

Distributed MST Construction

Thomas Locher, ETH Zurich
Roger Wattenhofer, ETH Zurich

Summer 2007



Principles of Distributed Computing

Overview



- I. Introduction of the Problem
- II. Previous Results
- III. The $O(\log \log n)$ MST Algorithm
- IV. Analysis of the Algorithm
- V. Summary
- VI. Extensive Example



Principles of Distributed Computing SS 2007

T. Locher

2

Overview



- I. Introduction of the Problem
 - Definitions & System Model
 - A Simple Algorithm
- II. Previous Results
- III. The New Algorithm
- IV. Analysis of the Algorithm
- V. Summary
- VI. Extensive Example



Principles of Distributed Computing SS 2007

T. Locher

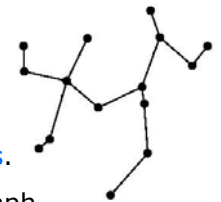
3

I. Introduction of the Problem



Definitions

- A **tree** is a **connected** graph **without cycles**.
- A **subgraph** that **spans all** vertices of a graph is called a **spanning subgraph**.
- Among all the **spanning trees** of a **weighted** and **connected graph**, the one (possibly more) with the least total weight is called a **minimum spanning tree (MST)**.



Principles of Distributed Computing SS 2007

T. Locher

4

I. Introduction of the Problem

Definitions

- In a **MST algorithm**, $|V| - 1$ edges have to be chosen in total. In each **phase** of the algorithm, probably only a fraction of those edges are chosen.
- Nodes that are directly or indirectly connected using chosen edges only belong to the same **cluster**.
- The **minimum weight outgoing edge (MWOE)** is the edge with the lowest weight among all incident edges leading to other **clusters**.



I. Introduction of the Problem

Usage of MST

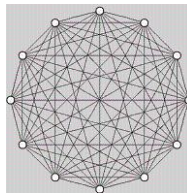
- Minimize the cost associated with global operations such as **broadcasts!**
- Minimize the message complexity:
Avoid traffic explosion by using a **spanning tree** (no cycles!)
- Minimize the time complexity:
If the edge weights represent the delay on the link, then MST minimizes the **execution time**.



I. Introduction of the Problem

System Model

- The system is represented by a **complete weighted undirected graph** $G=(V,E,w)$ where $w(e)$ denotes the weight of edge $e \in E$ and $|V|=n$. All **edge weights** are different (w.l.o.g.).
- 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 model** is used.



I. Introduction of the Problem

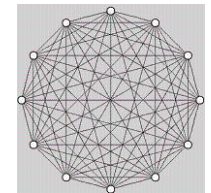
Synchronous Model

Communication advances in **global rounds**.

In each **round**, processes **send** messages, **receive** messages and do some **local computation**.

The **time complexity** is the number of rounds until the computation terminates in the worst case.

The **message complexity** is the number of messages exchanged in the worst case.



I. Introduction of the Problem

Getting a Feeling for the Problem...

How hard is it to compute the MST in a distributed system (assuming a fully connected graph)?

All nodes know the weights of all incident edges. If all nodes send this information to all other nodes, then all nodes suddenly have the entire picture!

→ A simple algorithm that requires only one round!

However, that is not really interesting...

Therefore, the message size is limited to $O(\log n)$ bits!

Note that the previous algorithm requires messages of size $O(n \log n)$!



I. Introduction of the Problem

Getting a Feeling for the Problem...

Since each **node ID** (and edge weight) requires $O(\log n)$ bits, this implies that only a constant number of node IDs (and edge weight) can be packed into a single message!

We demand that all nodes know the **MST** at the end of the computation!

How can the MST be constructed now?

→ Let's look at a simple algorithm first...

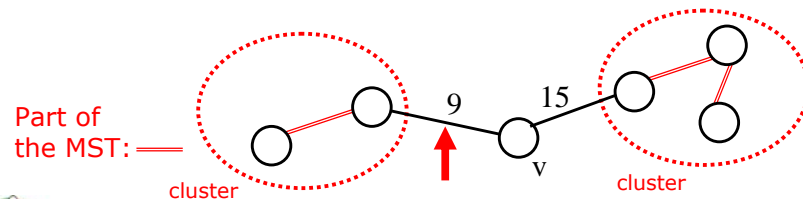


I. Introduction of the Problem

A Simple Algorithm

All MST algorithms (local or distributed) are based on the following lemma:

Lemma 1: It is always safe to add an edge to the spanning tree, if this edge is the **MWOE** of a node v .



I. Introduction of the Problem

A Simple Algorithm

Phase k: Code for node v in cluster F

Input: Set of chosen edges that build node clusters

1. Compute the MWOE
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 and add this edge (→ All nodes have to know all clusters after each round)
5. Receive other broadcast messages and add those edges as well



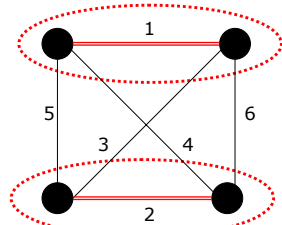
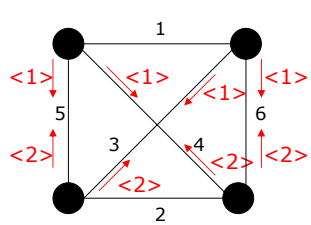
I. Introduction of the Problem

A Simple Algorithm

Example

$$\langle w(\{v,u\}) \rangle = \langle v,u,w(\{v,u\}) \rangle$$

Round 1:



Broadcast the lightest edge to the other nodes

Add edges and update clusters



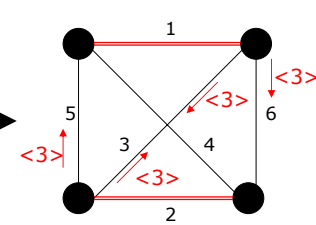
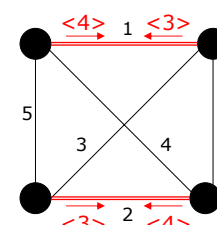
I. Introduction of the Problem

A Simple Algorithm

Example

$$\langle w(\{v,u\}) \rangle = \langle v,u,w(\{v,u\}) \rangle$$

Round 2:



Send MWOE to all nodes in the same cluster

Broadcast the lightest edge to the other nodes



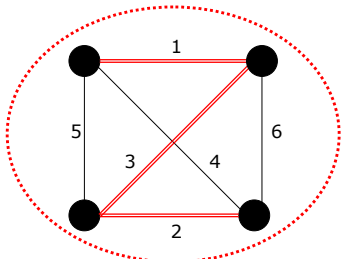
I. Introduction of the Problem

A Simple Algorithm

Example

$$\langle w(\{v,u\}) \rangle = \langle v,u,w(\{v,u\}) \rangle$$

Round 2:



Add edges and update clusters

The algorithm is obviously correct. Since the minimum cluster size doubles in each round, the algorithm computes the MST in

$O(\log n)$ rounds!

Can it be improved???

Lower bound???



Overview

- I. Introduction of the Problem
- II. Previous Results
 - Lower and Upper Bounds
 - Open Questions
- III. The $O(\log \log n)$ MST Algorithm
- IV. Analysis of the Algorithm
- V. Summary
- VI. Extensive Example



II. Related Work



Previous Results

Δ denotes the constant **diameter** of the graph, i.e. the maximum **distance** between any two nodes of the graph.

	Known Algorithms	Known Lower Bounds
$\Delta = 1$	$O(\log n)$???
$\Delta = 2$	$O(\log n)$???
$\Delta \geq 3$	$O(\sqrt{n})$	$\Omega(n^{1/4})$



II. Related Work



Previous Results

Even our simple algorithm has this complexity!!

Interesting „jump“!

	Known Algorithms	Known Lower Bounds
$\Delta = 1$	<u>$O(\log n)$</u>	???
$\Delta = 2$	$O(\log n)$???
$\Delta \geq 3$	$O(\sqrt{n})$	$\Omega(n^{1/4})$



II. Related Work



Previous Results

We will now derive an algorithm with time complexity $O(\log \log n)$!

	Known Algorithms	Known Lower Bounds
$\Delta = 1$	$O(\log \log n)$???
$\Delta = 2$	$O(\log n)$???
$\Delta \geq 3$	$O(\sqrt{n})$	$\Omega(n^{1/4})$



Overview



- I. Introduction of the Problem
- II. Previous Results
- III. The $O(\log \log n)$ MST Algorithm
 - General Idea & Problems
 - The Algorithm Step by Step
- IV. Analysis of the Algorithm
- V. Summary
- VI. Extensive Example



III. The $O(\log \log n)$ MST Algorithm

General Idea

In order to reduce the number of rounds, obviously clusters have to grow faster!

In our simple algorithm, we used the **MWOE** of each cluster to merge clusters.

With this approach, the **minimum cluster size doubled** in each phase.

It would certainly be faster, if the **t** lightest outgoing edges of each cluster were used, where **t** is in the order of the number of nodes in the cluster!

→ This is exactly what our new algorithm will do!



III. The $O(\log \log n)$ MST Algorithm

General Idea

In order to reduce the number of rounds, obviously clusters have to grow faster!

Let β_k denote the **minimum cluster size in phase k**, then it holds for our simple algorithms that

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

and

$$\beta_0 := 1$$

thus

$$2^k \leq \beta_k \leq n \rightarrow k = O(\log n)$$



III. The $O(\log \log n)$ MST Algorithm

General Idea

Our goal is to improve the following inequality:

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

We will derive an algorithm for which it holds that:

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

Thus the cluster sizes grow quadratically as opposed to merely double in each phase! In order to achieve such a rate, information has to be spread faster!

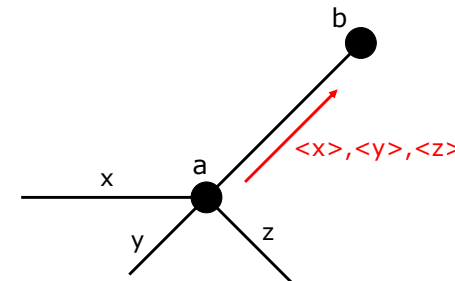
We will use a simple trick for that...



III. The $O(\log \log n)$ MST Algorithm

General Idea

Unfortunately, we cannot send a lot of information over a single link...



III. The $O(\log \log n)$ MST Algorithm

General Idea

However, we can send a lot of information from different nodes to a particular node v_0 !

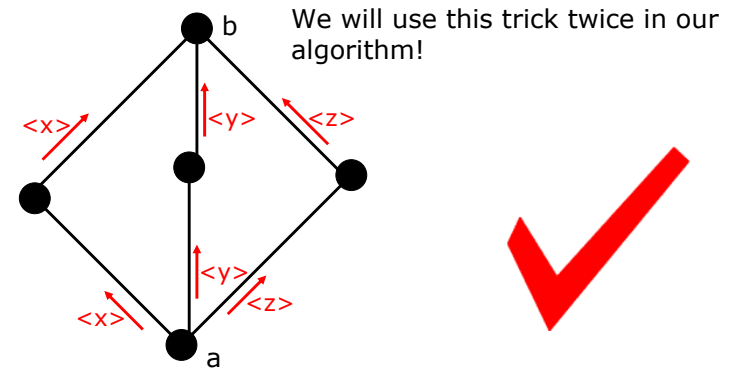
A node can simply send parts of the information, that it wants to transmit to a specific node, to other nodes. These nodes can send all parts to the specific node in one step!!

This can be used to share workload!!!



III. The $O(\log \log n)$ MST Algorithm

General Idea



III. The $O(\log \log n)$ MST Algorithm

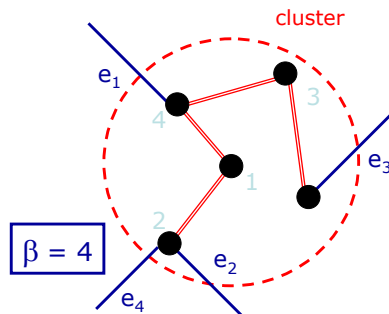
General Idea

Our new algorithm will execute the following steps in each phase.

Let β be the minimal cluster size.

1. Each cluster computes the β lightest edges e_1, \dots, e_β to other distinct clusters

2. Assign at most one of those lightest edges to the members of the cluster!



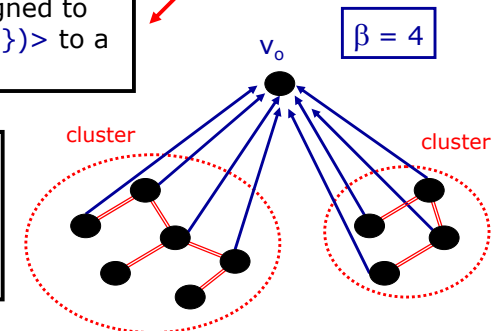
III. The $O(\log \log n)$ MST Algorithm

General Idea

3. Each node with an edge $\langle v, u, w(\{v, u\}) \rangle$ assigned to it, sends $\langle v, u, w(\{v, u\}) \rangle$ to a specific node v_0

Step 2 and 3 together is exactly our trick!

4. Node v_0 computes the lightest edges that can be safely added to the spanning tree

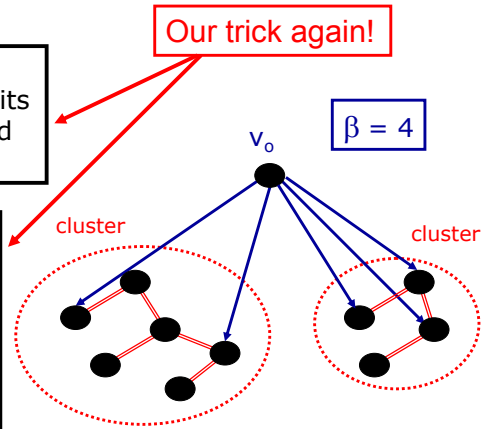


III. The $O(\log \log n)$ MST Algorithm

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 (\rightarrow All nodes have to now about all added edges!)



III. The $O(\log \log n)$ MST Algorithm

General Idea

This way, more edges can be added in one phase!
However, it is not clear yet how fast it really is...

Furthermore, we do not know yet how these steps work in detail!!!

\rightarrow There are a few obvious problems...



III. The $O(\log \log n)$ MST Algorithm

Problems

First problem:

How can the β lightest outgoing edges of a specific cluster be computed?

\rightarrow This is actually not so difficult. The procedure `Cheap_Out` in the actual algorithm copes with this problem. We will treat it in the following section.



III. The $O(\log \log n)$ MST Algorithm

Problems

Second problem:

How can the designated node v_0 know which edges can be added without creating a cycle in the constructed graph?

Let's illustrate the problem with an example graph!



III. The $O(\log \log n)$ MST Algorithm

Problems

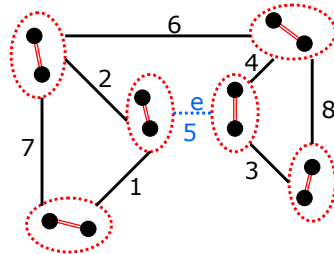
In our example:

$|V| = n = 12$

$\beta = 2$ (minimum cluster size)

This is the picture of the designated node v_0 after receiving the $b = 2$ lightest outgoing edges of each cluster

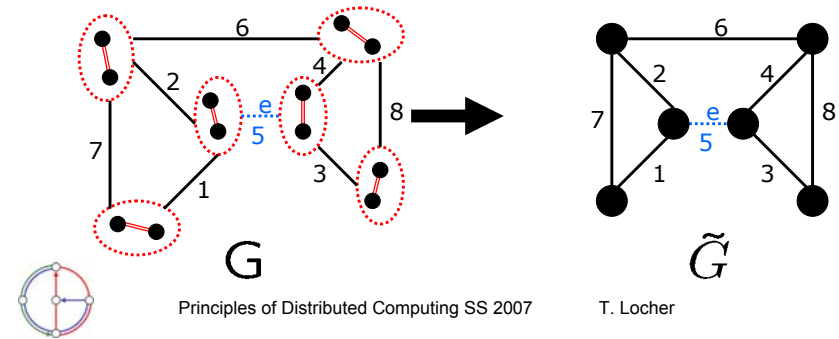
v_0 does not know about the edge e ! It is the 3rd lightest edge of both adjacent clusters!



III. The $O(\log \log n)$ MST Algorithm

Problems

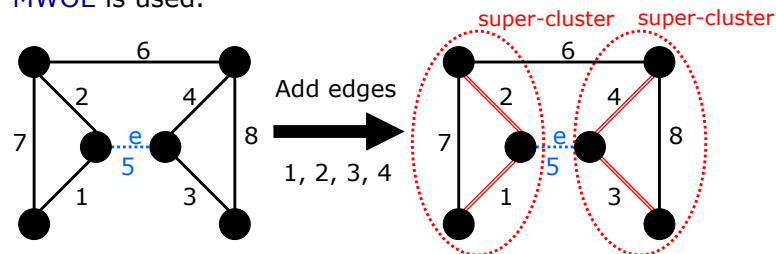
v_0 can construct a logical graph. Its nodes are the clusters and its edges are the $\beta = 2$ lightest outgoing edges.



III. The $O(\log \log n)$ MST Algorithm

Problems

Based on the knowledge of the $\beta = 2$ lightest outgoing edges, v_0 can locally merge nodes of the logical graph into clusters. The 4 edges with weights 1, 2, 3 and 4 can be chosen safely, since always the MWOE is used.



III. The $O(\log \log n)$ MST Algorithm

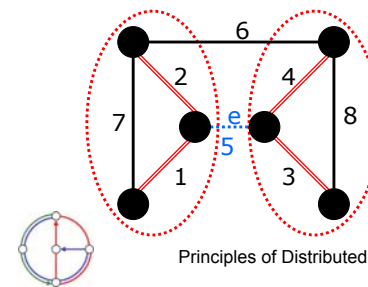
Problems

If the edge with weight 6 is used to finish the construction of the spanning tree, then the resulting tree is not the MST!!!



The problem is that in both (super-)clusters, at least one of the nodes has already used up all of its β outgoing edges. The $(\beta+1)$ th outgoing edge might be lighter than other edges!!!

So, when is it safe to add an edge???



III. The $O(\log \log n)$ MST Algorithm

The Algorithm Step by Step

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

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

The algorithm consists of 6 steps. Each step can be performed in constant time.

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 cluster F , e.g. the node with minimum ID, is considered the leader $\ell(F)$ of the cluster F .

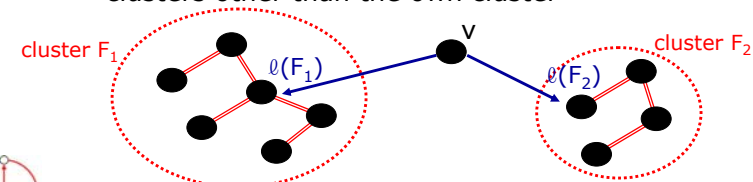


III. The $O(\log \log n)$ MST Algorithm

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 cluster F for all clusters other than the own cluster
- (b) Each node v sends $e(v, F)$ to the leader $\ell(F)$ for all clusters other than the own cluster

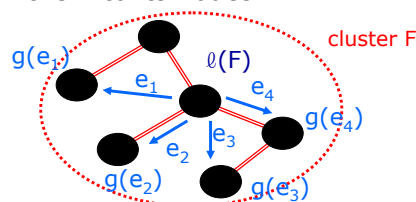


III. The $O(\log \log n)$ MST Algorithm

The Algorithm Step by Step

Step 2

- (a) Each leader v of a cluster F (i.e. $\ell(F) = v$) computes the lightest edge between F and every other cluster
- (b) Each leader v performs procedure `Cheap_Out` → Selects the β lightest outgoing edges and appoints them to its nodes



If edge e is appointed to v , then v is denoted e 's guardian $g(e)$



III. The $O(\log \log n)$ MST Algorithm

The Algorithm Step by Step

Procedure `Cheap_Out`

Code for the leader of cluster F

Input: Lightest edge $e(F, F')$ for every other cluster F'

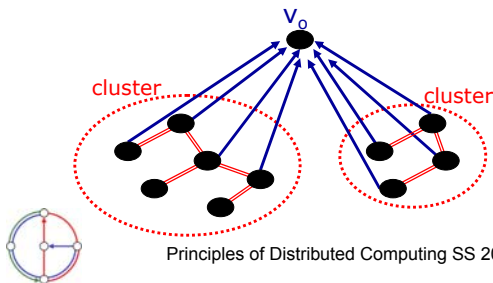
1. Sort the input edges in increasing order of weight
2. Define $\beta = \min\{|F|, \text{<\# of clusters>}\}$
3. Choose the first β nodes of the sorted list
4. Appoint the node with the i th largest ID as the guardian of the i th edge, $i = 1, \dots, \beta$
5. Send a message about the edge to the node it is appointed to



The Algorithm Step by Step

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 **minimal ID** in the graph

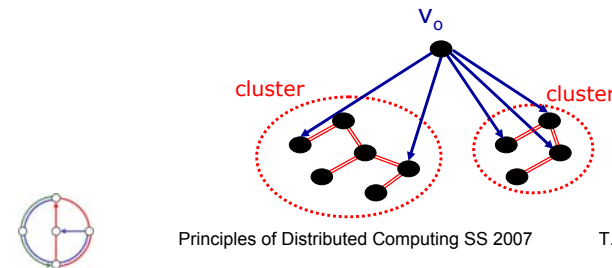


v_0 knows the β lightest outgoing edges of each cluster!

The Algorithm Step by Step

Step 4

- (a) v_0 locally performs procedure `Const_Frags` → Computes the edges to be added
- (b) For all added edges, v_0 sends a message to $g(e)$

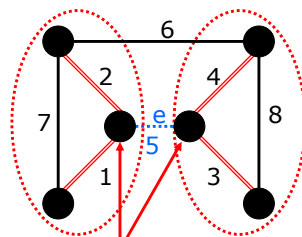


The Algorithm Step by Step

How does `Const_Frags` work?

As we've seen before, a problem occurs when all β outgoing edges of a cluster are used up!

More precisely, a problem occurs only if there is at least one cluster in each of the two **super-clusters** that are supposed to be merged!!



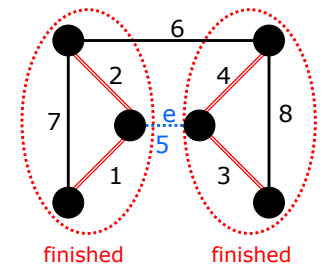
Used up all edges!

The Algorithm Step by Step

How does `Const_Frags` work?

We call a super-cluster containing a cluster that used up all of its β edges **finished**.

If an edge is the lightest outgoing edge of one super-cluster that is not finished, then it is still safe to add it, no matter if the other super-cluster is finished, since we are sure that there is no better edge to connect the unfinished super-cluster to other clusters.



finished finished



The Algorithm Step by Step

Procedure Const_Frags

Code for the designated node v_0

Input: the β lightest outgoing edges of each cluster

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



The Algorithm Step by Step

Procedure Const_Frags

If the two super-clusters were merged then the new super-cluster is declared finished if

- the edge is the heaviest edge of a cluster in any of the two super-clusters or
- any of the two super-clusters is already finished

If the edge is dropped (\rightarrow both clusters already belong to the same super-cluster) then the super-cluster containing the clusters the edge connects is declared finished if

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



The Algorithm Step by Step

Procedure Const_Frags

Note: If a super-cluster is declared finished then it will remain finished

All edges between finished super-clusters are deleted (before looking at the next lightest edge)



Those are the dangerous edges!



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 clusters

If the number of clusters is greater than 1, then the next phase starts



III. The $O(\log \log n)$ MST Algorithm

The Algorithm Step by Step

The entire algorithm for node v in cluster F

1. Compute the minimum-weight edge $e(v, F')$ that connects v to cluster F' and send it to $\ell(F')$ for all clusters $F' \neq F$
2. if $v = \ell(F)$: Compute lightest edge between F and every other cluster. Perform **Cheap_Out**
3. if $v = g(e)$ for some edge e : Send $\langle e \rangle$ 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 clusters



Overview

- I. Introduction of the Problem
- II. Previous Results
- III. The $O(\log \log n)$ MST Algorithm
- IV. Analysis of the Algorithm
 - Correctness
 - Time and Message Complexity
- V. Summary
- VI. Extensive Example

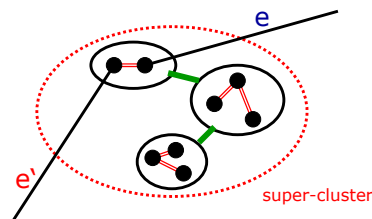


IV. Analysis of the Algorithm

Correctness

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

Proof [Sketch]: We always add the lightest outgoing edge of each super-cluster. Because of Lemma 1, this is always the right choice! Therefore, we only have to watch out that we choose the right edges from the various clusters (since we only know some of the outgoing edges)!!!

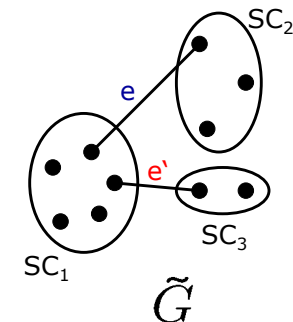


IV. Analysis of the Algorithm

Correctness

Assume edge e is used to merge super-cluster SC_1 and SC_2 . Further assume there is a lighter outgoing edge e' ($w(e') < w(e)$) that connects super-cluster SC_1 with another super-cluster, say, SC_3 .

● is a cluster



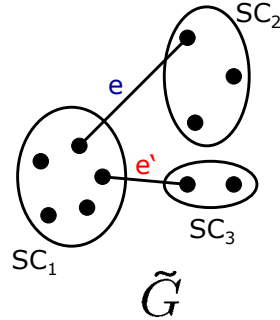
IV. Analysis of the Algorithm

Correctness

Case 1: e' is among the β lightest outgoing edges

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!



IV. Analysis of the Algorithm

Correctness

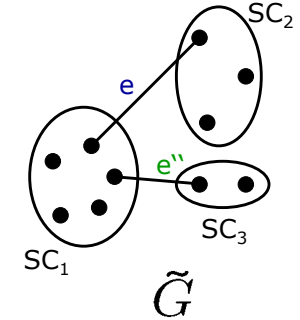
Case 2: e' is not among the β lightest outgoing edges

Case 2.1: There is a lighter edge e'' from the cluster to the super-cluster SC_3 :

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

$w(e) < w(e')$.

→ Contradiction!



IV. Analysis of the Algorithm

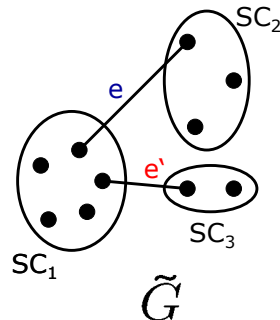
Correctness

Case 2: e' is not among the β lightest outgoing edges

Case 2.2: There is no lighter edge from the cluster to the super-cluster SC_3 :

Thus all β outgoing edges have lower weights than e' . In particular, this holds also for e , thus $w(e) < w(e')$.

→ Contradiction!



IV. Analysis of the Algorithm

Time Complexity

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

Reminder: β_k denotes the **minimum cluster size in phase k** .

Lemma 2: It holds that

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



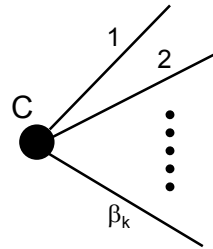
IV. Analysis of the Algorithm

Time Complexity

Proof [Sketch]:

We prove a stronger claim: Whenever a super-cluster is declared finished in phase $k+1$, it contains at least $\beta_k + 1$ clusters.

Each cluster has (at least) β_k outgoing edges in phase $k + 1$, since β_k is the minimal cluster size after phase k .



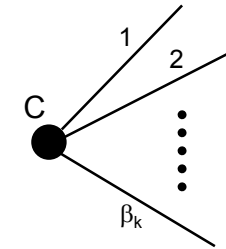
IV. Analysis of the Algorithm

Time Complexity

Case 1: The super-cluster is declared finished after one of its clusters has used up all of its β_k outgoing edges. Let C be a cluster.

Let's call those edges edge 1, 2, ..., β_k leading to the clusters $C_1, C_2, \dots, C_{\beta_k}$.

If the inspection of an edge does not result in a merge, then the clusters already belong to the same super-cluster! If there is a merge, then they belong to the same super-cluster afterwards.



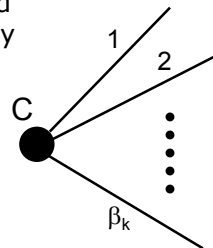
IV. Analysis of the Algorithm

Time Complexity

Thus, at the end, the super-cluster contains at least $C, C_1, C_2, \dots, C_{\beta_k}$! \rightarrow The super-cluster contains at least $\beta_k + 1$ clusters.

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

Using an inductive argument, the finished super-cluster must already contain at least $\beta_k + 1$ clusters, since one of its clusters has used up all of its β_k edges. ■



IV. Analysis of the Algorithm

Time Complexity

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

Proof: By Lemma 2, it holds that $\beta_{k+1} \geq \beta_k (\beta_k + 1)$. Furthermore it holds 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 $O(\log \log n)$. ■



IV. Analysis of the Algorithm



Message 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 that here.

Adler et al. showed that the minimal 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 of the Problem
- II. Previous Results
- III. The New Algorithm
- IV. Analysis of the Algorithm
- V. Summary
 - **Results & Conclusions**
- VI. Extensive Example



V. Summary



Results

The presented algorithm solves the MST problem in the given model in $O(\log \log n)$ rounds.

The algorithm sends $O(n^2 \log n)$ bits in total, which is asymptotically optimal.



V. Summary



Conclusions

„An obvious question we leave open is whether the algorithm can be improved, or is there an inherent lower bound of $\Omega(\log \log n)$ on the number of communication rounds required to construct an MST in this model.“

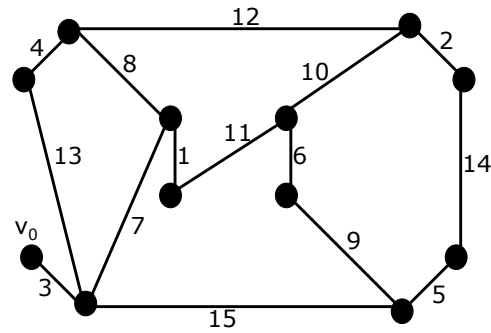
- Is there a faster algorithm?
- Is $\Omega(\log \log n)$ a lower bound?
- Is there an algorithm with time complexity $O(\log \log n)$ for graphs of diameter 2?



VI. Extensive Example



The graph



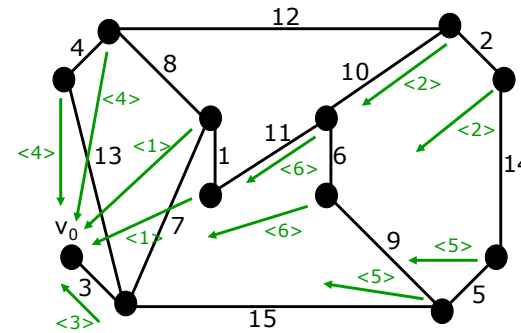
All other edges are heavier!!!



VI. Extensive Example



Phase 1



1. Not necessary
2. Not necessary
3. Send MWOE to v_0
4. Const_Frags!

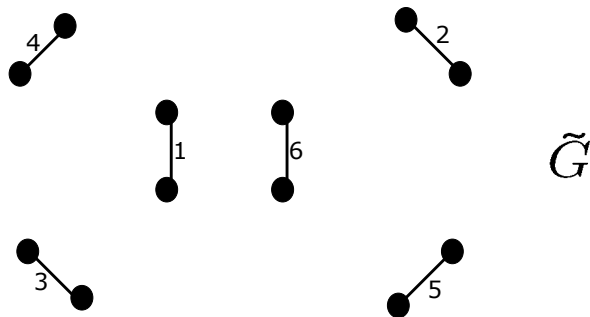


VI. Extensive Example



Phase 1 – Const_Frags

1. Construct logical graph

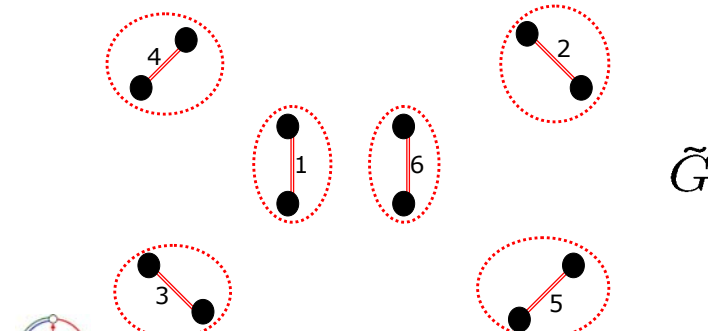


VI. Extensive Example



Phase 1 – Const_Frags

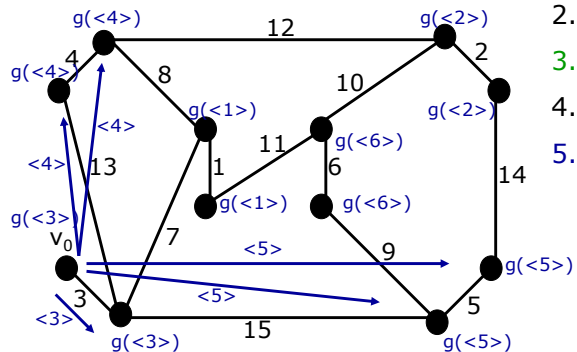
2. Add edges



VI. Extensive Example



Phase 1



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

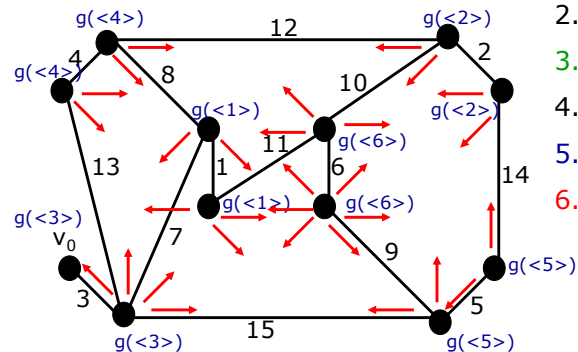
Only some messages are displayed



VI. 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 clusters

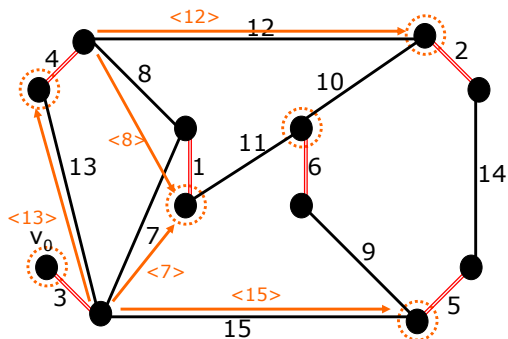
Only some messages are displayed



VI. Extensive Example



Phase 2



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

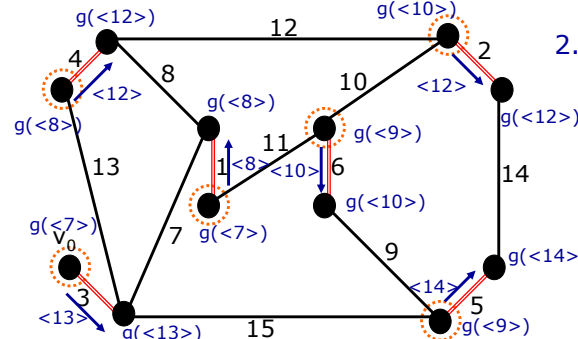
Only some messages are displayed



VI. 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

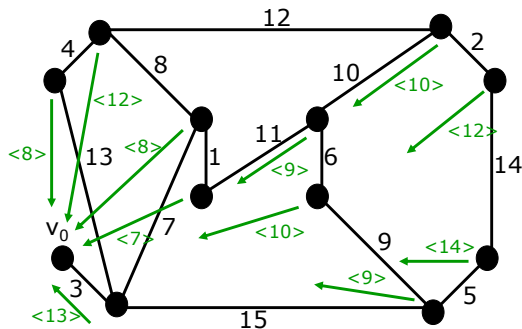
Only some messages are displayed



VI. 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!

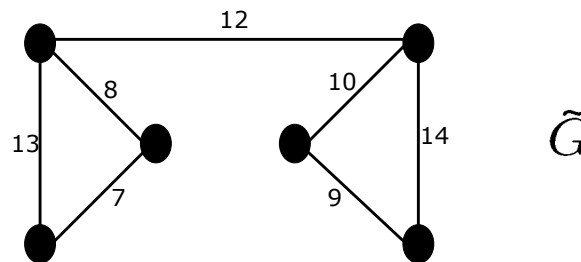


VI. Extensive Example



Phase 2 – Const_Frags

1. Construct logical graph

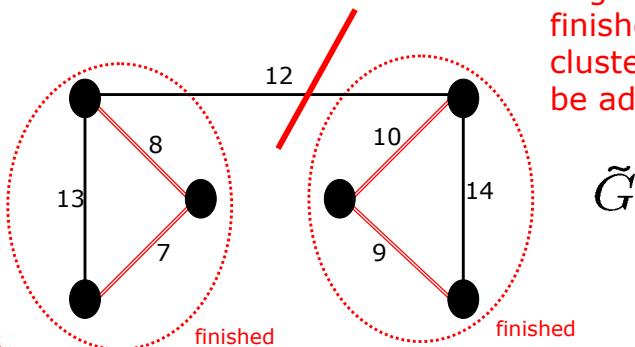


VI. Extensive Example



Phase 2 – Const_Frags

1. Add edges



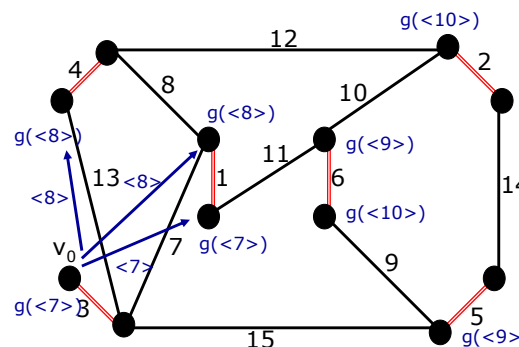
Edges between finished super-clusters must not be added!



VI. 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 $g(e)$

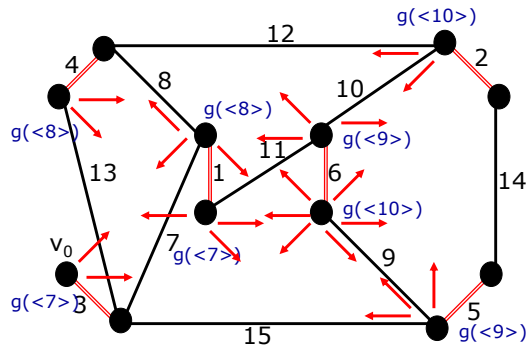
Only some messages are displayed



VI. Extensive Example



Phase 2



Only some messages are displayed

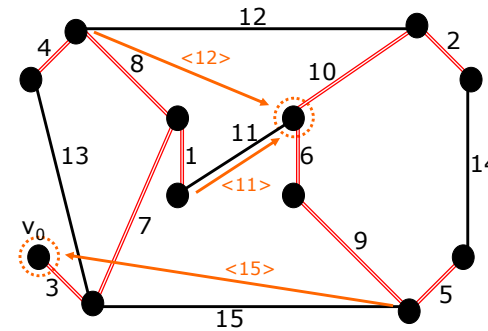
1. Compute $e(v, F')$ and send it to $\ell(F')$
2. Select $\beta = 2$ lightest outgoing edges and appoint guardians
3. Send appointed edge to v_0
4. Const_Frags!
5. Send e to $g(e)$
6. Broadcast e and update the clusters



VI. Extensive Example



Phase 3



Only some messages are displayed

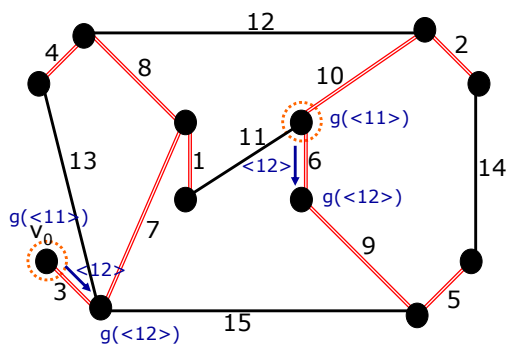
1. Compute $e(v, F')$ and send it to $\ell(F')$



VI. Extensive Example



Phase 3



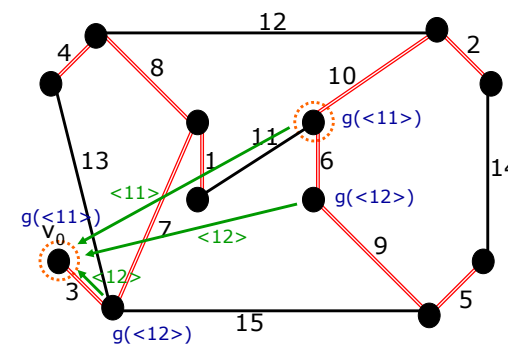
1. Compute $e(v, F')$ and send it to $\ell(F')$
2. Select $\beta = 2$ lightest outgoing edges and appoint guardians



VI. Extensive Example



Phase 3



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!

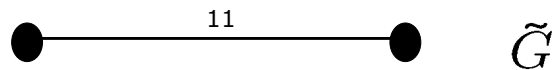


VI. Extensive Example



Phase 3 – Const_Frags

1. Construct logical graph

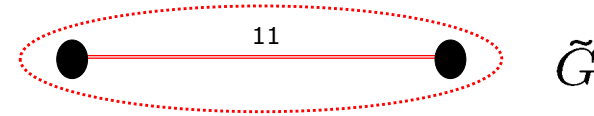


VI. Extensive Example



Phase 3 – Const_Frags

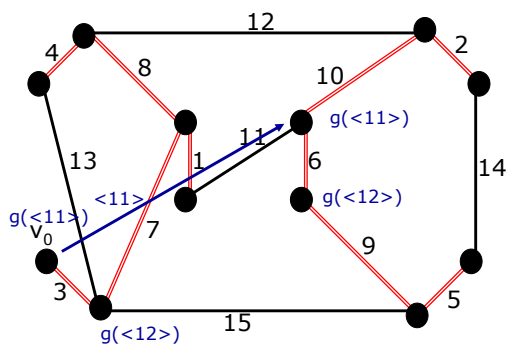
1. Add edges



VI. Extensive Example



Phase 3



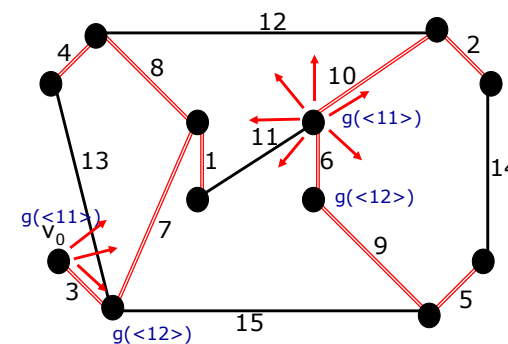
1. Compute $e(v, F')$ and send it to $\ell(F')$
2. Select $\beta = 2$ lightest outgoing edges and appoint guardians
3. Send appointed edge to v_0
4. Const_Frags!
5. Send e to $g(e)$



VI. Extensive Example



Phase 3



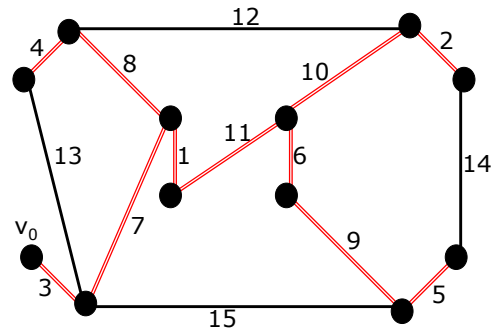
1. Compute $e(v, F')$ and send it to $\ell(F')$
2. Select $\beta = 2$ lightest outgoing edges and appoint guardians
3. Send appointed edge to v_0
4. Const_Frags!
5. Send e to $g(e)$
6. Broadcast e and update the clusters

Only some messages are displayed



VI. Extensive Example

After phase 3



Done! 😊



References

- M. Adler, W. Dittrich, B. Juurlink, M. Kutylowski, and I. Rieping. Communication-Optimal Parallel Minimum Spanning Tree Algorithms. In *Proc. 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 27-36, 1998.
- Z. Lotker, B. Patt-Shamir, and D. Peleg. Distributed MST for Constant Diameter Graphs. In *Proc. 20th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 63-71, 2001.
- Z. Lotker, E. Pavlov, B. Patt-Shamir, and D. Peleg. MST Construction in $O(\log \log n)$ Communication Rounds. In *Proc. 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 94-100, 2003.
- D. Peleg and V. Rubinovich. Near-Tight Lower Bound on the Time Complexity of Distributed MST Construction. *SIAM J. Comput.*, 30:1427-1442, 2000.