Chapter 5

Minimum Spanning Trees

This chapter is based on well-known results. Our discussion follows Ahuja, Magnanti & Orlin [1, Chapter 13] and the survey by Magnanti and Wolsey [8].

The focus of this chapter is on the linear programming formulation of the minimum spanning tree problem due to Edmonds. Also, we describe the well-known algorithms of Kruskal and Prim for finding minimum spanning trees. For further details and data structures for efficiently implementing these algorithms, we refer the reader to Cormen et al [4].

5.1 Applications

We start with some applications of the minimum spanning tree problem.

(a) Minimum-cost road interconnection network:
There are n towns in a region. For certain pairs i and j it is feasible to build a direct road between i and j, and there is a cost c_{ij} incurred if the road ij is built. The problem is to construct enough roads so that every pair of towns can communicate (perhaps indirectly), and the total construction cost must be minimized.

(b) Finding routes with maximum bottleneck capacity:
There are n computers connected by a network such that for certain pairs i and j there is a direct link with a capacity of c_{ij} bits/second. For each pair of computers, the problem is to find a path between them (i.e., a sequence of direct links) such that the bottleneck capacity (i.e., the smallest link capacity in the path) is as large as possible. (See Problem 3 for more details.)

(c) Reducing data storage:
There is a 2-dimensional array such that the rows have similar entries and differ only at a few places. Let c_{ij} denote the number of different entries in rows i and j. The problem is to store the array using a small amount of storage space. One solution is to store a reference row i completely, and for the remaining rows j to store only the positions and entries where rows i and j differ.

(d) Cluster analysis:
Given a set of n data points, the problem is to partition it into "clusters" such that data points within a cluster are "closely related" to each other. Kruskal's minimum spanning tree algorithm (see Section 5.4) maintains "clusters" and at each iteration "merges" the "closest" two clusters. The algorithm starts with n clusters, and ends with one. Each stage of the algorithm gives a partition of the data points into clusters. Consequently, several solutions to the clustering analysis problem can be found by running Kruskal's algorithm on the given data points.

5.2 Trees and cuts

This section has some fundamental results on trees and cuts that are used in this chapter and the later chapters. We recall a few definitions from Chapter 1. A tree is a connected graph that has no cycles. Given a node set $Q \subseteq V$, $\delta(Q)$ denotes the set of all edges with one end in Q and the other end in $V \backslash Q$. A *cut* consists of all edges that have one end in Q and the other end in $V \backslash Q$, where Q is a node set. this cut is denoted $(Q, V \backslash Q)$. Clearly, if $\emptyset \neq Q \neq V$, then $\delta(Q) = (Q, V \backslash Q)$.

Proposition 5.1 *A graph $G = (V, E)$ is connected if and only if for every node set $Q \subset V$, $\emptyset \neq Q \neq V$, $\delta(Q) \neq \emptyset$.*

Proof: Suppose that there is a node set Q, $\emptyset \neq Q \neq V$, with $\delta(Q) = \emptyset$. Then G is not connected because for any node $v \in Q$ and any node $w \in V \backslash Q$ (both v and w exist) there is no path from v to w in G. For the other direction, start with any node v; let $Q = \{v\}$ and let F be an edge set that is initially empty. As long as $Q \neq V$, there is an edge qz in $\delta(Q)$ with $q \in Q$ and $z \notin Q$. Repeatedly, add the edge qz to F and add z to Q. By induction on the number of steps, observe that the subgraph (Q, F) contains a path from the start node v to each node $q \in Q$. Hence, when $Q = V$, then (Q, F) is a connected spanning subgraph of G, implying that G is connected. (In fact, the final F is (the edge set of) a spanning tree of G.) ∎

Theorem 5.2 *Let $T = (V, F)$ be a graph. The following statements are equivalent:*

(a) T is a tree, or equivalently, T is connected and has no cycles.

(b) There is exactly one path between each pair of nodes in T.

(c) T contains $|V| - 1$ edges and is connected.

(d) T contains $|V| - 1$ edges and has no cycles.

Theorem 5.3 *Let $G = (V, E)$ be a graph, and let $F \subseteq E$ be an edge set such that $T = (V, F)$ is a spanning tree of G.*

(a) For every edge $e \in F$ (e is a tree edge) the subgraph $T \backslash \{e\} = (V, F \backslash \{e\})$ is a forest consisting of two trees, say, $T_1 = (V_1, F_1)$ and $T_2 = (V_2, F_2)$ (i.e., removing e disconnects T into two trees). The node sets V_1, V_2 form a partition of V. For every edge $\bar{e}$ in the cut (V_1, V_2), the subgraph $(T \backslash \{e\}) \cup \{\bar{e}\} = (V, (F \backslash \{e\}) \cup \{\bar{e}\})$ is a spanning tree of G (i.e., adding $\bar{e}$ joins the two trees into one).

(b) If e' is an edge in $E \backslash F$ (e' is a nontree edge), then the subgraph $T' = (V, F \cup \{e'\})$ has exactly one cycle (i.e., adding e' to T gives one cycle). The cycle consists of e' and the unique path in T between the ends of e'.

(c) Let T' be the subgraph in part (b). For every edge e in the unique cycle of T' the subgraph $T' \backslash \{e\} = (V, (F \cup \{e'\}) \backslash \{e\})$ is a spanning tree of G (i.e., removing e from T' gives a spanning tree).

5.3 Minimum spanning trees

Let $G = (V, E)$ be a graph, and let each edge $ij \in E$ have a real-valued cost $c(ij)$. We also use c_{ij} to denote the cost. The cost of a subgraph $G' = (V', E')$, denoted $c(G')$ or $c(E')$, is defined to be the sum of the costs of all edges in G', $\sum_{e \in E'} c(e)$. A *minimum spanning tree* of G is defined to be a spanning tree of G whose cost is minimum. Recall that a spanning tree is a connected subgraph of G that has no cycles.

The next two results give two alternative characterizations of minimum spanning trees.

Proposition 5.4 *(**Cut optimality condition**) A spanning tree $T = (V, F)$ of a graph $G = (V, E)$, c is a minimum spanning tree w. r. t. c if and only if for every edge $ij \in F$ and every edge $k\ell$ in the cut (V_1, V_2), where V_1 and V_2 are the node sets of the two trees in $T \backslash \{ij\}$,*

$$c_{ij} \leq c_{k\ell}.$$

Proof: Suppose that T is a minimum spanning tree of G. Let ij be any edge in F, and let $k\ell$ be any edge such that nodes k and ℓ are in different trees of the subgraph $T \backslash \{ij\}$ (i.e., $k\ell$ is as in the proposition). By Theorem 5.3 the subgraph $H = (T \backslash \{ij\}) \cup \{k\ell\}$ is a spanning tree of G, so $c(H) \geq c(T)$. This implies that $c_{ij} \leq c_{k\ell}$.

For the other direction, suppose that T satisfies the cut optimality condition but T is not a minimum spanning tree of G. Let T^* be a minimum spanning tree of G that has as many edges as possible in common with T, i.e., T^* maximizes $|E(T^*) \cap E(T)|$ among all minimum spanning trees of G. Let ij be an edge of T that is not in T^* (ij exists since $T \neq T^*$). Let (V_1, V_2) be the cut formed by removing ij from T, where V_1 and V_2 are the node sets of the two trees in $T \backslash \{ij\}$, and let i be in V_1 and let j be in V_2. Consider the unique path in T^* from i to j. At least one edge $k\ell$ in this path is in the cut (V_1, V_2) (otherwise all nodes in the path would be in V_1). Since T satisfies the cut optimality condition, $c_{ij} \leq c_{k\ell}$. Hence, removing the edge $k\ell$ from T^* and adding the edge

ij gives a new subgraph $T^{**} = (T^*\backslash\{k\ell\}) \cup \{ij\}$ that has $c(T^{**}) \leq c(T^*)$. T^{**} is a spanning tree by Theorem 5.3. This contradicts our definition of T^*, since T^{**} is a minimum spanning tree of G that has one more edge in common with T than T^*. ∎

Proposition 5.5 *(***Path optimality condition***) A spanning tree $T = (V, F)$ of a graph $G = (V, E)$, c is a minimum spanning tree w. r. t. c if and only if for every edge $ij \in E\backslash F$ (ij is a nontree edge) and every edge $k\ell$ in the unique path of T between i and j*

$$c_{ij} \geq c_{k\ell} \ .$$

Proof: Suppose that T is a minimum spanning tree of G. Let ij be any edge in $E\backslash F$, and let $k\ell$ be any edge in the unique path of T between i and j. By Theorem 5.3 the subgraph $H = (T \cup \{ij\})\backslash\{k\ell\}$ is a spanning tree of G, so $c(H) \geq c(T)$. This implies that $c_{ij} \geq c_{k\ell}$.

The other direction can be proved in one of two ways:

(i) Similarly to the previous proposition, assume that T satisfies the path optimality condition but T is not a minimum spanning tree of G. Let T^* be a minimum spanning tree of G that has as many edges as possible in common with T, let ij be an edge of T^* that is not in T, and let $k\ell$ be an edge of the unique i-j path of T that is in the cut given by $T^*\backslash\{ij\}$ (i.e., $k\ell \in F$ and $k\ell \in (V_1, V_2)$, where V_1 and V_2 are the node sets of the two trees in $T^*\backslash\{ij\}$.) Then the spanning tree $(T^*\backslash\{ij\}) \cup \{k\ell\}$ gives the desired contradiction.

(ii) Alternatively, show that if T satisfies the path optimality condition, then it satisfies the cut optimality condition (we skip the details). Then the proof is completed by applying the previous proposition. ∎

Proposition 5.6 *Let $T = (V, F)$ be a minimum spanning tree of $G = (V, E)$, and let F' be a subset of F. Let $(S, V\backslash S)$ be a cut that contains no edge of F', and let ij be an edge of minimum cost in this cut. Then there exists a minimum spanning tree of G that contains all edges in $F' \cup \{ij\}$.*

Proof: If the edge ij is in F, then the proof is done. Otherwise, T contains at least one edge pq such that pq is in the cut $(S, V\backslash S)$ (since T is connected) and $pq \neq ij$. By the proposition $c_{ij} \leq c_{pq}$. Then the subgraph $H = (T\backslash\{pq\}) \cup \{ij\}$ is a spanning tree (by Theorem 5.3) and $c(H) \leq c(T)$. That is, H is a minimum spanning tree whose edge set contains $F' \cup \{ij\}$. ∎

5.4 Algorithms for minimum spanning trees

This section presents two efficient algorithms for constructing minimum spanning trees, namely, Kruskal's algorithm and Prim's algorithm. Kruskal's algorithm is also called the greedy algorithm. Given a set of objects, the greedy algorithm attempts to find a feasible subset with minimum (or maximum) objective value by repeatedly choosing an object of minimum (maximum) cost from among the unchosen ones and adding it to the current subset provided the resulting subset is feasible. In particular, Kruskal's algorithm works by repeatedly choosing an edge of minimum cost among the edges not chosen so far, and adding this edge to the "current spanning forest" provided this does not create a cycle. The algorithm terminates when the current spanning forest becomes connected.

Algorithm Kruskal's Minimum Spanning Tree Algorithm
input: Connected graph $G = (V, E)$ and edge costs $c : E \to \Re$.
output: Edge set $F \subseteq E$ of minimum spanning tree of G.

$F := \emptyset$; (F is the edge set of the current spanning forest)
linearly order the edges in E according to nondecreasing cost;
let the ordering be $e_1, e_2, \ldots, e_{|E|}$;
for each edge e_i, $i = 1, 2, \ldots, |E|$, **do**
    if $F \cup \{e_i\}$ has no cycle
    then $F := F \cup \{e_i\}$; (add the edge to the current forest)
        if $|F| = |V| - 1$ **then stop** and **output** F; **end**;
    end; (if)
end; (for)

Theorem 5.7 *Kruskal's algorithm is correct and it finds a minimum spanning tree. Its running time is $O(|V||E|)$.*

Proof: At termination of the algorithm, F is (the edge set of) a spanning tree since it has $|V| - 1$ edges and contains no cycle (see Theorem 5.2). Further, F satisfies the path optimality condition: Consider any edge ij that is not in F. At the step when the algorithm examined ij, F contained the edges of a path between i and j. Each edge kl in this path has $c_{kl} \leq c_{ij}$, since kl is examined before ij. This shows that the path optimality condition holds. Then by Proposition 5.5, F is a minimum spanning tree. ∎

Remark: By using appropriate data structures, the running time of Kruskal's algorithm can be improved to $O(|E| \log |V|)$.

Prim's algorithm starts with a "current tree" T that consists of a single node. In each iteration, a minimum-cost edge in the "boundary" $\delta(V(T))$ of T is added to T, and this is repeated till T is a spanning tree.

Algorithm Prim's Minimum Spanning Tree Algorithm
input: Connected graph $G = (V, E)$ and edge costs $c : E \to \Re$.
output: Minimum spanning tree $T = (S, F)$ of G.

$F := \emptyset$; (F is the edge set of the current tree T)
$S := \{v\}$, where v is an arbitrary node; (S is the node set of T)
while $S \neq V$ **do**
    among the edges having exactly one end in S, find an edge ij of minimum cost;
    $F := F \cup \{ij\}$;
    $S := S \cup (\{i, j\} \backslash S)$; (add the end node in $V \backslash S$)
end; (while)

Theorem 5.8 *Prim's algorithm is correct and it finds a minimum spanning tree. Its running time is* $O(|V|^2)$.

Remark: By using the Fibonacci heaps data structure, the running time of Prim's algorithm can be improved to $O(|E| + |V| \log |V|)$.

5.5 LP formulation of the minimum spanning tree problem

We start with an integer linear programming formulation, called (IP), of the minimum spanning tree problem, and then study the linear programming (LP) relaxation of (IP). Usually, when the integrality constraints of an integer program are relaxed to give a linear program, then the feasible region becomes larger, and so the optimal solution of the linear program may be considerably better than that of the integer program. Surprisingly, Theorem 5.9 below shows that an optimal solution of (IP) is also an optimal solution of the LP relaxation. In other words, the LP relaxation exactly formulates the minimum spanning tree problem.

First, introduce a zero-one variable x_{ij} for each edge ij in the given graph $G = (V, E)$. The intention is that the set of edges whose variables take on the value one, $F = \{ij \in E : x_{ij} = 1\}$, should form (the edge set of) a spanning tree. By Theorem 5.2, this can be achieved by ensuring that $|F| = \sum_{ij \in E} x_{ij} = |V| - 1$ and F contains no cycle. To impose the latter condition, we use the so-called *subtour-elimination constraints* :

$$\sum_{ij \in E: i \in S, j \in S} x_{ij} \leq |S| - 1, \qquad \forall S \subseteq V.$$

To see that these constraints "eliminate" all cycles in $F = \{ij \in E : x_{ij} = 1\}$, suppose that there is a cycle $D = v_1 \ldots v_\ell v_1$ ($v_{\ell+1} = v_1$) such that for each edge $e = v_i v_{i+1}$, $1 \leq i \leq \ell$, in the cycle $x_e = 1$; then for the set $S = V(D) = \{v_1, \ldots, v_\ell\}$, we have $\sum_{ij \in E:\, i \in S, j \in S} x_{ij} \geq \ell = |S|$, i.e., the subtour-elimination constraint for $S = V(D)$ is violated.

The constraint $\sum_{ij \in E} x_{ij} = |V| - 1$ is implied by the first constraint in (IP) together with the subtour-elimination constraint for $S = V$.

$$\boxed{\begin{array}{llrcll}
\text{(IP)} & \text{minimize} & \sum_{ij \in E} c_{ij} x_{ij} & & & \\
 & \text{subject to} & \sum_{ij \in E} x_{ij} & \geq & |V| - 1 & \\
 & & \sum_{ij \in E:\, i \in S, j \in S} x_{ij} & \leq & |S| - 1, & \forall S \subseteq V \\
 & & x_{ij} & \in & \{0, 1\}, & \forall ij \in E
\end{array}}$$

The LP relaxation of (IP) is obtained by replacing the integrality constraints $x_{ij} \in \{0, 1\}$ ($\forall ij \in E$) by the constraints $x_{ij} \geq 0$ ($\forall ij \in E$). The constraints $x_{ij} \leq 1$ need not be added explicitly, since they are implied by the subtour-elimination constraints for node sets S of size two. The LP and its dual LP (in standard form) are given below; n denotes $|V|$.

$$
\begin{array}{|llrcll|}
\hline
\text{(LP)} & \text{minimize} & \displaystyle\sum_{ij\in E} c_{ij}x_{ij} & & & \\
 & \text{subject to} & \displaystyle\sum_{ij\in E} x_{ij} & \geq & n-1 & \\
 & & \displaystyle\sum_{ij\in E:\ i\in S, j\in S} -x_{ij} & \geq & -|S|+1, & \forall S\subseteq V \\
 & & x_{ij} & \geq & 0, & \forall ij\in E \\
\hline
\end{array}
$$

$$
\begin{array}{|llrcll|}
\hline
\text{(Dual LP)} & \text{maximize} & (n-1)y_1 - \displaystyle\sum_{S\subseteq V}(|S|-1)y_S & & & \\
 & \text{subject to} & y_1 - \displaystyle\sum_{S\subseteq V:\ S\ni i, S\ni j} y_S & \leq & c_{ij}, & \forall ij\in E \\
 & & y_1 & \geq & 0 & \\
 & & y_S & \geq & 0, & \forall S\subseteq V \\
\hline
\end{array}
$$

Recall from linear programming duality that given a feasible solution $x^* = [x^*_{ij}\ (\forall ij \in E)]$ of (P) and a feasible solution $y^* = [y_1, y_S\ (\forall S \subseteq V)]$ of (D), both are optimal solutions of the respective LPs if and only if the complementary slackness conditions hold. The complementary slackness conditions are as follows:

$$
\begin{array}{lllcll}
\text{[primal]} & x_{ij} > 0 \implies & y_1 - \displaystyle\sum_{S\subseteq V:\ S\ni i, S\ni j} y_S & = & c_{ij}, & \forall ij\in E \\
\text{[dual]} & \left\{\begin{array}{l} y_1 > 0 \implies \\ y_S > 0 \implies \end{array}\right. & \begin{array}{c} \displaystyle\sum_{ij\in E} x_{ij} \\ \displaystyle\sum_{ij\in E:\ i\in S, j\in S} x_{ij} \end{array} & \begin{array}{c} = \\ = \end{array} & \begin{array}{l} |V|-1 \\ |S|-1, \end{array} & \forall S\subseteq V.
\end{array}
$$

Theorem 5.9 *Let T^* be a minimum spanning tree constructed by Kruskal's algorithm, and let x^* be the incidence vector of the edges in T^*, i.e.,*

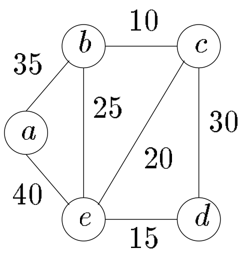
$$
x^*_{ij} = \begin{cases} 1 & \text{if } ij \in E(T^*) \\ 0 & \text{otherwise.} \end{cases}
$$

Then x^* is an optimal solution of the LP relaxation of (IP).

Proof: We construct a feasible solution y^* of the dual LP that satisfies the complementary slackness conditions with respect to x^*. It follows from linear programming duality that x^* and y^* are optimal solutions of the LP relaxation of (IP) and its dual, respectively.

The procedure for constructing y^* is called the *dual greedy algorithm.* Let m denote the number of edges, $|E|$; since G is a connected graph, $m \geq |V|-1$. Order the edges in E in nondecreasing order of the costs, and let the ordering be $e_1, e_2, \ldots, e_m$, where $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$. For each i, $1 \leq i \leq m$, let E_i denote the edge set $\{e_1, \ldots, e_i\}$. For each i, $1 \leq i \leq m$, let S_i denote the node set of the connected component that contains e_i in the subgraph (V, E_i); so $S_m = V$. In general, several indices $i, j, k, \ldots$ may have $S_i = S_j = S_k = \cdots$. Observe that the node sets S_i are explicitly "constructed" during the execution of the greedy algorithm (Kruskal's algorithm):

$G = (V, E)$, c

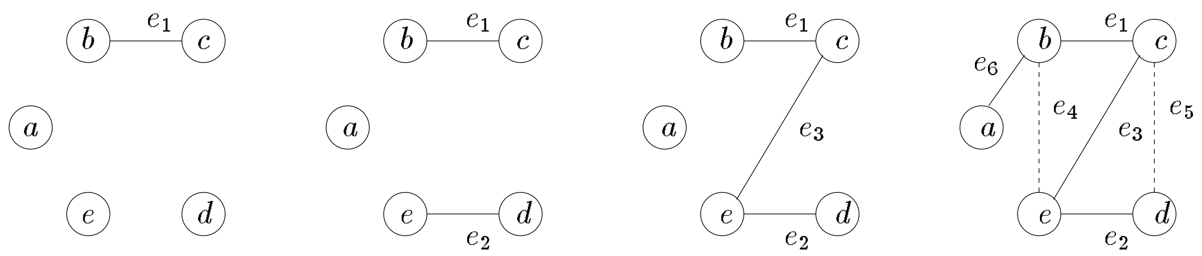


Edges ordered by cost:

e_1, e_2, e_3, e_4, e_5, e_6, e_7

bc, de, ce, be, cd, ab, ae

Execution of Kruskal's algorithm: current spanning forest (V, F)



	S_i : node sets of components of (V, F)			
	$S_1 = \{b, c\}$	$S_2 = \{d, e\}$	$S_3 = \{b, c, d, e\} = S_4 = S_5$	$S_6 = V = \{a, b, c, d, e\}$
$e(S_i)$ last edge of MST with both ends in S_i	bc	de	ce	ab
$f(S_i)$ first edge of MST with exactly one end in S_i	ce	ce	ab	—
$y_{S_i} = c(f(S_i)) - c(e(S_i))$	10	5	15	0
y_1	35, since $c(e(V)) = c(ab) = 35$ is positive			

Figure 5.0: Illustrating the LP formulation of the minimum spanning tree problem on an example. The optimal values of the dual variables y_1 and y_{S_i} are computed from the execution of Kruskal's algorithm.

these node sets are precisely the node sets of the connected components of the current spanning forest (V, F). For each node set $S \subset V$, $S \neq V$, if $S = S_i$ for some i, $1 \leq i \leq m$, then define $e(S)$ to be the edge e_{i^*} where $i^* = \min_{i:\ 1 \leq i \leq m} S_i = S$, and define $f(S)$ to be the edge e_{j^*} where $j^* = \min_{j:\ 1 \leq j \leq m} S_j \supset S, S_j \neq S$. Let $f(V)$ be undefined, and define $e(V)$ to be the edge e_{k^*} where $k^* = \min_{k:\ 1 \leq k \leq m} S_k = V$, i.e., $e(V)$ is the last edge added to the minimum spanning tree T^* by the greedy algorithm. If there is no i, $1 \leq i \leq m$, such that $S = S_i$, then $e(S)$ and $f(S)$ are undefined. Again, $e(S)$ and $f(S)$ have specific meanings in the execution of the greedy algorithm: if at some step of the execution, S is the node set of a connected component of the current spanning forest (V, F), then $e(S) = vw$ denotes the *last* edge added to the current F that has both end nodes v and w in S, and (assuming $S \neq V$) $f(S)$ denotes the *first* edge added to the minimum spanning tree T^* (i.e., the final F) with exactly one end node in S. Fix the dual solution y^* to be

$$
\begin{aligned}
y_S^* &= \begin{cases} c(f(S)) - c(e(S)) & \text{if } S = S_i \ (1 \leq i \leq m) \text{ and } S \neq V \\ -c(e(V)) & \text{if } S = V \text{ and } c(e(V)) \leq 0 \\ 0 & \text{otherwise} \end{cases} \\
y_1^* &= \begin{cases} 0 & \text{if } c(e(V)) \leq 0 \\ c(e(V)) & \text{otherwise.} \end{cases}
\end{aligned}
$$

Clearly, y^* is nonnegative. To prove that y^* satisfies the remaining constraints of the dual LP, consider an arbitrary edge $e_i = vw$. The constraint for vw may be written as

$$\sum_{S \subseteq V:\ S \ni v, S \ni w} y_S^* \ \geq\ -c(vw) + y_1^*. \tag{$*$}$$

Focus on the left-hand side of inequality $(*)$ above. Assume that the edge $e(V)$ has $c(e(V)) \leq 0$; the other case $(c(e(V)) > 0)$ is handled similarly. Then $y_1^* = 0$ and $y_V^* = -c(e(V))$. Define $U_1, U_2, \ldots, U_\ell$ to be the sequence of node sets

$$
\begin{aligned}
U_1 &\equiv S_i \\
U_2 &\equiv S_{i'}, \qquad \text{where } e_{i'} = f(U_1) \\
&\cdots \\
U_{j+1} &\equiv S_k, \qquad \text{where } e_k = f(U_j) \\
&\cdots \\
U_\ell &\equiv V \equiv S_{(k:\ e_k = f(U_{\ell-1}))}.
\end{aligned}
$$

In the execution of the greedy algorithm, $U_1, U_2, \ldots, U_\ell$ $(U_1 = S_i)$ are the node sets of the successive connected components of (V, F) that contain the edge $e_i = vw$; in other words, the step of the execution that adds the edge $f(U_j)$, $1 \leq j < \ell$, to the current spanning subgraph (V, F) "constructs" the node set U_{j+1} by merging U_j with the node set of another connected component of (V, F). Now observe that

$$\sum_{S \subseteq V:\ S \ni v, S \ni w} y_S^* = \sum_{j=1}^{\ell} y_{U_j}^*,$$

because for every node set S, if $S \neq U_j$ (for all j, $1 \leq j \leq \ell$), then either $y^*_S = 0$ (since $S \neq S_i$, for all i, $1 \leq i \leq m$) or S does not contain both the ends v and w of the edge e_i. Inequality (*) follows because

$$\begin{aligned}\sum_{j=1}^{\ell} y^*_{U_j} &= [c(f(U_1)) - c(e(U_1))] + [c(f(U_2)) - c(e(U_2))] + \cdots + \\ &\quad [c(f(U_{\ell-1})) - c(e(U_{\ell-1}))] + [-c(e(V))] = \\ &= -c(e(U_1)) \geq -c(vw),\end{aligned}$$

since for $j = 1, \ldots, \ell - 1$, $f(U_j) = e(U_{j+1})$, and $c(e(S_i)) \leq c(e_i)$.

The primal complementary slackness conditions hold because for every edge $e_i = vw$ in the minimum spanning tree T^*, $c(e(S_i)) = c(e_i)$, therefore y^* satisfies the constraint $(*)$ of the dual LP with equality.

Observe that the dual complementary slackness conditions hold, because for every node set S_i, $1 \leq i \leq m$, there are exactly $|S_i| - 1$ edges of T^* that have both ends in S_i (since for each S_i, the subgraph of T^* induced by S_i is connected); the remaining node sets S have $y^*_S = 0$. This completes the proof of the theorem. ∎

5.6 More LP formulations of the minimum spanning tree problem

We give two more linear programming relaxations of the minimum spanning tree problem that are obtained by relaxing the integrality constraints in two natural integer programming formulations. The second relaxation here, but not the first, is an exact formulation of the minimum spanning tree problem.

Our first integer programming formulation, (IP$_{cut}$), is based on Theorem 5.2(c) and Proposition 5.1: A spanning tree is a connected graph with $|V| - 1$ edges, where a graph is connected iff every node set $Q \subset V$, $\emptyset \neq Q \neq V$, has $|\delta(Q)| \geq 1$.

$$\begin{array}{llrcll}
(\text{IP}_{cut}) & \text{minimize} & \sum_{ij \in E} c_{ij} x_{ij} & & & \\
& \text{subject to} & \sum_{ij \in E} x_{ij} & = & |V| - 1 & \\
& & \sum_{ij \in E:\, ij \in \delta(S)} x_{ij} & \geq & 1, & \forall S \subseteq V,\ \emptyset \neq S \neq V \\
& & x_{ij} & \in & \{0, 1\}, & \forall ij \in E
\end{array}$$

The LP relaxation (LP$_{cut}$) of (IP$_{cut}$) is obtained by replacing the integrality constraints $x_{ij} \in \{0, 1\}$ ($\forall ij \in E$) by the constraints $x_{ij} \geq 0$ ($\forall ij \in E$). In general, the optimal solution x^* of (LP$_{cut}$) may not correspond to a spanning tree, because some of the values x^*_{ij} may be fractional (not integral). In other words, the feasible region of (LP$_{cut}$) may have fractional extreme points.

Our second integer programming formulation, (IP$_{dcut}$), is a "directed" version of (IP$_{cut}$). We obtain a directed graph $D = (V, A)$ from the original graph $G = (V, E)$, by replacing each edge $ij \in E$ by the arcs (directed edges) (i, j) and (j, i); so, $A = \{(i, j),\ (j, i)\ :\ ij \in E\}$. We choose an arbitrary node $r \in V$ to be the "root node". The goal is to find a directed spanning tree $T = (V, F)$

of D rooted at r, that is, to find $F \subseteq A$ such that $|F| = |V| - 1$, and for each node $v \in V \backslash \{r\}$, F contains the arc set of a directed path from r to v. In other words, $|F| = |V| - 1$ and for every node set $S \subseteq V$ $(S \neq V)$ with $r \in S$, F must contain at least one arc from the directed cut $(S, V \backslash S)$. Here, a directed cut $(S, V \backslash S)$ consists of all the arcs $(i, j) \in A$ such that $i \in S$ and $j \in V \backslash S$. Such an arc set F is necessarily the arc set of a directed spanning tree rooted at r, and its undirected version gives the edge set of a spanning tree of G.

$$
\begin{array}{llrcll}
(\mathrm{IP}_{dcut}) & \text{minimize} & \sum_{ij \in E} c_{ij} x_{ij} & & & \\
& \text{subject to} & x_{ij} & = & y_{(i,j)} + y_{(j,i)}, & \forall ij \in E \\
& & \sum_{(i,j) \in A} y_{(i,j)} & = & |V| - 1 & \\
& & \sum_{(i,j) \in A:\ (i,j) \in (S, V \backslash S)} y_{(i,j)} & \geq & 1, & \forall S \subseteq V,\ S \neq V,\ r \in S \\
& & x_{ij} & \geq & 0, & \forall ij \in E \\
& & y_{(i,j)} & \in & \{0,1\}, & \forall (i,j) \in A
\end{array}
$$

The LP relaxation (LP_{dcut}) of (IP_{dcut}), obtained by replacing the integrality constraints $y_{(i,j)} \in \{0,1\}$ $(\forall (i,j) \in A)$ by the constraints $y_{(i,j)} \geq 0$ $(\forall (i,j) \in A)$, gives an exact formulation of the minimum spanning tree problem.

5.7 Exercises

1. Let $G = (V, E)$ be a graph, and let the edges be partitioned into two sets R and B (i.e., the edges are colored either red or blue). Suppose that there exists a spanning tree T_R with $E(T_R) \subseteq R$ (i.e., every edge in T_R is red), and another spanning tree T_B with $E(T_B) \subseteq B$ (i.e., every edge in T_B is blue). Let k be an integer between 0 and $|V| - 1$. Use induction to show that G has a spanning tree T with $|E(T) \cap R| = k$ (i.e., the number of red edges in T is k).

2. Let G be a graph, and let each of its edges have distinct cost, i.e, for any two different edges ij and kl, $c_{ij} \neq c_{kl}$. Let r be a given node of G. Prove or disprove:

   (a) G has a unique minimum spanning tree.

   (b) G has a unique shortest paths tree with root node r.

3. (Bottleneck spanning trees.) Let $G = (V, E)$ be a graph, and let c be a cost function on the edges. A spanning tree such that the maximum cost of an edge in it is as small as possible among all spanning trees of G, c is called a *bottleneck spanning tree.*
   (a) Show that a minimum spanning tree of G, c is also a bottleneck spanning tree.
   (b) Prove or disprove the converse:
   A bottleneck spanning tree of G, c is a minimum spanning tree.

4. (Most vital edge.) Let $G = (V, E)$ be a graph, and let c be a cost function on the edges. Let $mst(G, c)$ denote the cost of a minimum spanning tree of G, c. An edge $e \in E$ is called *vital* if $mst(G', c') > mst(G, c)$, where G', c' is obtained by deleting edge e from G, c (i.e., e is vital

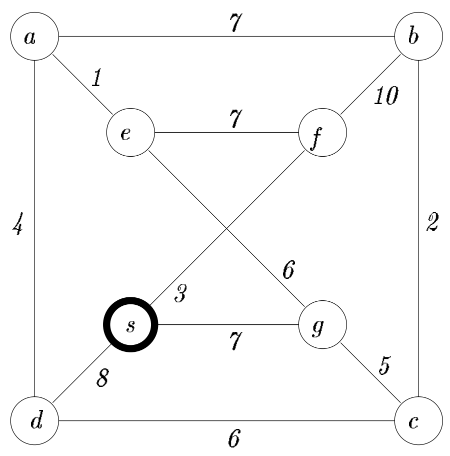


Figure 5.1: Network $G,\ c$ for Problem 5.

if its deletion strictly increases the cost of a minimum spanning tree). A *most vital edge* is a vital edge whose deletion increases the cost of the minimum spanning tree by the maximum amount. Does every network $G,\ c$ contain a vital edge? Suppose that a network contains a vital edge. Describe an efficient algorithm for finding a most vital edge.
(Hint: Use the cut optimality conditions.)

5. Consider the graph and edge costs in Figure 5.1.
   (a) Find a minimum spanning tree using Kruskal's algorithm.
   (b) Find a minimum spanning tree using Prim's algorithm.
   (c) Find a shortest paths tree with root node s using Dijkstra's algorithm.

6. (a) For the graph $G = (V, E)$ and edge costs c given in Figure 5.2, formulate the problem of finding a spanning tree of maximum cost as a linear program.
   (b) Write the dual LP and the complementary slackness conditions for the example in part (a).
   (c) For $G,\ c$ and the LP you wrote in part (a), use the dual greedy method to write down an optimal solution to the dual LP. Verify the complementary slackness conditions.
   (d) Generalize your solution of the previous part to an arbitrary graph G, and arbitrary edge costs c.
   (e) Solve the LP in part (a) using the CPLEX optimizer.

7. (Sensitivity analysis.) Let $G = (V, E)$ be a graph, and let c be a cost function on the edges. Let T^* be a minimum spanning tree of $G,\ c$. For an edge $ij \in E$, define its *cost interval* to be the set of all real numbers γ such that if the cost c_{ij} is changed to γ (but all other edge costs stay the same), then T^* continues to be a minimum spanning tree w.r.t. the new costs.
(a) Describe an efficient method for determining the cost interval of a given edge ij.

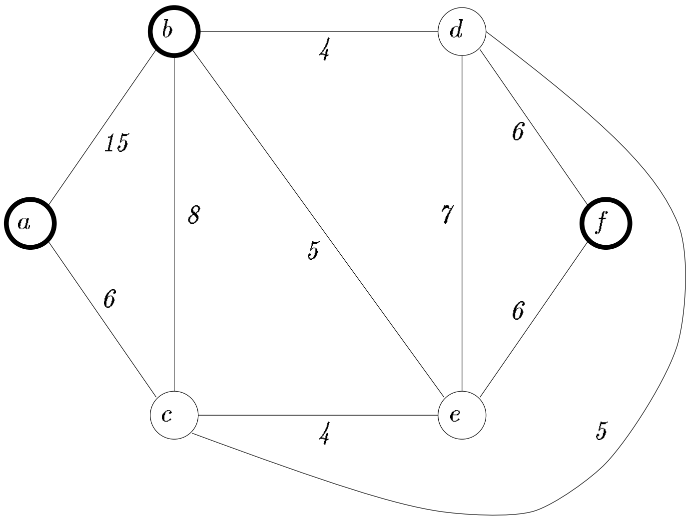


Figure 5.2: Network $G,\ c$ for Problem 6.

(Hint: Consider two cases: when $ij \in T^*$ and when $ij \notin T^*$, and use the cut and path optimality conditions.)
(b) Determine the cost interval of every edge for $G,\ c$ in Figure 5.2.

8. Construct an example to show that the feasible region of the linear programming relaxation (LP_{cut}) (page 73) may have fractional extreme points.
Prove that the feasible region of (LP_{cut}) contains the feasible region of the linear programming relaxation (LP) in Section 5.5, by showing that every feasible solution $x : E \to \Re_+$ of (LP) satisfies all the constraints of (LP_{cut}).

9. Prove that the linear programming relaxation (LP_{dcut}) of (IP_{dcut}) (page 74) gives an exact formulation of the minimum spanning tree problem.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Englewood Cliffs, N. J., 1993.

[2] B. Bollobas, *Graph Theory: An Introductory Course*. Springer-Verlag, New York, 1979.

[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. Macmillan, London, 1976.

[4] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1992.

[5] M. Grötschel, L. Lovász and A. Schrijver, *Geometric algorithms and combinatorial optimization*, (1988) Springer-Verlag, Berlin.

[6] R. C. Larson and A. R. Odoni, *Urban Operations Research*. Prentice-Hall, Englewood Cliffs, N. J., 1981.

[7] T. L. Magnanti and L. A. Wolsey, *Optimal Trees*. CORE discussion paper 9426, C. O. R. E., Universite Catholique de Louvain, Louvain-la-neuve, Belgium, May 1994.