Distributed Algorithms, MIT Course 6.852, Fall 2015
Lecture 5: Minimum Spanning Trees Revisited

Stephan Holzer Nancy Lynch
MIT
Cambridge, MA, USA
holzer@mit.edu lynch@csail.mit.edu

09/24/2015

If you have comments or spot mistakes, please email Stephan: holzer@mit.edu. Note that we often use generous bounds in the analysis to explain main techniques fast while not getting the very best bounds. Several algorithms are described in a rather informal way to communicate the high level ideas.

To obtain a faster algorithm for MST than the GHS algorithm from lecture 3, we combine ideas from the $O(n \log n)$ round GHS algorithm (Kruskal style) with the $O(nD)$ round Dijkstra/Prim style algorithm that was presented in lecture 3 as well. While the GHS algorithm is faster than the Dijkstra/Prim algorithm, the key-issue with the GHS algorithm is, that nodes of the same component communicate using edges within their component. As components can have diameter[1] $\Theta(n)$, this determines the runtime.

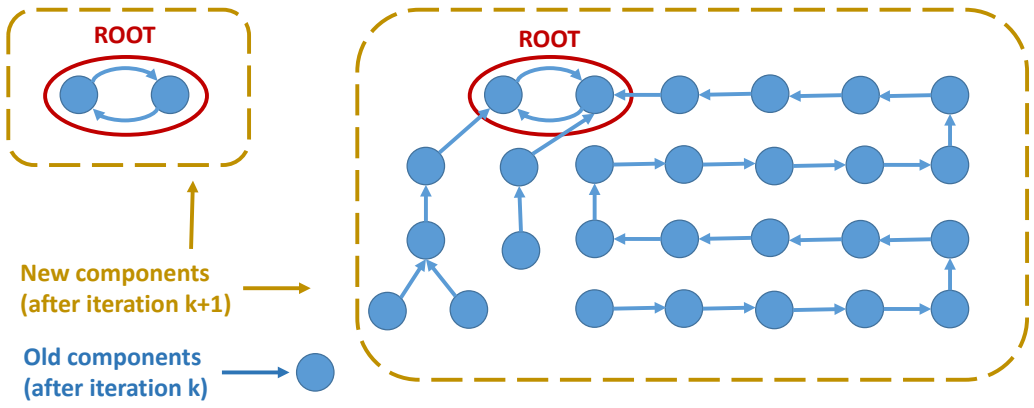

Figure 1: Example of two $(k+1)$-components, one with small and one with large component-diameter. Observe that the $(k+1)$-components obtained during the GHS algorithm in lecture 3 consist of trees of k-components that are connected by their MWOE edges (see lecture 3 for definitions). Note that the root of each tree consists of two k-components that have the same $MWOE$.

[1]The diameter of a component is the largest distance between any two nodes in the component when using only component-edges

We study an algorithm that needs only $\tilde{O}(\sqrt{n}+D)$ many rounds[2] to compute an MST. We derive this algorithm in two steps. In Section 1 we review an algorithm that needs time $\tilde{O}(n^{0.775}+D)$ and refine it in Section 2 to use time $\tilde{O}(\sqrt{n}+D)$.

1 An $\tilde{O}(n^{0.775}+D)$ Algorithm

The key idea in a fist step towards an algorithm that uses $\tilde{O}(n^{0.775}+D)$ rounds is to define two phases in which the algorithm operates. In the first phase it executes a certain number of iterations of a modified version of the GHS algorithm. The modification ensures that the diameter of components stays low, which results in runtime $\tilde{O}(n^{0.775}+D)$ even when using intra-component communication. At the same time we ensure that this phase reduces the number of components to $O(n^{0.775})$. In the second phase we switch from intra-component communication to communication via any edge in G (global communication). While intra-component communication avoids congestion, global communication faces congestion. Fortunately we we are able to bound by the congestion by $O(n^{0.775})$ messages per node per iteration, as there are at most $O(n^{0.775})$ many components left after the first phase and each of these need to determine their $MWOEs$. Using a technique called pipelining, we compute all $O(n^{0.775})$ $MWOEs$ in each iteration in time $O(n^{0.775}+D)$. Doing so in each of $\log n$ iterations, where each iteration merges components, ensures that this second phase ends after $\tilde{O}(n^{0.775}+D)$ rounds as well.

It is important to note that we do not change which edges the algorithm selects for the tree when compared to the GHS algorithm. We only change the round in which an edges is added to the tree.

First phase: In the first phase we execute $\log_{22} n$ iterations[3] of the GHS algorithm and modify the GHS algorithm to split up trees of k-components in each iteration to prevent them from getting too large diameter. In particular, we ensure that k-components have diameter at most 11^k in iteration k. This guarantees that at the end of the first phase the diameter of each component is not too large, such that so far component-internal communication could be performed fast. At the same time we can guarantee that there are at most $n/2^I$ I-components left after this first phase. We achieve this by merging components while controlling the growth of the diameter. We know that in iteration k of the GHS algorithm, a $(k+1)$-component consists of merged k-components that are connected by $MWOEs$ and form an oriented tree of k-components. This tree can have arbitrary diameter. We modify the GHS algorithm to control the diameter growth by splitting each tree of k-components into subtrees of constant diameter before forming $(k+1)$-components out of these subtrees. We do so by computing an MIS on each oriented tree using the Algorithm from Lecture 4, Lemma 3.

To compute the MIS on each tree of k-components, the leaders of each k-component execute the Algorithm from Lecture 4, Lemma 3, and communicate with each other using edges of their respective k-component as well as the $MWOE$ connecting them with neighboring k-components in the tree. Note that there is no congestion, as a leader only needs to receive a message from its parent and send its ID to its children.

After the MIS is computed, each k-component that is not in the MIS joins one of its neighboring k-components that is in the MIS. Subsequently all k-components that are in the MIS and lonely, i.e. they were not joined by another k-component, join a neighboring k-component. The result are $(k+1)$-components in which the each leader of an k-component that is in the MIS and was joined by other k-components become the leader of the respective $(k+1)$-component. Figure 2 illustrates this process. The leader of a $(k+1)$-component is determined in the same way as in the GHS algorithm. Nodes are updated about their new leader in the same way as in the GHS algorithm.

[2]The notation $\tilde{O}$ indicates that polylog factors are hidden in this statement to present a simplified formula that focuses only on important terms. E.g., the precise runtime of the algorithm stated in this lecture is $O(\sqrt{n}\log^2(n)\log^* n+D\log n)$.

[3]In the algorithm and runtime bound we use I iterations and later determine that choosing $I=\log_{22} n$ results in the best runtime.

Second phase: node v participates in computing a BFS-tree T_r rooted in $r = 1$ and in computing the depth d of this tree. While doing this, node v also learns its own distance $d(v, r)$, see Remark 1 for details.

Remark 1. *[Algorithmic idea to compute distances and depth d of T_r] The BFS-tree T_r is computed using a broadcast starting at the root. While involved in computing T_r, node v can learn its distance $d(v, r)$ if distances are piggybacked while computing T_r. The root would simply tell its children that they are at distance 1, then these children tell their children that they are at distance 2 and so on. We can compute the depth d of T_r by asking each node to send its distance $d(r, v)$ back to its parent once it knows its distance. The parent would simply forward the distance to its parent until it reaches the root. Note that in each time step any node v receives the same distance-value forwarded from its children. Thus node v only forwards one distance-value to its own parent such that there is no congestion. Also note that the root receives a larger distance-vale every two time steps for d times. After depth d received values, no further distance-values arrive. Therefore the root knows d once it does not receive a message for 3 consecutive rounds. Then the root can broadcast d.*

This is followed by $O(\log n)$ iterations that merge components based on their $MWOE$ edges. Note that compared to the first phase we do not restrict which k-components are allowed to merge besides being connected by an $MWOE$. At the beginning of each iteration node v knows an index $i(v)$ that indicates to which component $C_{i(v)}$ it belongs and determines its $MWOE(v)$ as described in the GHS algorithm in lecture 3. Each iteration computes the current set $\hat{E} := \{MWOE(T)|T \in F\}$ of $MWOEs$ connecting k-components, where F indicates the set of all k-components. Edges in set $\hat{E}$ are computed using global communication, where the leader of each k-component $C_i \in F$ knows $MWOE(C_i)$ in the end. This means that nodes communicate using any edge in G to compute $MWOE(C_{i(v)})$, not only those of the component $C_{i(v)}$ they currently belong to. The benefit of this is that we can use the (potentially) smaller diameter D of G to achieve a better runtime. To compute $\hat{E}$ we perform for each component C_i a convergecast that aggregates $MWOE(C_i)$ in the same way as the GHS algorithm of lecture 3 with a small modification: In lecture 3 the leader of component C_i served as the root of a BFS tree in C_i to compute $MWOE(C_i)$ via aggregation on this tree. Now node 1 serves as root of a BFS tree of G and the $MWOE(C_i)$ is aggregated using this tree. This step gets more interesting, as we now we perform $n/2^I$ aggregations–one for each remaining component–one after each other. We schedule the $n/2^I$ aggregations in the right way such that they finish in time $n/2^I + D$. This is much faster than time $(n/2^I) \cdot D$ required in a basic approach, where each aggregation starts after the previous one finishes. This technique of performing several aggregations at once with this optimized schedule is called pipelining.

In more detail, each node maintains sets $Q_1, \ldots, Q_{n/2^I} = \emptyset$ that store potential $MWOE(C_i)$-edges and corresponding edge weights. Node v initializes $Q_{i(v)} := \{(MWOE(v), w(MWOE(v)))\}$ at the beginning.

Now node v waits until time t, where all nodes in the network finish the above computations. This point in time t can be computed based on d. Then node v waits $d - d(v, r)$ additional rounds. In case v is an inner node[4], it sends the best candidate for $MWOE(C_i)$ (the minimum weight edge) it knows to its parent in round $t + i$ and receives the best candidates for $MWOE(C_{i+1})$ from its children at the same time. This is done for all (up to $n/2^I$) components $C_1, \ldots, C_{n/2^I}$. The code for the root and leaves differs slightly from the code for inner nodes, as the root has no parent and the leaves have no children. In case a node knows no candidate for $MWOE(C_i)$, it informs its parent about this and the parent adds no element to its set Q_i. In the end the root broadcasts its value of $\hat{E} := \cup_{1 \leq i \leq n/2^I} \{e | w(e) = \min\{w(e) | (e, w(e)) \in Q_i\}\}$ to the network. Based on this, node v updates $i(v)$ using the rules of the GHS algorithm.

As a result, we obtain:

Theorem 1. *Algorithm 1 computes an MST in the CONGEST model in $O(n^{0.775} \log n \log^* n + D)$ rounds of communication.*

[4]Node in a tree that is neither a leave nor the root.

Algorithm 1 Faster MST, as executed by node v

Assume:

- The graph is undirected.
- All edge weights are distinct and can be encoded in $O(\log n)$ bits.
- Each node knows weights of incident edges.
- Processes have UIDs.
- Nodes know (a good upper bound on) n or just compute n.

Required:

- Each process should decide which of its incident edges are in the MST and which are not.

Pseudocode:

//** F denotes the set of components, node v is located in component $C_{i(v)}$

First Phase

for $k = 0, \ldots, I$ **do**
  participate in computing $MWOE(C_{i(v)})$ like in lecture 3 using communication in $C_{i(v)}$;
  //** define $\hat{E} := \{MWOE(T) | T \in F\}$ and $\hat{G} := (F, \hat{E})$, the oriented forest of k-components
  participate in computing MIS M on $\hat{G}$ by simulating the Algorithm of lecture 4, Lemma 3 on $\hat{G}$;
  if v is leader of a k-component T, and $T \in F \setminus M$ **then**
    choose a $\hat{G}$-neighbor T', $T' \in M$;
    merge T and T', inform all nodes in T that the leader of T' is now their leader;
  end if
  if v is leader of k-component T that did not merge in this iteration **then**
    //** We say that these components are *lonely*
    choose a $\hat{G}$-neighbor T', $T' \in M$;
    merge T and T', inform all nodes in T that the leader of T' is now their leader;
  end if
end for

Second Phase

$Q_1, \ldots, Q_{n/2^I} = \emptyset$; //** stores potential $MWOE(C_i)$-edges and weights node v knows of
$Q_{i(v)} := \{(MWOE(v), w(MWOE(v)))\}$;
participate in computing a BFS-tree T_r rooted in r, its depth d and distance $d(v, r)$;
for $\Theta(\log n)$ iterations do **do**
  //** compute $\hat{E} := \{MWOE(T) | T \in F\}$ globally;
  wait $d - d(v, r)$ rounds after network finished above computations; //** can be determined
  //** Code for inner nodes. Root and leaves receive/send slightly differently
  for $i = 1, \ldots, n/2^I$ **do**
    send $(e, w(e))$ s.t. $w(e) = \min\{w(e) | (e, w(e)) \in Q_i\}$ to parent;
    receive $(e, w(e))$ from children, add them to Q_{i+1};
  end for
  participate in root's broadcast of $\hat{E} := \cup_{1 \leq i \leq n/2^I} \{e | w(e) = \min\{w(e) | (e, w(e)) \in Q_i\}\}$;
  update $i(v)$ based on $\hat{E}$;
end for

Output: Any edge incident to v that is used to join components; //** MST edges;

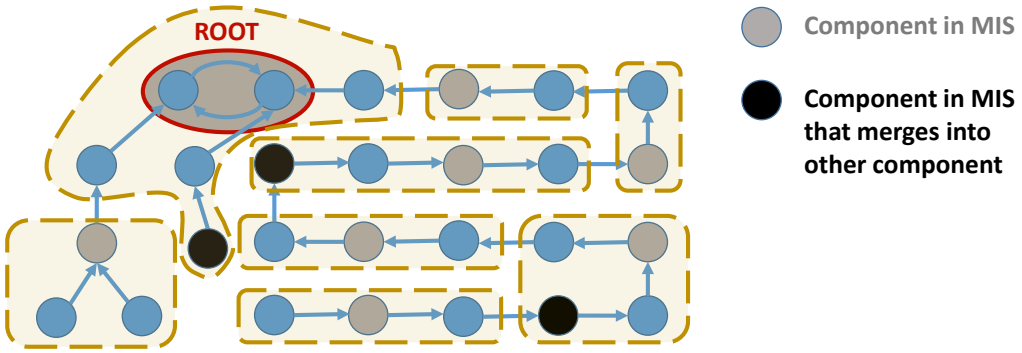

Figure 2: Example of a tree of k-components, that has a large diameter as a result of using GHS – and how this tree is split into smaller diameter trees using the MIS algorithm for oriented trees from lecture 4. Now these small diameter trees are merged to form $(k+1)$-components. Note that for any component C_i we orient $MWOE(C_i)$ to point from C_i to the adjacent component in the tree.

of Theorem 1. Note that this algorithm computes an MST, as the GHS algorithm does and we only change the time at which edges are added to the MST.

We analyze the runtime of the two phases separately. In order to proof time bounds on the first phase of Algorithm 1, we introduce the notion of a (σ, ρ)-component to help describe the size and diameter of components.

Definition 1 ((σ, ρ)-component)**.** *A (σ, ρ)-component contains at least σ nodes and has component-diameter at most ρ.*

We start the proof with stating Lemma 1 that bounds the size and diameter of k-components in iteration k of the first phase.

Lemma 1. *Components after iteration k in the first phase of Algorithm 1 are $(2^k, 11^k)$-components.*

Proof. We prove this lemma by induction on the number of rounds k. Before the algorithm starts (iteration $k = 0$), all components have one element and have component-diameter 0. Thus they are $(2^0, 11^0)$-components. Assume Lemma 1 is true for all iterations up to iteration k. We show that the lemma is true for iteration $k+1$ as well:

- After iteration $k+1$ the component diameter is at most 11^{k+1}: The base of each new $(k+1)$-component is the root, where two k-components merge. The diameter can only increase by merging with k-components that are direct neighbors of the root. Finally, the diameter can only increase once more by merging with lonely k-components that are direct neighbors of any of the already merged k-components. Thus the largest component-diameter occurs when at least the following is true:
  - Two different k-components form the root that is in the MIS. Each of these components is joined by two different k-component that then are joined by two different lonely k-components. Figure 3 displays such a scenario.
  - All k-components mentioned above have maximal component-diameter (at most 11^k due to induction statement).
  - k-components are connected via nodes that are at maximal distance within the k-component.

  In this case the new component-diameter is at most $6 \cdot 11^k + 5 \leq 11^{k+1}$.
- The size of $(k+1)$-components is at least 2^{k+1} after iteration k: Each k-component merges with at least one other k-component and each has size at least 2^k due to the induction statement.

□

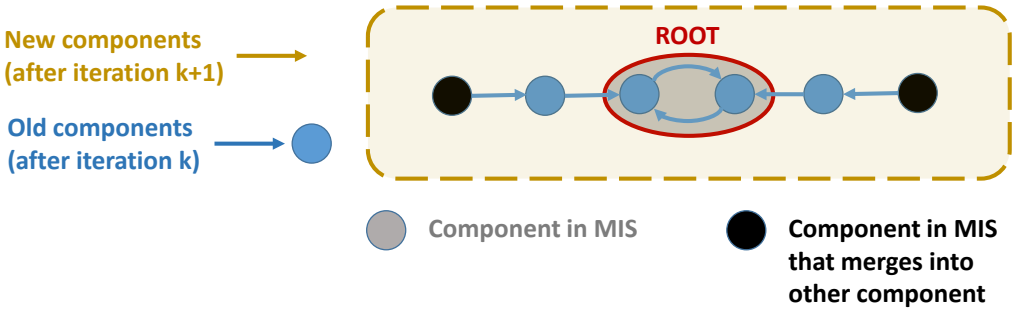

Figure 3: Worst case scenario of component-diameter growth.

Now we can bound the runtime of the first phase by $\sum_{k=1}^{I} O(11^k \cdot (1+\log^* n))$, as the component-diameter in iteration k is at most 11^k and we compute the $MWOE$ for each k-component using communication inside of k-components and then simulate $\log^* n$ rounds of the MIS algorithm from lecture 4, Lemma 3, on oriented trees of k-components using communication inside of k-components, see Figure 2. This is bounded by $O(11^I \cdot \log^* n)$.

Note that after the first phase each I-component contains at least 2^I nodes, such that there are at most $n/2^I$ components left.

The runtime of the second phase is $O((n/2^I + D)\log n)$: Computing T_r and d takes $O(D)$. Next we perform $\log n$ iterations, where a pipelined aggregation takes place in each iteration that takes at most $n/2^I + D$ rounds and the bound follows.

Combining the runtime of the two phases results in a runtime of $O(11^I \cot \log^* n + \frac{n}{2^I}\log n + D\log n)$, which is minimized to be $\tilde{O}(n^{0.775} + D)$ for $I = \log_{22} n$ □

Remark 2. *A more careful analysis yields $\tilde{O}(n^{0.613} + D)$ and is presented in [4].*

2 A $\tilde{O}(\sqrt{n} + D)$ algorithm

Observe that the upper bound diameter of components in Algorithm 1 grows faster than the lower bound on their size (11^k vs. 2^k). We also recall, that the runtime of the first phase mainly depends on the diameter of components and the runtime of the second phase mainly depends on the number of components left after the first phase, which in turn depends on the bound on their minimal size. This implies that one way to improve the runtime is to stop components from getting large diameter or from making sure they grow faster. We consider the first option and carefully chose a point at which we stop components from growing their diameter to fast. In particular we stop components that have diameter at least $\sqrt{n}$ from merging with other components of diameter at least $\sqrt{n}$. Note that components that have diameter at least $\sqrt{n}$ can still merge with smaller components to ensure growth of small diameter component's size.

This modification implies:

Lemma 2. *After the first phase each I-component is a $(\sqrt{n}, 11\sqrt{n}\lceil\log n\rceil)$-component.*

Proof. First we show that the size of I-components is at least $\sqrt{n}$. Let's consider an I-component C and distinguish two cases:

1. Assume that for any iteration k, every k-component that merged into what is now I-component C had diameter less than $\sqrt{n}$. In this case these components merged in the same way as in

Algorithm 2 Fastest MST

High-level modifications of Algorithm 1:

1: Set $I := \lceil \log \sqrt{n} \rceil$ instead of $I := \lceil \log_{22} n \rceil$; //** Can replace $n/2^I$ by $\sqrt{n}$
//** Changes in First Phase:
2: No two components of component-diameter larger than $\sqrt{n}$ are allowed to merge.
//** Changes in Second Phase:
3: Use only sets $Q_1, \ldots, Q_{\sqrt{n}}$;

Algorithm 1 without restriction. From Lemma 1 we conclude that the size of C is at least $2^I \geq \sqrt{n}$.

2. Assume that for some iteration k, there was at least one k-component that merged into what is now I-component C, that had diameter at least $\sqrt{n}$. As this k-component had diameter at least $\sqrt{n}$, it also had size at least $\sqrt{n}$. We conclude that the size of C is at least $2^I \geq \sqrt{n}$, as C contains all nodes from that component.

Now we show by induction on k the diameter of k-components is at most $11k\sqrt{n}$. In the case $k = 0$, i.e. before the first phase starts, each component consist of a single node and has diameter 0. Assume the statement is true up to iteration k, we show it is true in iteration $k+1$. The largest diameter that can occur in a $(k+1)$-component uses the same construction as in the proof of Lemma 1. However, as components that have diameter larger than $\sqrt{n}$ are not allowed to merge, only one component can have diameter (at most) $11k\sqrt{n}$ and the other 5 k-components that merge with it have diameter (at most) $\sqrt{n}-1$ each. Thus the maximal diameter of $(k+1)$-components is $11k\sqrt{n}+5(\sqrt{n}+1)+5 \leq 11(k+1)\sqrt{n}$. The generous bound stated in the lemma follows for iteration $k = I = \lceil \log \sqrt{n} \rceil$. □

Theorem 2. *This deterministic algorithm computes an MST in the CONGEST model in* $O(\sqrt{n} \log^2 n \log^* n + D \log n)$ *rounds of communication.*

Proof. Like before, we only change the time at which an edge is added to the tree, such that the final tree is an MST. It remains to analyze the runtime.

In the first phase we execute $\lceil \log \sqrt{n} \rceil$ iterations. The main cost of each iteration is to simulate the MIS algorithm from lecture 4, Lemma 3, on the tree of k-components. This algorithm takes $O(\log^* n)$ rounds and simulating each round takes up to 2 times the maximal component-diameter. Combined with Lemma 2 this results in a runtime of $O(\sqrt{n} \log^2 n \log^* n)$.

Lemma 2 states that the size if each component is at least $\sqrt{n}$ after the first phase. This implies that there are at most $n/\sqrt{n} = \sqrt{n}$ components left at the start at the second phase. Performing $\lceil \log n \rceil - I$ iterations, where in each iteration $\sqrt{n}$ pipelined aggregations take place, takes time $O(\log(n)(\sqrt{n}+D))$ and the result follows. □

2.1 Bibliographical Notes

Algorithm 1 follows the presentation of [4], while simplifying bounds and stating a tailor-made pipeline procedure. The pipeline procedure described in chapter 5 of [4] is much more general and more involved to state. We use a trick taken from [2] to obtain Algorithm 2. The runtime of Algorithm 2 is optimal up to polylog factors, see e.g. [1, 3].

References

[1] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing (special issue of STOC'11)*, 41(5):1235–1265, 2012.

[2] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998.

[3] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM Journal on Computing*, 30(5):1427–1442, 2001.

[4] David Peleg. Distributed computing. *SIAM Monographs on discrete mathematics and applications*, 5, 2000.