Contents

9 The Minimum Cost Spanning Tree Problem 628
9.1 Some Results on Minimum Cost Spanning Trees 631
9.2 Algorithms for the Minimum Cost Spanning Tree Problem 635
9.3 An Algorithm for Ranking the Spanning Trees in Ascending Order of Cost 645
9.4 Other Types of Minimum Cost Spanning Tree Problems 648
9.5 Exercises 651
9.6 References 664

Chapter 9

The Minimum Cost Spanning Tree Problem

Given a set of points on the two-dimensional plane, the problem of finding the shortest connecting network that connects all the points, using only lines joining pairs of points from the given set arises in many applications. If the length between every pair of points is positive, the shortest connecting network will obviously be a spanning tree, it will be a **minimum length spanning tree**.

The minimum length (or cost) spanning tree problem is one of the nicest and simplest problems in network optimization, and it has a wide variety of applications. The input in this problem is a connected undirected network $G = (\mathcal{N}, \mathcal{A})$ with $c = (c_{ij} : (i; j) \in \mathcal{A})$ as the vector of edge costs or lengths. Define the cost (or length) of a tree in G to be the sum of the costs (or lengths) of edges in it. The problem is to find a minimum cost (or length) spanning tree in G. Applications include the design of various types of distribution networks in which the nodes represent cities, centers etc.; and edges represent communication links (fiber glass phone lines, data transmission lines, cable TV lines, etc.), high voltage power transmission lines, natural gas or crude oil pipelines, water pipelines, highways, etc. The objective is to design a network that connects all the nodes using the minimum length of cable or pipe or other resource. The minimum cost spanning tree problem also appears as a subproblem in algorithms for many routing problems

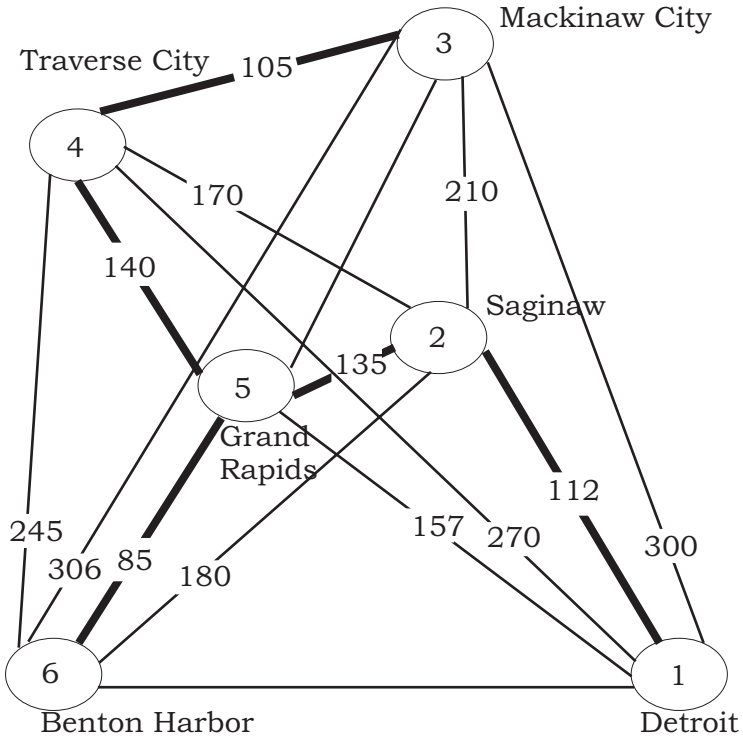


Figure 9.1:

such as the traveling salesman problem.

As an example consider the network G with 6 nodes representing 6 Michigan cities, and edges joining every pair of these nodes, given in Figure 9.1. The number on each edge is its length in miles. The minimum length spanning tree in this network is marked with thick lines.

The network G may in fact be a directed or mixed network. The problem that we consider in this chapter is that of finding a minimum cost spanning tree in G without paying any regard to the orientations of the arcs in G. So, if there are any arcs in the input network, we ignore their orientations and treat every line as an edge. We also assume that the network G has no self loops. If there are r parallel edges joining a pair of nodes in G, any tree can contain at most one of them, and if

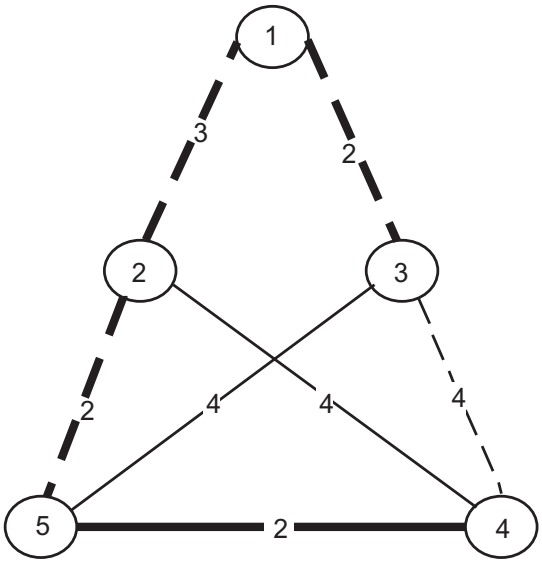


Figure 9.2: The shortest path tree rooted at a node may not be a minimum length spanning tree.

a minimum cost spanning tree contains any of them it will be a least cost edge among them. So, we identify one least cost edge among these r parallel edges and keep it, but delete all the rest. Hence, in the sequel, we assume that there is at most one edge joining any pair of nodes in the input network $G = (\mathcal{N}, \mathcal{A})$. Let $|\mathcal{N}| = n, |\mathcal{A}| = m$. So, $m \leq n(n-1)/2 < n^2/2$.

What we are considering is the **unconstrained minimum cost spanning tree problem**. The **constrained minimum cost spanning problem** with constraints on the type of spanning tree obtained is discussed briefly in Section 9.4.

A Method That Does Not Work

Let the input network be $G = (\mathcal{N}, \mathcal{A}, c)$ with $c > 0$. Consider the following approach. Select any node, say 1, and find the shortest paths from it to all the other nodes in G. This yields a shortest path tree $\mathbb{T}_0$ rooted at 1. For every node $j \neq 1$, the path between 1 and j in

$\mathbb{T}_0$ is a shortest path between these nodes in G. Because of this one may be tempted to conclude that $\mathbb{T}_0$ is a minimum length spanning tree in G. This may not be the case as illustrated in Figure 9.2. Here $\mathbb{T}_0$, consisting of the dashed arcs; has length 11. It is not a minimum length spanning tree; since the spanning tree consisting of the thick edges (dashed or continuous) has length only 9.

9.1 Some Results on Minimum Cost Spanning Trees

Let the input network be $G = (\mathcal{N}, \mathcal{A}, c)$ with $|\mathcal{N}| = n$. No assumptions are made on the entries in c, each of them could be positive, 0, or negative.

THEOREM 9.1 *Let $\mathbb{T}_0$ be a minimum cost spanning tree in* G. *Then every in-tree edge in $\mathbb{T}_0$ must be a minimum cost edge in the fundamental cutset associated with it.*

Proof Let $(\mathbf{X}; \overline{\mathbf{X}})$ be the fundamental cutset associated with the in-tree edge $(i; j)$ in $\mathbb{T}_0$. So, $(i; j)$ is the only in-tree edge in $(\mathbf{X}; \overline{\mathbf{X}})$, and deletion of $(i; j)$ from $\mathbb{T}_0$ disconnects it into two smaller trees, one spanning the nodes in $\mathbf{X}$, and the other spanning the nodes in $\overline{\mathbf{X}}$. Suppose a minimum cost edge in the cutset $(\mathbf{X}; \overline{\mathbf{X}})$ is an out-of-tree edge $(p; q)$ and not $(i; j)$, i.e., $c_{pq} < c_{ij}$. Then replacing $(i; j)$ in $\mathbb{T}_0$ by $(p; q)$ leads to a spanning tree $\mathbb{T}_1$ with cost = cost of $\mathbb{T}_0 - (c_{ij} - c_{pq}) <$ cost of $\mathbb{T}_0$, contradicting the optimality of $\mathbb{T}_0$. Hence $(i; j)$ must be a minimum cost edge in the fundamental cutset $(\mathbf{X}; \overline{\mathbf{X}})$ associated with it in $\mathbb{T}_0$. ∎

THEOREM 9.2 *Let $\mathbb{T}_0$ be a minimum cost spanning tree in* G. *Then every out-of-tree edge must be a maximum cost edge in the fundamental cycle associated with it.*

Proof Let $(p; q)$ be an out-of-tree edge. If there is an in-tree edge $(i; j)$ on the fundamental cycle of $(p; q)$ satisfying $c_{pq} < c_{ij}$, replacing $(i; j)$ in $\mathbb{T}_0$ by $(p; q)$ leads to a new spanning tree with cost = cost of

$\mathbb{T}_0 + (c_{pq} - c_{ij}) <$ cost of $\mathbb{T}_0$, contradicting the optimality of $\mathbb{T}_0$. So, c_{pq} must be $\geqq$ the cost of every in-tree edge in the fundamental cycle of $(p; q)$. ∎

THEOREM 9.3 *Any spanning tree in* G *satisfying*

> *Every out-of-tree edge has maximum cost in its fundamental cycle* wrt *that tree.* (9.1)

is a minimum cost spanning tree.

Proof Let $\mathbb{T}_1$, $\mathbb{T}_2$ be two distinct spanning trees in G satisfying (9.1). We will now prove that the costs of $\mathbb{T}_1$ and $\mathbb{T}_2$ must be equal.

Classify the edges in $\mathbb{T}_1 \cup \mathbb{T}_2$ into 3 classes. The $\mathbb{T}_1$-edges are those which are in $\mathbb{T}_1$ but not in $\mathbb{T}_2$. The $\mathbb{T}_2$-edges are those in $\mathbb{T}_2$ but not in $\mathbb{T}_1$. The $\mathbb{T}_1\mathbb{T}_2$-edges are those which are in both $\mathbb{T}_1$ and $\mathbb{T}_2$. Since $\mathbb{T}_1$ and $\mathbb{T}_2$ are distinct, both the sets of $\mathbb{T}_1-$, $\mathbb{T}_2$-edges are nonempty.

Select any $\mathbb{T}_2$-edge, say $(r; s)$. Let $\mathcal{P}_1$ be the unique path in $\mathbb{T}_1$ between r, s. Then $\mathbb{C}_1 = \{(r; s)\} \cup \mathcal{P}_1$ is the fundamental cycle of $(r; s)$ wrt $\mathbb{T}_1$. All edges on $\mathcal{P}_1$ are in $\mathbb{T}_1$, and at least one of them is not in $\mathbb{T}_2$, as otherwise $\mathbb{T}_2$ contains the entire simple cycle $\mathbb{C}_1$, a contradiction since $\mathbb{T}_2$ is a tree. So, there are some $\mathbb{T}_1$- edges on $\mathcal{P}_1$. Since $\mathbb{T}_1$ satisfies (9.1), all these $\mathbb{T}_1$ edges on $\mathcal{P}_1$ have cost coefficients $\leqq c_{rs}$.

Claim: At least one of the $\mathbb{T}_1$-edges on $\mathcal{P}_1$ has cost coefficient $= c_{rs}$.

We will prove the claim by contradiction. Suppose each $\mathbb{T}_1$-edge on $\mathcal{P}_1$ has cost $< c_{rs}$. Let $(p; q)$ be a $\mathbb{T}_1$-edge on $\mathcal{P}_1$. So, $c_{pq} < c_{rs}$. $(p; q)$ is an out-of-tree edge for $\mathbb{T}_2$, let $\mathcal{S}_{pq}$ be the unique path between p and q in $\mathbb{T}_2$. Since $\mathbb{T}_2$ satisfies (9.1), each edge $(i; j)$ on $\mathcal{S}_{pq}$ satisfies $c_{ij} \leqq c_{pq} < c_{rs}$. Hence $\mathcal{S}_{pq}$ does not contain the edge $(r; s)$. Replace the $\mathbb{T}_1$-edge $(p; q)$ on $\mathcal{P}_1$ by the path $\mathcal{S}_{pq}$, but continue denoting it by the same symbol $\mathcal{P}_1$. If there are other $\mathbb{T}_1$-edges remaining on $\mathcal{P}_1$, carry out the same procedure for them. When this process is completed, the path $\mathcal{P}_1$ gets converted into a path from r to s, not containing the edge $(r; s)$, but completely contained in $\mathbb{T}_2$, call it $\mathcal{P}$ at this stage. So, $\mathbb{T}_2$ contains the edge $(r; s)$ and the path $\mathcal{P}$ from r to s not containing the

edge $(r; s)$, a contradiction since $\mathbb{T}_2$ is a tree. So, the claim must be true.

So, at least one of the $\mathbb{T}_2$-edges on $\mathcal{P}_1$ has cost $= c_{rs}$, suppose it is $(j_1; j_2)$. Replace $(j_1; j_2)$ from $\mathbb{T}_1$ by $(r; s)$. This leads to a new spanning tree $\mathbb{T}_1'$ with the same cost as $\mathbb{T}_1$, but it has one more edge in common with $\mathbb{T}_2$, and it can be verified that it also satisfies (9.1). If $\mathbb{T}_1'$ and $\mathbb{T}_2$ are distinct, repeat this process again with them. After each repetition we get another spanning tree satisfying (9.1) and having the same cost as $\mathbb{T}_1$, but containing one more edge in common with $\mathbb{T}_2$. So, after at most $n - 1$ repetitions of this process it must lead to $\mathbb{T}_2$. Hence $\mathbb{T}_1$ and $\mathbb{T}_2$ have the same cost.

There are only a finite number of spanning trees in G, and hence a minimum cost spanning tree exists in G. If $\mathbb{T}_0$ is a minimum cost spanning tree, it satisfies (9.1) by Theorem 9.2. But we have just proved that any pair of spanning trees satisfying (9.1) have the same cost. Hence every spanning tree in G satisfying (9.1) is a minimum cost spanning tree. ∎

THEOREM 9.4 *Let* $\mathbf{F}$ *be the set (possibly empty) of edges in a forest in* G. *Let* $(\mathbf{X}; \overline{\mathbf{X}})$ *be a cut in* G *containing none of the edges from* $\mathbf{F}$. $(p; q)$ *is a minimum cost edge in the cut* $(\mathbf{X}; \overline{\mathbf{X}})$. *Consider the problem of finding a minimum cost tree among the spanning trees of* G *containing all the edges in* $\mathbf{F}$. *There is a minimum cost spanning tree for this problem which contains the edge* $(p; q)$.

Proof Since G is connected and $\mathbf{F}$ is the set of edges in a forest, there exist spanning trees of G which contain all the edges in $\mathbf{F}$; let $\mathbb{T}_0$ be a minimum cost one among them. If $\mathbb{T}_0$ contains $(p; q)$ we are done. Suppose $(p; q) \notin \mathbb{T}_0$. Let $\mathbb{C}$ be the fundamental cycle of $(p; q)$ wrt $\mathbb{T}_0$. Since $\mathbb{C}$ is a cycle and it contains the edge $(p; q)$ from the cut $(\mathbf{X}; \overline{\mathbf{X}})$, it must contain at least another edge, say $(r; s)$ from $(\mathbf{X}; \overline{\mathbf{X}})$. By the hypothesis $(\mathbf{X}; \overline{\mathbf{X}})$ contains no edges from $\mathbf{F}$, so $(r; s) \notin \mathbf{F}$ and since $(p; q)$ is a minimum cost edge in $(\mathbf{X}; \overline{\mathbf{X}})$, $c_{pq} \leq c_{rs}$. Let $\mathbb{T}_1$ be the spanning tree obtained by replacing $(r; s)$ in $\mathbb{T}_0$ with $(p; q)$. Since $(r; s) \notin \mathbf{F}$, $\mathbb{T}_1$ contains all the edges in $\mathbf{F}$, and also the edge $(p; q)$. If $c_{pq} < c_{rs}$, the cost of $\mathbb{T}_1$ would be $<$ cost of $\mathbb{T}_0$, contradicting the definition of $\mathbb{T}_0$. So, c_{pq} must equal c_{rs}, and $\mathbb{T}_1$ is also a minimum cost

one among the spanning trees of G containing all the edges in $\mathbf{F}$, and it contains $(p; q)$, proving the theorem. ∎

COROLLARY 9.1 *Let $(p; q)$ be a minimum cost edge in a cut in* G. *Then there is a minimum cost spanning tree containing $(p; q)$.*

Proof Follows from Theorem 9.4 with $\mathbf{F} = \emptyset$. ∎

COROLLARY 9.2 *Suppose $\mathbf{F} \subset \mathcal{A}$ satisfies the property that there exists a minimum cost spanning tree in G containing all the edges in $\mathbf{F}$. Let $(p; q)$ be a minimum cost edge in a cut in G which contains no edges from $\mathbf{F}$. Then, there exists a minimum cost spanning tree in G containing all the edges in $\mathbf{F} \cup \{(p; q)\}$.*

Proof Follows from Theorem 9.4. ∎

THEOREM 9.5 *Let $\mathbf{F} \subset \mathcal{A}$ be the set of edges in a forest in a connected undirected network* G. *Let $\mathbb{C}$ be a simple cycle in* G, *and $(r; s)$ a maximum cost edge among those edges in $\mathbb{C}$ not in $\mathbf{F}$. Consider the problem of finding a minimum cost spanning tree among those spanning trees of* G *containing all the edges in $\mathbf{F}$. There is a minimum cost spanning tree for this problem which does not contain $(r; s)$.*

Proof Let $\mathbb{T}_0$ be a minimum cost spanning tree in G among those containing all the edges in $\mathbf{F}$. If $(r; s) \notin \mathbb{T}_0$ we are done. Suppose $\mathbb{T}_0$ contains $(r; s)$. Let $(\mathbf{X}; \overline{\mathbf{X}})$ be the fundamental cutset of $(r; s)$ wrt $\mathbb{T}_0$. So, $(r; s)$ is the only in-tree arc in $(\mathbf{X}; \overline{\mathbf{X}})$, and hence this cut contains no arcs from $\mathbf{F}$. Since $(\mathbf{X}; \overline{\mathbf{X}})$ contains $(r; s)$ from $\mathbb{C}$, it must contain at least one other edge from $\mathbb{C}$, suppose it is $(p; q)$. So, $(p; q)$ is an out-of-tree edge on $\mathbb{C}$, and since $(p; q) \notin \mathbf{F}$ we have $c_{rs} \geq c_{pq}$. Let $\mathbb{T}_1$ be the spanning tree obtained by replacing $(r; s)$ from $\mathbb{T}_0$ by $(p; q)$. $\mathbb{T}_1$ contains all the edges in $\mathbf{F}$ and does not contain $(r; s)$. If $c_{rs} > c_{pq}$, cost of $\mathbb{T}_1 <$ cost of $\mathbb{T}_0$, contradicting the definition of $\mathbb{T}_0$. So, we must have $c_{rs} = c_{pq}$, and $\mathbb{T}_1$ is a minimum cost spanning tree among those containing all the edges in $\mathbf{F}$, and does not contain $(r; s)$. ∎

Exercises

9.1 Prove the following converse of Theorem 9.1. If $\mathbb{T}$ is a spanning tree in G satisfying

$$\text{each edge in } \mathbb{T} \text{ is a min. cost edge in its fundamental cutset } \mathsf{wrt}\ \mathbb{T} \tag{9.2}$$

then $\mathbb{T}$ is a minimum cost spanning tree in G.

9.2 Let $G = (\mathcal{N}, \mathcal{A})$ be an undirected connected network with $|\mathcal{N}| = n$. With each spanning tree $\mathbb{T}$ in G associate its 0-1 incidence vector defined on $\mathcal{A}$ to be the vector $(a_{ij} : (i;j) \in \mathcal{A})$ where $a_{ij} = 1$ if $(i;j)$ is an in-tree edge, 0 otherwise. Consider the following system of constraints in variables $(x_{ij} : (i;j) \in \mathcal{A})$

$$\begin{aligned} \sum(\quad x_{ij} \quad &: \text{ over } (i;j) \in \mathcal{A}) && = n-1 \\ \sum(\quad x_{ij} \quad &: \text{ over } (i;j) \in \mathcal{A}; i,j \in \mathbf{J}) \overset{\leq}{=} |\mathbf{J}| - 1, \\ &\quad \text{for all } \mathbf{J} \subset \mathcal{N} \text{ with } |\mathbf{J}| \overset{\geq}{=} 2 && \\ x_{ij} \quad &\overset{\geq}{=} 0, \text{ for all } (i;j) \in \mathcal{A} && \end{aligned} \tag{9.3}$$

Prove that $\mathbf{K}$, the set of feasible solutions of (9.3), is the convex hull of the 0-1 incidence vectors of all the spanning trees in G. Also prove that the 0-1 incidence vector of any spanning tree in G is an extreme point of $\mathbf{K}$ and conversely.

Given two extreme points of $\mathbf{K}$, derive necessary and sufficient conditions for them to be adjacent, based on the properties of spanning trees corresponding to them.

9.2 Algorithms for the Minimum Cost Spanning Tree Problem

All efficient algorithms known for this problem are based on an incremental approach which builds up a minimum cost spanning tree, edge

by edge, called the **greedy method**. It is so named because at each stage it selects the next edge to be the cheapest that can be selected at that time. The greedy method is very naive, once a decision to include an edge is made, that decision is never reversed (so, no backtracking), and the selection at each stage is based on the situation at that time without any features of look-ahead etc. Hence, the method is also known as a **myopic method**. The fact that such a simple approach solves the minimum cost spanning tree problem indicates the very nice structure and the simplicity of the problem.

All the algorithms begin with the trivial spanning forest in G consisting of isolated nodes: ($\{1\}$, $\emptyset$), ..., ($\{n\}$, $\emptyset$). In each step they select one or more edges merging the forest components. Finally, after $(n-1)$ edges are selected, all the forest components merge into a single spanning tree, which will be a minimum cost spanning tree. We will denote the set of trees in the forest in a general stage by $\mathbf{F}_1 = (\mathcal{N}_1, \mathcal{A}_1), \ldots, \mathbf{F}_l = (\mathcal{N}_l, \mathcal{A}_l)$, where each of $\mathbf{F}_1, \ldots, \mathbf{F}_l$ is a tree; and $\mathcal{N}_1, \ldots, \mathcal{N}_l$ is a partition of $\mathcal{N}$. $\mathbf{F}_1, \ldots, \mathbf{F}_l$ are the connected components in the forest at this stage.

The first algorithm that we discuss is known as **Prim's algorithm** (Jarnik [1930], Prim [1957], Dijkstra [1959 of Chapter 4]). It grows only one component in the forest as a connected tree, leaving all the other components as isolated nodes. So, we will call the component being grown as the "tree." Select a node arbitrarily, say node 1, and make it the root node of the tree. Since the network is undirected, we maintain the predecessor indices of the nodes in the tree without any + or − signs. Nodes in the tree are labeled with predecessor indices, which are called **permanent labels**. Each out-of-tree node j carries a **temporary label** (which can change from step to step until this node joins the tree) of the form (P_j, d_j), where $d_j = +\infty$ and $\mathrm{P}_j = \emptyset$ if there is no edge joining j with an in-tree node so far; and $d_j = \min. \{c_{ij}$: over i an in-tree node such that $(i;j) \in \mathcal{A}\}$ and P_j is an i that attains this minimum, otherwise. So, for an out-of-tree node j with temporary label (P_j, d_j), if $d_j < \infty$, P_j is a "nearest" in-tree node to j at this stage. We will now describe Prim's algorithm.

PRIM'S ALGORITHM

Initialization Permanently label the root node 1 with the label $\emptyset$. For each node j such that $(1; j) \in \mathcal{A}$ $[(1; j) \notin \mathcal{A}]$ temporary label j with $(1, c_{1j})$ $[(\emptyset, \infty)]$.

General step Among all out-of-tree nodes j at this stage, find one with the smallest d_j, suppose it is r with temporary label (P_r, d_r). Delete this temporary label on r and give it the permanent label P_r (its predecessor index in the tree), i.e., make (P_r, r) an in-tree edge and r an in-tree node. If there are no out-of-tree nodes left, terminate, the spanning tree defined by the permanent labels is a minimum cost spanning tree in G. If there are some out-of-tree nodes left update their temporary labels as follows: for each out-of-tree node j with temporary label (P_j, d_j), change it to (r, c_{rj}) only if $(r; j) \in \mathcal{A}$ and $c_{rj} < d_j$; leave it unchanged otherwise. Then go to the next step.

Discussion

At some stage in the algorithm let $\mathbf{X}$ be the set of in-tree nodes and $\overline{\mathbf{X}}$ its complement. So, there are no in-tree edges in the cut ($\mathbf{X}$; $\overline{\mathbf{X}}$) at this stage. The edge selected at this stage for being the next in-tree edge is a minimum cost edge in the cut ($\mathbf{X}$, $\overline{\mathbf{X}}$) by the manner in which it is selected. This is a **greedy selection**, that's what makes this a greedy method. Applying Corollaries 9.1, 9.2 in each step, we conclude that at every stage there is a minimum cost spanning tree in G containing all the in-tree edges at that stage. Hence when the tree becomes a spanning tree in G, it is a minimum cost spanning tree in G.

The computational effort in each step is clearly $\mathrm{O}(n)$ and there are exactly $n-1$ steps. Hence the overall computational complexity of this algorithm is $\mathrm{O}(n^{\mathbf{2}})$.

As an example we apply this algorithm on the network in Figure 9.1. We select node 1 as the root node. We show the node labels, and the in-tree edge selected in each step in the following table. Permanent labels are shown in bold face. Non-bold face labels are temporary.

Step	Label on node at end of step						In-tree node and in-tree edge selected in step
	1	2	3	4	5	6	
Initial	$\emptyset$	(1,112)	(1,300)	(1,270)	(1,157)	(1,195)	
1	$\emptyset$	**1**	(2,210)	(2,170)	(2,135)	(2,180)	2, (1; 2)
2	$\emptyset$	**1**	(5,206)	(5,140)	**2**	(5,85)	5, (2; 5)
3	$\emptyset$	**1**	(5,206)	(5,140)	**2**	**5**	6, (5; 6)
4	$\emptyset$	**1**	(4,105)	**5**	**2**	**5**	4, (5; 4)
5	$\emptyset$	**1**	**4**	**5**	**2**	**5**	3, (4; 3)

There can be at most $\binom{n}{2}$ edges in G, the network is dense if their number is of this order. Since the cost of each edge has to be examined at least once, for dense networks $O(n^2)$ is the best complexity we can expect for this problem. Hence Prim's algorithm attains the best complexity for this problem on dense networks. However, on sparse networks, i.e., when $|\mathcal{A}| = m$ is much smaller than n^2, it is possible to construct better algorithms. The next algorithm that we discuss is **Kruskal's algorithm**. It begins with the trivial spanning forest ({1}, $\emptyset$), ..., ({n}, $\emptyset$), and it examines the edges in $\mathcal{A}$ in increasing order of cost, one after the other, and either discards it or includes it in the forest. The forest develops in several disconnected components until all of them are connected in the end. The various components may be of different sizes during the algorithm. It stores and updates each component as a tree, and maintains the set of nodes in each component at each stage.

KRUSKAL'S ALGORITHM

Initialization Order the edges in $\mathcal{A}$ in increasing order of cost. Begin with the trivial spanning forest ({1}, $\emptyset$), ..., ({n}, $\emptyset$).

General step Get the next least cost remaining edge in $\mathcal{A}$. Suppose it is $(i; j)$. If both the nodes i, j on this edge lie in the same component of the forest at this stage, discard this edge and go to the next step. If i, j lie in different components of the forest at this stage, include $(i; j)$ as an in-forest edge and merge the two components that it connects into a single component. If there

is only one component now, it is a minimum cost spanning tree, terminate. Otherwise go to the next step.

Discussion

Let $(i;j)$ be the edge that has come up for examination at some stage of this algorithm. Suppose i,j both appear in a component, $\mathbf{F}_r = (\mathcal{N}_r, \mathcal{A}_r)$ say, at this stage. All the arcs in $\mathcal{A}_r$ were examined earlier, so the cost of every edge in $\mathcal{A}_r$ is $\leq c_{ij}$. So, if $\mathbb{C}_1$ is the fundamental cycle of $(i;j)$ wrt $\mathbf{F}_r$, then $(i;j)$ is a maximum cost edge on $\mathbb{C}_1$, and by Theorem 9.5 there exists a minimum cost spanning tree not containing edge $(i;j)$, among all the spanning trees containing all the edges in the forest at this stage.

Suppose i,j appear in different components, $\mathbf{F}_r = (\mathcal{N}_r, \mathcal{A}_r)$, and $\mathbf{F}_s = (\mathcal{N}_s, \mathcal{A}_s)$, say. Then $(i;j)$ is a least cost edge in the cut $(\mathcal{N}_r; \mathcal{N} \backslash \mathcal{N}_r)$, and this cut contains none of the forest edges at this stage. So, by Theorem 9.4, there exists a minimum cost spanning tree containing edge $(i;j)$ among the spanning trees containing all the edges in the forest at this stage.

Applying these arguments in each step from the beginning, we conclude that the spanning tree obtained at the termination of this algorithm is a minimum cost spanning tree in G. It can also be verified that this spanning tree satisfies (9.1).

Computationally, the most expensive operation in this algorithm is that of arranging the edges in $\mathcal{A}$ in ascending order of cost at the beginning, which requires $O(m \log m)$ effort, where $m = |\mathcal{A}|$. Also since $m < n^2$, this is the same as $O(m \log n)$, where $n = |\mathcal{N}|$. This is the worst case computational complexity of this method.

It is not actually necessary to order all the edges in $\mathcal{A}$ in ascending order of cost, since we will actually select only $(n-1)$ of the edges for the final spanning tree. Any partial quick sort scheme which produces the least cost remaining edge would be adequate. An ideal scheme for this is a multipass sorting routine in which each pass produces the next edge in the ordered sequence very efficiently. Even with all these refinements, this method is clearly suitable to solve minimum cost spanning tree problems in relatively sparse networks.

When applied to find a minimum cost spanning tree in the network in Figure 9.1, this algorithm examines the edges in the order (5; 6), (3; 4), (1; 2), (2; 5), (4; 5), includes each of them in the forest and then terminates with the spanning tree consisting of the thick edges.

The next algorithm for this problem that we discuss is **Boruvka's algorithm**, it dates back to 1926, and seems to be one of the earliest algorithms for a network optimization problem. One of the main features of this algorithm is that it grows the forest, with all its components growing simultaneously in every step. This algorithm does not need the edges ordered in any specific order. The operations in this algorithm can be carried out unambiguously if all the edge cost coefficients are distinct. However, if some edge cost coefficients are equal, we need a unique tie breaking rule for the minimum in every pair of edge cost coefficients, that holds throughout the algorithm; otherwise there is ambiguity in the selection of the edges to be included in the forest, and this may result in cycles being formed. So, we adopt the following simple tie breaking strategy. Let $e_1, \ldots, e_m$ be the edges in the order in which they are recorded in the input data, with their cost coefficients $c_1, \ldots, c_m$ respectively. Brand the edges with these numbers. Among a pair of edges, say e_r, e_s; the edge e_r is considered to be the least cost one iff either $c_r < c_s$, or $c_r = c_s$ and $r < s$ (conceptually, this strategy is equivalent to replacing c_p by $c_p + p\epsilon$, for $p = 1$ to m, where ϵ is treated as an arbitrarily small positive parameter without being given any specific value).

BORUVKA'S ALGORITHM

Initialization Begin with the trivial spanning forest ({1}, $\emptyset$), ..., ({n}, $\emptyset$).

General step Let $\mathbf{F}_1 = (\mathcal{N}_1, \mathcal{A}_1), \ldots, \mathbf{F}_l = (\mathcal{N}_l, \mathcal{A}_l)$, be the connected components in the forest at this stage. Find a least cost edge in the cut $(\mathcal{N}_h, \mathcal{N} \backslash \mathcal{N}_h)$, for each $h = 1$ to l. This can be carried out in a single operation of examining all the edges once. Add all these edges to the forest. Determine the connected components in the new forest. If only one, it is a minimum cost spanning tree in G, terminate. Otherwise, go to the next step.

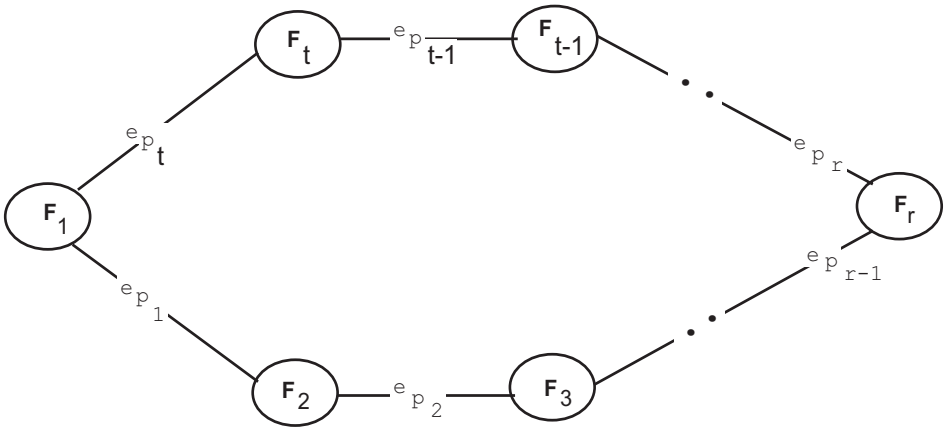


Figure 9.3: A situation for the selected edges that could lead to a cycle in Boruvka's algorithm.

Discussion

We need to show that the addition of all the new edges in a step does not create a cycle. Suppose at the beginning of some step, the trees in the forest are $\mathbf{F}_1 = (\mathcal{N}_1, \mathcal{A}_1), \ldots, \mathbf{F}_l = (\mathcal{N}_l, \mathcal{A}_l)$. Suppose a cycle is created when all the new edges selected in this step are added to this forest. Each of the selected edges joins a node in one of these trees, to a node outside this tree. So, a cycle can only be created if a subset of 2 or more selected edges, $e_{p_1}, \ldots, e_{p_t}$, say, are such that : e_{p_r}, the least cost edge in the cut $(\mathcal{N}_r, \mathcal{N} \backslash \mathcal{N}_r)$, joins $\mathbf{F}_r$ and another tree, $\mathbf{F}_{r+1}$ say, for $r = 1$ to t, with $t + 1$ being 1. So, the trees that these selected edges join are as in Figure 9.3.

Both edges $e_{p_{r-1}}$ and e_{p_r} are in the cut $(\mathcal{N}_r, \mathcal{N} \backslash \mathcal{N}_r)$, but e_{p_r} has been selected as the least cost edge in this step. So, by the arrangement made at the beginning of this algorithm, $c_{p_r} \leqq c_{p_{r-1}}$, and if $c_{p_r} = c_{p_{r-1}}$ then $p_r < p_{r-1}$ for $r = 1$ to t with $t + 1$ being 1. So, $c_{p_t} \leqq c_{p_{t-1}} \leqq \ldots \leqq c_{p_1} \leqq c_{p_t}$ which implies that $c_{p_t} = c_{p_{t-1}} = \ldots = c_{p_1} = c_{p_t}$. Hence we must have $p_t < p_{t-1} < \ldots < p_1 < p_t$, which is impossible. Hence cycles can never be created in this algorithm.

The validity of this algorithm follows from Theorem 9.4. In this

method, the number of connected components in the forest is reduced by a factor of 2 in each step. So, the method terminates after at most $\log n$ steps, and the worst case computational complexity of the method is $O(m \log n)$.

When applied to find the minimum spanning tree in the network in Figure 9.1, this algorithm selects the edges (1; 2), (3; 4), (5; 6) in the first step after initialization. At this stage there are three trees in the forest, each one consisting of a single edge. In the next step it selects the edges (2; 5), (4; 5) to be added to the forest. With this addition, we get the spanning tree marked by the thick edges in Figure 9.1, and the method terminates.

There are several other algorithms for the minimum cost spanning tree problem based on the same idea of growing a forest using the greedy principle, and on the results in Section 9.1, but using very sophisticated data structures. Some of the best algorithms have the worst case computational complexity of $O(m \log \log n)$. The interested reader is referred to Yao [1975] and Tarjan [1983 of Chapter 1].

Other Problems Solvable by the Greedy Approach

The greedy method applies to a variety of problems besides the minimum cost spanning tree problem, in the general setting of matroidal optimization, see Lawler [1976 of Chapter 1]. However, the real and fundamental importance of the greedy approach is not mainly due to its ability to solve a few theoretical problems in matroidal optimization exactly; but because of its amenability in constructing heuristic algorithms to obtain reasonable solutions to many hard problems for which efficient exact algorithms are not known. There are a large number of combinatorial optimization problems which do not seem to have the basic structural properties to guarantee that an optimum solution for them can be found efficiently by any known approach. The greedy approach is the basis for the most commonly used heuristic algorithms to attack such problems that arise in applications. Thus the practical importance of the greedy approach is enormous. The greedy approach has 3 characteristic features.

The Incremental Feature It poses the problem in such a way that

the solution can be viewed as a set of elements. The approach builds up this solution set one element at a time.

The No-backtracking Feature Once an element is selected for inclusion in the solution set, it is never taken back or replaced by some other element.

The Greedy Selection Feature Each additional element selected for inclusion in the solution set is the best among those available for selection at that stage.

Several different criteria could be used to characterize the "best" when making the greedy selection, depending on the problem being solved. When used as a base for developing heuristic algorithms, the success of the greedy approach depends critically on the choice of this criterion.

A second approach that is commonly used in designing heuristic algorithms is the **interchange approach**. This approach obtains a solution set for the problem first, by using a greedy or any other approach. Then it selects a small positive integer, r, and searches to see if a better solution set can be obtained by exchanging r or less elements from the present solution set with those outside it. If an improvement is obtained, the whole process is repeated with the new solution set. If no improvement is obtained by the search, the method terminates with the present solution. Since the computational effort for the search grows rapidly with r, it is usually taken to be a small positive integer. The greedy approach coupled with an interchange approach is the basis for the most commonly used methods by practitioners to handle hard combinatorial optimization problems.

The Worst Case Performance of the Greedy Approach

On some problems like the minimum cost spanning tree problem, the greedy method has been shown to yield an optimum solution. However, it is frequently used on a variety of other combinatorial optimization problems on which it is not guaranteed to find an optimum solution. How does it perform on such problems? Here we summarize the disturbing results of Jenkyns [1986], who has constructed examples

of assignment and traveling salesman problems on which the greedy method produces the worst possible solution.

Consider the assignment problem of order n in which the objective function is required to be *maximized* (the algorithms for the assignment problem in Chapter 3 are described in terms of minimization). Let $C = (c_{ij})$ be the objective coefficient matrix. The most direct version of the greedy method identifies a cell (p, q) with the maximum coefficient in C, puts an allocation in that cell, strikes off row p and column q, and continues in the same manner with the remaining matrix until n allocations are made. Let $n = 6$, consider this problem with the following matrix $C = (c_{ij})$.

	c_{ij}						
$j =$	1	2	3	4	5	6	$\bar{u}_i$
$i = 1$	1	2	6	10	17	29	0
2	3	4	8	11	20	30	2
3	5	7	9	12	22	33	4
4	13	14	15	16	23	34	9
5	18	19	21	24	25	35	16
6	26	27	28	31	32	36	24
$\bar{v}_j$	1	2	5	7	9	12	

It can be verified that the greedy method produces the unit matrix with all allocations along the main diagonal, as the optimum assignment. However, from the $\bar{u}, \bar{v}$ vectors given in the above tableau, it can be verified that $\bar{u}_i + \bar{v}_j \overset{\leq}{=} c_{ij}$ for all i, j; and $\bar{u}_i + \bar{v}_i = c_{ii}$ for $i = 1$ to 6; from the results in Chapter 3 this establishes that the unit matrix is the minimum objective value assignment in this problem. Hence the unit matrix is the worst candidate for the problem of maximizing the objective function considered here. So, on this problem, any random selection is a better answer than that produced by the greedy algorithm.

A square matrix C of order n is defined to be *gullible* (to indicate that it will be an easy victim of a shady lady) if the greedy algorithm produces the worst possible answer for the objective maximizing assignment problem with C as the objective coefficient matrix. Let $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_n), \hat{v} = (\hat{v}_1, \ldots, \hat{v}_n)$ be any pair of vectors in which

the components are nondecreasing left to right. Select c_{ij} to satisfy $\hat{u}_i + \hat{v}_j \leqq c_{ij} \leqq \hat{u}_m + \hat{v}_m$ where $m = \max. \{i, j\}$, for all i, j. Then it can be verified that the greedy method produces the unit matrix of order n as the solution to the objective maximizing assignment problem with C as the objective coefficient matrix, and this is actually a minimum objective solution in this problem. Hence such matrices are gullible. Jenkyns [1986] has given many other procedures to generate gullible matrices and shows that they are not intricately contrived pathological cases.

At least for the assignment problem there are very efficient exact algorithms available, and no one would have to resort to heuristics to solve it. However, the traveling salesman problem (TSP) is a well known NP-hard problem, large scale versions of which are commonly solved using greedy type methods in many applications. Jenkyns [1986] has constructed examples of TSPs in n cities for all n, and showed that a simple greedy heuristic produces the worst possible tour on each of them.

These results show that it is not wise to rely entirely on a greedy heuristic by itself, to tackle hard combinatorial optimization problems without some sort of error analysis.

How to Find Maximum Length Spanning Trees

If a maximum length spanning tree in G = $(\mathcal{N}, \mathcal{A})$ with c as the edge length vector is required, it can be obtained by finding a minimum length spanning tree in G with $-c$ as the edge length vector. The algorithms discussed above for the minimum cost spanning tree problem work for arbitrary cost coefficients.

9.3 An Algorithm for Ranking the Spanning Trees in Ascending Order of Cost

Let G = $(\mathcal{N}, \mathcal{A}, c)$ be a connected undirected network with c as the edge cost vector, and $|\mathcal{N}| = n, |\mathcal{A}| = m$. In Section 9.2 we discussed several algorithms for finding an unconstrained minimum cost spanning tree in

G. In some applications, a minimum cost spanning tree satisfying some constraints may be required. One possible approach for finding such a tree is to rank the spanning trees in G in increasing order of cost until one satisfying the constraints is found. Here we present an algorithm for this ranking from Murty, Saigal, and Suurballe [1974] based on the application of the partitioning technique discussed in Sections 3.6 and 4.7. We denote the ranked sequence of spanning trees in G by $\mathbb{T}_1, \mathbb{T}_2, \ldots$ where $\mathbb{T}_1$ is a minimum cost spanning tree in G, and for $u \geq 1$, $\mathbb{T}_{u+1}$ is a minimum cost spanning tree in G excluding the trees $\mathbb{T}_1, \ldots, \mathbb{T}_u$.

The algorithm deals with subsets of spanning trees in G which will be denoted using a special notation defined below. Here $a_1, \ldots, a_r, b_1, \ldots, b_s$ are edges in $\mathcal{A}$.

$$\tau_0 = [a_1, \ldots, a_r; \bar{b}_1, \ldots, \bar{b}_s] \quad = \quad \begin{array}{l}\text{Set of all spanning trees in G containing each of the edges } a_1, \ldots, a_r; \\ \text{and none of the edges } b_1, \ldots, b_s.\end{array} \tag{9.4}$$

The edges $a_1, \ldots, a_r$ are said to be the *included edges* in τ_0; and the edges $b_1, \ldots, b_s$ are said to be the *excluded edges* from τ_0.

Let $\tau = [a_1, \ldots, a_r; \bar{b}_1, \ldots, \bar{b}_s]$ be a set of spanning trees in G, and let $\mathbb{T}$ consisting of edges a_1, ..., a_r, e_{r+1}, ..., e_{n-1}; be a minimum cost spanning tree among those in τ. Define the following new subsets.

$$\begin{aligned}
\tau_1 &= [a_1, \ldots, a_r; \bar{b}_1, \ldots, \bar{b}_s, \bar{e}_{r+1}] \\
\tau_2 &= [a_1, \ldots, a_r, e_{r+1}; \bar{b}_1, \ldots, \bar{b}_s, \bar{e}_{r+2}] \\
\tau_3 &= [a_1, \ldots, a_r, e_{r+1}, e_{r+2}; \bar{b}_1, \ldots, \bar{b}_s, \bar{e}_{r+3}] \\
&\vdots \\
\tau_{n-r-1} &= [a_1, \ldots, a_r, e_{r+1}, \ldots, e_{n-2}; \bar{b}_1, \ldots, \bar{b}_s, \bar{e}_{n-1}]
\end{aligned}$$

For $j = 1$ to $n-r-1$, it is easy to check whether $\tau_j \neq \emptyset$, and if so to find a minimum cost spanning tree in it. Find the fundamental cutset

of the in-tree arc e_{r+j} wrt $\mathbb{T}$, suppose it is $(\mathbf{X}; \overline{\mathbf{X}})$. The edge e_{r+j} is the unique in-tree edge in $\mathbb{T}$ in the cutset $(\mathbf{X}; \overline{\mathbf{X}})$, and it is a minimum cost edge among edges of this cutset not in $\{b_1, \ldots, b_s\}$ since $\mathbb{T}$ is a minimum cost spanning tree among those in τ. The subset τ_j is empty iff the cutset $(\mathbf{X}; \overline{\mathbf{X}}) \subset \{b_1, \ldots, b_s, e_{r+j}\}$, the set of excluded edges in τ_j. If $(\mathbf{X}; \overline{\mathbf{X}}) \not\subset \{b_1, \ldots, b_s, e_{r+j}\}$, let e'_{r+j} be a minimum cost edge in the cutset $(\mathbf{X}; \overline{\mathbf{X}})$ that is not contained in $\{b_1, \ldots, b_s, e_{r+j}\}$. Then the spanning tree $\mathbb{T}^j$ obtained by replacing e_{r+j} in $\mathbb{T}$ by e'_{r+j}; is a minimum cost spanning tree among those in τ_j. Clearly, the computational effort needed to check whether $\tau_j = \emptyset$ and finding a minimum cost spanning tree $\mathbb{T}^j$ in it if it is nonempty is at most $O(m)$ by this approach.

Each of the sets $\tau_1, \ldots, \tau_{n-r-1}$ is again in the same form as in (9.4), and clearly they are mutually disjoint, and their union is $\tau \backslash \{\mathbb{T}\}$. These sets are said to be the *new sets generated when τ is partitioned using the minimum spanning tree $\mathbb{T}$ in it.* The number of these sets is $n - 1 -$(number of included edges in τ), and some of these sets may be empty. Clearly, a minimum cost spanning tree among those in $\tau \backslash \{\mathbb{T}\}$ is the best among $\{\mathbb{T}^j : j = 1 \text{ to } n - r - 1 \text{ such that } \tau_j \neq \emptyset\}$.

The ranking algorithm generates various sets of spanning trees in G of the form defined in (9.4). Each nonempty set among these is stored in a list, together with a minimum cost spanning tree in it, in increasing order of the cost of this tree, going from top to bottom. Each step of the algorithm generates one additional spanning tree in the ranked sequence, and adds at most $n - 1$ new sets to the list. The computational effort of each step is at most $O(mn)$, and the method can be terminated any time, after enough spanning trees in the ranked sequence have been obtained.

RANKING ALGORITHM

Initial step Find a minimum cost spanning tree in G and call it $\mathbb{T}_1$, the first tree in the ranked sequence. Suppose the edges in it are $e_1, \ldots, e_{n-1}$. Define the subsets: $\tau_1 = [\bar{e}_1]$, $\tau_2 = [e_1; \bar{e}_2]$, $\ldots, \tau_{n-1} = [e_1, \ldots, e_{n-2}; \bar{e}_{n-1}]$. Find a minimum cost spanning tree in each of these subsets, and arrange the nonempty subsets among them in increasing order of cost of the minimum cost

spanning tree in them (as you go from top to bottom) in the list.

General step Suppose the trees $\mathbb{T}_1, \ldots, \mathbb{T}_u$ in the ranked sequence have already been obtained. Pull out the set from the top of the list now, suppose it is τ, and delete it from the list. $\mathbb{T}_{u+1}$ is the stored minimum cost spanning tree in τ. If no more elements in the ranked sequence are needed, terminate. Otherwise, partition τ using $\mathbb{T}_{u+1}$. Find the minimum cost spanning tree in each of the new sets generated at this partition, as described above, and add the nonempty sets among them, together with the minimum cost spanning tree in it in the appropriate place in the list, and go to the next step.

Discussion

If only a certain number, α say, of the spanning trees in the ranked sequence are required, only the top α sets in the list need be kept, and the rest discarded. If the aim is to obtain all spanning trees whose cost is $\leq$ some specified β, any set generated during the algorithm is stored only if the cost of the minimum spanning tree in it is $\leq \beta$.

9.4 Other Types of Minimum Cost Spanning Tree Problems

So far we have dealt with the unconstrained minimum cost spanning tree problem in networks, ignoring any orientations of the lines. Here we will discuss other types of minimum spanning tree problems briefly.

Constrained Minimum Cost Spanning Tree Problems

If we are required to find a minimum spanning tree subject to degree constraints at one or more nodes (i.e., those of the form: at certain nodes i, the tree can have no more than a specified number d_i of incident edges, or should have exactly a specified number of incident edges, etc.), we have a **degree constrained minimum cost spanning tree problem**. Problems with a degree constraint at only one node can be

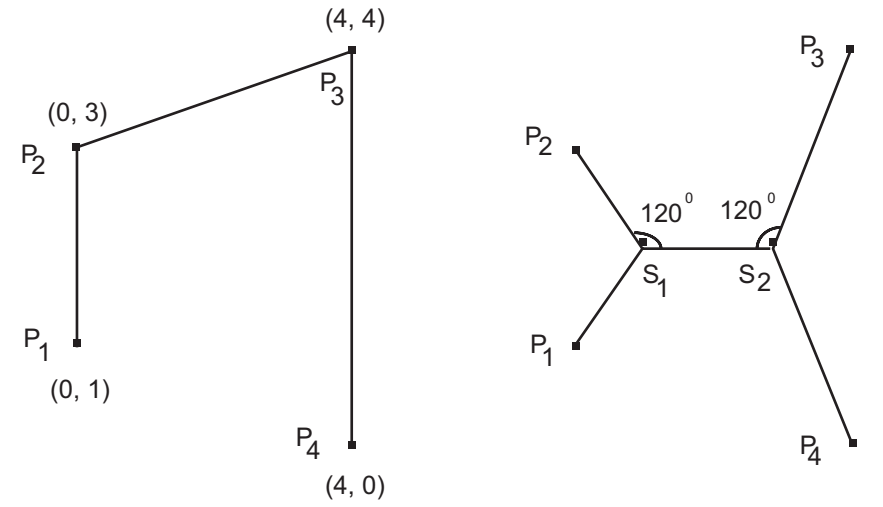
transformed into unconstrained problems (Gabow and Tarjan [1984]). Problems with degree constraints at nodes no pair of which are adjacent in G can be formulated as weighted matroid intersection problems and solved by efficient matroidal algorithms (Gabow and Tarjan [1984], Lawler [1976 of Chapter 1]).

But if degree constraints appear at all the nodes in G, the problem becomes a hard combinatorial optimization problem. As an example consider the case where the degree in the tree at every node is required to be $\leqq 2$. Any feasible spanning tree to this problem is a node covering simple path and vice versa, such paths are called **Hamiltonian paths**. To check whether a Hamiltonian path exists in an undirected network is NP-complete, and to find a minimum cost Hamiltonian path is NP-hard.

The Steiner Tree Problem in Networks

So far we have considered problems of finding optimum trees that span all the nodes in the network. In the **Steiner tree problem**, we are given a connected undirected network $G = (\mathcal{N}, \mathcal{A}, c)$ and a subset of nodes $\mathbf{X} \subset \mathcal{N}$. The problem is to find a minimum cost tree $\mathbb{T}$ in G subject to the constraint that $\mathbb{T}$ must include all the nodes in $\mathbf{X}$, but may or may not include the nodes in $\mathcal{N} \setminus \mathbf{X}$. So, the problem is equivalent to finding a subset of nodes $\mathbf{Y}$ satisfying $\mathbf{X} \subseteq \mathbf{Y} \subseteq \mathcal{N}$ such that the cost of a minimum spanning tree in the subnetwork of G induced by $\mathbf{Y}$ is the least subject to this constraint. Nodes in the optimal $\mathbf{Y}$ which are not in $\mathbf{X}$ are called **Steiner points** for this problem. So, the problem boils down to the optimal choice of Steiner points from $\mathcal{N} \setminus \mathbf{X}$.

This problem is the network version of the well known **Euclidean or rectilinear Steiner problem in geometry**. There, we are given a set $\mathbf{P}$ of points in the 2-dimensional plane. The problem is to connect the points in $\mathbf{P}$ by straight line segments so that the total length (Euclidean or rectilinear) of the line segments drawn is a minimum. Some of these lines could meet at points in the plane which are not in the set $\mathbf{P}$, such points are the **Steiner points** in the solution of this problem. As an example consider the problem in which $\mathbf{P} = \{P_1$, to



(a) Shortest connecting network with no out side points

(b) Shortest connecting network with two Steiner points S_1, S_2

Figure 9.4:

$P_4\}$ as shown in Figure 9.4 (a). If no outside points are allowed, the shortest connecting network has Euclidean length of 10.123. By introducing 2 Steiner points S_1, S_2 as shown in Figure 9.4 (b), the Euclidean length of the connecting network reduces to 9.196.

The Steiner tree problem in networks is a hard combinatorial optimization problem. Exact enumerative algorithms are available for solving it, but the computational effort required by these algorithms grows exponentially with $|\mathcal{N} \backslash \mathbf{X}|$.

Directed Spanning Tree Problems

So far we have discussed spanning tree problems in undirected networks. Consider now a directed network $G = (\mathcal{N}, \mathcal{A}, c)$ with $|\mathcal{N}| = n, |\mathcal{A}| = m$. Suppose we are given a designated node, say node 1, as the root node. A **branching** rooted at 1 is a spanning outtree rooted at 1. The problem of finding a minimum cost branching rooted at 1 in

G

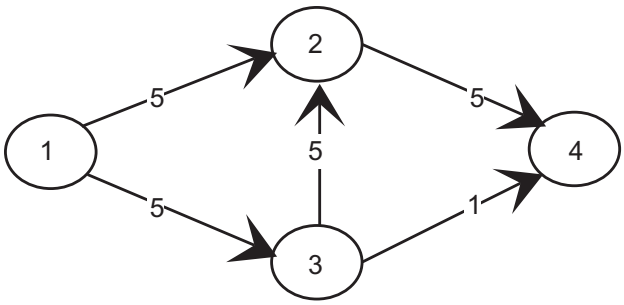


Figure 9.5:

has a very efficient algorithm of time complexity O(nm). But it does not seem to be of much practical importance, so we will not discuss it. See references given at the end of this chapter.

9.5 Exercises

9.3 Construct a directed network with positive arc lengths to show that a shortest chain tree rooted at a node may not be a minimum cost branching rooted at that node (try the network in Figure 9.5, with node 1 as the root node. Arc lengths are entered on the arcs).

9.4 Given a minimum cost spanning tree in an undirected network, develop an efficient procedure for doing cost ranging on it, i.e., to determine the range of values of each cost coefficient which does not affect the minimality of the tree (Tarjan [1982]).

9.5 We are given a minimum cost spanning tree $\mathbb{T}$ in an undirected connected network. Develop an efficient procedure to update $\mathbb{T}$ into a minimum cost spanning tree when a new node and incident edges are added to the network (Spira and Pan [1975]).

9.6 Develop good heuristic algorithms based on the greedy approach for the following combinatorial optimization problems.

(a) The Knapsack Problem In this problem we are given n objects with w_i, v_i as the weight, value of the ith object, $i = 1$ to n. A knapsack with weight capacity w is provided. It is required to find an optimal subset of objects to be loaded into the knapsack, so as to maximize its value, subject to the knapsack's weight capacity. All data are positive integers. Mathematically, it is to find $x = (x_i : i = 1 \text{ to } n)$ to

$$\begin{array}{lllll} \text{maximize} & v_1x_1 & + & \ldots + v_nx_n & \\ \text{subject to} & w_1x_1 & + & \ldots + w_nx_n \leqq w & \\ \text{and } x_i = 0, \text{ or } 1, & i & = 1 \text{ to } n & & \end{array}$$

Apply your heuristic algorithm for obtaining a solution to the following journal subscription problem faced by a librarian. The librarian has to decide an optimum subset among 8 journals to renew the subscription, with only 500$ available in the new budget for these journals. Other data is given below. The journals selected by the librarian should satisfy the maximum number of users subject to the budget constraint.

Journal i	w_i, annual subscription in \$	v_i, no. of users per year
1	80	7840
2	95	6175
3	115	8510
4	165	15015
5	125	7375
6	78	1794
7	69	897
8	99	8315

(b) The Set Covering Problem This is a 0-1 integer programming problem of the following form

$$\begin{array}{lll} \text{minimize} & cx & \\ \text{subject to} & Ax & \geqq \mathbf{e} \\ x \text{ is a } 0-1 & & \text{vector} \end{array}$$

where **e** on the right hand side is a column vector of all 1's in $\mathbb{R}^m$, $A = (a_{ij})$ is a given 0-1 matrix of order $m \times n$, and c is a positive row vector in $\mathbb{R}^m$. Apply your heuristic algorithm on the following numerical example. Here $m = 10$, $n = 13$, c is the row vector of all 1's in $\mathbb{R}^{13}$, and the sets $\Gamma_i = \{j : j = 1 \text{ to } 13 \text{ such that } a_{ij} = 1\}$ are: $\Gamma_1 = \{7, 9, 10, 13\}$, $\Gamma_2 = \{2, 8, 9, 13\}$, $\Gamma_3 = \{3, 9, 10, 12\}$, $\Gamma_4 = \{4, 5, 8, 9\}$, $\Gamma_5 = \{3, 6, 8, 11\}$, $\Gamma_6 = \{3, 6, 7, 10\}$, $\Gamma_7 = \{2, 4, 5, 12\}$, $\Gamma_8 = \{4, 5, 6, 13\}$, $\Gamma_9 = \{1, 2, 4, 11\}$, $\Gamma_{10} = \{1, 5, 7, 12\}$.

9.7 Let $G = (\mathcal{N}, \mathcal{A}, c)$ be an undirected connected network. Let d_1 be the degree on node 1 in G. Let $\mathbf{D}_r$ denote the set of spanning trees in G containing exactly r edges incident at 1. Show that there exists a $u \geq 1$ such that the set of all r for which $\mathbf{D}_r \neq \emptyset$ is $u \leq r \leq d_1$.

Suppose $\mathbb{T}$ is a minimum cost spanning tree in $\mathbf{D}_r$ for some r. Then prove that

(a) if $\mathbf{D}_{r-1} \neq \emptyset$, there are edges e_1 of $\mathbb{T}$ incident at 1, and $e_2 \in \mathcal{A}$ not incident at 1 and not in $\mathbb{T}$; such that replacing e_1 in $\mathbb{T}$ by e_2 yields a minimum cost tree in $\mathbf{D}_{r-1}$

(b) if $\mathbf{D}_{r+1} \neq \emptyset$, there are edges e_1 not in $\mathbb{T}$ and incident at 1, and e_2 in $\mathbb{T}$ not incident at 1; such that replacing e_2 in $\mathbb{T}$ by e_1 yields a minimum cost tree in $\mathbf{D}_{r+1}$.

Use these results to develop an efficient algorithm for finding a minimum cost spanning tree in G satisfying a degree constraint at node 1, as this specified degree varies parametrically (Gabow [1978]).

9.8 Minimum Ratio Spanning Trees Let $G = (\mathcal{N}, \mathcal{A})$ be a connected undirected network with two vectors $a = (a_{ij}), b = (b_{ij})$ of edge cost coefficients defined over $\mathcal{A}$. For any tree $\mathbb{T}$ in G, define $z(\mathbb{T}) = (\sum(a_{ij} : \text{ over } (i;j) \in \mathbb{T})/(\sum(b_{ij} : \text{ over } (i;j) \in \mathbb{T})$. Assume that the vector b is such, that the denominator of $z(\mathbb{T})$ has the same sign for every spanning tree in G, without any loss of generality we assume that this sign is positive. Consider the problem of finding a spanning tree $\mathbb{T}$ in G that minimizes $z(\mathbb{T})$. Let z^* denote the unknown optimum objective value for this problem

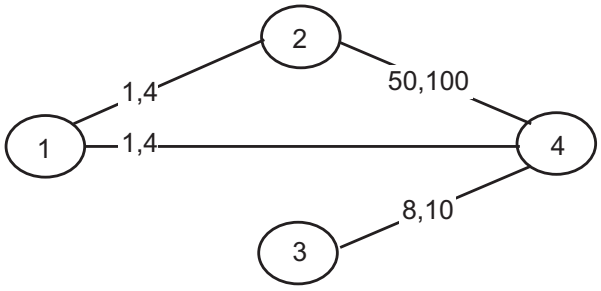


Figure 9.6: Entries on the edges are a_{ij}, b_{ij} in that order.

Consider the following greedy approach for this problem. It grows a connected tree by adding one selected edge at a time, making sure that each edge added does not form a cycle with those already selected, and among such edges it selects the one which increases the objective function by the smallest quantity. Use the network in Figure 9.6 to show that this approach may not lead to an optimum solution of this problem.

Let α be a real valued parameter. Define for each $(i;j) \in \mathcal{A}, c_{ij}(\alpha) = a_{ij} - \alpha b_{ij}$. Let $\hat{\mathbb{T}}(\alpha)$ denote a minimum cost spanning tree in G with $c(\alpha) = (c_{ij}(\alpha))$ as the edge cost vector, and let $h(\alpha)$ be its cost. Prove that

$$h(\alpha) \begin{cases} > 0 \text{ iff } z^* > \alpha \\ < 0 \text{ iff } z^* < \alpha \\ = 0 \text{ iff } z^* = \alpha \end{cases}$$

Let e, g denote any pair of edges in G. When $a_e - \alpha b_e, a_g - \alpha b_g$ are plotted against α on the 2-dimensional Cartesian plane, we get two straight lines which may either be identical, or nonintersecting parallel lines, or two lines intersecting at a unique point. For every pair of edges e, g in G for which the two lines given above intersect at a unique point, define $\alpha(e,g) = (a_e - a_g)/(b_e - b_g)$, and let Δ be the set of all such $\alpha(e,g)$. So, $|\Delta| \overset{\leq}{=} |\mathcal{A}|^{\mathbf{2}}$. Suppose there are r distinct elements in Δ, order them in increasing order, and then suppose they are $-\infty < \alpha_1 < \alpha_2 < \cdots < \alpha_r < +\infty$. Prove that for any α in any one of the open intervals $(-\infty, \alpha_1), (\alpha_1, \alpha_2), \ldots, (\alpha_r, +\infty)$, the minimum

cost spanning tree $\hat{\mathbb{T}}(\alpha)$ is actually a minimum cost spanning tree for every α in that interval. Hence $h(\alpha)$ is linear in each of these intervals. Using this result, develop an efficient algorithm for finding a spanning tree $\mathbb{T}$ in G that minimizes $z(\mathbb{T})$, and determine its computational complexity (Chandrasekaran [1977]).

9.9 In applying Kruskal's algorithm to find a minimum cost spanning tree in G, if r is the number of forest edges in a step, prove that the forest at that stage is a minimum cost forest of cardinality r in G.

9.10 It is required to find a single tree in G of minimum cost containing r edges. Develop an efficient approach for solving this problem.

9.11 G $= (\mathcal{N}, \mathcal{A}, c)$ is a connected undirected network with $c > 0$. Develop efficient methods for finding spanning trees in G that minimize : (a) the product of the cost coefficients of in-tree edges, (b) the cost of the costliest in-tree edge, (c) any symmetric cost function of edge cost coefficients.

9.12 Let G $= (\mathcal{N}, \mathcal{A})$ be a connected undirected network with $|\mathcal{N}| = n, |\mathcal{A}| = m$. The **characterization of the convex hull of spanning tree incidence vectors** in G described in Exercise 9.2 has about $2^{\mathbf{n}}$ constraints. Here we discuss a constraint formulation of the convex hull of spanning trees in G, in a higher dimensional space, with only O(mn) constraints. Additional variables called *auxiliary variables* are introduced. In this formulation, a particular node, say 1, is selected as a source node. Each of the other nodes $r \neq 1$ is associated with a distinct commodity for which it is the sink node. So, there are $(n-1)$ different commodities in this formulation, and node 1 has one unit of each of these commodities available for shipment.

For each $(i; j) \in \mathcal{A}$ with both $i, j \neq 1$ create two arcs (i, j) and (j, i). For each edge $(1; j)$ in G create an arc $(1, j)$. Let $\mathbf{A}$ denote the set of all these created arcs. So, $|\mathbf{A}| = 2m -$ degree of node 1 in G. Define $\mathbf{A}_i = \{j : (i, j) \in \mathbf{A}\}$, $\mathbf{B}_i = \{j : (j, i) \in \mathbf{A}\}$, for each $i \in \mathcal{N}$.

Introduce a variable $x_{i;j}$ for each $(i; j) \in \mathcal{A}$; it is the spanning tree-edge incidence variable for that edge; and auxiliary variables y^r_{ij} for each arc $(i, j) \in \mathbf{A}$ and $r = 2$ to n. y^r_{ij} denotes the flow of commodity

r on arc (i, j). Let $x = (x_{i;j}), y = (y^r_{ij})$. Consider the following system of constraints.

$$\begin{aligned}
\sum(x_{i;j} : \text{ over } (i;j) \in \mathcal{A}) \quad &= \quad n-1 \\
\sum(y^r_{1i} : \text{ over } i \in \mathbf{A}_1) \quad &= \quad 1, \quad r = 2 \text{ to } n \\
\sum(y^r_{ij} : \text{ over } i \in \mathbf{B}_j) & \\
-\sum(y^r_{ji} : \text{ over } i \in \mathbf{A}_j) \quad &= \quad 0, \quad j = 2 \text{ to } n, j \neq r; \\
& \qquad\quad r = 2 \text{ to } n \\
\sum(y^r_{ir} : \text{ over } i \in \mathbf{B}_r) \quad &= \quad 1, \quad r = 2 \text{ to } n \qquad (9.5) \\
y^r_{1j} - x_{1;j} \quad &\leqq \quad 0, \quad j \in \mathbf{A}_1, r = 2 \text{ to } n \\
y^r_{ij} + y^p_{ji} - x_{i;j} \quad &\leqq \quad 0, \quad i, j \text{ both } \neq 1, (i;j) \in \mathcal{A}; \\
& \qquad\quad r, p = 2 \text{ to } n \\
0 \leqq x_{i;j} \leqq 1, & \qquad (i;j) \in \mathcal{A} \\
y^r_{ij} \geqq 0, & \qquad (i,j) \in \mathbf{A}, r = 2 \text{ to } n
\end{aligned}$$

Prove that the projection of the feasible solution set of (9.5) in the x-space, is the convex hull of spanning tree incidence vectors in G (Kipp Martin [1986], Wong [1984]).

9.13 The Capacitated Minimum Spanning Tree Problem G $= (\mathcal{N}, \mathcal{A}, c, k)$ is a connected undirected network with c as the vector of edge cost coefficients, and k as the vector of edge flow capacities. One of the nodes in G, 1, is the sink node; and all other nodes are source nodes, with $i \neq 1$ required to ship a_i units of a commodity from it to node 1. The problem is to find a spanning tree in G along the simple paths in which all these shipments will be carried out. A spanning tree $\mathbb{T}$ is said to be feasible to this problem if all these flows can be carried out on it without violating the edge capacity on any in-tree edge, infeasible otherwise. For example, the spanning tree on the left in Figure 9.7 is infeasible for the capacitated problem, whereas the one on the right is feasible (in the figure, capacities of all the edges are 5, and for each $i \neq 1$, a_i is entered by the side of node i, the data is the same in both the trees). Develop either an exact method, or a good

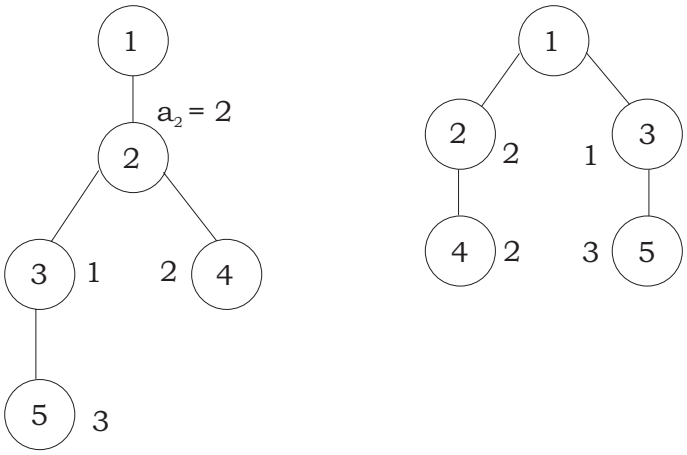


Figure 9.7: Capacity of every edge is 5. a_i is entered by the side of node i. The tree at the left is infeasible, and the one on the right is feasible, for the capacitated problem.

heuristic method, for finding a minimum cost feasible spanning tree for the capacitated problem (Chandy and Lo [1973]).

9.14 A Constrained Minimum Spanning Tree Problem Node 1 is the center that will be the root of a spanning tree in $G = (\mathcal{N}, \mathcal{A}, c)$. Each node $i \neq 1$ is associated with a load of a_i. It is required to find a minimum cost spanning tree in G satisfying: there should be no more than n_0 nodes connected to the center by an in-tree edge, there should be no more than l edges in tandem in a path connecting a node to the center, the total load on any edge (the sum of the loads of all nodes it connects to the center) cannot be greater than a specified d, and that no node other than the center have degree greater than a specified number r. Constrained minimum spanning tree problems of this type arise in designing centralized telecommunication and data communication networks using a single speed line. Develop a heuristic algorithm based on Kruskal's algorithm to generate high quality solutions to this problem (Kershenbaum [1974]).

9.15 G $= (\mathcal{N}, \mathcal{A}, c)$ is a connected undirected network with $c \geqq 0$. Develop 0-1 integer programming formulations for the following constrained minimum cost spanning tree problems in G: a) for each node in a specified subset of nodes, an upper bound on the number of in-tree edges incident at that node is specified, b) the number of descendents of each son node of a specified root node is required to be $\leqq$ a specified positive integer α (Gavish [1982]).

9.16 For any subset $\mathbf{N} \subseteq \mathcal{N}$ of nodes in an undirected network G $= (\mathcal{N}, \mathcal{A})$, let $\mathbf{E}(\mathbf{N})$ denote the set of edges with both their nodes in $\mathbf{N}$. Prove that the set of feasible solutions $x = (x_e : e \in \mathcal{A})$ of the following system, is the convex hull of forest-edge incidence vectors in G.

$$\begin{aligned} \sum(x_e : \text{ over } e \in \mathbf{E}(\mathbf{N})) &\leqq |\mathbf{N}| - 1, \text{ for each } \mathbf{N} \subseteq \mathcal{N} \text{ with } |\mathbf{N}| \geqq 2 \\ x_e &\geqq 0, \text{ for all } e \in \mathcal{A} \end{aligned}$$

(Edmonds [1969]).

9.17 Let l_{ij} denote the number of lines on the simple path between nodes i, j in a tree $\mathbb{T}$. For each i in $\mathbb{T}$, define $\theta_i = \max. \{l_{ij} : \text{over nodes } j \text{ in } \mathbb{T}\}$. A *center* of $\mathbb{T}$ is defined to be a node in $\mathbb{T}$ that minimizes θ_i. Show that the following algorithm finds a center for $\mathbb{T}$. Also, find the center of the tree in Figure 9.8 using this algorithm.

Step 1 If the number of nodes in the tree is $\leqq 2$, any node in the tree is a center, terminate. Otherwise go to Step 2.

Step 2 Identify all the terminal nodes in the tree, and delete them and the lines incident at them from the tree. Go to Step 3.

Step 3 If the number of remaining nodes is $\leqq 2$, any of these is a center for the original tree, terminate. Otherwise return to Step 2 with the remaining tree.

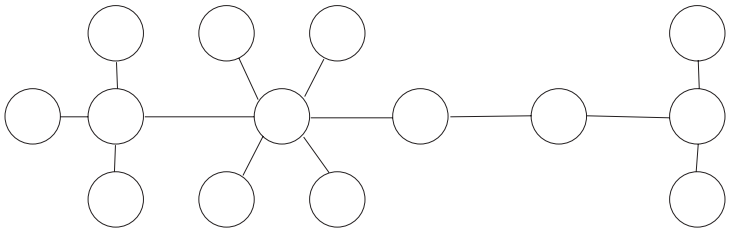


Figure 9.8:

9.18 Let $\mathbb{T}$ be a tree spanning n nodes. For nodes i, j in $\mathbb{T}$ define l_{ij} as in Exercise 9.17. For a node i in $\mathbb{T}$ let $j_1, \ldots, j_p$ be all its adjacent nodes in $\mathbb{T}$, and let $\mathbb{T}_{j_r}$ be the subtree containing node j_r when the line joining i and j_r is deleted from $\mathbb{T}$, $r = 1$ to p. Let n_r be the number of nodes in $\mathbb{T}_{j_r}, r = 1$ to p. Node i in $\mathbb{T}$ is said to be a *centroid* of $\mathbb{T}$ iff $n_r \leq n/2$ for all $r = 1$ to p. Prove that $\sigma_i = \sum(l_{ij} : \text{over } j \text{ in } \mathbb{T})$ is minimized iff i is a centroid of $\mathbb{T}$. Develop an efficient algorithm for finding a centroid of $\mathbb{T}$. Apply your algorithm to find a centroid of the tree given in Figure 9.9 (Kang, Lee, Chang, and Chang [1977], Kang and Ault [1975]).

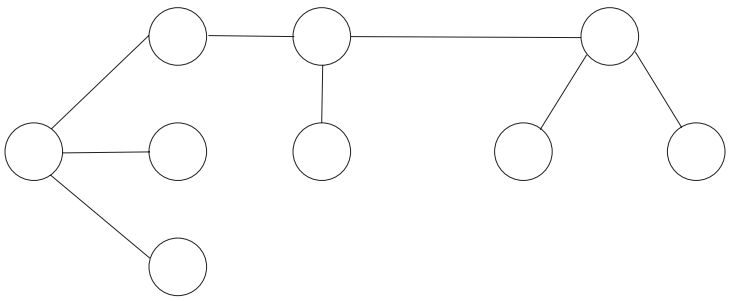


Figure 9.9:

9.19 Shallow Minimum Spanning Trees $G = (\mathcal{N}, \mathcal{A}, c)$ is a connected undirected network. Among all minimum cost spanning trees in G, it is required to find one which is as shallow as possible (i.e., has the smallest number of levels when it is drawn as a rooted tree). Show

that the following variant of Prim's algorithm finds such a tree. In this algorithm, $\mathbf{X}$ denotes the set of in-tree nodes, which grows by one in each step.

Step 0 Select any node i, let $\mathbf{X} = \{i\}$. Go to Step 1.

Step 1 For each $j \in \mathcal{N} \backslash \mathbf{X}$, define $d_j = \min. \{c_{ij}$: over $i \in \mathbf{X}\}$. Find a node $q \in \mathcal{N} \backslash \mathbf{X}$ which minimizes d_j over j in this set. Let $\mathbf{Y}$ be the set of all i in $\mathbf{X}$ which tie for the minimum in the definition of d_q. If $\mathbf{Y}$ is a singleton set, let p be the node in it, otherwise define p to be the node in $\mathbf{Y}$ which has the maximum number of in-tree edges incident at it at this stage. Select $(p; q)$ as the next in-tree edge, and transfer q into $\mathbf{X}$ from $\mathcal{N} \backslash \mathbf{X}$. If $\mathbf{X}$ is now $\mathcal{N}$, terminate. Otherwise, repeat this step.

Develop appropriate data structures to obtain an efficient implementation of this algorithm (Kang, Lee, Chang, and Chang [1977]).

9.20 A min-max spanning tree in G is one which minimizes the maximum cost among all in-tree edges. Prove that every minimum cost spanning tree is also a min-max spanning tree, but the converse may not be true. Develop an $O(m)$ complexity algorithm to find a min-max spanning tree in G (Camerini [1978]).

9.21 $\mathbb{T}$ is a minimum cost spanning tree in the connected undirected network $G = (\mathcal{N}, \mathcal{A}, c)$. $G' = (\mathcal{N}', \mathcal{A}', c')$ is the network obtained when a node i and all the edges incident at it are deleted from G. Assume that G' is connected. Let $\mathbb{T}_i = (\mathcal{N}', E_i)$ be the forest remaining when node i and all the in-tree edges incident at it are deleted from $\mathbb{T}$.

If i is a leaf node in $\mathbb{T}$, show that $\mathbb{T}_i$ is a minimum cost spanning tree in G'.

Suppose i is not a leaf node in $\mathbb{T}$. Let $G'' = (\mathcal{N}'', \mathcal{A}'')$ be an undirected network whose vertex set is the set of connected components of $\mathbb{T}_i$, and every edge $e = (a, b)$ in $\mathcal{A}''$ corresponds to an edge e' in $\mathcal{A}'$ such that e' has the smallest cost among all the edges connecting the components a and b of $\mathbb{T}_i$ in G'. Shoe that G'' is connected, and that if $\mathbb{T}'' = (\mathcal{N}'', E'')$ is a minimum cost spanning tree in G'' then

$E' = E_i \cup E''$ is such that $T' = (\mathcal{N}', E')$ is a minimum cost spanning tree in G'.

Using these results, develop an efficient algorithm to update the minimum cost spanning tree when a vertex is deleted from the network (Chin and Houck [1978], Tsin [1988]).

9.22 Minimum Cost Spanning Tree Problen as a Min-Max Path Problem Let $G = (\mathcal{N} = \{1, 2, \ldots, n\}, \mathcal{A}, c)$ be a connected undirected network. Assume that all the entries in c are distinct, otherwise the perturbation discussed under Bourvka's can be used to achieve this. Define the cost of any path in G to be the max. $\{c_{ij} : (i; j)$ is an edge on the path$\}$. Define $l_{ij}^0 = \infty$ if $(i; j) \notin \mathcal{A}$, c_{ij} if $(i; j) \in \mathcal{A}$. For all $i, j, r = 1$ to n define

$$l_{ij}^r = \text{cost, as defined above, of the least cost path from } i \text{ to } j \text{ that passes through only vertices in the set } \{1, 2, \ldots, r\}.$$

For any $i, j \in \mathcal{N}$, the least cost path between i and j with no intermediate vertex higher than r either passes through r (in this case $l_{ij}^r = \max. \{l_{ir}^r, l_{rj}^r\}$) or does not (in this case $l_{ij}^r = l_{ij}^{r-1}$). Using this, prove that $l_{ij}^r = \min. \{l_{ij}^{r-1}, \max. \{l_{ir}^{r-1}, l_{rj}^{r-1}\}\}$.

Prove that the unique minimum cost spanning tree in G in this case can be recovered from the costs of the least cost paths using the rule: the edge $(i; j) \in \mathcal{A}$ is in the minimum cost spanning tree in G iff $l_{ij}^0 = l_{ij}^n$ (Maggs and Plotkin [1988]).

9.23 Develop an efficient algorithm for finding a minimum cost spanning tree in G, subject to the constraint that it contain a specified number, r, of edges from a given subset $\mathbf{S} \subset \mathcal{A}$ (Gabow and Tarjan [1984]).

9.24 Optimum Face of the Minimum Cost Spanning Tree Problem Let $G = (\mathcal{N}, \mathcal{A}, c)$ be a connected undirected network with $c = (c_{ij})$ as the vector of edge cost coefficients. There may be several spanning trees in G, which tie for being a minimum cost spanning tree in G. Let $\mathbf{E}$ denote the set of all edges in $\mathcal{A}$, each of which is contained on at least one minimum cost spanning tree in G.

1. Is the statement "every spanning tree in the partial subnetwork ($\mathcal{N}$, $\mathbf{E}$) is a minimum cost spanning tree in G" correct? Either prove it, or construct a counterexample.

2. If all the entries in the cost vector c are distinct, prove that there is a unique minimum cost spanning tree in G.

3. Suppose that some of the edge cost coefficients are equal to each other. Arrange the distinct values among $\{c_{ij} : (i;j) \in \mathcal{A}\}$ in increasing order, and let them be $\alpha_1, \ldots, \alpha_p$. For $t = 1$ to p, let $\mathbf{S}_t = \{(i;j) : c_{ij} = \alpha_t\}$. Then $\mathbf{S}_1$ is the set of least cost edges in G, $\mathbf{S}_2$ is the set of edges with the second best cost, etc.

(a) If after deleting all the edges in $\mathbf{S}_p$ (the maximum cost edges) the network remains connected, then prove that none of the edges in $\mathbf{S}_p$ is contained on any minimum cost spanning tree in G.

(b) Prove that every minimum cost spanning tree in G must contain at least one of the edges in $\mathbf{S}_1$ as an in-tree edge.

(c) Let $\mathbb{T}_0$ be any minimum cost spanning tree in G, and let r_t be the number of edges from $\mathbf{S}_t$ in $\mathbb{T}_0$, for $t = 1$ to p. Then prove that every spanning tree in G containing exactly r_t edges from $\mathbf{S}_t$ for each $t = 1$ to p is a minimum cost spanning tree in G and vice versa.

In this case, prove that the convex hull of minimum cost spanning tree incidence vectors in G is the set of feasible solutions of (9.3) and the additional constraints

$$\sum(x_{ij} : \quad \text{over} \quad (i;j) \in \mathbf{S}_t) = r_t, \quad t = 1 \quad \text{to} \quad p$$

Define a *second best valued spanning tree* in G to be a minimum cost spanning tree among those with cost $>$ cost of $\mathbb{T}_0$. In this case, using the algorithm mentioned in Exercise 9.23, develop an $O(pm \log n)$ algorithm to find a second best valued spanning tree in G.

(Murty, Hamacher, and Maffioli [1991]).

Comment 9.1 The minimum cost spanning tree problem (MCST) is a classical problem concerned with the design of optimum networks. For example, it arises when several points in a region have to be linked via cable at minimum cost. It is the first network optimization problem for which an efficient algorithm has been developed. Graham and Hell [1985] survey the history of the problem.

Boruvka [1926] seems to have been the first to develop an algorithm for this problem. Choquet [1938] has rediscovered the same algorithm. We discussed this algorithm, as well as those of Kruskal [1956] and Prim [1957]. All these methods are based on the incremental technique of building the optimum tree, edge by edge, using the greedy selection rule. The greedy method solves a variety of other problems besides the MCST. The MCST has a powerful generalization in the setting of matroid theory, it is the problem of finding a minimum cost base of a matroid (the cost of a base is the sum of the costs associated with the elements in it, see Lawler [1976 of Chapter 1] for a discussion of matroid theory relevant to the area of optimization). The MCST is a special case of this problem corresponding to a graphic matroid. The greedy method can be used to find a minimum cost base of a matroid.

Prim's algorithm has been developed earlier by Jarnik [1930], and improved by Dijkstra [1959 of Chapter 4].

A variety of techniques have been developed to improve the worst computational complexity of these algorithms on sparse networks. These techniques involve sophisticated data structures. See Cheriton and Tarjan [1976], Johnson [1975], and Yao [1975]. Tarjan [1983 of Chapter 1] contains surveys on the MCST and variants of it. Parallel algorithms for the MCST are discussed by Akl [1986], Lavallee [1984], Lavallee and Roucairol [1986], and Nath and Maheswari [1982]. Some applications of the MCST are discussed by Ali and Kennington [1986], Gavish [1982], Held and Karp [1970], Hu[1961], Kang, Lee, Chang, and Chang [1977], and Magnanti and Wong [1984]. Sensitivity analysis and updating an optimum MCST under changes in the data are discused in Chin and Houck [1978], Spira and Pan [1975], Tarjan [1982], and Tsin [1988].

The related theoretical problem of characterizing $\mathbf{K}$ = convex hull of the spanning tree incidence vectors in G = $(\mathcal{N}, \mathcal{A})$ with n nodes and m edges, through a system of linear constraints has attracted a

good deal of attention. If we use only the edge incidence variables, this characterization requires about $2^{\mathbf{n}}$ constraints in them (Exercise 9.2). Recently Kipp Martin [1986] and Wong [1984] have shown that by introducing additional variables called auxiliary variables, a convex polyhedron Γ can be defined through a system of $O(nm)$ linear constraints only, such that $\mathbf{K}$ is the projection of Γ in the subspace of the edge incidence variables.

The greedy algorithm has been used for a long time as a heuristic approach to obtain solutions for many hard combinatorial optimization problems arising in real world applications, even though the method is not guaranteed to produce an optimum or even near optimum solution on such problems. Recently Jenkins [1986] has constructed simple assignment and traveling salesman problems on which he shows that the greedy method leads to the worst possible solution for them (i.e., a solution actually minimizing the objective function which is required to be maximized). These examples indicate the need for extreme caution while using such heuristic methods to solve problems on which the methods performance is unknown.

Most of the constrained minimum cost spanning problems tend to be hard problems. See Chandy and Lo [1973], Gabow [1978], and Kershenbaum [1974] for a discussion of some constrained spanning tree problems.

9.6 References

S. AKL, 1986, "An Adaptive and Cost-Optimal Parallel Algorithm for Minimum Spanning Trees," *Computing*, 36(271-277).

I. ALI and J. KENNINGTON, 1986, "The Asymmetric M-Traveling Salesman Problem: A Duality Based Branch and Bound Algorithm," *DAM*, 13(259-276).

O. BORUVKA, 1926, "O Jistém Problému Minimalnim," *Pracá Moravské Přirodovědecké Společnosti*, 3(37-58), in Czech.

P. M. CAMERINI, Jan. 1978, "The Min-Max Spanning Tree Problem and Some Extensions," *IPL*, 7, no. 1(10-14).

P. M. CAMERINI, L. FRATTA, and F. MAFFIOLI, 1979, "A Note on Finding Optimum Branchings," *Networks*, 9(309-312).

R. CHANDRASEKARAN, 1977, "Minimal Ratio Spanning Trees," *Networks*, 7,

no. 4(335-342).
K. M. CHANDY and T. LO, 1973, "The Capacitated Minimum Spanning Tree," *Networks*, 3, no. 2(173-181).
D. CHERITON and R. E. TARJAN, 1976, "Finding Minimum Spanning Trees," *SIAM Journal on Computing*, 5(724-742).
F. CHIN and D. HOUCK, 1978, "Algorithm for Updating Minimum Spanning Trees," *Journal of Computer and System Science*, 16(333-344).
G. CHOQUET, 1938, "Etude de Certains Reseaux de Routes," *C. R. Acad. Sci. Paris*, 206(310-313).
J. EDMONDS, 1970, "Submodular Functions, Matroids and Certain Polyhedra," PP 69-87 in R. Guy(ed.), *Combinatorial Structures and Their Applications, Proceedings of the Calgary International Conference*, Gordon Breach, N.Y.
H. N. GABOW, 1978, "A Good Algorithm for Smallest Spanning Tree With a Degree Constraint," *Networks*, 8(201-208).
H. N. GABOW and R. E. TARJAN, Mar. 1984, "Efficient Algorithms for a Family of Matroid Intersection Problems," *Journal of Algorithms*, 5, no. 1(80-131).
R. G. GALLAGER, P. A. HUMBLET, and P. M. SPIRA, 1983, "A Distributed Algorithm for Minimum-Weight Spanning Trees," *ACM Transactions on Programming Languages and Systems*, 5(66-77).
B. GAVISH, 1982, "Topological Design of Centralized Computer Networks - Formulations and Algorithms," *Networks*, 12(355-377).
R. L. GRAHAM and P. HELL, Jan. 1985, "On the History of the Minimum Spanning Tree Problem," *Ann. History of Computing*, 7, no. 1(43-57).
M. HELD and R. KARP, 1970, "The Traveling Salesman Problem and Minimum Spanning Trees," *OR*, 18(1138-1162).
T. C. HU, 1961, "The Maximum Capacity Route Problem," *OR*, 9(898-900).
V. JARNIK, 1930, "O Jistém Problému Minimalnim," *Pracá Moravské Přirodovědecké Společnosti*, 6(57-63), in Czech.
T. A. JENKYNS, 1986, "The Greedy Algorithm is a Shady Lady," *Congressus Numerantium*, 51(209-215).
D. B. JOHNSON, 1975, "Priority Queues with Update and Finding Minimum Spanning Trees," *IPL*, 4(53-57).
A. KANG and A. AULT, Sept. 1975, "Some Properties of a Centroid of a True Tree," *IPL*, 4, no. 1(18-20).
A. N. C. KANG, R. C. T. LEE, C. CHANG, and S. K. CHANG, May 1977, "Storage Reduction Through Minimal Spanning Trees and Spanning Forests," *IEEE*

Transactions on Computers, C-26, no. 5(425-434).

A. KERSHENBAUM, 1974, "Computing Capacitated Minimal Spanning Trees Efficiently," *Networks*, 4, no. 4(299-310).

R. KIPP MARTIN, May 1986, "A Sharp Polynomial Size Linear Programming Formulation of the Minimum Spanning Tree Problem," Graduate school of Business, U. of Chicago.

J. B. KRUSKAL, 1956, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem ," *Proceedings of the AMS*, 7(48-50).

I. LAVALLEE, 1984, "An Efficient Parallel Algorithm for Computing a Minimum Spanning Tree," *Parallel Computing*, 83(259-262).

I. LAVALLEE and G. ROUCAIROL, 1986, "A Fully Distributed (Minimal) Spanning Tree Algorithm," *IPL*, 23(55-62).

B. M. MAGGS and S. A. PLOTKIN, Jan. 1988, "Minimum-cost Spanning as a Path-Finding Problem," *IPL*, 26, no. 6(291-293).

T. L. MAGNANTI and R. T. WONG, 1984, "Network Design and Transportation Planning: Models and Algorithms," *TS*, 18(1-55).

N. MEGIDDO, 1979, "Combinatorial Optimization with Rational Objective Functions," *MOR*, 4(414-424).

K. G. MURTY, R. SAIGAL, and J. SUURBALLE, 1974, "Ranking Algorithm for Spanning Trees," Bell labs. memo.

K. G. MURTY, H. HAMACHER, and F. MAFFIOLI, 1991, "Characterization of the optimum face of the minimum cost spanning tree problem, and applications," Tech. report, IOE Dept., University of Michigan, Ann Arbor, Mich.

D. NATH and S. MAHESWARI, 1982, "Parallel Algorithms for the Connected Components and Minimal Spanning Tree Problems," *IPL*, 14, no. 1(7-11).

R. C. PRIM, 1957, "Shortest Connection Networks and Some Generalizations," *Bell System Tech. J.*, 36(1389-1401).

P. M. SPIRA and A. PAN, 1975, "On Finding and Updating Spanning Trees and Shortest Paths," *SIAM J. on Computing*, 4(375-380).

R. E. TARJAN, 1982, "Sensitivity Analysis of Minimum Spanning Trees and Shortest Path Trees," *IPL*, 14(30-33).

Y. H. TSIN, April 1988, "On Handling Vertex Deletion in Updating Minimum Spanning Trees," *IPL*, 27, no. 4(167-168).

A. YAO, 1975, "An O($|E|$log log $|V|$) Algorithm for Finding Minimum Spanning Trees," *IPL*, 4(21-23).

References on Steiner Tree Problems in Networks

Y. P. ANEJA, 1980, "An Integer Linear Programming Approach to the Steiner Problem in Graphs," *Networks*, 10(167-178).

S. E. DREYFUS and R. A. WAGNER, 1972, "The Steiner Problem in Graphs," *Networks*, 1(195-207).

E. N. GILBERT and H. O. POLLOCK, Jan. 1968, "Steiner Minimal Trees," *SIAM J. Applied Math.*, 16, no. 1(1-29).

S. L. HAKIMI, 1971, "Steiner's Problem in Graphs and Its Implications," *Networks*, 1(113-133).

M. HANAN, Mar. 1966, "On Steiner's Problem with Rectilinear Distance," *SIAM J. Applied Math.*, 14, no. 2(255-265).

Z. A. MELZAK, 1961, "On the Problem of Steiner," *Canadian Mathematical Bulletin*, 4(143-148).

R. T. WONG, 1984, "A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph," *MP*, 28(271-287).

References on Optimum Branchings

F. BOCK, 1971, "An Algorithm to Construct a Minimum Directed Spanning Tree in a Directed Network," PP 29-44 in *Developments in Operations Research*, Gordon and Breach, N.Y.

Y. J. CHU and T. H. LIU, 1965, "On the Shortest Arborescence of a Directed Graph," *Sin. Sinica*, 14(1396-1400).

J. EDMONDS, 1967, "Optimum Branchings," *J. Res. Nat. Bur. Standards*, Section B, 71(233-240).

R. M. KARP, 1971, "A Simple Derivation of Edmonds' Algorithm for Optimum Branchings," *Networks*, 1(265-272).

R. E. TARJAN, 1977, "Finding Optimum Branchings," *Networks*, 7(25-35).

Index

For each index entry we provide the page number where it is defined or discussed first.

Boruvka's method 640
Branching 650

Greedy method 636
Selection 637
Worst case performance 643

Kruskal's method 638

Min. length spanning trees 628
Constrained 630, 648
Linear representation 655
Optimum face 661
Unconstrained 630
Myopic method 636

Permanent labels 636
Prim's method 636

Ranking method 645

Steiner 649
Points 649
Trees 649

Temporary labels 636