All-to-All Communication

Thanks to Thomas Locher from ABB Research for basis of slides!
Repetition

- **Tree**: connected graph without cycles
- **Spanning subgraph**: A subgraph that spans all vertices of a graph
- **Minimum spanning tree (MST)**: Spanning tree with the least total weight among all the spanning trees of a weighted and connected graph
Definitions

Tree?
Spanning tree?
Definitions

Tree? Yes
Spanning tree? No
Definitions

Tree?
Spanning tree?
Definitions

Tree? No

Spanning tree? No, but spanning subgraph!
Definitions

BFS from v?
BFS from v? No.
Definitions

BFS from v?
BFS from v? Yes.
Definitions

MST?
MST? No.
Note: BFS can also be defined wrt weights (shortest path spanning tree)!
System Model

• The network is a clique $G=(V,E,w)$ where $w(e)$ denotes the weight of edge $e \in E$ and $|V|=n$

• **Edge weights** unique (not critical, can break symmetries by Ids), can be represented with $O(\log n)$ bits

• Each node has a **distinct ID** of $O(\log n)$ bits

• Each node knows all the edges it is incident to and their weights

• Each node knows about all the other nodes

• The **synchronous communication model** is used

• Results so far?
Definitions

• Many algorithms run in **phases**: in each phase, a subset of the \(|V|-1\) MST edges are chosen. The MST «grows over time»!

• We say that nodes that are directly or indirectly connected by the edges chosen so far belong to the same **fragment (or «cluster»)**

• **Blue edge**: lightest edge out of a fragment

• **Minimum weight outgoing edge (MWOE)** is the edge with the lowest weight incident to a **given node** and leading to another fragment. This edge is a **blue edge candidate**!
The GHS Algorithm

Gallager-Humblet-Spira

Initially, each node is root of its own fragment.
Repeat (until all nodes in same fragment)
1. nodes learn fragment IDs of neighbors
2. root of fragment finds blue edge \((u,v)\) by convergecast
3. root sends message to \(u\) (inverting parent-child)
4. if \(v\) also sent a merge request over root depending on smaller ID (make)
5. new root informs fragment about „MST“ of fragment: new fragment
Repetition

**Time Complexity**

\[ O(n \log n) \] where \( n \) is graph size.

**Message Complexity**

\[ O(m \log n) \] where \( m \) is number of edges: at most \( O(1) \) messages on each edge in a phase.

Can we do better on special networks? E.g., the clique?
MST on Clique

Can we do better on special networks? E.g., the clique?
Yes we can:
1. Send all weights to «leader» (or even to all other nodes)
2. Compute solution locally (Prim/Kruskal)
3. Broadcast result

Complexity? Time $O(1)$, Messages $O(n)$, so optimal! 😊
Bounded Message Size

Not very scalable! Messages are huge: contain n-1 weights

Message Size

Message size limited to $O(\log n)$ bits. (Assume all variables in network all of size $O(\log n)$, e.g., weights, identifiers, ...)

Simple algorithm required messages of size $O(n \log n)$ bits! So $n$ rounds to transmit over a single link.

Note: locality is no longer the problem, but the communication!
Bounded Message Size

Not very scalable! Messages are huge: contain n-1 weights

Message Size

Message size limited to $O(\log n)$ bits.

So how to do it in time less than $O(n)$ with bounded message size? What about GHS algo?

Guess possible performance!
A Simple Algorithm

**Phase k**: Code for node v in fragment F

**Input**: Set of chosen edges that build node fragments

1. Compute the MWOE of v («blue edge candidate»)
2. Send the MWOE to all nodes in the same fragment
3. Receive messages from the other nodes
4. If own MWOE is the lightest in fragment, then broadcast it to all other nodes in the clique and add this edge. So all other nodes always know which fragments there are / merge currently!
5. Receive other broadcast messages and add those edges as well
A Simple Algorithm

Example: How will it proceed?

**Phase k:** Code for node $v$ in cluster $F$

**Input:** Set of chosen edges that build node clusters

1. Compute the MWOE («blue edge candidate»)
2. Send the MWOE to all nodes in the same cluster
3. Receive messages from the other nodes
4. If own MWOE is the lightest, then broadcast it to all other nodes in the clique and add this edge. So all other nodes always know which clusters there are / merge currently!
5. Receive other broadcast messages and add those edges as well
Example:

**Round 1:**

Single node component: Broadcast the lightest edge to all fragments = neighbors. Here, the lightest incident edge of each individual node is a blue edge!

Add edges and update fragment
A Simple Algorithm

Example:

Round 2:

Send MWOE edge to all nodes in the same fragment

Broadcast the lightest edge to all other nodes in remaining fragments.
A Simple Algorithm

• Example:

Round 2:

Why correct?

What is runtime?

Add edges and update fragments
A Simple Algorithm

• Example:

Round 2:

Why correct?
- By blue edge rule, MST will emerge
- No message too large: one weight per edge and time

What is runtime?
Since the minimum fragment size doubles in each round, the algorithm computes the MST in $O(\log n)$ rounds!

Can we do better?
- Note: we did not exploit fact that we can send different messages to different neighbors! But could be exploited to speed up!
Fast MST Algorithm: General Idea

• To reduce the number of rounds, fragments have to grow faster!

• In our simple algorithm, we used the MWOE / blue edge of each fragment to merge clusters.

• With this approach, the minimum fragment size doubled in each phase.

• Idea how to speed it up?

• What if we could use the k lightest outgoing edges of each fragment, where k is the fragment size?
Impact of Fragment Growth

“What if we could use the k lightest outgoing edges of each fragment, where k is the fragment size?”

• Cluster of size k merges with k clusters of size k, so next fragment of quadratic size k*k (in contrast to 2*k so far)

• So:
  • 1=2^0, 2=2^1, 4=2^2, 16=2^4, 256=2^8 ....

• In general:
  • After i steps, 2^{2i} in contrast to 2^i so far.

• How fast is this?
  • n = 2^{2i} ↔ log n = 2^i ↔ log log n = i
Impact of Fragment Growth

Let $\beta_k$ denote the minimum fragment size after phase $k$, then it holds for our simple algorithm that

$$\beta_{k+1} \geq 2 \cdot \beta_k$$

and $\beta_0 := 1$

thus $\beta_k \geq 2^k \implies k \in O(\log n)$
Impact of Fragment Growth

• We will derive an algorithm for which it holds that:

$$\beta_{k+1} \geq \beta_k \cdot (\beta_k + 1)$$

• Thus the fragment sizes grow quadratically as opposed to merely double

• In order to achieve such a rate, information has to be spread faster!

• We will use a simple trick for that...
A Trick to Avoid Link Congestion

- Link capacity bounded: we cannot send much information over a single link...
A Trick to Avoid Link Congestion

• However, much information can be sent from different nodes to a particular node $v_0$!

• A node can simply send parts of the information that it wants to transmit to a specific node to some other nodes. These nodes can send all parts to the specific node in one step!

• This can be used to share workload!
Our new algorithm will execute the following steps in each phase.

Let $\beta$ be the minimal fragment size (the actual size can be larger!)

1. Each fragment computes the $\beta$ lightest edges $e_1, ..., e_\beta$ to other distinct fragments

2. Assign at most one of those lightest edges to the members of the fragment («responsible» for this edge)!
Fast MST Algorithm: General Idea

3. Each node with an edge \( <v,u,w({v,u})> \) assigned to it, sends \( <v,u,w({v,u})> \) to a specific node \( v_0 \) (global!)

4. Node \( v_0 \) computes the lightest edges that can be safely added to the spanning tree (no cycle)

Step 2 and 3 together is exactly our trick!
Fast MST Algorithm: General Idea

5. Node $v_0$ sends a message to a node, if its assigned edge is added to the spanning tree.

6. Each node, that received a message, broadcasts it to all other nodes (→ All nodes have to know about all added edges and new fragments!)
Fast MST Algorithm: General Idea

- This way, more edges can be added in one phase!
- However, how does it really work?
- There are a few obvious problems...
Fast MST Algorithm: Problem 1

First problem:

• How to compute the $\beta$ lightest outgoing edges of a specific fragment?

→ Not so difficult: procedure Cheap_Out (idea?)
Second problem:

• How can the designated node $v_0$ know which edges can be added safely?

• Let's illustrate this problem with an example graph!
Fragment Cycle Detection

- In our example:
  \[ |V| = n = 12 \]
  \[ \beta = 2 \text{ (minimum fragment size)} \]

- Which are the lightest \( \beta = 2 \) outgoing edges of the fragments?

- All edges except for edge \( e \)! So this is the picture of the designated node \( v_0 \) after receiving the \( \beta = 2 \) lightest outgoing edges of each fragment.

- \( v_0 \) does not know about the edge \( e \)! It is the 3rd lightest edge of both adjacent nodes!
$\nu_0$ can construct a logical graph:
its nodes are the fragments and its edges are the $\beta = 2$ lightest outgoing edges
Fragment Cycle Detection

Based on the knowledge of the $\beta = 2$ lightest outgoing edges, $v_0$ can locally merge nodes of the logical graph into fragments.

Can for sure take the MWOE / blue edges of each component!

So build super-fragments by connecting any pair with the lightest connecting edge: edges with weights 1, 2, 3 and 4 are safe.

But can I continue adding edges in ascending order of weight, as long as no loop occurs?
Fragment Cycle Detection: Problem

Can I add edge with weight 6? Loop-free in logical graph!

No! The edge is not part of the MST! Not a blue edge between the components!

The problem is that in both (super)fragments at least one of the nodes has already used up all of its $\beta$ outgoing edges. The $(\beta+1)$th outgoing edge might be lighter than other edges, but does not appear in logical graph!

So, when is it safe to add an edge?
Fast MST Algorithm: The Algorithm Step by Step

• Let's put everything together and solve the open problems!

• Initially, each node is itself a fragment of size 1 and no edges are selected.

• The algorithm consists of 6 steps. Each step can be performed in constant time (communication round).

• All 6 steps together build one phase of the algorithm, thus the time complexity of one phase is $O(1)$.

• A specific node in each fragment $F$, e.g. the node with the smallest ID, is considered the leader $\ell(F)$ of the fragment $F$ (since our algorithm ensures that nodes know fragment, they also know leader).
Fast MST Algorithm: The Algorithm Step by Step

Step 1

a) Each node $v$ computes the minimum-weight edge $e(v,F)$ that connects $v$ to any node of some other fragment $F$, for all other fragments $F$

b) Each node $v$ sends $e(v,F)$ to the leader $\ell(F)$ for all fragments other than the own fragment
Fast MST Algorithm: The Algorithm Step by Step

Step 2

a) Each leader \( v \) of a fragment \( F \) (i.e. \( \ell(F) = v \)) computes the lightest edge between \( F \) and every other fragment

b) Each leader \( v \) performs procedure \texttt{Cheap\_Out} \( \rightarrow \) Selects the \( \beta \) lightest outgoing edges and appoints them to its nodes

If edge \( e \) is appointed to \( v \), then \( v \) is denoted \( e \)'s guardian \( g(e) \) (assignment needed for our load balancing trick!)
Fast MST Algorithm: The Algorithm Step by Step

Procedure Cheap_Out

Code for the leader of fragment F

Input: Lightest edge \( e(F,F') \) for every other fragment \( F' \)

1. Sort the input edges in increasing order of weight
2. Define \( \beta = \min\{|F|, \text{<# of fragments>}\} \)
   (Size of own fragment as long as many other fragments. Why not more? Cannot communicate so much info!)
3. Choose the first \( \beta \) nodes of the sorted list
4. Appoint the node with the \( i \)-th largest ID as the guardian of the \( i \)-th edge, for \( i = 1,\ldots,\beta \)
5. Send a message about the edge to the node it is appointed to
Step 3

All nodes that are guardians for a specific edge send a message to the designated node $v_0$, e.g. the node with the **smallest ID in the whole graph**

$v_0$ knows the $\beta$ lightest outgoing edges of each fragment!
Fast MST Algorithm: The Algorithm Step by Step

Step 4

a) $v_0$ locally performs procedure \textbf{Const\_Frags} $\rightarrow$ Computes the edges to be added

b) For all added edges, $v_0$ sends a message to $g(e)$
Fast MST Algorithm: The Algorithm Step by Step

• How does Const_Frags work?
• We have seen: problem occurs when all $\beta$ outgoing edges of a fragment are used up!
• More precisely, a problem occurs only if there is at least one “full” fragment in each of the two super-fragments which are supposed to be merged! (Otherwise we would see the edge.)

Used up all edges!
Fast MST Algorithm: The Algorithm Step by Step

• How does Const_Frags work?

• We call a super-cluster containing a cluster that used up all of its $\beta$ edges finished (“not safe”).

If an edge is the lightest outgoing edge of one super-fragment that is not finished, then it is still safe to add it, no matter if the other super-fragment is finished, since we are sure that there is no better edge to connect the unfinished super-fragment to other fragments.
Fast MST Algorithm: The Algorithm Step by Step

Procedure Const_Frags

Code for the designated node $v_0$

Input: the $\beta$ lightest outgoing edges of each fragment

1. Construct the logical graph
2. Sort the input edges in increasing order of weight
3. Go through the list, starting with the lightest edge:
   - If the edge can be added without creating a cycle then add it
   - else drop it
Fast MST Algorithm: The Algorithm Step by Step

Procedure Const_Frags

If two (super-)fragments are merged, then the new super-fragment is declared finished if

- the edge is the heaviest edge of a fragment in any of the two super-fragment or
- any of the two super-fragments is already finished.

If the edge is dropped (→ both clusters already belong to the same super-fragment), then the super-fragment is declared finished if

- the edge is the heaviest edge of any of the two fragments

Note: If a super-fragment is declared finished then it will remain finished until the end of the phase.

Final Step

All edges between finished super-clusters are deleted (before looking at the next lightest edge)
Fast MST Algorithm: The Algorithm Step by Step

Step 5

All nodes that received a message from $v_0$ broadcast their edge to all other nodes.

Step 6

Each node adds all edges and computes the new fragments (complete view!).

If the number of clusters is greater than 1, then the next phase starts.
The entire algorithm for node $v$ in fragment $F$

1. Compute the minimum-weight edge $e(v,F')$ that connects $v$ to fragment $F'$ and send it to $\ell(F')$ for all fragments $F' \neq F$

2. if $v = \ell(F)$: Compute lightest edge between $F$ and every other fragment. Perform Cheap_Out

3. if $v = g(e)$ for some edge $e$: Send $<e>$ to $v_0$

4. if $v = v_0$: Perform Const_Frags. Send message to $g(e)$ for each added edge $e$

5. if $v$ received a message from $v_0$: Broadcast it

6. Add all received edges and compute the new fragments
Analysis: Correctness

• It suffices to show that whenever an edge is added, it is part of the MST → We only have to analyze Const_Frags!

• Proof [Sketch]: We only have to show that we always add the lightest outgoing edge of each super-fragment. This is always the right choice!
Analysis: Correctness

- Assume edge $e$ is used to merge super-fragment $SC_1$ and $SC_2$. W.l.o.g., assume that $SC_1$ is not finished and that $e$ is one of the $\beta$ lightest outgoing edges of its fragment.

- We will show now that $e$ is the MWOE of $SC_1$!

- Assume that there is a lighter outgoing edge $e'$ ($w(e') < w(e)$), incident to a fragment $C$ that connects super-fragment $SC_1$ to super-fragment $SC_3$. 

![Diagram showing the relationship between fragments SC1, SC2, SC3, and the edge e connecting them.]

- is a fragment
IV. Analysis: Correctness

• Case 1: $e'$ is among the $\beta$ lightest outgoing edges of its fragment $C$.
  - Since $w(e') < w(e)$, $e'$ must have been considered before $e$, thus either $SC_1$ and $SC_3$ have been merged before or $e'$ was dropped because $SC_1 = SC_3$. Either way, $e'$ cannot be an outgoing edge when the algorithm adds $e$.
  - Contradiction!

[Diagram of a graph $\tilde{G}$ with fragments $SC_1$, $SC_2$, and $SC_3$. The edges $e$ and $e'$ are shown, with $e'$ being the heaviest outgoing edge of its fragment $C$.]

is a fragment
IV. Analysis: Correctness

• Case 2: \(e'\) is **not** among the \(\beta\) lightest outgoing edges of its fragment \(C\).

• Case 2.1: There is an edge \(e''\) among the \(\beta\) lightest outgoing edges from fragment \(C\) leading to the same fragment \(C'\).

It follows that \(w(e'') < w(e')\). Since \(SC_1 \neq SC_3\), \(e''\) has not been considered yet, thus \(w(e) < w(e'')\).

Hence we have that \(w(e) < w(e')\).

\(\rightarrow\) Contradiction!
IV. Analysis: Correctness

• Case 2: $e'$ is not among the $\beta$ lightest outgoing edges of its fragment $C$.

• Case 2.2: None of the $\beta$ lightest outgoing edges of $C$ lead to $C'$.

Thus, all $\beta$ outgoing edges have lower weights than $e'$, also the heaviest of these edges $e''$, i.e., $w(e'') < w(e') < w(e)$. Hence, edge $e''$ must have been inspected already. Since it is the heaviest (last) edge of some fragment, $SC_1$ must now be finished.

$\rightarrow$ Contradiction!
Analysis: Time Complexity

• Each phase requires $O(1)$ rounds, but how many phases are required until termination?

• Reminder: $\beta_k$ denotes the minimum fragment size in phase $k$.

\[
\text{Lemma 2: It holds that } \beta_{k+1} \geq \beta_k(\beta_k+1).
\]
Analysis: Time Complexity

• Proof [Sketch]:
We prove a stronger claim: Whenever a super-fragment is declared finished in phase $k+1$, it contains at least $\beta_k+1$ fragments.

• Each fragment has (at least) $\beta_k$ outgoing edges in phase $k+1$, since $\beta_k$ is the minimum fragment size after phase $k$. 

![Diagram](image-url)
Analysis: Time Complexity

• Case 1: The super-fragment is declared finished after one of its fragment has used up all of its $\beta_k$ outgoing edges. Let C be this fragment.

• Let's call those edges 1, 2, ..., $\beta_k$ leading to the fragments $C_1$, $C_2$, ..., $C_\beta$.

• If the inspection of an edge does not result in a merge, then the clusters already belong to the same super-fragment! If there is a merge, then they belong to the same super-fragment afterwards.
Analysis: Time Complexity

• Thus, at the end, the super-fragment contains at least \( C, C_1, C_2, ..., C_\beta \)!
\[ \rightarrow \] The super-fragment contains at least \( \beta_k + 1 \) fragments.

• Case 2: The super-fragment is declared finished after merging with an already finished super-fragment.

• Using an inductive argument, the finished super-fragment must already contain at least \( \beta_k + 1 \) clusters, since one of its clusters has used up all of its \( \beta_k \) edges.
Analysis: Time Complexity

Theorem 1: The time complexity is $O(\log \log n)$ rounds.

Proof: According to Lemma 2, it holds that $\beta_{k+1} \geq \beta_k(\beta_k + 1)$. Furthermore, we have that $\beta_0 := 1$. Hence it follows that

$$\beta_k \geq 2^{2^{k-1}}$$

for every $k \geq 1$. Since $\beta_k \leq n$, it follows that $k \leq \log(\log n) + 1$. Since each phase requires $O(1)$ rounds, the time complexity is $O(\log \log n)$. 
Analysis: Time Complexity

**Theorem 2**: The message complexity is $O(n^2 \log n)$.

- The proof is simple: Count the messages exchanged in Steps 1, 3, 4, and 5. We will not do this here.

- Adler et al. showed that the minimum number of bits required to solve the MST problem in this model is $\Omega(n^2 \log n)$. Thus, this algorithm is *asymptotically optimal*!
Overview

I. Introduction
II. Previous Results
III. Fast MST Algorithm
IV. Analysis
V. Summary
  ➢ Results & Conclusions
VI. Extensive Example
"An obvious question we leave open is whether the algorithm can be improved, or whether there is an inherent lower bound of $\Omega(\log \log n)$ on the number of communication rounds required to construct an MST in this model."
Extensive Example: The Graph

All other edges are heavier!!!
Extensive Example: Phase 1

1. Not necessary
2. Not necessary
3. Send MWOE to $v_0$
4. Const_Frags!
Extensive Example: Phase 1

Const_Frags

1. Construct logical graph
Extensive Example: Phase 1

Const_Frags

2. Add edges

\[\tilde{G}\]
Extensive Example: Phase 1

1. Not necessary
2. Not necessary
3. Send MWOE to $v_0$
4. Const_Frags!
5. Send $e$ to $g(e)$

Only some messages are displayed
Extensive Example: Phase 1

1. Not necessary
2. Not necessary
3. Send MWOE to \( v_0 \)
4. Const_Frags!
5. Send \( e \) to \( g(e) \)
6. Broadcast \( e \) and update the fragments

Only some messages are displayed
1. Compute $e(v,F')$ and send it to $\mathcal{E}(F')$.
1. Compute $e(v,F')$ and send it to $\ell(F')$.

2. Select $\beta = 2$ lightest outgoing edges and appoint guardians.
1. Compute $e(v,F')$ and send it to $\ell(F')$

2. Select $\beta = 2$ lightest outgoing edges and appoint guardians

3. Send appointed edge to $v_0$

4. Const_Frags!
Extensive Example: Phase 2

Const_Frags

1. Construct logical graph

\[ \tilde{G} \]
Extensive Example: Phase 2

Const_Frags

2. Add edges

Edges between finished super-fragments must not be added!
### Extensive Example: Phase 2

1. Compute $e(v,F')$ and send it to $\ell(F')$
2. Select $\beta = 2$ lightest outgoing edges and appoint guardians
3. Send appointed edge to $v_0$
4. Const_Frags!
5. Send $e$ to $g(e)$

Only some messages are displayed
Extensive Example: Phase 2

1. Compute $e(v,F')$ and send it to $\ell(F')$
2. Select $\beta = 2$ lightest outgoing edges and appoint guardians
3. Send appointed edge to $v_0$
4. Const_Frags!
5. Send $e$ to $g(e)$
6. Broadcast $e$ and update the clusters

Only some messages are displayed
1. Compute $e(v,F')$ and send it to $\ell(F')$.
1. Compute $e(v,F')$ and send it to $\ell(F')$

2. Select $\beta = 2$ lightest outgoing edges and appoint guardians

Only some messages are displayed
1. Compute $e(v,F')$ and send it to $\ell(F')$

2. Select $\beta = 2$ lightest outgoing edges and appoint guardians

3. Send appointed edge to $v_0$

4. Const_Frags!
Extensive Example: Phase 3

Const_Frags

1. Construct logical graph

\[ \tilde{G} \]
Extensive Example: Phase 3

Const_Frags

2. Add edges
1. Compute $e(v,F')$ and send it to $\ell(F')$

2. Select $\beta = 2$ lightest outgoing edges and appoint guardians

3. Send appointed edge to $v_0$

4. Const_Frags!

5. Send $e$ to $g(e)$
1. Compute $e(v,F')$ and send it to $\ell(F')$

2. Select $\beta = 2$ lightest outgoing edges and appoint guardians

3. Send appointed edge to $v_0$

4. Const_Frags!

5. Send $e$ to $g(e)$

6. Broadcast $e$ and update the fragments
Extensive Example: After Phase 3

Done!
References


