Combinatorial Optimization

CO 450

Ricardo Fukasawa

LaTeXed by Sibelius Peng

Preface

Disclaimer Much of the information on this set of notes is transcribed directly/indirectly from the lectures of CO 450 during Fall 2020 as well as other related resources. I do not make any warranties about the completeness, reliability and accuracy of this set of notes. Use at your own risk.

For any questions, send me an email via https://notes.sibeliusp.com/contact.

You can find my notes for other courses on https://notes.sibeliusp.com/.

Sibelius Peng

Contents

1

Minimum spanning trees

The goal is to develop algorithms to find a minimum-cost spanning tree (MST). We follow these steps:

- Characterize spanning trees;
- Characterize MSTs;
- Use such characterizations to derive algorithms (and prove correctness);
- Alternatively prove correctness using linear programming (why is this useful?).

1.1 Spanning Tree

spanning tree

Given $G = (V, E)$, a subgraph T is **spanning tree** of G if the following conditions hold

- $V(T) = V(G)$;
- T is connected; and
- T is acyclic.

Theorem 1.1

Let $G = (V, E)$ be a connected graph, T is a subgraph of G with $V(T) = V(G)$. $|V| = n$. TFAE:

1. T is a spanning tree of G.
2. T is minimally connected.
3. T is maximally acyclic,
4. T is connected and has $n - 1$ edges.
5. T is acyclic and has $n - 1$ edges.
6. $\forall u, v \in V$, there exists a unique $u - v$ path in T (denoted by $T_{u,v}$).

Theorem 1.2

A graph $G = (V, E)$ is connected if and only if $\forall A \subseteq V$ with $\varnothing \neq A \neq V$, we have $\delta(A) \neq \varnothing$.

Here for $A \subseteq V$, $\delta(A) := \{e \in E : |e \cap A| = 1\}$.

1.2 Minimum Spanning Tree

Minimum Spanning Tree Problem

Input: Connected graph $G = (V, E)$, costs $c_e \ \ \forall e \in E$.

Output: A spanning tree T of G of minimum cost $c(T) := \sum_{e \in E(T)} c_e$

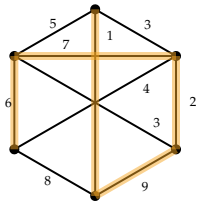
Theorem 1.3

Let $G = (V, E)$, $\mathbf{c} : E \to \mathbb{R}$, T a spanning tree of G. Then TFAE:

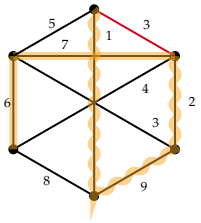
(a) T is a MST.

(b) $\forall uv \in E \setminus E(T)$, all edges e on $T_{u,v}$ have $c_e \leq c_{uv}$.

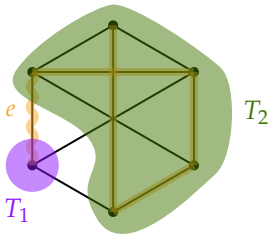
(c) $\forall e \in E(T)$, let T_1, T_2 be the two connected components obtained from T when removing e. Then e is a minimum cost edge in $\delta(T_1)$.



Not a MST



Violate (b)



Violate (c)

Proof idea:

(a) ⇒ (b) Suppose $\exists uv \in E \setminus E(T)$ and $e \in T_{uv}$ such that $c_e > c_{uv}$. Then we can show that $T' = T + uv - e$ is a spanning tree with strictly smaller cost, then implies T is not a MST.

(b) ⇒ (c) Suppose $\exists e \in T$, and T_1, T_2 be two connected components from $T - e$. Let $uv \in \delta(T_1)$ such that $c_{uv} < c_e$. Then we can show that $uv \notin E(T)$ and $e \in T_{uv}$, which implies (b) is false.

(c) ⇒ (a) Suppose T satisfies (c). Ket T^* be a MST with largest $k := |E(T) \cap E(T^*)|$. If $k = n - 1$, then $T = T^*$, we are done. Otherwise, $\exists e \in E(T) \setminus E(T^*)$. Let T_1, T_2 be connected components of $T - e$, then there exists $e^* \in E(T^*_{u,v}) \cap \delta(T_1)$. Then we can show that $e^* \notin E(T)$. Then $T' = T^* - e^* + e$ is a spanning tree such that $c(T') \leq c(T^*)$ as $c_e \leq c^*_e$ by assumption. Then T' is a MST, contradicts choice of T^*.

□

1.3 Kruskal's Algorithm

So far we have characterized spanning trees and MSTs.

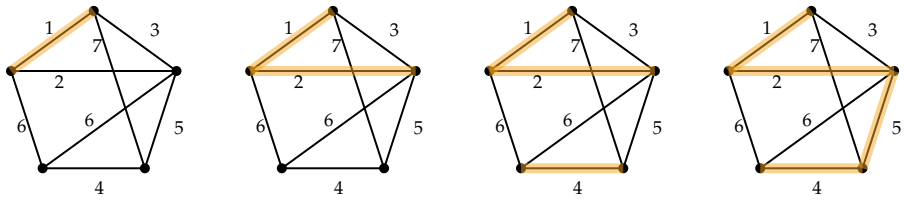
Let $n = |V|$, $m = |E|$.

Algorithm 1: Kruskal's algorithm (Assume G is connected)

1 Initialization: $H = (V, \varnothing)$
2 **while** *H is not a Spanning tree* **do**
3 $\quad e =$ cheapest edge whose endpoints are in different connected components of H;
4 $\quad$ Add e to H;
5 **return** H

Let's take a look at an example.

Example:



Algorithm 2: Kruskal's algorithm (Alternative version)

1 Initialization: $H = (V, \varnothing)$
2 **while** *H is not a Spanning tree* **do**
3 $\quad$ Sort edges so that $c_{e_1} \leq \cdots \leq c_{e_m}$ // a)
4 $\quad$ **for** $i = 1, \ldots, m$ **do** // b)
5 $\quad\quad$ **if** *end points u, v of e_i are in different connected components of H* **then** // c)
6 $\quad\quad\quad H \leftarrow H + e_i$ // d)
7 **return** H

Naïve implementation: Keep array `comp`, with `comp[v]` $\leftarrow$ `v` for all $v \in V$ initially. c) can be done by checking if `comp[u] == comp[v]`, for $e = uv$. This is $O(1)$. When d) executed, go through `comp[t]`, $\forall t \in V$ and if `comp[t] == comp[v]`, `comp[t]` $\leftarrow$ `comp[u]`. This is $O(n)$. a) is sorting, which is $O(m \log m)$. b) is $O(m)$. Thus the overall running time is $O(mn)$ which is polytime.

We have established runtime. Let's now establish the correctness.

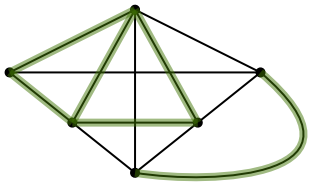
First we want to show step 3 of Algorithm 1 is well-defined. Why e always exists? This can be shown using previous results. Then we need to make sure we are not in an infinite loop. Every time step 4 is executed, number of connected components of H reduces by 1, and H is acyclic. Therefore, while loop terminates in $O(n)$ iterations. As it exits from while loop, the algorithm returns a connected, acyclic graph, i.e., a spanning tree.

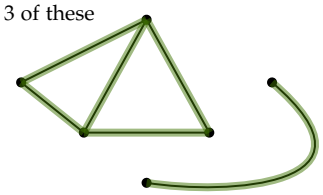
Why does it return MST? Now let's return to Algorithm 2. Suppose it does not return a MST. Then $\exists uv \in E \setminus E(H)$ and $e \in H_{u,v}$ with $c_{uv} < c_e$. When uv is being tested in c), $H_{u,v}$ still does not exist. Then uv would have been added to H.

1.4 Correctness via LP

Previously, we have shown the correctness of Kruskal's algorithm with pure combinatorial argument. In this section, we will prove the correctness via linear programming. Why bothering to prove via LP? First, it shows techniques that can be used in other settings. Second, this can be adapted to "good" approaches for more challenging problems, for example, approximation algorithms.

First, let's formulate MST problem as an integer programming. Let $x_e \in \{0,1\}$ be a variable that indicates if edge e is in the MST, with value 1, or not, with value 0. Recall that spanning trees are acyclic, and have $n-1$ edges, where n is the number of vertices as usual. Here we introduce a notation: for $F \subseteq E(G)$, $x(F) := \sum_{e \in F} x_e$. How do we characterize acyclic condition? Consider $F \subseteq E$, which are labelled green in the graph below. What is the largest number of edges that any spanning trees can have?





≤ 4 edges

Picking at most 4 edges can make an acyclic graph from F. Let $\kappa(F)$ be the number of connected components of (V, F). Thus we can pick at most $n - \kappa(F)$ edges. This constraint on acyclic conditions is then $x(F) \leq n - \kappa(F)$ for all $F \subseteq E$. Note that if we let $F = \{e\}$, this constraint also gives an upper bound on x_e: $x_e \leq n - (n-1) = 1$. Then we can write the integer programming in the following way:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e = c^T x \\ \text{s.t.} \quad & x(E) = n - 1 \\ & x(F) \leq n - \kappa(F) \quad \forall F \subseteq E \\ & x \geq \mathbf{0}, \qquad\qquad x \in \mathbb{Z}^E \end{aligned} \tag{IP$_{\text{st}}$}$$

The LP relaxation is then

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e = c^T x \\ \text{s.t.} \quad & x(E) = n - 1 \\ & x(F) \leq n - \kappa(F) \quad \forall F \subseteq E \\ & x \geq \mathbf{0} \end{aligned} \tag{P$_{\text{st}}$}$$

Since the graph is assumed to be connected, there always exists one spanning tree. Then IP, LP are feasible. LP is not unbounded, then by the Fundamental Theorem of LP, (P$_{\text{st}}$) has an optimal solution. Denote the optimal solution by $z^*_{\text{P}_{\text{st}}}$.

Note that (P$_{\text{st}}$) has exponentially many constraints, then we cannot input it to the software to solve. However, we can show that Kruskal's algorithm gives an optimal solution to (P$_{\text{st}}$).

Proof:
First note that any spanning tree corresponds to a feasible solution to (P$_{\text{st}}$), which implies the characteristic vector $c(T) \geq z^*_{\text{P}_{\text{st}}}$. We wish to use complementary slackness to show the spanning tree produced by Kruskal's algorithm is optimal for (P$_{\text{st}}$).

In (P$_{\text{st}}$), the first constraint is actually in the second constraint if we pick $F = E$. Therefore, we

change the quantifier in the second constraint to $\forall F \subset E$. Then we take the dual of (P_{st}):

$$\begin{aligned} \max \quad & \sum_{F \subseteq E} (n - \kappa(F)) y_F \\ \text{s.t.} \quad & \sum_{F: e \in F} y_F \leq c_E, \qquad \forall e \in E \\ & y_F \leq 0, \qquad \forall F \subset E \end{aligned} \tag{D_{st}}$$

Let $E = \{e_1, \ldots, e_m\}$ with $c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$. Let $E_i = \{e_1, \ldots, e_i\}$. Consider the following dual variables:

$$\begin{aligned} \bar{y}_{E_i} &= c_{e_i} - c_{e_{i+1}} \leq 0, \quad \forall i = 1, \ldots, m-1 \\ \bar{y}_E &= c_{e_m}, \quad \bar{y}_F = 0, \ \forall \text{ other } F \end{aligned}$$

Then we can see that $\bar{y}$ is feasible for (D_{st}) and all constraints are satisfied at equality (except $y_F \leq 0$ ones) by summing them up.

Now let $\bar{x}$ be the incidence vector of tree T constructed by Kruskal's algorithm. Note the notation: $\bar{x}(E_i) = \sum_{e \in E_i} \bar{x}_e = |E(T) \cap E_i|$. Then we can show that $T_i = (V, E_i \cap E(T))$ is a maximally acyclic subgraph of $H_i(V, E_i)$. One way to show this is to look at each connected components of T_i and how Kruskal's algorithm works. As a consequence, $\bar{x}(E_i) = n - \kappa(E_i)$ for $i = 1, \ldots, m$. It turns out $(\bar{x}, \bar{y})$ satisfy complementary slackness. Therefore, $c^T \bar{x} = c(T) = z^*_{P_{st}}$ which is optimal. □

In this proof, first thing to notice is that we are solving this LP. There's no guarantee that we are getting an integral solution. In most cases, the optimal solution to LP is fraction numbers. However, what we have shown is that $c(T^*) = z^*_{P_{st}}$, where T^* is MST, is an optimal solution to LP.

Also, we have an alternative formulation of LP:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e = c^T x \\ \text{s.t.} \quad & x(E) = n - 1 \\ & x(E(S)) \leq |S| - 1 \quad \forall \varnothing \subsetneq S \subsetneq V \\ & x \geq \mathbf{0} \end{aligned} \tag{P'_{st}}$$

where $E(S) = \{e \in E : |e \cap S| = 2\}$.

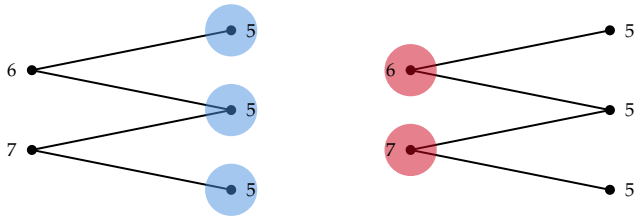
1.5 Greedy algorithm

In general, greedy algorithms make best decision based only on local structures. MST algorithms are greedy. Does it always work?

Example: Max weight independent set

Given $G = (V, E)$, $S \subseteq V$ is an independent set if $\forall u, v \in S$, $\{u, v\} \notin E$.

The problem: Given c_v, $\forall v \in V$, find independent set S maximizing $c(S) := \sum_{v \in S} c_v$.



Greedy gives the picture on the right, but clearly the left picture is optimal.

Given $G = (V, E)$, a forest is a subgraph (V, F), with $F \subseteq E$ that is acyclic. We will refer to a forest by its set of edges.

Maximum cost forest problem

Given $G = (V, E)$, $c_e, \forall e \in E$, find a forest F maximizing $c(F) := \sum_{e \in F} c_e$.

This problem can use MST algorithm.

Algorithm 3: Use MST to solve max cost forest

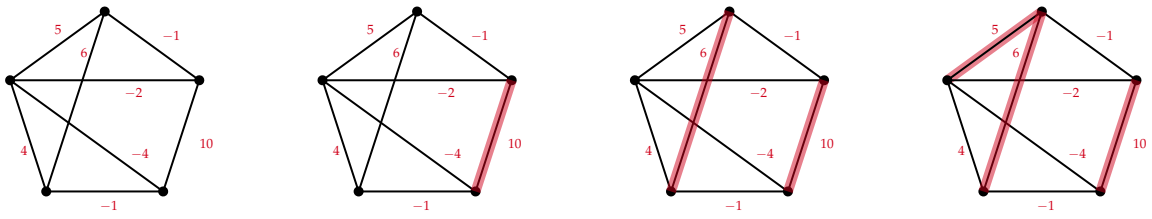
1 If G is not connected, add edges to it with cost $-M$ minimally.
2 Compute MST with respect to costs $c'_e = -c_e$.
3 Delete from MST all edges with $c_e \leq 0$

The algorithm above computes maximum cost forest (with respect to c_e).

Algorithm 4: Kruskal's algorithm for max cost forest

1 Initialization: $H = (V, \varnothing)$
2 **while** $\exists e \in E : c_e > 0$ *and whose end points are in different components of* H **do**
3 $\quad e =$ Highest cost such edge;
4 $\quad$ Add e to H;
5 **return** H

Example:



Also, we can use max forest algorithm as a black box to solve MST:

- Add $-M$ to $c_e, \forall e$, so that $c_e - M < 0$.
- Solve max cost forest with respect to $c'_e = -(c_e - M)$.

Then assume G is connected, the algorithm above produces MST.

2

Matriods

Index