GREEDY ALGORITHMS, PART 3: MINIMUM SPANNING TREES

Marcus Ritt

CMP 601 – Algorithms and Theory of Computation —

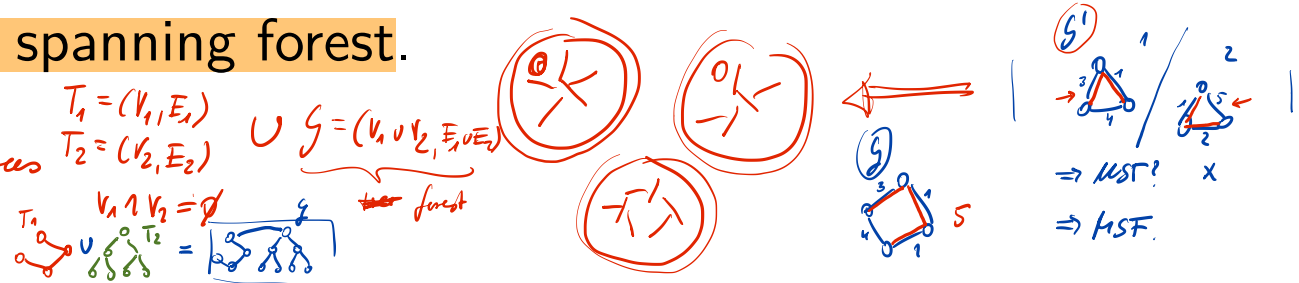
<2020-04-14 ter>

Outline

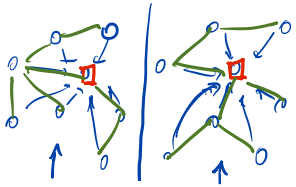
1. Minimum spanning trees

The problem

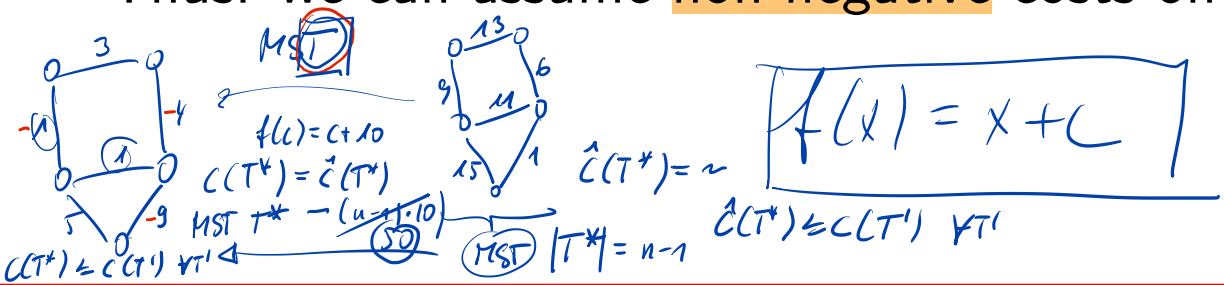
- Undirected graph $G = (V, E, c)$ with costs $c : E \to \mathbb{R}_+$
- Definition: A spanning subgraph is one that connects all vertices.
- *Want*: Minimum cost spanning subgraph.
- *Observation*: This must be a tree.
  Suppose we have a cycle: we can remove an edge, and the graph stays connected. We end up with a connected, cycle-free graph, by definition a tree.
- Thus: we can focus on minimum spanning trees.
- If the graph is connected: application to all components yields a minimum spanning forest.



Related problems



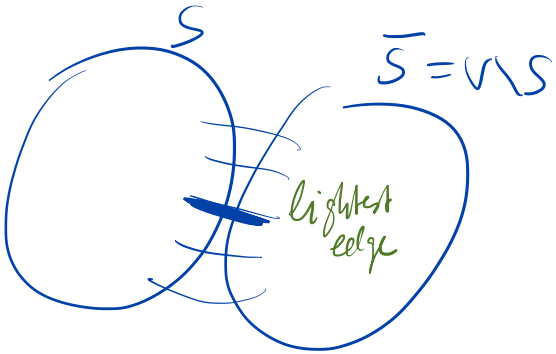
- *Observation*: Reduction to a tree fails when we have negative costs.
- For negative costs the minimum cost spanning subgraphs could contain all edges!
- On the other hand: when we require trees, we have exactly $n-1$ edges, and every monotome transformation of the costs maintains minimality.
- In particular: linear transformations, and even more particular: addition of a constant.
- Thus: we can assume non-negative costs on trees.



Additional assumptions

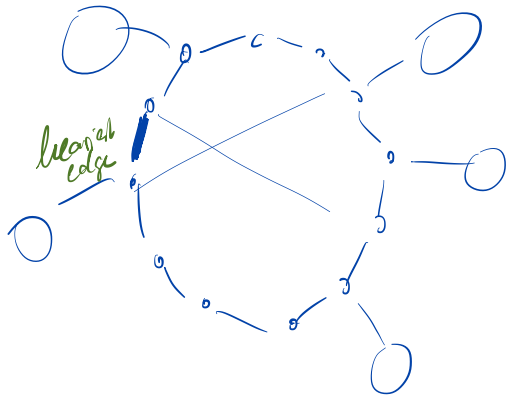
- We can assume that all edge costs are different.
- All algorithms will work when edge costs are the same.
- Argument: by perturbation of the costs.

Properties



Cut property For every cut $(S, \bar{S})$: the lightest cut edge is part of the MST

Cycle property For every cycle C: the heaviest cycle edge is not part of the MST



Main ideas

- Grow a component S (Prim, Jarník)
  - Starting from $S = \{s\}$, repeatedly add lightest $(S, \bar{S})$-edge
  - Using a priority queue: $O(n \log n + m)$
- Merge components (Kruskal)
  - Starting from sets $\{v\}$ for all $v \in V$, process edges in ~~non-decreasing~~ increasing cost, add edges that connect components
  - Using a union/find data structure: $O(m \log n)$.
- Parallel merge (Borůvka)
  - Starting from sets $\{v\}$ for all $v \in V$, repeatedly find lightest cut edges for all components, add all edges
  - Takes $\log_2 n$ iterations, (number of components is at least halved) in $O(m)$.
- Reverse delete (Kruskal)
  - Process edges in ~~non-increasing~~ decreasing cost, remove edges that do not disconnect the graph.
  - Check can be done in $O(\log n (\log \log n)^3)$ (Thorup 2000).