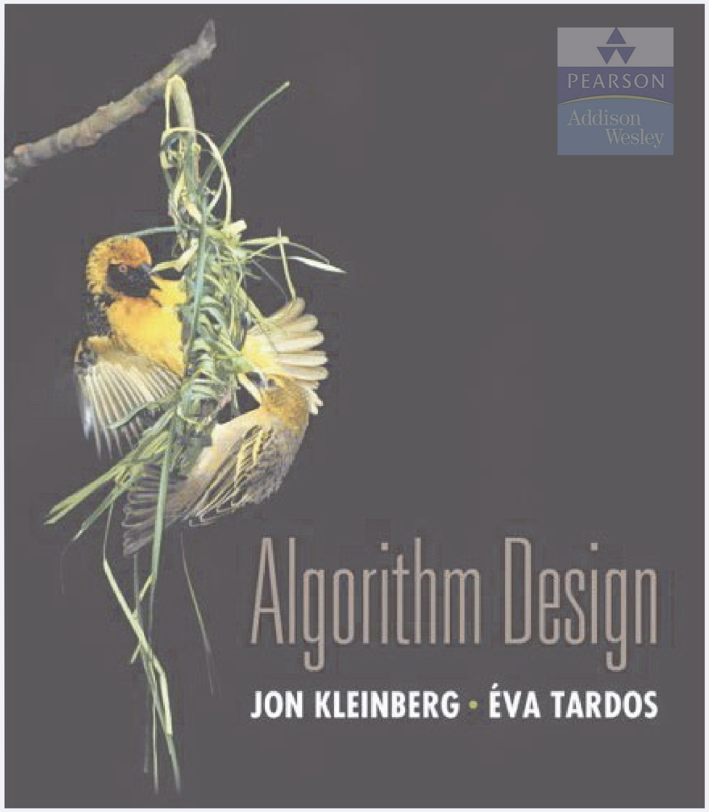


SECTION 4.9

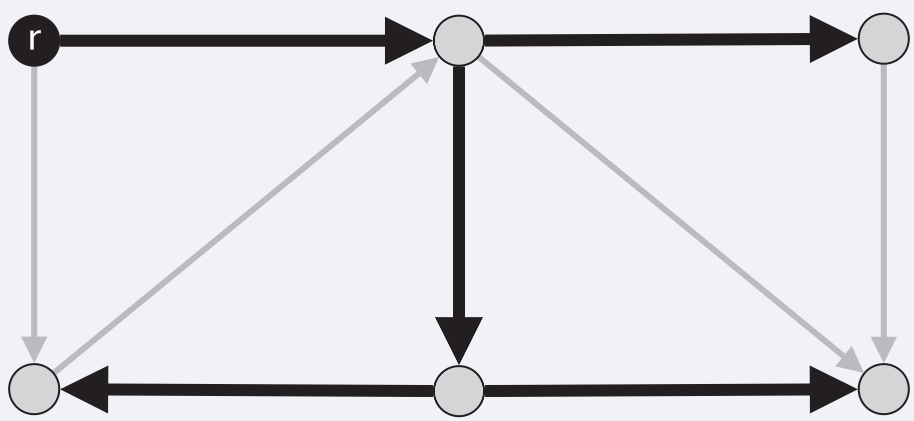
4. GREEDY ALGORITHMS II

- Dijkstra's algorithm
- minimum spanning trees
- Prim, Kruskal, Boruvka
- single-link clustering
- min-cost arborescences

Arborescences

Def. Given a digraph $G = (V, E)$ and a root $r \in V$, an arborescence (rooted at r) is a subgraph $T = (V, F)$ such that

- T is a spanning tree of G if we ignore the direction of edges.
- There is a directed path in T from r to each other node $v \in V$.



Warmup. Given a digraph G, find an arborescence rooted at r (if one exists).

Algorithm. BFS or DFS from r is an arborescence (iff all nodes reachable).

Arborescences

Def. Given a digraph $G = (V, E)$ and a root $r \in V$, an arborescence (rooted at r) is a subgraph $T = (V, F)$ such that

- T is a spanning tree of G if we ignore the direction of edges.
- There is a directed path in T from r to each other node $v \in V$.

Proposition. A subgraph $T = (V, F)$ of G is an arborescence rooted at r iff T has no directed cycles and each node $v \neq r$ has exactly one entering edge.

Pf.

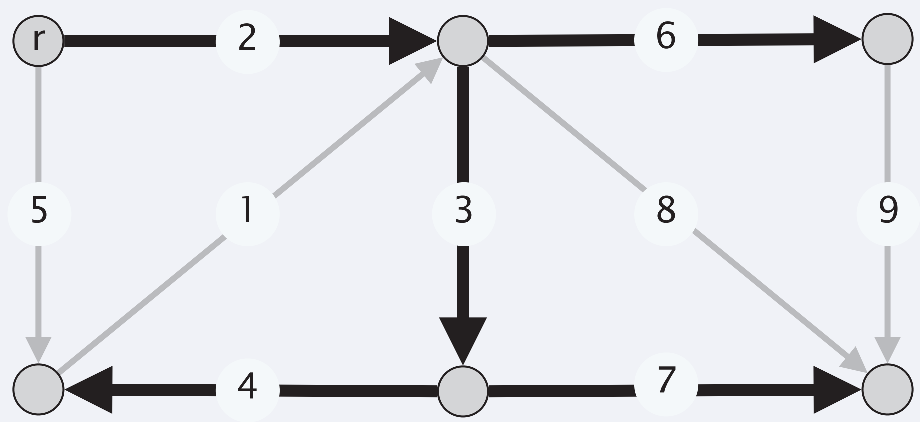
$\Rightarrow$ If T is an arborescence, then no (directed) cycles and every node $v \neq r$ has exactly one entering edge—the last edge on the unique $r \leadsto v$ path.

$\Leftarrow$ Suppose T has no cycles and each node $v \neq r$ has one entering edge.

- To construct an $r \leadsto v$ path, start at v and repeatedly follow edges in the backward direction.
- Since T has no directed cycles, the process must terminate.
- It must terminate at r since r is the only node with no entering edge. ▪

Min-cost arborescence problem

Problem. Given a digraph G with a root node r and with a nonnegative cost $c_e \geq 0$ on each edge e, compute an arborescence rooted at r of minimum cost.



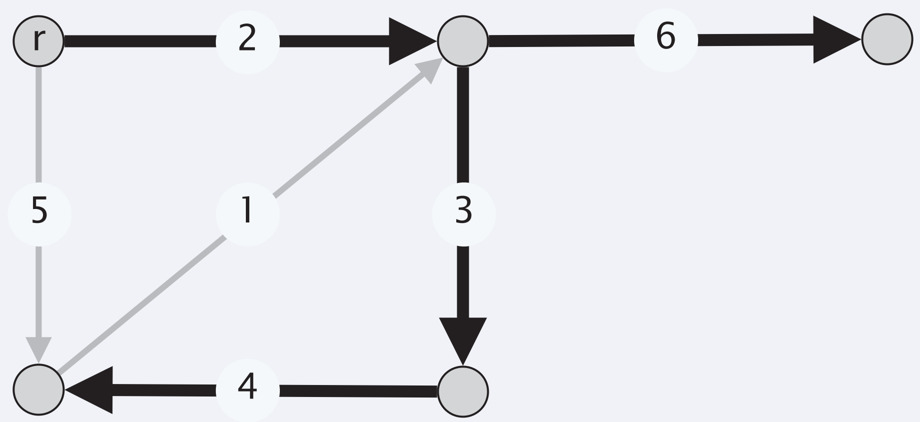
Assumption 1. G has an arborescence rooted at r.

Assumption 2. No edge enters r (safe to delete since they won't help).

Simple greedy approaches do not work

Observations. A min-cost arborescence need not:

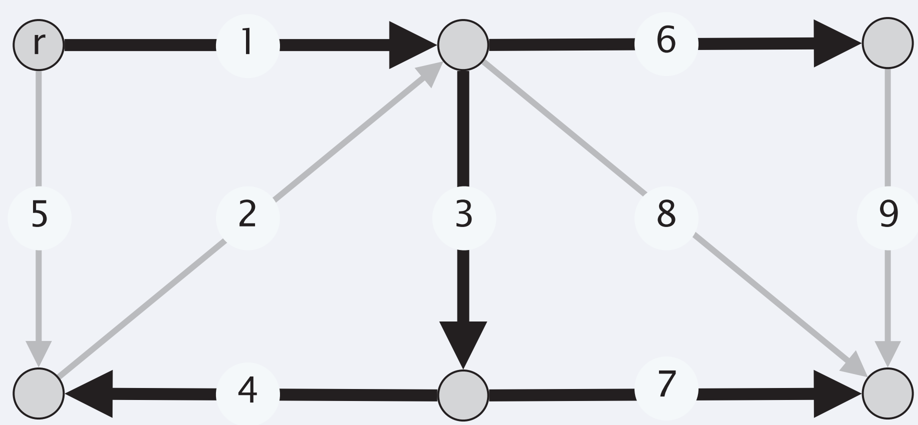
- Be a shortest-paths tree.
- Include the cheapest edge (in some cut).
- Exclude the most expensive edge (in some cycle).



A sufficient optimality condition

Property. For each node $v \neq r$, choose one cheapest edge entering v and let F^* denote this set of $n - 1$ edges. If (V, F^*) is an arborescence, then it is a min-cost arborescence.

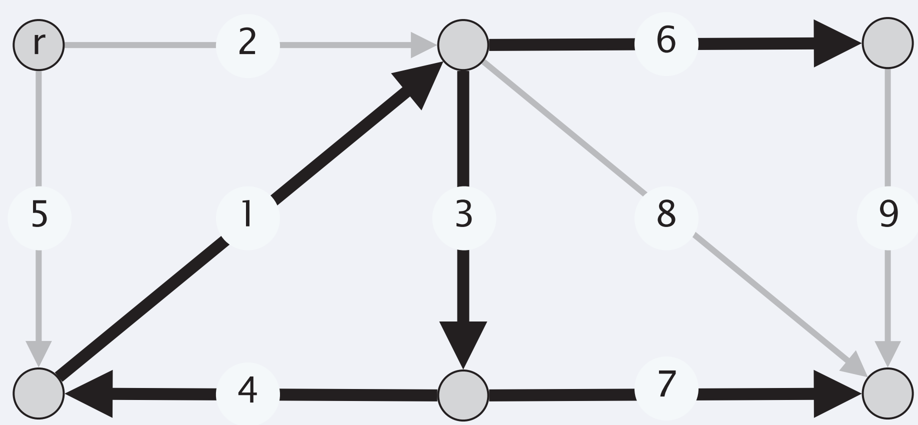
Pf. An arborescence needs exactly one edge entering each node $v \neq r$ and (V, F^*) is the cheapest way to make these choices. ▪



A sufficient optimality condition

Property. For each node $v \neq r$, choose one cheapest edge entering v and let F^* denote this set of $n - 1$ edges. If (V, F^*) is an arborescence, then it is a min-cost arborescence.

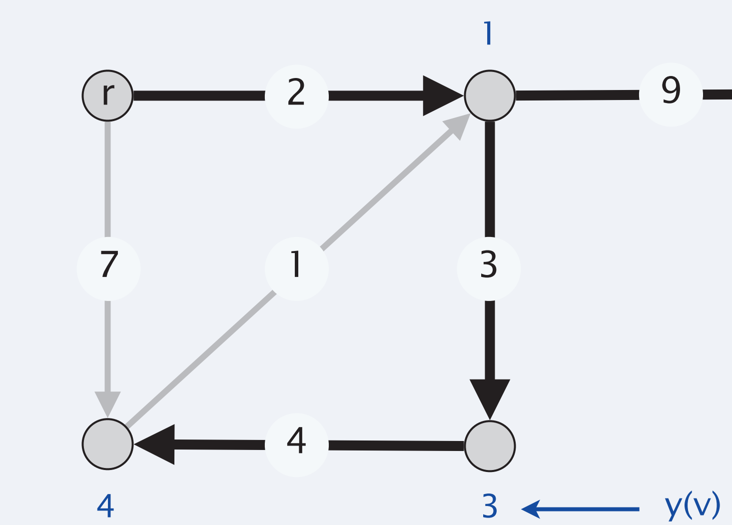
Note. F^* may not be an arborescence (since it may have directed cycles).

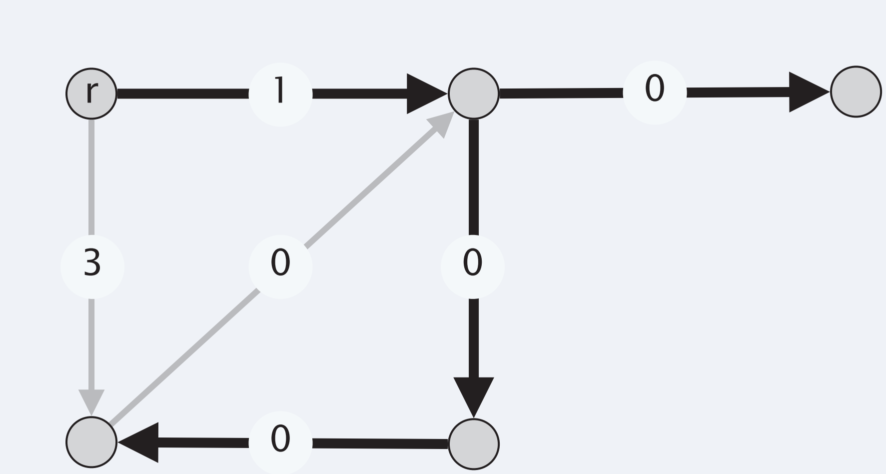


Reduced costs

Def. For each $v \neq r$, let $y(v)$ denote the min cost of any edge entering v.
The **reduced cost** of an edge (u, v) is $c'(u, v) = c(u, v) - y(v) \ \geq 0$.

Observation. T is a min-cost arborescence in G using costs c iff
T is a min-cost arborescence in G using reduced costs c'.
Pf. Each arborescence has exactly one edge entering v.

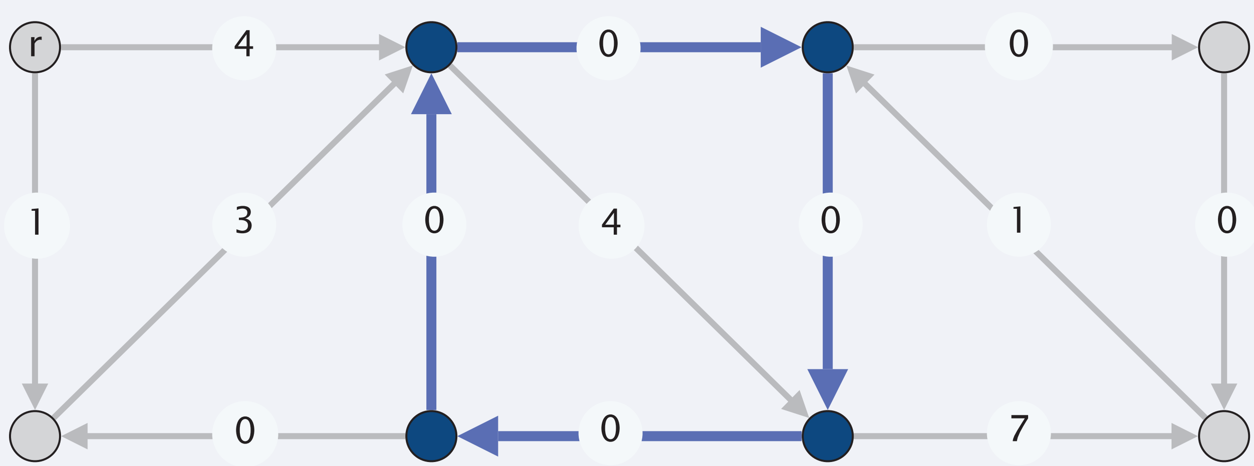




Edmonds branching algorithm: intuition

Intuition. Recall F^* = set of cheapest edges entering v for each $v \neq r$.

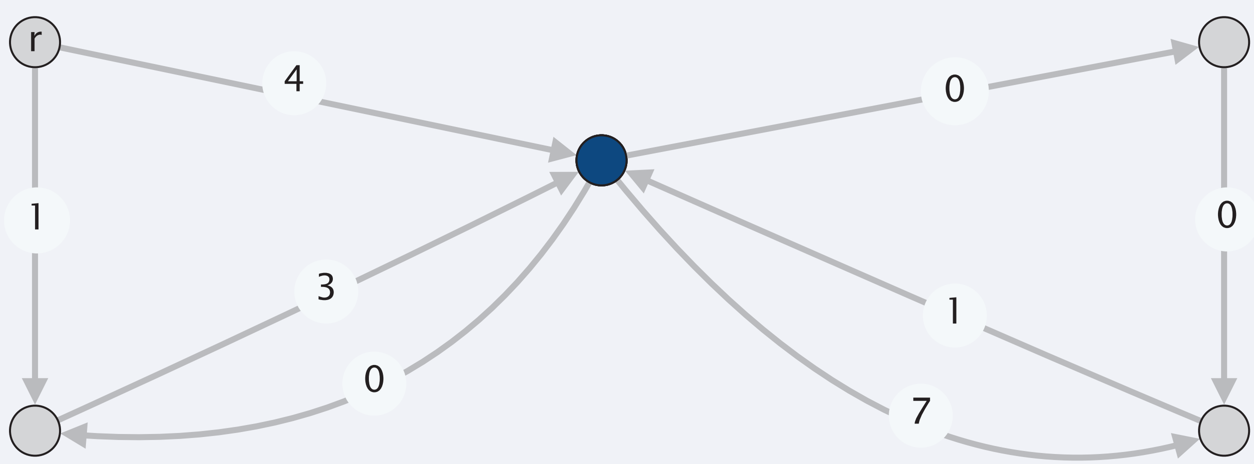
- Now, all edges in F^* have 0 cost with respect to costs $c'(u, v)$.
- If F^* does not contain a cycle, then it is a min-cost arborescence.
- If F^* contains a cycle C, can afford to use as many edges in C as desired.
- Contract nodes in C to a supernode.
- Recursively solve problem in contracted network G' with costs $c'(u, v)$.



Edmonds branching algorithm: intuition

Intuition. Recall F^* = set of cheapest edges entering v for each $v \neq r$.

- Now, all edges in F^* have 0 cost with respect to costs $c'(u, v)$.
- If F^* does not contain a cycle, then it is a min-cost arborescence.
- If F^* contains a cycle C, can afford to use as many edges in C as desired.
- Contract nodes in C to a supernode (removing any self-loops).
- Recursively solve problem in contracted network G' with costs $c'(u, v)$.



Edmonds branching algorithm

EDMONDSBRANCHING(G, r, c)

FOREACH $v \neq r$

$y(v) \leftarrow$ min cost of an edge entering v.

$c'(u, v) \leftarrow c'(u, v) - y(v)$ for each edge (u, v) entering v.

FOREACH $v \neq r$: choose one 0-cost edge entering v and let F^* be the resulting set of edges.

IF F^* forms an arborescence, RETURN $T = (V, F^*)$.

ELSE

$C \leftarrow$ directed cycle in F^*.

Contract C to a single supernode, yielding $G' = (V', E')$.

$T' \leftarrow$ EDMONDSBRANCHING(G', r, c')

Extend T' to an arborescence T in G by adding all but one edge of C.

RETURN T.

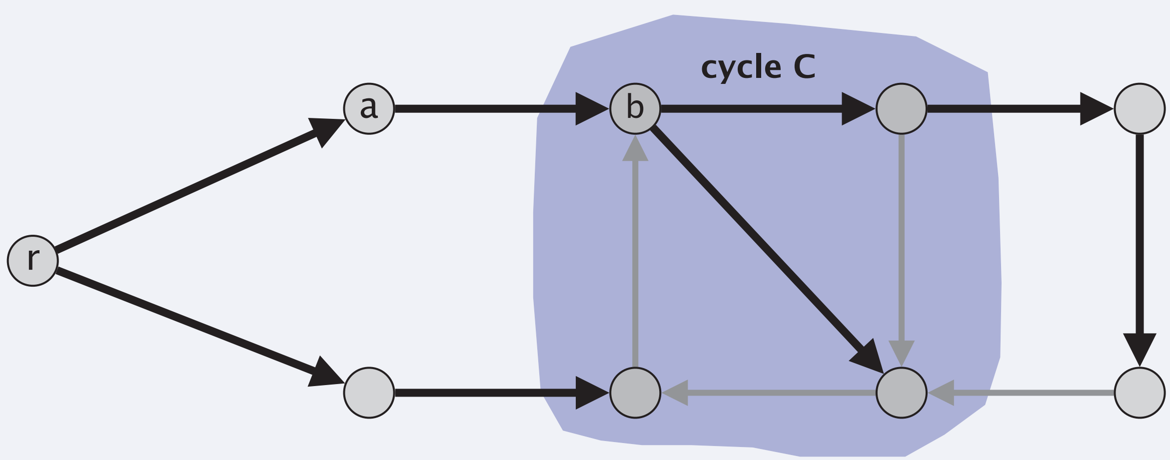
Edmonds branching algorithm

Q. What could go wrong?

A.

- Min-cost arborescence in G' has exactly one edge entering a node in C (since C is contracted to a single node)
- But min-cost arborescence in G might have more edges entering C.

min-cost arborescence in G



Edmonds branching algorithm: key lemma

Lemma. Let C be a cycle in G consisting of 0-cost edges. There exists a min-cost arborescence rooted at r that has exactly one edge entering C.

Pf. Let T be a min-cost arborescence rooted at r.

Case 0. T has no edges entering C.
Since T is an arborescence, there is an $r \rightarrow v$ path fore each node v $\Rightarrow$ at least one edge enters C.

Case 1. T has exactly one edge entering C.
T satisfies the lemma.

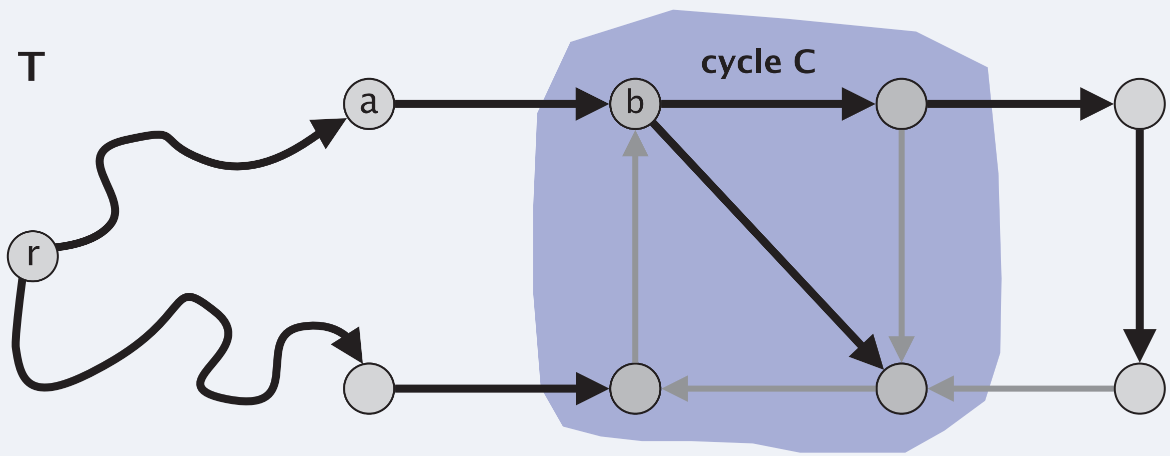
Case 2. T has more than one edge that enters C.
We construct another min-cost arborescence T' that has exactly one edge entering C.

Edmonds branching algorithm: key lemma

Case 2 construction of T'.

- Let (a, b) be an edge in T entering C that lies on a shortest path from r.
- We delete all edges of T that enter a node in C except (a, b).
- We add in all edges of C except the one that enters b.

path from r to C uses only one node in C



Edmonds branching algorithm: key lemma

Case 2 construction of T'.

- Let (a, b) be an edge in T entering C that lies on a shortest path from r.
- We delete all edges of T that enter a node in C except (a, b).
- We add in all edges of C except the one that enters b.

path from r to C uses only one node in C

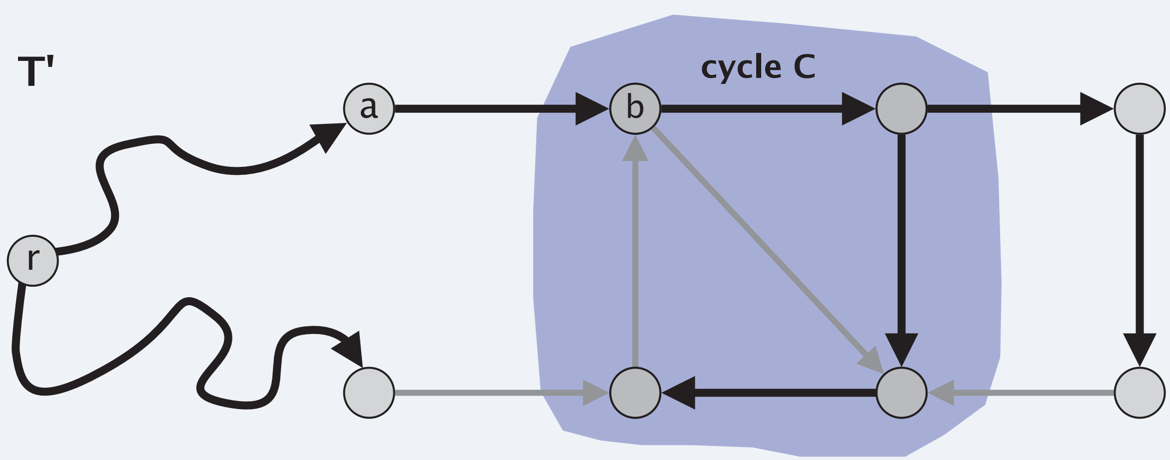
Claim. T' is a min-cost arborescence.

- The cost of T' is at most that of T since we add only 0-cost edges.
- T' has exactly one edge entering each node $v \neq r$.
- T' has no directed cycles.

  (T had no cycles before; no cycles within C; now only (a, b) enters C)

T is an arborescence rooted at r

and the only path in T' to a is the path from r to a (since any path must follow unique entering edge back to r)



Edmonds branching algorithm: analysis

Theorem. [Chu-Liu 1965, Edmonds 1967] The greedy algorithm finds a min-cost arborescence.

Pf. [by induction on number of nodes in G]

- If the edges of F^* form an arborescence, then min-cost arborescence.
- Otherwise, we use reduced costs, which is equivalent.
- After contracting a 0-cost cycle C to obtain a smaller graph G', the algorithm finds a min-cost arborescence T' in G' (by induction).
- Key lemma: there exists a min-cost arborescence T in G that corresponds to T'. ▪

Theorem. The greedy algorithm can be implemented in $O(m\,n)$ time.

Pf.

- At most n contractions (since each reduces the number of nodes).
- Finding and contracting the cycle C takes $O(m)$ time.
- Transforming T' into T takes $O(m)$ time. ▪

Min-cost arborescence

Theorem. [Gabow-Galil-Spencer-Tarjan 1985] There exists an $O(m + n \log n)$ time algorithm to compute a min-cost arborescence.

COMBINATORICA 6 (2) (1986) 109—122

EFFICIENT ALGORITHMS FOR FINDING MINIMUM SPANNING TREES IN UNDIRECTED AND DIRECTED GRAPHS

H. N. GABOW*, Z. GALIL**, T. SPENCER*** and R. E. TARJAN

Received 23 January 1985
Revised 1 December 1985

Recently, Fredman and Tarjan invented a new, especially efficient form of heap (priority queue). Their data structure, the *Fibonacci heap* (or F-heap) supports arbitrary deletion in $O(\log n)$ amortized time and other heap operations in $O(1)$ amortized time. In this paper we use F-heaps to obtain fast algorithms for finding minimum spanning trees in undirected and directed graphs. For an undirected graph containing n vertices and m edges, our minimum spanning tree algorithm runs in $O(m \log \beta(m, n))$ time, improved from $O(m\beta(m, n))$ time, where $\beta(m, n) = \min \{i \mid \log^{(i)} n \le m/n\}$. Our minimum spanning tree algorithm for directed graphs runs in $O(n \log n + m)$ time, improved from $O(n \log n + m \log \log \log_{(m/n+2)} n)$. Both algorithms can be extended to allow a degree constraint at one vertex.