trees

Issue #2

How do we find the Minimum-Weight Outgoing Edge for a component?

Finding MWOEs

- Recall **Convergecast** from BFS lecture. Computing minimums is assosciative, commutative.
- Each node keeps track of which edges lead to a node already in the component.
- Each node sends a $\textsc{Test}(L)$ message to its neighbors iff it's not known to already be in the component.
- A node responds to a $\textsc{Test}(L)$ message with T iff it is in the same component (checked by comparing L). Else, responds with F.

Minimum Spanning Trees

Issue #2

How do we find the Minimum-Weight Outgoing Edge for a component?

Finding MWOEs

- Recall **Convergecast** from BFS lecture. Computing minimums is assosciative, commutative.
- Each node keeps track of which edges lead to a node already in the component.
- Each node sends a $\textsc{Test}(L)$ message to its neighbors iff it's not known to already be in the component.
- A node responds to a $\textsc{Test}(L)$ message with T iff it is in the same component (checked by comparing L). Else, responds with F.
- MWOE is propagated up to leader by convergecast aggregation.

Minimum Spanning Trees

Algorithm Sketch

1. Keep track of components ✓
2. Find MWOE for each component ✓
3. Merge components
    1. Elect a new leader
4. Terminate when only one component.

Minimum Spanning Trees

Issue #3

How do we merge components?

Issue #3

How do we merge components?

Merging components

- The leader of each component notifies the node on the other end of each MWOE that the corresponding edge is in the tree.
- The end of the MWOE already in the component notes that the corresponding edge is in the tree.

Minimum Spanning Trees

Issue #3

How do we merge components?

Merging components

- The leader of each component notifies the node on the other end of each MWOE that the corresponding edge is in the tree.
- The end of the MWOE already in the component notes that the corresponding edge is in the tree.
- A new leader for the component is elected. The new component is notified by broadcasting along edges known to be in the tree.

Minimum Spanning Trees

Algorithm Sketch

1. Keep track of components ✓
2. Find MWOE for each component ✓
3. Merge components ✓
    1. Elect a new leader
4. Terminate when only one component.

Minimum Spanning Trees

Issue #3a

How do we elect a new leader for the merged component?

Minimum Spanning Trees

Issue $\#3a$

How do we elect a new leader for the merged component?

Theorem

Let G' be the graph where the nodes are the (unmerged) components, and the edges (directed) are the MWOEs. Then, there's a cycle of length 2 in each component of this graph.

Minimum Spanning Trees

Issue $\#3$a

How do we elect a new leader for the merged component?

Theorem

Let G' be the graph where the nodes are the (unmerged) components, and the edges (directed) are the MWOEs. Then, there's a cycle of length 2 in each component of this graph.

Corollary

For any set of components $\{C_i\}_{i\in[k]}$ being merged, there are two components C_m, C_n such that they have the same MWOE.

Minimum Spanning Trees

Issue #3a

How do we elect a new leader for the merged component?

Theorem

Let G' be the graph where the nodes are the (unmerged) components, and the edges (directed) are the MWOEs. Then, there's a cycle of length 2 in each component of this graph.

Corollary

For any set of components $\{C_i\}_{i\in[k]}$ being merged, there are two components C_m, C_n such that they have the same MWOE.

Idea

We let the leader be the endpoint of the aforementioned MWOE with higher UID. This can determined this locally, and then broadcast.

Minimum Spanning Trees

Theorem

Let G' be the graph where the nodes are the (unmerged) components, and the edges (directed) are the MWOEs. Then, there's a cycle of length 2 in each component (under weak-conectedness) of this graph.

Proof.

- Consider any component of G'. Suppose there isn't a cycle, then there's a node with no incoming edge. Picking it as a starting point, since weakly connected traverse to last node in component. Since every node has 1 outgoing edge, this must too, which must go into the component.

$\square$

Minimum Spanning Trees

Theorem

Let G' be the graph where the nodes are the (unmerged) components, and the edges (directed) are the MWOEs. Then, there's a cycle of length 2 in each component (under weak-conectedness) of this graph.

Proof.

- Consider any component of G'. Suppose there isn't a cycle, then there's a node with no incoming edge. Picking it as a starting point, since weakly connected traverse to last node in component. Since every node has 1 outgoing edge, this must too, which must go into the component.
- Next, since we assumed edge weights are unique, for cycles of length > 2, the edge weights of MWOEs induce a strict total order.

$\square$

Minimum Spanning Trees

Theorem

Let G' be the graph where the nodes are the (unmerged) components, and the edges (directed) are the MWOEs. Then, there's a cycle of length 2 in each component (under weak-conectedness) of this graph.

Proof.

- Next, since we assumed edge weights are unique, for cycles of length > 2, the edge weights of MWOEs induce a strict total order.
- Thus, the weights would satisfy $w_n < w_{n-1} < \cdots < w_1 < w_n$ which is a contradiction as $w_1 \not< w_1$, where w_i is wlog the edge from node $(i-1)$ to $i \pmod{|C|}$.

□

Minimum Spanning Trees

Theorem

Let G' be the graph where the nodes are the (unmerged) components, and the edges (directed) are the MWOEs. Then, there's a cycle of length 2 in each component (under weak-conectedness) of this graph.

Proof.

- Next, since we assumed edge weights are unique, for cycles of length > 2, the edge weights of MWOEs induce a strict total order.
- Thus, the weights would satisfy $w_n < w_{n-1} < \cdots < w_1 < w_n$ which is a contradiction as $w_1 \not< w_1$, where w_i is wlog the edge from node $(i-1)$ to $i \pmod{|C|}$.
- Note a cycle of length 2 does not violate this, as this is the only case where the MWOEs are the same edge in the underlying graph.

□

Minimum Spanning Trees

Algorithm Sketch

1. Keep track of components ✓
2. Find MWOE for each component ✓
3. Merge components ✓
    1. Elect a new leader ✓
4. Terminate when only one component.

Minimum Spanning Trees

Issue #4

When do we know to terminate?

Termination

Once there is only a single remaining component, all test messages sent by nodes in the component to find a MWOE will fail, and thus the leader will determine the algorithm has terminated.

Minimum Spanning Trees

Algorithm Sketch

1. Keep track of components ✓
2. Find MWOE for each component ✓
3. Merge components ✓
    1. Elect a new leader ✓
4. Terminate when only one component. ✓

Are we done?

Minimum Spanning Trees

Algorithm Sketch

1. Keep track of components ✓
2. Find MWOE for each component ✓
3. Merge components ✓
   1. Elect a new leader ✓
4. Terminate when only one component. ✓

Are we done?

One More Issue: Synchronization

At each round, components may be of different sizes, and thus finding MWOEs, and merging components may take a different number of rounds!

Minimum Spanning Trees

One More Issue: Synchronization

At each round, components may be of different sizes, and thus finding MWOEs, and merging components may take a different number of rounds!

Solution

- We a priori pick an upper bound on the number of rounds any component may take, and arrange execution into a sequence of phases where a component will simply wait if it finishes early in a phase.

Minimum Spanning Trees

One More Issue: Synchronization

At each round, components may be of different sizes, and thus finding MWOEs, and merging components may take a different number of rounds!

Solution

- We a priori pick an upper bound on the number of rounds any component may take, and arrange execution into a sequence of phases where a component will simply wait if it finishes early in a phase.
- In particular, we pick this bound to be $O(|V|)$ since all steps including
  1. sending $\textsc{Test}$ messages to neighbors (each vertex has at most $|V|$ neighbors)
  2. convergecasting
  3. broadcasting from the leader

  are bounded as $O(|V|)$, as well as merging and leader election.

Minimum Spanning Trees: Analysis

- **Time Complexity**: $O(n \log n)$. As in the classical case, in each phase the size of each component grows by at least a factor of 2. Thus, at most $O(\log n)$ phases, and we have $O(n)$ rounds per phase.
- **Communication Complexity:** $O(n \log n + |E|)$. In each phase $O(n)$ messages are sent during leader election. As above, there are $O(\log n)$ phases. Moreover, in finding MWOEs, $O(|E|)$ messages are sent in total, since a $\textsc{Test}$ message is sent along any given edge exactly once.

Minimum Spanning Trees: Analysis

- **Time Complexity**: $O(n \log n)$. As in the classical case, in each phase the size of each component grows by at least a factor of 2. Thus, at most $O(\log n)$ phases, and we have $O(n)$ rounds per phase.
- **Communication Complexity:** $O(n \log n + |E|)$. In each phase $O(n)$ messages are sent during leader election. As above, there are $O(\log n)$ phases. Moreover, in finding MWOEs, $O(|E|)$ messages are sent in total, since a $\textsc{Test}$ message is sent along any given edge exactly once.
- **Relaxing edge-weight uniqueness**: Make the comparison key (w, u, v) where w is the weight of the edge, and $u < v$ are the UIDs of the endpoints. This induces a strict total order which is necessary and sufficient.

References

Gallager, R. G., Humblet, P. A., and Spira, P. M. (1983).
A distributed algorithm for minimum-weight spanning trees.
ACM Transactions on Programming Languages and systems (TOPLAS), 5(1):66–77.

Questions?

Thank You!