Linear time MST, Directed MST

Today we see a randomized $O(m+n)$ time MST algorithm by Karger–Klein–Tarjan (1995), called KKT.

Definition: Suppose $F \subseteq G$ is a forest. An edge $e \in E$ is F-heavy if e creates a cycle in $F \cup \{e\}$ and it is the heaviest edge in that cycle. Otherwise e is F-light

observation:

i) e is F-light $\iff e \in MST(F \cup \{e\})$
ii) if T is an MST then e is T-light $\iff e \in T$
iii) For any forest F, the F-light edges contain MST of G.
i.e. for any F-heay edge e, $MST(G-e) = MST(G)$.

- How to use this? if F is a forest, we can discard F-heavy edges (from G). Goal: find a forest with many F-heavy edges (so F is close to an MST!) Then we can recurse on the remaining edges.

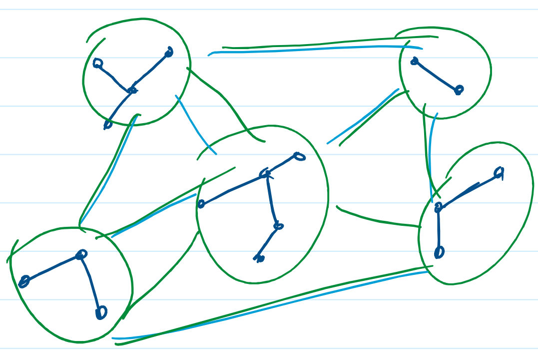
issues: 1) how to find good forest F?
2) how to quickly determine/classify F-heavy edges?

- Theorem ⊛: Given a forest $F \subseteq G$, there is an algorithm that outputs all F-heavy (or F-light) edges in $O(m+n)$.
(Komlós 85, Dixon/Rauch/Tarjan 92, King 97)

- It is actually used in study of fast MST verification.
- For now lets assume this theorem.
- The idea of KKT is: randomly choose half of the edges and find a min-spanning forest F over them. Find the F-heavy edges & discard them; recurse on the remaining graph.

KKG-Algorithm (G)

1) Run 3 rounds of Borůvka's algorithm on G, Contract the edges to find $G' = (V', E')$ with $|V'| = n' \le n/8$ and $|E'| = m' \le m$
2) let E_1 be a random sample of E' w.p. $\frac{1}{2}$
   let $F_1 = KKG(G_1 = (V', E_1))$ (MSF of G_1)
3) let E_2 be all F_1-light edges of E' using ⊛
4) let $F_2 = KKG(G_2 = (V', E_2))$ (i.e. delete F_1-heavy edges & compute MSF of it)
5) return F_2 and edges found in step one.



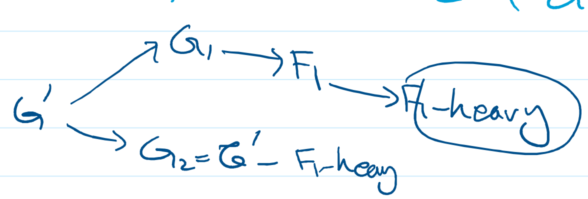
G'

G_1 : random $\frac{1}{2}$ of E'

F_1 : MSF on G_1

delete F_1-heavy from G' to get G_2

find MSF F_2 in G_2



Proof of correctness!

- Proof of Borůvka's algorithm shows edges picked

- Proof of Borůvka's algorithm shows edges picked in the first step belong to MST
- Also clearly (using observation (ii)) the F_i-heavy edges discarded cannot be in MST.

Runtime Analysis

We use the following two claims:

Claim 1: $\mathbb{E}[|E_1|] = \frac{m'}{2}$

Claim 2: $\mathbb{E}[|E_2|] \leq 2(n'-1)$

The proof of claim 1 is trivial (random sample).

We show (assuming) Claims 1&2, KKG terminates in expected time $O(m+n)$.

Assume that the ~~total~~ time for step 1&3 on a graph of order/size m,n takes $c(m+n)$ time.

Define: T_G : expected time of KKG(G)

$$T_{m,n} : \max_{G:\ |V|=n,\ |E|=m} \{T_G\}$$

Then: $T_G \leq c(m+n) + \mathbb{E}[T_{G_1} + T_{G_2}]$

$\leq c(m+n) + T_{m_1,n'} + T_{m_2,n'}$

$m_1 = \mathbb{E}[|E_1|] \leq \frac{m'}{2}$

$m_2 = \mathbb{E}[|E_2|] \leq 2(n'-1)$

We use induction to show $T_{m,n} \leq 2c(m+n)$

$$T_G \le c(m+n) + \mathbb{E}[2c(m_1+n')] + \mathbb{E}[2c(m_2+n')]$$

$$\le c(m+n) + c(m'+2n') + 2c(2n'+n')$$

since $\mathbb{E}[m_1] \le \frac{m'}{2}$, $\mathbb{E}[m_2] \le 2n'$

$$= c(m+n+m'+8n')$$

$$\le 2c(m+n) \qquad \text{since } n' \le \frac{n}{8} \quad m' \le m$$

Proof of claim 2:

- need to show the expected # of F_1-light edges is $\le 2(n-1)$
- Since F_1 is the min-spanning forest of G_1 we can assume (for the sake of this proof) it is obtained by kruskal's algorithm.
- We also assume the process of randomly selecting edges for E_1 is mixed with computing F_1 in the following way.
- Sort all edges of G': $e_1, \dots, e_{m'}$
- when we consider an edge e_i, if it creates a cycle we clearly ignore it; this will be F_1-heavy
- if not then flip a coin and add to F_1 with Probability $1/2$.
- Observation: this process generates the same distribution for F_1 as first choosing G_1 and then running Kruskal's.

- The number of F_i-light edges is bounded by the number of edges we didn't ignore. these are the edges we tossed a coin for.
- Since F_i has $\leq n-1$ edges at the end and Probability of coin toss $= \frac{1}{2}$, the expected # of edges we tossed a coin for (i.e. light) is $\leq 2(n-1)$. ∎

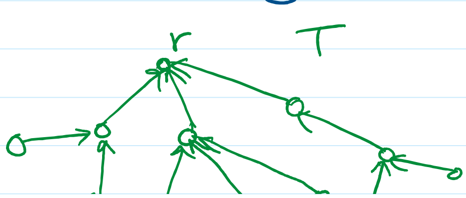
For proof of Theorem ⊛ see references.

Open problem: Find a deterministic linear time MST alg.

Arborescences: Directed Spanning Tree

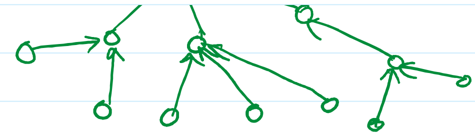
- We will see how a refined greedy algorithm we saw for MST will work for directed MST
- Given a digraph $G(V, A)$ where A is the set of Arcs (directed edges), $w: A \longrightarrow \mathbb{R}^+$, a root $r \in V$.

Definition: An r-arborescence is a subgraph $T=(V, A')$ with $A' \subseteq A$ that is a directed tree rooted at r with each $v \in V$ having a directed path to r.



So ignoring the directions, T

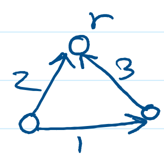
So ignoring the directions, T is a spanning tree.



Question: how to check if G has an r-arborescence?

how to find a min-cost arborescence?

- A greedy like Prim? Fails



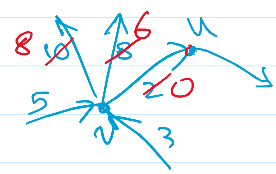
- A greedy like Kruskal? Fails

(can you build an example?)

Chu-Liu/Edmonds/Bock Algorithm 1967

$\delta^+(v)/\delta^-(v)/\delta^+(S)$: Arcs going out / into node v; or set S

- We reduce weights on out-going arcs s.t. for each v, at least one of them is 0



$M_G(v)$: weight of min-arc going out of v

- $\forall\ v \in G$ and outgoing arc a: $w'(a) \leftarrow w(a) - M_G(v)$

Call this new graph G'

lemma 1: T is a min-cost arborescence in G

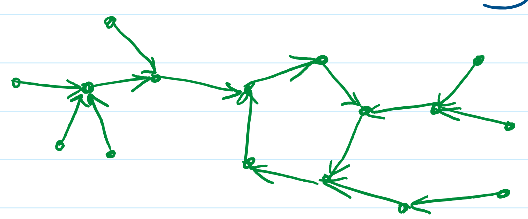
$\Longleftrightarrow$ it is one in G'

Proof: in each arborescence each vertex has one outgoing edge. So decreasing all outgoing edges of each node by $M_G(v)$ does not change opt. ∎

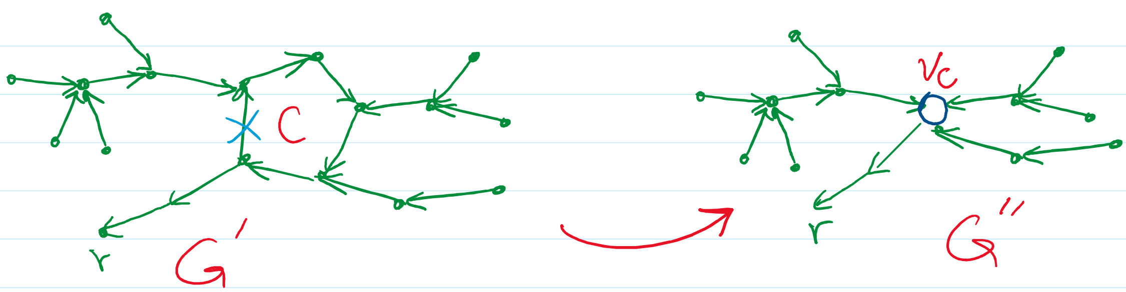
Idea of the algorithm:

- For each $v \in G$, pick one 0-weight outgoing edge.

- for each $v \in G$, pick one 0-weight outgoing edge.
- If we find an arborescence it is a minimum.
- Else, each component consists of a cycle and some smaller arborescences with roots being nodes of cycle:



- Contract this 0-weight cycle C into a single node and keep cheapest edge among parallel edges; let the new graph be $G'' = G'/C$



lemma 2: $opt(G'') = opt(G')$.

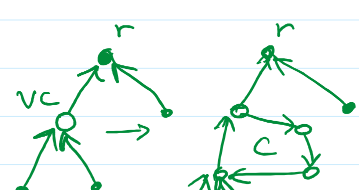
Proof: ▶ first show $opt(G') \le opt(G'')$

let T'' a min arborescence for G''. Consider T' obtained from T'' by expanding v_c (contracted node) to C and removing any of edges of C.

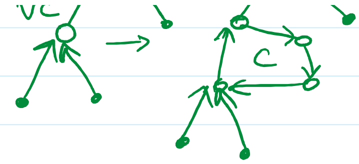
Since C is a 0-weight cycle: $w(T') = w(T'')$

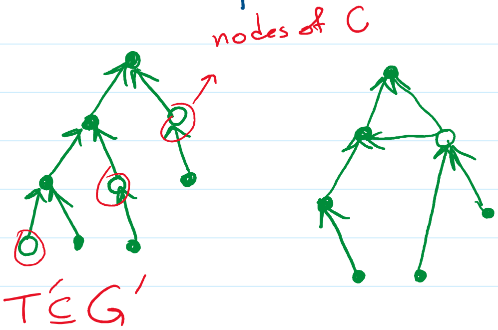
▶ to show $opt(G'') \le opt(G')$

Suppose T' is a solution for



Suppose T' is a solution for G'. Identify nodes of C into a single node. We still have a dipath from each node to r. We can remove some edges to find an arborescence T'' for G''; cost can only go down.





- We can iteratively run this:

Minimum Arborescence algorithm

- Obtain G' by reducing outgoing weights for each $v \in V$ by $M_G(v)$.
- Take an arbitrary 0-weight edge of each v
- Contract 0-weight cycles to get G''
- recurse on G''; let T'' be the solution
- Add back edges of the cycles (except one) to expand T'' to a solution T' for G'.
- return T'.

Time complexity:

- each round # of vertices reduces → $O(n)$ iter's

- Weight reduction, Contraction, and lifting: $O(m)$
- Total time $O(mn)$

Improvements: Best time known $O(m \log\log n)$

Open Question: Is there a linear time for this?

We will see later that one can use linear programming for minimum arborescence too.