Shu Lu *and* Quoc Tran-Dinh

Introduction to Optimization

From Linear Programming to Nonlinear Programming

October 16, 2019

STOR-UNC-Chapel Hill

Chapter 7

Applications of Linear Programming

7.1 Overview

Linear programming has many applications in fields such as computer science, operations research, engineering, data science, economics, and finance. We can classify LP applications into two categories: direct and indirect applications. Chebyshev

7.1.1 Direct applications

In direct applications, we model the practical problem into an LP or a sequence of LPs. Although these LP models only use linear functions to represent the objective function and all the constraints, they are still widely used in various applications. Below is a non-exhaustive list of applications of LPs and possibly mixed integer programs (due to R. E. Bixby [3]):

- **Classical problems in applied mathematics:** Weber's problem, Chebyshev approximation, Chebyshev center, and the ℓ_1-norm approximation.
- **Engineering:** Synthesis of filters, trust topology design, sparse signal recovery, and model predictive control (MPC) [1].
- **Statistics and machine learning:** Classification, data fitting, robust LASSO, support vector machine, and sparse/structural optimization [4].
- **Transportation:** Vehicle routing, freight vehicle scheduling and assignment, depot/warehouse allocation, public transportation system operation, maintenance scheduling, and gate scheduling.
- **Industry and manufacturing:** Plant production scheduling and logistic, gasoline and chemical blending, coal blending, mining operations management,

product mix planning, manufacturing scheduling, inventory management, personnel scheduling, oil and gas production scheduling, food blending, food transportation logistics, pattern layout and cutting, production scheduling, inventory planning, and waste water recycling.

- **Finance:** Portfolio selection and optimization, cash management, lease analysis, capital budgeting and rationing, bank financial planning, accounting allocation, etc.
- **Agriculture and forestry:** Farm land management, agricultural pricing models, crop and product mix decision models, product distribution, forest land management, and planting and harvesting models
- **Public utilities and natural resources:** Power generator scheduling, natural gas distribution planning, water resource management, and public water transportation models.
- **Communications and computing:** Circuit board (VLSI) layout, logical circuit design, complex computer graphics, curve fitting, computer system capacity planning, multiprocessor scheduling, telecommunications scheduling, and telephone operator scheduling.
- **Health care:** Staff scheduling, hospital layout, health cost reimbursement, and ambulance scheduling.
- **Government and military:** Post office scheduling and planning, military logistics, target assignment, and manpower deployment.

Some applications mentioned above may not be directly formulated into a LP, but rather a mixed-integer program. In this case, LPs can be used as relaxations of such mixed-integer programs.

7.1.2 Indirect applications

Indirect applications use LPs as subproblems, intermediate steps, or auxiliary problems in other computational methods. For instance, the sequential linear programming (SLP) approach uses LPs as subproblems in solving nonlinear programs [6]. Frank-Wolfe method also uses LPs as subproblems to generate iterates [7]. This method was invented since 1950, but has recently gained attention due to its applications in machine learning and other large-scale convex programs. Branch-and-

bounds methods for integer programming also use LPs as subproblems to compute lower-bounds for each branch-and-bound iteration [9].

In this chapter, we focus on some direct applications of LPs, including the Chebyshev center problem, robust sparse signal recovery, blending problems, inventory problems, transportation problems, minimum cost network flow problems, assignment problems, and shortest path problems.

7.2 Applications of LPs

In this section, we discuss using linear programming to solve the Chebyshev center problem, robust sparse signal recovery, the blending problem, and the inventory problem.

7.2.1 Chebyshev center of a polyhedron

We consider a problem of finding the largest ball that is contained in a polyhedron described by the following linear inequalities

$$\mathscr{P} := \left\{ x \in \mathbb{R}^n \mid a_i^T x \leq b_i,\ i = 1, \cdots, m \right\}.$$

The center of this largest ball is called the *Chebyshev center* of the polyhedron.

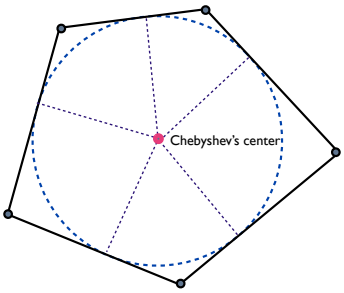

Fig. 7.1 An illustration of the Chebyshev center problem in the two dimensional space $\mathbb{R}^2$

Mathematical model: To model this problem, we present the ball centered at $\bar{x}$ of the radius r as

$$\mathscr{B}_r(\bar{x}) = \left\{ x \in \mathbb{R}^n \mid \sum_{i=1}^n (x_i - \bar{x}_i)^2 \leq r^2 \right\}.$$

If we define $u = x - \bar{x}$, then $x = \bar{x} + u$ and $\sum_{i=1}^n (x_i - \bar{x}_i)^2 = \sum_{i=1}^n u_i^2$. For any point $x \in \mathscr{B}_r(\bar{x})$ to belong to the polyhedron $\mathscr{P}$, we need $a_i^T x = a_i^T \bar{x} + a_i^T u \leq b_i$ to hold for $i = 1, \cdots, m$. By the Cauchy-Schwarz inequality, we have $a_i^T u \leq \|a_i\| \|u\| \leq r\|a_i\|$, where $\|a\| = \sqrt{\sum_{i=1}^n a_i^2}$ is the norm of a. Hence, the condition $a_i^T \bar{x} + a_i^T u \leq b_i$

holds if $a_i^T \bar{x} + r\|a_i\| \leq b_i$. The ball $\mathscr{B}_r(\bar{x})$ is contained in the polyhedron $\mathscr{P}$ when the following m inequalities are simultaneously satisfied:

$$a_i^T \bar{x} + r\|a_i\| \leq b_i,\ i = 1, \cdots, m.$$

Our goal is to maximize the radius r while satisfying m constraints above. Hence, we can formulate this problem into the following linear program:

$$\begin{cases} \max\limits_{\bar{x}, r} & r \\ \text{s.t.} & a_i^T \bar{x} + r\|a_i\| \leq b_i,\ i = 1, \cdots, m \\ & r \geq 0. \end{cases}$$

If we define $y = (\bar{x}, r)$, $c = (0, 0, \cdots, 0, 1)^T \in \mathbb{R}^{n+1}$, $A = \begin{bmatrix} a_{11} & \cdots & a_{1n} & \|a_1\| \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & \cdots & a_{mn} & \|a_m\| \end{bmatrix}$, then we can write the above problem into the following compact form:

$$\begin{cases} \max\limits_{y \in \mathbb{R}^{n+1}} & z = c^T y \\ \text{s.t.} & By \leq b, \\ & y_{n+1} \geq 0. \end{cases}$$

Example 7.1. We consider a polyhedron $\mathscr{P}$ in $\mathbb{R}^2$ generated by the following six inequalities:

$$\begin{cases} -x_1 - x_2 \leq -2 \\ -x_1 \leq 0 \\ -x_2 \leq 0 \\ -2x_1 + 3x_2 \leq 12 \\ 3x_1 - 2x_2 \leq 15 \\ 4x_1 + 5x_2 \leq 50. \end{cases}$$

Let us formulate this problem in Matlab (Excel or GAMS) using the input data as

$$B = \begin{bmatrix} -1 & -1 & \sqrt{2} \\ -1 & 0 & 1 \\ 0 & -1 & 1 \\ -2 & 3 & \sqrt{13} \\ 3 & -2 & \sqrt{13} \\ 4 & 5 & \sqrt{41} \end{bmatrix}, \quad b = \begin{pmatrix} -2 \\ 0 \\ 0 \\ 12 \\ 15 \\ 50 \end{pmatrix}, \quad \text{and} \quad c = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Then, we solve this problem to obtain $r = 2.8661$, and $\bar{x} = (3.4661, 2.8661)^T$. Figure 7.2 illustrates the result of this example. The ball is only tangents to three edges

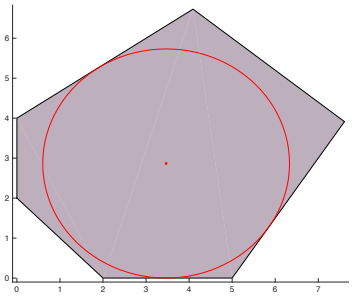

Fig. 7.2 An illustration of the Chebyshev center in $\mathbb{R}^2$ for Example 7.1

over five edges of the polyhedron.

Modeling in GAMS: If we use GAMS, then the following code will solve the above problem:

```
* Chebyshev_center.gms
SETS J /x1, x2, r/, I /c1*c6/;
PARAMETERS b /c1 -2, c2 0, c3 0, c4 12, c5 15, c6 50/;
TABLE A(I,J)
            x1     x2          r
      c1    -1     -1      1.4142
      c2    -1      0      1
      c3     0     -1      1
      c4    -2      3      3.6056
      c5     3     -2      3.6056
      c6     4      5      6.4031;
VARIABLES x(J), z;
```

```
EQUATIONS constraints(I), radius, positive_radius;
radius..z =E= x("r");
constraints(I).. sum(J, A(I,J)*x(J)) =L=b(I);
positive_radius..x("r") =G= 0;
MODEL ChebyshevCenter /all/;
SOLVE ChebyshevCenter USING LP MAXIMIZING z;
```

Note that the values in `TABLE A(I,J)` are the corresponding coefficients of the variables $\bar{x}_1$, $\bar{x}_2$ and r.

7.2.2 Robust sparse signal recovery

Problem description: Assume that we have a signal $s(t)$ that varies over time t. Via a discrete Fourier transformation, this signal is transformed into a sparse vector x in a high dimensional space $\mathbb{R}^n$, meaning that the vector x has few nonzero entries while the others are zero. The signal $s(t)$ can (approximately) be reconstructed by using the inverse of the Fourier transformation F as $s = Fx$. In signal processing, we are not able to collect the full signal s, but we rather sample it at a given sampling rate to obtain a small number of measurements y via a linear measurement operator (or a dictionary) M. Hence, we obtain the output $\bar{b} = Ms = MFx$. However, due to noise, we may obtain $b = MFx + \varepsilon$, where ε is an additive noise vector. Therefore, at the end, we obtain the following model

$$b = Ax + \varepsilon,$$

where $A = MF$ is a given matrix of size $m \times n$, where m is much smaller than n. Our goal is to minimize the mismatch between the actual output b and the clean

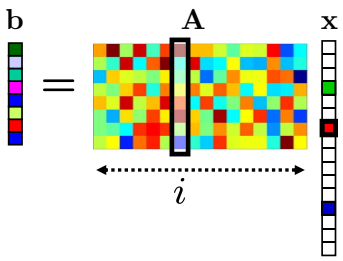

Fig. 7.3 The illustration of a sparse signal recovery model

output $\bar{b}$. We can measure the differences between these vectors by $\|b - \bar{b}\|_1 =$

$\sum_{i=1}^{m} |b_i - \bar{b}_i|$. At the same time, we also want to keep the signal vector x in the Fourier domain to be sparse. Hence, we minimize the following objective function

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{m} |b_i - (Ax)_i| + \lambda \sum_{j=1}^{n} |x_i| .$$

Here, $(Ax)_i$ is the i-th component of the vector $\bar{y} = Ax$, and $\lambda > 0$ is a fixed parameter to trade-off the sparsity level in x. That is if λ is large, then we get a more sparse solution; otherwise, we get a less sparse solution.

LP model: Using the fact that $|z| \le s$ is equivalent to $-s \le z \le s$, the above problem can be reformulated into a linear program by introducing intermediate variables u and v as follows:

$$\begin{aligned} \min_{x\in\mathbb{R}^n,u,v} \ & \textstyle\sum_{i=1}^{m} u_i + \lambda \sum_{j=1}^{n} v_i, \\ \text{s.t.} \quad & -u_i \le b_i - \textstyle\sum_{j=1}^{n} A_{ij}x_j \le u_i, \ i = 1,\cdots,m \\ & -v_j \le x_j \le v_j, \ \ j = 1,\cdots,n. \end{aligned}$$

Each constraint in the first line can be split into two distinct constraints: $\sum_{j=1}^{n} A_{ij}x_j - u_i \le b_i$ and $\sum_{j=1}^{n} A_{ij}x_j + u_i \ge b_i$ for $i = 1,\cdots,m$. Each constraint in the second line can also be written as $x_j + v_j \ge 0$ and $x_j - v_j \le 0$ for $j = 1,\cdots,n$.

Example 7.2. We consider an example as follows. First, we generate a sparse vector $x^\natural$ as a ground-truth (which is generally unknown in reality) that has s nonzero elements. Then, we generate a measurement operator A of the size $m \times n$. We form the output $b = Ax^\natural + \varepsilon$, with a sparse and random noise ε (with 20% of nonzero elements). We solve the above problem using the two different data sets as shown in the following figure, Figure 7.4. We use a Matlab code to solve this problem, and figure 7.4 shows that the recovery signal fits very well the original signal $x^\natural$ in both the small and medium size problems. As an exercise, students are asked to model this problem in Matlab using `linprog` to solve it.

7.2.3 Blending problems

Blending problems are a typical application of mathematical optimization. They involve blending several resources or materials to create one or more products corresponding to a demand. The goal is often to maximize the total profit/net sales or to minimize the cost, while satisfying all the constraints on available resources, product quality, or certain regulations, etc. Such problems arise in domains such

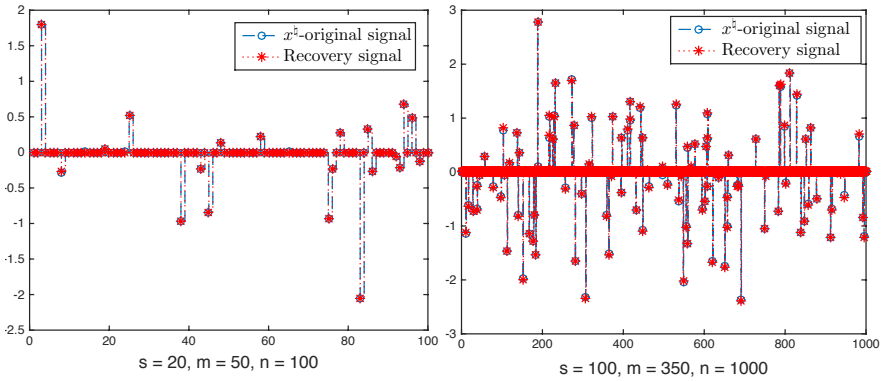

Fig. 7.4 An illustration of sparse signal recovery. Left: small problem, Right: medium problem

as metal, oil, paint, and food-processing industries. Some documents refer to this problem as an assembly problem. Since the form of this problem varies from one application to another, which is very hard to give a general formulation, we therefore prefer to describe it via a concrete example as follows.

7.2.3.1 A concrete example

We illustrate a way to solve a blending problem by performing the following steps.
Problem description: Sunco Oil manufactures three types of gasolines (gas 1, gas 2, gas 3). Each type is produced by blending three types of crude oils (crude 1, crude 2 and crude 3). The prices of selling gasolines and of purchase crude oils are given in the following table:

Gas	Sales price ($/Barrel)	Crude	Purchase price ($/Barrel)
1	70	1	45
2	60	2	35
3	50	3	25

Sunco can purchase up to 5000 barrels of each type of crude oil daily. It costs $4 to transform one barrel of oil into one barrel of gasoline, and Sunco's refinery can produce up to 14000 barrels of gasoline daily.

The three types of crude oil differ in their octane levels and sulfur content, see the table below.

Crude	Octane level	Sulfur content (%)
1	12	0.5
2	6	2.0
3	8	3.0

In a mixture of different types of crude oil, the octane levels and sulfur content blend linearly. The mixture in gas 1 (or gas 2, gas 3) must have an average octane level of at least 10 (8 for gas 2 and 6 for gas 3) and contain at most 1% sulfur (2% for gas 2 and 1% for gas 3).

The demand of each type of gasoline is as follows:

gas 1 – 3000 barrels per day
gas 2 – 2000 barrels per day
gas 3 – 1000 barrels per day

Sunco considers it an obligation to meet these demands. Assume that the gasoline cannot be stored, so it does not bring any revenue if not sold on the day it is produced. How does Sunco maximize its daily profit (The daily profit is defined as the revenue subtracting the cost)?

7.2.3.2 Mathematical model

To model Sunco's problem as a linear program, we let x_{ij} denote the number of barrels of crude oil i used daily to produce gas j, for $i = 1,2,3,\ j = 1,2,3$. We let s_j be the sales price for gas j and p_i be the purchase price of crude oil i, and let d_j be the demand of gas j.

Objective function: The objective function is to maximize the daily profit:

$$z = \underbrace{\sum_{j=1}^{3} s_j \sum_{i=1}^{3} x_{ij}}_{\text{total sales of gasolines}} - \underbrace{\sum_{i=1}^{3} p_i \sum_{j=1}^{3} x_{ij}}_{\text{total cost of crude oils}} - \underbrace{4\sum_{i=1}^{3}\sum_{j=1}^{3} x_{ij}}_{\text{total cost of transformation}} = \sum_{i=1}^{3}\sum_{j=1}^{3}(s_j - p_i - 4)x_{ij}.$$

Constraints: There are different types of constraints:

- **Daily purchase limit of the crude oil:** The purchase limit is 5000 barrels, which requires $\sum_{j=1}^{3} x_{ij} \le 5000$ for $i = 1,2,3$.
- **Daily production limit of the gasoline:** The total production limit is 14000 barrels, which requires $\sum_{i=1}^{3}\sum_{i=1}^{3} x_{ij} \le 14000$.
- **Demand constraints:** Each type of gasoline should meet the demand, which is $\sum_{i=1}^{3} x_{ij} \ge d_j$ for $j = 1,2,3$.

- **Octane level constraints:** The octane level in the gasoline and the crude, which has 3 constraints.
- **Sulfur content limit constraints:** The sulfur content in the gasoline and the crude, which also needs three constraints.

In a summary, the mathematical formulation of the LP problem for this blending process is as follows:

$$
\begin{cases}
\max\limits_{x_{ij}} \; z = \sum\limits_{j=1}^{3} s_j \sum\limits_{i=1}^{3} x_{ij} - \sum\limits_{i=1}^{3} p_i \sum\limits_{j=1}^{3} x_{ij} - 4 \sum\limits_{i=1}^{3} \sum\limits_{j=1}^{3} x_{ij} & \text{(total profit)} \\
\text{s.t.} \; \sum\limits_{j=1}^{3} x_{ij} \le 5000, \; i = 1,2,3 & \text{(daily purchase limits of crude oils)} \\
\sum\limits_{i=1}^{3} \sum\limits_{j=1}^{3} x_{ij} \le 14000 & \text{(daily production limit of gasolines)} \\
\sum\limits_{i=1}^{3} x_{ij} \ge d_j, \; j = 1,2,3 & \text{(demand constraints)} \\
\left.\begin{array}{l} 12x_{11} + 6x_{21} + 8x_{31} \ge 10(x_{11} + x_{21} + x_{31}) \\ 12x_{12} + 6x_{22} + 8x_{32} \ge 8(x_{12} + x_{22} + x_{32}) \\ 12x_{13} + 6x_{23} + 8x_{33} \ge 6(x_{13} + x_{23} + x_{33}) \end{array}\right\} & \text{(octane level constraints)} \\
\left.\begin{array}{l} 0.005x_{11} + 0.02x_{21} + 0.03x_{31} \le 0.01(x_{11} + x_{21} + x_{31}) \\ 0.005x_{12} + 0.02x_{22} + 0.03x_{32} \le 0.02(x_{12} + x_{22} + x_{32}) \\ 0.005x_{13} + 0.02x_{23} + 0.03x_{33} \le 0.01(x_{13} + x_{23} + x_{33}) \end{array}\right\} & \text{(sulfur content limits)} \\
x_{ij} \ge 0, \; i = 1,2,3, \; j = 1,2,3.
\end{cases}
$$

7.2.3.3 GAMS model

In the file `blendingoil.gms`, we define two sets `I` and `J` to be index sets of the three types of crude oil and the three types of gas respectively. The asterisk "*" is used to omit intermediate elements in a set. The set `I` as defined in this example contains three labels "crude-1", "crude-2" and "crude-3". That is

```
SET  I /crude-1*crude-3/, J /gas-1*gas-3/;
```

We then define a variable `x` over `(I, J)`; this defines nine variables, one for each pair that contains a label in `I` and a label in `J`. For example, the GAMS variable `x("crude-1","gas-1")` represents the variable x_{11} in the mathematical formulation.

In the definition of the equation `obj` we use the expression

```
profit =E= sum((I,J), (sellingPrice(J)-purchasePrice(I)-unitPCost)*x(I,J))
```

to represent the sum of `sellingPrice(J)-purchasePrice(I)-unitPCost)*x(I,J)` over all `(I,J)` pairs. While the equation `obj` uses sets `I` and `J` in its definition, it is declared to be a single equation. In contrast, the equation `proDem` is declared over the set `J`. In other words, `J` is the domain of the equation `proDem`. In the definition statement of the equation,

```
proDem(J).. sum(I, x(I,J))  =g= demand(J);
```

the set `J` appears in the parentheses after the equation name `proDem` before the two dots, and works as the "controlling set" or "controlling index" of the equation. (In general, if an equation is declared over a domain, then this domain needs to be used as the controlling set in its definition statement. More experienced users can also use subsets of the domain as the controlling set.) Since the controlling set `J` consists of three labels, GAMS compiler will generate three constraints, one for each label in `J`. In contrast, `I` is used as the index of summation in the equation. The expression `sum(I, x(I,J))` gives the sum of `x(I,J))` over all indices `I`, for a fixed label `J`. After compiling `blendingoil.gms`, we can find the following excerpt from the output file `blendingoil.lst`:

```
proDem(gas-1).. x(crude-1,gas-1) + x(crude-2,gas-1) + x(crude-3,gas-1) =G= 3000;
proDem(gas-2).. x(crude-1,gas-2) + x(crude-2,gas-2) + x(crude-3,gas-2) =G= 2000;
proDem(gas-3).. x(crude-1,gas-3) + x(crude-2,gas-3) + x(crude-3,gas-3) =G= 1000;
```

Note that the above lines are extracted from the GAMS output and should not be used as GAMS statements.

As we now see, by declaring an equation over a domain, we can create multiple equations under a common name, indexed by labels in the domain. Equations `limCrude`, `octaneGas`, and `sulfurGas` are similarly declared over domains. For example, `limCrude` is declared over the set `I`, so GAMS will create one constraint for each label in `I` as can be found in `blendingoil.lst`.

Finally, we provide a complete GAMS code for this example as below:

```
*blendingoil.gms
SET
       I /crude-1*crude-3/, J /gas-1*gas-3/;
SCALAR upCrude "Crude oil available (barrels)" /5000/,
       upTotalGas "Upper limit on gasoline (barrels)" /14000/,
       unitPCost "Production cost (dollar per barrel)" /4/;
```

```
PARAMETERS
       sellingPrice(J)   /gas-1 70, gas-2 60, gas-3 50/,
       purchasePrice(I)  /crude-1 45, crude-2 35, crude-3 25/,
       octane(I)         /crude-1 12, crude-2 6, crude-3 8/,
       sulfur(I)         /crude-1 0.5, crude-2 2, crude-3 3/,
       loOctane(J)       /gas-1 10, gas-2 8, gas-3 6/,
       upSulfur(J)       /gas-1 1, gas-2 2, gas-3 1/,
       demand(J)         /gas-1 3000, gas-2 2000, gas-3 1000/;
FREE VARIABLE
       profit "total profit";
POSITIVE VARIABLES
       x(I,J) "barrels of crude I used to produce gas J";
EQUATIONS
       obj            "max total profit",
       proDem(J)      "production meets demand",
       limCrude(I)    "Limit on each type of crude oil",
       limGas         "Limit on total gasoline",
       octaneGas(J)   "Gasoline octane level",
       sulfurGas(J)   "Gasoline sulfur content";
       obj..          sum((I,J), (sellingPrice(J)-purchasePrice(I)
                      -unitPCost)*x(I,J)) =E= profit;
       proDem(J)..    sum(I, x(I,J))  =G= demand(J);
       limCrude(I)..  sum(J,x(I,J)) =L= upCrude;
       limGas..       sum((I,J),x(I,J)) =L= upTotalGas;
       octaneGas(J).. sum(I,octane(I)*x(I,J)) =G= loOctane(J)*sum(I,x(I,J));
       sulfurGas(J).. sum(I,sulfur(I)*x(I,J)) =L= upSulfur(J)*sum(I,x(I,J));
MODEL blending /ALL/;
SOLVE blending USING LP MAXIMIZING profit;
```

7.2.4 The inventory problem

Managing and maintaining inventories (goods and materials held for sale or use) are necessary for every business that handles physical products. Many companies use mathematical models built with operations research techniques to improve their inventory management procedure [8].

Inventory models can be divided into two broad categories: deterministic models and stochastic models, depending on whether the demand is predictable. They can also be classified into continuous-review models and periodic-review models. Continuous-review models treat demand as continuous flow and replenish inven-

tory whenever the inventory level drops sufficiently low, while periodic-review models divide the time span into periods and plan for how much to produce or order during each period to satisfy the demand of each period. A third approach to classify inventory models is based on whether the model considers a single location (single-echelon) or a network of locations (multiple-echelon).

Due to the scope of this course, we only present this problem through a simple example. The example we present is a deterministic periodic-review single-echelon model.

7.2.4.1 A concrete example: Sailboat inventory problem.

(a) Problem description: Sailco Corporation must determine how many sailboats to produce for each of the next four quarters (one quarter = three months). The demand d_t during quarter t ($t = 1,2,3,4$) is as given below, and the demand must be met on time.

First quarter	$d_1 = 40$ sailboats
Second quarter	$d_2 = 60$ sailboats
Third quarter	$d_3 = 75$ sailboats
Fourth quarter	$d_4 = 25$ sailboats

At the beginning of the first quarter, Sailco has an inventory of 10 sailboats. During each quarter, Sailco can produce up to 40 sailboats with regular-time labor at a cost of $400 per sailboat. By having employees work overtime during a quarter, Sailco can produce additional sailboats with overtime labor at a cost of $450 per sailboat.

For simplicity, we assume that sailboats manufactured during a quarter can be used to meet demand for that quarter. At the end of each quarter (after production has occurred and the current quarter's demand has been satisfied), a carrying or holding cost of $20 per sailboat is incurred for sailboats remaining in stock.

Goal: Sailco's objective is to minimize the total cost. How many sailboats should be produced during each quarter?

(b) Mathematical model: We define the following variables for each quarter t ($t = 1,2,3,4$):

- x_t = number of sailboats produced by regular-time labor (at $400/boat) during quarter t;

- y_t = number of sailboats produced by overtime labor (at \$450/boat) during quarter t;
- i_t = number of sailboats at hand at the end of the quarter t (will be in an inventory).

If we specify a constant $i_0 = 10$, then the number of sailboats at the end of the quarter i is

$$i_t = i_{t-1} + x_t + y_t - d_t, \quad \text{for } t = 1,2,3,4.$$

The total cost is

$$z = \sum_{t=1}^{4} (400x_t + 450y_t + 20i_t),$$

which includes the labor cost of manufacturing in regular-time and during overtime, and the incurred cost of holding the boats. We also have a constraint on the maximum production capacity: $x_t \leq 40$ for $t = 1,2,3,4$.

Putting these components together, we can write the mathematical formulation of this inventory problem as follows:

$$\begin{cases} \min\limits_{x_t, y_t, i_t} \sum\limits_{t=1}^{4} (400x_t + 450y_t + 20i_t) \\ \text{s.t. } i_0 = 10, \;\; i_t = i_{t-1} + x_t + y_t - d_t, \;\; t = 1,2,3,4, \\ \quad x_t \leq 40, \;\; t = 1,2,3,4, \\ \quad i_t, x_t, y_t \geq 0, \;\; t = 1,2,3,4. \end{cases}$$

In the above formulation, we use separate variables for sailboats produced during regular time and those produced overtime, in order to express the total cost as a linear function of variables. The irrational choices such that $(x_t, y_t) = (35, 25)$ will not appear in an optimal solution, since it can be replaced by $(x_t, y_t) = (40, 10)$ for lower cost without affecting values of other variables.

This problem is an LP problem with $n = 12$ variables, 4 equality constraints, and 4 bound constraints on x_t. All variables are nonegative.

7.2.4.2 GAMS model

To model this problem in GAMS, as before, we use the keyword `SET` to define the index set t for the quarters. For example, with the following declaration

```
SET t /1*4/;
```

the labels in the set `t` will be ordered as "1," "2," "3" and "4.". The main component is the inventory equation $i_t = i_{t-1} + x_t + y_t - d_t$. The equation `invRel` declared on the domain `t` is used to represent the constraints this constraint. To understand how the equation `invRel(t)` is defined, we need to understand the use of **ordered sets**, **logical conditions** and **conditional expressions** in GAMS.

As we know, a GAMS set is a collection of labels. For modeling convenience, GAMS puts the labels in an order. Interested readers can find more details about ordered sets in Section 13.2 of `GAMS User Guide`. For this course it suffices to know that if the labels in a (static) set are not used as labels in another set, then their order is determined by their order in the set declaration statement. The function `ord()` returns the ordinal number associated with a given label. For the above set one has `ord("1") = 1`, `ord("2") = 2`, etc, if the set `t` is defined as `SET t /1*4/;` as above. If we instead declare the set `t` as

```
SET t /4, 1, 3, 2/;
```

then the labels in the set `t` will be ordered as "4", "1", "3" and "2" with `ord("4") = 1`, `ord("1") = 2`, etc.

While elements of a set are labels instead of numbers, the expression `t-1` is recognized by GAMS as the lag operator for order sets. For an equation declared over the set `t`, the expression `t-1` in the equation means the previous element (of the current element). In contrast, `t+1` in the equation definition refers to the next element.

If we would define the equation `invRel(t)` as

```
invRel(t)..  i(t) =e=  i(t-1) + x(t)+ y(t) - d(t);
```

then GAMS would attempt to generate the following four equations, one for each element of `t`, as follows:

```
t="1": i("1") =e= i("0") + x("1") + y("1") - d("1");
t="2": i("2") =e= i("1") + x("2") + y("2") - d("2");
t="3": i("3") =e= i("2") + x("3") + y("3") - d("3");
t="4": i("4") =e= i("3") + x("4") + y("4") - d("4");
```

The problem with the above definition is that `i("0")` is undefined in GAMS, and will be treated as zero instead of the desired value 10. To overcome this problem, we modify the equation definition as follows:

```
invRel(t).. i(t) =e= i(t-1)$(ord(t) > 1) + iniInv$(ord(t)=1)+ x(t)+ y(t) - d(t);
```

The expression `i(t-1)$(ord(t) > 1)` is a conditional expression. When GAMS creates the equation for a fixed element of `t`, it will check if the condition `ord(t) > 1` holds. If that conditions holds, then the conditional expression takes the value of `i(t-1)`; otherwise, the conditional expression takes the value of zero. Similarly, the expression `iniInv$(ord(t)=1)` is a conditional expression that takes the value of `iniInv` if `ord(t) = 1` holds and zero otherwise. With the modified definition, GAMS will generate the following four individual equations:

```
t="1": i("1") =e= iniInv + x("1") + y("1") - d("1");
t="2": i("2") =e= i("1") + x("2") + y("2") - d("2");
t="3": i("3") =e= i("2") + x("3") + y("3") - d("3");
t="4": i("4") =e= i("3") + x("4") + y("4") - d("4");
```

The conditions `ord(t) > 1` and `ord(t) = 1` are examples of logical conditions, that are heavily used in GAMS. Logical conditions are special expressions that evaluate to a value of `True` or `False`. Numerical comparisons and logical operators can be used in logical conditions.

A numerical comparison compares the values of two numerical expressions, with one of the following operators:

$$<,\ =,\ >,\ <=,\ >=\ \text{and}\ <>.$$

For example `(2 < 1)` is a numerical comparison that is always `False`, and `ord(t) > 1` is `True` if the element of `t` considered is not the first label. A logical operator can be one of the following:

`not, and, or.`

These single expressions can be combined in a more complicated expression. For example,

- the logical condition `(2 < 1) and (3 < 4)` is `False`,
- the logical condition `not(2 < 1)` is `True`,
- the logical condition `(2 < 1) or (3 < 4)` is `True`.

Another operator that can be used in logical conditions is the function `sameas(A,B)`. This function compares if two strings `A` and `B` are exactly the same. For example, `sameas(i,"3")` evaluates to be `True` if the element of `i` considered here is the same as the string `"3"`. Using this function, we can rewrite the definition of `invRel(t)` as

```
invRel(t).. i(t) =e= i(t-1)$ (not sameas(t,'1')) + iniInv$(sameas (t,'1'))+ x(t)+ y(t) - d(t);
```

Note that it is wrong to write the above equation as

```
invRel(t).. i(t)=e=i(t-1)$(t>1)+iniInv$(t=1)+ x(t)+ y(t)-d(t);
```

because elements of the set `t` are not numbers and cannot be compared using the numerical comparators.

In general, a conditional expression is of the form

```
numerical_expression$logical_condition
```

where we put a dollar sign ($) after a numerical expression `numerical_expression`, followed by a logical condition `logical_condition`. If this logical condition is true, then the entire expression takes the value of the numerical expression. Otherwise the entire expression takes the value of zero. We will see more ways of using the dollar sign for conditional assignments and control of the domain.

The constraints $x_t \leq 40$ for $t = 1,2,3,4$ in the mathematical model are treated in GAMS by the following statement

```
x.up(t)=upRegProd;
```

which assigns the upper bound `upRegProd` for each $x(t)$. Note that the above statement is an assignment statement in GAMS, instead of a definition statement for an equation. For this reason, we use the equal sign instead of `"=e="` between the two sides of the equality. Note that the above statement assigns a common upper bound for `x("1"), x("2"), x("3")` and `x("4")` simultaneously. If one would need to assign an upper bound to `x("1")` only, then one should write

```
x.up("1")=upRegProd;
```

For lower bounds, one must use `.lo` instead of `.up`.

There is a potential benefit with assigning upper bounds to variables, instead of declaring and defining equations for them. Some LP solvers in GAMS handle bound constraints in special ways, different from other (general) constraints. This can bring much more efficiency in some large-scale problems.

To this end, we can summarize the entire GAMS code for the above inventory problem.

```
*sailcoinv.gms
SET
    t "indices of quarters" /1*4/;
SCALAR
    iniInv "Inventory at the beginning of first quarter"     /10/,
    upRegProd "Upper limit of regular-time production"       /40/,
    regUnitCost "Regular time unit cost (dollar per boat)"   /400/,
    overUnitCost "Overtime unit cost (dollar per boat)"      /450/,
    holdCost "Holding cost (dollar per boat)"                /20/;
PARAMETERS
    d(t) /1 40, 2 60, 3 75, 4 25/;
FREE VARIABLE cost "total cost";
POSITIVE VARIABLES
    x(t),
    y(t),
    i(t);
EQUATIONS
    obj        "min total cost",
    invRel(t)  "Relation between inventory from different periods";
  obj.. cost =E= SUM(t, regUnitCost*x(t) + overUnitCost*y(t) + holdCost*i(t));
  invRel(t).. i(t) =E= i(t-1)$(ord(t)>1) + iniInv$(ord(t) = 1) + x(t) + y(t) - d(t);
  x.up(t) = upRegProd;
MODEL sailboat /ALL/;
SOLVE sailboat USING LP MINIMIZING cost;
```

We note that the command line in this GAMS code

```
  invRel(t).. i(t) =E=  i(t-1)$(ord(t)>1) + iniInv$(ord(t) = 1) + x(t) + y(t) - d(t);
```

can be replaced by

```
invRel(t).. i(t) =e=i(t-1)$ (not sameas(t,'1')) + iniInv$(sameas (t,'1'))+ x(t)+ y(t) - d(t);
```

7.3 Transportation problems

In operations research, the transportation problem is often reformulated into an LP problem. However, this LP formulation has special structure that allows us to treat it individually, and to develop different simplex algorithmic variants to solve it efficiently. Due to time limit, we only focus in this section the mathematical modeling aspects of this problem. Methods for solving this problem can be found in many textbooks including [2, 5, 10].

7.3.1 Problem description and mathematical formulation

Problem description: The transportation problem is a type of problems that seek a minimum-cost distribution of a given commodity/product from a group of supply points to a group of demand points.

- Each supply point $i = 1, \cdots, m$ has a fixed supply s_i that is ready to distribute.
- Each demand point $j = 1, \cdots, n$ requires a demand d_j that must be satisfied.
- It costs c_{ij} to ship each unit of the commodity from the supply point i to the demand point j.

The goal is to find a transportation plan that minimizes the total shipping cost, while it satisfies the demand at each demand point, and does not exceed the supply limitation at each supply point.

If we consider supply points and demand points as nodes of a graph (network), then we can visualize the transportation problem as a directed bipartite graph as in Figure 7.5. Here, each supply node i with a supply capacity s_i is connected to each

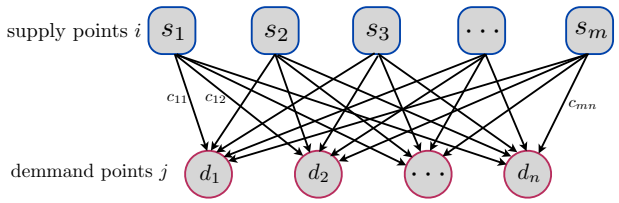

Fig. 7.5 An illustration of a transportation network.

demand node j with a demand d_j by a link.

Mathematical formulation: Let us define x_{ij} to be the amount to ship from supply point i to demand point j. These variables form a matrix $X = (x_{ij})$ called the transportation plan. Then, we can formulate the problem as follows:

$$\begin{cases} \min\limits_{x_{ij}} \;\; z = \sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n} c_{ij}x_{ij} \\ \text{s.t.} \qquad \sum\limits_{j=1}^{n} x_{ij} \le s_i,\; i = 1, \cdots, m \;\; \text{(supply constraints)} \\ \qquad\quad \sum\limits_{i=1}^{m} x_{ij} \ge d_j,\; j = 1, \cdots, n \;\; \text{(demand constraints)} \\ \qquad\quad x_{ij} \ge 0,\; i = 1, \cdots, m,\; j = 1, \cdots, n \end{cases} \tag{7.1}$$

Let us demonstrate how to formula this problem using GAMS via the following example.

Example 7.3. Powerco has three electric power plants that supply the needs of four cities. The supply capacity of each power plant and the demand in each city (in million kwh/hour) are given in the last column and the last row, respectively of the table below.

From/to	City 1	City 2	City 3	City 4	Supply
Plant 1	\$8	\$6	\$10	\$9	35
Plant 2	\$9	\$12	\$13	\$7	50
Plant 3	\$14	\$9	\$16	\$5	40
Demand	45	20	30	30	

The cost of sending 1 million kwh of electricity from each plant to each city is also given in the above table. How does Powerco minimize the total cost of meeting all cities' demands?

GAMS model: To model this problem, we use a two-dimensional table `unitCost(I, J)` to represent the costs c_{ij} between the plant i to the city j. The domain of this table is the set of ordered pairs in the Cartesian product of sets `I` and `J`. When using a table one needs to line up the entries properly so that the values are under the appropriate headings. If there are blank entries in the table, they are interpreted as zero. The table declaration statement does not use two slashes to enclose data, as the declaration statement of parameters, but it still needs to end with a semicolon. Experienced GAMS users may sometimes prefer to omit the comma separating different items in a statement when starting each item with a new line, as at some places in `transportation.gms`.

We can model this problem with the following complete GAMS model.

```
*transportation.gms
SETS
    I "plants" /p1*p3/
    J "cities" /c1*c4/;
PARAMETERS
    s(I)  "Supply of each plant"
         /p1 35
          p2 50
          p3 40/,
```

```
     d(J) "Demand of each city"
          /c1    45
           c2    20
           c3    30
           c4    30/;
TABLE
    unitCost(I, J)
                    c1       c2       c3       c4
        p1          8        6        10       9
        p2          9        12       13       7
        p3          14       9        16       5;
VARIABLES z, x(I, J);
POSITIVE VARIABLE x;
EQUATIONS cost, supply(I), demand(J);
  cost..      SUM( (I, J), unitCost(I, J)*x(I, J) ) =E= z;
  supply(I).. SUM(J, x(I, J)) =L=  s(I);
  demand(J).. SUM(I, x(I, J)) =G=  d(J);
MODEL transport /ALL/;
SOLVE transport USING LP MINIMIZING z;
```

7.3.2 Balanced transportation problems

We say that a transportation problem is **balanced** if the total supply equals total demand, i.e.:

$$\sum_{i=1}^{m} s_i = \sum_{j=1}^{n} d_j.$$

In this case, we call it a **balanced transportation problem**. For example, the Powerco example above is such a balanced transportation problem.

Theorem 7.1. *In a balanced transportation problem, any feasible solution x satisfies*

$$\sum_{j=1}^{n} x_{ij} = s_i \text{ for each } i = 1, \cdots, m,$$

and

$$\sum_{i=1}^{m} x_{ij} = d_j \text{ for each } j = 1, \cdots, n.$$

Proof. If $X = (x_{ij})$ is a feasible solution in a balanced transportation problem, then

$$\sum_{j=1}^{n} d_j \leq \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} \leq \sum_{i=1}^{m} s_i,$$

where the first item equals the last item by assumption. So

$$\sum_{j=1}^{n} d_j = \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} = \sum_{i=1}^{m} s_i.$$

Since $d_j \le \sum_{i=1}^{m} x_{ij}$ for each $j = 1,\cdots,n$, from the equality $\sum_{j=1}^{n} d_j = \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}$ we can deduce $d_j = \sum_{i=1}^{m} x_{ij}$ for each $j = 1,\cdots,n$. Similarly, since $\sum_{j=1}^{n} x_{ij} \le s_i$ for each $i = 1,\cdots,m$, from the equality $\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} = \sum_{i=1}^{m} s_i$ we can deduce $\sum_{j=1}^{n} x_{ij} = s_i$ for each $i = 1,\cdots,m$. □

According to Theorem 7.1, a balanced transportation problems can alternatively be written as:

$$\begin{cases} \min\limits_{x_{ij}} \;\; z = \sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n} c_{ij}x_{ij} \\ \text{s.t.} \quad\;\; \sum\limits_{j=1}^{n} x_{ij} = s_i,\; i = 1,\cdots,m \quad \text{(supply constraints)} \\ \qquad\;\; \sum\limits_{i=1}^{m} x_{ij} = d_j,\; j = 1,\cdots,n \;\text{(demand constraints)} \\ \qquad\qquad x_{ij} \ge 0,\; i = 1,\cdots,m, j = 1,\cdots,n \end{cases} \tag{7.2}$$

The above new formulation is much more efficient to solve than the original formulation (7.1). More explanation will be given in the subsequent sections on network flows.

Now, we investigate more properties of the balanced transportation problem (7.2). First we arrange the variables x_{ii} into a vector $x \in \mathbb{R}^{mn}$ of the form

$$x = (x_{11}, x_{12}, \cdots, x_{1n}, x_{21}, x_{22}, \cdots, x_{2n}, \cdots, x_{m1}, x_{m2}, \cdots, x_{mn})^T.$$

Next, we write the input data (A, b, c) of the balanced transportation problem (7.2) as

$$A = \left[\begin{array}{cccccccccccc} 1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 1 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 1 \end{array}\right] \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \\ \\ \end{array}\right\} m \text{ rows} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} n \text{ rows} \end{array} \quad b = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \\ d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix}, \text{ and } c = \begin{pmatrix} c_{11} \\ \vdots \\ c_{1n} \\ c_{21} \\ \vdots \\ c_{2n} \\ \vdots \\ c_{m1} \\ \vdots \\ c_{mn} \end{pmatrix}.$$

Matrix A is of the size $(m+n)\times mn$. Moreover, from the form of A, we can see that each column of A has exactly two entries 1, and other entries are zero. Therefore, if we sum up each column of A, we obtain 2. This shows that A has less than $m+n$ linear independent rows. On the other hand, from the last n rows, we can form an $n\times n$ identity matrix by taking the left-bottom $n\times n$-submatrix. From the second to the m-th rows, we can form another $(m-1)\times(m-1)$ identity submatrix by taking the entries at the columns $n+1$, $2n+1,\cdots,(m-1)n+1$. Putting these submatrices together, we obtain an identity submatrix of the size $(m+n-1)\times(m+n-1)$. Hence, we can conclude that A has exactly $m+n-1$ independent rows.

Theorem 7.2. *The balanced condition $\sum_{i=1}^m s_i = \sum_{j=1}^n d_j$ is necessary and sufficient for* (7.2) *to have an optimal solution.*

Proof. If (7.2) has an optimal solution $X^* = (x_{ij}^*)$, then we have

$$\sum_{j=1}^n x_{ij}^* = s_i, \;\; i = 1,\cdots,m, \quad \text{and} \quad \sum_{i=1}^m x_{ij}^* = d_j, \;\; j = 1,\cdots,n.$$

Summing up the first expression from $i=1$ to m and the second one from $j=1$ to n, we get

$$\sum_{i=1}^m s_i = \sum_{i=1}^m\sum_{j=1}^n x_{ij}^* = \sum_{j=1}^n\sum_{i=1}^m x_{ij}^* = \sum_{j=1}^n d_j.$$

Hence, we obtain the balanced condition $\sum_{i=1}^m s_i = \sum_{j=1}^n d_j$.

Conversely, assume that $\sum_{i=1}^m s_i = \sum_{j=1}^n d_j$. We define $M := \sum_{i=1}^m s_i = \sum_{j=1}^n d_j > 0$ and $\bar{X} = (\bar{x}_{ij})$ with $\bar{x}_{ij} = \frac{s_i d_j}{M}$. Clearly, we can check that $\bar{x}_{ij} \geq 0$ for $i = 1,\cdots,m$ and $j = 1,\cdots,n$, Moreover, we have

$$\sum_{j=1}^{n} \bar{x}_{ij} = \sum_{j=1}^{n} \frac{s_i d_j}{M} = \frac{s_i}{M} \sum_{j=1}^{n} d_j = s_i, \quad i = 1, \cdots, m$$
$$\sum_{i=1}^{m} \bar{x}_{ij} = \sum_{n=1}^{m} \frac{s_i d_j}{M} = \frac{d_j}{M} \sum_{i=1}^{m} s_i = d_j, \quad j = 1, \cdots, n.$$

Hence, $\bar{X}$ is a feasible solution to (7.2). Now, we consider an arbitrary feasible solution $X = (x_{ij})$ of (7.2). It must satisfy all the constraints of (7.2). Hence, we have

$$0 \le x_{ij} \le \sum_{j=1}^{n} x_{ij} = s_i, \quad i = 1, \cdots, m, \quad j = 1, \cdots, n.$$

Therefore, the feasible set of (7.2) is nonempty and bounded. We can show that

$$Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \ge \sum_{i=1}^{m} \sum_{j=1}^{n} \min\left\{0, c_{ij} s_i\right\} = \text{constant}.$$

This shows that the objective function of (7.2) is bounded from below. By Theorem **??**, problem (7.2) must have an optimal solution. □

7.3.3 Transportation tableaux and solution methods

Since a transportation problem is an LP problem, in general, we can apply any simplex method in the previous chapter to solve this problem. However, due to its special structure, the simplex method has been specialized to solve transportation problems in a much simpler manner and efficient for large-scale problems. In this subsection, we describe one method for solving transportation problems based on transportation tableaux as a special case of the simplex method.

(a) Transportation tableaux: We can present the input data of a transportation problem in a table called **transportation tableau**. This tableau can also be used to perform a simplex method for solving the given transportation problem. A transportation tableau is a table of $m+2$ rows and $n+2$ columns as below.

	D_1	D_2	D_3	...	D_n	Supply
S_1	c_{11} x_{11}	c_{12} x_{12}	c_{13} x_{13}	... ...	c_{1n} x_{1n}	s_1
S_2	c_{21} x_{21}	c_{22} x_{22}	c_{23} x_{23}	... ...	c_{2n} x_{2n}	s_2
S_3	c_{31} x_{31}	c_{32} x_{32}	c_{33} x_{33}	... ...	c_{3n} x_{3n}	s_3
...	... ...	... ...	... ...	... ...	... ...	...
S_m	c_{m1} x_{m1}	c_{m2} x_{m2}	c_{m3} x_{m3}	... ...	c_{mn} x_{mn}	s_m
Demand	d_1	d_2	d_3	...	d_n	

The first row (called row 0) and the first column (called column 0) mark the labels of the demand nodes and supply nodes, respectively. The last column $(n+1)$ provides the supply capacity of each supply node, and the last row $(m+1)$ shows the demand of each demand node. The cells from 2nd to mth rows (row 1 to row m) and 2nd to nth columns (column 1 to column n) contains the cost c_{ij} and a feasible solution x_{ij}. The cost c_{ij} is given on the left-top conner, and the component x_{ij} is on the right-bottom conner of the cell.

(b) Finding a basic feasible solution: In order to start the simplex method for solving the transportation problem (7.2), we need to find a basic feasible solution $X = (x_{ij})$ first. There are at least two methods to find such a BFS X: The **north-west corner** method, and the **least-cost** method (also called **least-remaining cost** method). Let us describe the north-west corner method as follows:

> **Step 1:** Start from the left-top corner cell $(i, j) = (1, 1)$ (also called the north-west corner). Allocate x_{ij} with $x_{ij} = \min\{s_i, d_j\}$. There are two situations:
>
> - If $s_i > d_j$, then we cross column j, and substitute s_i by $s_i - d_j$. We obtain a new transportation tableau with the same number of rows, but less than one column.
> - If $s_i < d_j$, then we cross row i, and substitute d_j by $d_j - s_i$. We obtain a new transportation tableau with the same number of column, but less than one row.
>
> In any case, we remove either one column or one row.
>
> **Step 2:** Repeat **Step 1** with the new transportation tableau as a part of the whole transportation tableau.

The algorithm terminates after we cross all rows and columns. Record all the entries x_{ij} corresponding at each step of the algorithm, we obtain a basic feasible solution $X = (x_{ij})$.

For the least-cost method, instead of starting from the left-top conner, we start at the cell with minimum cost c_{ij}. If there are many cells, we can choose any of them. The theory of such a method can be found in several textbooks including [2, 5, 10]. We do not go into detail in this course.

Example 7.4. In order to see how the north-west corner method works, we consider Example 7.3 above. The following steps are carried out:

1. Allocate $x_{11} = \min\{35, 45\} = 35$. Since $s_1 < d_1$, we cross row 1 and replace d_1 with $d_1 \to d_1 - s_1 = 45 - 35 = 10$. The new tableau has 2 rows and 4 columns in the main part.
2. Allocate $x_{21} = \min\{50, 10\} = 10$. Since $s_2 > d_1$, we cross column 1 in the new tableau, and replace s_2 with $s_2 - d_1 = 40$. The new tableau has 2 rows and 3 columns in the main part.
3. Allocate $x_{22} = \min\{40, 20\} = 20$. Since $s_2 > d_2$, we cross column 2 in the new tableau, and replace s_2 with $s_2 - d_2 = 20$. The new tableau has 2 rows and 2 columns in the main part.
4. Allocate $x_{23} = \min\{20, 30\} = 20$. Since $s_2 < d_3$, we cross row 2 in the new tableau, and replace d_3 with $d_3 - s_2 = 10$. The new tableau has 1 rows and 2 columns in the main part.
5. Allocate $x_{33} = \min\{40, 10\} = 10$. Since $s_3 > d_3$, we cross column 3 in the new tableau, and replace s_3 with $s_3 - d_3 = 30$. The new tableau has 1 rows and 1 columns in the main part with $s_3 = 30$ and $d_4 = 30$.
6. Allocate $x_{34} = 30$, and the algorithm is completed.

From/To	City 1	City 2	City 3	City 4	Supply
Plant 1	8 [35]	6	10	9	35
Plant 2	9 [10]	12 [20]	13 [20]	7	50
Plant 3	14	9	16 [10]	5 [30]	40
Demand	45	20	30	30	125

We finally obtain a basic feasible solution as

$$X = \begin{bmatrix} 35 & 0 & 0 & 0 \\ 10 & 20 & 20 & 0 \\ 0 & 0 & 10 & 30 \end{bmatrix}.$$

This BFS has exactly $m + n - 1 = 3 + 4 - 1 = 6$ positive entries. Hence, it is a non-degenerate basic feasible solution.

(c) The simplex method: We describe one simplex method to solve the balanced transportation problem (7.2). We first define a concept so-called **closed loop** of

a BFS on a transportation tableau: A set of cells $\mathscr{L} = \{(i_k, j_k) \mid 1 \le k \le s\}$ on a transportation tableau is called a **closed loop** if

- every consecutive cells (i_k, j_k) and (i_{k+1}, j_{k+1}) are on the same row or the same column;
- no three cells are on the same row or column;
- the first and last cells are on the same row or column.

The basis $\mathscr{U}$ of any non-degenerated BFS X of (7.2) does not contain any closed loop. If we add a new cell to $\mathscr{U}$, then $\mathscr{U}$ contains a closed loop.

This method is described in detail as follows:

Initialization: Apply either the north-west corner method or the least-cost method to find a BFS $X = (x_i j)$. We denote by $\mathscr{U} = \big\{(i, j) \mid x_{ij} > 0\big\}$ the basis of the BFS X. This set has $m+n-1$ entries whenever X is a non-degenerated BFS.

Step 1: Find u_i $(i = 1, \cdots, m)$ and v_j $(j = 1, \cdots, n)$ from the following system of linear equations:

$$u_i + v_j = c_{ij}, \;\; (i, j) \in \mathscr{U}.$$

This is a system of $m+n-1$ equations with $m+n$ unknowns u_i, v_j. We can set one unknown to be zero and solve for $m+n-1$ other remaining unknowns, e.g., by a substitution method. Once u_i and v_j are computed, we compute $\bar{c}_{ij}$ as $\bar{c}_{ij} = u_i + v_j - c_{ij}$ for $(i, j) \in \mathscr{U}$.

Step 2: If $c_{ij} \le 0$ for all $(i, j) \in \mathscr{U}$, we terminate the algorithm, and conclude that X is an optimal solution.

Step 3: (*Choose entering variable*) If $\bar{c}_{ij} > 0$ for some $(i, j) \in \mathscr{U}$, then choose $x_{i_* j_*}$ as the entering variable from the rule

$$\bar{c}_{i_* j_*} = \max\big\{\bar{c}_{ij} \mid \bar{c}_{ij} > 0, (i, j) \in \mathscr{U}\big\}.$$

Step 4: (*Find a closed loop*) The set $\mathscr{U} \cup \{(i_*, j_*)\}$ contains a closed loop $\mathscr{L}$ so that $(i_*, j_*) \in \mathscr{L}$. Find this closed loop. Then, alternatively mark all cells of $\mathscr{L}$ with a "+" or "−" sign starting from (i_*, j_*) with a "+" such that any two consecutive cells must have different signs. We split $\mathscr{L}$ into two subsets $\mathscr{L}_+$ and $\mathscr{L}_-$ that contain all cells of $\mathscr{L}$ with "+" and "−" signs, respectively.

Step 5: (*Choose leaving variable*) Choose $x_{i_0j_0}$ as a leaving variable from the rule

$$x_{i_0j_0} = \min\left\{x_{ij} \mid (i,j) \in \mathscr{L}_-\right\}.$$

Step 6: (*Construct a new BFS*) We construct a new BFS $\bar{X} = (\bar{x}_{ij})$ as

$$\bar{x}_{ij} = \begin{cases} x_{ij} - x_{i_0j_0} & \text{if } (i,j) \in \mathscr{L}_- \\ x_{ij} + x_{i_0j_0} & \text{if } (i,j) \in \mathscr{L}_+ \\ x_{ij} & \text{if } (i,j) \notin \mathscr{L}. \end{cases}$$

We repeat **Step 1** to **Step 6** until the termination condition at **Step 2** is satisfied. Clearly, $\bar{X}$ constructed at **Step 6** does not contain a closed loop, and its becomes a new basic feasible solution. We note that $\bar{c}_{ij}$ computed at **Step 2** can be considered as the negative values of the reduced cost in the simplex method from the previous chapter since we are solving the minimization problem.

Example 7.5. We consider a balanced transportation problem with the input data given in the following tableau ($m = 3$ supply points and $n = 3$ demand points):

D_j / S_i	D_1	D_2	D_3	Supply
S_1	5	4	1	50
S_2	3	2	6	40
S_3	7	9	11	70
Demand	80	20	60	160

Let us apply the above simplex method to solve this problem:

- **Initialization:** We can check that the following matrix is a BFS to this transportation problem:

$$X = \begin{bmatrix} 0 & 0 & 50 \\ 20 & 20 & 0 \\ 60 & 0 & 10 \end{bmatrix} \quad \text{with } f(X) = 680.$$

We locate this BFS into the transportation tableau to obtain an initial tableau as

D_j / S_i	D_1	D_2	D_3	Supply	
S_1	5	4	1 [50]	50	$u_1 = 6$
S_2	3 [20]	2 [20]	6	40	$u_2 = 0$
S_3	7 [60]	9	11 [10]	70	$u_3 = 4$
Demand	80	20	60	160	
	$v_1 = 3$	$v_2 = 2$	$v_3 = 7$		

- **Iteration 0:** Perform the following steps:

  1. Compute u_1, u_2, u_3 and v_1, v_2, v_3 from the following linear system:

  $$u_1 + v_3 = 1,\ \ u_2 + v_1 = 3,\ \ u_2 + v_2 = 2,\ \ u_3 + v_1 = 7,\ \ u_3 + v_3 = 11.$$

  Choose $u_2 = 0$, and solve this system, we obtain $v_1 = 3$, $v_2 = 2$, $v_3 = 7$, $u_1 = 6$, $u_2 = 0$ and $u_3 = 4$. Compute $\bar{c}_{ij} = u_i + v_j - c_{ij}$ and store them into the tableau:

  $$\bar{c}_{11} = -8,\ \ \bar{c}_{12} = -8,\ \ \bar{c}_{13} = 0,\ \ \bar{c}_{21} = 0,\ \ \bar{c}_{22} = 0,\ \ \bar{c}_{23} = 1,\ \ \bar{c}_{31} = 0,\ \ \bar{c}_{32} = -3,\ \ \bar{c}_{33} = 0.$$

  2. Since $\bar{c}_{23} > 1$ (the unique value). This BFS X is not yet optimal, we continue.
  3. We choose x_{23} as the entering variable.
  4. We can form a closed loop as

  $$\mathscr{L} = \left\{(2,3)^+,\ (3,3)^-,\ (3,1)^+,\ (2,1)^-\right\}.$$

  It is easy to find

  $$\mathscr{L}_+ = \{(2,3),(3,1)\} \quad \text{and} \quad \mathscr{L}_- = \{(3,3),(2,1)\}.$$

  5. We choose $x_{i_0 j_0} = \min\{x_{33}, x_{21}\} = \min\{10, 20\} = 10 = x_{33}$ to be the leaving variable.

6. We construct the new basic feasible solution $\bar{X} = (\bar{x}_{ij})$ as

$$\begin{cases} \bar{x}_{21} = x_{21} - x_{33} = 20 - 10 = 10 \\ \bar{x}_{33} = 0 \\ \bar{x}_{23} = 0 + x_{33} = 10 \\ \bar{x}_{31} = x_{31} + x_{33} = 60 + 10 = 70 \end{cases} \quad \text{or } \bar{X} = \begin{bmatrix} 0 & 0 & 50 \\ 10 & 20 & 10 \\ 70 & 0 & 0 \end{bmatrix} \text{ with } f(\bar{X}) = 670.$$

In this case, we obtain a new transportation tableau:

D_j / S_i	D_1	D_2	D_3	Supply	
S_1	5	4	1 [50]	50	$u_1 = -5$
S_2	3 [10]	2 [20]	6 [10]	40	$u_2 = 0$
S_3	7 [70]	9	11	70	$u_3 = 4$
Demand	80	20	60	160	
	$v_1 = 3$	$v_2 = 2$	$v_3 = 6$		

- **Iteration 1:** Perform the following steps:

1. Compute new u_i $(i = 1,2,3)$ and new v_j $(j = 1,2,3)$ by solving the following linear system:

$$u_1 + v_3 = 1,\ u_2 + v_1 = 3,\ u_2 + v_2 = 2,\ u_2 + v_3 = 6,\ u_3 + v_1 = 7.$$

Choose $u_2 = 0$ we can easily obtain $v_1 = 3, v_2 = 2, v_3 = 6, u_1 = -5$, and $u_3 = 4$. Next, we compute $\bar{c}_{ij} = u_j + v_j - c_{ij}$ and obtain

$$\bar{c}_{11} = -7,\ \bar{c}_{12} = -7,\ \bar{c}_{13} = 0,\ \bar{c}_{21} = 0,\ \bar{c}_{22} = 0,\ \bar{c}_{23} = 0,\ \bar{c}_{31} = 0,\ \bar{c}_{32} = -3,\ \bar{c}_{33} = -1.$$

2. Since $\bar{c}_{ij} \le 0$ for all $(i,j) \in \mathscr{U}$, the current basic feasible solution $\bar{X}$ is optimal. We terminate the algorithm.

Finally, we obtain an optimal solution $X^* = \begin{bmatrix} 0 & 0 & 50 \\ 10 & 20 & 10 \\ 70 & 0 & 0 \end{bmatrix}$ with the optimal value $f(X^*) = 670$.

7.3.4 Unbalanced transportation problems

A transportation problem in which the total supply does not equal the total demand is called an **unbalanced transportation** problem. There are two types of unbalanced transportation problems. The first type consists of transportation problems in which the **total supply exceeds the total demand**.

An example can be obtained if we reduce the demand for city 1 in the Powerco example to 40 million kwh and keep other data unchanged. The formulation in (7.1) is still valid and feasible in this case, but formulation (7.2) becomes infeasible. To take advantage of the computational efficiency of the second formulation, we can transform the problem into a balanced problem, by adding a dummy demand point that has a demand equal to the amount of excess supply, and assigning a zero unit cost of shipments to the dummy demand point. In some situations, any unused supply is charged a holding fee. In those situations, let the unit cost to ship to the dummy demand point be the unit holding cost. In the optimal solution for the resulting balanced problem, the amount of shipments to the dummy demand point is exactly the amount of unused supply.

Example 7.6. For the Powerco example with the demand for city 1 being 40 million kwh, adding the dummy demand point results in a change in its data:

From/to	City 1	City 2	City 3	City 4	Dummy	City 5 Supply
Plant 1	$8	$6	$10	$9	$0	35
Plant 2	$9	$12	$13	$7	$0	50
Plant 3	$14	$9	$16	$5	$0	40
Demand	40	20	30	30	5	

By solving the balanced transportation problem with the above data, we can obtain an optimal solution to the unbalanced problem.

In the second type of unbalanced transportation problems, **the total demand exceeds the total supply**. We can construct an example of this type by reducing the supply at plant 1 in the Powerco example to 30 million kwh and keeping other data as in the original Example 7.3. In this case, both formulations in (7.1) and (7.2) become infeasible, as it is impossible to meet all the demand with the available supply. A common approach to modeling such situations is to impose a penalty for unmet demand at each demand point. To formulate the problem as a balanced

transportation problem, we add a dummy supply point that has a supply equal to the amount of excess demand, and let the unit cost for shipments from the dummy supply point be the unit penalty for unmet demand. In the optimal solution for the resulting balanced problem, the amount of shipments from the dummy supply point is exactly the amount of unmet demand.

Example 7.7. Consider the Powerco example with the supply at plant 1 being 30 million kwh. Suppose that other data are as given in Example 7.3, and that the penalty for each million kwh of unmet demand is \$2 for city 1, \$1 for city 2, \$1 for city 3 and \$4 for city 4. After adding the dummy demand point, the data becomes

From/to	City 1	City 2	City 3	City 4	Supply
Plant 1	\$8	\$6	\$10	\$9	30
Plant 2	\$9	\$12	\$13	\$7	50
Plant 3	\$14	\$9	\$16	\$5	40
Dummy Plant 4	\$2	\$1	\$1	\$4	5
Demand	45	20	30	30	

The above data lead to a balanced transportation problem, which we can solve to find an optimal solution to the unbalanced problem with excess demand.

7.4 The minimum cost network flow problem and its extensions

The minimum cost network flow problem (MCNFP) is to find a feasible flow in a network that achieves the minimum total cost.

7.4.1 Problem description and mathematical formulation

Problem description: A network $\mathcal{N} = (V, A)$ consists of a set V of **nodes** (or vertices), and a set A of **arcs** (or edges). Each arc is an ordered pair of nodes; the arc from node i to node j is denoted by (i, j). For any arc (i, j), we say that i is the start node and j is the end node. The arc (i, j) is said to be outgoing from node i, incoming to node j. In the type of networks we consider, there is at most one arc from each node to each other node. It is fine to have an arc from node i to j, and another arc from j to i. If we use $V \times V$ to denote the Cartesian product of V and V (i.e., the set of all ordered pairs of elements of V), then A is a subset of $V \times V$.

In an MCNFP, each node $i \in V$ of the network has a **net supply** s_i, which may be positive, zero, or negative. Each arc (i, j) is associated with a unit cost c_{ij}, an

upper bound of the flow on this arc denoted by u_{ij}, and a **lower bound** for flow on this arc denoted by l_{ij}. It is possible for u_{ij} to be ∞, and the lower bound l_{ij} is usually 0 unless explicitly stated otherwise.

Given the network $\mathscr{N} = (V,A)$ as described above, the goal of the MCNFP is to assign a number x_{ij} to each arc (i,j), which is the **flow** on that arc, such that:

(1) the flow bounds on all arcs are observed;
(2) the **flow conservation constraints** for all nodes are satisfied; and
(3) the total cost is minimized.

The flow conservation constraint for each node is also called the **balance equation** for each node. The balance equation for a given node i requires the difference between the total flow out of node k and the total flow into node k to be equal to the net supply s_i. This constraint can be written as

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{k:(k,i)\in A} x_{ki} = s_i.$$

Note that i is a fixed index in the above constraint. The expression $\sum_{j:(i,j)\in A} x_{ij}$ is the sum of x_{ij} over all nodes j such that (i,j) is an arc, and therefore gives the total amount of flow out of node i. Similarly, the expression $\sum_{k:(k,i)\in A} x_{ki}$ is the sum of x_{ki} over all nodes k such that (k,i) is an arc, and gives the total amount of flow into node i.

Example 7.8. Consider a network shown in Figure 7.6 below: Here, $V = \{1,2,3,4\}$

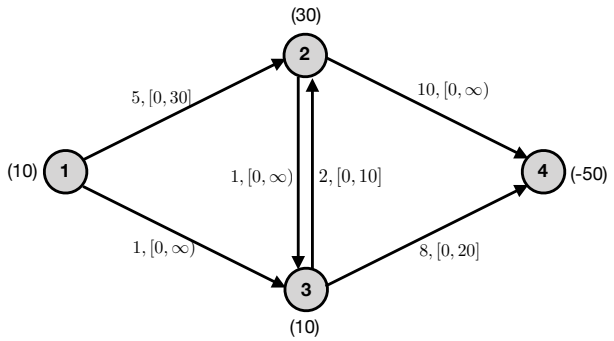

Fig. 7.6 A network representation of an MCNFP.

is the set of nodes, and $A = \{(1,2),(1,3),(2,3),(3,2),(2,4),(3,4)\}$ is the set of

arcs. The numbers next to the nodes are net supplies (e.g., the net supply for node 1 is 10, and net supply for node 4 is -50).

On each of the arcs, the first number is the unit cost and the interval gives flow bounds. For example on arc $(1,3)$ the unit cost is 1, the lower bound of flow is 0, and the upper bound is ∞.

Mathematical model: Based on the problem described above, the following is a general formulation of an MCNFP, in which $\sum_{(i,j)\in A} c_{ij}x_{ij}$ is the sum of cost over all arcs (i,j).

$$\begin{cases} \min\limits_{x_{ij}} \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} \\ \text{s.t.} \displaystyle\sum_{j:(i,j)\in A} x_{ij} - \sum_{k:(k,i)\in A} x_{ki} = s_i \quad \text{for each } i \in V \quad \text{(balance equation)} \\ \quad l_{ij} \le x_{ij} \le u_{ij} \text{ for each } (i,j) \in A \qquad\qquad \text{(flow bounds)} \end{cases} \tag{7.3}$$

Summing up all the balance equations will result in an equality $\sum_{i\in N} s_i = 0$ as the left hand sides of all balance equations be canceled out. This implies that $\sum_{i\in N} s_i = 0$ is a necessary condition for the MCNFP to be feasible.

GAMS model: We can model this problem into a GAMS file, named by `mcnfp.gms`. In this file, the set `A` is declared as a subset of the Cartesian product set $V \times V$. The latter set contains 16 elements, each being an ordered pair of nodes, among which 6 belong to `A`. In general, the following statement

```
SET set1(set2) ;
```

declares `set1` as a subset of the larger set `set2`. We will use `A` as a filter in conditional index operations and conditional equation assignments, and as the index of summation in the equation that defines the objective function.

One caution about using the set `A` in `mcnfp.gms` is to avoid declaring parameters, variables, or equations over it. In general, one can use a one-dimensional set (a set that contains a sequence of elements) or a product of multiple one-dimensional sets, but not a subset of the product of two sets, as the domain of a parameter, variable or equation. In `mcnfp.gms`, we declare parameters `c` and `u` over (V,V), even though we will only use values of those parameters for (i,j) in the set `A`. The values of parameters `c` and `u` that are not explicitly assigned in the declaration statement

are zero by default; for example, `c('1','1')` and `u('1','4')` are both zero by default.

The variable `x` is declared over `(V, V)`, so it contains 16 components, one for each ordered pair of nodes. However, as will be clear from subsequent statements, only `x(i, j)` for `(i,j)` in the set `A` will appear in the equations. The statement

```
alias (V, j);
```

declares `j` as an alias of the set `V`, i.e., an alternative name for `V`. Aliases are useful when a set has to be referred to by more than one name. For example, in the definition of the balance equation,

```
balance(V)..sum(j$A(V, j), x(V, j)) - sum(j$A(j,V), x(j,V)) =e= S(V);
```

`V` and `j` serve very different roles. The set `V` appears in the parentheses after the equation name `balance` before the two dots, and is the "controlling set" or "controlling index" of the equation. GAMS will generate an equation for each element of `N`. In each such equation, `V` is fixed. In contrast, `j` is used as the index of summation in the equation. The expression `sum(j$A(V, j), x(V,j))` in the equation gives the sum of `x(V, j)` over all indices `j` such that `(V, j)` is in the set `A`. Compiling `mcnfp.gms` will generate the following excerpt from `mcnfp.lst`:

```
balance(1)..  x(1,2) + x(1,3) =E= 10 ;
balance(2)..  - x(1,2) + x(2,3) + x(2,4) - x(3,2) =E= 30 ;
balance(3)..  - x(1,3) - x(2,3) + x(3,2) + x(3,4) =E= 10 ;
```

Note that one cannot use the same name of a set as both the controlling set and the index of summation.

In the expression `sum(j$A(V, j), x(V, j))`, a dollar condition is used to control the domain of the summation operator. With the dollar condition the summation is done over all indices `j` for which `(V, j)` belongs to the set `A`. Without that dollar condition, the expression `sum(j, x(V, j))` would give the sum of `x(V, j)` over all `j` for the fixed `V`, which is incorrect. The statement

```
x.up(A) = u(A);
```

specifies `u(i,j)` as the upper bound for `x(i, j)` for each `(i, j)` in `A`.

In summary, we can write the complete GAMS code for this example as follows:

```
*mcnfp.gms
SETS
        V       "nodes" /1*4/,
        A(V, V) "arcs"  /1.2, 1.3, 2.3, 3.2, 2.4, 3.4/;
ALIAS (V, j);
PARAMETERS
        S(V)    "net supplies at nodes"
          /1 10, 2 30, 3 10, 4 -50/
        c(V, V) "unit cost on arcs"  /
          1.2   5
          1.3   1
          2.3   1
          3.2   2
          2.4   10
          3.4   8/,
        u(V, V) "upper bounds" /
          1.2   30
          1.3   100
          2.3   100
          3.2   10
          2.4   100
          3.4   20/;
VARIABLES totalcost, x(V, V);
POSITIVE VARIABLE x;
EQUATIONS balance(V), obj;
    balance(V).. sum(j$A(V, j), x(V, j)) - sum(j$A(j, V), x(j, V)) =E= S(V);
    obj..         totalcost =E= sum(A, c(A)*x(A));
    x.up(A) = u(A);
MODEL MCNFP /ALL/;
SOLVE MCNFP USING LP MINIMIZING totalcost;
```

7.4.2 Integrality of MCNFP optimal solutions

The MCNFP as formulated in (7.3) is a special case of linear programming. Any algorithm for linear programming, including the simplex method introduced earlier in this course, can be directly applied to solve the MCNFP. On the other hand, the special structure of the MCNFP makes it possible to substantially simplify the simplex method. The resulting simplified algorithm is called the **network simplex** method, which is more efficient than the general simplex method when applied

to an MCNFP. The detailed procedure of the network simplex method is beyond the scope of this course. As in the general simplex method, the network simplex method updates a basic feasible solution in each iteration. Instead of maintaining a simplex tableau, the network simplex method updates the basic feasible solution and the reduced costs based on the network structure.

An important feature of the MCNFP is the integrality of its solutions, as stated in the following theorem. The proof of the theorem is omitted due to the scope of this course.

Theorem 7.3. *If an MCNFP has an optimal solution, and all the net supplies and all the flow bounds are integers, then the MCNFP has an integer optimal solution (i.e., an optimal solution with all components being integer).*

It is possible for an MCNFP to have more than one optimal solutions. According to the above theorem, at least one optimal solution is entirely integer. In addition to its computational efficiency, the network simplex method guarantees finding an integer optimal solution for an MCNFP satisfying conditions in the above theorem.

To inform GAMS to use the "network simplex method" to solve a model, one needs to place the following statements with the standard model and solve statements:

```
MODEL MCNFP /ALL/;
OPTION LP = cplex;
$onecho > cplex.opt
lpmethod 3
$offecho
MCNFP.optfile = 1;
SOLVE MCNFP USING LP MINIMIZING totalcost;
```

The word `MCNFP` that appears in the above statements is the name of the model chosen by the modeler. The statement

```
OPTION LP = cplex;
```

is an option statement; it sets CPLEX as the LP solver. The three lines from `$onecho` to `$offecho` creates a file named `cplex.opt` in the GAMS project folder and writes a line "`lpmethod 3`" into the file. The statement

```
MCNFP.optfile = 1;
```

informs the solver to read the option file `cplex.opt`. After reading that file, the solver will set the parameter `lpmethod` to 3. The CPLEX solver is equipped with several alternative algorithms to solve linear programs, including the primal simplex method that we studied in this course, the dual simplex method, the network simplex method, the interior point method (also called the barrier method), and others. Setting `lpmethod` to 3 specifies the network simplex method as the algorithm to be used to solve the model `MCNFP`.

7.4.3 Balanced transportation problems as minimum cost network flow problems

Balanced transportation problems can be treated as a special case of the minimum cost network flow problems (MCNFP). It is natural to consider the supply points and demand points as nodes in the graph, and connect each supply point and each demand point by an arc that points from the supply point to the demand point. For each supply point, let its net supply be its supply capacity. For each demand point, let its net supply be the negative of its demand.

Example 7.9. Consider the transportation problem discussed in the previous lecture.

From/to	City 1	City 2	City 3	City 4	Supply
Plant 1	\$8	\$6	\$10	\$9	35
Plant 2	\$9	\$12	\$13	\$7	50
Plant 3	\$14	\$9	\$16	\$5	40
Demand	45	20	30	30	

To formulate it as an MCNFP, we create two sets

$$V = \{p_1, p_2, p_3, c_1, c_2, c_3, c_4\},$$

$$A = \{(p_i, c_j),\ i = 1,2,3,\ j = 1,2,3,4\},$$

and assign the net supplies of p_1, p_2, p_3 to be $35, 50, 40$, and the net supplies of c_1, c_2, c_3, c_4 to be $-45, -20, -30, -30$ respectively. The unit cost of each arc is the unit shipping cost between the two endpoints of this arc, as given in the table. Since

the problem does not specify bounds on the amount of traffic for any arcs, we let $l_{ij} = 0$ and $u_{ij} = \infty$ for all $(i, j) \in A$.

Note that only a balanced transportation problem can be treated as a special MCNFP. An unbalanced transportation problem needs to be transformed into a balanced problem before being modeled as an MCNFP. Also note that the graph generated by a balanced transportation problem has a special structure: each node in this graph is either a supply point or a demand point, and each arc is from a supply point to a demand point. There are no arcs between two supply points or arcs between two demand points. Such a graph is called a bipartite graph.

7.4.4 Shortest path problems as minimum cost network flow problems

Problem description: A shortest path problem seeks a shortest path from a given node to another given node in a network. In such a problem, a network consists of a set V of nodes, and a set A of (directed) arcs, each of which is an ordered pair of nodes. Each arc (i, j) has a cost $c_{ij} > 0$. A path from a node s to another node y is a sequence of consecutive arcs joining s to y: the first arc in such a path starts from s, each arc in the sequence starts from where the previous arc ends, and the last arc ends with y. The cost of a path is the sum of costs of all arcs in it.

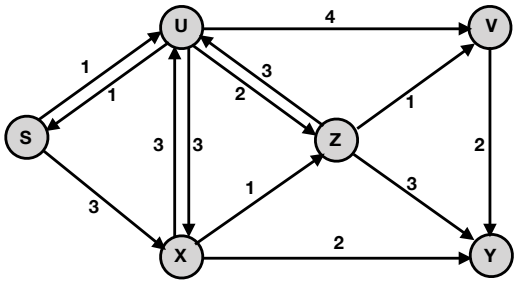

Fig. 7.7 The network for a shortest path problem. This graph has 6 notes: s, u, v, x, y, and z, and 13 arcs. The numbers on arcs are costs (or the lengths of the arcs).

There are specific algorithms for finding shortest paths, such as the Dijkstra algorithm. Details of those algorithms are beyond the scope of this course. Below, we describe how to formulate and solve a shortest path problem as a special MCNFP.

Mathematical model: Suppose we want to find a shortest path from node s to node y. To model the problem as an MCNFP, we specify the net supply of s to be 1, the

net supply of y to be -1, and the net supply to each other node to be 0. We specify the lower bounds and upper bounds on all arcs to be 0 and ∞ respectively. Note that the properties about net supplies and flow bounds are not given in the original network in which a shortest path is to be found. We add such properties to the network arbitrarily in order to model it as an MCNFP. In addition, let the unit cost on each arc (i, j) be the cost c_{ij} as given in the original network.

With the above information we can solve the resulting MCNFP using the network simplex method to find an integer optimal solution, provided that an optimal solution exists. In such an integer optimal solution, the flow on each arc is either 1 or 0, and the arcs with one unit of flow are along a shortest path from s to y. To understand why this is true, first consider the node s. Because the net supply for s is 1, there must exist an arc, say (s,x), that starts with s and carries positive flow. If node x is not y, there must be an arc out of x with positive flow, because the net supply of x is 0. We can repeat this procedure to follow a sequence of consecutive arcs with positive flow, starting from node s. Since the flow has minimum cost, the arcs that carry positive flow do not form any cycle. For this reason, we will never revisit any node and will eventually reach the node y. This gives a path from s to y along which all arcs have positive flow. By the integrality of the flow, the amount of flow on each arc along this path is at least 1. Indeed, the flow on each arc along this path must be exactly 1, since any extra flow will increase the total cost and therefore will not be present in a minimum cost flow. Similarly, the flow on any arc not belonging to this path must be zero, for otherwise we would be able to remove flow from those arcs to further reduce the cost. This path that carries one unit of flow from s to y must be a shortest path from s to y.

A complete GAMS code: We now give a complete GAMS code for this example.

```
*shortestpath.gms
SETS
        V /s,u,x,v,z,y/,
        A(V, V) /s.(u,x), u.(s,x,z,v), x.(u,z,y), z.(u,v,y), v.y/;
ALIAS (V, j);
PARAMETERS
        S(V)    "net supplies for nodes" /s 1, y -1/,
        c(V, V) "unit cost on arcs"
         /s.u   1
          s.x   3
```

```
            u.s   1
            u.x   3
            u.z   2
            u.v   4
            x.u   3
            x.z   1
            x.y   2
            z.u   3
            z.v   1
            z.y   3
            v.y   2/;
  VARIABLES totalcost, x(V, V);
  POSITIVE VARIABLE  x;
  EQUATIONS balance(V), obj;
      balance(V).. sum(j$A(V, j), x(V, j)) - sum(j$A(j, V), x(j, V)) =E= s(V);
      obj..        totalcost =E= sum(A, c(A)*x(A));
  MODEL shortestpath /balance, obj/;
  OPTION LP = cplex;
  $onecho > cplex.opt
      lpmethod 3
  $offecho
  shortestpath.optfile = 1;
  SOLVE shortestpath USING LP MINIMIZING totalcost;
```

7.4.5 The assignment problem

The assignment problem is an important type of problem that arises from many practical contexts. The goal of an assignment problem is to assign members of one group to members of another group to maximize the total "value" of assignments, or to minimize the total cost. Typical examples include assigning employees to jobs, machines to tasks, and so on.

Example 7.10. In this subsection, we present one example to illustrate this type of problems.

Problem description: Machineco needs to assign four jobs to four machines, with each machine handling a different job. The time required to set up each machine for completing each job is shown below. Help Machineco to minimize the total setup time needed to complete the four jobs.

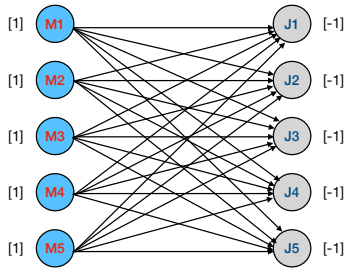

Fig. 7.8 A bipartite graph illustrates an assignment problem: 5 machines are assigned to perform 5 jobs

	Time (Hours)			
Machine	Job 1	Job 2	Job 3	Job 4
1	14	5	8	7
2	2	12	6	5
3	7	8	3	9
4	2	4	6	10

Mathematical model: Binary linear programming and its LP relaxation. Let c_{ij} denote the time to set up machine j to do job i. Since Machineco's problem is to decide which job to assign to each machine, we can use a binary variable x_{ij} to represent the decision on whether to assign job i to machine j. A binary variable is a variable that takes the value either 0 or 1. We use $x_{ij} = 1$ to represent the decision of assigning job i to machine j, and $x_{ij} = 0$ to represent not assigning job i to machine j. With these variables, we formulate the following problem:

$$\begin{cases} \min\limits_{x_{ij}} \sum\limits_{i=1}^{4}\sum\limits_{j=1}^{4} c_{ij}x_{ij} & \\ \text{s.t.} \sum\limits_{j=1}^{4} x_{ij} = 1, \ \ i = 1,2,3,4 & \text{(Each job assigned to a machine)} \\ \sum\limits_{i=1}^{4} x_{ij} = 1, \ \ j = 1,2,3,4 & \text{(Each machine assigned a job)} \\ x_{ij} \in \{0,1\}, \ \ i = 1,2,3,4, \ j = 1,2,3,4 & \end{cases} \tag{7.4}$$

It is important to note that the problem in (7.4) is NOT a linear program, because the constraints that require x_{ij} to be either 0 or 1 are not linear constraints. Indeed, the problem (7.4) is an integer programming problem, and we will introduce techniques for general integer programming problems later. However, as we explain below,

the special structure of (7.4) makes it possible to solve it without using integer programming techniques.

Consider the following linear programming problem:

$$\begin{cases} \min\limits_{x_{ij}} \sum\limits_{i=1}^{4}\sum\limits_{j=1}^{4} c_{ij}x_{ij} \\ \text{s.t.} \sum\limits_{j=1}^{4} x_{ij} = 1, \ \ i = 1,2,3,4 & \text{(Each job assigned to a machine)} \\ \quad \sum\limits_{i=1}^{4} x_{ij} = 1, \ \ j = 1,2,3,4 & \text{(Each machine assigned a job)} \\ \quad 0 \le x_{ij} \le 1, \ \ i = 1,2,3,4, \ j = 1,2,3,4 & \text{(A relaxation of the binary constraint).} \end{cases} \tag{7.5}$$

The difference between (7.4) and (7.5) is about their last group of constraints. The constraint $x_{ij} \in \{0,1\}$ for each (i,j) pair can be equivalently stated as $0 \le x_{ij} \le 1$ and x_{ij} is integer. If we can find an integer optimal solution to (7.5), then that solution will be an optimal solution to (7.4). In general, a linear program with an optimal solution does not necessarily have an integer optimal solution. However, as stated in Theorem 7.3, an MCNFP with integer data always has an integer optimal solution provided it has an optimal solution.

Reformulation as an MCNFP: Next, we observe that the problem in (7.5) is a special MCNFP. To this end, consider a graph with 8 nodes, with 4 nodes representing jobs and the other 4 representing machines. Connect each job node with each machine node by an arc that points from the job to the machine, and let each job node have net supply 1 and each machine node have net supply -1. Let c_{ij} be the unit cost on the arc (i,j) that points from job i to machine j. Finally, let the lower bounds and upper bounds for flow on each arc to be 0 and 1, respectively. The MCNFP associated with such a graph can be written as

$$\begin{cases} \min\limits_{x_{ij}} \sum\limits_{i=1}^{4}\sum\limits_{j=1}^{4} c_{ij}x_{ij} \\ \text{s.t.} \sum\limits_{j=1}^{4} x_{ij} = 1, \ \ i = 1,2,3,4 & \text{(Balance equation for job } i\text{)} \\ \quad -\sum\limits_{i=1}^{4} x_{ij} = -1, \ \ j = 1,2,3,4 & \text{(Balance equation for machine } j\text{)} \\ \quad 0 \le x_{ij} \le 1, \ \ i = 1,2,3,4, \ j = 1,2,3,4. \end{cases} \tag{7.6}$$

The MCNFP (7.6) is clearly equivalent to (7.5). Accordingly, one can solve (7.5) using the network simplex method to find an integer optimal solution, which will be an optimal solution to the integer program (7.4).

A complete GAMS model: The model `assignment` formulated in assignment.gms consists of all constraints and the objective function of (7.5), except the constraints $x_{ij} \leq 1$ for each (i, j). It is fine to omit those constraints, because the balance equations $\sum_{j=1}^{4} x_{ij} = 1$ for each i and the non-negativity constraints on x_{ij} automatically imply $x_{ij} \leq 1$. With the statement `lpmethod 3` in the option file, the model is solved by the CPLEX solver with the network simplex method, and an integer optimal solution is found by the solver.

```
*assignment.gms
SETS
        i  "jobs"     / job1*job4 /,
        j  "machines" / m1*m4 /;
TABLE
        c(i,j)
                       m1       m2       m3      m4
          job1         14        2       7       2
          job2         5        12       8       4
          job3         8         6       3       6
          job4         7         5       9       10;
VARIABLES
        x(i,j), z;
POSITIVE VARIABLE x;
EQUATIONS
        cost, jobEq(i), machine(j);
    cost ..       SUM((i, j), c(i, j)*x(i, j)) =E= z;
    jobEq(i) ..   SUM(j, x(i, j))              =E= 1;
    machine(j) .. SUM(i, x(i, j))              =E= 1;
MODEL assignment /ALL/ ;
OPTIONS LP = cplex;
$onecho > cplex.opt
   lpmethod 3
$offecho
assignment.optfile = 1;
SOLVE assignment USING LP MINIMIZING z;
```

7.5 Exercises

Exercise 7.1. A company supplies goods to three customers, who require 40, 50, and 40 units respectively. The company has three warehouses, each of which has 30 units available. The costs of shipping 1 unit from each warehouse to each customer are shown in the table below.

	To		
From	Customer 1	Customer 2	Customer 3
Warehouse 1	\$15	\$35	\$25
Warehouse 2	\$10	\$50	\$40
Warehouse 3	\$20	\$40	\$30

There is a penalty for unmet demand: With customer 1, a penalty cost of \$70 per unit is incurred; with customer 2, \$75 per unit; and with customer 3, \$65 per unit. The company needs to minimize the total cost.

1. Does the following formulation correctly model the problem? Why? (Here, x_{ij} denotes the amount to ship from warehouse i to customer j, and c_{ij} is the corresponding unit shipping cost.)

$$\min \sum_{i=1}^{3}\sum_{j=1}^{3} c_{ij}x_{ij} + 70(40 - \sum_{i=1}^{3} x_{i1}) + 75(50 - \sum_{i=1}^{3} x_{i2}) + 65(40 - \sum_{i=1}^{3} x_{i3})$$

$$\text{s.t. } \sum_{j=1}^{3} x_{ij} \le 30, \ \ i = 1, \cdots, 3 \text{ (supply constraints)}$$

$$x_{ij} \ge 0, \ \ i = 1, \cdots, 3, j = 1, \cdots, 3$$

2. Formulate the problem as a balanced transportation problem. Create a GAMS file named *company.gms* to solve the problem. (The optimal value is \$4950.)

Exercise 7.2. Given the following two balanced transportation problems:

From/to	D_1	D_2	D_3	Supply
S_1	\$3	\$3	\$4	30
S_2	\$2	\$2	\$2	60
S_3	\$5	\$5	\$3	10
Demand	25	35	40	100

From/to	D_1	D_2	D_3	D_4	Supply
S_1	\$4	\$3	\$3	\$4	20
S_2	\$2	\$4	\$5	\$2	30
S_3	\$4	\$2	\$1	\$6	50
Demand	25	25	15	35	100

Carry out the following tasks:

(a) Use the north-west corner method to find a BFS for each problem. Is that BFS degenerated?

(b) Use the least-cost method to find a BFS for each problem. Is that BFS degenerated?

(c) Apply the simplex method for transportation problems to solve these problems.

Exercise 7.3. Each year, Data Corporal produces 400 computers in Boston and 300 computers in Raleigh. Los Angeles customers must receive 400 computers, and Austin customers must receive 300 computers. Data Corporal can ship computers from each production city to each demand city directly, or through Chicago. The costs of sending a computer among pairs of cities are shown below.

	To		
From	Chicago	Austin	Los Angeles
Boston	$80	$220	$280
Raleigh	$100	$140	$170
Chicago	-	$40	$50

1. Suppose that no more than 200 computers can be shipped between each pair of cities. Formulate an MCNFP to minimize the total shipping cost, and provide a graph to include all information about the MCNFP. Create a GAMS file named *datacorporal.gms* to solve the problem. (The optimal value is $118000.)
2. Suppose now the total amount of computers shipped through Chicago cannot exceed 100 (that is, the total amount of computers entering Chicago cannot exceed 100). How do you formulate the problem as an MCNFP? Note that an MCNFP is only allowed to have two types of constraints: bound constraints on arcs, and balance equations for nodes. If you add a constraint that does not belong to one of these two types, then the new model is not a MCNFP. (The optimal value is $133000.)

For part 2, do not hand in your GAMS code. Just explain how to modify the model (or the graph), and interpret the optimal solution and optimal value you find.

Exercise 7.4. Oilco has oil fields in San Diego and Los Angeles. The San Diego field can produce 500,000 barrels per day, and the Los Angeles field can produce 400,000 barrels per day.

Oil is sent from the fields to a refinery, in either Dallas or Houston (assume each refinery has unlimited capacity). To refine 100,000 barrels costs $700 at Dallas and $900 at Houston.

Refined oil is shipped to customers in Chicago and New York. Chicago customers require 400,000 barrels per day, and New York customers require 300,000 barrels per day.

The costs of shipping 100,000 barrels of oil (refined or unrefined) between cities are shown below.

	To($)			
From	Dallas	Houston	New York	Chicago
Los Angels	300	110	-	-
San Diego	420	100	-	-
Dallas	-	-	450	550
Houston	-	-	470	530

Formulate an MCNFP to minimize the total cost of meeting all demands; provide a graph to include all information about the MCNFP. Create a GAMS file *oilco.gms* to solve the problem (the optimal value is $10,470).

Exercise 7.5. Consider a transportation problem where the input data is given by the following table:

From/To	Demand point 1	Demand point 2	Demand point 3	Demand point 4	Supply
Supply point 1	2	3	5	6	5
Supply point 2	2	1	3	5	10
Supply point 3	3	8	4	5	15
Demand	12	8	4	6	30

Here, we have three supply points with the supply capacities are given in the last column, and four demand points with the demand values are given in the last row. The value in the cell (i, j) provides the unit cost to ship from the supply point i to the demand point j.

(a) Code this problem in GAMS. Then, do not submit the code in Sakai, but run the code and present the result. Interpret the solution in the context of this transportation problem.

(b) Using one of the methods we described in class to find a basic solution for this problem (e.g., the northwest conner rule, or the minimum cost rule).

(c) Due to the lack of warehouse, we have to transfer some supply value Δ from Supply point 3 to Supply point 1 so that they become $15-\Delta$ and $5+\Delta$, respectively. Use the sensitivity analysis theory of LPs to compute the value of Δ so that we do not need to change the transportation route obtained in Question (a) (That is, it does not change the basis of this optimal solution).

(d) Now, assume that the unit cost given in the cells of the above table is the price which a transportation company will charge when it does transportation services. Modify the GAMS code in Question (a) to solve the problem of maximizing the total revenue this company makes. Interpret the resulting optimal solution in this case.

Exercise 7.6. A company produces cars in Atlanta, Boston, Chicago and Los Angeles. The cars are then shipped to warehouses in Memphis, Milwaukee, New York City, Denver and San Francisco. The number of cars available at each plant is given in the last column, and the capacity of each warehouse is given in the last row of the following table. In addition, each cell of this table presents a distance (in miles) between these cities.

	Memphis	Milwaukee	New York	Denver	San Francisco	Plants
Atlanta	371	761	841	1398	2496	5000
Boston	1296	1050	206	1949	3095	6000
Chicago	530	87	802	996	2142	4000
Los Angeles	1817	2012	2786	1059	379	3000
Warehouses	6000	4000	4000	2000	2000	18000

(a) Assume that the cost (in dollars) of shipping a car from plants to warehouses is given by the function $c(x)=\sqrt{x}$, where x is the distance in miles between two cities. Formulate this problem as a transportation problem. Then, model it in GAMS. Do not submit your code on Sakai, but run your code and interpret your result in detail.

(b) Assume that due to long distance, the company does not want to ship cars through two routes: from Boston to San Francisco, and from Los Angeles to New York. Formulate this problem as a minimum cost network flow problem with the cost function as in Question (a). Then, model it in GAMS. Do not

submit your code in Sakai, but run your code and interpret your result in detail. Compare this result with the one in Question (a).

Exercise 7.7. The data given in the following table are distances of arcs between pairs of nodes i and j in a network, where $i, j = 1, \cdots, 11$. An empty cell indicates that there is no arc between that pair.

	To										
From	1	2	3	4	5	6	7	8	9	10	11
1		5	2	4							
2					3	4					
3				3	4	8					
4						6	3				
5						2		9			10
6								12	9		
7									9	11	
8									6		10
9											4
10											7

(a) Formulate an MCNFP, whose solution can be used to find the shortest path from node 1 to node 11.

(b) Code this MCNFP in GAMS. Do not submit the code in Sakai, but run your code and interpret your result in detail.

(c) Assume that for each route on arc (i, j), we need to pay a cost $c(x) = \sqrt{x}$, where x is the length of this arc (the distance between node i and node j). Reformulate this problem as an MCNFP, and code it in GAMS. Do not submit the code in Sakai, but run your code and interpret your result in detail. Compare this result with the one in Question (b).

References

1. A. Ben-Tal, T. Margalit, and A. Nemirovski. The ordered subsets mirror descent optimization method with applications to tomography. *SIAM J. Optim.*, 12:79–108, 2001.
2. Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.
3. Robert E Bixby. Solving real-world linear programs: A decade and more of progress. *Operations research*, 50(1):3–15, 2002.

4. S. Boyd and L. Vandenberghe. *Convex Optimization*. University Press, Cambridge, 2004.
5. G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.
6. R. Fletcher. *Practical Methods of Optimization*. Wiley, Chichester, 2nd edition, 1987.
7. M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.
8. S Hillier Frederick and J Lieberman Gerald. Introduction to operations research, 2005.
9. George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.
10. R.J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, 2015.