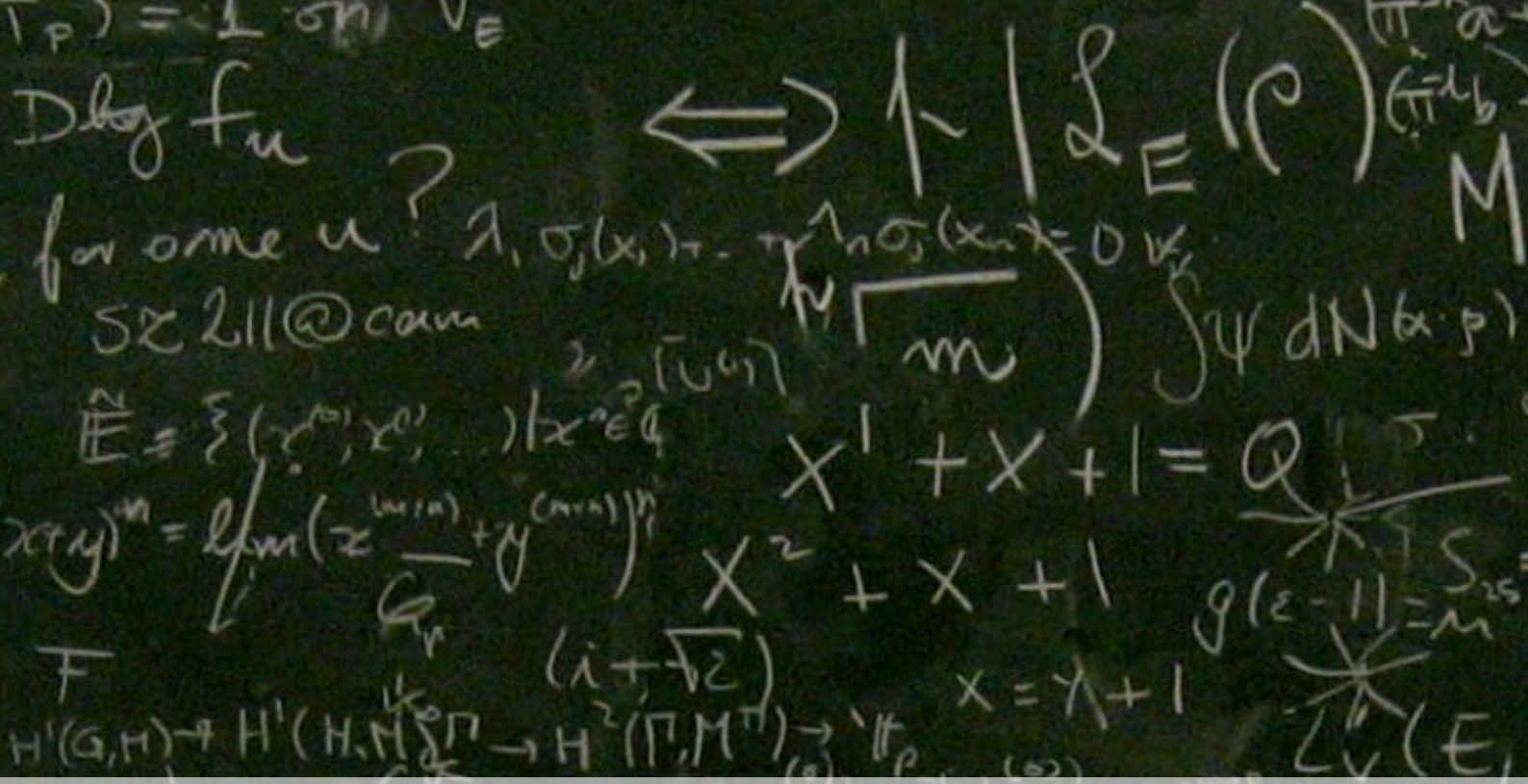


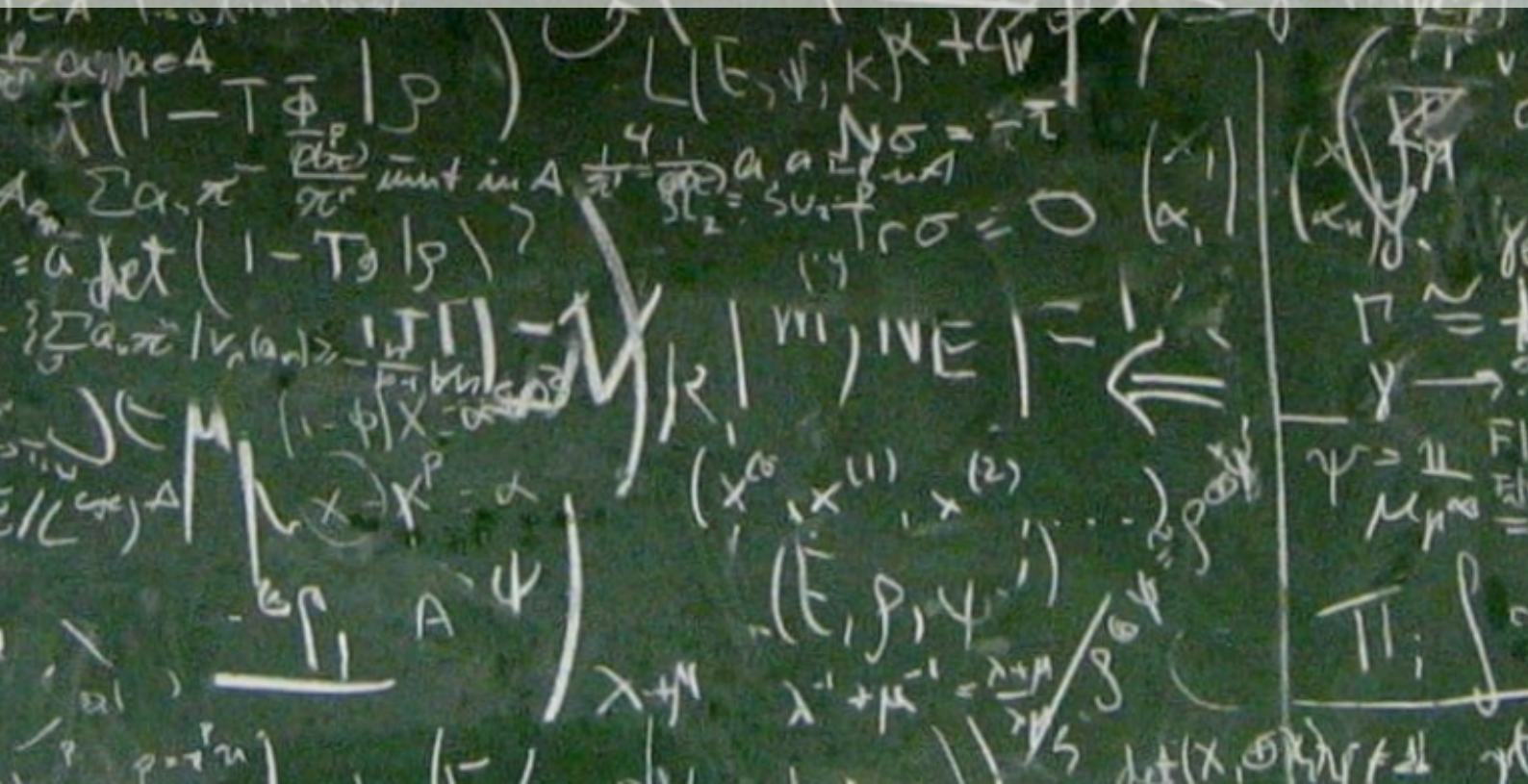
Optimization Methods for ICT

Lecture Notes

Roberto Roberti

Department of Information Engineering, University of Padua

roberto.roberti@unipd.it



Contents

Mandatory Textbook

AMO93 R. K. Ahuja, T. L. Magnanti, J. B. Orlin.
Network Flows: Theory, Algorithms, and Applications.
Wiley, 1993

1. Introduction: [AMO93] Chapter 1
2. Linear Programming: [AMO93] Appendix C
3. Paths, Trees, and Cycles: [AMO93] Chapter 2
4. Shortest Paths: [AMO93] Chapter 4
5. Maximum Flows: [AMO93] Chapter 6
6. Minimum Spanning Trees: [AMO93] Chapter 13
7. Minimum Cost Flows: [AMO93] Chapter 9

Chapter 1

Introduction

1.1 Introduction

Networks appear everywhere in our daily lives. Think about electrical and power networks, telephone networks, highway systems, rail networks, airline service networks, manufacturing and distribution networks, computer networks, etc. In these problem domains, we wish to move some entity (electricity, consumer products, people, vehicles) from one point to another in the underlying network and to do so as efficiently as possible, both to provide good service to the users and use the underlying transmission facilities effectively.

In this course, we learn how to model application settings as mathematical objects known as Network Flow Problem (NFP) and study algorithms to solve the resulting models. NFP are studied in several research domains, including applied mathematics, computer science, engineering, management, and operations research. We wish to address basic questions, such as:

1. Shortest Path Problem (SPP): What is the best way to traverse a network to get from one point to another as cheaply as possible?
2. Maximum Flow Problem (MFP): If a network has capacities on arc flows, how can we send as much flow as possible between two points while honoring the arc flow capacities?
3. Minimum Cost Flow Problem (MCFP): If we incur a cost per unit flow on a network with arc capacities and need to send units of a good that reside at one or more points in the network to one or more other points, how can we send the material at minimum possible cost?

From a pure mathematical perspective, most of these problems are trivial to solve as we need to consider a finite number of alternatives. A traditional mathematician might argue that the problems are well solved: simply enumerate all feasible solutions, and choose the best one. Unfortunately, this approach is inapplicable since the number of solutions can be very large. So instead, we devise algorithms that are *good*, i.e., whose computation time is *reasonable* for problems met in practice.

1.2 Network Flow Problems

The Minimum Cost Flow Problem (MCFP) is a fundamental, easy-to-state problem: we wish to find a least-cost shipment of a commodity through a network to satisfy demands at certain nodes from

available supplies at other nodes. The MCFP has many applications: the distribution of a product from manufacturing plants to warehouses or from warehouses to retailers; the flow of raw material and intermediate goods through machine stations in a production line; the routing of automobiles through an urban street network; and the routing of calls through the telephone system.

Let $G = (N, A)$ be a directed network defined by a set N of n nodes and a set A of m arcs. Each arc $(i, j) \in A$ has an associated cost c_{ij} that denotes the cost per unit flow on that arc – the flow cost varies linearly with the flow. We also associate, with each arc $(i, j) \in A$, a capacity u_{ij} (i.e., the maximum flow on the arc) and a lower bound l_{ij} (i.e., the minimum flow on the arc). We associate, with each node $i \in N$, an integer number b_i representing its supply/demand. If $b_i > 0$, node i is a supply node; if $b_i < 0$, node i is a demand node with a demand of $-b_i$; and if $b_i = 0$, node i is a transshipment node. We assume $\sum_{i \in N} b_i = 0$. The decision variables in the MCFP are arc flows, represented by flow variables $x_{ij} \in \mathbb{R}_+$, for each arc $(i, j) \in A$. The MCFP can be formulated as follows

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{1.1a}$$

$$\text{s.t.} \sum_{j \in N : (i,j) \in A} x_{ij} - \sum_{j \in N : (j,i) \in A} x_{ji} = b_i \qquad \forall i \in N \tag{1.1b}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i, j) \in A \tag{1.1c}$$

Constraints (1.1b) are called mass balance constraints, stating that, for each node, its outflow (i.e., the flow emanating from it) minus its inflow (i.e., the flow entering the node) must be equal to b_i. Constraints (1.1c) stipulate that the flow of each arc must be between its lower bound and its capacity.

In most problems we consider, we assume that the data (i.e., arc capacities, arc costs, supply, demands, etc) are integral. This is the integrality assumption, which is not restrictive because (i) we can always transform rational data to integer data by multiplying them by a suitably large number, and (ii) we need to convert irrational numbers to rational numbers to represent them on a computer.

Some special cases of the MCFP play a central role in network optimization.

1.2.1 Shortest Path Problem

In the Shortest Path Problem (SPP), we wish to find a min-cost path from a specified source node s to another specified sink node t, assuming that each arc $(i, j) \in A$ has an associated cost c_{ij}. SPP arises in many problem domains, e.g., equipment replacement, project scheduling, cash flow management, message routing, and traffic flow. If we set $b_s = 1$, $b_t = -1$, and $b_i = 0$ for all other nodes in the MCFP, and we set $l_{ij} = 0$ and $u_{ij} = 1$, for each arc $(i, j) \in A$, we get the SPP.

1.2.2 Assignment Problem

The data of the Assignment Problem (AP) consist of two equally sized sets N_1 and N_2 (i.e., $|N_1| = |N_2|$), a collection of pairs $A \subseteq N_1 \times N_2$ representing possible assignments, and a cost c_{ij} associated with each $(i, j) \in A$. In the AP, we wish to pair, at minimum cost, each object in N_1 with exactly one object in N_2, and vice versa. Examples of the AP include assigning people to projects, jobs to machines, tenants to apartments, swimmers to events in a swimming meet, and school graduates to available internships. The AP is a MCFP in a network $G = (N_1 \cup N_2, A)$ with $b_i = 1$ for all $i \in N_1$, $b_i = -1$ for all $i \in N_2$, and $u_{ij} = 1$ for all $(i, j) \in A$.

Chapter 2

Linear Programming

2.1 Introduction

Linear Programming (LP)[1] is the core model of constrained optimization. Developed by George Dantzig in 1947, the simplex method for solving LP is a cornerstone of applied mathematics, computer science, and operations research. The simplex method has been applied to thousands of applications in a wide variety of fields, such as agriculture, communications, computer science, engineering, finance, manufacturing, transportation, and urban logistics. The majority of the models studied in this course are either LP or Integer Linear Programming (ILP) extensions of LP.

An LP is an optimization problem with (i) q non-negative decision variables, (ii) a linear objective function, (iii) a set of p linear constraints, and (iv) a set of non-negativity restrictions imposed upon the decision variables.

An LP takes the *standard* form

$$\min \sum_{j=1}^{q} c_j x_j \tag{2.1a}$$

$$\text{s.t.} \sum_{j=1}^{q} a_{ij} x_j = b_i \qquad \forall i = 1, \dots, p \tag{2.1b}$$

$$x_j \in \mathbb{R}_+ \qquad \forall j = 1, \dots, q \tag{2.1c}$$

Without loss of generality, an LP in the standard form assumes that (i) the objective function is in minimization form, and (ii) the p linear constraints are in the "equal to" form and $b_i \in \mathbb{R}_+$.

In economic planning, each decision variable x_j models the level of the jth activity, each constraint corresponds to a scarce resource available in b_i quantity, and a_{ij} is the amount of the ith resource consumed per unit of the jth activity. The model seeks the best uses of resources to maximize the revenue.

In matrix notation, an LP has the form

$$\min_{\boldsymbol{x} \in \mathbb{R}_+^q} \{ \boldsymbol{cx} \mid \boldsymbol{Ax} = \boldsymbol{b} \} \tag{2.2}$$

[1] LP can refer to linear programming, linear program, or linear problem

where $\boldsymbol{A} = (a_{ij})$ has p rows and q columns, $\boldsymbol{c} \in \mathbb{R}^q$ is a q-dimensional row vector, $\boldsymbol{x} \in \mathbb{R}^q_+$ is a q-dimensional column vector, and $\boldsymbol{b} \in \mathbb{R}^p_+$ is a p-dimensional column vector. We assume that the rows of $\boldsymbol{A}$ are linearly independent, and $\boldsymbol{A}$ has full row rank.

Exercise 2.1.

A firm produces two types of wood desks (rolltop and regular). Wood is cut to a uniform thickness of 1 cm, so wood is measured in units of square meters (m^2). One rolltop desk requires 10 m^2 of pine, 4 m^2 of cedar, and 15 m^2 of maple. One regular desk requires 20 m^2 of pine, 16 m^2 of cedar, and 10 m^2 of maple. The desks yield €115 and €90 profit per sale, respectively.
The firm has currently available 200 m^2 of pine, 128 m^2 of cedar, and 220 m^2 of maple. The firm has backlogged orders for both desks and would like to produce the number of rolltop and regular desks that maximize the profit.
How many of each should it produce?

First, we identify the input data

- D set of desk types
- W set of wood types
- p_d unit profit for desk type $d \in D$
- b_w m^2 of wood type $w \in W$ available
- a_{dw} m^2 of wood type $w \in W$ needed for a single desk of type $d \in D$

Then, we introduce the decision variables

- $x_d \in \mathbb{R}_+$ number of desks of type $d \in D$ to produce

Finally, we write the LP

$$\max \sum_{d \in D} p_d x_d \tag{2.3a}$$

$$\text{s.t.} \sum_{d \in D} a_{dw} x_d \leq b_w \quad \forall w \in W \tag{2.3b}$$

$$x_d \in \mathbb{R}_+ \quad \forall d \in D \tag{2.3c}$$

The objective function (2.3a) aims at maximizing the total profit. Constraints (2.3b) ensure that the square meters of each wood type needed to produce the desks do not exceed the square meters available. Constraints (2.3c) are non-negativity constraints.

Notice that by defining the $\boldsymbol{x}$ variables as real, we are not forcing to produce an integer number of desks of each type. Indeed, the problem should have been formulated as an ILP (rather than an LP). However, a feasible non-necessarily optimal solution can be achieved by properly rounding the optimal solution of (2.3).

Exercise 2.2.

The Grand Prix Cars Company manufactures cars in three plants and ships them to four regions of the country. The plants can supply 450, 600, and 500 cars, respectively. The customer demands by region are 450, 200, 300, and 300, respectively. The unit costs of shipping a car from each plant to each region are listed below

	Region 1	Region 2	Region 3	Region 4
Plant 1	131	218	266	120
Plant 2	250	116	263	278
Plant 3	178	132	122	180

Grand Prix wants to find the lowest-cost shipping plan for meeting the demands of the four regions without exceeding the capacities of the plants. Formulate the problem as an LP.

First, we identify the input data

- O set of origins/plants
- D set of destinations/regions
- s_i supply of origin $i \in O$
- d_j demand of destination $j \in D$
- c_{ij} unit shipping cost from origin $i \in O$ to destination $j \in D$

Then, we introduce the decision variables

- $x_{ij} \in \mathbb{R}_+$ number of cars to ship from origin $i \in O$ to destination $j \in D$

Finally, we write the LP

$$\min \sum_{i \in O} \sum_{j \in D} c_{ij} x_{ij} \tag{2.4a}$$

$$\text{s.t.} \sum_{j \in D} x_{ij} \leq s_i \qquad \forall i \in O \tag{2.4b}$$

$$\sum_{i \in O} x_{ij} = d_j \qquad \forall j \in D \tag{2.4c}$$

$$x_{ij} \in \mathbb{R}_+ \qquad \forall i \in O \quad \forall j \in D \tag{2.4d}$$

The objective function (2.4a) aims at minimizing the total shipping costs. Constraints (2.4b) ensure that each origin's supply is not exceeded. Constraints (2.4c) guarantee that customer demands are fulfilled. Constraints (2.4d) are non-negativity constraints.

Notice that by defining the $\boldsymbol{x}$ variables as real, we are not forcing the solution to send an integer number of cars from each pair of origin and destination. However, as the matrix $\boldsymbol{A}$ is totally unimodular and we can assume that all s_i and d_j are integer, any feasible solution of (2.4) is integer.[2]

[2] More details about this are beyond the scope of the course

2.2 Graphical Solution Procedure

An LP with two or three variables has a convenient graphical representation that helps understand the nature of LP and the simplex method, which is the most popular method for solving an LP. Consider the following LP

$$\begin{aligned} \max \quad & z = && x_1 + x_2 && \text{(2.5a)} \\ & \text{s.t.} && 2x_1 + 3x_2 \leq 12 && \text{(2.5b)} \\ & && x_1 \leq 4 && \text{(2.5c)} \\ & && x_2 \leq 3 && \text{(2.5d)} \\ & && x_1, x_2 \in \mathbb{R}_+ && \text{(2.5e)} \end{aligned}$$

The shaded region of Figure 2.1 is the set of feasible solutions for (2.5). This set is a *polyhedron*. The points A, B, C, D, and E are the *extreme points* of the polyhedron, where x is an extreme point if it is not a strict convex combination of two distinct points (x^1 and x^2) of the polyhedron, that is, it cannot be represented as $x = \theta x^1 + (1 - \theta)x^2$, for $0 < \theta < 1$.

The LP seeks a point (x_1, x_2) in the polyhedron $ABCDEA$ that achieves the maximum possible value of $x_1 + x_2$, i.e., the largest value z for which line $z = x_1 + x_2$ has at least a point in common with $ABCDEA$. The maximum value is $\frac{16}{3}$ achieved at extreme point $D = (4, \frac{4}{3})$. Indeed, every LP always has an extreme point solution as one of its optimal solutions. To solve an LP, we can focus on a finite number of points: the extreme points. The simplex method makes use of this extreme point property: it starts at some extreme point and visits adjacent extreme points to improve the objective function value until it reaches an optimal extreme point.

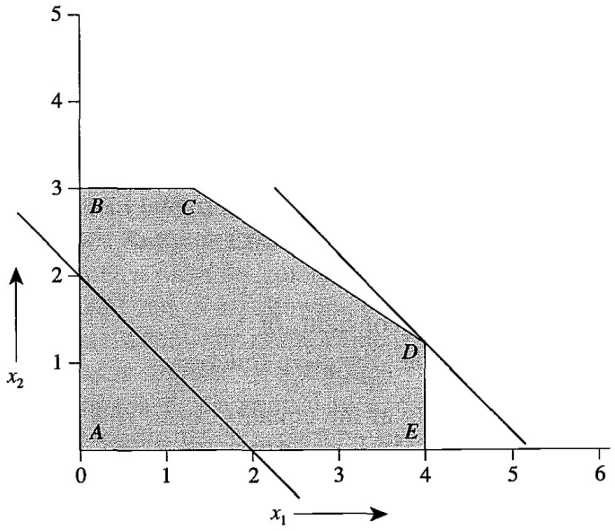


Figure 2.1: Set of feasible solutions for problem (2.5)

2.3 Linear Programming Duality

Each LP, which we call the *primal problem*, has an associated LP, called the *dual problem*. The primal and dual problems define *duality theory*, which we summarize in the following.

Let the primal problem be defined as the following LP in the *canonical form*

$$\min \sum_{j=1}^{q} c_j x_j \tag{2.6a}$$

$$\text{s.t.} \sum_{j=1}^{q} a_{ij} x_j \geq b_i \qquad \forall i = 1, \dots, p \tag{2.6b}$$

$$x_j \in \mathbb{R}_+ \qquad \forall j = 1, \dots, q \tag{2.6c}$$

Let $\pi_i \in \mathbb{R}_+$ be a dual variable associated with the ith constraint (2.6b). The dual of (2.6) is

$$\max \sum_{i=1}^{p} b_i \pi_i \tag{2.7a}$$

$$\text{s.t.} \sum_{i=1}^{p} a_{ij} \pi_i \leq c_j \qquad \forall j = 1, \dots, q \tag{2.7b}$$

$$\pi_i \in \mathbb{R}_+ \qquad \forall i = 1, \dots, p \tag{2.7c}$$

Each constraint in the primal has an associated variable in the dual (and vice versa). The right-hand side of each constraint of the primal (dual, respectively) becomes the cost coefficient of the associated dual (primal, respectively) variable. The dual of the dual is the primal.

The following table summarizes the relationships between the primal and the dual ($\boldsymbol{A}_i$ represents the ith row of matrix $\boldsymbol{A}$, and $\boldsymbol{a}_j^T$ the transpose of the jth column of matrix $\boldsymbol{A}$)

Primal	Dual
min	max
q variables	q constraints
p constraints	p variables
constraint $\boldsymbol{A}_i \boldsymbol{x} \leq b_i$	variable $\pi_i \leq 0$
constraint $\boldsymbol{A}_i \boldsymbol{x} \geq b_i$	variable $\pi_i \geq 0$
constraint $\boldsymbol{A}_i \boldsymbol{x} = b_i$	variable $\pi_i \in \mathbb{R}$
variable $x_j \geq 0$	constraint $\boldsymbol{a}_j^T \boldsymbol{\pi} \leq c_j$
variable $x_j \leq 0$	constraint $\boldsymbol{a}_j^T \boldsymbol{\pi} \geq c_j$
variable $x_j \in \mathbb{R}$	constraint $\boldsymbol{a}_j^T \boldsymbol{\pi} = c_j$

Exercise 2.3.

Write the dual of problem (2.3), which is

$$\max \sum_{d \in D} p_d x_d$$

$$\text{s.t.} \sum_{d \in D} a_{dw} x_d \leq b_w \qquad \forall w \in W$$

$$x_d \in \mathbb{R}_+ \qquad \forall d \in D$$

Let $\boldsymbol{\pi} \in \mathbb{R}_+^{|W|}$ be the dual variables associated with constraints (2.3b). The dual of (2.3) is

$$\begin{aligned} \min \sum_{w \in W} & b_w \pi_w \\ \text{s.t.} \sum_{w \in W} & a_{dw} \pi_w \geq p_d && \forall d \in D \\ & \pi_w \in \mathbb{R}_+ && \forall w \in W \end{aligned}$$

Exercise 2.4.

Write the dual of problem (2.4), which is

$$\begin{aligned} \min \sum_{i \in O} \sum_{j \in D} & c_{ij} x_{ij} \\ \text{s.t.} \sum_{j \in D} & x_{ij} \leq s_i && \forall i \in O \\ \sum_{i \in O} & x_{ij} = d_j && \forall j \in D \\ & x_{ij} \in \mathbb{R}_+ && \forall i \in O \quad \forall j \in D \end{aligned}$$

Let $\boldsymbol{u} \in \mathbb{R}_-^{|O|}$ and $\boldsymbol{v} \in \mathbb{R}^{|D|}$ be the dual variables associated with constraints (2.4b) and (2.4c), respectively. The dual of (2.4) is

$$\begin{aligned} \max \sum_{i \in O} & s_i u_i + \sum_{j \in D} d_j v_j \\ \text{s.t.} \quad & u_i + v_j \leq c_{ij} && \forall i \in O \quad \forall j \in D \\ & u_i \in \mathbb{R}_- && \forall i \in O \\ & v_j \in \mathbb{R} && \forall j \in D \end{aligned}$$

Theorem 2.1. ***(Weak Duality)*** *If $\boldsymbol{x}$ is any feasible solution of the primal and $\boldsymbol{\pi}$ is any feasible solution of the dual problem, then*

$$\sum_{i=1}^{p} b_i \pi_i \leq \sum_{j=1}^{q} c_j x_j$$

Proof. Multiplying the ith constraint (2.6b) by π_i and adding them all yields

$$\sum_{i=1}^{p} b_i \pi_i \leq \sum_{i=1}^{p} \pi_i \Big(\sum_{j=1}^{q} a_{ij} x_j \Big), \tag{2.9}$$

while multiplying the jth constraint (2.7b) by x_j and adding them all yields

$$\sum_{j=1}^{q} x_j \Big(\sum_{i=1}^{p} a_{ij} \pi_i \Big) \leq \sum_{j=1}^{q} c_j x_j \tag{2.10}$$

Therefore

$$\sum_{i=1}^{p} b_i \pi_i \leq \sum_{i=1}^{p} \pi_i \Big(\sum_{j=1}^{q} a_{ij} x_j \Big) = \sum_{j=1}^{q} x_j \Big(\sum_{i=1}^{p} a_{ij} \pi_i \Big) \leq \sum_{j=1}^{q} c_j x_j \tag{2.11}$$

which proves the theorem. ∎

The weak duality theorem has a number of consequences.

Lemma 2.1. *The objective function value of any feasible dual solution is a Lower Bound (LB) on the objective function value of every feasible primal solution.*

Lemma 2.2. *The objective function value of any feasible primal solution is an Upper Bound (UB) on the objective function value of every feasible dual solution.*

Lemma 2.3. *If the primal (dual, respectively) has an unbounded solution, the dual (primal, respectively) is infeasible.*

Lemma 2.4. *If the primal has a feasible solution $\boldsymbol{x}$ and the dual has a feasible solution $\boldsymbol{\pi}$ such that $\sum_{i=1}^{p} b_i \pi_i = \sum_{j=1}^{q} c_j x_j$, then $\boldsymbol{x}$ is an optimal primal solution, and $\boldsymbol{\pi}$ is an optimal dual solution.*

Theorem 2.2. ***(Strong Duality)*** *If the primal and dual problems are feasible, there exists an optimal primal solution $\boldsymbol{x}^*$ and an optimal dual solution $\boldsymbol{\pi}^*$ and $\sum_{i=1}^{p} b_i \pi_i^* = \sum_{j=1}^{q} c_j x_j^*$.*

Property 2.1. ***(Complementary Slackness)*** *A pair $(\boldsymbol{x}, \boldsymbol{\pi})$ of the primal and dual feasible solutions satisfies the complementary slackness property if*

$$\pi_i \Big(\sum_{j=1}^{q} a_{ij} x_j - b_i \Big) = 0 \quad \forall i = 1, \ldots, p \tag{2.12}$$

and

$$x_j \Big(c_j - \sum_{i=1}^{p} a_{ij} \pi_i \Big) = 0 \quad \forall j = 1, \ldots, q \tag{2.13}$$

Observe that $\sum_{j=1}^{q} a_{ij} x_j - b_i$ is the amount of slack in the ith primal constraint (2.6b), and π_i is the corresponding dual variable. Similarly, $c_j - \sum_{i=1}^{p} a_{ij} \pi_i$ is the amount of slack in the jth dual constraint (2.7b), and x_j is the corresponding primal variable. The complementary slackness property states that, for each primal and dual solution that fulfill the complementary slackness property, the product of the slack in the constraint and its associated variable is zero.

Theorem 2.3. ***(Complementary Slackness Optimality Conditions)*** *A feasible primal solution $\boldsymbol{x}$ and a feasible dual solution $\boldsymbol{\pi}$ are optimal solutions of the primal and dual problems if and only if they satisfy the complementary slackness property.*

Chapter 3

Paths, Trees, and Cycles

3.1 Introduction

Graphs and networks arise everywhere, so several disciplines have contributed important ideas to the evolution of network flows. The main drawback of this is that the literature on networks and graph theory lacks unity and uniform conventions, notation, and terminology.

In this chapter, we have three objectives. First, we bring together many basic definitions of network flows and graph theory, and we set the notation we will use. Second, we introduce several different data structures used to represent networks within a computer and discuss the relative advantages and disadvantages. Third, we discuss a number of different ways to transform a network flow problem and obtain an equivalent model.

3.2 Notation and Definitions

In this section, we give several basic definitions from graph theory and present some basic notation. We also state some elementary properties of graphs.

Directed Graph A *directed graph* (or directed network) $G = (N, A)$ consists of a set N of nodes (or vertexes) and a set A of arcs whose elements are ordered pairs of distinct nodes. Figure 3.1 gives an example of a directed graph, where $N = \{1, 2, 3, 4, 5, 6, 7\}$ and $A = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 6), (4, 5), (4, 7), (5, 2), (5, 3), (5, 7), (6, 7)\}$. An arc (i, j) behaves like a one-way street and permits flow from node i to node j only, but not vice versa. Some numerical values (i.e., costs, capacities, etc) are typically associated with the arcs. We let n denote the number of nodes (i.e., $n = |N|$) and m denote the number of arcs (i.e., $m = |A|$) in G

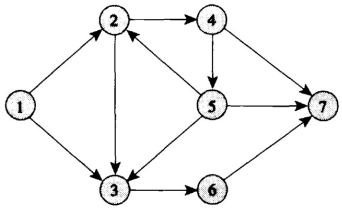


Figure 3.1: Directed graph

Undirected Graph An *undirected graph* $G = (N, E)$ consists of a set N of n nodes (or vertexes) and a set E of m edges whose elements are unordered pairs of distinct nodes. Figure 3.2 gives an example of an undirected graph. In an undirected graph, we can refer to an edge joining the node pair i and j as either $\{i, j\}$ or $\{j, i\}$. An edge $\{i, j\}$ can be regarded as a two-way street with flow permitted in both directions

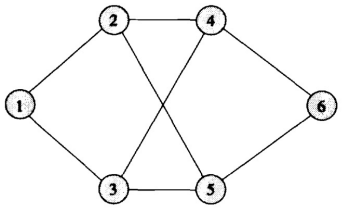


Figure 3.2: Undirected graph

Tail and Head An arc (i, j) has two *endpoints* i and j: i is the *tail*, and j is the *head*. We say that the arc (i, j) *emanates* from node i and *terminates* at node j. An arc (i, j) is *incident* to nodes i and j. The arc (i, j) is an *outgoing arc* of node i and an *incoming arc* of node j. Whenever an arc $(i, j) \in A$, we say that node j is *adjacent* to node i

Degree The *indegree* of a node is the number of its incoming arcs and its *outdegree* is the number of its outgoing arcs. The *degree* of a node is the sum of its indegree and outdegree. For example, in Figure 3.1, node 3 has an indegree of 3, an outdegree of 1, and a degree of 4. Notice that the sum of indegrees of all nodes equals the sum of outdegrees of all nodes and both are equal to m

Adjacency List The *arc adjacency list* $A(i)$ of a node i is the set of arcs emanating from i, i.e., $A(i) = \{(i, j) \in A \mid j \in N\}$. The *node adjacency list* $A(i)$ is the set of nodes adjacent to i; in this case, $A(i) = \{j \in N \mid (i, j) \in A\}$. We will often omit the terms "arc" and "node" and simply refer to the adjacency list; in all cases, it will be clear from context whether we mean arc adjacency list or node adjacency list. Notice that $|A(i)|$ equals the outdegree of i. Since the sum of all node outdegrees equals m, we obtain the following property:

Property 3.1. $\sum_{i \in N} |A(i)| = m$

Subgraph A graph $G' = (N', A')$ is a *subgraph* of $G = (N, A)$ if $N' \subseteq N$ and $A' \subseteq A$. We say that $G' = (N', A')$ is the subgraph of G *induced* by N' if A' contains each arc of A with both endpoints in N'

Walk A *walk* in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes $i_1 - i_2 - \ldots - i_r$ satisfying the property that for $k = 1, \ldots, r - 1$ either $(i_k, i_{k+1}) \in A$ or $(i_{k+1}, i_k) \in A$. A walk in the graph of Figure 3.1 is $1 - 2 - 5 - 7$. A *directed walk* is a walk such that $(i_k, i_{k+1}) \in A$ for $k = 1, \ldots, r - 1$. A directed walk in the graph of Figure 3.1 is $1 - 2 - 4 - 5 - 2 - 3$

Path A *path* is a walk without any repetition of nodes, e.g., $1 - 2 - 5 - 7$ is a path of the graph of Figure 3.1. A *directed path* is a directed walk without any repetition of nodes

Cycle A *cycle* is a path $i_1 - i_2 - \ldots - i_r$ such that $i_1 = i_r$. A *directed cycle* is a directed path such that the first and the last nodes coincide

Acyclic Graph A graph is a *acyclic* if it does not contain any directed cycles

Connectivity Two nodes i and j are connected if the graph contains at least one path from i to j. A graph is *connected* if every pair of its nodes is connected; otherwise, the graph is *disconnected*

Cut A *cut* is a partition of the node set N into two parts, S and $\overline{S} = N \setminus S$. Each cut defines a *cutset* consisting of those arcs that have one endpoint in S and another endpoint in $\overline{S}$. We refer to the cutset defined by S and $\overline{S}$ as $A(S, \overline{S})$. Figure 3.3 shows the cutset $A(S, \overline{S}) = \{(2, 4), (3, 6), (5, 2), (5, 3)\}$ with $S = \{1, 2, 3\}$ and $\overline{S} = \{4, 5, 6, 7\}$

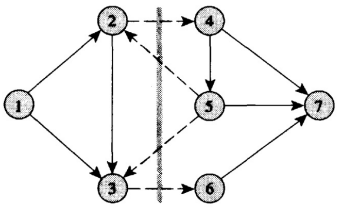


Figure 3.3: Cutset

s-t Cut An *s-t cut* is a cut defined with respect to two distinguished nodes s and t such that $s \in S$ and $t \in \overline{S}$. For instance, if $s = 1$ and $t = 6$, the cut depicted in Figure 3.3 is an *s-t cut*

Tree A *tree* is a connected graph that contains no cycle. In the graph of Figure 3.1, the arcs $\{(1, 2), (1, 3), (3, 6)\}$ represent a tree.

Property 3.2. *Trees have the following properties*

1. *A tree on k nodes contains exactly k − 1 arcs*
2. *Every two nodes of a tree are connected by a unique path*

Spanning Tree A tree T is a *spanning tree* of G if T is a spanning subgraph of G. Figure 3.4 shows two spanning trees of the graph shown in Figure 3.1. Every spanning tree of a connected n-node graph G has $(n - 1)$ arcs. We refer to the arcs belonging to a spanning tree T as *tree arcs* and arcs not belonging to T as *non-tree arcs*

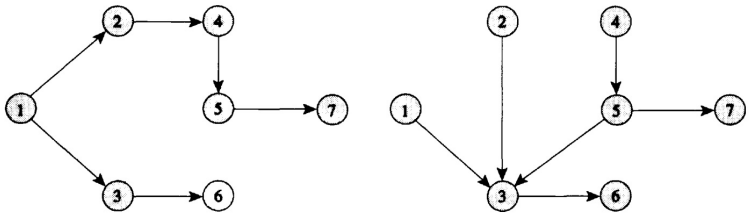


Figure 3.4: Spanning trees

Exercise 3.1.

The first paper on graph theory was written by Leonhard Euler in 1736. He started with the following mathematical puzzle:

The city of Konigsberg has seven bridges, arranged as shown in Figure 3.5. Is it possible to start at some place in the city and cross every bridge exactly once?

Either specify such a tour or prove that it is impossible to do so.

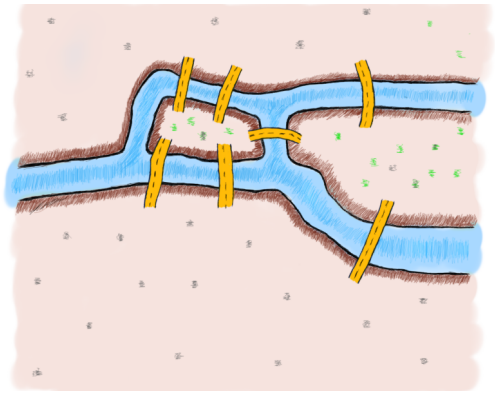


Figure 3.5: The Seven Bridges of Konigsberg

Euler proved that such a tour cannot exist. Indeed, he observed that an odd-degree node must be at the beginning and at the end of the trip. Since there can only be one beginning and one end, there can only be two odd-degree nodes. Since the bridge problem has four odd-degree nodes, it is impossible to do so!

Exercise 3.2.

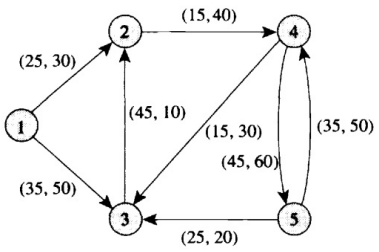
At the beginning of a dinner party, several participants shake hands with each other. Show that the participants that shook hands an odd number of times must be even in number

Construct an undirected graph with a node corresponding to each participant and an edge between two nodes if and only if the corresponding participants shook hands with one another. The number of people shaking hands an odd number of times equals the number of nodes having odd degree. Let δ_i denote the degree of node i. Notice that $\varphi = \sum_{i \in N} \delta_i$ is even because each edge contributes 2 to φ. Hence, the number of nodes having odd degrees is even; otherwise, φ would not be even.

3.3 Graph Representations

3.3.1 Node-Arc Incidence Matrix

The *node-arc incidence matrix* representation, or simply the *incidence matrix* representation, represents a graph $G = (N, A)$ as the constraint matrix of the MCFP problem that we discussed in Section 1.2. This representation stores the graph as an $n \times m$ matrix $\mathcal{N}$ that contains one row for each node and one column for each arc. The column corresponding to arc $(i, j) \in A$ has only two nonzero elements: a +1 in the row corresponding to node i and a −1 in the row corresponding to node j. The right part of Figure 3.6 gives this representation for the graph shown in the left part of the figure. The



	(1, 2)	(1, 3)	(2, 4)	(3, 2)	(4, 3)	(4, 5)	(5, 3)	(5, 4)
1	1	1	0	0	0	0	0	0
2	−1	0	1	−1	0	0	0	0
3	0	−1	0	1	−1	0	−1	0
4	0	0	−1	0	1	1	0	−1
5	0	0	0	0	0	−1	1	1

Figure 3.6: Incidence matrix representation

incidence matrix has a special structure: only $2m$ out of its nm entries are nonzero; all of its nonzero entries are +1 or −1; each column has exactly one +1 and one −1; the number of +1s in a row equals the outdegree of the corresponding node; the number of −1s in the row equals the indegree of the node. The incidence matrix contains few nonzero coefficients, so it is not space-efficient. However, it represents the constraint matrix of the MCFP and possesses several interesting theoretical properties (we will study them).

Additional information, such as arc costs and capacities, can be stored in m-dimensional arrays.

3.3.2 Node-Node Adjacency Matrix

The *node-node adjacency matrix* representation, or simply the *adjacency matrix* representation, stores the graph as an $n \times n$ matrix $\mathcal{H} = \{h_{ij}\}$. The matrix has a row and a column corresponding to every node, and its ijth entry h_{ij} equals 1 if $(i, j) \in A$ and equals 0 otherwise. Figure 3.7 specifies this representation for the graph of Figure 3.6. If we wish to store arc costs and capacities as well as the network topology, we can store this information in two additional $n \times n$ matrices. The adjacency matrix has n^2 elements, only m of which are nonzero. Consequently, this representation is space-efficient only if the graph is dense.

3.3.3 Adjacency Lists

The *adjacency list* representation stores the node adjacency list of each node as a linked list. A linked list is a collection of cells each containing one or more fields. The node adjacency list for node i is a linked list having $|A(i)|$ cells and each cell corresponds to an arc $(i, j) \in A$. The cell corresponding to the arc (i, j) has as many fields as the amount of information we wish to store. One data field

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	0	0	1	1	0

Figure 3.7: Adjacency matrix representation

stores node j. We might use two other data fields to store the arc cost c_{ij} and the arc capacity u_{ij}. Each cell contains one additional field, called the *link*, which stores a pointer to the next cell in the adjacency list. If a cell happens to be the last cell in the adjacency list, by convention we set its link to value zero. Since we need to store and access n linked lists, one for each node, we also need an array of pointers that point to the first cell in each linked list. We accomplish this by defining an n-dimensional array, $first$, whose element $first(i)$ stores a pointer to the first cell in the adjacency list of node i. If the adjacency list of node i is empty, we set $first(i) = 0$. Figure 3.8 specifies the adjacency list representation of the graph shown in Figure 3.6.

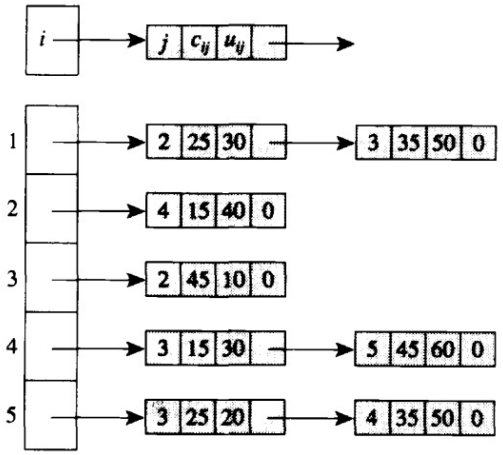


Figure 3.8: Adjacency list representation

3.3.4 Forward-Star Representation

The *forward-star* representation of a graph is similar to the adjacency list representation because it stores the node adjacency list of each node. However, instead of maintaining these lists as linked lists, it stores them in a single array. To develop this representation, we first associate a unique sequence number with each arc, thus defining an ordering of the arc list. We number the arcs in a specific order: first those emanating from node 1, then those emanating from node 2, and so on. We number the arcs emanating from the same node in an arbitrary fashion. We then sequentially store information about each arc (i.e., tail, head, cost, capacity, etc) in the arc list in different arrays. We also maintain

a pointer with each node i, denoted by $point(i)$, that indicates the smallest-numbered arc in the arc list that emanates from node i – if node i has no outgoing arcs, we set $point(i) = point(i + 1)$. Therefore, the forward-star representation stores the outgoing arcs of node i at positions $point(i)$ to $(point(i + 1) - 1)$ in the arc list. If $point(i) = point(i + 1)$, node i has no outgoing arc. We also set $point(1) = 1$ and $point(n + 1) = m + 1$. Figure 3.9 specifies the forward -star representation of the graph given in Figure 3.6 - notice that the array $tail(i)$ may also be removed.

	point
1	1
2	3
3	4
4	5
5	7
6	9

	tail	head	cost	capacity
1	1	2	25	30
2	1	3	35	50
3	2	4	15	40
4	3	2	45	10
5	4	3	15	30
6	4	5	45	60
7	5	3	25	20
8	5	4	35	50

Figure 3.9: Forward-star representation

Exercise 3.3.

Consider the graph of Figure 3.10. Specify (1) the incidence matrix, (2) the adjacency matrix, and (3) the forward-star representation.

1. Incidence matrix

	Arc					
Node	*(1,2)*	*(1,3)*	*(2,4)*	*(3,2)*	*(3,5)*	*(5,4)*
1	1	1	0	0	0	0
2	-1	0	1	-1	0	0
3	0	-1	0	1	1	0
4	0	0	-1	0	0	-1
5	0	0	0	0	-1	1

2. Adjacency matrix

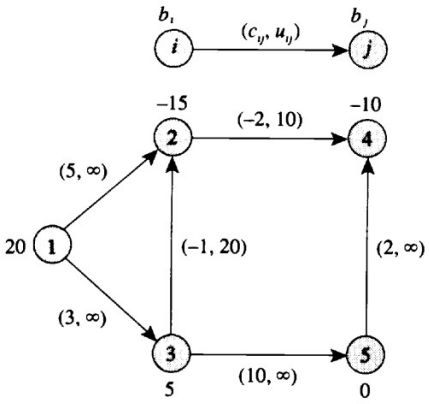


Figure 3.10: Directed graph

Node	*1*	*2*	*3*	*4*	*5*
1	0	1	1	0	0
2	0	0	0	1	0
3	0	1	0	0	1
4	0	0	0	0	0
5	0	0	0	1	0

3. Forward star representation

point					
1	*2*	*3*	*4*	*5*	*6*
1	3	4	6	6	7

tail	*head*	*cost*	*capacity*	*index*
1	2	5	∞	1
1	3	3	∞	2
2	4	-2	10	3
3	2	-1	20	4
3	5	10	∞	5
5	4	2	∞	6

3.4 Graph Transformations

Frequently, we require graph transformations to simplify a graph, show equivalences between different problems, or state a network problem in a standard form required by some software. We describe some of these important transformations by assuming that the network problem is a MCFP formulated with model (1.1).

3.4.1 Edges to Arcs

If the MCFP contains an edge $\{i, j\}$ with cost $c_{ij} \geq 0$ and capacity u_{ij}, model (1.1) needs two decision variables x_{ij} and x_{ji} to represent the flow along edge $\{i, j\}$ in the two directions. The objective function (1.1a) features the term $c_{ij}x_{ij} + c_{ij}x_{ji}$. Since $c_{ij} \geq 0$, it does not exist any optimal solutions where both $x_{ij} > 0$ and $x_{ji} > 0$. Notice that this transformation is correct only if the minimum flow required through edge $\{i, j\}$ is 0 (i.e., $l_{ij} = 0$) and cost c_{ij} is non-negative.

3.4.2 Removing Nonzero Lower Bounds

If an arc (i, j) has a nonzero lower bound l_{ij} on the arc flow x_{ij}, we replace x_{ij} by $x'_{ij} + l_{ij}$ in the problem formulation, where x'_{ij} represents the incremental flow beyond l_{ij}. The flow bound constraint becomes $l_{ij} \leq x'_{ij} + l_{ij} \leq u_{ij}$, or $0 \leq x'_{ij} \leq (u_{ij} - l_{ij})$. Making this substitution in the mass balance constraints (1.1b) decreases b_i by l_{ij} units and increases b_j by l_{ij} units. This substitution changes the objective function value by a constant $c_{ij}l_{ij}$. Figure 3.11 illustrates this transformation graphically.

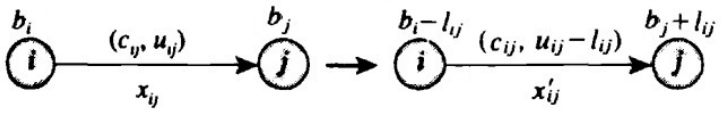


Figure 3.11: Removing nonzero lower bounds

Exercise 3.4.

Consider the MCFP shown in Figure 3.10. Suppose that arcs $(1, 2)$ and $(3, 5)$ have lower bounds equal to $l_{12} = l_{35} = 5$. Transform this problem to one where all arcs have zero lower bounds.

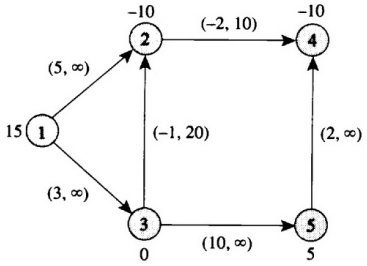


Figure 3.12: Directed graph after removing lower bounds $l_{12} = l_{35} = 5$

Chapter 4

Shortest Paths

4.1 Introduction

Shortest Path Problem (SPP) allure researchers and practitioners for several reasons: (i) they arise in a lot of applications, (ii) they can be solved efficiently, (iii) they capture the core ingredients of many NFP, and (iv) they frequently arise as sub-problems when solving other NFP.

We consider a directed graph $G = (N, A)$ with an *arc cost* (or *arc length*) c_{ij} associated with each arc $(i, j) \in A$. The graph has n nodes (i.e., $|N| = n$) and a distinguished node $s \in N$, called the *source*. Let $C = \max\{c_{ij} : (i, j) \in A\}$; moreover, let $\delta_i^+ \subseteq N$ ($\delta_i^- \subseteq N$, respectively) the set of successors (predecessors, resp.) of node $i \in N$, i.e., $\delta_i^+ = \{j \in N \,|\, (i, j) \in A\}$ ($\delta_i^- = \{j \in N \,|\, (j, i) \in A\}$, resp.). We define the *length of a directed path* as the sum of the lengths of arcs in the path. The SPP is to determine, for every node $i \in N \setminus \{s\}$, a shortest length directed path from node s to node i.

By introducing continuous variables $x_{ij} \in \mathbb{R}_+$ representing the flow through arc $(i, j) \in A$, the SPP can be formulated as follows

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{4.1a}$$

$$\text{s.t.} \sum_{j\in\delta_s^+} x_{sj} = n - 1 \tag{4.1b}$$

$$\sum_{i\in\delta_k^-} x_{ik} - \sum_{j\in\delta_k^+} x_{kj} = 1 \quad \forall k \in N \setminus \{s\} \tag{4.1c}$$

$$x_{ij} \in \mathbb{R}_+ \qquad \forall (i, j) \in A \tag{4.1d}$$

The objective function (4.1a) aims at minimizing the total cost of the selected paths. Constraint (4.1b) ensures that $n - 1$ units of commodity emanate from the source node. Constraints (4.1c) guarantee that a unit of commodity is intended for each non-source node. Constraints (4.1d) define the range of the decision variables.

Assumption 4.1. *All arc lengths are integers.*

Some algorithms need the integrality assumption. Others do not. Note that rational arc lengths can always be transformed to integer arc lengths by multiplying them by a suitably large number. Irrational numbers need to be converted to rational numbers to represent them on a computer. The integrality assumption is not restrictive in practice.

Assumption 4.2. *The graph contains a directed path from node s to every other node in the graph.*

This assumption can always be satisfied by adding a "fictitious" arc (s, i) of suitably large cost for each node i not connected to s by a directed path.

Assumption 4.3. *The graph does not contain a negative cycle (i.e., a directed cycle of negative length).*

Without this assumption, formulation (4.1) has an unbounded solution because an infinite flow can be sent through a negative cycle, and the SPP becomes significantly harder to solve.

Assumption 4.4. *The graph is directed.*

If the graph is undirected, it can be suitably transformed in a directed graph.

4.2 Applications

SPP arise in a wide variety of practical problems settings, both as stand-alone models and as subproblems in more complex problem settings. For example, they arise in telecommunications, transportation, urban traffic planning, project management, inventory planning, DNA sequencing, etc.

Exercise 4.1.

A production line consists of an ordered sequence of n production stages. Each stage has a manufacturing operation followed by a potential inspection. The products enter the production line in batches of size $B \geq 1$. As the items within a batch move through the manufacturing stages, the operations might introduce defects. The probability of producing a defect at stage i is α_i. Defects are not repairable, so we must scrap any defective item. After each stage, we can either inspect all of the items or none of them (we do not sample the items). The inspection identifies every defective item. The production line must end with an inspection station so that we do not ship any defective units.

Our decision-making problem is to find an optimal inspection plan that specifies at which stages we should inspect the items so that we minimize the total cost of production and inspection. The following cost data are available: (i) p_i, the manufacturing cost per unit in stage i; (ii) f_{ij}, the fixed cost of inspecting a batch after stage j given that we last inspected the batch after stage i; and (iii) g_{ij}, the variable unit cost for inspecting an item after stage j, given that we last inspected the batch after stage i.

How can this decision-making problem be formulated as a SPP?

We can formulate this inspection problem as a SPP on a graph with $n + 1$ nodes ($N = \{0, 1, \dots, n\}$). The graph contains an arc (i, j) for each pair of nodes i and j such that $i < j$, i.e., $A = \{(i, j) \mid i, j \in N : i < j\}$. Figure 4.1 shows the graph for an inspection problem with four stations. Each path in the graph from node 0 to node 4 defines an inspection plan. For example, the path $0 - 2 - 4$ implies that we inspect the batches after the second and the fourth stages.

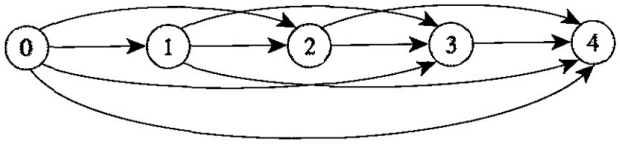


Figure 4.1: Graph of an inspection problem with four stations

The inspection problem can be solved as a SPP on such a graph, where costs c_{ij} are defined for each arc $(i, j) \in A$ as follows. Let $b_i = B\prod_{k=1}^{i}(1 - \alpha_k)$ denote the expected number of non-defective units at the end of stage i. Costs c_{ij} are defined as

$$c_{ij} = f_{ij} + b_i g_{ij} + b_i \sum_{k=i+1}^{j} p_k \qquad \forall (i, j) \in A$$

Exercise 4.2.

A hiker must decide which goods to include in her knapsack on a trip. She must choose from among p objects. Object i has a weight w_i (in kilos) and a utility u_i to the hiker. The objective is to maximize the utility of the hiker's trip subject to the weight limitation that she can carry no more than W kilos. How can this decision-making problem be formulated as a SPP?

We can define a graph with $p(W + 1) + 2$ nodes. The node set N contains two dummy nodes s and t plus $p(W + 1)$ nodes i^k (with $i = 1, \dots, p$ and $k = 0, 1, \dots, W$) representing the decision to use k kilos of capacity to carry a subset of the first i objects. The arc set A is defined as

$$A = A^s \cup A^0 \cup A^1 \cup A^t,$$

where

$$\begin{aligned}
A^s &= \{(s, 1^0), (s, 1^{w_1})\} \\
A^0 &= \{(i^k, (i+1)^k) \mid i = 1, \dots, p-1,\ k = 0, 1, \dots, W\} \\
A^1 &= \{(i^k, (i+1)^{k+w_{i+1}}) \mid i = 1, \dots, p-1,\ k = 0, 1, \dots, W : k + w_{i+1} \le W\} \\
A^t &= \{(p^k, t) \mid k = 0, 1, \dots, W\}
\end{aligned}$$

The cost c_{ij} of each arc $(i, j) \in A$ is defined as

$$c_{ij} = \begin{cases} 0 & \text{if } (i,j) \in \{(s, 1^0)\} \cup A^0 \cup A^t \\ -u_1 & \text{if } (i,j) = (s, 1^{w_1}) \\ -u_j & \text{if } (i,j) \in A^1 \end{cases}$$

Finding a shortest path from s to t corresponds to finding the highest-utility subset of objects to select, where traversing an arc (i, j) with a strictly negative cost means that object j is selected and traversing an arc (i, j) with a zero cost means that object j is not selected.

Consider the following example: $p = 4$, $W = 6$, $\boldsymbol{u} = \{40, 15, 20, 10\}$, $\boldsymbol{w} = \{4, 2, 3, 1\}$. The corresponding graph is depicted in Figure 4.2, where the number close to each arc is the opposite of

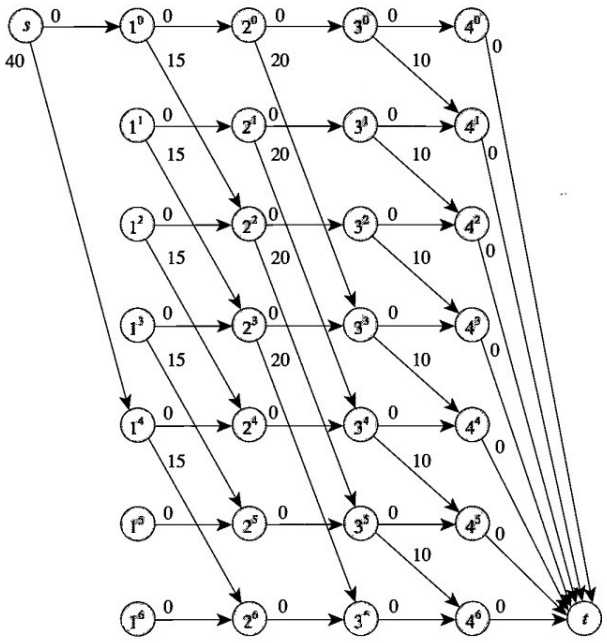


Figure 4.2: Graph representation of the example of the knapsack problem

its costs (i.e., $-c_{ij}$). The shortest path from s to t is $s - 1^4 - 2^6 - 3^6 - 4^6 - t$, corresponding to selecting objects 1 and 2 with a cost of −55 (i.e., a utility of 55 for the hiker).

This problem is known as the $0-1$ knapsack problem. Let $y_i \in \{0, 1\}$ be a binary variable equal to 1 if object i ($i = 1, \dots, p$) is selected (0 otherwise). The problem can be formulated as

$$\max \sum_{i=1}^{p} u_i y_i \tag{4.2a}$$

$$\text{s.t.} \sum_{i=1}^{p} w_i y_i \leq W \tag{4.2b}$$

$$y_i \in \{0, 1\} \qquad \forall i = 1, \dots, p \tag{4.2c}$$

4.3 Tree of Shortest Paths

In the SPP, we wish to determine a shortest path from the source node to all other $(n-1)$ nodes. How much storage do we need to store these paths? A naive answer is $(n-1)^2$ storage locations since each path contain at most $(n-1)$ arcs. Fortunately, $(n-1)$ storage locations are sufficient. Indeed, we can always find a tree rooted from the source with the property that the unique path from s to any node is a shortest path to that node.

Property 4.1. *If the path $s = i_1 - i_2 - \cdots - i_h = k$ is a shortest path from node s to node k, then for every $q = 2, 3, \dots, h-1$, the subpath $s = i_1 - i_2 - \cdots - i_q$ is a shortest path from s to node i_q.*

Proof. We provide an informal proof. In Figure 4.3, we assume that the shortest path $P_1 - P_3$ from s to k passes through a node p, but the sub-path P_1 up to node p is not a shortest path to node p. Suppose

instead that path P_2 is shorter than P_1. Then path $P_2 - P_3$ is shorter than $P_1 - P_3$, which contradicts the assumption of optimality of path $P_1 - P_3$. Therefore, P_1 must be a shortest path from s to p.∎

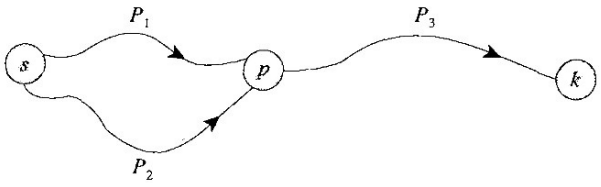


Figure 4.3: Optimality of sub-paths of shortest paths

Let $d(\cdot)$ denote the shortest path distances. The previous property implies that, if P is a shortest path from s to k, then $d(j) = d(i) + c_{ij}$ for every arc $(i,j) \in P$. The converse is also true: if $d(j) = d(i) + c_{ij}$ for every arc in a directed path P from s to node k, then P must be a shortest path.

Indeed, we can observe that

$$\begin{aligned} d(k) &= d(i_h) \\ &= d(i_{h-1}) + c_{i_{h-1}i_h} = \\ &= d(i_{h-2}) + c_{i_{h-2}i_{h-1}} + c_{i_{h-1}i_h} = \\ &= d(i_{h-3}) + c_{i_{h-3}i_{h-2}} + c_{i_{h-2}i_{h-1}} + c_{i_{h-1}i_h} = \\ &\vdots \\ &= d(i_1) + c_{i_1 i_2} + c_{i_2 i_3} + \ldots + c_{i_{h-1}i_h} = \\ &= 0 + \sum_{(i,j)\in P} c_{ij} \end{aligned}$$

Consequently, P is a directed path from s to k of length $d(k)$. Since, by assumption, $d(k)$ is the shortest path distance to k, P is a shortest path to k.

Property 4.2. *Let the vector* $\mathbf{d}$ *represent the shortest path distances. A directed path* P *from* s *to* k *is a shortest path if and only if* $d(j) = d(i) + c_{ij}$ *for every arc* $(i,j) \in P$.

4.4 Dijkstra's Algorithm

The Dijkstra's algorithm finds shortest paths from source node s to all other nodes in a graph with non-negative arc lengths. It maintains a distance label $d(i)$ with each node i, which is a UB on the shortest path length to node i, and the predecessor of node i ($pred(i)$) in the shortest path from s. At any intermediate step, the algorithm divides the nodes into two groups: *permanent* nodes (P) and *temporary* nodes (T). The distance label to any permanent node is the shortest distance from s to that node. For any temporary node, the distance label is a UB on the shortest path distance to that node.

The basic idea of the algorithm is to fan out from s and permanently label nodes in the order of their distances from s. Initially, s is given a permanent label of zero, and each other node j a temporary label of ∞. At each iteration, the algorithm selects a node i with the minimum temporary label (breaking ties arbitrarily), makes it permanent, and reaches out from the node - that is, scans arcs emanating from i to update the distances of the adjacent nodes. The algorithm terminates when

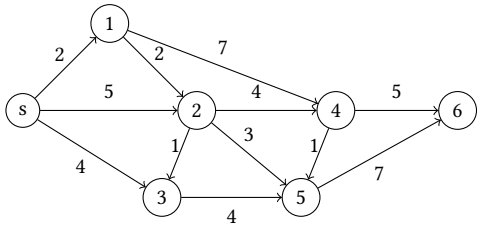
Algorithm 1: Dijkstra's Algorithm

```
1  P ← ∅; T ← N;
2  d(i) ← ∞ for each i ∈ N;
3  d(s) ← 0 and pred(s) ← 0;
4  while P ≠ N do
5      Let i ∈ T be a node for which d(i) = min{d(j) : j ∈ T};
6      P ← P ∪ {i}; T ← T \ {i};
7      for each (i, j) ∈ A do
8          if d(j) > d(i) + c_ij then
9              d(j) ← d(i) + c_ij;
10             pred(j) ← i;
```

all nodes are permanent. The pseudocode of the Dijkstra's algorithm is provided in Algorithm 1 – we do not prove the correctness of the algorithm, which can be established with inductive arguments.

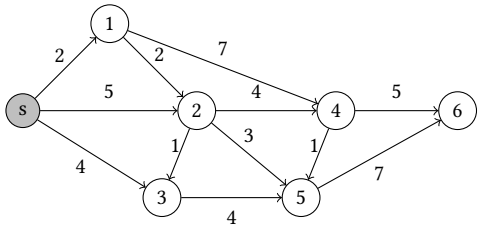
We illustrate the Dijkstra's algorithm with the following example.

- Initialization



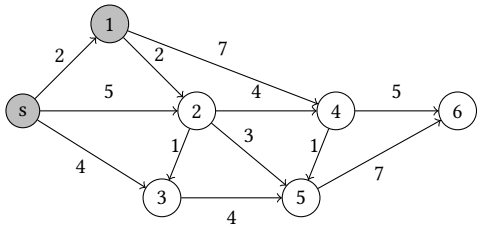
j	s	1	2	3	4	5	6
$d(j)$	0	∞	∞	∞	∞	∞	∞
$pred(j)$	0	-	-	-	-	-	-
i =? $P = \emptyset$ $T = \{s, 1, 2, 3, 4, 5, 6\}$							

- Iteration 1



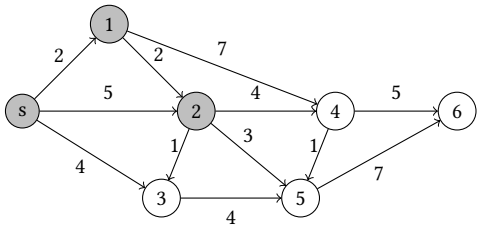
j	s	1	2	3	4	5	6
$d(j)$	0	2	5	4	∞	∞	∞
$pred(j)$	0	s	s	s	-	-	-
$i = s$ $P = \{s\}$ $T = \{1, 2, 3, 4, 5, 6\}$							

- Iteration 2



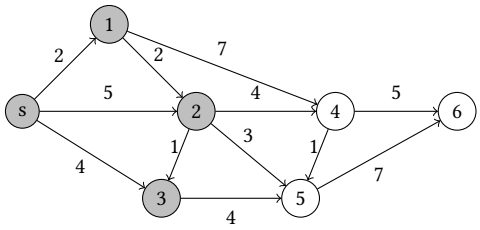
j	s	1	2	3	4	5	6
$d(j)$	0	2	4	4	9	∞	∞
$pred(j)$	0	s	1	s	1	-	-
$i = 1$ $P = \{s, 1\}$ $T = \{2, 3, 4, 5, 6\}$							

- Iteration 3



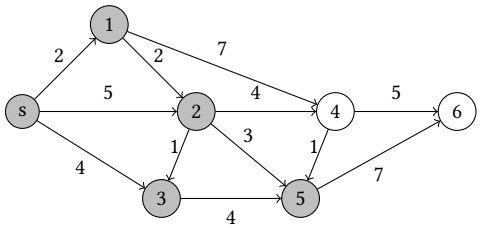
j	s	1	2	3	4	5	6
$d(j)$	0	2	4	4	8	7	∞
$pred(j)$	0	s	1	s	2	2	-
$i = 2$ $P = \{s, 1, 2\}$ $T = \{3, 4, 5, 6\}$							

- Iteration 4



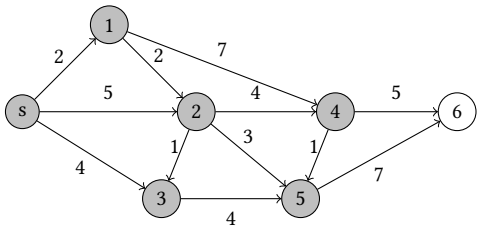
j	s	1	2	3	4	5	6
$d(j)$	0	2	4	4	8	7	∞
$pred(j)$	0	s	1	s	2	2	-
$i = 3$ $P = \{s, 1, 2, 3\}$ $T = \{4, 5, 6\}$							

- Iteration 5



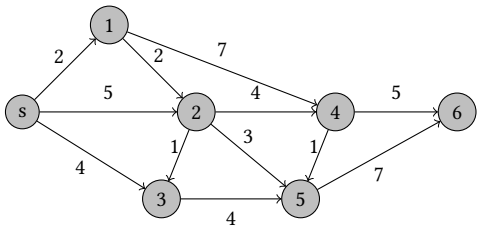
j	s	1	2	3	4	5	6
$d(j)$	0	2	4	4	8	7	14
$pred(j)$	0	s	1	s	2	2	5
$i = 5$ $P = \{s, 1, 2, 3, 5\}$ $T = \{4, 6\}$							

- Iteration 6



j	s	1	2	3	4	5	6
$d(j)$	0	2	4	4	8	7	13
$pred(j)$	0	s	1	s	2	2	4
$i = 4$ $P = \{s, 1, 2, 3, 4, 5\}$ $T = \{6\}$							

- Iteration 7



j	s	1	2	3	4	5	6
$d(j)$	0	2	4	4	8	7	13
$pred(j)$	0	s	1	s	2	2	4
$i = 6$ $P = \{s, 1, 2, 3, 4, 5, 6\}$ $T = \emptyset$							

Exercise 4.3.

Consider a set of n scalar numbers $a_1, a_2, \dots, a_n$ arranged in non-decreasing order of their values. We wish to partition these numbers into clusters so that (i) each cluster contains at least p numbers, (ii) each cluster contains consecutive numbers from the list, and (iii) the sum of the squared deviation of the numbers from their cluster means is as small as possible. Let $m(S) = \sum_{i \in S} a_i/|S|$ denote the mean of a set S of numbers defining a cluster. If the number a_k belongs to cluster S, the squared deviation of a_k from the cluster mean is $(a_k - m(S))^2$.

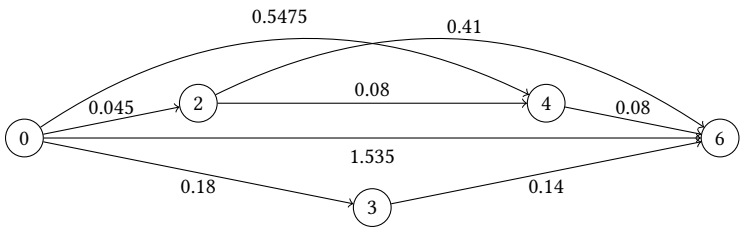
Formulate this cluster analysis problem, and provide an optimal solution of the following case: $p = 2$, $n = 6$, $a_1 = 0.5$, $a_2 = 0.8$, $a_3 = 1.1$, $a_4 = 1.5$, $a_5 = 1.6$, and $a_6 = 2.0$.

This cluster analysis problem can be formulated and solved as a SPP on the following graph $G = (N, A)$. The set of nodes N contains $n + 1$ nodes and is defined as $N = \{0, 1, \dots, n\}$. The node 0 is a dummy node, and the other n nodes correspond to the n scalars. The arc set is defined as $A = \{(i, j) \mid i, j \in N : i + p \le j\}$. Each arc $(i, j) \in A$ represents a cluster of scalars containing all scalars from $i+1$ to j. The cost c_{ij} of arc $(i, j) \in A$ is computed as $\sum_{k=i+1}^{j} (a_k - m_{ij})^2$, where $m_{ij} = \sum_{k=i+1}^{j} a_k/(j - i)$. An optimal clustering of the scalars corresponds to a shortest path from node 0 to node n.

In the case, we have $N = \{0, 1, \dots, 6\}$ and $A = \{(0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 3), (1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 5), (3, 6), (4, 6)\}$. Notice that some arcs can be ruled out as it does not exist any paths from 0 to 6 traversing them, namely, $(0, 5), (1, 3), (1, 4), (1, 5), (1, 6), (2, 5), (3, 5)$. This observation allows to reduce the size of A from 15 to 8, so we focus on a reduced arc set $A = \{(0, 2), (0, 3), (0, 4), (0, 6), (2, 4), (2, 6), (3, 6), (4, 6)\}$. Let us compute the cost of each arc

- $(0, 2)$: $m_{02} = (0.5 + 0.8)/2 = 0.65$, $c_{02} = (0.5 - 0.65)^2 + (0.8 - 0.65)^2 = 0.045$
- $(0, 3)$: $m_{03} = (0.5 + 0.8 + 1.1)/3 = 0.8$, $c_{03} = (0.5 - 0.8)^2 + (0.8 - 0.8)^2 + (1.1 - 0.8)^2 = 0.18$
- $(0, 4)$: $m_{04} = (0.5 + 0.8 + 1.1 + 1.5)/4 = 0.975$, $c_{04} = (0.5 - 0.975)^2 + (0.8 - 0.975)^2 + (1.1 - 0.975)^2 + (1.5 - 0.975)^2 = 0.5475$
- $(0, 6)$: $m_{06} = (0.5 + 0.8 + 1.1 + 1.5 + 1.6 + 2)/6 = 1.25$, $c_{06} = 1.535$
- $(2, 4)$: $m_{24} = (1.1 + 1.5)/2 = 1.3$, $c_{24} = 0.08$
- $(2, 6)$: $m_{26} = (1.1 + 1.5 + 1.6 + 2.0)/4 = 1.55$, $c_{26} = 0.41$
- $(3, 6)$: $m_{36} = (1.5 + 1.6 + 2.0)/3 = 1.7$, $c_{36} = 0.14$
- $(4, 6)$: $m_{46} = (1.6 + 2.0)/2 = 1.8$, $c_{46} = 0.08$

The graph $G = (N, A)$ is



The Dijkstra's algorithm goes through the following iterations

				$d(\cdot)$					$pred(\cdot)$				
Iter	i	P	T	0	2	3	4	6	0	2	3	4	6
0		$\emptyset$	$\{0,2,3,4,6\}$	0	∞	∞	∞	∞	0				
1	0	$\{0\}$	$\{2,3,4,6\}$	0	0.045	0.18	0.5475	1.535	0	0	0	0	0
2	2	$\{0,2\}$	$\{3,4,6\}$	0	0.045	0.18	0.125	0.455	0	0	0	2	2
3	4	$\{0,2,4\}$	$\{3,6\}$	0	0.045	0.18	0.125	0.205	0	0	0	2	4
4	3	$\{0,2,3,4\}$	$\{6\}$	0	0.045	0.18	0.125	0.205	0	0	0	2	4
5	6	$\{0,2,3,4,6\}$	$\emptyset$	0	0.045	0.18	0.125	0.205	0	0	0	2	4

The shortest path is $0-2-4-6$, which corresponds to clustering the scalars in three sets: $\{1,2\}$, $\{3,4\}$, and $\{5,6\}$. The sum of the squared deviations is 0.205.

Exercise 4.4.

A construction company's work schedule on a certain site requires the following number of skilled personnel, called steel erectors, in the months of March through August

Month	Mar	Apr	May	Jun	Jul	Aug
Personnel	4	6	7	4	6	4

Personnel work at the site on the monthly basis. Suppose that three steel erectors are on the site in February and three steel erectors must be on site in September. The problem is to determine how many workers to have on site in each month in order to minimize costs, subject to the following conditions:

- *Transfer Costs*: adding a worker to this site costs €100 per worker and redeploying a worker to another site costs €160
- *Transfer Rules*: the company can transfer no more than three workers at the start of any month, and under a union agreement, it can redeploy no more than one of the current workers in any trade from a site at the end of any month
- *Surplus and Shortage*: the company incurs a cost of €200 per worker per month for having a surplus of steel erectors on site and a cost of €200 per worker per month for having a shortage of workers at the site.

Formulate this personnel planning problem as a SPP.

Let us introduce the following notation to refer to the input data:

K set of months (i.e., $K = \{2,3,4,5,6,7,8,9\}$, months from 2 – February until 9 – September)

w_k workers required at the end of month $k \in K$

W maximum number of workers required per month (i.e., $W = \max_{k \in K}\{w_k\} = 7$)

The problem can be formulated on a directed graph $G = (N, A)$, where each node of N is defined as k^r, where $k \in K$, $r = 3$ if $k = 2, 9$ and $r = 0, 1, \ldots, W$ if $k \in K \setminus \{2, 9\}$. A node $k^r \in N$ represents the decision of deploying r workers at the end of month k. The arc set contains an arc (i^r, j^s) for each pair of nodes $i^r, j^s \in N$ satisfying the following conditions: $j = i + 1$, $r - 1 \leq s \leq r + 3$. The cost of each arc (i^r, j^s) is defined as follows $c_{i^r j^s} = \max\{s - r, 0\} \cdot 100 + \max\{r - s, 0\} \cdot 160 + |s - w_j| \cdot 200$. The personnel planning problem calls for finding a shortest path from node 2^3 to node 9^3 in this graph. A shortest path can be found with the Dijkstra's algorithm.

Exercise 4.5.

Modify the Dijkstra's algorithm so that it identifies a shortest directed path from each node $j \in N \setminus \{t\}$ to a given node $t \in N$.

The required changes are:

- we initialize the distance label of node t to zero instead of node s;
- we replace the vector of predecessors $pred(\cdot)$ with $next(\cdot)$, where $next(i)$ represents the next node visited after i in the shortest paths;
- at each iteration, once node i having the minimum distance among the temporary nodes is selected, we examine the incoming arcs $(j, i) \in A$ and update label $d(j)$ as $d(j) = d(i) + c_{ji}$ if $d(j) > d(i) + c_{ji}$.

Algorithm 2: Modified Dijkstra's Algorithm

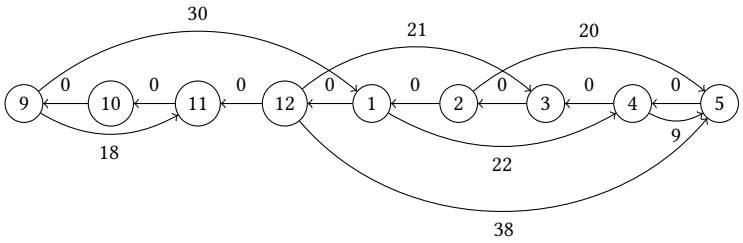
```
1  P ← ∅; T ← N;
2  d(i) ← ∞ for each i ∈ N;
3  d(t) ← 0 and next(t) ← 0;
4  while P ≠ N do
5      Let i ∈ T be a node for which d(i) = min{d(j) : j ∈ T};
6      P ← P ∪ {i}; T ← T \ {i};
7      for each (j, i) ∈ A do
8          if d(j) > d(i) + c_ji then
9              d(j) ← d(i) + c_ji;
10             next(j) ← i;
```

Exercise 4.6.

The following table illustrates some possible duties for the drivers of a bus company. We wish to ensure, at the lowest possible cost, that at least one driver is on duty for each hour of the planning period (9 a.m. to 5 p.m.). Formulate and solve this scheduling problem as a SPP.

Duty hours	9 - 1	9 - 11	12 - 3	12 - 5	2 - 5	1 - 4	4 - 5
Cost	30	18	21	38	20	22	9

We can construct a network $G = (N, A)$, where N contains a node for each hour from 9 a.m. to 5 p.m. (9 nodes in total). The arc set A consists of two sets of arcs $A = A^d \cup A^h$. A^d contains an arc between node s and node e for each duty that starts at time s and ends at time e. A^h contains an arc (h_{i+1}, h_i) between each pair of consecutive hours $i \in \{9, 10, \ldots, 3, 4\}$. The cost of each arc in A^d corresponds to the cost of the corresponding duty. All arcs in A^h has an associated zero cost.



A shortest path from the node corresponding to 9 a.m. to the node corresponding to 5 p.m. determines an optimal crew scheduling. The shortest path is 9 – 1 – 4 – 5; hence, the optimal set of duty hours is 9 – 1, 1 – 4, and 4 – 5, with a total cost of 61 units.

Chapter 5

Maximum Flows

5.1 Introduction

The Maximum Flow Problem (MFP) is easy to state: in a capacitated network, we wish to send as much flow as possible between two special nodes, a source node s and a sink node t, without exceeding the capacity of any arc.

5.1.1 Notation

We consider a capacitated graph $G = (N, A)$ with a non-negative capacity u_{ij} associated with each arc $(i, j) \in A$. Let $\overline{u} = \max\{u_{ij} \,|\, (i, j) \in A\}$. Moreover, let $\delta_i^+ \subseteq N$ ($\delta_i^- \subseteq N$, respectively) the set of successors (predecessors, resp.) of node $i \in N$, i.e., $\delta_i^+ = \{j \in N \,|\, (i, j) \in A\}$ ($\delta_i^- = \{j \in N \,|\, (j, i) \in A\}$, resp.). To define the MFP, we distinguish two special nodes in the graph G: a source node s and a sink node t. We wish to find the maximum flow from s to t that satisfies the arc capacities and mass balance constraints at all nodes. We can formally state the problem formally as

$$\max \quad v \tag{5.1a}$$

$$\text{s.t.} \sum_{j \in \delta_s^+} x_{sj} - \sum_{i \in \delta_s^-} x_{is} = v \tag{5.1b}$$

$$\sum_{j \in \delta_k^+} x_{kj} - \sum_{i \in \delta_k^-} x_{ik} = 0 \qquad \forall k \in N \setminus \{s, t\} \tag{5.1c}$$

$$\sum_{j \in \delta_t^+} x_{tj} - \sum_{i \in \delta_t^-} x_{it} = -v \tag{5.1d}$$

$$0 \le x_{ij} \le u_{ij} \qquad \forall (i, j) \in A \tag{5.1e}$$

We refer to a vector $\boldsymbol{x} \in \mathbb{R}_+^{|A|}$ satisfying constraints (5.1b)-(5.1e) as a *flow* and the corresponding value of v as the *value of the flow*.

5.1.2 Assumptions

We consider the MFP subject to the following assumptions:

Assumption 5.1. *The graph is directed.*

We can always fulfill this assumption by transforming a undirected graph into a directed graph.

Assumption 5.2. *All capacities are non-negative integers.*

The integrality assumption is not restrictive because all computers store capacities as rational numbers and we can always transform rational numbers to integer numbers by multiplying them by a suitably large number.

Assumption 5.3. *The graph does not contain a directed path from node s to t composed only of infinite capacity arcs.*

Whenever every arc on a directed path P from s to t has infinite capacity, the maximum flow value is unbounded.

Assumption 5.4. *Whenever an arc (i, j) belongs to A, arc (j, i) also belongs to A.*

This assumption is nonrestrictive because we allow arcs with zero capacity.

5.2 Applications

The MFP arises in a wide variety of situations. We describe a few such applications.

5.2.1 Feasible Flow Problem

The feasible flow problem requires that we identify a flow $\boldsymbol{x}$ in a graph $G = (N, A)$ satisfying the following constraints

$$\sum_{j \in \delta_k^+} x_{kj} - \sum_{i \in \delta_k^-} x_{ik} = b_k \quad \forall k \in N \tag{5.2a}$$

$$0 \le x_{ij} \le u_{ij} \qquad \forall (i, j) \in A \tag{5.2b}$$

where we assume that $\sum_{k \in N} b_k = 0$.

This problem arises, for example, in distribution. Suppose that merchandise is available at some seaports and is desired by other ports. We know the stock of merchandise available at the ports, the amount required at the other ports, and the maximum quantity of merchandise that can be shipped on a particular sea route. We wish to know if we can satisfy all of the demands with the supplies.

We can solve the feasible flow problem by solving a MFP defined on an augmented network as follows. We introduce two new nodes, a source node s and a sink node t. For each node i with $b_i > 0$, we add an arc (s, i) with capacity b_i, and for each node i with $b_i < 0$, we add an arc (i, t) with capacity $-b_i$. We solve a MFP from s to t in this augmented network. If the maximum flow saturates all the source and sink arcs, the feasible flow problem has a feasible solution; otherwise, it is infeasible.

5.2.2 Problem of Representatives

A town has r residents $R_1, R_2, \ldots, R_r$; q clubs $C_1, C_2, \ldots, C_q$; and p political parties $P_1, P_2, \ldots, P_p$. Each resident is a member of at least one club and belongs to exactly one political party. Each club must nominate one of its members to represent it on the town's governing council so that the number of

council members belonging to the political party P_k is at most u_k. Is it possible to find a council that satisfies this balancing property?

We illustrate this formulation with an example with $r = 7$, $q = 4$, and $p = 3$. We formulate it as a MFP (see Figure 5.1). The nodes $R_1, R_2, \ldots, R_7$ represent the residents, the nodes C_1, C_2, C_3, C_4 the clubs, and the nodes P_1, P_2, P_3 the political parties.

The graph also contains a source node s and a sink node t. It contains an arc (s, C_i) for each club C_i, an arc (C_i, R_j) whenever the resident R_j is a member of the club C_i, and an arc (R_j, P_k) if the resident R_j belongs to the political party P_k. Finally, we add an arc (P_k, t) for each party P_k of capacity u_k; all other arcs have unit capacity. We find a maximum flow in this graph. If the maximum flow value equals q, the town has a balanced council; otherwise, it does not.

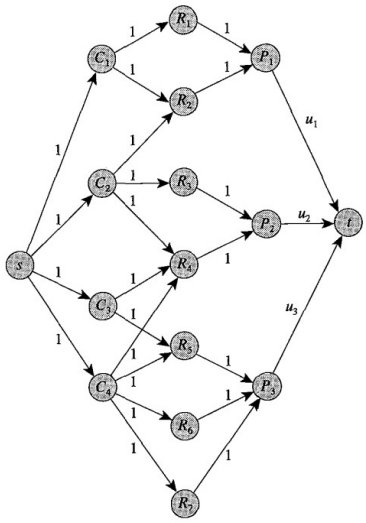


Figure 5.1: Problem of representatives

Exercise 5.1.

> Several families go out to dinner together. To increase their social interaction, they would like to sit at tables so that no two members of the same family are at the same table. Show how to formulate finding a seating arrangement that meets this objective as a MFP. Assume that the dinner contingent has p families and that the ith family has a_i members. Also assume that q tables are available and that the jth table has a seating capacity of b_j.

Construct the graph $G = (N_1 \cup N_2 \cup \{s, t\}, A)$, where N_1 contains p nodes (one per family) and N_2 contains q nodes (one per table). The source node s is connected to each node $i \in N_1$ by an arc (s, i) of capacity a_i. Each node $j \in N_2$ is connected to the sink node t by an arc (j, t) of capacity b_j. An arc (i, j) of unit capacity exists between every pair of nodes (i, j) such that $i \in N_1$ and $j \in N_2$. A feasible seating arrangement exists if and only if all the arcs emanating from node s are saturated in a maximum flow from s to t. Such a maximum flow defines a feasible seating arrangement in which a person from family i is to sit on table j if and only if arc (i, j) carries unit flow.

Exercise 5.2.

We are given a directed graph $G = (N, A)$. The set N consists of two sets V and W. Each node $i \in V$ has a supply $a_i \in \mathbb{Z}_+$, and each node $j \in W$ has a demand $b_j \in \mathbb{Z}_+$ – we assume that $\sum_{i \in V} a_i = \sum_{j \in W} b_j$. We can send a flow from each node $i \in V$ to each node $j \in W$, i.e., $A = \{(i, j) \mid i \in V, j \in W\}$. The unit cost to send a flow from i to j is c_{ij}. We wish to find a flow $\boldsymbol{x}$ from the nodes of the set V to the nodes of the set W such that $\max\{c_{ij}x_{ij} \mid (i, j) \in A\} \leq \lambda$, where λ is a given parameter. Show how to formulate this problem as a MFP

We solve a MFP on a directed graph $G' = (N', A')$ defined as follows. The node set N' contains the sets V and W plus two dummy nodes s and t, i.e., $N' = V \cup W \cup \{s, t\}$. The arc set is defined as $A' = A^s \cup A \cup A^t$, where $A^s = \{(s, i) \mid i \in V\}$ and $A^t = \{(j, t) \mid j \in W\}$. The arc capacities are defined as follows: (i) for each $(s, i) \in A^s$, $u_{si} = a_i$; (ii) for each $(i, j) \in A$, $u_{ij} = \lfloor \frac{\lambda}{c_{ij}} \rfloor$; (iii) for each $(j, t) \in A^t$, $u_{jt} = b_j$.

A feasible solution to the problem exists if and only if the maximum flow from s to t has a value $v = \sum_{i \in V} a_i$. Otherwise, no feasible solution such that $\max\{c_{ij}x_{ij} \mid (i, j) \in A\} \leq \lambda$ exists.

5.3 Flows and Cuts

In this section, we discuss some elementary properties of flows and cuts. We use these properties to prove the max-flow min-cut theorem to establish the correctness of the augmenting path algorithm.

Residual Graph Given a flow $\boldsymbol{x}$, the residual capacity r_{ij} of any arc $(i, j) \in A$ is the maximum additional flow that can be sent from i to j using the arcs (i, j) and (j, i). The residual capacity r_{ij} has two components: (i) $u_{ij} - x_{ij}$, the unused capacity of arc (i, j), and (ii) the current flow x_{ji} on arc (j, i), which we can cancel to increase the flow from i to j. Consequently, $r_{ij} = u_{ij} - x_{ij} + x_{ji}$. We refer to the graph $G(\boldsymbol{x})$ consisting of the arcs with positive residual capacities as the *residual graph* (with respect to the flow $\boldsymbol{x}$). Figure 5.2 gives an example of a residual graph

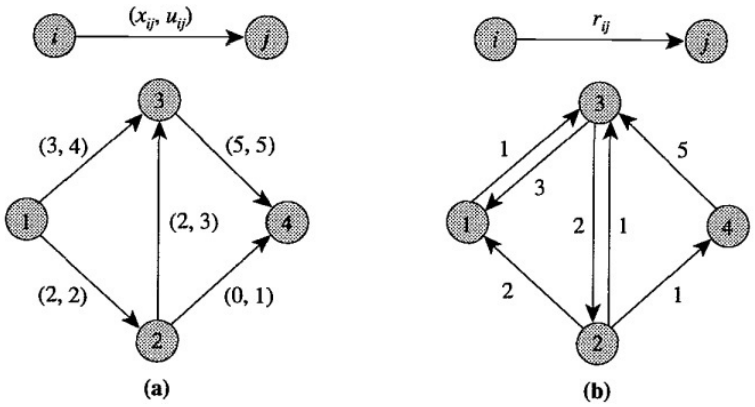


Figure 5.2: A residual graph: (a) original graph G with a flow $\boldsymbol{x}$; (b) residual graph $G(\boldsymbol{x})$

$s - t$ cut A cut is a partition of the node set N into two subsets S and $\overline{S} = N \setminus S$; we represent this cut with $(S, \overline{S})$. We refer to a cut as an $s - t$ cut if $s \in S$ and $t \in \overline{S}$. A forward arc (i, j) of the

cut is such that $i \in S$ and $j \in \overline{S}$, and a backward arc (i, j) of the cut is such that $i \in \overline{S}$ and $j \in S$. Let $A(S, \overline{S})$ denote the set of forward arcs in the cut, and let $A(\overline{S}, S)$ denote the set of backward arcs. For example, in Figure 5.3, the dashed arcs constitute an $s - t$ cut, where $S = \{1, 3, 5\}$, $\overline{S} = \{2, 4, 6\}$, $s = 1$, $t = 6$, $A(S, \overline{S}) = \{(1, 2), (3, 4), (5, 6)\}$, and $A(\overline{S}, S) = \{(2, 3), (4, 5)\}$

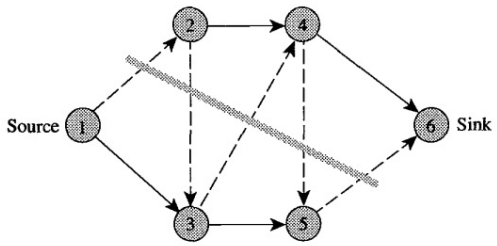


Figure 5.3: Example of an $s - t$ cut

Capacity of an $s - t$ cut The capacity $u(S, \overline{S})$ of an $s - t$ cut $(S, \overline{S})$ is the sum of the capacities of the forward arcs in the cut, i.e., $u(S, \overline{S}) = \sum_{(i,j) \in A(S,\overline{S})} u_{ij}$.

Minimum cut An $s - t$ cut whose capacity is minimum among all $s - t$ cuts is a *minimum cut*

Residual capacity of an $s - t$ cut The residual capacity $r(S, \overline{S})$ of an $s - t$ cut $(S, \overline{S})$ is the sum of the residual capacities of forward arcs in the cut

Flow across an $s - t$ cut Let $\boldsymbol{x}$ be a flow in the graph. Adding the mass balance constraint (5.1b)-(5.1d) for the nodes in S, we have

$$v = \sum_{i \in S} \left(\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} \right) = \sum_{(i,j) \in A(S,\overline{S})} x_{ij} - \sum_{(j,i) \in A(\overline{S},S)} x_{ji} \leq \sum_{(i,j) \in A(S,\overline{S})} u_{ij} = u(S, \overline{S}) \tag{5.3}$$

Thus, the value of any flow is less than or equal to the capacity of any $s - t$ cut: indeed, any flow from s to t must pass through every $s - t$ cut in the graph

Property 5.1. *The value of any flow is less than or equal to the capacity of any $s - t$ cut in the graph.*

If we discover a flow $\boldsymbol{x}$ whose value equals the capacity of some $s - t$ cut $(S, \overline{S})$, then $\boldsymbol{x}$ is a maximum flow and the cut $(S, \overline{S})$ is a minimum cut.

Suppose that $\boldsymbol{x}$ is a flow of value v and that $\boldsymbol{x}'$ is a flow of value $v + \Delta v$ for some $\Delta v \geq 0$. The inequality (5.3) implies that

$$v + \Delta v \leq \sum_{(i,j) \in A(S,\overline{S})} u_{ij} \quad \Longrightarrow \quad \Delta v \leq \sum_{(i,j) \in A(S,\overline{S})} (u_{ij} - x_{ij}) + \sum_{(j,i) \in A(\overline{S},S)} x_{ij} = \sum_{(i,j) \in A(S,\overline{S})} (u_{ij} - x_{ij} + x_{ji}) = \sum_{(i,j) \in A(S,\overline{S})} r_{ij}$$

Property 5.2. *For any flow $\boldsymbol{x}$ of value v, the additional flow that can be sent from s to t is less than or equal to the residual capacity of any $s - t$ cut.*

5.4 Generic Augmenting Path Algorithm

In this section, we describe one of the simplest algorithms, known as the *augmenting path algorithm*, for solving the MFP. We refer to a directed path from the source to the sink in the residual graph as an *augmenting path*. We define the *residual capacity* of an augmenting path as the minimum residual capacity of any arc in the path, which is always positive by definition. Whenever the graph contains an augmenting path, we can send additional flow from the source to the sink. The algorithm (see Algorithm 3) proceeds by identifying augmenting paths and augmenting flows on these paths until the graph contains no such path.

Algorithm 3: Augmenting Path Algorithm (Ford-Fulkerson Algorithm)

```
1 x ← 0;
2 while G(x) contains a directed path from s to t do
3     Identify an augmenting path P from s to t;
4     δ ← min{r_ij | (i, j) ∈ P};
5     Augment δ units of flow along P and update G(x);
```

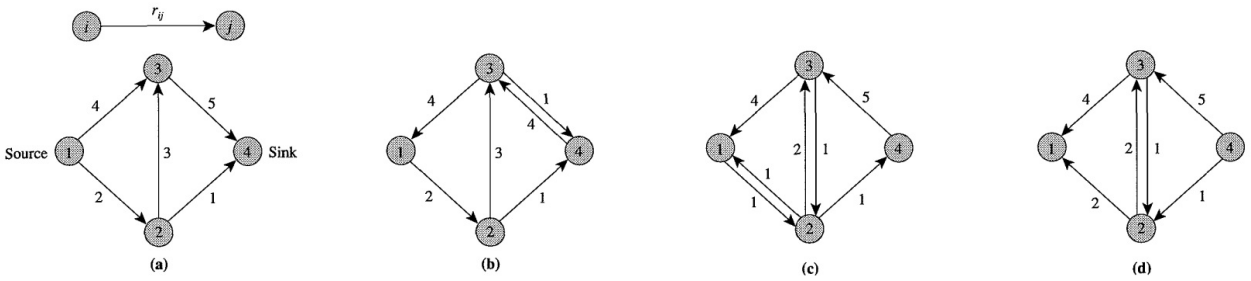


Figure 5.4: Illustrating the generic augmenting path algorithm: (a) residual graph for the zero flow; (b) graph after augmenting four units along 1 – 3 – 4; (c) graph after augmenting one unit along 1 – 2 – 3 – 4; (d) graph after augmenting one unit along 1 – 2 – 4

5.4.1 Relationship between the Original and Residual Graphs

Suppose that we find an augmenting path P in the residual graph and send δ units of flow through P. What is the effect on the arc flows x_{ij}? The definition of the residual capacity (i.e., $r_{ij} = u_{ij} - x_{ij} + x_{ji}$) implies that an additional flow of δ units on arc (i, j) in the residual graph corresponds to (i) an increase in x_{ij} by δ units in the original graph, or (ii) a decrease in x_{ji} by δ units in the original graph.

Consider Figure 5.5(a) and the corresponding residual graph of Figure 5.5(b). Augmenting 1 unit of flow on the path 1 – 2 – 4 – 3 – 5 – 6 produces the residual graph in Figure 5.5(c) with the corresponding arc flows shown in Figure 5.5(d). Comparing the solution in Figure 5.5(d) with that in Figure 5.5(a), we find that the flow augmentation increases the flow on arcs $(1, 2)$, $(2, 4)$, $(3, 5)$, $(5, 6)$, and decreases the flow on arc $(3, 4)$.

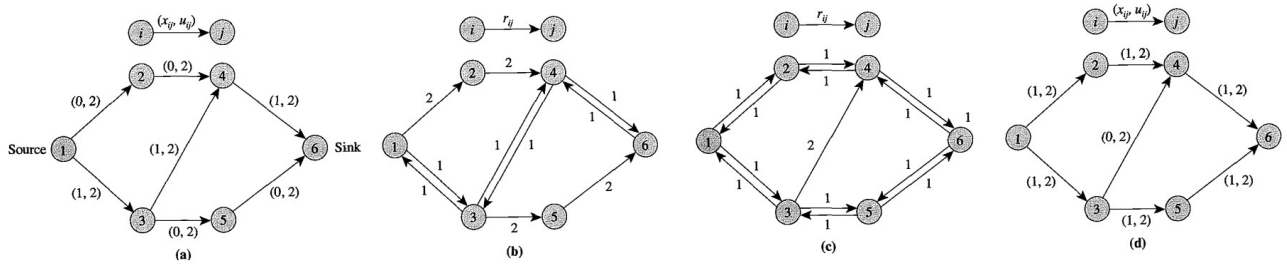


Figure 5.5: (a) original graph with a flow $\boldsymbol{x}$; (b) residual graph for flow $\boldsymbol{x}$; (c) residual graph after augmenting one unit along 1 – 2 – 4 – 3 – 5 – 6; (d) flow in the original graph after the augmentation

5.4.2 Effect of Augmentation on Flow Decomposition

Let us illustrate the effect of an augmentation on the flows of the example of Figure 5.5. Figure 5.6(a) gives the decomposition of the initial flow, and Figure 5.6(b) gives the decomposition of the flow after augmenting one unit of flow on the path 1 – 2 – 4 – 3 – 5 – 6. Although we augmented one unit of flow along the path 1 – 2 – 4 – 3 – 5 – 6, the flow decomposition contains no such path.

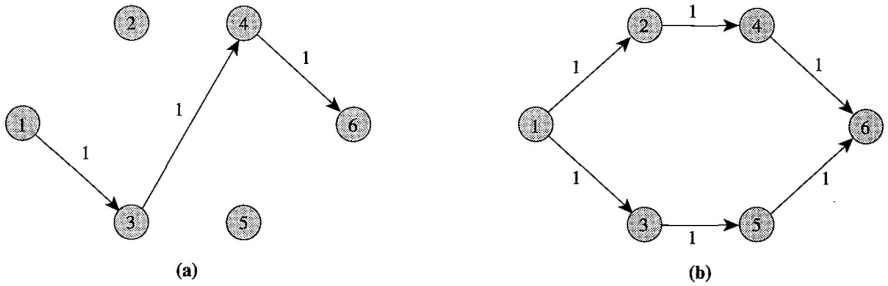


Figure 5.6: Flow decomposition of the solution in (a) Figure 5.5(a) and (b) Figure 5.5(d)

The path 1 – 3 – 4 – 6 defining the flow in Figure 5.5(a) contains three segments: the path up to node 3, arc (3, 4) as a forward arc, and the path up to node 6. We can view this path as an augmentation on the zero flow. Similarly, the path 1 – 2 – 4 – 3 – 5 – 6 contains three segments: the path up to node 4, arc (3, 4) as a backward arc, and the path up to node 6. We can view the augmentation on the path 1 – 2 – 4 – 3 – 5 – 6 as linking the initial segment of the path 1 – 3 – 4 – 6 with the last segment of the augmentation path, linking the last segment of the path 1 – 3 – 4 – 6 with the initial segment of the augmentation path, and canceling the flow on arc (3, 4), which then drops from both the path 1 – 3 – 4 – 6 and the augmentation path. In general, we can view each augmentation as "pasting together" segments of the current flow decomposition to obtain a new flow decomposition.

5.5 Labeling Algorithm and the Max-Flow Min-Cut Theorem

Now, we discuss the augmenting path algorithm in detail. So far, we did not discuss some important details, such as (i) how to identify an augmenting path or show that the graph contains no such path, and (ii) whether the algorithm terminates in finite number of iterations, and when it terminates,

whether it has obtained a maximum flow. We consider these issues for a specific implementation of the generic augmenting path algorithm known as the *labeling algorithm*.

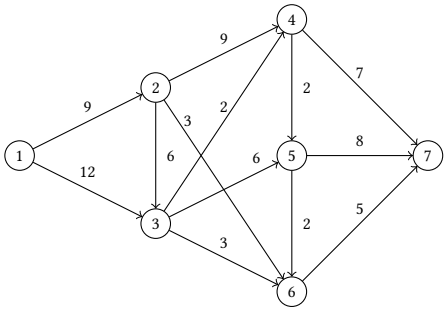
The labeling algorithm uses a search technique to identify a directed path in $G(\boldsymbol{x})$ from s to t. The algorithm fans out from s to find all nodes reachable from s along a directed path in the residual graph. At any step, the algorithm maintains two sets of nodes: the *labeled* nodes (L) and the *temporary* nodes (T). Labeled nodes are those nodes reached in the fanning out process, so the algorithm has determined a directed path from s to these nodes in the residual graph. The temporary nodes are those nodes that are examined in the fanning-out process. The algorithm iteratively selects a temporary node and scans its arc adjacency list (in the residual graph) to reach and label additional nodes. Eventually, t becomes labeled, and the algorithm sends the maximum possible flow on the path from s to t. It then erases the labels and repeats this process. The algorithm terminates when it has scanned all the labeled nodes and t remains unlabeled, implying that s is not connected to t in the residual graph. Algorithm 4 gives a step-by-step description of the labeling algorithm.

Algorithm 4: Labeling Algorithm (Ford-Fulkerson Algorithm)

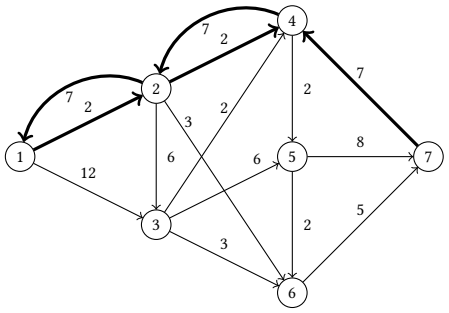
```
1  Set x ← 0, L ← {t}, and create the residual graph;
2  while t ∈ L do
3      Set L ← {s}, T ← {s}, pred(j) ← 0 for each j ∈ N;
       // Identify an augmenting path
4      while T ≠ ∅ and t ∉ L do
5          Remove a node i from T;
6          for each arc (i, j) emanating from node i in the residual graph do
7              if r_ij > 0 and node j ∉ L then
8                  Set pred(j) ← i, L ← L ∪ {j}, T ← T ∪ {j};
       // Augmenting phase
9      if t ∈ L then
10         Obtain an augmenting path P from s to t by using the predecessor labels;
11         Compute δ = min{r_ij : (i, j) ∈ P};
12         Augment δ units of flow along P, and update the residual capacities;
```

To better understand the labeling algorithm, consider the following graph, where the label close to each arc is the corresponding capacity, u_{ij}. We search for the maximum flow we can send from source 1 to sink 7. We describe the steps the algorithm goes through in detail.

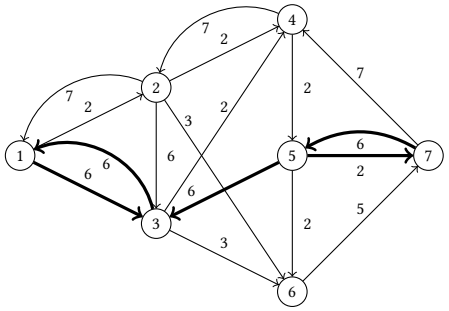
1. As the initial flow is set to 0 in every arc, this graph also corresponds to the residual graph.



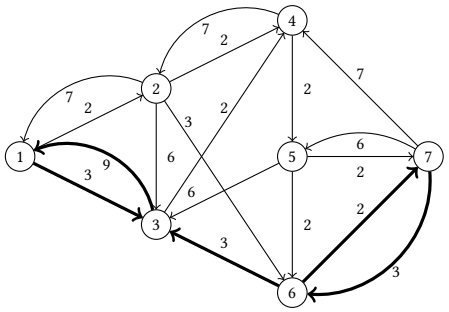
2. Let us assume that, in the first iteration, the algorithm identifies the augmenting path 1 – 2 – 4 – 7 with an augmenting flow of 7. The corresponding residual graph is



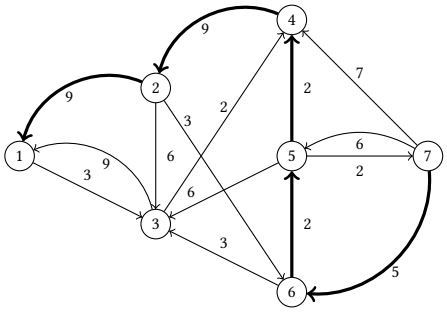
3. We now assume that, in the second iteration, the algorithm identifies the augmenting path 1 – 3 – 5 – 7 with an augmenting flow of 6. The corresponding residual graph is



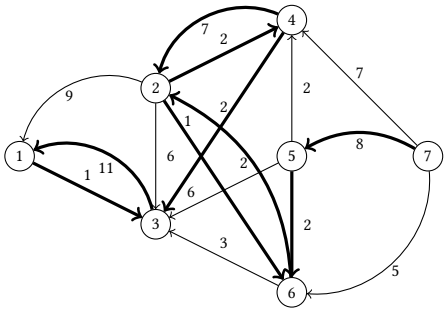
4. In the next iteration, the algorithm can find the augmenting path 1 – 3 – 6 – 7 with an augmenting flow of 3. The corresponding residual graph is



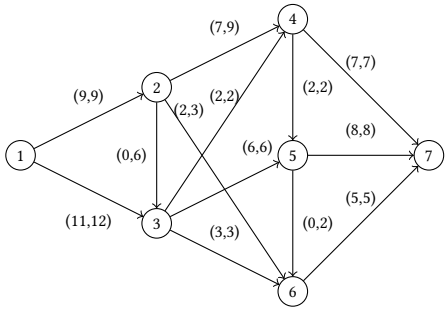
5. In the next iteration, the algorithm can find the augmenting path 1 – 2 – 4 – 5 – 6 – 7 with an augmenting flow of 2. The corresponding residual graph is



6. Then, the algorithm can find the augmenting path 1 – 3 – 4 – 2 – 6 – 5 – 7 with an augmenting flow of 2. The corresponding residual graph is



This is the final residual graph. The algorithm finds out that the maximum flow from node 1 to node 7 is 20. In particular: 7 units go through path 1 – 2 – 4 – 7, 2 units go through path 1 – 2 – 6 – 7, 3 units go through path 1 – 3 – 6 – 7, 6 units go through path 1 – 3 – 5 – 7, and 2 units go through path 1 – 3 – 4 – 5 – 7. This residual graph corresponds to the following solution, where the label close to each arc indicates the flow and the capacity, i.e., (x_{ij}, u_{ij}).



Note that in each iteration of the labeling algorithm, the algorithm either performs an augmentation or terminates because it cannot label the sink. In the latter case, we must show that the current flow $\boldsymbol{x}$ is a maximum flow. Suppose, at this stage, that S is the set of labeled nodes (i.e., $S = L$) and $\overline{S} = N \setminus S$ is the set of unlabeled nodes. Clearly, $s \in S$ and $t \in \overline{S}$. Since the algorithm cannot label any node in $\overline{S}$ from any node in S, $r_{ij} = 0$ for each $(i, j) \in A(S, \overline{S})$. Furthermore, since $r_{ij} = (u_{ij} - x_{ij}) + x_{ji}$, $x_{ij} \le u_{ij}$ and $x_{ji} \ge 0$, the condition $r_{ij} = 0$ implies that $x_{ij} = u_{ij}$ for every arc $(i, j) \in A(S, \overline{S})$ and $x_{ji} = 0$ for every arc $(j, i) \in (\overline{S}, S)$. Substituting these flow values in (5.3), we find that

$$v = \sum_{(i,j)\in A(S,\overline{S})} x_{ij} - \sum_{(j,i)\in A(\overline{S},S)} x_{ji} = \sum_{(i,j)\in A(S,\overline{S})} u_{ij} = u(S, \overline{S})$$

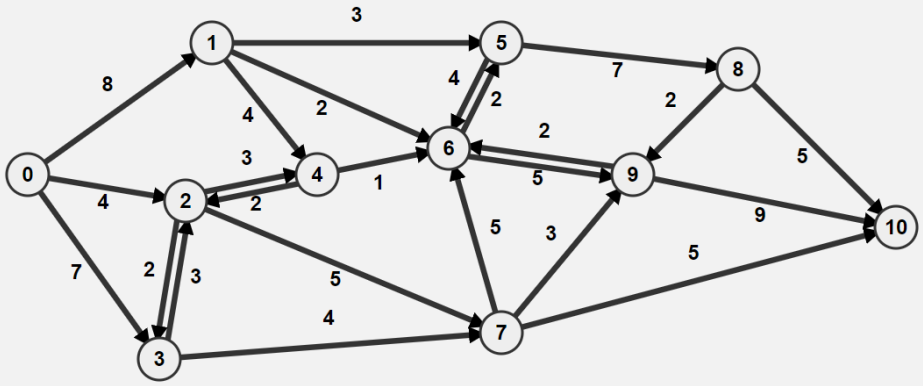
which shows that the value of the current flow $\boldsymbol{x}$ equals the capacity of the $s-t$ cut $(S, \overline{S})$. Therefore, $\boldsymbol{x}$ is a maximum flow, and $(S, \overline{S})$ is a minimum $s-t$ cut.

Theorem 5.1. ***(Max-Flow Min-Cut Theorem)*** *The maximum value of the flow from a source node s to a sink node t in a capacitated graph equals the minimum capacity among all $s-t$ cuts.*

Theorem 5.2. ***(Augmenting Path Theorem)*** *A flow $\boldsymbol{x}^*$ is a maximum flow if and only if the residual graph $G(\boldsymbol{x}^*)$ contains no augmenting path.*

Exercise 5.3.

By applying the labeling algorithm, determine a maximum flow from node $s = 0$ to node $t = 10$ of the following graph. What is the maximum flow? What is the corresponding minimum cut?



The maximum flow has a value of $v = 15$ and corresponds to the $s-t$ cut $(S, \overline{S})$, $S = \{0, 1, 2, 3, 4\}$, $\overline{S} = \{5, 6, 7, 8, 9, 10\}$.

Exercise 5.4.

In some graphs $G = (N, A)$, in addition to arc capacities, each node $i \in N$ might have an upper bound (say w_i) on the flow that can pass through it. In these graphs, we are interested in determining the maximum flow satisfying both the arc and node capacities. Transform this problem to the standard MFP.

Let $G = (N, A)$ be a graph having node capacities. Construct the graph $G' = (N', A')$ in which every node $i \in N$ is replaced by two nodes $i', i'' \in N$. Furthermore, the arc set A' is defined as $A' = \{(i'', j') \,|\, (i, j) \in A\} \cup \{(i', i'') \,|\, i \in N\}$. Each arc $(i', i'') \in A$ has a capacity of w_i. A maximum flow in G' is equal to a maximum flow in G.

Chapter 6

Minimum Spanning Trees

6.1 Introduction

In this chapter, we consider the Minimum Spanning Tree Problem (MSTP). Recall that a spanning tree T of a graph G is a connected acyclic subgraph that spans all the nodes. Every spanning tree of G has $n - 1$ edges. Given an undirected graph $G = (N, E)$ with $|N| = n$ nodes and $|E| = m$ edges with a *length* or *cost* c_{ij} associated with each edge $\{i, j\} \in E$, we wish to find a spanning tree, called a MST, that has the smallest total cost (or length) of its edges, measured as the sum of the costs of the edges in the spanning tree.

6.2 Applications

The MSTP arises in one of two ways, directly or indirectly. In some *direct* applications, we wish to connect a set of points using the least-cost connection of edges. The points represent physical entities such as components of a computer chip or users of a system who must be connected to each other or to a central service such as a central processor in a computer system. In *indirect* applications, we may wish to connect some set of points using a measure of performance.

6.2.1 Designing Physical Systems

The design of physical systems is a complex task involving an interplay between performance objectives (such as throughput and reliability), design costs, operating economics, and available technology. In many settings, the major criterion is simple: we need to design a network that connects geographically dispersed system components or just provides the infrastructure needed for users to communicate with each other. In many of these settings, the system does not need any redundancy, so we are interested in the simplest possible connection, i.e., a spanning tree. This type of application arises in the construction (or installation) of numerous physical systems: highways, computer networks, telephone networks, railroads, cable television lines, and electrical power transmission lines. For example, this type of MSTP arises in the following problem settings:

- Connect terminals in cabling the panels of electrical equipment. How should we wire the terminals to use the least possible length of the wire?

- Constructing a pipeline network to connect a number of towns using the smallest possible total length of pipeline.
- Linking isolated villages in a remote region connected by roads but not yet by telephone service. We wish to determine along which stretches of roads we should place telephone lines, using the minimum possible total miles of the lines, to link every pair of villages.
- Constructing a digital computer system, composed of high-frequency circuitry, when it is important to minimize the length of wires between different components to reduce both capacitance and delay line effects. Since all components must be connected, we face a MSTP.
- Connecting a number of computer sites by high-speed lines. Each line is available for leasing at a certain monthly cost, and we wish to determine a configuration that connects all the sites at the least possible cost.

6.2.2 All-Pairs Minimax Path Problem

In a graph $G = (N, E)$ with edge costs c_{ij}, $\{i, j\} \in E$, we define the value of a path P from node k to node l as the maximum cost edge in P. The all-pairs minimax path problem requires that we determine, for every pair $[k, l]$ of nodes, a minimum value path from k to l. We show how to solve the all-pairs minimax path problem on an undirected graph by solving a single MSTP.

The minimax path problem arises in a variety of situations. As an example, consider a spacecraft that is about to enter the earth's atmosphere. The craft passes through different pressure and temperature zones that we can represent by edges of a graph. It needs to fly along a trajectory that will bring the craft to the surface of the earth while keeping the maximum temperature to which the surface of the craft is exposed as low as possible.

To transform the all-pairs minimax path problem into a MSTP, let T^* be a minimum spanning tree of G. Let P denote the unique path in T^* between a node pair $[p, q]$, and let $\{i, j\}$ denote the maximum cost edge in P. Observe that the value of the path P is c_{ij}. By deleting edge $\{i, j\}$ from T^*, we partition the node set N into two subsets and therefore define an $i - j$ cut $(S, \overline{S})$ with $i \in S$ and $j \in \overline{S}$ (see Figure 6.1). We later show that this cut satisfies the following property:

$$c_{ij} \leq c_{kl} \quad \forall \{k, l\} \in E \ : \ k \in S, l \in \overline{S} \tag{6.1}$$

for otherwise by replacing the edge $\{i, j\}$ by an edge $\{k, l\}$, we can obtain a spanning tree of smaller cost. Now, consider any path P' from node p to node q. This path must contain at least one edge $\{k, l\}$ crossing $p - q$ cut $(S, \overline{S})$. Condition (6.1) implies that the value of the path P' will be at least c_{ij}. Since c_{ij} is the value of the path P, P must be a minimum value path from node p to node q. This observation establishes the fact that the unique path between any pair of nodes in T^* is the minimum value path between that pair of nodes.

6.2.3 Reducing Data Storage

In several different application contexts, we wish to store data specified in the form of a two-dimensional array more efficiently than storing all the elements of the array (to save memory space). We assume that the rows of the array have many similar entries and differ only at a few places.

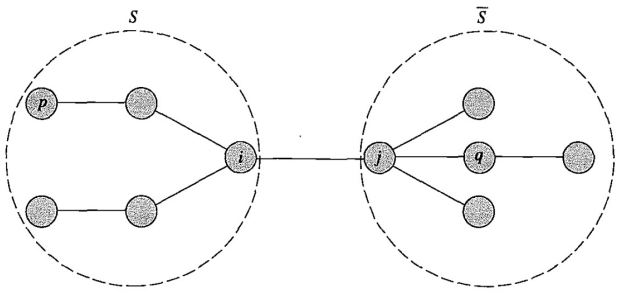


Figure 6.1: Cut $(S, \overline{S})$ formed by deleting the edge $\{i, j\}$ from a spanning tree

Since the entities in the rows are similar, one approach for saving memory is to store one row, called the reference row, completely, and to store only the differences between some of the rows so that we can derive each row from these differences and the reference row. Let c_{ij} denote the number of different entries in rows i and j; that is, if we are given row i, then by making c_{ij} changes to the entries in this row we can obtain row j, and vice versa. Suppose that the array contains four rows, represented by R_1, R_2, R_3, and R_4, and we decide to treat R_1 as a reference row. Then one plausible solution is to store the differences between R_1 and R_2, R_2 and R_4, and R_1 and R_3. Clearly, from this solution, we can obtain rows R_2 and R_3 by making c_{12} and c_{13} changes to the elements in row R_1. Having obtained row R_2, we can make c_{24} changes to the elements of this row to obtain R_4.

It is sufficient to store differences between those rows that correspond to edges of a spanning tree. These differences permit us to obtain each row from the reference row. The total storage requirement for a particular storage scheme is the length of the reference row plus the sum of the differences between the rows. Therefore, a MST provides a least-cost storage scheme.

6.2.4 Cluster Analysis

The essential issue in cluster analysis is to partition a set of data into *natural groups*; the data points within a particular group of data, or a *cluster*, should be more closely related to each other than the data points not in that cluster.

Cluster analysis is important in a variety of disciplines that rely on empirical investigations. Consider, for example, an instance of a cluster analysis arising in medicine. Suppose that we have data on a set of 350 patients, measured with respect to 18 symptoms. Suppose, further, that a doctor has diagnosed all of these patients as having the same disease, which is not well understood. The doctor would like to know if he can develop a better understanding of this disease by categorizing the symptoms into smaller groupings that can be detected through cluster analysis. Doing so might permit the doctor to find more natural disease categories to replace or subdivide the original disease.

We can solve a class of problems arising in cluster analysis by solving a MSTP. Suppose that we are interested in finding a partition of a set of n points in two-dimensional Euclidean space into clusters. A popular method for solving this problem is finding a MST and obtain k partitions by starting with a MST and delete k arcs from the MST one by one in non-increasing order of their lengths.

We illustrate the latter approach using an example. Consider a set of 27 points shown in Figure 6.2(a). Suppose that the network in Figure 6.2(b) is a MST for these points. Deleting the three largest length arcs from the MST gives a partition with four clusters shown in Figure 6.2(c).

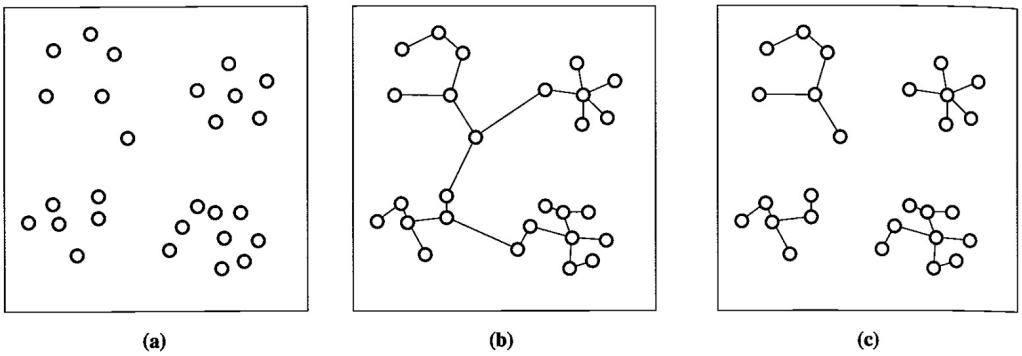


Figure 6.2: Identifying clusters by finding a minimum spanning tree

6.3 Optimality Conditions

For the MSTP, we can formulate the optimality conditions in two equivalent ways: *cut optimality conditions* and *path optimality conditions*.

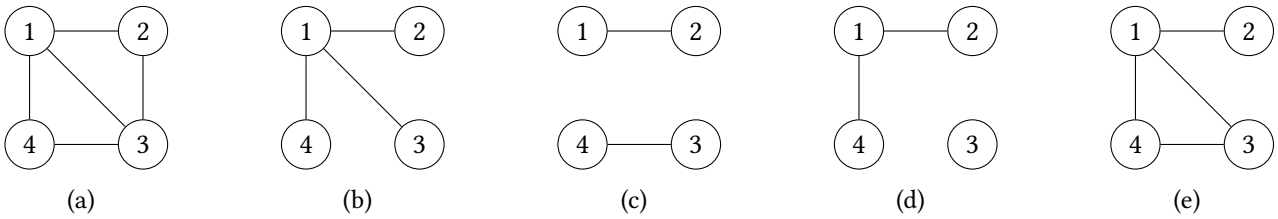


Figure 6.3: A graph (a) and a corresponding spanning tree (b), along with three non-spanning trees (a disconnected one (c), one that doesn't span all nodes (d), and a cyclic subgraph (e))

Before considering these optimality conditions, let us establish some further notation and illustrate some basic concepts. Figure 6.3(a) shows a graph with four nodes. The subgraph 6.3(b) is a spanning tree. The other three subgraphs 6.3(c), 6.3(d), 6.3(e) are non-spanning tree because (c) is not connected, (d) does not span all nodes, and (e) contains a cycle. We refer to those edges contained in a given spanning tree as *tree edges* (i.e., edges $\{1,2\}$, $\{1,3\}$, and $\{1,4\}$ in Figure 6.3(b)) and to those edges not contained in a given spanning tree as *non-tree edges* (i.e., edges $\{2,3\}$ and $\{3,4\}$ in Figure 6.3(b)).

The following two elementary observations will arise frequently in our development in this chapter.

1. For every non-tree edge $\{k,l\}$, the spanning tree T contains a unique path from node k to node l. The edge $\{k,l\}$ together with this unique path defines a cycle (see Figure 6.4(a)).

2. If we delete any tree edge $\{i,j\}$ from a spanning tree, the resulting graph partitions the node set N into two subsets (see Figure 6.4(b)). The edges from the underlying graph G whose two endpoints belong to the different subsets constitute an $i-j$ cut.

We next prove the two optimality conditions.

Theorem 6.1. ***(Cut Optimality Conditions)*** *A spanning tree T^* is a MST if and only if it satisfies the following cut optimality conditions: for every tree edge $\{i,j\} \in T^*$, $c_{ij} \le c_{kl}$ for every edge $\{k,l\}$ contained in the $i-j$ cut formed by deleting edge $\{i,j\}$ from T^*.*

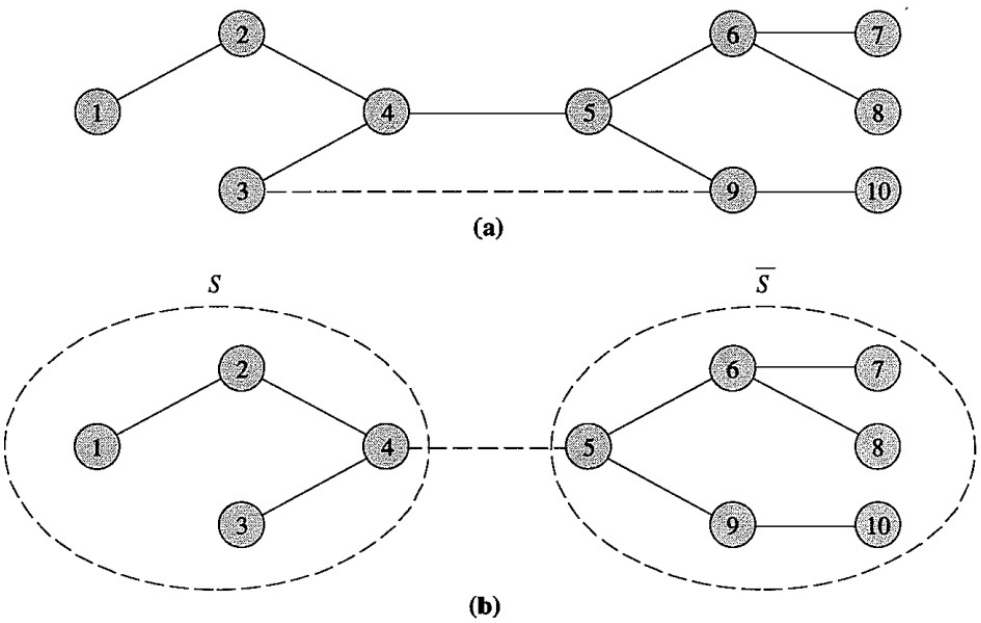


Figure 6.4: Illustrating properties of a spanning tree: (a) adding edge $\{3, 9\}$ to the spanning tree forms the unique cycle $3 - 4 - 5 - 9 - 3$; (b) deleting edge $\{4, 5\}$ forms the $4 - 5$ cut $(S, \overline{S})$ with $S = \{1, 2, 3, 4\}$

Proof. It is easy to see that every MST T^* must satisfy the cut optimality conditions. For, if $c_{ij} > c_{kl}$ and edge $\{k, l\}$ is contained in the $i - j$ cut formed by deleting edge $\{i, j\}$ from T^*, then introducing edge $\{k, l\}$ into T^* in place of edge $\{i, j\}$ would create a spanning tree with a cost less than T^*, contradicting the optimality of T^*.

We next show that if any tree T^* satisfies the cut optimality conditions, it must be optimal. Suppose that T' is a MST and $T' \neq T^*$. Then, T^* contains an edge $\{i, j\}$ that is not in T'. Deleting edge $\{i, j\}$ from T^* creates a cut, say $(S, \overline{S})$. If we add the edge $\{i, j\}$ to T', we create a cycle W that must contain an edge $\{k, l\}$ (other than edge $\{i, j\}$) with $k \in S$ and $l \in \overline{S}$. Since T^* satisfies the cut optimality conditions, then $c_{ij} \leq c_{kl}$. Moreover, since T' is a MST, $c_{ij} \geq c_{kl}$, for otherwise we could improve on its cost by replacing edge $\{k, l\}$ by edge $\{i, j\}$. Therefore, $c_{ij} = c_{kl}$. If we introduce edge $\{k, l\}$ in the tree T^* in place of edge $\{i, j\}$, we produce another MST and it has one more edge in common with T'. Repeating this argument several times, we can transform T^* into the MST T'. This construction shows that T^* is also a MST and completes the proof. ∎

The cut optimality conditions imply that every edge in a MST is a minimum cost edge across the cut that is defined by removing it from the tree.

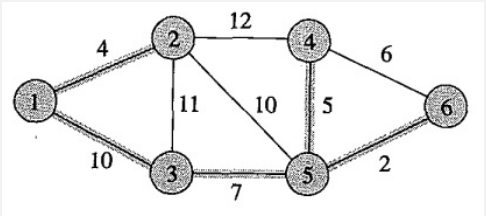
Theorem 6.2. ***(Path Optimality Conditions)*** *A spanning tree T^* is a MST if and only if it satisfies the following path optimality conditions: for every non-tree edge $\{k, l\}$ of G, $c_{ij} \leq c_{kl}$ for every edge $\{i, j\}$ contained in the path in T^* connecting nodes k and l.*

Proof. It is easy to show the necessity of the path optimality conditions. Suppose T^* is a MST satisfying these conditions and edge $\{i, j\}$ is contained in the path in T^* connecting nodes k and l. If $c_{ij} > c_{kl}$, introducing edge $\{k, l\}$ into T^* in place of edge $\{i, j\}$ would create a spanning tree with a cost less than T^*, contradicting the optimality of T^*.

We establish the sufficiency of the path optimality conditions by using the sufficiency of the cut optimality conditions. Let $\{i,j\}$ be any tree edge in T^*, and let S and $\overline{S}$ be the two sets of connected nodes produced by deleting edge $\{i,j\}$ from T^*. Suppose $i \in S$ and $j \in \overline{S}$. Consider any edge $\{k,l\}$ with $k \in S$ and $l \in \overline{S}$. Since T^* contains a unique path joining nodes k and l and edge $\{i,j\}$ is the only edge in T^* joining a node in S and a node in $\overline{S}$, edge $\{i,j\}$ must belong to this path. The path optimality condition implies that $c_{ij} \leq c_{kl}$. Since this condition must be valid for every non-tree edge $\{k,l\}$ in the cut $(S,\overline{S})$ formed by deleting any tree edge $\{i,j\}$, T^* satisfies the cut optimality conditions and must be a MST. ∎

Exercise 6.1.

In the following graph, the bold lines represent a MST



1. By listing each tree edge $\{i,j\}$ and the minimum length edge in the cut defined by the edge $\{i,j\}$, verify that the tree satisfies the cut optimality conditions.
2. By listing each non-tree edge $\{k,l\}$ and the maximum length edge on the tree path from node k to node l, verify that this tree satisfies the path optimality conditions.

1. For each tree edge $\{i,j\}$, we list the edges of the $i-j$ cut $(S,\overline{S})$ (i.e., $E(S,\overline{S})$) and check that edge $\{i,j\}$ has the minimum cost among the edge in the cutset
   - $\{i,j\} = \{1,2\}$, $E(S,\overline{S}) = \{\{1,2\},\{2,3\},\{2,4\},\{2,5\}\}$, $c_{12} = 4$, $c_{23} = 11$, $c_{24} = 12$, $c_{25} = 10$
   - $\{i,j\} = \{1,3\}$, $E(S,\overline{S}) = \{\{1,3\},\{2,3\},\{2,4\},\{2,5\}\}$, $c_{13} = 10$, $c_{23} = 11$, $c_{24} = 12$, $c_{25} = 10$
   - $\{i,j\} = \{3,5\}$, $E(S,\overline{S}) = \{\{2,4\},\{2,5\},\{3,5\}\}$, $c_{24} = 12$, $c_{25} = 10$, $c_{35} = 7$
   - $\{i,j\} = \{4,5\}$, $E(S,\overline{S}) = \{\{2,4\},\{4,5\},\{4,6\}\}$, $c_{24} = 12$, $c_{45} = 5$, $c_{56} = 6$
   - $\{i,j\} = \{5,6\}$, $E(S,\overline{S}) = \{\{4,6\},\{5,6\}\}$, $c_{46} = 6$, $c_{56} = 2$
2. For each non-tree edge $\{k,l\}$, we list the path $P(k,l)$ connecting k and l and the cost of the maximum-cost edge in the path
   - $\{k,l\} = \{2,3\}$, $P(k,l) = \{\{1,2\},\{1,3\}\}$, $c_{13} = 10 \leq c_{23} = 11$
   - $\{k,l\} = \{2,4\}$, $P(k,l) = \{\{1,2\},\{1,3\},\{3,5\},\{4,5\}\}$, $c_{13} = 10 \leq c_{24} = 12$
   - $\{k,l\} = \{2,5\}$, $P(k,l) = \{\{1,2\},\{1,3\},\{3,5\}\}$, $c_{13} = 10 \leq c_{25} = 10$
   - $\{k,l\} = \{4,6\}$, $P(k,l) = \{\{4,5\},\{4,6\}\}$, $c_{45} = 5 \leq c_{46} = 6$

6.4 Kruskal's Algorithm

The Kruskal's algorithm is an efficient algorithm that builds upon the path optimality conditions. It first sorts all the edges in non-decreasing order of their costs and defines a set, L, that is the set of edges chosen to be part of a MST. Initially, the set L is empty. It then examines the edges in the sorted order one-by-one and checks whether adding each edge to L creates a cycle with the edges already in L. If it does not, the edge is added to L; otherwise, it is discarded. It terminates when L contains $n - 1$ edges. At termination, the edges in L constitute a MST T^*.

The correctness of Kruskal's algorithm follows from the fact that it discards each non-tree edge $\{k, l\}$ with respect to T^* at some stage because it would create a cycle with the edges already in L. However, observe that the cost of edge $\{k, l\}$ is greater than or equal to the cost of every edge in that cycle because the edges are examined in non-decreasing order of their costs. Therefore, the spanning tree T^* satisfies the path optimality conditions, so it is a MST.

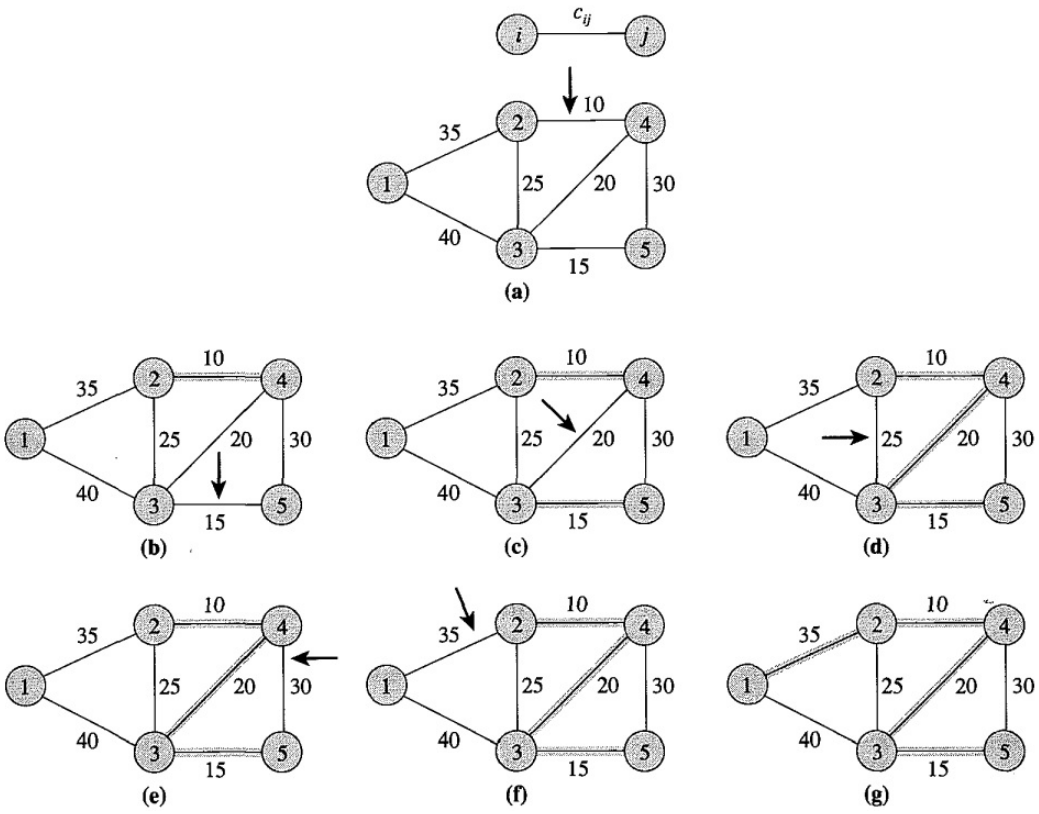
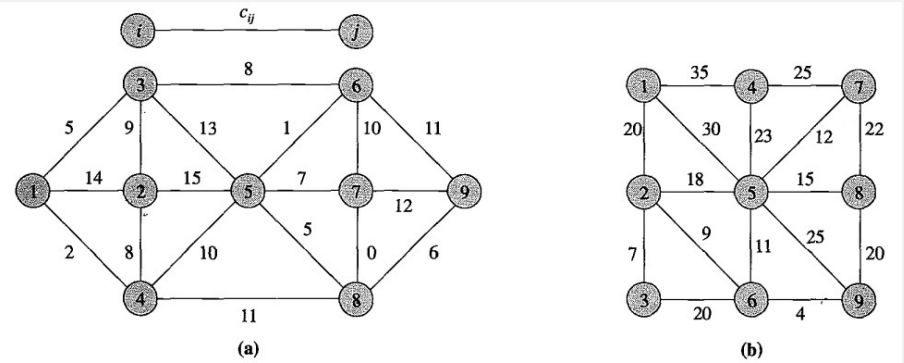


Figure 6.5: Illustrating Kruskal's algorithm

To illustrate Kruskal's algorithm on a numerical example, we consider the network shown in Figure 6.5(a). Sorted in the order of their costs, the edges are $\{2, 4\}$, $\{3, 5\}$, $\{3, 4\}$, $\{2, 3\}$, $\{4, 5\}$, $\{1, 2\}$, and $\{1, 3\}$. In the first three iterations, the algorithm adds the edges $\{2, 4\}$, $\{3, 5\}$, and $\{3, 4\}$ to L (see Figures 6.5(a)-(c)). In the next two iterations, the algorithm examines edges $\{2, 3\}$ and $\{4, 5\}$ and discards them because the addition of each edge to L creates a cycle (see Figure 6.5(d)-(e)). Then the algorithm adds edge $\{1, 2\}$ to L and terminates. Figure 6.5(g) shows the MST.

Exercise 6.2.

Using Kruskal's algorithm, find a MST of the following graphs



When Kruskal's algorithm is applied to the first graph, then the order in which the edges are included in the spanning tree is $\{7,8\}$, $\{5,6\}$, $\{1,4\}$, $\{1,3\}$, $\{5,8\}$, $\{8,9\}$, $\{3,6\}$, and $\{2,4\}$.

In the case of the second graph, the order in which edges are included in the spanning tree is $\{6,9\}$, $\{2,3\}$, $\{2,6\}$, $\{5,6\}$, $\{5,7\}$, $\{5,8\}$, $\{1,2\}$, and $\{4,5\}$.

6.5 Prim's Algorithm

Just as the path optimality conditions allowed us to develop Kruskal's algorithm, the cut optimality conditions permit us to develop another simple algorithm for the MSTP, known as Prim's algorithm. This algorithm builds a spanning tree from scratch by fanning out from a single node and adding edges one at a time. It maintains a tree spanning on a subset S of nodes and adds a nearest neighbor to S. The algorithm does so by identifying an edge $\{i,j\}$ of minimum cost in the cut $[S,\overline{S}]$. It adds edge $\{i,j\}$ to the tree, node j to S, and repeats this basic step until $S = N$.

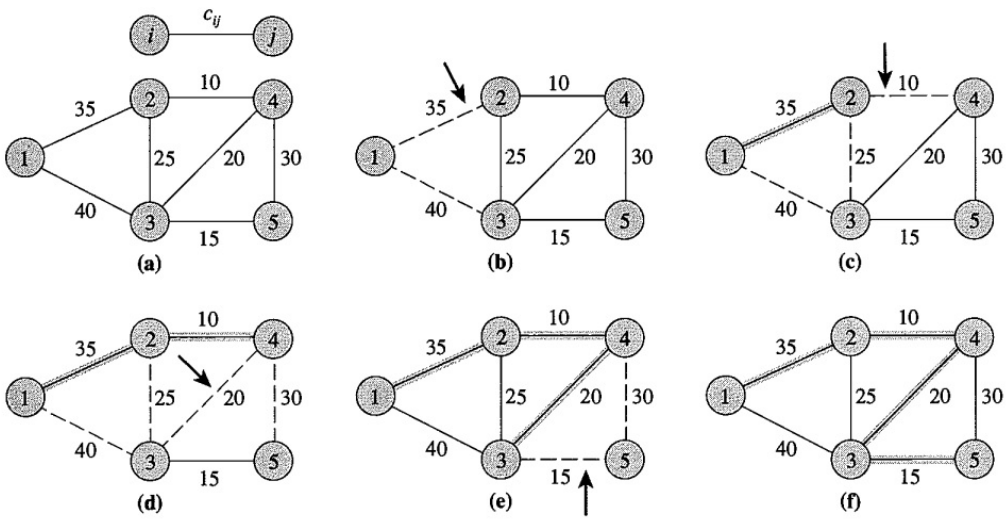
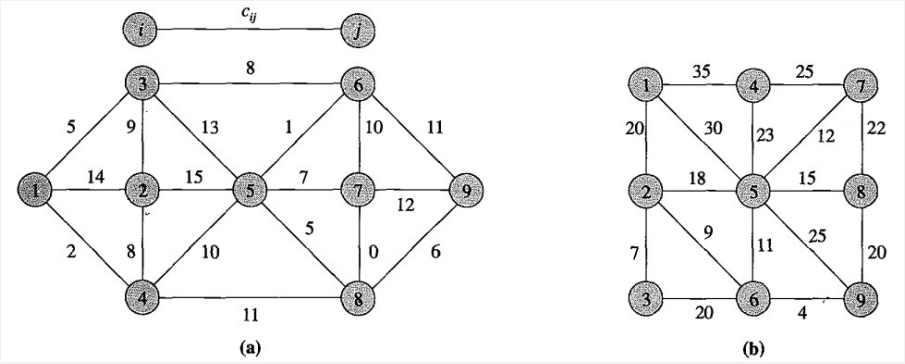


Figure 6.6: Illustrating Prim's algorithm

We illustrate Prim's algorithm on the same example, shown in Figure 6.6(a), that we used earlier to illustrate Kruskal's algorithm. Suppose, initially, that $S = \{1\}$. The cut $(S, \overline{S})$ contains two edges, $\{1, 2\}$ and $\{1, 3\}$, and the algorithm selects the edge $\{1, 2\}$ (see Figure 6.6(b)). At this point $S = \{1, 2\}$, and the cut $(S, \overline{S})$ contains the edges $\{1, 3\}$, $\{2, 3\}$, and $\{2, 4\}$. The algorithm selects edge $\{2, 4\}$ since it has the minimum cost among these three edges (see Figure 6.6(c)). In the next two iterations, the algorithm adds edge $\{3, 4\}$ and then edge $(3, 5)$; Figure 6.6(d)-(e) shows the details of these iterations. Figure 6.6(f) shows the MST produced by the algorithm.

Exercise 6.3.

Using Prim's algorithm, find a MST of the following graphs



When Prim's algorithm is applied to the first graph, by initially setting $S = \{1\}$, then the order in which the edges are included in the spanning tree is $\{1, 4\}$, $\{1, 3\}$, $\{2, 4\}$, $\{3, 6\}$, $\{5, 6\}$, $\{5, 8\}$, $\{7, 8\}$, and $\{8, 9\}$. In the case of the second graph, by initially setting $S = \{1\}$, the order in which edges are included in the spanning tree is $\{1, 2\}$, $\{2, 3\}$, $\{2, 6\}$, $\{6, 9\}$, $\{5, 6\}$, $\{5, 7\}$, $\{5, 8\}$, and $\{4, 5\}$.

6.6 Minimum Spanning Trees and Linear Programming

It is interesting to discuss how to formulate the MSTP as an ILP.

Let $E(S)$ denote the set of edges contained in the subgraph of $G = (N, E)$ induced by the node set S (i.e., $E(S)$ is the set of edges of E with both endpoints in S). Consider the following ILP formulation of the MSTP

$$\min \sum_{\{i,j\} \in E} c_{ij} x_{ij} \tag{6.2a}$$

$$\text{s.t.} \sum_{\{i,j\} \in E} x_{ij} = n - 1 \tag{6.2b}$$

$$\sum_{\{i,j\} \in E(S)} x_{ij} \leq |S| - 1 \quad \forall S \subset N : |S| \geq 3 \tag{6.2c}$$

$$x_{ij} \in \{0, 1\} \qquad \forall \{i, j\} \in E \tag{6.2d}$$

In this formulation, the binary variable x_{ij} indicates whether we select edge $\{i, j\}$ as part of the chosen spanning tree (note that constraints (6.2c) with $|S| = 2$ imply that each $x_{ij} \leq 1$). Constraint (6.2b) is a

cardinality constraint implying that we choose exactly $n - 1$ edges, and constraint (6.2c) imply that the set of chosen edges contain no cycles. As a function of the number of nodes in the network, this model contains an exponential number of constraints.

Exercise 6.4.

Suppose that you want to determine a spanning tree T that minimizes the objective function $\sqrt{\sum_{\{i,j\}\in T} c_{ij}^2}$. How would you solve this problem?

Observe that a spanning tree T that minimizes the objective function $\sqrt{\sum_{\{i,j\}\in T} c_{ij}^2}$ must also be optimal when the objective function is $\sum_{\{i,j\}\in T} c_{ij}^2$. To optimize this latter objective, one can identify a spanning tree in the graph G with the cost of each edge $\{i,j\}$ as c_{ij}^2.

Exercise 6.5.

Let T^* be a minimum spanning tree of a graph $G = (N, E)$.

Sensitivity Analysis For any edge $\{i,j\} \in E$, we define its cost interval as the set of values of c_{ij} for which T^* continues to be a minimum spanning tree. Describe a method for determining the cost interval of a given edge $\{i,j\} \in E$.

Arc Deletions How can you re-optimize the minimum spanning tree T^* after deleting an edge $\{i,j\} \in E$ from the graph?

Arc Additions How can you re-optimize the minimum spanning tree T^* after adding an edge $\{i,j\}$ to E?

Sensitivity Analysis There are two cases:

- When $\{i,j\} \in T^*$, then the maximum value of c_{ij} is the minimum value of the cost of any nontree edge in the cut formed upon removing $\{i,j\}$ from T^*. Let $\{k,\ell\}$ be such a nontree edge, then the cost interval of the edge $\{i,j\}$ is $[-\infty, c_{k\ell}]$;
- When $\{i,j\} \notin T^*$, then the minimum value of c_{ij} is equal to the maximum value of the cost of any tree edge in the cycle W formed by adding edge $\{i,j\}$ to T^*. Let $\{k,\ell\}$ be such a maximum cost tree edge in W, then the cost interval of edge $\{i,j\}$ is $[c_{k\ell}, +\infty]$.

Arc Deletions If the edge being deleted is a nontree edge, then the minimum spanning tree T^* does not change. On the other hand, if the edge $\{i,j\}$ being deleted is a tree edge, then we should obtain a minimum cost edge $\{k,\ell\}$ in the $i - j$ cut $(S, \overline{S})$ formed by removing edge $\{i,j\}$ from T^*. The new minimum spanning tree is simply the one formed by replacing edge $\{i,j\}$ in T^* by the edge $\{k,\ell\}$.

Arc Additions Let P be the unique path from node i to node j in T^* and edge $\{k,\ell\}$ be a maximum cost edge in P. If $c_{ij} \geq c_{k\ell}$, then T^* remains optimal; otherwise, replacing edge $\{k,\ell\}$ by the edge $\{i,j\}$ yields an optimal tree of the modified graph.

Exercise 6.6.

A spanning tree T is a *balanced spanning tree* if, among all spanning trees, the difference between the maximum edge cost in T and the minimum edge cost in T is as small as possible. Provide an ILP model of the problem.

$$\min \quad \beta - \alpha \tag{6.3a}$$

$$\text{s.t.} \sum_{\{i,j\} \in E} x_{ij} = n - 1 \tag{6.3b}$$

$$\sum_{\{i,j\} \in E(S)} x_{ij} \leq |S| - 1 \qquad \forall S \subset N \,:\, |S| \geq 3 \tag{6.3c}$$

$$c_{ij} x_{ij} \leq \beta \qquad \forall \{i,j\} \in E \tag{6.3d}$$

$$M - (M - c_{ij}) x_{ij} \geq \alpha \qquad \forall \{i,j\} \in E \tag{6.3e}$$

$$x_{ij} \in \{0,1\} \qquad \forall \{i,j\} \in E \tag{6.3f}$$

$$\alpha, \beta \in \mathbb{R}_+ \tag{6.3g}$$

where α and β are two auxiliary real variables representing the minimum and maximum cost of the edges of the selected spanning tree, and M is a sufficiently large number (e.g., $M = \max_{\{i,j\} \in E} \{c_{ij}\}$).

Chapter 7

Minimum Cost Flows

7.1 Introduction

Let $G = (N, A)$ be a directed graph with a cost c_{ij} and a capacity u_{ij} associated with every arc $(i, j) \in A$. We associate, with each node $i \in N$, a number b_i that indicates its supply (if $b_i > 0$) or demand (if $b_i < 0$). Nodes featuring $b_i = 0$ are called transshipment nodes. The Minimum Cost Flow Problem (MCFP) calls for finding a min-cost flow to satisfy the demands from the supplies without exceeding arc capacities. The MCFP can be stated as

$$z(\boldsymbol{x}) = \min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{7.1a}$$

$$\text{s.t.} \sum_{j\in N:(i,j)\in A} x_{ij} - \sum_{j\in N:(j,i)\in A} x_{ji} = b_i \quad \forall i \in N \tag{7.1b}$$

$$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in A \tag{7.1c}$$

Let $n = |N|$ and $m = |A|$. We make the following assumptions:

Assumption 7.1. *All data (cost, supply/demand, and capacity) are integral.*

Assumption 7.2. *The graph is directed.*

Assumption 7.3. *The supplies/demands at the nodes satisfy the condition $\sum_{i\in N} b_i = 0$, and the MCFP has a feasible solution.*

Assumption 7.4. *All arc costs are non-negative.*

The algorithms to solve the MCFP rely on the concept of residual graphs, as defined in Chapter 5. The residual graph $G(\boldsymbol{x})$ corresponding to a flow $\boldsymbol{x}$ is defined as follows. We replace each arc $(i, j) \in A$ by two arcs (i, j) and (j, i). The arc (i, j) has cost c_{ij} and residual capacity $r_{ij} = u_{ij} - x_{ij}$, and the arc (j, i) has cost $c_{ji} = -c_{ij}$ and residual capacity $r_{ji} = x_{ij}$. The residual graph consists of arcs with positive residual capacity only.

7.2 Applications

7.2.1 Distribution Problems

A car manufacturer has several manufacturing plants and produces several car models at each plant that it then ships to geographically dispersed retail centers throughout the country. Each plant has a maximum production capacity for each car model. Each retail center requests a specific number of cars of each model. The firm must determine the production plan of each model at each plant and a shipping pattern that satisfies the demands of each retail center and minimizes the overall cost of production and transportation knowing that there are bounds on the number of cars of each model that can be shipped from each plant and each retailer. Two types of costs must be considered: a production cost for each car model and each plant, and a transportation cost for each car model between each plant/retailer pair. We assume that the total production capacity of all the plants for each car model exceeds the sum of the retailer demands of the corresponding car model.

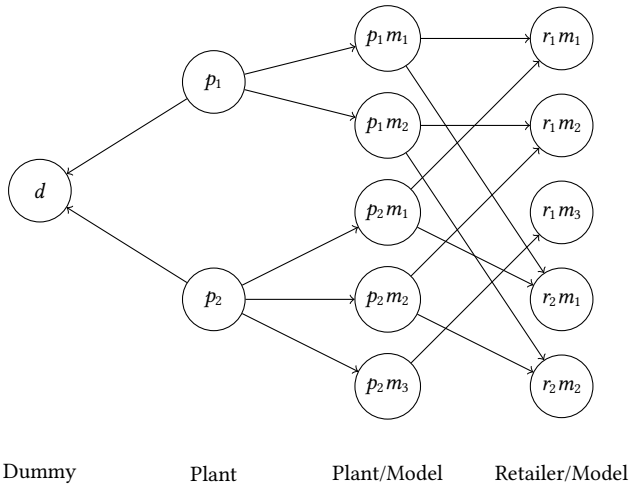


Figure 7.1: Production-distribution graph

Figure 7.1 illustrates a situation with two manufacturing plants, two retailers, and three car models, where the third car model cannot be produced at the first plant and the first retailer does not require any car of the third model. This graph has four types of nodes: (i) plant nodes, representing various plants; (ii) plant/model nodes, corresponding to each model made at a plant; (iii) retailer/model nodes, corresponding to the models required by each retailer; and (iv) a dummy node to handle the extra production capacity of the plants.

The b_i value of each plant node is equal to the sum of the production capacities of the car models that can be produced there. Plant/model nodes are transshipment nodes. The b_i value of each retailer/model node is equal to the opposite of the request of the corresponding retailer of the corresponding car model. The b_i of the dummy node is equal to the sum of all the requests of the retailers minus the sum of the production capacities of the plants.

The graph contains three types of arcs:

Production arcs These arcs connect a plant node to a plant/model node; the cost of this arc is the

cost of producing the model at that plant. We place upper bounds on these arcs to control for the max production of each particular car model at the plants.

Transportation arcs These arcs connect plant/model nodes to retailer/model nodes; the cost of such an arc is the total cost of shipping one car from the manufacturing plant to the retail center. These arcs have upper bounds imposed on their flows to model contractual agreements with shippers or capacities imposed on any distribution channel.

Dummy arcs These arcs plant nodes to the dummy nodes. These arcs have zero costs and infinite capacity.

The production and shipping schedules for the automobile company correspond in a one-to-one fashion with the feasible flows in this graph. Consequently, a min-cost flow yields an optimal production and shipping schedule.

7.2.2 Balancing of Schools

Suppose that a municipality has a set S of schools. We divide the municipality into a set L of districts, and let b_i and g_i denote the number of boys and girls at district $i \in L$. Let d_{ij} be a distance measure that approximates the distance any student at district $i \in L$ must travel if he or she is assigned to school $j \in S$. School $j \in S$ can enroll u_j students. Finally, let $\overline{p}$ denote an upper bound on the percentage of boys assigned to each school (we choose these numbers so that school $j \in S$ has same percentage of boys as does the municipality). The objective is to assign students to schools in a manner that maintains the stated boys/girls balance and minimizes the total distance traveled by the students. For the sake of simplicity, we assume that $\sum_{i \in L}(b_i + g_i) = \sum_{j \in S} u_j$.

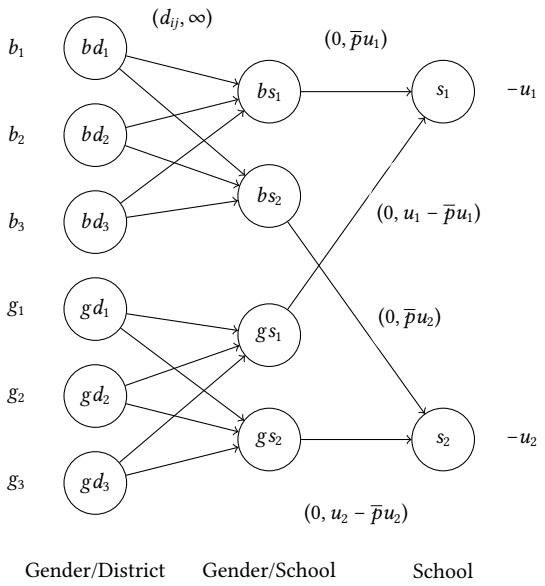


Figure 7.2: Graph for the school balancing problem

We model this problem as a MCFP. Figure 7.2 shows the graph for a three-district, two-school problem. We model each district $i \in L$ as two nodes bd_i and gd_i (which are the supply nodes) and each school $j \in S$ as three nodes bs_j, gs_j, and s_j - nodes s_j are the demand nodes. The decision variables are the number of boys assigned from district $i \in L$ to school $j \in S$ (which we represent by an arc from node bd_i to node bs_j) and the number of girls assigned from district $i \in L$ to school $j \in S$ (which we represent by an arc from node gd_i to node gs_j). These arcs are uncapacitated, and we set their per unit flow cost equal to d_{ij}. For each $j \in S$, we connect the nodes bs_j and gs_j to the school node s_j. The flow on the arcs (bs_j, s_j) and (gs_j, s_j) denotes the total number of boys and girls assigned to school j. Since each school must satisfy upper bounds on the number of boys it enrolls, we set the arc capacity of the arc (bs_j, s_j) equal to $\overline{p}u_j$. For each school, arc (gs_j, s_j) has a capacity of $u_j - \overline{p}u_j$.

Please notice that the proposed solution, represented in Figure 7.2, is not the only correct way of modeling the problem as a MCFP.

7.2.3 Optimal Loading of Hopping Airline

A commuter airline uses a plane, with a capacity to carry p passengers, on a "hopping flight", as shown in Figure 7.3. The hopping flight visits the cities 1, 2, 3, ..., n, in a fixed sequence. The plane can pick up passengers at any node and drop them off at any other node. Let b_{ij} denote the number of passengers available at node i who want to go to node j, and let f_{ij} denote the fare per passenger from node i to node j. The airline would like to determine the number of passengers that the plane should carry between the various origins to destinations to maximize the total fare per trip while never exceeding the plane capacity.

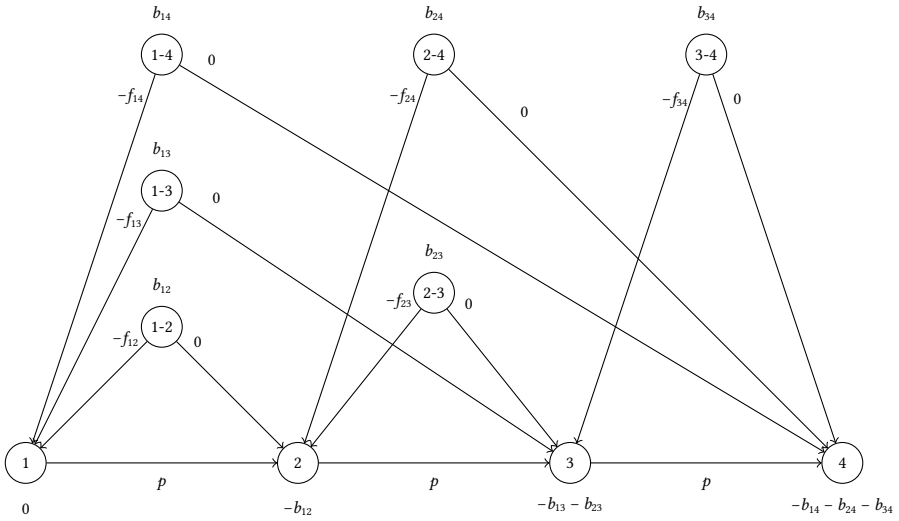


Figure 7.3: Formulating the hopping plane flight problem as a MCFP: labels next to nodes indicate b_i, labels above arcs the arc cost c_{ij}, labels below arcs the arc capacity u_{ij}

Figure 7.3 shows a MCFP formulation of this hopping plane flight problem. The graph contains data for only those arcs with nonzero costs and with finite capacities: any arc without an associated cost has a zero cost; any arc without an associated capacity has an infinite capacity. Consider, for

example, node 1. Three types of passengers are available at node 1, those whose destination is node 2, node 3, or node 4. We represent these three types of passengers by the nodes $1-2$, $1-3$, and $1-4$ with supplies b_{12}, b_{13}, and b_{14}. A passenger available at any such node, say $1-3$, either boards the plane at its origin node by flowing through the arc $(1-3, 1)$, and thus incurring a cost of $-f_{13}$ units, or never boards the plane which we represent by the flow through the arc $(1-3, 3)$.

Exercise 7.1.

Centralized teleprocessing graphs often contain many (as many as tens of thousands) geographically dispersed terminals. These terminals need to be connected to a CPU either by direct lines or though concentrators. Each concentrator is connected to the CPU through a high-speed, cost-effective line that is capable of merging data flow streams from different terminals and sending them to the CPU. Suppose that the concentrators are in place and that each concentrator can handle at most K terminals. For each terminal j, let c_{oj} denote the cost of laying down a direct line from the CPU to the terminal, and let c_{ij} denote the line construction cost for connecting concentrator i to terminal j. The decision problem is to construct the min-cost graph for connecting the terminals to the CPU. Formulate this problem as a MCFP.

Construct the graph $G = (\{s\} \cup N_c \cup N_t, A)$, where the node s represents the CPU and has a supply of $|N_t|$ units (i.e., $b_s = |N_t|$), each node $i \in N_c$ denotes the concentrator i ($b_i = 0$), and each node $j \in N_t$ denotes the terminal j and has a demand of 1 unit (i.e., $b_j = -1$). The arc set A contains three types of arcs: (i) arcs $\{(s, j) \mid j \in N_t\}$ of cost c_{oj} and capacity 1 representing which terminals are directly connected to the CPU; (ii) arcs $\{(i, j) \mid i \in N_c, j \in N_t\}$ of cost c_{ij} and capacity 1 representing the terminals connected to the CPU via the concentrators; (iii) arcs $\{(s, i) \mid i \in N_c\}$ of cost 0 and capacity K representing the number of terminals connected to the CPU via concentrator i. A min-cost flow in this graph determines a min-cost graph for connecting the terminals to the CPU.

Exercise 7.2.

A library facing insufficient primary storage space for its collection is considering the possibility of using secondary facilities to store portions of its collection. These options are preferred to an expensive expansion of primary storage. Each secondary storage facility has limited capacity and a particular access costs for retrieving information. Through appropriate data collection, we can determine the usage rates for the information needs of the users. Let q_j denote the capacity of storage facility j, and v_j denote the access cost per unit item from this facility. In addition, let a_i denote the number of items of a particular class i requiring storage and let u_i denote the expected rate (per unit time) that we will need to retrieve books from this class. Our goal is to store the books in a way that will minimize the expected retrieval cost. Show how to formulate the problem of determining an optimal policy as a MCFP.

Construct the graph $G = (C \cup F \cup \{t\}, A)$, where C contains one node corresponding to each book class, F contains one node corresponding to each storage facility, and t is a dummy demand node. The arc set A consists of two arc sets, A_1 and A_2 (i.e., $A = A_1 \cup A_2$) defined as follows: $A_1 = \{(i, j) \mid i \in C, j \in F\}$ and $A_2 = \{(j, t) \mid j \in F\}$. The cost of each arc $(i, j) \in A_1$ is equal to $u_i \cdot v_j$, and

the capacity of these arcs is infinite. The cost of each arc $(j, t) \in A_2$ is zero, but their capacity is equal to q_j. The supply of each node $i \in C$ is equal to a_i (i.e., $b_i = a_i$) while the demand of node t is equal to $\sum_{i \in C} a_i$ (i.e., $b_i = -\sum_{i \in C} a_i$). Nodes of the set F are transshipment nodes.

A minimum cost flow is found in the network thus constructed. The flow on the arc $(i, j) \in A_1$ for $i \in C$ and $j \in F$ denotes the number of books in the ith class assigned to the jth facility.

Acronyms

AP Assignment Problem.

ILP Integer Linear Programming.

LB Lower Bound.

LP Linear Programming.

MCFP Minimum Cost Flow Problem.

MFP Maximum Flow Problem.

MST Minimum Spanning Tree.

MSTP Minimum Spanning Tree Problem.

NFP Network Flow Problem.

SPP Shortest Path Problem.

UB Upper Bound.