Lecture 6

Minimum Spanning Trees

In this lecture, we study another classic graph problem from the distributed point of view: *minimum spanning tree* construction.

Definition 6.1 (Minimum Spanning Tree (MST)). *Given a simple weighted connected graph $G = (V, E, W)$, $W\colon E \to \{1, \dots, n^{\mathcal{O}(1)}\}$, a* minimum spanning tree *$T \subseteq E$ minimizes $\sum_{e \in t} W(e)$ among all spanning trees of G. Here* spanning *means that all nodes are connected by the respective edge set.*

Our goal today is to develop efficient MST algorithms in the Congest model, on an arbitrary simple weighted connected graph G. The Congest model is identical to the synchronous message passing model, but with the additional restriction that $\mathcal{O}(\log n)$ bits are sent in each round along each edge.

Remarks:

- In the Congest model (for a connected graph), essentially every problem can be solved in $\mathcal{O}(|E|)$ rounds by collecting and distributing to all nodes the entire topology, and then having each node solve the problem at hand locally.
- Initially nodes know the weights of their incident edges. In the end, nodes need to know which of their incident edges are in the MST.
- If you're wondering about the weird constraint on the range of the weight function: This is so we can encode an edge weight in a message. Fractional edge weights are fine; we simply multiply all weights by the smallest number that is an integer multiple of the denominators of all fractions (where w.l.o.g. we assume that all fractions are reduced).
- One can handle weights with range $1, \dots, 2^{n^{\mathcal{O}(1)}}$ by rounding up to the next integer power of $1 + \varepsilon$ for $\varepsilon > n^{-\mathcal{O}(1)}$. Then an edge weight can be encoded with

$$\left\lceil \log \log_{1+\varepsilon} 2^{n^{\mathcal{O}(1)}} \right\rceil = \log\left(\frac{n^{\mathcal{O}(1)}}{\log(1+\varepsilon)}\right) \subseteq \mathcal{O}\left(\log n + \log\frac{1}{\varepsilon}\right) = \mathcal{O}(\log n)$$

  bits as well. However, then we will get only $(1+\varepsilon)$-approximate solutions.

6.1 MST is a Global Problem

Recall that in the message passing model without restrictions on message size, an r-round algorithm is some mapping from r-neighborhoods[1] labeled by inputs, identifiers, and, for randomized algorithms, the random strings to outputs. Thus, r puts a bound on the *locality* of a problem: If topology or inputs are locally changed, the outputs of the algorithm change only up to distance r (both in the new and old graph). In contrast, if no r-round algorithm exists for $r \in o(D)$, a problem is *global.*

Theorem 6.2. *MST is a global problem, i.e., any distributed algorithm computing an MST must run for $\Omega(D)$ rounds. This holds even when permitting outputs that are merely spanning subgraphs (i.e., not necessarily trees) and, for any $1 \leq \alpha \in n^{\mathcal{O}(1)}$, α-approximate.*

Proof. Consider the cycle consisting of n nodes, and denote by e_1 and e_2 two edges on opposite sides of the cycle (i.e., in maximal distance from each other). For $1 \leq \alpha \in n^{\mathcal{O}(1)}$, define

$$W_1(e) := \begin{cases} 2\alpha^2 n & \text{if } e = e_1 \\ \alpha n & \text{if } e = e_2 \\ 1 & \textit{else} \end{cases} \qquad W_2(e) := \begin{cases} \alpha n & \text{if } e = e_2 \\ 1 & \textit{else.} \end{cases}$$

On the cycle with weights W_1, the MST consists of all edges but e_1. In fact, *any* solution containing e_1 has approximation ratio at least

$$\frac{W_1(e_1)}{W_2(e_2) + n - 2} > \frac{2\alpha^2 n}{(\alpha+1)n} \geq \alpha.$$

Thus, any α-approximate solution must output all edges but e_1. Likewise, on the cycle with weights W_2, the MST consists of all edges but e_2, and any solution containing e_1 has approximation ratio at least

$$\frac{\alpha n}{n-1} > \alpha.$$

Thus, any α-approximate solution must output all edges but e_2. As by construction the nodes of e_1 and those of e_2 are in distance $\Omega(D) = \Omega(n)$, finding any spanning subgraph that is by at most factor α heavier than an MST is a global problem. □

Remarks:

- The restriction that $\alpha \in n^{\mathcal{O}(1)}$ is only a formality in this theorem. We decided that only polynomial edge weights are permitted in the problem description, so we will be able to *talk* about the weights. But if they are so large that we can't talk about them, this doesn't make our job easier!
- W.l.o.g., we assume in the following that all edge weights are distinct, i.e., $W(e) \neq W(e')$ for $e \neq e'$. This is achieved by attaching the identifiers of the endpoints of e to its weight and use them to break symmetry in case of identical weights. This also means that we can talk of *the* MST of G from now on, as it must be unique.

[1]Here, edges connecting two nodes that are both exactly in distance r are not included.

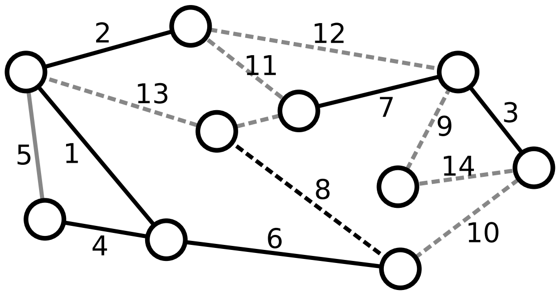

Figure 6.1: Snapshot of Kruskal's algorithm. Solid black edges have been added to T, the solid gray edges have been discarded. The dashed black edge is being processed and will be added to T since it does not close a cycle; the dashed gray edges will be processed in later iterations.

6.2 Being Greedy: Kruskal's Algorithm

Considering greedy strategies is always a good starting point. In the case of an MST, this means to always add the cheapest useful edge first! As closing cycles is pointless, this yields Kruskal's algorithm, compare Figure 6.1:

Algorithm 13 Kruskal's algorithm (centralized version)

```
1: sort $E$ in ascending order of weights; denote the result by $(e_1, \dots, e_{|E|})$
2: $T := \emptyset$
3: for $i = 1, \dots, |E|$ do
4:     if $T \cup \{e_i\}$ is a forest then
5:         $T := T \cup \{e_i\}$
6:     end if
7: end for
8: return $T$
```

Lemma 6.3. *If an edge is heaviest in a cycle, it is not in the MST. If all such edges are deleted, the MST remains. In particular, Kruskal's algorithm computes the MST.*

Proof. Denote the MST by M and suppose for contradiction that $e \in M$ is heaviest in a cycle C. As $C \setminus \{e\}$ connects the endpoints of e, there must be an edge $e' \in C$ so that $(M \setminus \{e\}) \cup \{e'\}$ is a spanning tree (i.e., e' connects the two components of $M \setminus \{e\}$). However, as $W(e') < W(e)$, $(M \setminus \{e\}) \cup \{e'\}$ is lighter than M, contradicting that M is the MST.

Thus, when deleting all edges that are heaviest in some cycle, no MST edge is deleted. As this makes sure that there is no cycle contained in the obtained edge set, the result is a forest. As this forest contains the MST, but a tree is a maximal forest, the forest must in fact be the MST. □

Let's make this into a distributed algorithm! Of course we don't simply collect all edges and execute the algorithm locally. Still, we need to somehow collect the information to compare. We do this, but drop all edges which are certainly "unnecessary" on the fly.

Algorithm 14 Kruskal's algorithm (distributed version)

```
1: compute an (unweighted) BFS, its depth $d$, and $n$; denote the root by $R$
2: for each $v \in V$ in parallel do
3:   $E_v := \{\{v, w\} \in E\}$
4:   $S_v := \emptyset$ // sent edges
5: end for
6: for $i = 1, \ldots, n + d - 2$ do
7:   for each $v \in V \setminus \{R\}$ in parallel do
8:     $e := \text{argmin}_{e' \in E_v \setminus S_v}\{W(e')\}$ // lightest unsent edge
9:     send $(e, W(e))$ to $v$'s parent in the BFS tree
10:    $S_v := S_v \cup \{e\}$
11:  end for
12:  for each $v \in V$ in parallel do
13:    for each received $(e, W(e))$ do
14:      $E_v := E_v \cup \{e\}$ // also remember $W(e)$ for later use
15:    end for
16:    for each cycle $C$ in $E_v$ do
17:      $e := \text{argmax}_{e' \in C}\{W(e')\}$ // heaviest edge in cycle
18:      $E_v := E_v \setminus \{e\}$
19:    end for
20:  end for
21: end for
22: $R$ broadcasts $E_R$ over the BFS tree
23: return $E_R$
```

Intuitively, MST edges will never be deleted, and MST edges can only be "delayed," i.e., stopped from moving towards the root, by other MST edges. However, this is not precisely true: There may be other lighter edges that go first, as the respective senders do not know that they are not MST edges. The correct statement is that for the k^{th}-lightest MST edge, at most $k - 1$ lighter edges can keep it from propagating towards the root. Formalizing this intuition is a bit tricky.

Definition 6.4. *Denote by B_v the subtree of the BFS tree rooted at node $v \in V$, by d_v its depth, by $E_{B_v} := \bigcup_{w \in B_v}\{\{w, u\} \in E_w\}$ the set of edges known to some node in B_v, and by $F_{B_v} \subseteq E_{B_v}$ the lightest maximal forest that is a subset of E_{B_v}.*

Lemma 6.5. *For any node v and any $E' \subseteq E_{B_v}$, the result of Algorithm 13 contains $F_{B_v} \cap E'$.*

Proof. By the same argument as for Lemma 6.3, Kruskal's algorithm deletes exactly all edges from a given edge set that are heaviest in some cycle. F_{B_v} is the set of edges that survive this procedure when it is applied to E_{B_v}, so $F_{B_v} \cap E'$ survives for any $E' \subseteq E_{B_v}$. $\square$

Lemma 6.6. *For any $k \in \{0, 1, \ldots, |F_{B_v}|\}$, after $d_v + k - 1$ rounds of the second FOR loop of the algorithm, the lightest k edges of F_{B_v} are in E_{B_v}, and have been sent to the parent by the end of round $d_v + k$.*

Proof. We prove the claim by double induction over the depth d_v of the subtrees and k. The claim holds trivially true for $d_v = 0$ and all k, as for leaves v we have $E_{B_v} = F_{B_v}$. Now consider $v \in V$ and assume that the claim holds for all $w \in V$ with $d_w < d_v$ and all k. It is trivial for d_v and $k = 0$, so assume it also holds for d_v and some $k \in \{0, \dots, |F_{B_v}| - 1\}$ and consider index $k + 1$.

Because

$$E_{B_v} = \{\{v, w\} \in E_v\} \cup \bigcup_{w \text{ child of } v} E_{B_w},$$

we have that the $(k+1)^{th}$ lightest edge in F_{B_v} is already known to v or it is in E_{B_w} for some child w of v. By the induction hypothesis for index $k + 1$ and $d_w < d_v$, each child w of v has sent the $k + 1$ lightest edges in F_{B_w} to v by the end of round $d_w + k + 1 \leq d_v + k$. The $(k + 1)^{th}$ lightest edge of F_{B_v} must be contained in these sent edges: Otherwise, we can take the sent edges (which are a forest) and edges $k + 1, \dots, |F_{B_v}|$ out of F_{B_v} to either obtain a cycle in which an edge from F_{B_v} is heaviest or a forest with more edges than F_{B_v}, in both cases contained in E_{B_v}. In the former case, this contradicts the fact that F_{B_v} has minimal weight among the maximal forests contained in E_{B_v}, in the latter case, it contradicts the maximality of F_{B_v}. Either way, the assumption that the edge was not sent must be wrong, implying that v learns of it at the latest in round $d_v + k$ and adds it to E_v. By Lemma 6.5, it never deletes the edge from E_v. This shows the first part of the claim.

To show that by the end of the round d_v+k+1, v sends the edge to its parent, we apply the induction hypothesis for d_v and k. It shows that v already sent the k lightest edges from F_{B_v} before round $d_v + k + 1$, and therefore will send the next in round $d_v + k + 1$ (or has already done so), unless there is a lighter unsent edge in E_v. However, such an edge would not close a cycle with the edges sent earlier, so similarly to before, we could use it to either obtain a lighter maximal forest than F_{B_v} or a forest with more edges than F_{B_v}, both contradicting the definition of F_{B_v}. This concludes the induction step and therefore the proof. □

Theorem 6.7. *For a suitable implementation, Algorithm 14 computes the MST M in $\mathcal{O}(n)$ rounds.*

Proof. The algorithm is correct, if at the end of the second FOR loop, it holds that $E_R = M$. To see this, observe that $E_{B_R} = \bigcup_{v \in V}\{\{v, w\} \in E\} = E$ and hence $F_{B_R} = M$. We apply Lemma 6.6 to R and round $n+d-2 = |M|+d_R-1$. The lemma then says that $M \subseteq E_R$. As in each iteration of the FOR loop, all cycles are eliminated from E_R, E_R is a forest. A forest has at most $n - 1$ edges, so indeed $E_R = M$.

Now let us check the time complexity of the algorithm. From Lecture 2, we know that a BFS tree can be computed in $\mathcal{O}(D)$ rounds using messages of size $\mathcal{O}(\log n)$.[2] Computing the depth of the constructed tree and its number of nodes n is trivial using messages of size $\mathcal{O}(\log n)$ and $\mathcal{O}(D)$ rounds. The second FOR loop runs for $n + d - 1 \in n + \mathcal{O}(D)$ rounds. Finally, broadcasting the $n - 1$ edges of M over the BFS tree takes $n + d - 1 \in n + \mathcal{O}(D)$ rounds as well, as in each round, the root can broadcast another edge without causing congestion;

[2]If there is no special node R, we may just pick the one with smallest identifier, start BFS constructions at all nodes concurrently, and let constructions for smaller identifiers stop and "overwrite" those for larger identifiers.

the last edge will have arrived at all leaves in round $n-1+d$, as it is sent by R in round $n-1$. As $D \leq n-1$, all this takes $\mathcal{O}(n+D) = \mathcal{O}(n)$ rounds. $\square$

Remarks:

- A clean and simple algorithm, and running time $\mathcal{O}(n)$ beats the trivial $\mathcal{O}(|E|)$ on most graphs.
- This running time is optimal up to constants on cycles. But (non-trivial) graphs with diameter $\Omega(n)$ are rather rare in practice. We shouldn't stop here!
- Also nice: everyone learns about the entire MST.
- Of course, there's no indication that we can't be faster in case $D \ll n$. We're not asking for everyone to learn the entire MST (which trivially would imply running time $\Omega(n)$ in the worst case), but only for nodes learning about their incident MST edges!
- To hope for better results, we need to make sure that we work concurrently in many places!

6.3 Greedy Mk II: Gallager-Humblet-Spira (GHS)

In Kruskal's algorithm, we dropped all edges that are heaviest in some cycle, because they cannot be in the MST. The remaining edges form the MST. In other words, picking an edge that cannot be heaviest in a cycle is always a correct choice.

Lemma 6.8. *For any $\emptyset \neq U \subset V$,*

$$e_U := \operatorname*{argmin}_{e \in (U \times V \setminus U) \cap E} \{W(e)\}$$

is in the MST.

Proof. As G is connected, $(U \times V \setminus U) \cap E \neq \emptyset$. Consider any cycle $C \ni e_U$. Denoting $e_U = \{v, w\}$, $C \setminus \{e_U\}$ connects v and w. As $e_U \in U \times V \setminus U$, it follows that $(C \setminus \{e_U\}) \cap (U \times V \setminus U) \neq \emptyset$, i.e., there is another edge $e \in C$ between a node in U and a node in $V \setminus U$ besides e_U. By definition of e_U, $W(e_U) < W(e)$. As $C \ni e_U$ was arbitrary, we conclude that there is no cycle C in which e_U is the heaviest edge. Therefore, e_U is in the MST by Lemma 6.3. $\square$

This observation leads to *another* canonical greedy algorithm for finding an MST. You may know the centralized version, Prim's algorithm. Let's state the distributed version right away.

Admittedly, this is a very high-level description, but it is much easier to understand the idea of the algorithm this way. It also makes proving correctness very simple, as we don't have to show that the implementations of the individual steps work correctly (yet). We call an iteration of the REPEAT statement a *phase*.

Algorithm 15 GHS (Gallager–Humblet–Spira)

```
1: $T := \emptyset$ // $T$ will always be a forest
2: **repeat**
3:     $\mathcal{F}$ := set of connectivity components of $T$ (i.e., maximal trees)
4:     Each $F \in \mathcal{F}$ determines the lightest edge leaving $F$ and adds it to $T$
5: **until** $T$ is a spanning subgraph (i.e., there is no outgoing edge)
6: **return** $T$
```

Corollary 6.9. *In each phase of the GHS algorithm, T is a forest consisting of MST edges. In particular, the algorithm returns the MST of G.*

Proof. It suffices to show that T contains only MST edges. Consider any connectivity component $F \in \mathcal{F}$. As the algorithm has not terminated yet, F cannot contain all nodes. Thus, Lemma 6.8 shows that the lightest outgoing edge of F exists and is in the MST. □

Great! But now we have to figure out how to implement this idea efficiently. It's straightforward to see that we won't get into big trouble due to having too

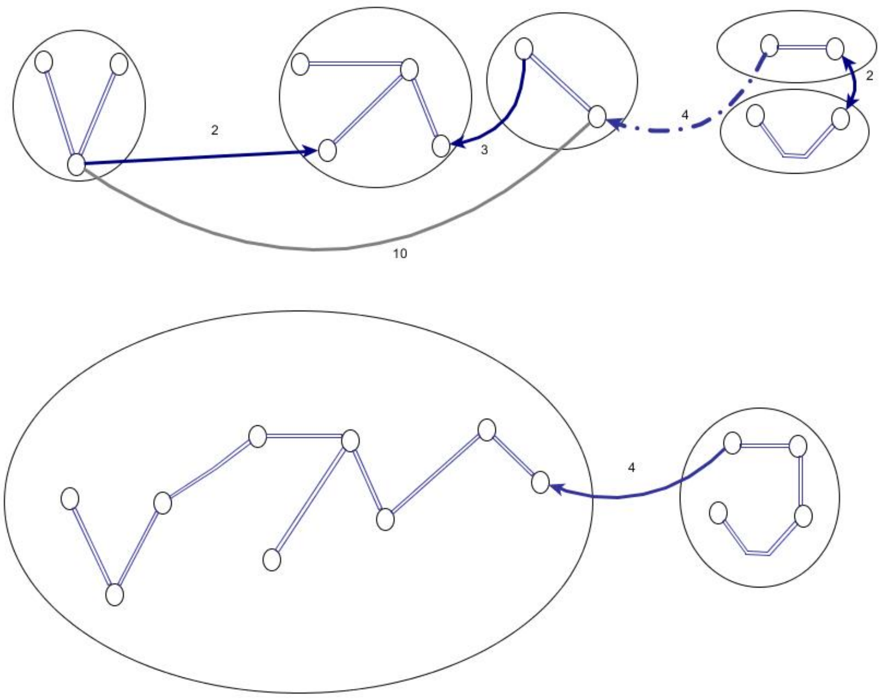

Figure 6.2: Two iterations of the GHS algorithm. The circled areas contain the components at the beginning of the iteration, connected by the already selected MST edges. Each component selects the lightest outgoing edge into the MST (blue solid arrows). Other edges within components, like the grey one after the first iteration, are discarded, as they are heaviest in some cycle.

many phases.

Lemma 6.10. *Algorithm 15 terminates within $\lceil \log n \rceil$ phases.*

Proof. Denote by n_i the number of nodes in the smallest connectivity component (maximal tree) in $\mathcal{F}$ at the end of phase i. We claim that $n_i \geq \min\{2^i, n\}$. Trivially, this number is $n_0 := 1$ "at the end of phase 0," i.e., before phase 1. Hence, it suffices to show that $n_i \geq \min\{2n_{i-1}, n\}$ for each phase i. To this end, consider any $F \in \mathcal{F}$ at the beginning of phase i. Unless F already contains all nodes, it adds its lightest outgoing edge to T, connecting it to at least one other component $F' \in \mathcal{F}$. As $|F| \geq n_{i-1}$, $|F'| \geq n_{i-1}$, and connectivity components are disjoint, $|F \cup F'| \geq 2n_{i-1}$. The claim follows. □

Ok, so what about the phases? We need to do everything concurrently, so we cannot just route all communication over a single BFS tree without potentially causing a lot of "congestion." Let's use the already selected edges instead!

Lemma 6.11. *For a given phase i, denote by D_i the maximum diameter of any $F \in \mathcal{F}$ at the beginning of the phase, i.e., after the new edges have been added to T. Then phase i can be implemented in $\mathcal{O}(D_i)$ rounds.*

Proof. All communication happens on edges of T that are selected in or before phase i. Consider a connectivity component of T at the beginning of phase i. By Corollary 6.9, it is a tree. We root each such tree at the node with smallest identifier and let each node of the tree learn this identifier, which takes $\mathcal{O}(d)$ time for a tree of depth d. Clearly, $d \leq D_i$. Then each node learns the identifiers of the roots of all its neighbors' trees (one round). On each tree, now the lightest outgoing edge can be determined by every node sending their lightest edge leaving its tree (alongside its weight) to the root; each node only forwards the lightest edge it knows about to the root. Completing this process and announcing the selected edges to their endpoints takes $\mathcal{O}(D_i)$ rounds. As the communication was handled on each tree separately without using external edges (except for exchanging the root identifiers with neighbors and "marking" newly selected edges), all this requires messages of size $\mathcal{O}(\log n)$ only. □

We've got all the pieces to complete the analysis of the GHS algorithm.

Theorem 6.12. *Algorithm 15 computes the MST. It can be implemented in $\mathcal{O}(n \log n)$ rounds.*

Proof. Correctness was shown in Corollary 6.9. As trivially $D_i \leq n - 1$ for all phases i (a connectivity component cannot have larger diameter than the number of its nodes), by Lemma 6.3 each phase can be completed in $\mathcal{O}(n)$ rounds. This can be detected and made known to all nodes within $\mathcal{O}(D) \subseteq \mathcal{O}(n)$ rounds using a BFS tree. By Lemma 6.10, there are $\mathcal{O}(\log n)$ phases. □

Remarks:

- The $\log n$ factor can be shaved off.
- The original GHS algorithm is asynchronous and has a message complexity of $\mathcal{O}(|E| \log n)$, which can be improved to $\mathcal{O}(|E| + n \log n)$. It was celebrated for that, as this is way better than what comes from the use of an α-synchronizer. Basically, the constructed tree is used like a β-synchronizer to coordinate actions within connectivity components, and only the exchange of component identifiers is costly.
- The GHS algorithm can be applied in different ways. GHS for instance solves leader election in general graphs: once the tree is constructed, finding the minimum identifier using few messages is a piece of cake!

6.4 Greedy Mk III: Garay-Kutten-Peleg (GKP)

We now have two different greedy algorithms that run in $\mathcal{O}(n)$ rounds. However, they do so for quite different reasons: The distributed version of Kruskal's algorithm basically reduces the number of components by 1 per round (up to an additive D), whereas the GHS algorithm cuts the number of remaining components down by a factor of 2 in each phase. The problem with the former is that initially there are n components, the problem with the latter is that components of large diameter take a long time to handle.

If we could use the GHS algorithm to reduce the number of components to something small, say $\sqrt{n}$, quickly without letting them get too big, maybe we can then finish the MST computation by Kruskal's approach? Sounds good, except that it may happen that, in a single iteration of GHS, a huge component appears. We then wouldn't know that it is so large without constructing a tree on it or collecting the new edges somewhere, but both could take too long! The solution is to be more conservative with merges and apply a symmetry breaking mechanism: GKP grows components GHS-like until they reach a size of $\sqrt{n}$. Every node learns its component identifier (i.e., the smallest ID in the component), and GKP then joins them using the pipelined MST construction (where nodes communicate edges between connected components instead of all incident edges).

Let's start with correctness.

Lemma 6.13. *Algorithm 16 adds only MST edges to T. It outputs the MST.*

Proof. By Lemma 6.8, only MST edges are added to C in any iteration of the FOR loop, so at the end of the looop T is a subforest of the MST. The remaining MST edges thus must be between components, so contracting components and deleting loops does not delete any MST edges. The remaining MST edges are now just the MST of the constructed multigraph, and Kruskal's algorithm works fine on (loop-free) multigraphs, too. □

Also, we know that the second part will be fast if few components remain.

Corollary 6.14. *Suppose after the FOR loop of Algorithm 16 k components of maximum diameter $D_{\max}$ remain, then the algorithm terminates within $\mathcal{O}(D + D_{\max} + k)$ additional rounds.*

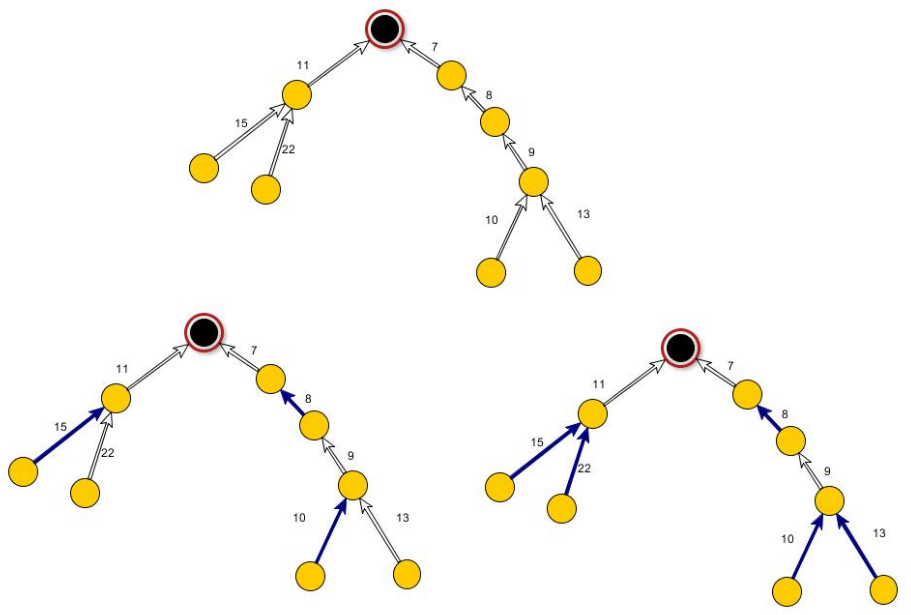

Figure 6.3: Top: A component of $(\mathcal{F}, C)$ in a phase of the first part of the GKP algorithm. "Large" components that marked no MST edge for possible inclusion can only be roots; we have such a root here. Bottom left: The blue edges show a matching constructed using the Cole-Vishkin algorithm. Bottom right: The blue edges are the final set of selected edges in this phase.

Algorithm 16 GKP (Garay–Kutten–Peleg). We slightly abuse notation by interpreting edges of C also as edges between components F. Contracting an edge means to identify its endpoints, where the new node "inherits" the edges from the original nodes.

```
1: $T := \emptyset$ // $T$ will always be a forest
2: for $i = 0, \dots, \lceil \log \sqrt{n} \rceil$ do
3:    $\mathcal{F}$ := set of connectivity components of $T$ (i.e., maximal trees)
4:    Each $F \in \mathcal{F}$ of diameter at most $2^i$ determines the lightest edge leaving
      $F$ and adds it to a candidate set $C$
5:    Add a maximal matching $C_M \subseteq C$ in the graph $(\mathcal{F}, C)$ to $T$
6:    If $F \in \mathcal{F}$ of diameter at most $2^i$ has no incident edge in $C_M$, it adds the
      edge it selected into $C$ to $T$
7: end for
8: denote by $G' = (V, E', W')$ the multigraph obtained from contracting all
   edges of $T$ (deleting loops, keeping multiple edges)
9: run Algorithm 14 on $G'$ and add the respective edges to $T$
10: return $T$
```

Proof. The contracted graph has exactly k nodes and thus $k-1$ MST edges are left. The analysis of Algorithm 14 also applies to multigraphs, so (a suitable implementation) runs for $\mathcal{O}(D+k)$ additional rounds; all that is required is that

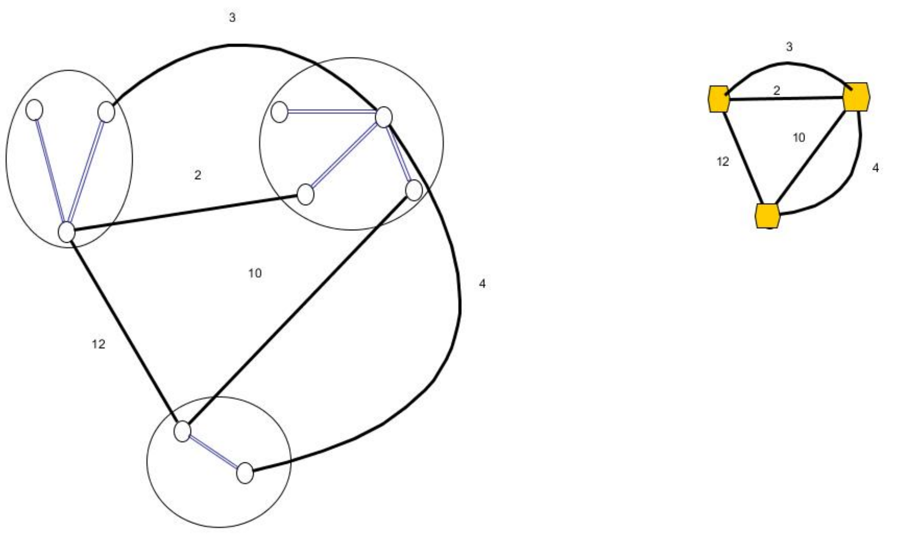

Figure 6.4: Left: Components and their interconnecting edges after the first stage of the GKP algorithm. Right: The multigraph resulting from contraction of the components.

the nodes of each component agree on an identifier for their component. This can be done by figuring out the smallest identifier in each component, which takes $\mathcal{O}(D_{\max})$ rounds. $\square$

It remains to prove three things: (i) components don't become too large during the FOR loop, (ii) we can efficiently implement the iterations in the FOR loop, and (iii) few components remain after the FOR loop. Let's start with (i). We again call one iteration of the FOR loop a phase.

Lemma 6.15. *At the end of phase i, components have diameter $\mathcal{O}(2^i)$.*

Proof. We claim that at the end of phase i, no component has diameter larger than $6 \cdot 2^i \in \mathcal{O}(2^i)$, which we show by induction. Trivially, this holds for $i = 0$ (i.e., at the start of the algorithm). Hence, suppose it holds for all phases $j \leq i \in \mathbb{N}_0$ for some i and consider phase $i + 1$.

Consider the graph $(\mathcal{F}, C)$ from this phase. We claim that each component of $(\mathcal{F}, C)$ has diameter at most 3. To see this, observe that if an unmatched $F \in \mathcal{F}$ adds an edge $\{F, F'\}$, F' must be matched: otherwise, $\{F, F'\}$ could be added to the matching, contradicting its maximality. Thus, all non-matching edges added to T "attach" some $F \in \mathcal{F}$ that was isolated in C to some F' that is not picking a new edge in this step. This increases the diameter of components from at most 1 (for a matching) to at most 3.

Next, we claim that no component contains more than one F of diameter larger than 2^i (with respect to G). This can be seen by directing all selected edges away from the $F \in \mathcal{F}$ that selected it (breaking ties arbitrarily). We obtain a directed graph of outdegree at most 1, in which any F of diameter larger than 2^i has outdegree 0 and is thus the root of a component that is an oriented tree.

Now consider a new component at the end of the phase. It is composed of at most one previous component of – by the induction hypothesis – size at most $6 \cdot 2^i$, while all other previous components have size at most 2^i. The longest possible path between any two nodes in the new component thus crosses the large previous component, up to 3 small previous components, and up to 3 edges between previous components, for a total of $6 \cdot 2^i + 3 \cdot 2^i + 3 \leq 6 \cdot 2^{i+1}$. □

Together with an old friend, we can exploit this to show (ii).

Corollary 6.16. *Each iteration of the FOR loop can be implemented with running time $\mathcal{O}(2^i \log^* n)$.*

Proof. By Lemma 6.15, in phase i components are of size $\mathcal{O}(2^i)$. We can thus root them and determine the edges in C in $\mathcal{O}(2^i)$ rounds. By orienting each edge in C away from the component $F \in \mathcal{F}$ that selected it (breaking ties by identifiers), $(\mathcal{F}, C)$ becomes a directed graph with outdegree 1. We simulate the Cole-Vishkin algorithm on this graph to compute a 3-coloring in $\mathcal{O}(2^i \log^* n)$ rounds. To this end, component F is represented by the root of its spanning tree and we exploit that it suffices to communicate only "in one direction," i.e., it suffices to determine the current color of the "parent." Thus, for each component, only one color each needs to be sent and received, respectively, which can be done with message size $\mathcal{O}(\log n)$ over the edges of the component. The time for one iteration then is $\mathcal{O}(2^i)$. By Theorem 1.7, we need $\mathcal{O}(\log^* n)$ iterations; afterwards, we can select a matching in $\mathcal{O}(2^i)$ time by going over the color classes sequentially (cf. Exercise 1) and complete the phase in additional $\mathcal{O}(2^i)$ rounds, for a total of $\mathcal{O}(2^i \log^* n)$ rounds. □

It remains to show that all this business really yields sufficiently few components.

Lemma 6.17. *After the last phase, at most $\sqrt{n}$ components remain.*

Proof. Observe that in each phase i, each component of diameter smaller than 2^i is connected to at least one other component. We claim that this implies that after phase i, each component contains at least 2^i nodes. This trivially holds for $i = 0$. Now suppose the claim holds for phase $i \in \{0, \ldots, \lceil\sqrt{n}\rceil - 1\}$. Consider a component of fewer than 2^{i+1} nodes at the beginning of phase $i+1$. It will hence add an edge to C and be matched or add this edge to T. Either way, it gets connected to at least one other component. By the hypothesis, both components have at least 2^i nodes, so the resulting component has at least 2^{i+1}.

As there are $\lceil \log \sqrt{n} \rceil$ phases, in the end each component contains at least $2^{\log \sqrt{n}} = \sqrt{n}$ nodes. As components are disjoint, there can be at most $n/\sqrt{n} = \sqrt{n}$ components left. □

Theorem 6.18. *Algorithm 16 computes the MST and can be implemented such that it runs in $\mathcal{O}(\sqrt{n} \log^* n + \mathcal{O}(D))$ rounds.*

Proof. Correctness is shown in Lemma 6.13. Within $\mathcal{O}(D)$ rounds, a BFS can be constructed and n be determined and made known to all nodes. By Corollary 6.16, phase i can be implemented in $\mathcal{O}(2^i)$ rounds, so in total

$$\sum_{i=0}^{\lceil \log \sqrt{n} \rceil} \mathcal{O}(2^i \log^* n) = \mathcal{O}(2^{\log \sqrt{n}} \log^* n) = \mathcal{O}(\sqrt{n} \log^* n)$$

rounds are required. Note that since the time bound for each phase is known to all nodes, there is no need to coordinate when a phase starts; this can be computed from the depth of the BFS tree, n, and the round in which the root of the BFS tree initiates the main part of the computation. By Lemmas 6.15 and 6.17, only $\sqrt{n}$ components of diameter $\mathcal{O}(\sqrt{n})$ remain. Hence, by Corollary 6.14, the algorithm terminates within additional $\mathcal{O}(\sqrt{n} + D)$ rounds. □

Remarks:

- The use of symmetry breaking might come as a big surprise in this algorithm. And it's the only thing keeping it from being greedy all the way!
- Plenty of algorithms in the CONGEST model follow similar ideas. This is no accident: The techniques are fairly generic, and we will see next time that there is an inherent barrier around $\sqrt{n}$, even if D is small!
- The time complexity can be reduced to $\mathcal{O}(\sqrt{n \log^* n} + D)$ by using only $\left\lceil \log(\sqrt{n/\log^* n}) \right\rceil$ phases, i.e., growing components to size $\Theta(\sqrt{n/\log^* n})$.
- Be on your edge when seeing $\mathcal{O}$-notation with multiple parameters. We typically *want* it to mean that no matter how the parameter combination is, the expression is an asymptotic bound where the constants in the $\mathcal{O}$-notation are *independent* of the parameter choice. However, in particular with lower bounds, this can become difficult, as there may be dependencies between parameters, or the constructions may apply only to certain parameter ranges.

What to take Home

- Studying sufficiently generic and general problems like MIS or MST makes sense even without an immediate application in sight. When I first encountered the MST problem, I didn't see the Cole-Vishkin symmetry breaking technique coming!
- If you're looking for an algorithm and don't know where to start, check greedy approaches first. Either you already end up with something nontrivial, or you see where it goes wrong and might be able to fix it!
- For global problems, it's very typical to use global coordination via a BFS tree, and also "pipelining," the technique of collecting and distributing k pieces of information in $\mathcal{O}(D + k)$ rounds using the tree.

Bibliographic Notes

Tarjan [Tar83] coined the terms *red* and *blue edges* for heaviest cycle-closing edges (which are not in the MST) and lightest edges in an edge cut[3] (which are always in the MST), respectively. Kruskal's algorithm [Kru56] and Prim's algorithm [Pri57] are classics, which are based on eliminating red edges and selecting blue edges, respectively. The distributed variant of Kruskal's algorithm shown today was introduced by Garay, Kutten, and Peleg [GKP98]; it was used in the first MST algorithm of running time $\mathcal{O}(o(n)+D)$. The algorithm then was improved to running time $\mathcal{O}(\sqrt{n}\log^* n+D)$ by introducing symmetry breaking to better control the growth of the MST components in the first phase [KP00] by Kutten and Peleg. The variant presented here uses a slightly simpler symmetry breaking mechanism. Algorithm 15 is called "GHS" after Gallager, Humblet, and Spira [GHS83]. The variant presented here is much simpler than the original, mainly because we assumed a synchronous system and did not care about the number of messages (only their size). As a historical note, the same principle was discovered much earlier by Otakar Boruvka and published in 1926 – in Czech (see [NMN01] for an English translation).

Awerbuch improved the GHS algorithm to achieve (asynchronous) time complexity $\mathcal{O}(n)$ at message complexity $\mathcal{O}(|E|+n\log n)$, which is both asymptotically optimal in the worst case [Awe87]. Yet, the time complexity is improved by the GKP algorithm! We know that this is not an issue of asynchrony vs. synchrony, since we can make an asynchronous algorithm synchronous without losing time, using the α-synchronizer. This is not a contradiction, since the "bad" examples have large diameter; the respective lower bound is *existential.* It says that for any algorithm, there *exists* a graph with n nodes for which it must take $\Omega(n)$ time to complete. These graphs all have diameter $\Theta(n)$! A lower bound has only the final word if it, e.g., says that *for all* graphs of diameter D, any algorithm must take $\Omega(D)$ time.[4] Up to details, this can be shown by a slightly more careful reasoning than for Theorem 6.2. We'll take a closer look at the $\sqrt{n\log^* n}$ part of the time bound next week!

Bibliography

[Awe87] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 230–240, New York, NY, USA, 1987. ACM.

[GHS83] R. G. Gallager, P. A. Humblet, and P. M. Spira. Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, January 1983.

[GKP98] Juan A Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, 1998.

[3] An edge cut is the set of edges $(U\times V\setminus U)\cap E$ for some $\emptyset\neq U\subset V$.

[4] Until we start playing with the model, that is.

[KP00] Shay Kutten and David Peleg. Fast Distributed Construction of Small k-Dominating Sets and Applications, 2000.

[Kru56] Jr. Kruskal, Joseph B. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[NMN01] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar Boruvka on Minimum Spanning Tree Problem Translation of Both the 1926 Papers, Comments, History. *Discrete Mathemetics*, 233(1–3):3–36, 2001.

[Pri57] R. C. Prim. Shortest Connection Networks and some Generalizations. *The Bell Systems Technical Journal*, 36(6):1389–1401, 1957.

[Tar83] Robert Endre Tarjan. *Data Structures and Network Algorithms*, chapter 6. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.