Algorithms and Uncertainty, Winter 2025/26 Lecture 2 (4 pages)

Online Steiner Tree

Thomas Kesselheim Last Update: October 15, 2025

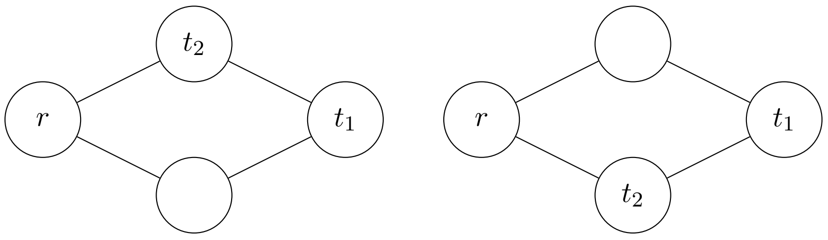
Today, we will consider an online version of the Steiner tree problem. The problem will also reappear later this semester in a different setting of uncertainty. All of these use similar techniques, which we will introduce today.

1 Problem Statement

Let us first state the classical offline Steiner tree problem. We state it here in a rooted variant because this makes our life easier in settings with uncertainty. We are given a connected graph $G = (V, E)$, edge weights $w(e) \geq 0$ for $e \in E$, a root $r \in V$, and a set of terminals $T \subseteq V$. Our task is to select a subset of the edges $S \subseteq E$ such that $\{r\} \cup T$ is connected in $G' = (V, S)$ and $\sum_{e \in S} w(e)$ is minimized. Observe that if $T = V$ then this problem is exactly the minimum spanning tree problem. It is an NP-hard problem.

In the online version, we only get to know the terminals in the set T one after the other. That is, $T = \{t_1, \ldots, t_k\}$ and in the i-th step, t_i is revealed. At any point in time, we are maintaining a set S such that S is a Steiner tree on $\{r\} \cup T$. We may only add but not remove edges from S. In other words, immediately when terminal t_i is revealed, we have to add edges to S such that $\{r, t_1, \ldots, t_i\}$ is connected in $G' = (V, S)$.

Example 2.1. *Consider the following graph, in which all edge weights are 1. The first terminal to appear is always the right node.*



The optimal offline solution always consist of two edges, namely connecting r and t_1 via t_2.

The difficulty is that in the online setting is as follows. The first terminal has to be connected, either via the top or the bottom path. Both choice seem equally good at this point because we do not know which of the two remaining nodes will become t_2 at this point.

For this reason, we can observe that any deterministic algorithm will always incur overall cost at least 3 in one of the two cases. This means that no deterministic algorithm is α-competitive for $\alpha < \frac{3}{2}$. Later, we will generalize this lower-bound construction.

2 Simplifications

Observe that any edge $e = \{u, v\}$ in the Steiner tree can also be replaced by a path $u, x_1, \ldots, x_\ell, v$. This keeps the solution feasible. Indeed, we should never add an edge to the Steiner tree if the sum of edge weights on such a path is cheaper than the direct connection.

To simplify our life, we make this change implicit in the following. We assume that $G = (V, E)$ is a complete graph and we assume that the weights $w(e)$ fulfill the triangle inequality.

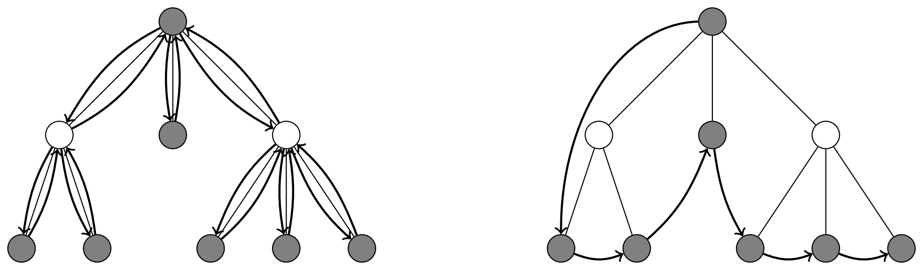


Figure 1: The idea of the proof of Lemma 2.2: Traverse the Steiner tree, then leave out Steiner vertices (white) and duplicate vertices.

That is, $w(\{u,v\}) \leq w(\{u,x\}) + w(\{x,v\})$ for all $u, v, x \in V$. Both assumptions are without loss of generality. In both cases, instead of using the (non-existing or more expensive) direct connection from u to v, we could take a shortest path from u to v.

3 Steiner Trees and Spanning Trees

As already mentioned, the spanning-tree problem is exactly the special case of the Steiner-tree problem in which all vertices are terminals. Indeed, Steiner trees can be approximated by minimum spanning trees. Such a spanning tree only uses edges between the nodes in the set $\{r\} \cup T$ and no edges to other vertices (called Steiner vertices). Let $\mathrm{MST}(T) \subseteq E$ be the minimum spanning tree on $G|_{\{r\}\cup T}$ and let $\mathrm{Steiner}(T) \subseteq E$ be the optimal Steiner tree connecting $\{r\} \cup T$.

Lemma 2.2. *A minimum spanning tree on $G|_{\{r\}\cup T}$ is a 2-approximation for the min-cost Steiner tree on $\{r\} \cup T$, formally*

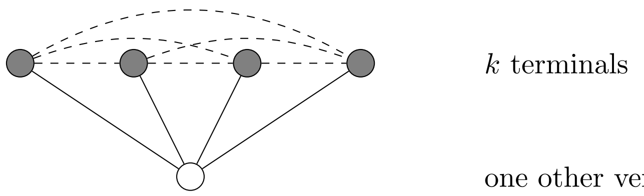
$$w(\mathrm{MST}(T)) \leq 2 \cdot w(\mathrm{Steiner}(T))$$

Proof Idea. Traverse the optimal Steiner tree in a depth-first-search manner. You cross each edge twice: Once when entering the subtree and once when exiting it again. Equivalently, you can double each edge in the tree and consider an Euler tour through these duplicated tree edges. As each edge is crossed twice, the sum of edge costs on this run is $2 \cdot w(\mathrm{Steiner}(T))$.

We get a sequence of vertices that contains r and each terminal from T at least once. Consider the path that shortcuts this sequence by only visiting r and the vertices in T exactly once. By triangle inequality, this path can only be shorter, so the sum of edge costs is at most $2 \cdot w(\mathrm{Steiner}(T))$.

This path is a spanning tree of $G|_{\{r\}\cup T}$. The minimum spanning tree has at most its cost. □

Note that this bound is tight. Consider the following graph construction. The filled vertices are terminals. The solid edges have weight 1, the dashed ones weight 2.



The minimum spanning tree will only used the dashed edges and therefore have a cost of $2(k-1)$ whereas the optimal Steiner tree will use the solid edges and have a cost of k.

4 Greedy Algorithm for Online Steiner Tree

Our algorithm for Online Steiner Tree is the greedy algorithm that always connects an arriving terminal to the existing Steiner Tree via the cheapest edge. More formally: Initialize $S = \emptyset$. When t_i arrives, let e_i be the cheapest edge between t_i and one of the vertices $r, t_1, \ldots, t_{i-1}$. Add e_i to S.

Theorem 2.3. *Greedy is $O(\log k)$-competitive.*

For every terminal t_i there is a connection cost $w(e_i)$, which is the cost increase in the i-th step. To prove the theorem, we will use the following lemma, which shows that not all of the connection costs that we see can be very high.

Lemma 2.4. *The j-th highest connection cost among $w(e_1), \ldots, w(e_k)$ is at most $\frac{2w(\mathrm{Steiner}(T))}{j}$.*

Proof. Let $T_j \subseteq T$ be the set of terminals of size j for which the connection cost is highest. Our claim is now that $\min_{i \in T_j} w(e_i) \leq \frac{w(\mathrm{MST}(T))}{j}$.

Consider $\mathrm{MST}(T_j)$ and $\mathrm{Steiner}(T_j)$, that is, the minimum spanning tree on the graph induced by $\{r\} \cup T_j$ and the optimal Steiner tree connecting $\{r\} \cup T_j$. By Lemma 2.2, we have that $w(\mathrm{MST}(T_j)) \leq 2w(\mathrm{Steiner}(T_j))$. Furthermore, $w(\mathrm{Steiner}(T_j)) \leq w(\mathrm{Steiner}(T))$ because any set of edges connecting T to the root also connects T_j to the root.

Furthermore, $\mathrm{MST}(T_j)$ contains exactly j edges. Therefore

$$\min_{e \in \mathrm{MST}(T_j)} w(e) \leq \frac{1}{j} \sum_{e \in \mathrm{MST}(T_j)} w(e) = \frac{w(\mathrm{MST}(T_j))}{j} .$$

Let e be the cheapest edge in $\mathrm{MST}(T_j)$. Defining $t_0 = r$, we can write $e = \{t_a, t_b\}$ for $a < b$. At the time t_b arrives, t_a is already a part of the Steiner tree. That is, t_b could be connected via edge e. The Greedy algorithm chooses the cheapest way to connect t_b. Therefore, $w(e_b) \leq w(e)$.

So, overall,

$$\min_{i \in T_j} w(e_i) \leq w(e_b) \leq w(e) \leq \frac{w(\mathrm{MST}(T_j))}{j} \leq \frac{2w(\mathrm{Steiner}(T_j))}{j} \leq \frac{2w(\mathrm{Steiner}(T))}{j} . \qquad \square$$

Proof of Theorem 2.3. Because of Lemma 2.4, we can rewrite and bound the cost of the algorithm as

$$\sum_{i=1}^{k} w(e_i) \leq \sum_{j=1}^{k} \frac{2w(\mathrm{Steiner}(T))}{j} = 2w(\mathrm{Steiner}(T)) \sum_{j=1}^{k} \frac{1}{j} .$$

Note that $\sum_{j=1}^{k} \frac{1}{j}$ is the k-th harmonic number. We can, for example, bound it by

$$\sum_{j=1}^{k} \frac{1}{j} \leq 1 + \int_1^k \frac{1}{x} \mathrm{d}x = 1 + \ln k$$

and hence, the theorem follows. $\square$

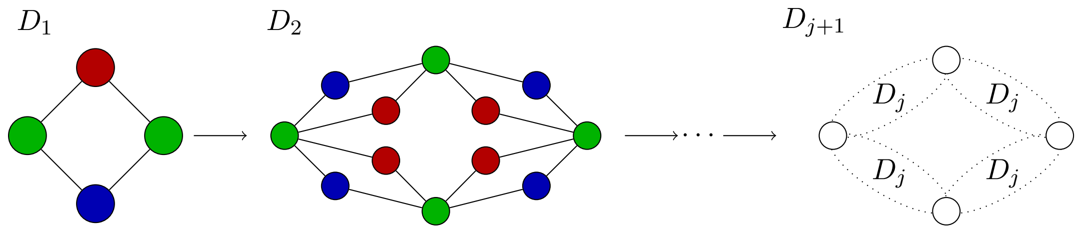
5 Lower Bound

We will now see how Example 2.1 can be generalized to derive a lower bound on the competitive ratio.

Theorem 2.5. *There is no α-competitive deterministic algorithm for $\alpha = o(\log k)$.*

Proof. We consider an arbitrary deterministic algorithm. Without loss of generality, we assume that whenever it picks edges it only chooses a simple path which does not contain any terminals except its endpoint. This is without loss because removing any further edges from the algorithm's choice will only decrease its cost.

Now, we construct a graph and a sequence of terminals on this graph. We only consider the case that k is a power of 2 and we let $\ell = \log_2 k$. We define the *diamond graph* D_j of level j recursively as follows. Let D_1 be the graph from Example 2.1. Now, D_{j+1} is defined by connecting 4 copies of D_j as depicted in the following picture.



Alternatively, we can view D_{j+1} as being derived from D_j by replacing each edge by a copy of D_1. Note that there are exactly 2^j edges on every path from the left to the right node. So in D_ℓ, these are $2^\ell = k$.

We will now define a sequence of terminals in D_ℓ. The root is the left node in D_ℓ. The sequence of terminals will consist of $\ell + 1$ phases. Phase 0 requests the right node in D_ℓ. The remaining phases $i \in \{1, \dots, \ell\}$ request 2^{i-1} terminals each.

For the following phases, we maintain the following invariant: At the beginning of phase $i \geq 1$, there are 2^{i-1} induced subgraphs $D^{(1)}_{\ell-i+1}, \dots, D^{(2^{i-1})}_{\ell-i+1}$ with the property that there are no terminals except for the left and the right node.

To construct phase i, we can make use of the fact that the algorithm is deterministic. This lets us anticipate all choices of the algorithm beforehand. In each of the graphs $D^{(j)}_{\ell-i+1}$, the algorithm can only have picked edges in either the top half or the bottom half but not both. In our sequence, we now add as a terminal the central node on the side that did not contain any edges so far. Left and right of it we find an induced subgraph isomorphic to $D_{\ell-i}$, which maintains our invariant.

Taking the alternative perspective, in which D_{j+1} is obtained from D_j by replacing edges by copies of D_1, this choice of terminal can be understood as follows. For $\ell = 1$, we consider one diamond graph D_1 where the left and right node are terminals. Anticipating the algorithm's choice to be without loss of generality the upper path, we set the next terminal to be the lower vertex. For $\ell = 2$, the edges of the previous graphs are themselves again replaced by new diamonds D_1. So, the algorithm also has to pick a path through them, which again we can anticipate. For higher levels of ℓ, this procedure continues, always replacing edges by D_1 graphs.

Note that by this construction, all terminals will lie on a single path from the left to the right node. Therefore, the offline optimum is always $c(\mathrm{OPT}(\sigma)) = 2^\ell = k$.

The cost that the algorithm incurs in phase 0 is 2^ℓ. In one step of phase $i \geq 1$, the cost is (at least) $2^{\ell-i}$. So, overall the cost of phase i will be $2^{i-1} \cdot 2^{\ell-i} = 2^{\ell-1}$. In total $c(\mathrm{ALG}(\sigma)) \geq 2^\ell + \ell \cdot 2^{\ell-1}$. So

$$\frac{c(\mathrm{ALG}(\sigma))}{c(\mathrm{OPT}(\sigma))} \geq 1 + \frac{\ell}{2} = 1 + \frac{\log_2 k}{2} \ .$$

□