COMP331/557

Chapter 6:
Optimal Trees and Paths

(Cook, Cunningham, Pulleyblank & Schrijver, Chapter 2)

Trees and Forests

Definition 6.1.

1. (i) An undirected graph having no circuit is called a forest.
2. (ii) A connected forest is called a tree.

Theorem 6.2.

Let $G = (V, E)$ be an undirected graph on $n = |V|$ nodes. Then, the following statements are equivalent:

- (i) G is a tree.
- (ii) G has $n - 1$ edges and no circuit.
- (iii) G has $n - 1$ edges and is connected.
- (iv) G is connected. If an arbitrary edge is removed, the resulting subgraph is disconnected.
- (v) G has no circuit. Adding an arbitrary edge to G creates a circuit.
- (vi) G contains a unique path between any pair of nodes.

Kruskal's Algorithm

Minimum Spanning Tree (MST) Problem

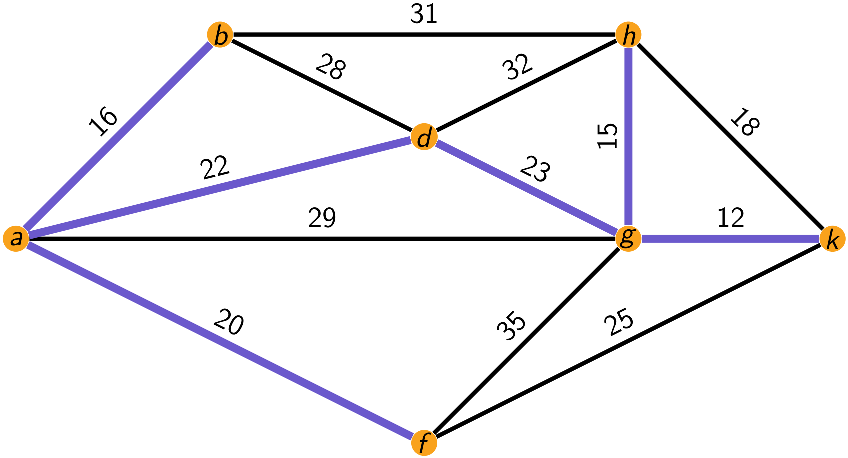
Given: connected graph $G = (V, E)$, cost function $c : E \to \mathbb{R}$.

Task: find spanning tree $T = (V, F)$ of G with minimum cost $\sum_{e \in F} c(e)$.

Kruskal's Algorithm for MST

1. Sort the edges in E such that $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$.
2. Set $T := (V, \emptyset)$.
3. For $i := 1$ to m do:
   If adding e_i to T does not create a circuit, then add e_i to T.

Example for Kruskal's Algorithm



Prim's Algorithm

Notation: For a graph $G = (V, E)$ and $A \subseteq V$ let

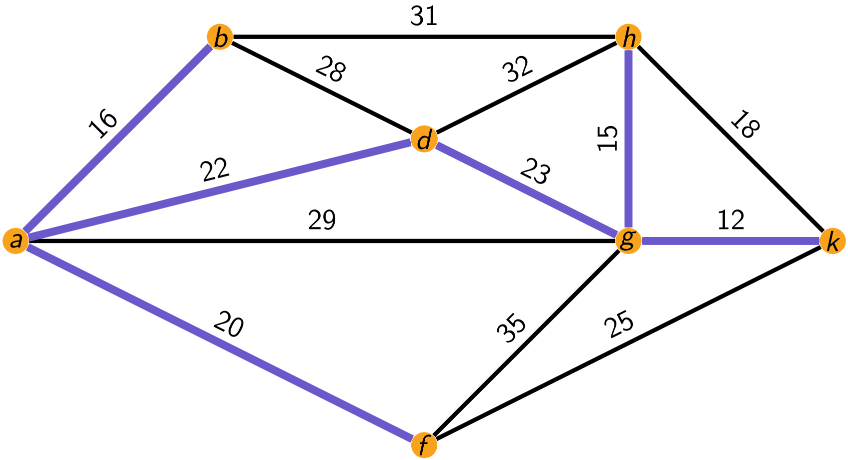
$$\delta(A) := \{e = \{v, w\} \in E \mid v \in A \text{ and } w \in V \setminus A\} \ .$$

We call $\delta(A)$ the cut induced by A.

Prim's Algorithm for MST

1. Set $U := \{r\}$ for some node $r \in V$ and $F := \emptyset$; set $T := (U, F)$.
2. While $U \neq V$, determine a minimum cost edge $e \in \delta(U)$.
3. Set $F := F \cup \{e\}$ and $U := U \cup \{w\}$ with $e = \{v, w\}$, $w \in V \setminus U$.

Example for Prim's Algorithm



Correctness of the MST Algorithms

Lemma 6.3.

A graph $G = (V, E)$ is connected if and only if there is no set $A \subseteq V$, $\emptyset \neq A \neq V$, with $\delta(A) = \emptyset$.

Notation: We say that $B \subseteq E$ is extendible to an MST if B is contained in the edge-set of some MST of G.

Theorem 6.4.

Let $B \subseteq E$ be extendible to an MST and $\emptyset \neq A \subsetneq V$ with $B \cap \delta(A) = \emptyset$.
If e is a min-cost edge in $\delta(A)$, then $B \cup \{e\}$ is extendible to an MST.

- Correctness of Prim's Algorithm immediately follows.
- Kruskal: Whenever an edge $e = \{v, w\}$ is added, it is cheapest edge in cut induced by subset of nodes currently reachable from v.

Efficiency of Prim's Algorithm

Prim's Algorithm for MST

1. Set $U := \{r\}$ for some node $r \in V$ and $F := \emptyset$; set $T := (U, F)$.
2. While $U \neq V$, determine a minimum cost edge $e \in \delta(U)$.
3. Set $F := F \cup \{e\}$ and $U := U \cup \{w\}$ with $e = \{v, w\}$, $w \in V \setminus U$.

- Straightforward implementation achieves running time $O(nm)$ where, as usual, $n := |V|$ and $m := |E|$:
  - the while-loop has $n - 1$ iterations;
  - a min-cost edge $e \in \delta(U)$ can be found in $O(m)$ time.
- Best known running time is $O(m + n \log n)$ (uses Fibonacci heaps).

Efficiency of Kruskal's Algorithm

Kruskal's Algorithm for MST

1. Sort the edges in E such that $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$.
2. Set $T := (V, \emptyset)$.
3. For $i := 1$ to m do:
   If adding e_i to T does not create a circuit, then add e_i to T.

Theorem 6.5.

Kruskal's Algorithm can be implemented to run in $O(m \log m)$ time.

Minimum Spanning Trees and Linear Programming

Notation:

- For $S \subseteq V$ let $\gamma(S) := \{e = \{v, w\} \in E \mid v, w \in S\}$.
- For a vector $x \in \mathbb{R}^E$ and a subset $B \subseteq E$ let $x(B) := \sum_{e \in B} x_e$.

Consider the following integer linear program:

$$
\begin{aligned}
\min \quad & c^T \cdot x & & \\
\text{s.t.} \quad & x(\gamma(S)) \leq |S| - 1 & \text{for all } \emptyset \neq S \subset V & \quad (6.1)\\
& x(E) = |V| - 1 & & \quad (6.2)\\
& x_e \in \{0, 1\} & \text{for all } e \in E &
\end{aligned}
$$

Observations:

- Feasible solution $x \in \{0, 1\}^E$ is characteristic vector of subset $F \subseteq E$.
- F does not contain circuit due to (6.1) and $n - 1$ edges due to (6.2).
- Thus, F forms a spanning tree of G.
- Moreover, the edge set of an arbitrary spanning tree of G yields a feasible solution $x \in \{0, 1\}^E$.

Minimum Spanning Trees and Linear Programming (cont.)

Consider LP relaxation of the integer programming formulation:

$$
\begin{aligned}
\min \quad & c^T \cdot x & \\
\text{s.t.} \quad & x(\gamma(S)) \leq |S| - 1 & \text{for all } \emptyset \neq S \subset V \\
& x(E) = |V| - 1 & \\
& x_e \geq 0 & \text{for all } e \in E
\end{aligned}
$$

Theorem 6.6.

Let $x^* \in \{0,1\}^E$ be the characteristic vector of an MST. Then x^* is an optimal solution to the LP above.

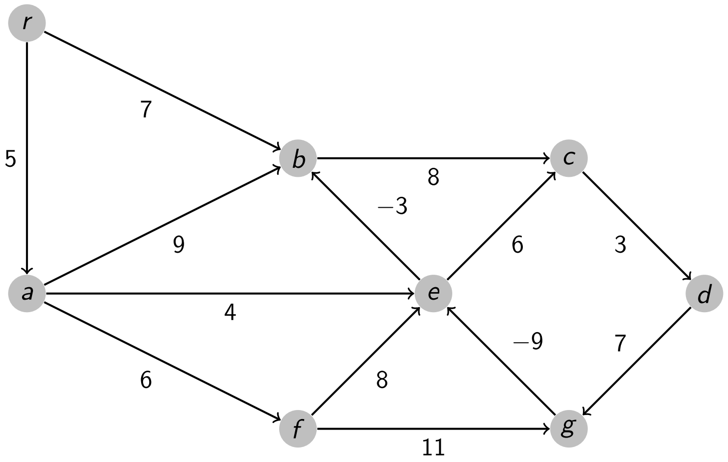
Corollary 6.7.

The vertices of the polytope given by the set of feasible LP solutions are exactly the characteristic vectors of spanning trees of G. The polytope is thus the convex hull of the characteristic vectors of all spanning trees.

Shortest Path Problem

Given: digraph $D = (V, A)$, node $r \in V$, arc costs c_a, $a \in A$.

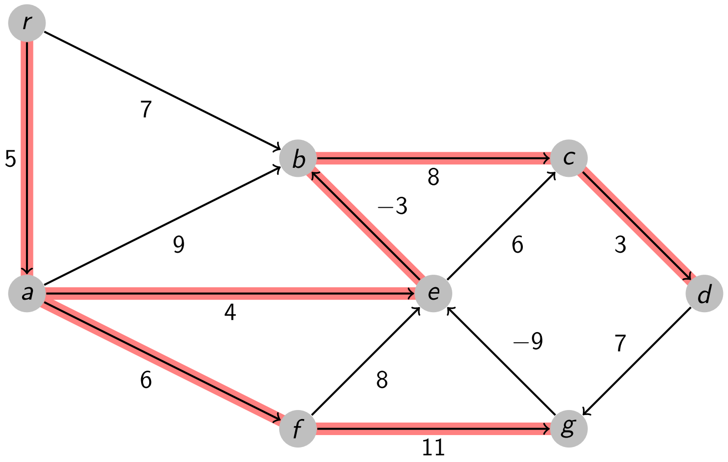
Task: for each $v \in V$, find dipath from r to v of least cost (if one exists)



Shortest Path Problem

Given: digraph $D = (V, A)$, node $r \in V$, arc costs c_a, $a \in A$.

Task: for each $v \in V$, find dipath from r to v of least cost (if one exists)



Shortest Path Problem

Given: digraph $D = (V, A)$, node $r \in V$, arc costs c_a, $a \in A$.

Task: for each $v \in V$, find dipath from r to v of least cost (if one exists)

Remarks:

- Existence of r-v-dipath can be checked, e. g., by breadth-first search.
- Ensure existence of r-v-dipaths: add arcs (r, v) of suffic. large cost.

Basic idea behind all algorithms for solving shortest path problem:

If y_v, $v \in V$, is the least cost of a dipath from r to v, then

$$y_v + c_{(v,w)} \geq y_w \quad \text{for all } (v, w) \in A. \tag{6.3}$$

Remarks:

- More generally, subpaths of shortest paths are shortest paths!
- If there is a shortest r-v-dipath for all $v \in V$, then there is a shortest path tree, i. e., a directed spanning tree T rooted at r such that the unique r-v-dipath in T is a least-cost r-v-dipath in D.

Feasible Potentials

Definition 6.8.

A vector $y \in \mathbb{R}^V$ is a feasible potential if it satisfies (6.3).

Lemma 6.9.

If y is feasible potential with $y_r = 0$ and P an r-v-dipath, then $y_v \leq c(P)$.

Proof: Suppose that P is $v_0, a_1, v_1, \ldots, a_k, v_k$, where $v_0 = r$ and $v_k = v$. Then,

$$c(P) = \sum_{i=1}^{k} c_{a_i} \geq \sum_{i=1}^{k} (y_{v_i} - y_{v_{i-1}}) = y_{v_k} - y_{v_0} = y_v \quad .$$

□

Corollary 6.10.

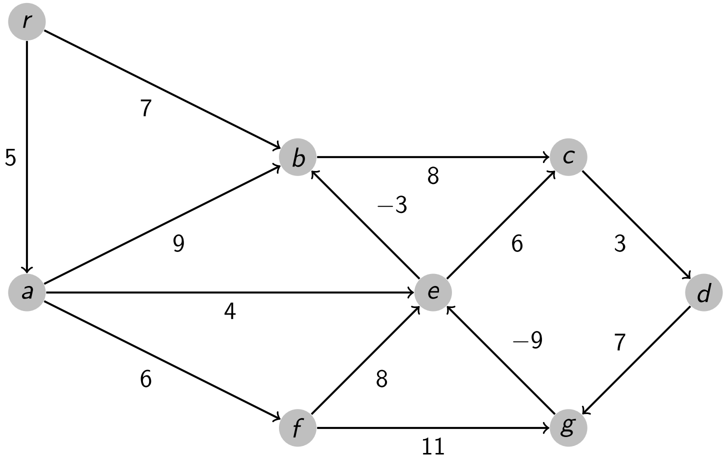
If y is a feasible potential with $y_r = 0$ and P an r-v-dipath of cost y_v, then P is a least-cost r-v-dipath. □

Ford's Algorithm

Ford's Algorithm

1. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
2. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v \quad .$$



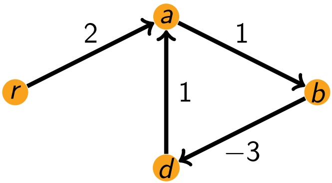
Ford's Algorithm

Ford's Algorithm

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set
$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v \ .$$

Question: Does the algorithm always terminate?

Example:



Observation:
The algorithm does not terminate because of the negative-cost dicircuit.

Validity of Ford's Algorithm

Lemma 6.11.

If there is no negative-cost dicircuit, then at any stage of the algorithm:

a) if $y_v \neq \infty$, then y_v is the cost of some simple dipath from r to v;
b) if $p(v) \neq$ null, then p defines a simple r-v-dipath of cost at most y_v.

Theorem 6.12.

If there is no negative-cost dicircuit, then Ford's Algorithm terminates after a finite number of iterations. At termination, y is a feasible potential with $y_r = 0$ and, for each node $v \in V$, p defines a least-cost r-v-dipath.

Feasible Potentials and Negative-Cost Dicircuits

Theorem 6.13.

A digraph $D = (V, A)$ with arc costs $c \in \mathbb{R}^A$ has a feasible potential if and only if there is no negative-cost dicircuit.

Remarks:

- If there is a dipath but no least-cost dipath from r to v, it is because there are arbitrarily cheap nonsimple r-v-dipaths.
- Finding a least-cost simple dipath from r to v is, however, difficult (see later).

Lemma 6.14.

If c is integer-valued, $C := 2 \max_{a \in A} |c_a| + 1$, and there is no negative-cost dicircuit, then Ford's Algorithm terminates after at most $C\, n^2$ iterations.

Proof: Exercise. □

Feasible Potentials and Linear Programming

As a consequence of Ford's Algorithm we get:

Theorem 6.15.

Let $D = (V, A)$ be a digraph, $r, s \in V$, and $c \in \mathbb{R}^A$. If, for every $v \in V$, there exists a least-cost dipath from r to v, then

$$\min\{c(P) \mid P \text{ an } r\text{-}s\text{-dipath}\} = \max\{y_s - y_r \mid y \text{ a feasible potential}\} \ .$$

Formulate the right-hand side as a linear program and consider the dual:

$$\begin{aligned} \max \quad & y_s - y_r \\ \text{s.t.} \quad & y_w - y_v \leq c_{(v,w)} \\ & \qquad \text{for all } (v,w) \in A \end{aligned}$$

$$\begin{aligned} \min \quad & c^T \cdot x \\ \text{s.t.} \quad & \sum_{a \in \delta^-(v)} x_a - \sum_{a \in \delta^+(v)} x_a = b_v \quad \forall v \in V \\ & x_a \geq 0 \qquad \text{for all } a \in A \end{aligned}$$

with $b_s = 1$, $b_r = -1$, and $b_v = 0$ for all $v \notin \{r, s\}$.

Notice: The dual is the LP relaxation of an ILP formulation of the shortest r-s-dipath problem ($x_a \hat{=}$ number of times a shortest r-s-dipath uses arc a).

Bases of Shortest Path LP

Consider again the dual LP:

$$\begin{aligned} \min \quad & c^T \cdot x \\ \text{s.t.} \quad & \sum_{a \in \delta^-(v)} x_a - \sum_{a \in \delta^+(v)} x_a = b_v \quad \text{for all } v \in V \\ & x_a \geq 0 \qquad \text{for all } a \in A \end{aligned}$$

The underlying matrix Q is the incidence matrix of D.

Lemma 6.16.

Let $D = (V, A)$ be a connected digraph and Q its incidence matrix. A subset of columns of Q indexed by a subset of arcs $F \subseteq A$ forms a basis of the linear subspace of $\mathbb{R}^n$ spanned by the columns of Q if and only if F is the arc-set of a spanning tree of D.

Proof: Exercise. □

Refinement of Ford's Algorithm

Ford's Algorithm

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v \quad .$$

- # iterations crucially depends on order in which arcs are chosen.
- Suppose that arcs are chosen in order $\mathcal{S} = f_1, f_2, f_3, \ldots, f_\ell$.
- Dipath P is embedded in $\mathcal{S}$ if P's arc sequence is a subsequence of $\mathcal{S}$.

Lemma 6.17.

If an r-v-dipath P is embedded in $\mathcal{S}$, then $y_v \leq c(P)$ after Ford's Algorithm has gone through the sequence $\mathcal{S}$.

Goal: Find short sequence $\mathcal{S}$ such that, for all $v \in V$, a least-cost r-v-dipath is embedded in $\mathcal{S}$.

Ford-Bellman Algorithm

Basic idea:

- Every simple dipath is embedded in $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{n-1}$ where, for all i, $\mathcal{S}_i$ is an ordering of A.
- This yields a shortest path algorithm with running time $O(nm)$.

Ford-Bellman Algorithm

i initialize y, p (see Ford's Algorithm);

ii for $i = 1$ to $n - 1$ do

iii   for all $a = (v, w) \in A$ do

iv   if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Theorem 6.18.

The algorithm runs in $O(nm)$ time. If, at termination, y is a feasible potential, then p yields a least-cost r-v-dipath for each $v \in V$. Otherwise, the given digraph contains a negative-cost dicircuit. □

Acyclic Digraphs and Topological Orderings

Definition 6.19.

Consider a digraph $D = (V, A)$.

a An ordering $v_1, v_2, \dots, v_n$ of V so that $i < j$ for each $(v_i, v_j) \in A$ is called a topological ordering.

b If D has a topological ordering, then D is called acyclic.

Observations:

- Digraph D is acyclic if and only if it does not contain a dicircuit.
- Let D be acyclic and $\mathcal{S}$ an ordering of A such that (v_i, v_j) precedes (v_k, v_ℓ) if $i < k$. Then every dipath of D is embedded in $\mathcal{S}$.

Theorem 6.20.

The shortest path problem on acyclic digraphs can be solved in time $O(m)$.

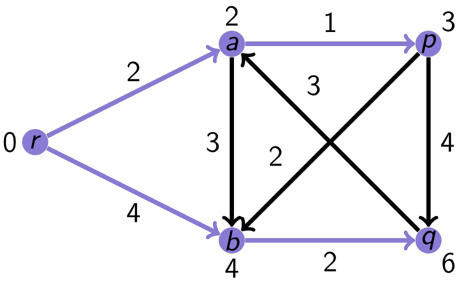
Dijkstra's Algorithm

Consider the special case of nonnegative costs, i. e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

- i initialize y, p (see Ford's Algorithm); set $S := V$;
- ii while $S \neq \emptyset$ do
- iii choose $v \in S$ with y_v minimum and delete v from S;
- iv for each $w \in V$ with $(v, w) \in A$ do
- v if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Example:



Correctness of Dijkstra's Algorithm

Lemma 6.21.

For each $w \in V$, let y'_w be the value of y_w when w is removed from S.
If u is deleted from S before v, then $y'_u \leq y'_v$.

Theorem 6.22.

If $c \geq 0$, then Dijkstra's Algorithm solves the shortest paths problem correctly in time $O(n^2)$. A heap-based implementation yields running time $O(m \log n)$.

Remark: The for-loop in Dijkstra's Algorithm (step iv) can be modified such that only arcs (v, w) with $w \in S$ are considered.