Lecture Notes on Operations Research 1

Shixuan Zhang

ISEN 320, Fall 2024

Contents

1 Basics of Mathematical Optimization

A mathematical optimization problem has either of the following forms:

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in X \end{array} \quad \text{or} \quad \begin{array}{ll} \max & f(x) \\ \text{s.t.} & x \in X. \end{array} \tag{1.1}$$

Here, X is a set of variables x, while f is a function defined on X such that we can compare its values (partially ordered). Let $\mathbb{R}$ denote the real numbers. In this course, and in many *real*-world problems, the set X is a subset of some n-dimensional real vector space $X \subseteq \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ takes value in real numbers. We call the set X the *feasible region* of the problem, the variables x the *decision variables*, and the function f the *objective function*. Any $x \in \mathbb{R}^n$ is *feasible* if $x \in X$ and *infeasible* otherwise. When $X = \varnothing$, we say that the problem is *infeasible*.

The minimum or maximum value of the objective function is called the *optimal value* of the optimization problem, if it exists (Example 1.1). Certain values of the decision variables $x^* \in \mathbb{R}^n$ are called *an optimal solution* if $f(x^*) = \min_{x \in X} f(x)$ or $f(x^*) = \max_{x \in X} f(x)$, and denoted as $x^* \in \arg\min_{x \in X} f(x)$ or $x^* \in \arg\max_{x \in X} f(x)$. Despite some notational difference, we do not really need to develop different theories for the two forms because we can transform $\max_{x \in X} f(x)$ into $\min_{x \in X} -f(x)$, where only the sign of the optimal value is changed and the set of optimal solutions remains unchanged. For now, we only discuss minimization problems for notational convenience.

An optimization problem does not necessarily have any optimal value or optimal solutions. When it does, it may not have a unique optimal solution. Thus to be rigorous, one would only say "the" optimal solution when it exists and is known to be unique.

Example 1.1.
- *Let $X = \mathbb{R}$ and $f(x) = x$. For any $x \in \mathbb{R}$, there is a real number $a < x$. Therefore, f does not have a minimum on X. This also implies that the optimization problems $\min_{x \in X} f(x)$ do not have optimal solutions.*

- *Let* $X = \{x \in \mathbb{R} : 0 < x < 1\} \subseteq \mathbb{R}$ *and* $f(x) = x$*. For any* $x \in X$*, notice that* $a := x/2 \in X$*. Clearly* $f(a) < f(x)$ *so* f *does not have a minimum on* X*. This also implies that the optimization problems* $\min_{x \in X} f(x)$ *do not have optimal solutions.*
- *Let* $X = \{x \in \mathbb{R} : x \geq 1\} \subseteq \mathbb{R}$ *and* $f(x) = 1/x$*. Notice that* $f(x) > 0$ *for any* $x \in X$*, and for any* $a > 0$*, we can find* $x = 1 + 1/a \in X$ *such that*

$$f(x) = \frac{1}{x} = \frac{a}{a+1} < a.$$

  In plain words, no matter how small a positive number a is, we can always find a decision variable x *such that* $f(x) < a$*. Therefore, the optimization problem* $\min_{x \in X} f(x)$ *does not have optimal solutions.*
- *Let* $X = \mathbb{R}$ *and* $f(x) = 0$ *(i.e., a constant function). Any number* $x \in \mathbb{R}$ *is an optimal solution to both the minimization problem, and therefore, the optimization problem does not have a unique solution.*

We say a minimization problem is *bounded* if we can find a real number $a \in \mathbb{R}$ such that $f(x) > a$ for all feasible decision variables $x \in X$, and *unbounded* otherwise. When the optimal value is not guaranteed to exist, some people write inf instead of min to denote the largest such lower bound, and use the convention that the optimal value of an unbounded minimization problem is $-\infty$. Similarly, the smallest upper bound of a maximization problem is sometimes denoted as sup and the optimal value of an unbounded maximization problem is $+\infty$. With this convention, an infeasible minimization (resp. maximization) problem has its optimal value $+\infty$ (resp. $-\infty$). Please note that the infinity notation is not a real number and should be treated with care. Whenever a problem is unbounded or infeasible, there is no optimal solution, $\arg\min_{x \in X} f(x) = \varnothing$.

When the optimization problem is bounded, the feasible region is *closed*, and the objective function is *continuous*, then the existence of the optimal value and optimal solutions is guaranteed. We do not define continuous functions in this class, but the common elementary functions (e.g. linear, polynomial, rational power, exponential, trigonometric, and their sums, products, inverses, compositions) are all continuous functions inside their domains. Using these functions in *nonstrict* inequalities would automatically define a closed feasible region. Requiring some of the variables to be integers also leads to a closed feasible region.

In this course, we will mostly consider the case where the feasible region X consists of discrete or continuous values, and is defined functionally by a finite number of equalities and inequalities. That is, given an index $n' \leq n$ and functions $g_1, \ldots, g_m : \mathbb{R}^n \to \mathbb{R}$,

$$X := \{x \in \mathbb{Z}^{n'} \times \mathbb{R}^{n-n'} : g_i(x) \leq 0, i = 1, \ldots, m', \, g_j(x) = 0, j = m'+1, \ldots, m\}. \quad (1.2)$$

Each of the equalities or inequalities is called a *(functional) constraint* on our decision variables x. We are using the convention $g_i(x) \leq 0$ for $i = 1, \ldots, m'$ because any inequality $g'(x) \geq 0$ can be equivalently rewritten as $-g'(x) \leq 0$, and only considering nonstrict inequalities out of the concern of optimal solution existence, as discussed above. The optimization problem (1.1) with a feasible region (1.2) can be more directly written as

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m', \\ & g_j(x) = 0, \quad j = m'+1, \ldots, m, \\ & x \in \mathbb{Z}^{n'} \times \mathbb{R}^{n-n'} \end{aligned} \tag{1.3}$$

without the need to explicitly specify the set X. With any functional constraint (1.3), i.e., $m > 0$, the problem is called *constrained* optimization, and *unconstrained* otherwise. We say that the problem (1.3) is *linear* if $f, g_1, \ldots, g_m$ are all affine linear functions, and *nonlinear* otherwise. When all of the variables must take integer values, i.e., $n' = n$, we say that the problem (1.3) is *discrete* or *integer* optimization; if all of the variables can take continuous values, i.e., $n' = 0$, then the problem is often called *continuous* optimization; and in the case $0 < n' < n$ we say that the problem is *mixed-integer* optimization.

For constrained optimization modeling, we can follow the procedure below.

(i) Describe the relevant data of the problem.
(ii) Identify and describe the decision variables.
(iii) Describe the sign, bounds, and type restrictions on individual variables.
(iv) Write the constraints in terms of the decision variables.
- If any additional variable is needed, go back to Step (ii).

(v) Write the objective function in terms of the decision variables.
- If any additional variable is needed, go back to Step (ii).

Example 1.2. *One wants to design a aluminum can in the shape of a cylinder with height h and radius r with the minimum usage of aluminum (Figure 1.1), such that the following requirements are satisfied.*

- *The height h must be at least three times as large as the radius r.*
- *The height h can be at most four times as large as the radius r.*
- *The volume needs to be at least V.*

To formulate an optimization problem, let $(r, h) \in \mathbb{R}^2$ be our decision variables. Assuming the aluminum sheet has a fixed thickness, the amount of aluminum used is determined by the surface area $f(r, h) = 2\pi r + 2\pi rh$, which will be our objective function. For the first requirement, we can write it as a constraint

$$h \geq 3r \iff -h + 3r \leq 0.$$

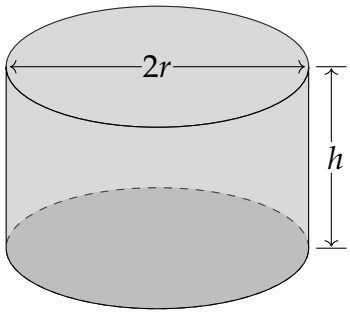


Figure 1.1: A cylindrical can of height h and radius r

Similarly, for the second requirement, we can write it as

$$h \leq 4r \iff h - 4r \leq 0.$$

The volume of this cylindrical can is $\pi r^2 h$, so with the given parameter V, the last requirement can be written as

$$V \leq \pi r^2 h \iff V - \pi r^2 h \leq 0.$$

While the variables r and h must of course be nonnegative, we do not need to add bounds $r \geq 0$ and $h \geq 0$ because the first two constraints imply that $4r \geq h \geq 3r$, which guarantees $r \geq 0$ and hence also $h \geq 3r \geq 0$. To summarize, we have formulated the following optimization problem.

$$\begin{aligned} \min \quad & 2\pi r^2 + 2\pi rh \\ \text{s. t.} \quad & -h + 3r \leq 0, \\ & h - 4r \leq 0, \\ & V - \pi r^2 h \leq 0, \\ & r, h \in \mathbb{R}. \end{aligned}$$

This is an example of continuous *and* nonlinear *optimization problem.*

2 Linear Optimization I: Simple Models

2.1 Linear Optimization Formulations

A linear optimization (LO) model is a constrained optimization model with continuous variables, affine linear constraints, and a linear objective function. Recall that a function $f : \mathbb{R}^n \to \mathbb{R}$ is called affine linear if there exist $f_0, f_1, \ldots, f_n \in \mathbb{R}$ such that

$$f(x) = f_0 + f_1 x_1 + \cdots + f_n x_n,$$

for any $(x_1, \ldots, x_n) \in \mathbb{R}^n$. It is further *linear* if $f_0 = 0$. Thus given problem data $c_1, \ldots, c_n, b_1, \ldots, b_m \in \mathbb{R}$, and $a_{ij} \in \mathbb{R}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, a simple form of linear optimization can be formulated by

$$\begin{array}{rrcccl} \min \,/\, \max & c_1 x_1 & + & \cdots & + c_n x_n & \\ \text{s.t.} & a_{11} x_1 & + & \cdots & + a_{1n} x_n & \le b_1 \\ & \vdots & & \ddots & \vdots & \vdots \\ & a_{m1} x_1 & + & \cdots & + a_{mn} x_n & \le b_m \\ & x_1, & & \cdots & x_n & \in \mathbb{R} \end{array} \tag{2.1}$$

or simply as

$$\begin{aligned} \min \,/\, \max \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ji} x_i \le b_j, \quad j = 1, \ldots, m, \\ & x_i \in \mathbb{R}, \quad i = 1, \ldots, n. \end{aligned}$$

An alternative way is to use a matrix and vectors

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}, \quad c = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}, \tag{2.2}$$

and write our LO model more compactly as

$$\begin{aligned} \min \,/\, \max \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax \le b, \\ & x \in \mathbb{R}^n. \end{aligned} \tag{2.3}$$

Here, the convention is that for any two vectors $u = (u_1, \ldots, u_n), v = (v_1, \ldots, v_n) \in \mathbb{R}^n$, we write

$$u \le v \iff u_i \le v_i, \quad i = 1, \ldots, n.$$

We claim that any linear optimization problem in general can be transformed into the above simple form (2.1) or (2.2). We have already seen that an inequality constraint $Ax \ge b$ can be rewritten as $-Ax \le -b$. When we have equality constraints, we can always reformulate them as

$$A^{\mathsf{eq}} x = b^{\mathsf{eq}} \iff \begin{bmatrix} A^{\mathsf{eq}} \\ -A^{\mathsf{eq}} \end{bmatrix} x \le \begin{bmatrix} b^{\mathsf{eq}} \\ -b^{\mathsf{eq}} \end{bmatrix}. \tag{2.4}$$

When we have explicit bounds on variables, for example,

$$x_i^{\mathtt{lb}} \leq x_i \leq x_i^{\mathtt{ub}}, \quad i = 1, \ldots, n, \tag{2.5}$$

we can also write them as inequality constraints, with $e_i \in \mathbb{R}^n$ that has 1 in its ith component and 0 in all other components,

$$\begin{bmatrix} -e_i^\mathsf{T} \\ e_i^\mathsf{T} \end{bmatrix} x \leq \begin{bmatrix} -x_i^{\mathtt{lb}} \\ x_i^{\mathtt{ub}} \end{bmatrix}. \tag{2.6}$$

Therefore, the formulation (2.1) or (2.2) is a conceptually simple way of writing *any* LO model. In practice, however, it is not always necessary that we convert a LO model into such formulation.

Example 2.1. *A farmer wants to determine how many acres of corn and wheat to plant this year. Related information is given as follows.*

- *An acre of wheat yields 25 bushels of wheat and requires 10 hours of labor per week.*
- *An acre of corn yields 10 bushels of corn and requires 4 hours of labor per week.*
- *All wheat and corn can be sold at \$4 a bushel.*
- *Seven acres of land and 40 hours of labor per week are available.*
- *Government regulations require that at least 30 bushels of corn be produced.*

The goal is to maximize the total revenue. To formulate it as a LO model, let x_1 denote the number of acres of corn to plant and x_2 the number of acres of wheat to plant this year, both of which are continuous and nonnegative

$$x_1, x_2 \geq 0 \iff -x_1 \leq 0, -x_2 \leq 0.$$

Their upper bounds are not explicitly given to us. For the first constraint, we notice that our land area is limited

$$x_1 + x_2 \leq 7.$$

We write the labor time restriction as our second constraint

$$4 \cdot x_1 + 10 \cdot x_2 \leq 40.$$

The government regulations on corn production gives us the third constraint

$$10 \cdot x_1 \geq 30 \iff -x_1 \leq -3.$$

Note that the variable bound $-x_1 \leq 0$ is implied by this constraint and can thus be discarded.

The objective is to maximize the revenue, which is

$$\max \quad 4 \cdot 10 \cdot x_1 + 4 \cdot 25 \cdot x_2.$$

This LO model can also be written in the matrix form (2.2) with

$$A = \begin{bmatrix} 1 & 1 \\ 4 & 10 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 7 \\ 40 \\ -3 \\ 0 \end{bmatrix}, \quad c = \begin{bmatrix} 40 \\ 100 \end{bmatrix}.$$

2.2 Computer Modeling Tools and Solvers

We have seen two mathematically equivalent ways of writing a LO model. However, depending on the input problem data, sometimes it is more convenient to use the first way (2.1), rather than the second one (2.2), in terms of computer programming. We will mainly use `Python 3.11` for our computer programming. There are two types of computer tools used for optimization problems:

- modeling interface, which facilitates model building by providing variable and constraint handles, and sometimes also a domain-specific language for constraints and objective expressions;
- underlying solver, which takes the problem data and executes appropriate numerical algorithms that aim to find optimal solutions.

For this course, we use the following two `Python` packages: `OR-Tools 9.5` and `SciPy 1.11`. The installation guides can be found on `https://developers.google.com/optimization/install/python` and `https://scipy.org/install/`, respectively. The package `OR-Tools` provides a full modeling interface to LO and mixed-integer linear optimization (MILO) problems. Its installation automatically includes some open-source solvers, such as `GLOP` and `PDLP`, and it also connects to proprietary solvers such as `Gurobi` and `CPLEX`. The package `SciPy` is a popular scientific computing `Python` package, which provides a thin wrapper of a powerful open-source LO and MILO solver `HiGHS`. Next we briefly describe and compare ways of building and solving LO models using `OR-Tools` and `SciPy`.

We illustrate the modeling using Example 2.1 and begin with `OR-Tools`, which is often the more convenient one to use, especially for beginners. The module can be loaded as follows.

```
from ortools.linear_solver import pywraplp
```

Next we declare a LO solver, `GLOP`.

```
solver = pywraplp.Solver.CreateSolver("GLOP")
```

To define continuous variables, we can use the `NumVar` function in the `Solver` class.

```
x1 = solver.NumVar(0.0, solver.infinity(), "x1")
x2 = solver.NumVar(0.0, solver.infinity(), "x2")
```

The first two arguments are the lower and upper bounds of the defined variable (as defind in (2.5)). Here, we set the lower bounds to be 0.0, and because we do not know any upper bound of the variables, we put `solver.infinite()` (i.e., $+\infty$) as the second argument. The third argument, `"x1"` or `"x2"`, is the variable name, which can be of help if we want to debug or export the model later. The constraints in Example 2.1 can be coded as follows.

```
solver.Add(x1 + x2 <= 7)
solver.Add(4*x1 + 10*x2 <= 40)
solver.Add(-x1 <= -3)
```

To set the objective function, we can use the following code.

```
solver.Maximize(40*x1 + 100*x2)
```

Now we are ready to let the solver solve this model.

```
status = solver.Solve()
```

The `status` stores the information returned by the solver, which can be used to check whether we have found an optimal solution.

```
if status == pywraplp.Solver.OPTIMAL:
    print("Optimal value =", solver.Objective().Value())
    print("x1 =", x1.solution_value())
    print("x2 =", x2.solution_value())
else:
    print("The solver is unable to find an optimal solution.")
```

After executing the script, an output is displayed below.

```
Optimal value = 399.99999999999994
x1 = 3.0
x2 = 2.7999999999999994
```

As `OR-Tools` connects to different solvers, we can replace the definition of `solver` in the `OR-Tools` model with the following line.

```
solver = pywraplp.Solver.CreateSolver("Clp")
```

Without any changes to other parts, the output could become the following.

```
Optimal value = 400.0
x1 = 5.0
x2 = 2.0
```

This shows us that calling different underlying solvers may give different optimal solutions, even when the model is unchanged.

Given the problem data matrix and vectors, we can also use `SciPy` to solve this LO model. The modules can be loaded using the following code.

```
import numpy as np
from scipy.optimize import linprog
```

The problem data can be coded as follows.

```
c = np.array([40, 100])
b = np.array([7, 40, -3, 0])
A = np.array([[1, 1],
              [4, 10],
              [-1, 0],
              [0, -1]])
```

Note that `SciPy` by default only takes minimization problems, so we call the `linprog` function with arguments `-c`, `b`, and `A`, to solve the LO model.

```
result = linprog(-c, A_ub=A, b_ub=b)
```

Here, keywords `A_ub` and `b_ub` refer to the inequality constraints in (2.2). When the problem has equality constraints (as in (2.4)) and variable bounds (as in (2.5)), they can be directly added using keywords `A_eq`, `b_eq`, and `bounds`. We now retrieve the results (with the opposite of the optimal value).

```
print(result.message)
print("Optimal value =", -result.fun)
print("[x1 x2] =", result.x)
```

The output is displayed below.

```
Optimization terminated successfully. (HiGHS Status 7: Optimal)
Optimal value = 400.0
[x1 x2] = [3.  2.8]
```

To summarize, compared with `SciPy`, the package `OR-Tools` allows

(i) adding constraints by directly using variable handles, without the need to write down the matrix representation;

(ii) both minimization and maximization;

(iii) an easy change of the underlying solver.

However, `SciPy` can be more lightweight sometimes and connects to the solver `HiGHS`. One should choose their computer tools depending on the problem and the purpose of use. For this course, we will mostly rely on `OR-Tools` for simplicity.

2.3 More Examples and Models

Example 2.2. *A student aims to improve their diet. Based on a nutrition specialist recommendation, they want their daily intake to contain at least 60 g of protein, 800 mg of calcium, 75 mg of vitamin C, and 2,000 calories. They would like to find a least expensive menu consisting of five food types: almond butter, brown rice, orange juice, salmon, and wheat bread. The serving size, cost per serving, and nutrition information for each food type is provided in the table below.*

Food type	Cost ($)	Protein (g)	Calcium (mg)	Vitamin C (mg)	Calories
Almond butter (100 g)	2.90	15	270	1	600
Brown rice (200 g)	3.20	5	20	0	215
Orange juice (250 g)	0.50	2	25	106	110
Salmon (150 g)	4.50	39	23	0	280
Wheat bread (25 g)	0.30	3	35	0	66
Required ingestion	-	60	800	75	2,000

We define decision variables for the amount of each food type to be consumed daily, all of which are nonnegative:

$x_1 \geq 0$: *servings of almond butter consumed daily,*
$x_2 \geq 0$: *servings of brown rice consumed daily,*
$x_3 \geq 0$: *servings of orange juice consumed daily,*
$x_4 \geq 0$: *servings of salmon consumed daily,*
$x_5 \geq 0$: *servings of wheat bread consumed daily.*

The constraints express the minimum daily requirements for protein

$$15x_1 + 4x_2 + 2x_3 + 39x_4 + 3x_5 \geq 60,$$

for calcium

$$270x_1 + 20x_2 + 25x_3 + 23x_4 + 35x_5 \geq 800,$$

for vitamin C

$$x_1 + 106x_3 \geq 75,$$

and for calories

$$600x_1 + 215x_2 + 110x_3 + 280x_4 + 66x_5 \geq 2000.$$

The objective is to minimize the cost, which is a linear function of the decision variables:

$$\min \quad 2.9x_1 + 3.2x_2 + 0.5x_3 + 4.5x_4 + 0.3x_5.$$

We program the model in the script `model_diet.py`*, and get the following output.*

```
The cost of a least expensive diet is $9.09.
The intake amount of each food type in the diet is shown below.
   Almond butter: 0.00 g
      Brown rice: 0.00 g
    Orange juice: 176.89 g
          Salmon: 0.00 g
     Wheat bread: 728.09 g
```

Example 2.3. *An investor is considering 6 projects for potential investment for the upcoming year. The required investment and end-of-year payout amounts are described in the following table. Partial investment (i.e., financing only a fraction of the project instead of the whole*

	Project					
	1	2	3	4	5	6
Investment ($·1000)	10	25	35	45	50	60
Payout ($·1000)	12	30	41	55	65	77

project) is allowed for each project, with the payout proportional to the investment amount. For example, if the investor decides to invest $5,000 in project 2, the corresponding payout will be $30,000·($5,000/$25,000)=$6,000. There are $100,000 available for investment.

We define variables

$$0 \leq x_i \leq 1\text{: fraction of project } i \text{ financed, for } i = 1, \ldots, 6.$$

The only constraint is the limit on the investment, which is

$$10x_1 + 25x_2 + 35x_3 + 45x_4 + 50x_5 + 60x_6 \leq 100.$$

The objective is to maximize the total payout:

$$\max \quad 12x_1 + 30x_2 + 41x_3 + 55x_4 + 65x_5 + 77x_6.$$

We program the model in the script `model_allocation.py`*, and get the following output.*

```
The maximum payout is $129166.67.
The investment on each project is shown below.
Project 0: $0.00
Project 1: $0.00
Project 2: $0.00
Project 3: $0.00
Project 4: $50000.00
Project 5: $50000.00
```

Some models are not immediately LO models, that is, there could be nonlinear constraints or integer variables, but they can be *reformulated* or *relaxed* as LO models.

Example 2.4. *A painter needs to complete a job that requires 50 gallons of brown paint and 50 gallons of gray paint. The required shades of brown and gray can be obtained my mixing the primary colors (red, yellow, and blue) in the proportions given in the following table. The same*

Color	Red	Yellow	Blue
Brown	40%	30%	30%
Gray	30%	30%	40%

shades can be obtained by mixing secondary colors (orange, green, and purple), each of which is based on mixing two out of three primary colors in equal proportions (red/yellow for orange, yellow/blue for green, and red/blue for purple). The painter currently has 20 gallons each of red, yellow, and blue paint, and 10 gallons each of orange, green, and purple paint. If needed, they can purchase any of the primary color paints for $20 per gallon, however they would like to save by utilizing the existing paint supplies as much as possible.

We use indices $i = 1, \ldots, 6$, for red, yellow, blue, orange, green, and purple colors, respectively, and indices $j = 1, 2$, for brown and gray colors, respectively. Our decision variables can be defined as

$$x_{ij} \geq 0\text{: gallons of paint of color } i \text{ used to obtain color } j \text{ paint,}$$

for $i = 1, \ldots, 6$, $j = 1, 2$, and

$$y_i \geq 0\text{: gallons of paint of color } i \text{ purchased, } i = 1, 2, 3.$$

The total amount of brown and gray paint made must be at least 50 gallons each:

$$\sum_{i=1}^{6} x_{ij} \geq 50, \quad j = 1, 2.$$

The amount of paint used should not exceed its availability

$$\begin{aligned} x_{i1} + x_{i2} - y_i &\leq 20, \quad i = 1, 2, 3, \\ x_{i1} + x_{i2} &\leq 10, \quad i = 4, 5, 6. \end{aligned}$$

To express the constraints ensuring that the mixing yields the right shade of brown, note that only three out of six colors used for mixing contain red, and the total amount of red paint (including that coming from orange and purple paints) used in the brown mix is

$$x_{11} + 0.5x_{41} + 0.5x_{61}.$$

Hence, a constraint for the proportion of red color in the brown mix can be written as follows:

$$\frac{x_{11} + 0.5x_{41} + 0.5x_{61}}{\sum_{i=1}^{6} x_{i1}} = 0.4.$$

We can multiply both sides by $\sum_{i=1}^{6} x_{i1}$ *and* reformulate *it as a linear constraint*

$$0.6x_{11} - 0.4x_{21} - 0.4x_{31} + 0.1x_{41} - 0.4x_{51} + 0.1x_{61} = 0.$$

Similarly, the proportion of yellow and blue colors in the brown mix is given by:

$$\frac{x_{21} + 0.5_{41} + 0.5x_{51}}{\sum_{i=1}^{6} x_{i1}} = 0.3 \iff -0.3x_{11} + 0.7x_{21} - 0.3x_{31} + 0.2x_{41} + 0.2x_{51} - 0.3x_{61} = 0,$$

and

$$\frac{x_{31} + 0.5x_{51} + 0.5x_{61}}{\sum_{i=1}^{6} x_{i1}} = 0.3 \iff -0.3x_{11} - 0.3x_{21} + 0.7x_{31} - 0.3x_{41} + 0.2x_{51} + 0.2x_{61} = 0.$$

The constraints describing the proportion of each of the primary colors in the gray paint mix can be derived analogously:

$$\begin{aligned} 0.7x_{12} - 0.3x_{22} - 0.3x_{32} + 0.2x_{42} - 0.3x_{52} + 0.2x_{62} &= 0, \\ -0.3x_{12} + 0.7x_{22} - 0.3x_{32} + 0.2x_{42} + 0.2x_{52} - 0.3x_{62} &= 0, \\ -0.4x_{12} - 0.4x_{22} + 0.6x_{32} - 0.4x_{42} + 0.1x_{52} + 0.1x_{62} &= 0. \end{aligned}$$

Finally, we aim to minimize the cost of purchasing primary color paints:

$$\min \quad 20 \sum_{i=1}^{3} y_i.$$

We code the LO model in the script `model_mixing.py` *and the output is displayed below.*

```
The minimum paint cost is 200.00.
x11 = 15.0  x12 = 10.0
x21 = 10.0  x22 = 10.0
x31 = 15.0  x32 = 10.0
x41 = 10.0  x42 = 0.0
x51 = 0.0   x52 = 10.0
x61 = 0.0   x62 = 10.0
y1 = 5.0
y2 = 0.0
y3 = 5.0
```

Example 2.5. *A hospital uses a 12-hour shift schedule for its nurses, with each nurse working either day shifts (7:00 am-7:00 pm) or night shifts (7:00 pm-7:00 am). Each nurse works 3 consecutive day shifts or 3 consecutive night shifts and then has 4 days off. The hospital is aiming to design a schedule for day-shift nurses that minimizes the total number of nurses employed. The minimum number of nurses required for each day shift during a week is given in the following table:*

Day of week/shift	Nurses required
Monday (Mo)	16
Tuesday (Tu)	12
Wednesday (We)	18
Thursday (Th)	13
Friday (Fr)	15
Saturday (Sa)	9
Sunday (Su)	7

In addition, it is required that at least half of the day-shift nurses have weekends (Saturday and Sunday) off.

Note that a nurse's schedule can be defined by the first day of the three-day cycle. Thus we define the decision variables as follows.

$x_1 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Mo-Tu-We schedule*
$x_2 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Tu-We-Th schedule*
$x_3 \in \mathbb{Z}_{\geq 0}$: *number of nurses on We-Th-Fr schedule*
$x_4 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Th-Fr-Sa schedule*
$x_5 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Fr-Sa-Su schedule*
$x_6 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Sa-Su-Mo schedule*
$x_7 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Su-Mo-Tu schedule*

Here, $x \in \mathbb{Z}_{\geq 0}$ means that $x \in \mathbb{Z}$ and $x \geq 0$. On Monday, there are $x_1 + x_6 + x_7$ nurses working, so by requirement we should have

$$x_1 + x_6 + x_7 \geq 16.$$

Similarly, for the other days of the week, we have constraints

$$\begin{aligned} x_1 + x_2 + x_7 &\geq 12, \\ x_1 + x_2 + x_3 &\geq 18, \\ x_2 + x_3 + x_4 &\geq 13, \\ x_3 + x_4 + x_5 &\geq 15, \\ x_4 + x_5 + x_6 &\geq 9, \\ x_5 + x_6 + x_7 &\geq 7. \end{aligned}$$

Clearly any of these constraints imply that $\sum_{i=1}^{7} x_i \geq 1$. *Thus the requirement that half of the day-shift nurses have weekends off can be expressed as*

$$\frac{x_1 + x_2 + x_3}{\sum_{i=1}^{7} x_i} \geq \frac{1}{2}.$$

As done in Example 2.4, we can multiply both sides by $2\sum_{i=1}^{7} x_i$, *and rewrite this constraint as a linear one*

$$x_1 + x_2 + x_3 - x_4 - x_5 - x_6 - x_7 \geq 0.$$

The objective is to minimize the total number of nurses $\sum_{i=1}^{7} x_i$, *so the model can be written as*

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{7} x_i & \\ \text{s.t.} & x_1 + x_6 + x_7 \geq 16, & \\ & x_1 + x_2 + x_7 \geq 12, & \\ & x_1 + x_2 + x_3 \geq 18, & \\ & x_2 + x_3 + x_4 \geq 13, & \xrightarrow{\text{relax}} \\ & x_3 + x_4 + x_5 \geq 15, & \\ & x_4 + x_5 + x_6 \geq 9, & \\ & x_5 + x_6 + x_7 \geq 7, & \\ & \displaystyle\sum_{i=1}^{3} x_i - \sum_{i=4}^{7} x_i \geq 0, & \\ & x_i \in \mathbb{Z}_{\geq 0}, \quad i = 1, \ldots, 7. & \end{array} \qquad \begin{array}{lll} \min & \displaystyle\sum_{i=1}^{7} x_i & \\ \text{s.t.} & x_1 + x_6 + x_7 \geq 16, & \\ & x_1 + x_2 + x_7 \geq 12, & \\ & x_1 + x_2 + x_3 \geq 18, & \\ & x_2 + x_3 + x_4 \geq 13, & \\ & x_3 + x_4 + x_5 \geq 15, & \\ & x_4 + x_5 + x_6 \geq 9, & \\ & x_5 + x_6 + x_7 \geq 7, & \\ & \displaystyle\sum_{i=1}^{3} x_i - \sum_{i=4}^{7} x_i \geq 0, & \\ & x_i \geq 0, \quad i = 1, \ldots, 7. & \end{array}$$

The original model (on the left) is not *a LO model due to the integrality conditions on the variables. Nevertheless, all of the constraints are affine linear and the objective function is also linear. We can thus* relax *the integrality conditions and get a LO model (on the right) by only imposing* $x_i \geq 0$ *for each* $i = 1, \ldots, 7$. *This relaxed LO model is coded in the script* `model_scheduling.py` *and returns the following result.*

```
The minimum number of nurses is 31.0.
x1 = 11.00
x2 = 0.00
x3 = 10.00
x4 = 3.00
x5 = 2.00
x6 = 4.00
x7 = 1.00
```

Note that although the integrality conditions were relaxed, the solver actually returns an integral *optimal solution to the LO model, which means that we have found an optimal solution to the original model.*

Example 2.6. *A company plans the monthly trampoline production quantities, where the demand during the next four months is*

$$d_1 = 110, \quad d_2 = 120, \quad d_3 = 130, \quad d_4 = 100.$$

Currently, the company has an inventory of 20 trampolines. During each month, it can manufacture up to 100 trampolines with regular-time labor for \$120 per unit. With overtime labor, it can manufacture more trampolines, costing \$150 per unit. A per unit inventory cost of \$10 is charged at the end of each month. The warehouse can fit up to 25 trampolines. The management wants to develop a plan to minimize the total production and inventory costs. To build the model, we denote the index set for the planning horizon as $T := \{1, 2, 3, 4\}$. The decision variables are

$x_t \geq 0$: *number of units made using regular-time labor during month $t \in T$,*
$y_t \geq 0$: *number of units made using overtime labor during month $t \in T$,*
$l_t \geq 0$: *inventory level at the end of month $t \in T$.*

While these variables should be integers in practice, we temporarily relax the integrality conditions to formulate a LO model. Note that

$$l_t = l_{t-1} + (x_t + y_t) - d_t, \quad t \in T,$$

where $l_0 = 20$. Here, the inventory variables can be eliminated, but they often help understand and interpret the model. The total cost consists of three parts:

- *regular-time production cost:* $120 \sum_{t=1}^{4} x_t$,
- *overtime production cost:* $150 \sum_{t=1}^{4} y_t$,
- *inventory cost:* $10 \sum_{t=1}^{4} l_t$.

Thus the model can be written as (with the parameter $l_0 = 20$)

$$\begin{aligned} \min \quad & 120 \sum_{t=1}^{4} x_t + 150 \sum_{t=1}^{4} y_t + 10 \sum_{t=1}^{4} l_t \\ \text{s.t.} \quad & x_t \leq 100, \quad t \in T, \\ & l_t = l_{t-1} + x_t + y_t - d_t, \quad t \in T, \\ & l_t \leq 25, \quad t \in T, \\ & x_t, y_t, l_t \geq 0, \quad t \in T. \end{aligned}$$

We code the LO model in the script `model_inventory.py` *and the output is displayed below.*

```
The minimum cost is $54100.00.
The regular-time, overtime labor, and inventory level in
each period are shown below.
Period 1:
regular-time labor is 100.00
overtime labor is 0.00
inventory level is 10.00
Period 2:
regular-time labor is 100.00
overtime labor is 10.00
inventory level is 0.00
Period 3:
regular-time labor is 100.00
overtime labor is 30.00
inventory level is 0.00
Period 4:
regular-time labor is 100.00
overtime labor is -0.00
inventory level is 0.00
```

The obtained solution is indeed integer-valued *so the solution is feasible and optimal even when we enforce the integrality conditions on the number of trampolines.*

Example 2.7. *A wholesale company specializing in one product has $m = 3$ warehouses W_i, $i = 1, \ldots, m$ serving $n = 4$ retail locations R_j, $j = 1, \ldots, n$. Transporting one unit of the product from W_i to R_j costs c_{ij} dollars, $i = 1, \ldots, m$, and $j = 1, \ldots, n$. The company has s_i units of product available to ship from W_i, $i = 1, \ldots, m$. To satisfy the demand, at least d_j units of the product must be delivered to R_j. The values of s_i, d_j, and c_{ij} for $i = 1, \ldots, m$ and*

$j = 1, \ldots, n$ are given by

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 40 \\ 50 \\ 60 \end{bmatrix}, \quad \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} 17 \\ 33 \\ 23 \\ 47 \end{bmatrix}, \quad \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{bmatrix} = \begin{bmatrix} 3 & 2 & 1 & 1 \\ 2 & 3 & 5 & 4 \\ 3 & 5 & 7 & 8 \end{bmatrix}.$$

The goal is to find out how many units of the product should be shipped from each warehouse to each retail location so that the company's overall transportation costs are minimized. The decision variables are

$$x_{ij} \geq 0\text{: the product quantitiy shipped from } W_i \text{ to } R_j,\ i = 1, \ldots, m,\ j = 1, \ldots, n.$$

We need to make sure that the number of units shipped out of W_i does not exceed s_i

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

To satisfy the demand at R_j, we must have

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

The objective is to minimize the total cost of transportation, so the LO model can be written as

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} \leq s_i, \quad i = 1, \ldots, m, \\ & \sum_{i=1}^{m} x_{ij} \geq d_j, \quad j = 1, \ldots, n, \\ & x_{ij} \geq 0, \quad i = 1, \ldots, m,\ j = 1, \ldots, n. \end{aligned}$$

We code this LO model in the script `model_transportation.py` *and the output is displayed below.*

```
The minimum shipment cost is 336.00.
The shipment plan is displayed below.
location\warehouse     1     2     3
        1           0.00  0.00 17.00
        2           0.00 20.00 13.00
        3          23.00  0.00  0.00
        4          17.00 30.00  0.00
```

3 Linear Optimization II: Sets and Functions

3.1 LO Representable Functions

Sometimes even when the objective function is nonlinear, the problem can be reformulated as a LO model. We call such objective functions as *LO representable functions*. To be precise, we consider the following optimization model

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & Ax \le b, \\ & x \in \mathbb{R}^n, \end{aligned} \tag{3.1}$$

where $f(x)$ is a function on $\mathbb{R}^n$ defined by

$$f(x) = \max_{k=1,\dots,l} \{c_{k0} + c_{k1}x_1 + \cdots + c_{kn}x_n\}, \tag{3.2}$$

for some given number of pieces $l \in \mathbb{Z}_{\ge 1}$ and coefficients $c_{ki} \in \mathbb{R}$, $k = 1,\dots,l$, $i = 0,1,\dots,n$. Clearly, f can be a nonlinear function. (Hint: take $n+1$ points such that the maximum at these points are not attained at the same index $k = 1,\dots,l$ and derive a contradiction.) Nevertheless, problem (3.1) can be rewritten as a LO model with an additional variable $y \in \mathbb{R}$

$$\begin{aligned} \min \quad & y \\ \text{s.t.} \quad & Ax \le b, \\ & y \ge c_{k0} + \sum_{i=1}^{n} c_{ki}x_i, \quad k = 1,\dots,l, \\ & x \in \mathbb{R}^n,\ y \in \mathbb{R}. \end{aligned} \tag{3.3}$$

To see this, note that

$$y \ge f(x) \iff y \ge c_{k0} + \sum_{i=1}^{n} c_{ki}x_i, \quad k = 1,\dots,l. \tag{3.4}$$

Thus the constraints involving $y \in \mathbb{R}$ in (3.3) is equivalent to $y \ge f(x)$. As y is not involved in any other constraint, an optimal solution (when it exists) must have $y = f(x)$. Therefore, (3.3) preserves the optimal value and any optimal solution in x that comes from (3.1).

Using such reformulation technique on the absolute value function $|x| = \max\{x, -x\}$, we can build a LO model in the following example.

Example 3.1. *A machine shop has a drill press and a milling machine which are used to produce*

two parts A and B. The required time (in minutes) per unit part on each machine is shown in the table below.

	Drill press	Milling machine
A	3	4
B	5	3

The shop must produce at least 50 units in total (both A and B) and at least 30 units of part A and 20 units of B, and it can make at most 100 units of part A and 80 units of part B. Assume that the shop can make fractional amount of the parts. The goal is to minimize the absolute difference between the total running time of the drill press and that of the milling machine. Our two decision variables are

$$\begin{aligned} 30 \le x_1 \le 100&: \quad \textit{units of part A to be produced,} \\ 20 \le x_2 \le 80&: \quad \textit{units of part B to be produced.} \end{aligned}$$

The linear constraint on the total units to be produced can be written as

$$x_1 + x_2 \ge 50.$$

The difference between the total running time of the drill press and that of the milling machine is

$$(3x_1 + 5x_2) - (4x_1 + 3x_2) = -x_1 + 2x_2.$$

To reformulate the absolute value function $|-x_1 + 2x_2| = \max\{-x_1 + 2x_2, x_1 - 2x_2\}$*, we need to introduce an additional decision variable*

$$y \in \mathbb{R}: \quad \textit{absolute difference between the total running times,}$$

and two additional linear constraints

$$\begin{aligned} y &\ge -x_1 + 2x_2, \\ y &\ge x_1 - 2x_2. \end{aligned}$$

In summary, our LO model can be written as

$$\begin{aligned} \min \quad & y \\ \text{s.t.} \quad & y \ge -x_1 + 2x_2, \\ & y \ge x_1 - 2x_2, \\ & x_1 + x_2 \ge 50, \\ & 30 \le x_1 \le 100, \quad 20 \le x_2 \le 80, \quad y \in \mathbb{R}. \end{aligned}$$

We code the LO model in the script `model_machine.py` *and the output is displayed below.*

```
The minimum absolute difference is 0.00.
Units of part A to be produced = 40.00
Units of part B to be produced = 20.00
```

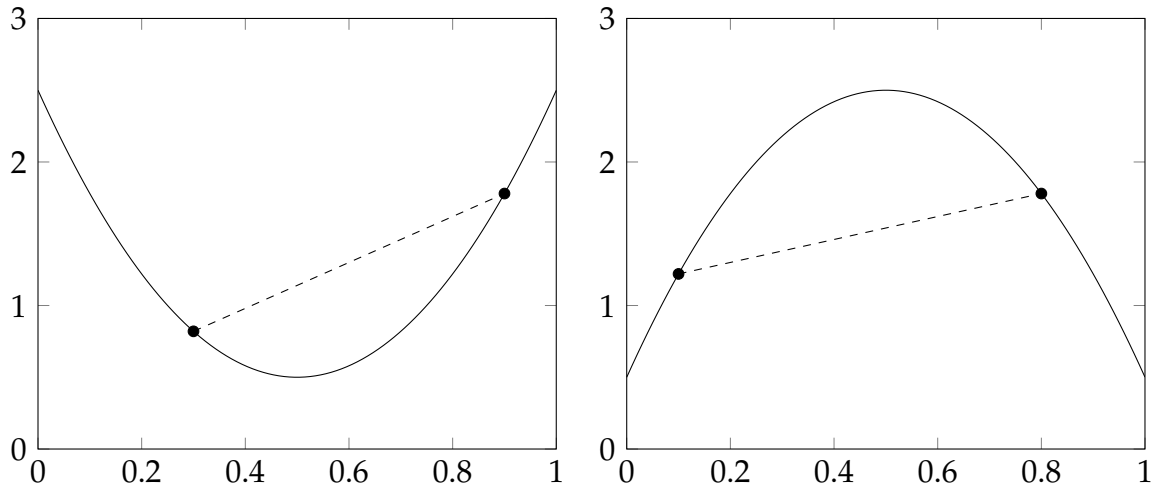
A natural question is how we can tell whether a nonlinear objective function $f(x)$ is LO representable or not. It turns out that the answer will depend on whether we are maximizing or minimizing our objective value. As we have seen above, for a minimization problem, any "finite-maximum" function (as defined in (3.2)) is LO representable. Using the same argument, a "finite-minimum" function can be reformulated in a LO maximization problem, as

$$y \leq \min_{k=1,\ldots,l} \{c_{k0} + c_{k1}x_1 + \cdots + c_{kn}x_n\} \iff y \leq c_{k0} + \sum_{i=1}^{n} c_{ki}x_i, \quad k = 1, \ldots, l.$$

One important characterization of these maximum or minimum function is by *convexity* or *concavity* defined as follows.

Definition 3.2. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is* convex *if for any $x, y \in \mathbb{R}^n$ and any $0 \leq t \leq 1$, we have $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$. A function f is* concave *if $-f$ is convex.*

Geometrically, the definition says that if you take a line segment between any two points in the graph of your function and it stays above (resp. below) the graph, then it is convex (resp. concave). You can see simple examples in Figure 3.1. Intuitively speaking, a convex function bends "upward" (slope increasing in any direction), and a concave function bends "downward" (slope decreasing in any direction).



(a) a convex function (b) a concave function

Figure 3.1: Illustration of convexity and concavity

We claim that a maximum of linear functions is always convex. To see this, let L be an index set and $f(x) := \max_{k \in L}\{c_{k0} + c_{k1}x_1 + \cdots + c_{kn}x_n\}$. Then for any $x, y \in \mathbb{R}^n$

and $0 \le t \le 1$, we have

$$\begin{aligned}
&f(tx + (1-t)y) \\
&= \max_{k \in L} \{c_{k0} + c_{k1}(tx_1 + (1-t)y_1) + \cdots + c_{kn}(tx_n + (1-t)y_n)\} \\
&= \max_{k \in L} \{t(c_{k0} + c_{k1}x_1 + \cdots + c_{kn}x_n) + (1-t)(c_{k0} + c_{k1}y_1 + \cdots + c_{kn}y_n)\} \\
&\le t \cdot \max_{k \in L} \{c_{k0} + c_{k1}x_1 + \cdots + c_{kn}x_n\} + (1-t) \cdot \max_{l \in L} \{c_{l0} + c_{l1}y_1 + \cdots + c_{ln}y_n\} \\
&= tf(x) + (1-t)f(y).
\end{aligned}$$

Here, the first equality is derived directly by the definition of f; the second equality is derived by rearranging terms (and by splitting c_{k0} into $tc_{k0} + (1-t)c_{k0}$); the inequality here is due to the fact that we allow the maximum to be taken at different indices k and l; and the last equality is again by the definition of f. By reverting the direction of the inequality here, the argument shows that a minimum of linear functions is concave.

It is not enough by convexity or concavity alone to guarantee that the function is LO representable. For example, if the index set is infinite (such as $L = \mathbb{Z}$), then we might need infinitely many constraints in the reformulation (3.4) (as each index could correspond to one constraint). Thus we would also need the function to have finitely many "pieces" for it to be LO representable. For univariate functions, this can be defined as follows.

Definition 3.3. *A univariate function $f : \mathbb{R} \to \mathbb{R}$ is* piecewise linear *(with finitely many pieces), if there are points $-\infty = a_0 < a_1 < \cdots < a_l = +\infty$ such that on each interval $I_k := \{x \in \mathbb{R} : a_{k-1} < x < a_k\}$, $k = 1, \ldots, l$, $f(x)$ is an affine linear function, i.e., there exist $b_k, c_k \in \mathbb{R}$ such that*

$$f(x) = b_k x + c_k, \quad \forall\, x \in I_k, \quad k = 1, \ldots, l.$$

Figure 3.2 illustrates some univariate piecewise linear functions on the interval $[0, 1]$. From the figures, we can see that a convex (resp. concave) piecewise linear function must have its dashed parts (i.e., the extension of each linear piece) below (resp. above) the function itself. In fact, the following statements are equivalent for a univariate piecewise linear function $f(x)$:

(i) $f(x)$ is convex;

(ii) $f(x) = \max_{k=1,\ldots,l}\{b_k x + c_k\}$;

(iii) the points and the coefficients in Definition 3.3 for $f(x)$ satisfy

$$f(a_k) = a_k b_k + c_k = a_k b_{k+1} + c_{k+1}, \text{ and } b_k \le b_{k+1} \quad \forall\, k = 1, \ldots, l-1.$$

The last statement essentially says that the function $f(x)$ is continuous and has non-

decreasing slopes. A possible hint for any reader interested in the proof is that the definition of convexity for $f(x)$ implies that for any $h > 0$

$$\frac{f(x) - f(x-h)}{h} \leq \frac{f(x+h) - f(x)}{h}, \quad \forall\, x \in \mathbb{R}.$$

This shows that the slope should be non-decreasing. Besides, taking limits of f from both sides towards a_k requires f to be continuous at a_k, for $k = 1, \ldots, l-1$. Similarly, the following statements are also equivalent for a piecewise linear function $f(x)$:

(i) $f(x)$ is concave;

(ii) $f(x) = \min_{k=1,\ldots,l}\{b_k x + c_k\}$;

(iii) the points and the coefficients in Definition 3.3 for $f(x)$ satisfy

$$f(a_k) = a_k b_k + c_k = a_k b_{k+1} + c_{k+1}, \text{ and } b_k \geq b_{k+1} \quad \forall\, k = 1, \ldots, l-1.$$

In practice, we may sometimes *approximate* a nonlinear objective function by piecewise linear functions. For example, given a nonlinear convex function $f(x)$ on an interval $[0, 1]$, we can take points $0 = a_0 < a_1 < \cdots < a_{l-1} < a_l = 1$, and then set

$$b_k = \frac{f(a_k) - f(a_{k-1})}{a_k - a_{k-1}}, \quad c_k = f(a_k) - a_k b_k, \quad k = 1, \ldots, l. \tag{3.5}$$

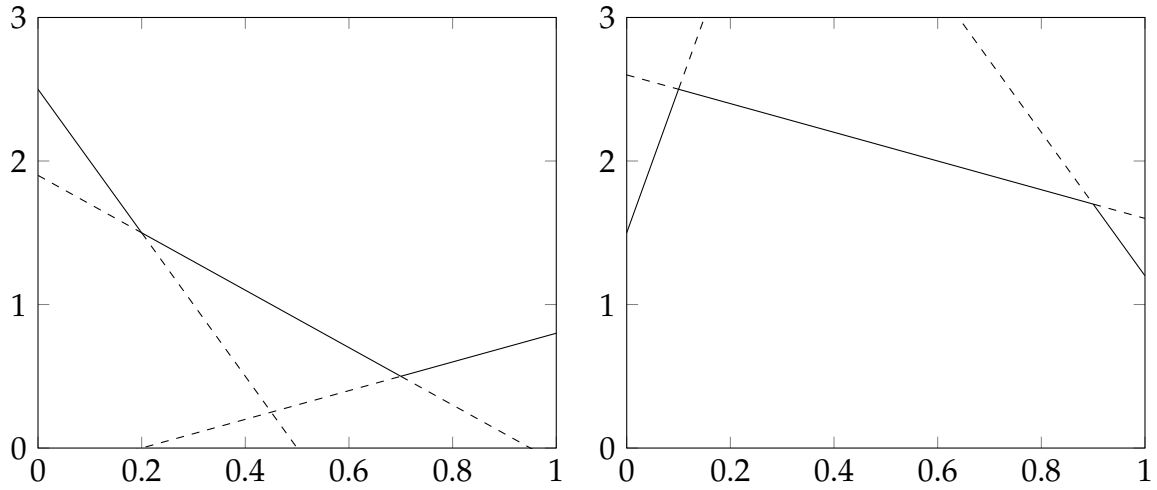
An illustration is plotted in Figure 3.3a. This procedure is often called *inner-approximation* (or *over-approximation*) of the nonlinear function f. Alternatively, if one can find differential information at points $0 \leq a'_1 < a'_2 < \cdots < a'_l \leq 1$, then an *outer-approximation* (or *under-approximation*) can be built as in Figure 3.3b.

Example 3.4. *An electric power grid operator wants to find a generation plan for two generators $i = 1$ and 2. The generation cost functions f_i for generators $i = 1, 2$ are described by two convex functions $f_1(x) = 2 + 0.5x + 0.01x^2$ and $f_2(x) = 3 + 0.4x + 0.02x^2$. The demand in the region is 10 MW for the next hour. Assume that there is no loss in the transmission. The goal is to minimize the total generation cost while meeting the demand. Let $x_i \geq$ denote the power generation from the generator $i = 1, 2$. The power demand constraint can then be written as*

$$x_1 + x_2 \geq 10.$$

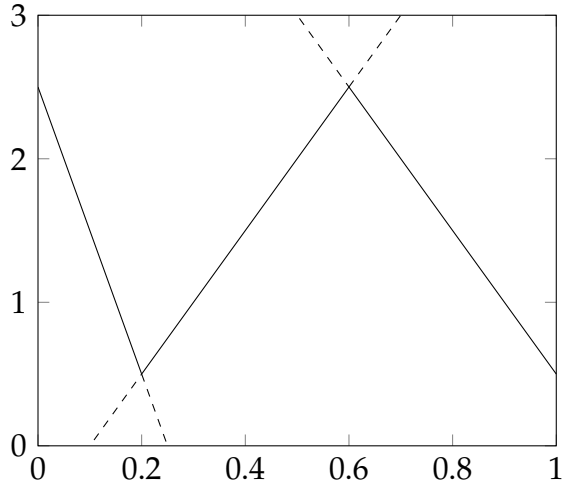
Note that it suffices to consider generation within the range $[0, 10]$ for both generators. To handle the nonlinearity, we build inner-approximations for the generation cost functions f_1 and f_2 over $[0, 10]$, using (3.5) on the function values at the points $x_1, x_2 = 0, 5, 10$:

$$f_1(x_1) \lessapprox \max\{0.55x_1 + 2,\ 0.65x_1 + 1.5\}$$
$$f_2(x_2) \lessapprox \max\{0.5x_2 + 3,\ 0.7x_2 + 2\}.$$

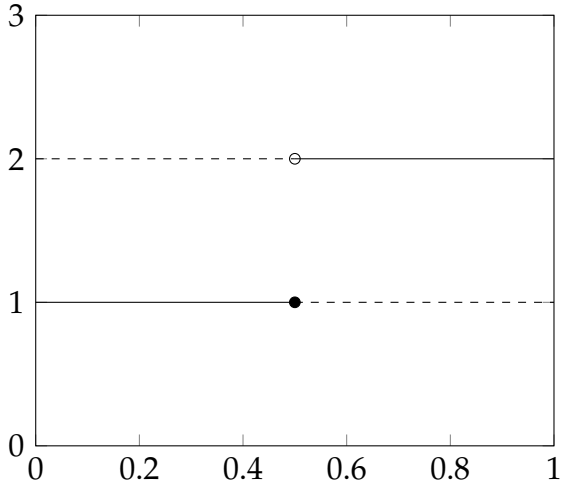


(a) a convex piecewise linear function

(b) a concave piecewise linear function

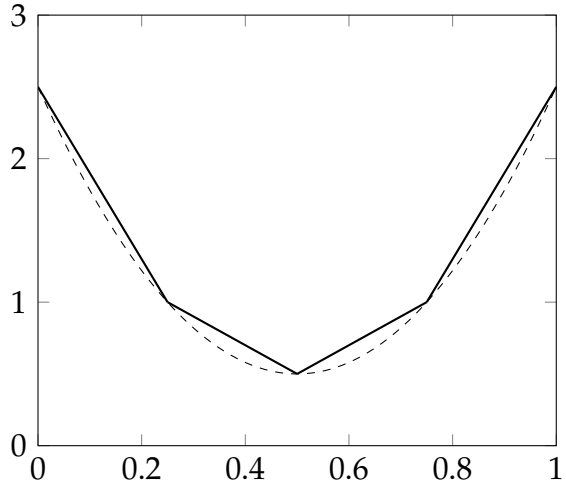


(c) neither convex nor concave

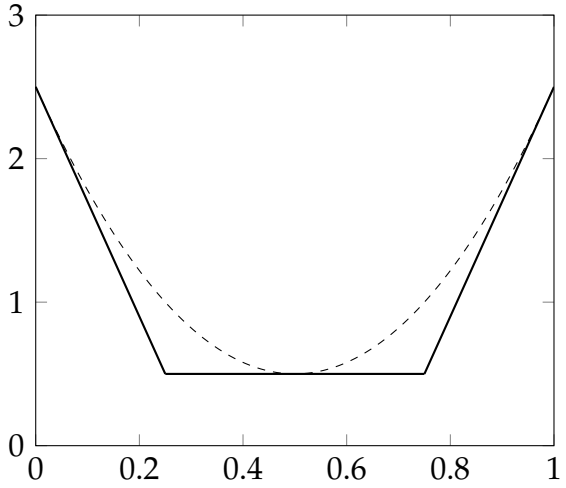


(d) neither convex nor concave

Figure 3.2: Illustration of piecewise linear functions



(a) a piecewise linear inner-approximation



(b) a piecewise linear outer-approximation

Figure 3.3: Illustration of inner- and outer-approximations of a nonlinear function

Therefore, by introducing additional variables $y_1, y_2 \in \mathbb{R}$ to be the approximate generation costs for generators $i = 1, 2$, we can write our approximate LO model as

$$\begin{aligned} \min \quad & y_1 + y_2 \\ \text{s. t.} \quad & x_1 + x_2 \geq 10, \\ & y_1 \geq 0.55x_1 + 2, \\ & y_1 \geq 0.65x_1 + 1.5, \\ & y_2 \geq 0.5x_2 + 3, \\ & y_2 \geq 0.7x_2 + 2, \\ & x_1, x_2 \geq 0,\ y_1, y_2 \in \mathbb{R}. \end{aligned}$$

We code the LO model in `model_generation.py` *and the output is displayed below.*

```
The minimum generation cost is 10.25.
Power generation at generator 1 = 5.00.
Approximate generation cost of generator 1 = 4.75.
Power generation at generator 2 = 5.00.
Approximate generation cost of generator 2 = 5.50.
```

We check that at the point $(x_1, x_2) = (5.0, 5.0)$, the actual generation cost $(f_1(x_1), f_2(x_2)) = (4.75, 5.5)$, which agrees with the our obtained approximate generation cost (y_1, y_2). This means that our approximation is tight *at the obtained solution and we have found an optimal solution exactly.*

3.2 LO Feasible Regions and Graphical Solutions

Other than the objective function, one may wonder what sets can be represented as the feasible region of a LO model. Such sets are known as *polyhedra*, which can be defined as follows.

Definition 3.5. (i) *A closed halfspace H in $\mathbb{R}^n$ is a subset*

$$H := \{x \in \mathbb{R}^n : a^\mathsf{T} x \leq b\}$$

for some vector $a \in \mathbb{R}^n$ and real number $b \in \mathbb{R}$.

(ii) *A polyhedron (or a polyhedral set) in $\mathbb{R}^n$ is an intersection of finitely many closed halfspaces in $\mathbb{R}^n$.*

Recall that a general LO feasible region can be written as $X := \{x \in \mathbb{R}^n : Ax \leq b\}$ for some matrix $A \in \mathbb{R}^{m \times n}$ and some vector $b \in \mathbb{R}^m$. It is then clear that X is a

polyhedron because

$$X = \bigcap_{j=1}^{m} \left\{ x \in \mathbb{R}^n : a_j^\mathsf{T} x \leq b_j \right\},$$

where a_j is the j-th row vector of A. There could be multiple ways to represent a polyhedron as a LO feasible region. For example, $X := \{x \in \mathbb{R}^2 : 0 \leq x_1, x_2 \leq 1\}$ and $X' := \{x \in \mathbb{R}^2 : 0 \leq x_1, x_2 \leq 1,\ x_1 + x_2 \leq 2\}$ are the feasible regions for two LO problems, but they represent the sample polyhedron, which is a square of side length 1.

Similar to LO representable functions, an important characterization of LO representable sets is convexity.

Definition 3.6. (i) *A set $X \subseteq \mathbb{R}^n$ is convex if for any two points $x, y \in X$ and any $0 \leq t \leq 1$, the point $tx + (1-t)y \in X$.*

(ii) *A closed convex set in $\mathbb{R}^n$ is an intersection of (possibly infinitely many) closed halfspaces in $\mathbb{R}^n$.*

Intuitively, a set is convex if we connect any two points in the set and the line segment would stay in the set. See Figure 3.4 for examples. To see that we are not abusing terminology, we show that a closed convex set is indeed convex as follows. Suppose J is a possibly infinite index set and

$$X = \bigcap_{j \in J} \left\{ x \in \mathbb{R}^n : a_j^\mathsf{T} x \leq b_j \right\}$$

is a closed convex set for vectors $a_j \in \mathbb{R}^n$ and real numbers $b_j \in \mathbb{R}$, $j \in J$. Take any points $x, y \in X$, which by definition satisfies

$$a_j^\mathsf{T} x \leq b_j, \text{ and } a_j^\mathsf{T} y \leq b_j, \quad \forall\, j \in J.$$

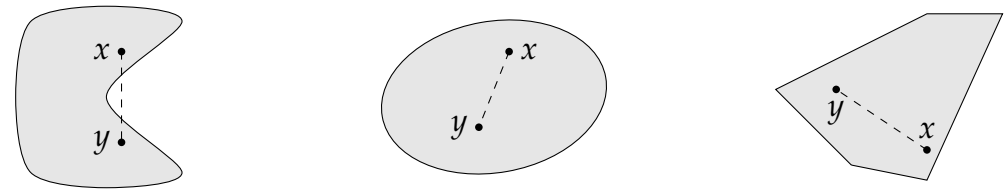
Thus using linearity, we see that

$$a_j^\mathsf{T} \left(tx + (1-t)y\right) = t \cdot (a_j^\mathsf{T} x) + (1-t) \cdot (a_j^\mathsf{T} y) \leq tb_j + (1-t)b_j = b_j.$$

As this holds for any $j \in J$, we conclude that $tx + (1-t)y \in X$, which shows the convexity of X.

Using these definitions, it is clear that a polyhedron is a closed convex set. The converse is not necessarily true, which can be seen from a planar example in Figure 3.4b.

Now we can finally answer the question about which functions are LO representable. For a minimization problem with decision variables $x \in \mathbb{R}^n$ and an auxiliary variable $y \in \mathbb{R}$, our reformulation technique (3.4) requires us to write the set $\{(x, y) \in \mathbb{R}^{n+1} : y \geq f(x)\}$ as a polyhedron (LO feasible region), so $f(x)$ must be a finite-maximum



(a) Not convex (b) Convex, not polyhedral (c) Polyhedral

Figure 3.4: Non-example and examples of convex sets

function, which is piecewise linear and convex. Similarly, for a maximization problem, our reformulation requires us to write the set $\{(x, y) \in \mathbb{R}^{n+1} : y \leq f(x)\}$ as a polyhedron (LO feasible region), so $f(x)$ must be a finite-minimum function, which is piecewise linear and concave.

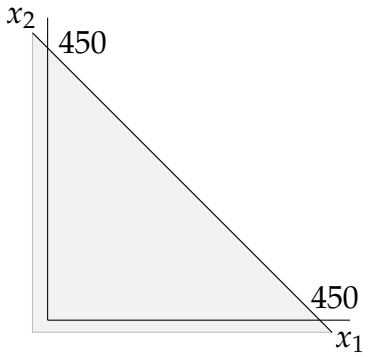
By interpreting polyhedra as intersection of halfspaces can help us visualize LO feasible regions. This is particularly useful for solving the LO problem when there are only two variables.

Example 3.7. *A company produces two types of baby carriers, non-reversible and reversible. Each non-reversible carrier sells for \$23, requires 2 linear yards of a solid color fabric, and costs \$8 to manufacture. Each reversible carrier sells for \$35, requires 2 linear yards of a printed fabric as well as 2 linear yards of a solid color fabric, and costs \$10 to manufacture. The company has 900 linear yards of solid color fabrics and 600 linear yards of printed fabrics available for its new carrier collection. It can spend up to \$4,000 on manufacturing the carriers. The demand is such that all reversible carriers made are projected to sell, whereas at most 350 non-reversible carriers can be sold. The goal of the company is to maximize its profit (e.g., the difference of revenues and expenses) resulting from manufacturing and selling the new carrier collection.*
We define $x_1, x_2 \geq 0$ to be the numbers of non-reversible and reversible carriers to manufacture. Then the LO model can be written as

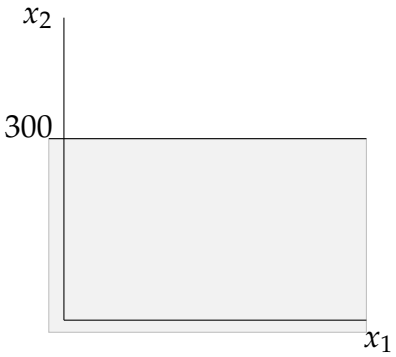
$$
\begin{array}{lrcrcll}
\max & 15x_1 & + & 25x_2 & & & \textit{(profit)} \\
\text{s.t.} & x_1 & + & x_2 & \leq & 450 & \textit{(solid color fabric constraint)} \\
 & & & x_2 & \leq & 300 & \textit{(printed fabric constraint)} \\
 & 4x_1 & + & 5x_2 & \leq & 2,000 & \textit{(budget constraint)} \\
 & x_1 & & & \leq & 350 & \textit{(demand constraint)} \\
 & & & x_1, x_2 & \geq & 0 & \textit{(nonnegativity constraints).}
\end{array}
$$

Each constraint can be plotted on the x_1-x_2 plane as in Figure 3.5. Putting the constraints together, we can find optimal solutions by moving in the improving direction of the linear objective function $z = 15x_1 + 25x_2$, as shown in Figure 3.6. The optimal solution is $(x_1, x_2) = (125, 300)$.

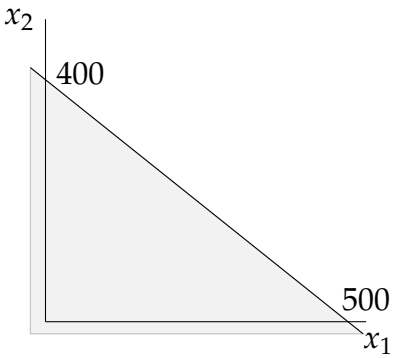
Example 3.8. *Now suppose in Example 6.1, the price of a non-reversible carrier is raised to*



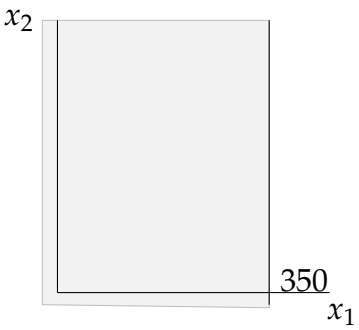
(a) solid fabric constraint $x_1 + x_2 \leq 450$



(b) printed fabric constraint $x_2 \leq 300$



(c) budget constraint $4x_1 + 5x_2 \leq 2,000$



(d) demand constraint $x_1 \leq 350$

(e) nonnegativity constraints

Figure 3.5: Constraints in the baby carrier problem

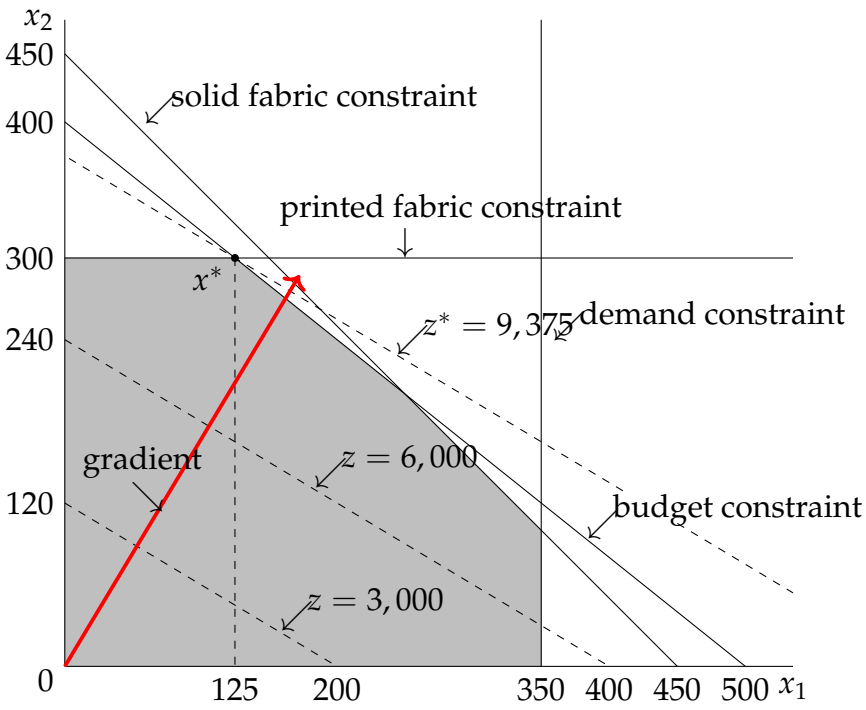


Figure 3.6: Feasible region and objective of the baby carriers problem

$28. The modified LO model becomes

$$
\begin{array}{lrcrcll}
\max & 20x_1 & + & 25x_2 & & & \textit{(profit)} \\
\text{s.t.} & x_1 & + & x_2 & \leq & 450 & \textit{(solid color fabric constraint)} \\
 & & & x_2 & \leq & 300 & \textit{(printed fabric constraint)} \\
 & 4x_1 & + & 5x_2 & \leq & 2,000 & \textit{(budget constraint)} \\
 & x_1 & & & \leq & 350 & \textit{(demand constraint)} \\
 & & & x_1, x_2 & \geq & 0 & \textit{(nonnegativity constraints).}
\end{array}
$$

Note that the feasible region is the same while only the improving direction (gradient) is changed. The modified LO model can be plotted as in Figure 3.7. Now we can see that any points between $x^* = (125, 300)$ and $x' = (250, 200)$ are optimal and the optimal objective value is $z^* = 10,000$.

Example 3.9. *A retail store is planning an advertising campaign aiming to increase the number of customers visiting its physical location, as well as its online store. The store manager would like to advertise through a local magazine and through an online social network. The manager estimates that each 1,000 dollars invested in magazine ads will attract 100 new customers to the store, as well as 500 new website visitors. In addition, each 1,000 dollars invested in online advertising will attract 50 new local store customers, as well as 1,000 new website visitors. The target for this campaign is to bring at least 500 new guests to the physical store and at least 5,000 new visitors to the online store. The decision variables are*

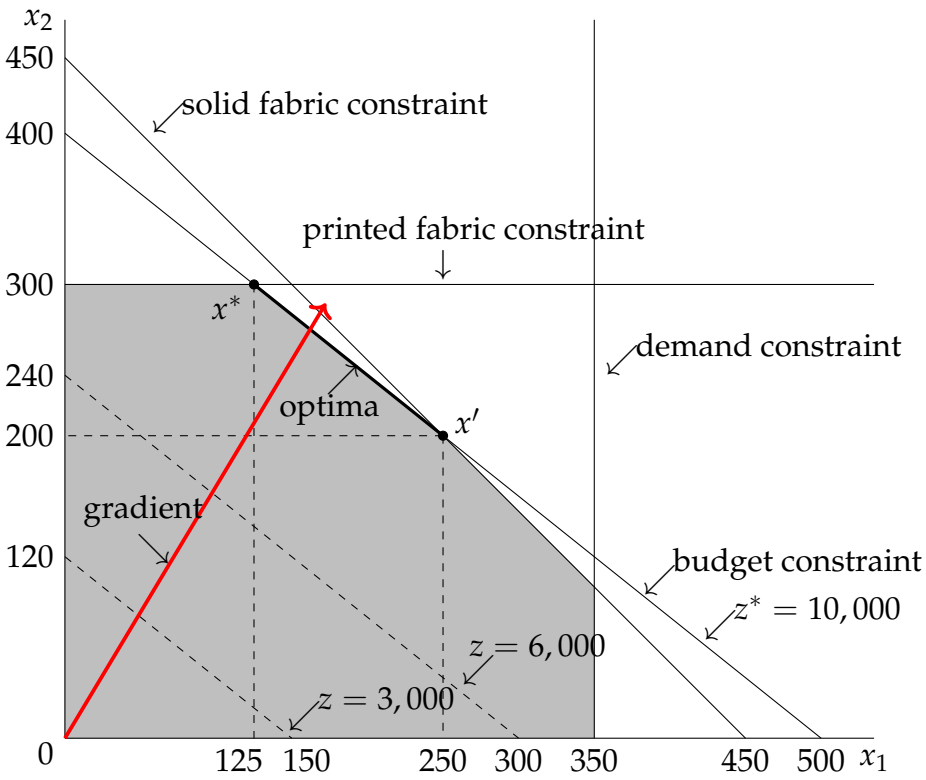


Figure 3.7: Feasible region and objective of the modified baby carriers problem

$x_1 \geq 0$: *budget for magazine advertising (in thousands of dollars)*
$x_2 \geq 0$: *budget for online advertising (in thousands of dollars),*

and the LO model can be written as

$$\begin{array}{lrcrcll} \min & x_1 & + & x_2 & & & \\ \text{s.t.} & 100x_1 & + & 50x_2 & \geq & 500 & \textit{(store visitors)} \\ & 500x_1 & + & 1,000x_2 & \geq & 5,000 & \textit{(website visitors)} \\ & & & x_1, x_2 & \geq & 0. & \textit{(nonnegativity)} \end{array}$$

We can plot the feasible region in Figure 3.8 and find that $(x_1, x_2) = (10/3, 10/3)$ *is the optimal solution with the optimal value* $z^* = 20/3$.

Instead of plotting the feasible regions manually, we can also use the `matplotlib` package in `Python` to (approximately) plot them. For example, we code the plotting procedure for Examples 6.1 and 3.9 in the scripts `plot_carriers.py` and `plot_advertising.py` and display the output in Figure 3.9.

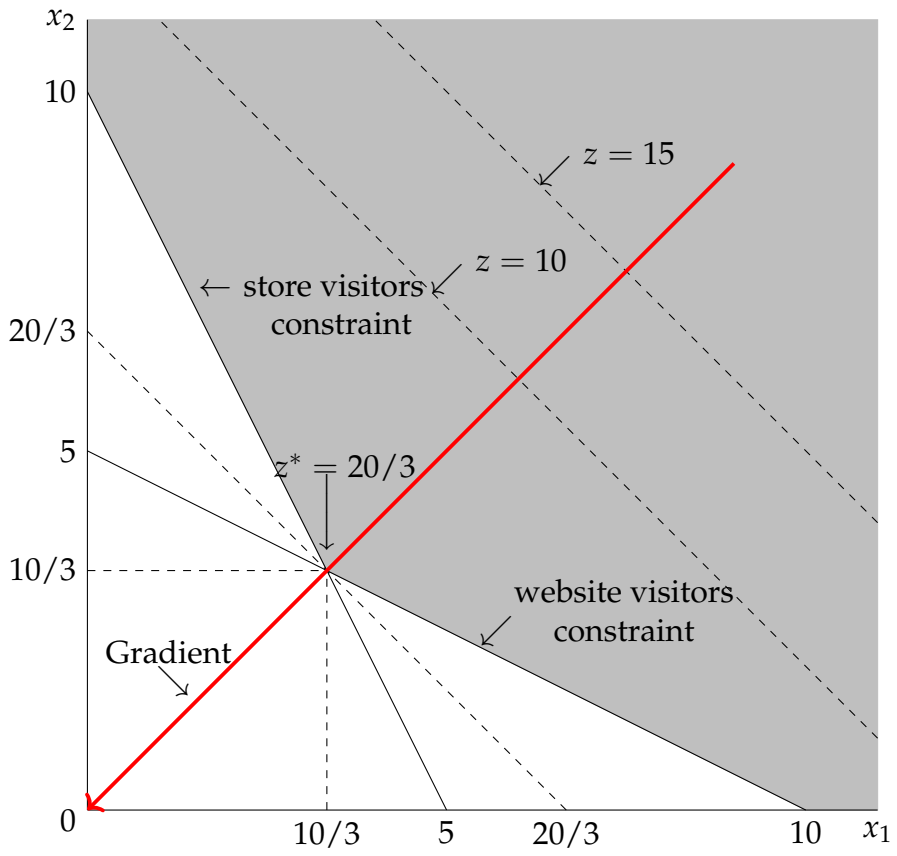


Figure 3.8: Feasible region and objective for advertising campaign problem

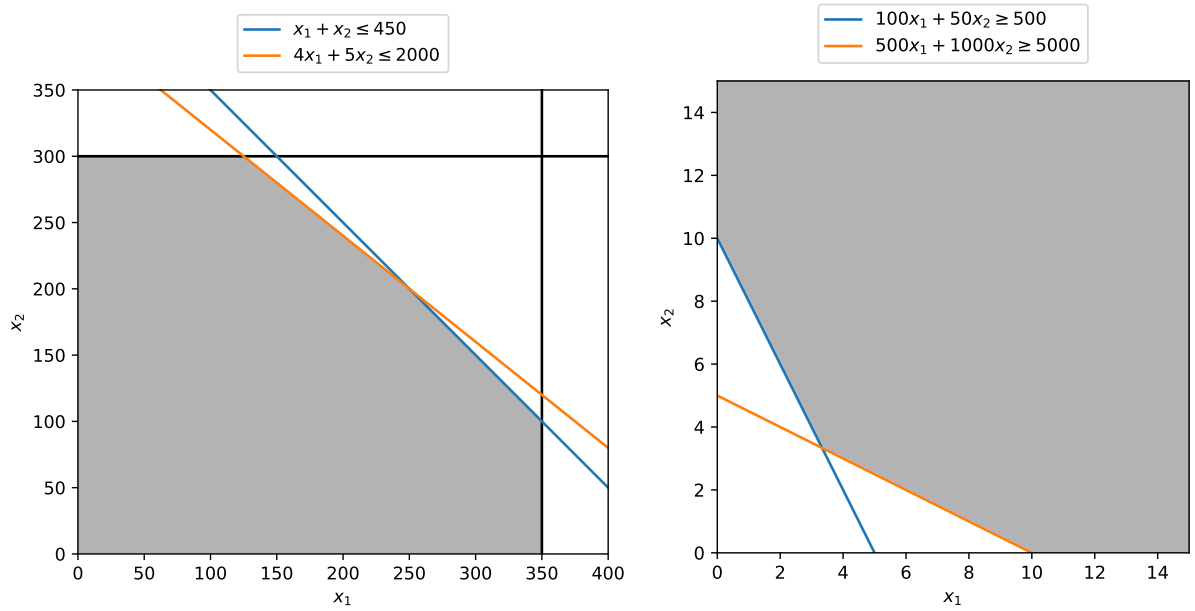


Figure 3.9: Feasible regions for Examples 6.1 and 3.9

4 Linear Optimization III: Simplex Method

4.1 Standard Form of Linear Optimization (LO)

In graphical solutions for LO problems with two variables, we can find optimal solutions on vertices (the corner points). This observation can be more formally exploited by the *simplex method*. Before we present the algorithm, it is beneficial to consider *standard forms* of LO models, which have the following form.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ \text{s. t.} \quad & \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \ldots, m, \\ & x_1, \cdots, x_n \geq 0. \end{aligned} \tag{4.1}$$

Namely, we only have linear equality constraints and variable nonnegativity conditions in the standard form. Any LO formulation can be converted into the standard form by introducing auxiliary variables when necessary. For a less-than-or-equal-to constraint, we can use a *slack variable* $s \geq 0$ to rewrite it as an equality constraint

$$\sum_{i=1}^{n} a_i x_i \leq b \iff a_1 x_1 + \cdots + a_n x_n + s = b, \quad s \geq 0. \tag{4.2}$$

Similarly, for the greater-than-or-equal-to constraint, we can use an *excess variable* $e \geq 0$ to rewrite it as an equality constraint

$$\sum_{i=1}^{n} a_i x_i \geq b \iff a_1 x_1 + \cdots + a_n x_n - e = b, \quad e \geq 0. \tag{4.3}$$

For a *nonpositive variable*, we use its opposite to replace it in the decision variables

$$x_i \leq 0 \iff x_i' \geq 0. \tag{4.4}$$

If a variable is *free* (or *unrestricted in sign*), then we replace it with two nonnegative auxiliary variables

$$x_i \in \mathbb{R} \iff x_i = x_i' - x_i'', \quad x_i', x_i'' \geq 0. \tag{4.5}$$

Example 4.1. *The standard form of the following LO problem*

$$\begin{array}{rrcrcrcl} \max & 3x_1 & - & 5x_2 & + & 7x_3 & & \\ \text{s.t.} & 2x_1 & + & 4x_2 & - & x_3 & \geq & -3, \\ & 4x_1 & - & 2x_2 & + & 8x_3 & \leq & 7, \\ & 9x_1 & + & x_2 & + & 3x_3 & = & 11, \\ & & & & & x_1 & \in & \mathbb{R}, \\ & & & & & x_2 & \leq & 0, \\ & & & & & x_3 & \geq & 0. \end{array}$$

is given by

$$\begin{array}{rrcrcrcrcrcl} \max & 3x_1' & - & 3x_1'' & + & 5x_2' & + & 7x_3 & & & & \\ \text{s.t.} & 2x_1' & - & 2x_1'' & - & 4x_2' & - & x_3 & - & e_1 & = & -3, \\ & 4x_1' & - & 4x_1'' & + & 2x_2' & + & 8x_3 & + & s_2 & = & 7, \\ & 9x_1' & - & 9x_1'' & - & x_2' & + & 3x_3 & & & = & 11, \\ & & & & & & & x_1', x_1'', x_2', x_3, e_1, s_2 & & & \geq & 0. \end{array}$$

The standard form helps us to determine all variable values once we fix a subset of them to zero. To be precise, let N denote a subset of indices $\{1, \ldots, n\}$ corresponding to *nonbasic* variables, and B a subset of indices of *basic* variables, such that the values of all basic variables can be determined through the equality constraints, once all nonbasic variables are fixed to zero. Sometimes B is called a *basis*. Any solution obtained by fixing nonbasic variables to zero is called a *basic solution*. The values of the basic variables in a basic solution could be negative. If all basic variables have nonnegative values, then the basic solution is further called a *basic feasible solution*.

Using linear algebra terminology, we can also say that B is an index set for basic variables when the submatrix A_B, that is formed by columns of A with indices in B, is nonsingular. Consequently, the number of basic variables should be the same as the number of constraints, assuming that the matrix A has full row rank. The geometric intuition of considering basic variables is that they can represent vertices of the feasible region. To see this, you may think about solving a system of m equations as finding the unique intersection point of m hyperplanes when the matrix A_B has full rank m.

Example 4.2. *By renaming the variables in Example 4.1, we can write it as*

$$\begin{array}{rrcrcrcrcrcl} \max & 3x_1 & - & 3x_2 & + & 5x_3 & + & 7x_4 & & & & \\ \text{s.t.} & 2x_1 & - & 2x_2 & - & 4x_3 & - & x_4 & - & x_5 & = & -3, \\ & 4x_1 & - & 4x_2 & + & 2x_3 & + & 8x_4 & + & x_6 & = & 7, \\ & 9x_1 & - & 9x_2 & - & x_3 & + & 3x_4 & & & = & 11, \\ & & & & & & & x_1, x_2, x_3, x_4, x_5, x_6 & & & \geq & 0. \end{array}$$

We can set $B = \{4,5,6\}$ *and* $N = \{1,2,3\}$. *In this case, by setting* $x_1 = x_2 = x_3 = 0$, *the last constraint tells us that* $x_4 = 11/3$, *which then implies*

$$x_5 = 3 - x_4 = -\frac{2}{3},$$
$$x_6 = 7 - 8x_4 = -\frac{67}{3}.$$

Here, as $x_5, x_6 < 0$, *we have an* infeasible *basic solution.*

Example 4.3. *Consider the following LO problem:*

$$\begin{array}{lrcrcrcl} \max & 5x_1 & + & 5x_2 & + & 3x_3 & & \\ \text{s.t.} & x_1 & + & 3x_2 & + & x_3 & \leq & 3, \\ & -x_1 & & & + & 3x_3 & \leq & 2, \\ & 2x_1 & - & x_2 & + & 2x_3 & \leq & 4, \\ & 2x_1 & + & 3x_2 & - & x_3 & \leq & 2, \\ & & & & & x_1, x_2, x_3 & \geq & 0. \end{array}$$

We introduce slack variables $x_4, x_5, x_6, x_7 \geq 0$, *the values of which can be uniquely determined by those of* x_1, x_2, x_3 *as*

$$\begin{array}{rcrcrcrcr} z & = & & & 5x_1 & + & 5x_2 & + & 3x_3 \\ \hline x_4 & = & 3 & - & x_1 & - & 3x_2 & - & x_3, \\ x_5 & = & 2 & + & x_1 & & & - & 3x_3, \\ x_6 & = & 4 & - & 2x_1 & + & x_2 & - & 2x_3, \\ x_7 & = & 2 & - & 2x_1 & - & 3x_2 & + & x_3. \end{array}$$

If we set $B = \{4,5,6,7\}$ *and* $N = \{1,2,3\}$, *then we can get a basic solution* $x_1 = x_2 = x_3 = 0$, $x_4 = 3$, $x_5 = 2$, $x_6 = 4$, *and* $x_7 = 2$, *which is feasible to the LO problem. The objective value, which we denote as* z, *is 0 at this basic feasible solution (bfs).*

4.2 Simplex Method and Tableau

To move from a basic feasible solution to a "better" solution, we can select a nonbasic variable with positive "impact" on the objective value to become a basic variable. The new basic variable is called an *entering variable*, as it enters the basis, while the new nonbasic variable is called a *leaving variable*. We use Example 4.3 to illustrate the procedure as follows.

Example 4.3 (continued)**.** *As both the nonbasic variables* x_1 *and* x_2 *have the largest coefficient* 5 *in the z-row, we pick* x_1 *to be the entering variable. To be more specific, we want to increase the value of* x_1 *until one of the basic variables* x_4, x_5, x_6, x_7 *reaches 0 (and thus becomes nonbasic),*

which is then the leaving variable.

Iteration 1. *We can do a* ratio test *based on the nonnegativity conditions:*

$$\begin{aligned} x_4 &= 3 - x_1 \geq 0, \\ x_5 &= 2 + x_1 \geq 0, \\ x_6 &= 4 - 2x_1 \geq 0, \\ x_7 &= 2 - 2x_1 \geq 0. \end{aligned}$$

The largest possible increase corresponds to the smallest ratio of the free coefficient to the absolute value of the coefficient for x_1 in the same row, assuming that the coefficient for x_1 is negative. As the ratios are $3, 2, 1$ for variables x_4, x_6, x_7, respectively, x_7 is then the leaving variable. We see that

$$x_1 = 1 - \frac{3}{2}x_2 + \frac{1}{2}x_3 - \frac{1}{2}x_7.$$

By substituting this expression for x_1 in the other rows, we obtain the following new solution:

$$\begin{array}{rcrcrcrl} z &=& 5 &-& \frac{5}{2}x_2 &+& \frac{11}{2}x_3 &- \frac{5}{2}x_7 \\ \hline x_1 &=& 1 &-& \frac{3}{2}x_2 &+& \frac{1}{2}x_3 &- \frac{1}{2}x_7, \\ x_4 &=& 2 &-& \frac{3}{2}x_2 &-& \frac{3}{2}x_3 &+ \frac{1}{2}x_7, \\ x_5 &=& 3 &-& \frac{3}{2}x_2 &-& \frac{5}{2}x_3 &- \frac{1}{2}x_7, \\ x_6 &=& 2 &+& 4x_2 &-& 3x_3 &+ x_7. \end{array}$$

Here, the basic variables are indicated by $B = \{1, 4, 5, 6\}$ and nonbasic variables by $N = \{2, 3, 7\}$. Now since we still have one variable x_3 with positive coefficient in the z-row, we continue this procedure by setting it to be the entering variable.

Iteration 2. *In the ratio test, we see that x_6 has the smallest ratio $\frac{2}{3}$ and thus should be the leaving variable. Using the relation*

$$x_3 = \frac{2}{3} + \frac{4}{3}x_2 - \frac{1}{3}x_6 + \frac{1}{3}x_7,$$

we get

$$\begin{array}{rcrcrcrl} z &=& \frac{26}{3} &+& \frac{29}{6}x_2 &-& \frac{2}{3}x_7 &- \frac{11}{6}x_6 \\ \hline x_3 &=& \frac{2}{3} &+& \frac{4}{3}x_2 &+& \frac{1}{3}x_7 &- \frac{1}{3}x_6, \\ x_1 &=& \frac{4}{3} &-& \frac{5}{6}x_2 &-& \frac{1}{3}x_7 &- \frac{1}{6}x_6, \\ x_4 &=& 1 &-& \frac{7}{2}x_2 & & &+ \frac{1}{2}x_6, \\ x_5 &=& \frac{4}{3} &-& \frac{29}{6}x_2 &-& \frac{4}{3}x_7 &+ \frac{5}{6}x_6. \end{array}$$

Now x_2 has a positive coefficient in the z-row, so we continue the procedure.

Iteration 3. *We determine the leaving variable through the ratio test and find x_5 is the one*

with the smallest ratio $\frac{8}{29}$. By expressing x_2 through x_5, x_6, x_7, we get

$$\begin{array}{rcrcrcrcr}
z & = & 10 & - & 2x_7 & - & x_6 & - & x_5 \\
\hline
x_2 & = & \frac{8}{29} & - & \frac{8}{29}x_7 & + & \frac{5}{29}x_6 & - & \frac{6}{29}x_5, \\
x_3 & = & \frac{30}{29} & - & \frac{1}{29}x_7 & - & \frac{3}{29}x_6 & - & \frac{8}{29}x_5, \\
x_1 & = & \frac{32}{29} & - & \frac{3}{29}x_7 & - & \frac{9}{29}x_6 & + & \frac{5}{29}x_5, \\
x_4 & = & \frac{1}{29} & + & \frac{28}{29}x_7 & - & \frac{3}{29}x_6 & + & \frac{21}{29}x_5.
\end{array}$$

All coefficients in the z-row are now negative. Hence, changing the value of any nonbasic variables will decrease the objective function value. We conclude that we have found an optimal solution for our LO problem, which has the value

$$x_1 = \frac{32}{29}, x_2 = \frac{8}{29}, x_3 = \frac{30}{29},$$

and all other variables being 0. The optimal objective value is $z = 10$.

The above computation procedure can be done more compactly in a *tableau* format. Without renaming the variables, we can also use BV_k and NV_k to indicate the basic and nonbasic variables in iteration k. We use the LO model from the baby carriers production problem as an example below.

Example 4.4. *Consider the LO problem*

$$\begin{array}{rrcrcrcrcrcrcl}
\max & z = 15x_1 & + & 25x_2 & & & & & & & & & \\
\text{s. t.} & x_1 & + & x_2 & + & s_1 & & & & & & = & 450, \\
& & & x_2 & & & + & s_2 & & & & = & 300, \\
& 4x_1 & + & 5x_2 & & & & & + & s_3 & & = & 2,000, \\
& x_1 & & & & & & & & & + \; s_4 & = & 350, \\
& & & & & & & & & x_1, x_2, s_1, s_2, s_3, s_4 & & \geq & 0.
\end{array}$$

We can write it in a tableau format as follows.

z	x_1	x_2	s_1	s_2	s_3	s_4	*rhs*	*basis*
1	−15	−25	0	0	0	0	0	z
0	1	1	1	0	0	0	450	s_1
0	0	1	0	1	0	0	300	s_2
0	4	5	0	0	1	0	2,000	s_3
0	1	0	0	0	0	1	350	s_4

Here, rhs stands for right-hand side values, and the basis stands for the basic variables corresponding to the rows. We use the convention that for the z-row, z is always in the basis and we revert the sign of the objective coefficients (by moving all variables in $z = 15x_1 + 25x_2$ to the

left-hand side). In this initial setup, the basic feasible solution (bfs) consists of basic variables $\text{BV}_0 = \{s_1, s_2, s_3, s_4\}$ *with values* $s_1 = 450$, $s_2 = 300$, $s_3 = 2,000$, $s_4 = 350$, *and nonbasic variables* $\text{NV}_0 = \{x_1, x_2\}$ *with values all being* 0.

Iteration 1. *To increase the value of* z, *we can increase the value of a nonbasic variable, which is called a* pivot variable, *and the corresponding column is called a* pivot column *in the tableau. As our objective function is linear, it is reasonable to select a nonbasic variable with the largest objective coefficient to be the pivot variable, which in this case is* x_2. *As before, we conduct a ratio test to detect the* pivot row *as follows.*

		↓							
z	x_1	x_2	s_1	s_2	s_3	s_4	*rhs*	*basis*	*ratio*	
1	−15	−25	0	0	0	0	0	z		
0	1	1	1	0	0	0	450	s_1	450	
0	0	1	0	1	0	0	300	s_2	300	←
0	4	5	0	0	1	0	2,000	s_3	400	
0	1	0	0	0	0	1	350	s_4	−	

Now we can perform elementary row operations involving the pivot row with the goal of turning all pivot column entries in the non-pivot rows into 0 *and the pivot row into* 1. *In our example, we multiply the pivot row by* 25, −1, *and* −5, *add the results to rows* 0, 1, *and* 3, *respectively.*

z	x_1	x_2	s_1	s_2	s_3	s_4	*rhs*	*basis*
1	−15	0	0	25	0	0	7,500	z
0	1	0	1	−1	0	0	150	s_1
0	0	1	0	1	0	0	300	x_2
0	4	0	0	−5	1	0	500	s_3
0	1	0	0	0	0	1	350	s_4

After the first iteration, the bfs consists of $\text{BV}_1 = \{s_1, x_2, s_3, s_4\}$ *with values* $s_1 = 150$, $x_2 = 300$, $s_3 = 500$, $s_4 = 350$, *with* $\text{NV}_1 = \{x_1, s_2\}$ *with values* 0. *The objective value* $z = 7,500$.

Iteration 2. *Now the variable* x_1 *has the largest coefficient in the objective, so we repeat the ratio test and find the pivot row.*

	↓								
z	x_1	x_2	s_1	s_2	s_3	s_4	*rhs*	*basis*	*ratio*	
1	−15	0	0	25	0	0	7,500	z		
0	1	0	1	−1	0	0	150	s_1	150	
0	0	1	0	1	0	0	300	x_2	−	
0	4	0	0	−5	1	0	500	s_3	125	←
0	1	0	0	0	0	1	350	s_4	350	

We perform the elementary row operations and get the updated tableau as follows.

z	x_1	x_2	s_1	s_2	s_3	s_4	*rhs*	*basis*
1	0	0	0	$25/4$	$15/4$	0	$9,375$	z
0	0	0	1	$1/4$	$-1/4$	0	25	s_1
0	0	1	0	1	0	0	300	x_2
0	1	0	0	$-5/4$	$1/4$	0	125	x_1
0	0	0	0	$5/4$	$-1/4$	1	225	s_4

Now the bfs consists of $\text{BV}_2 = \{s_1, x_2, x_1, s_4\}$ *with values* $s_1 = 25$, $x_2 = 300$, $x_1 = 125$, $s_4 = 225$, *and* $\text{NV}_2 = \{s_2, s_3\}$ *with values* 0. *The objective value* $z = 9,375$. *Note that increasing the value of any nonbasic variable would now decrease the objective value, which can be seen from the nonnegativity of the coefficients in the z-row of the tableau. We have thus found an optimal solution to the LO problem. From Figure 4.1, we see that the above algorithmic procedure corresponds to moving from point* $(0,0)$ *to* $(0,300)$, *and then to* $(125,300)$ *in the* (x_1, x_2)*-plane.*

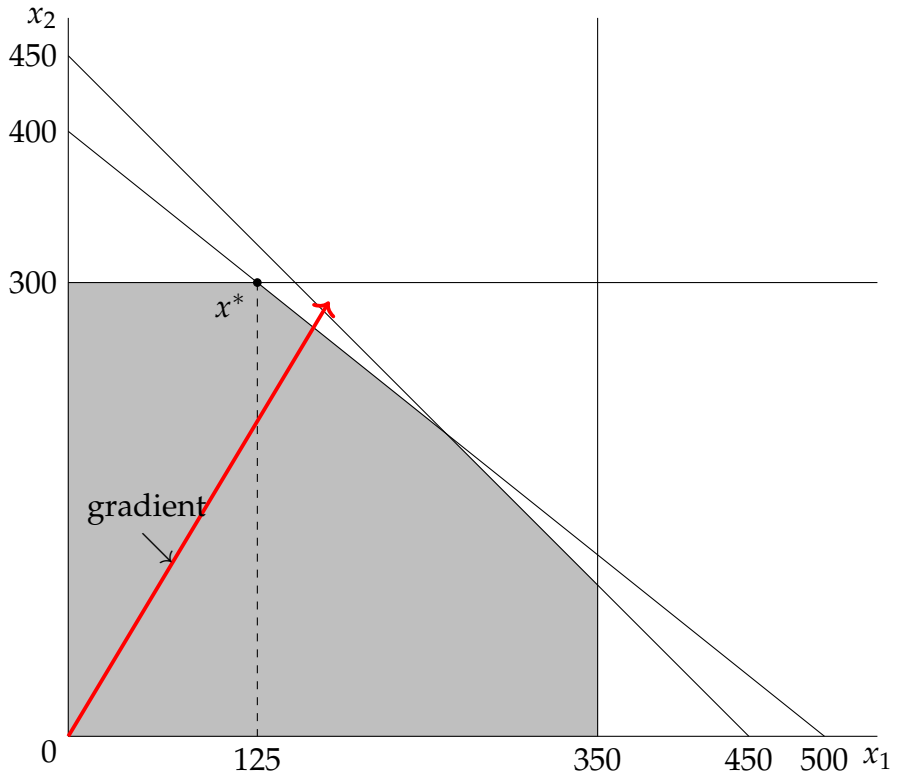


Figure 4.1: Feasible region and objective direction of Example 4.4

Remark. In a simplex tableau (of a LO maximization problem), the basic feasible solution is optimal if all nonbasic variables have nonnegative coefficients in the z-row.

Recall that a LO problem can be unbounded. We would like to detect unboundedness from the simplex tableau as described in the next example.

Example 4.5. *Consider the following tableau in a LO maximization problem.*

		↓						
z	x_1	x_2	s_1	s_2	s_3	s_4	*rhs*	*basis*
1	25	-4	0	0	0	0	90	z
0	14	-1	1	0	0	0	25	s_1
0	1	0	0	1	0	0	30	s_2
0	-5	-14	0	0	1	0	12	s_3
0	4	-7	0	0	0	1	22	s_4

We see that by increasing the value of the nonbasic variable x_2, the objective value should increase. However, there is no restriction on how much x_2 can increase (as no ratio test needs to be performed). This means that this LO problem is unbounded.

Remark. In a simplex tableau (of a LO maximization problem), the unboundedness is detected if there is a column with no positive entries.

4.3 Simplex Method Termination and Initialization

A natural question is whether simplex method can always find an optimal solution in finitely many steps/iterations. To answer this question, we need to note a special situation called *degeneracy*, where there is one or more basic variables equal to 0. Degeneracy (or *degenerate bfs*) may lead to the *cycling* phenomenon as in the next example.

Example 4.6. *Consider the LO problem*

$$\begin{aligned}
\max \quad & 5x_1 + 4x_2 - 20x_3 - 2x_4 \\
\text{s.t.} \quad & \tfrac{1}{4}x_1 - \tfrac{1}{8}x_2 + 12x_3 + 10x_4 \le 0, \\
& \tfrac{1}{10}x_1 + \tfrac{1}{20}x_2 + \tfrac{1}{20}x_3 + \tfrac{1}{5}x_4 \le 0, \\
& x_1, x_2, x_3, x_4 \ge 0.
\end{aligned}$$

The initial (iteration 0) tableau can be written as follows.

z	x_1	x_2	x_3	x_4	x_5	x_6	*rhs*	*basis*
1	-5	-4	20	2	0	0	0	z
0	$\boxed{\frac{1}{4}}$	$-\frac{1}{8}$	12	10	1	0	0	x_5
0	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{20}$	$\frac{1}{5}$	0	1	0	x_6

We continue the simplex method.

- *Iteration 1*

z	x_1	x_2	x_3	x_4	x_5	x_6	rhs	$basis$
1	0	$-\frac{13}{2}$	260	202	20	0	0	z
0	1	$-\frac{1}{2}$	48	40	4	0	0	x_1
0	0	$\boxed{\frac{1}{10}}$	$-\frac{19}{4}$	$-\frac{19}{5}$	$-\frac{2}{5}$	1	0	x_6

- *Iteration 2*

z	x_1	x_2	x_3	x_4	x_5	x_6	rhs	$basis$
1	0	0	$-\frac{195}{4}$	-45	-6	65	0	z
0	1	0	$\boxed{\frac{97}{4}}$	21	2	5	0	x_1
0	0	1	$-\frac{95}{2}$	-38	-4	10	0	x_2

- *Iteration 3*

z	x_1	x_2	x_3	x_4	x_5	x_6	rhs	$basis$
1	$\frac{195}{97}$	0	0	$-\frac{270}{97}$	$-\frac{192}{97}$	$\frac{7280}{97}$	0	z
0	$\frac{190}{97}$	1	0	$\boxed{\frac{304}{97}}$	$-\frac{8}{97}$	$\frac{1920}{97}$	0	x_2
0	$\frac{4}{97}$	0	1	$\frac{84}{97}$	$\frac{8}{97}$	$\frac{20}{97}$	0	x_3

- *Iteration 4*

z	x_1	x_2	x_3	x_4	x_5	x_6	rhs	$basis$
1	$\frac{15}{4}$	$\frac{135}{152}$	0	0	$-\frac{39}{19}$	$\frac{1760}{19}$	0	z
0	$-\frac{1}{2}$	$-\frac{21}{76}$	1	0	$\boxed{\frac{2}{19}}$	$-\frac{100}{19}$	0	x_3
0	$\frac{5}{8}$	$\frac{97}{304}$	0	1	$-\frac{1}{38}$	$\frac{120}{19}$	0	x_4

- *Iteration 5*

z	x_1	x_2	x_3	x_4	x_5	x_6	rhs	$basis$
1	-6	$-\frac{9}{2}$	$\frac{39}{2}$	0	0	-10	0	z
0	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	1	0	$\boxed{5}$	0	x_4
0	$-\frac{19}{4}$	$-\frac{21}{8}$	$\frac{19}{2}$	0	1	-50	0	x_5

- *Iteration 6*

z	x_1	x_2	x_3	x_4	x_5	x_6	rhs	$basis$
1	-5	-4	20	2	0	0	0	z
0	$\boxed{\frac{1}{4}}$	$-\frac{1}{8}$	12	10	1	0	0	x_5
0	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{20}$	$\frac{1}{5}$	0	1	0	x_6

Note that in iteration 6, we get the same tableau as we got in iteration 0. Thus if we continue with the execution of the simplex method, we will keep repeating the calculations in iterations 0-6 and will never be able to leave the same solution.

To avoid cycling in the simplex method, we can use certain *pivoting rule*, which determines the pivoting variable at every degenerate solution. A conceptually useful rule is called *Bland's rule*: assuming that the variables are indexed, for example, by $1, \ldots, n$, we always choose the entering or leaving variable with the smallest index. We illustrate Bland's rule by applying it to Example 4.6.

Example 4.6 (continued)**.** *Applying Bland's rule leads to the same first 5 iterations, at the end of which we have the following tableau.*

z	x_1	x_2	x_3	x_4	x_5	x_6	*rhs*	*basis*
1	-6	$-\frac{9}{2}$	$\frac{39}{2}$	0	0	-10	0	z
0	$\boxed{\frac{1}{2}}$	$\frac{1}{4}$	$\frac{1}{4}$	1	0	5	0	x_4
0	$-\frac{19}{4}$	$-\frac{21}{8}$	$\frac{19}{2}$	0	1	-50	0	x_5

Now the candidates for the entering variables are x_1, x_2, and x_6, so by Bland's rule we choose x_1 to enter the basis.

z	x_1	x_2	x_3	x_4	x_5	x_6	*rhs*	*basis*
1	0	$-\frac{3}{2}$	$\frac{45}{2}$	12	0	50	0	z
0	1	$\boxed{\frac{1}{2}}$	$\frac{1}{2}$	2	0	10	0	x_1
0	0	$-\frac{1}{4}$	$\frac{95}{8}$	$\frac{19}{2}$	1	$-\frac{5}{2}$	0	x_5

After one more iteration, we see optimality from the tableau.

z	x_1	x_2	x_3	x_4	x_5	x_6	*rhs*	*basis*
1	3	0	24	18	0	80	0	z
0	2	1	1	4	0	20	0	x_2
0	$\frac{1}{2}$	0	$\frac{97}{8}$	$\frac{21}{2}$	1	$\frac{5}{2}$	0	x_5

In fact, the solution value is the same as the initial bfs, but the last tableau shows its optimality.

It can be shown that with Bland's rule, simplex method does not have the cycling phenomenon. The proof is a little involved, so we do not go into the details here. In practice, Bland's rule may not lead to efficient implementation of the simplex method.

Other pivoting rules may be more favorable, such as the *steepest edge rule*, the *random edge rule*, or the *lexicographic rule*. The details can be found in [**MatousekGaertner2007**].

Without cycling, the simplex method is guaranteed to terminate in finitely many iteration (with either an optimal bfs or a certificate for unboundedness). In fact, given a standard LO form with n variables and m constraints, there are at most $\binom{n}{m}$ possible simplex tableaus. This is because each tableau has a unique set of basic variables, the cardinality (i.e., size) of which is exactly m. By definition, we will not see the same basis twice unless cycling happens.

Theorem 4.7. *If the simplex method does not have cycling, then it terminates with at most* $\binom{n}{m}$ *iterations.*

Another issue is regarding how to find feasible solutions to the LO problem. We have seen that if we start with a bfs, then the simplex method can keep feasibility through the ratio tests. Therefore, it suffices for us to discuss initialization procedures for feasible solutions. Here, we describe two methods: the *two-phase* simplex method and the *big-M* simplex method.

The main idea of both methods is to "relax" the constraints by introducing artificial variables, and then try to find solutions where the artificial variables are all zero. To be precise, for a standard form LO problem

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ji} x_i = b_j, \quad j = 1, \dots, m, \\ & x_i \geq 0, \quad i = 1, \dots, n, \end{aligned} \tag{4.6}$$

we define the following index subsets of $J := \{1, \dots, m\}$: $J^+ := \{j \in J : b_j \geq 0\}$, and $J^- := \{j \in J : b_j < 0\}$, representing the constraints that have positive or no violations, and those that have negative violations if we set $x = 0$. Then the two-phase method first solves the first-phase feasibility problem

$$\begin{aligned} \min \quad & \sum_{j \in J} u_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ji} x_i + u_j = b_j, \quad \forall\, j \in J^+, \\ & \sum_{i=1}^{n} a_{ji} x_i - u_j = b_j, \quad \forall\, j \in J^-, \\ & x_i \geq 0, \quad i = 1, \dots, n, \\ & u_j \geq 0, \quad j = 1, \dots, m. \end{aligned} \tag{4.7}$$

Clearly, if we set $x = 0$ and $u_j = |b_j|$ for $j = 1, \ldots, m$, then (x, u) is a bfs to the problem (4.7). Moreover, the original problem (4.6) is feasible if and only if (4.7) has an optimal basic solution (x', u') where $u'_1, \ldots, u'_m$ are all nonbasic variables. In this case, the objective value of (4.7) will also be 0, meaning that we do not need these artificial variables to make the original problem (4.6) feasible. Then we can use the solution x' as a bfs to (4.6). In practice, if an equality constraint j in (4.6) is converted from an inequality constraint, then we can directly take the artificial variable u_j to be the slack or excess variable. We illustrate the two-phase simplex method by the following example.

Example 4.8. *Consider the following LO problem:*

$$\begin{array}{lrcrcl} \max & 5x_1 & + & 10x_2 & & \\ \text{s.t.} & 2x_1 & + & x_2 & = & 4 \\ & x_1 & + & 2x_2 & \leq & 5 \\ & & & x_1, x_2 & \geq & 0. \end{array}$$

We can convert into the standard form (by introducing the slack variable $s_2 \geq 0$) and then construct the first-phase feasibility problem as

$$\begin{array}{lrcrcrcl} \max & & & & & - & a_1 & \\ \text{s.t.} & 2x_1 & + & x_2 & + & a_1 & = & 4 \\ & x_1 & + & 2x_2 & + & s_2 & = & 5 \\ & & & x_1, x_2, s_2, a_1 & & & \geq & 0. \end{array}$$

We start with the bfs $(x_1, x_2, s_2, a_1) = (0, 0, 5, 4)$ and use the simplex method as follows.

- *Iteration 0 (raw)*

z	x_1	x_2	a_1	s_2	*rhs*	*basis*
1	0	0	***1***	0	0	z
0	2	1	1	0	4	a_1
0	1	2	0	1	5	s_2

  As a basic variable, a_1 should not *have a nonzero coefficient in the z-row. We need to get a correct tableau by a row operation.*

- *Iteration 0*

z	x_1	x_2	a_1	s_2	*rhs*	*basis*
1	-2	-1	0	0	-4	z
0	$\boxed{2}$	1	1	0	4	a_1
0	1	2	0	1	5	s_2

- *Iteration 1*

z	x_1	x_2	a_1	s_2	*rhs*	*basis*
1	0	0	1	0	***0***	z
0	1	1/2	1/2	0	2	x_1
0	0	3/2	−1/2	1	3	s_2

The tableau is optimal and we can check that $(x_1, x_2, s_2) = (2, 0, 3)$ *is a bfs of the original problem. Thus the first-phase feasibility problem is solved.*

We start the second phase as follows.

- *Iteration 0 (raw)*

z	x_1	x_2	s_2	*rhs*	*basis*
1	***-5***	−10	0	0	z
0	1	1/2	0	2	x_1
0	0	3/2	1	3	s_2

As x_1 *is a basic variable, its coefficient in the z-row must be zero. We can correct this again by a row operation.*

- *Iteration 0*

z	x_1	x_2	s_2	*rhs*	*basis*
1	0	−15/2	0	10	z
0	1	1/2	0	2	x_1
0	0	$\boxed{3/2}$	1	3	s_2

The entering variable is x_2 *and the ratio test determines that* s_2 *should be leaving.*

- *Iteration 1*

z	x_1	x_2	s_2	*rhs*	*basis*
1	0	0	5	25	z
0	1	0	−1/3	1	x_1
0	0	1	2/3	2	x_2

The tableau is optimal. We have solved the second phase problem.

From the last tableau, we can see that an optimal solution is $(x_1^*, x_2^*) = (1, 2)$*, with the optimal value* $z^* = 25$*.*

The big-M method defines a big coefficient $M \gg 0$ for the artificial variables in the objective function. To be precise, for the problem (4.6), we pick a large constant $M > 0$

and solve the following problem

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} c_i x_i - M \sum_{j=1}^{m} u_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ji} x_i + u_j = b_j, \quad \forall\, j \in J^+, \\ & \sum_{i=1}^{n} a_{ji} x_i - u_j = b_j, \quad \forall\, j \in J^-, \\ & x_i \geq 0, \quad i = 1, \ldots, n, \\ & u_j \geq 0, \quad j = 1, \ldots, m. \end{aligned} \tag{4.8}$$

The subsets J^+ and J^- are defined similarly as in the two-phase method, so we can choose the obvious bfs $x_i = 0$ for $i = 1, \ldots, n$ and $u_j = |b_j|$ for $j = 1, \ldots, m$, as our starting point. We say that M is sufficiently large, if in the simplex iterations, the sign of any linear expression involving M would only depend on the coefficient of M. For example, $-M + 10 < 0$ as M has a coefficient of -1, and $2M + 30 > M + 100$ as the coefficient of M on the left-hand side is 2, which is greater than 1, the coefficient of M on the right-hand side. It can be shown that assuming M is sufficiently large,

- if the problem (4.8) has an optimal solution (x^*, u^*) with $u^* = 0$, then x^* is an optimal solution to (4.6);
- if the problem (4.8) has an optimal solution (x^*, u^*) with $u^* \neq 0$, (4.6) is infeasible;
- if the problem (4.8) is unbounded, then so is (4.6).

We illustrate below the big-M method on the same problem that appeared in Example 4.8.

Example 4.9. *For the problem*

$$\begin{array}{rrrrrrr} \max & 5x_1 & + & 10x_2 & & & \\ \text{s.t.} & 2x_1 & + & x_2 & = & 4 \\ & x_1 & + & 2x_2 & \leq & 5 \\ & & & x_1, x_2 & \geq & 0, \end{array}$$

the big-M formulation is

$$\begin{array}{rrrrrrrrr} \max & 5x_1 & + & 10x_2 & -Ma_1 & & & & \\ \text{s.t.} & 2x_1 & + & x_2 & + & a_1 & = & 4 \\ & x_1 & + & 2x_2 & + & s_2 & = & 5 \\ & & & & x_1, x_2, s_2, a_1 & & \geq & 0. \end{array}$$

- *Iteration 0 (raw)*

z	x_1	x_2	a_1	s_2	rhs	$basis$
1	-5	-10	M	0	0	z
0	2	1	1	0	4	a_1
0	1	2	0	1	5	s_2

We need to eliminate the coefficients of basic variables in the z-row through row operations.

- *Iteration 0*

z	x_1	x_2	a_1	s_2	rhs	$basis$
1	$-5-2M$	$-10-M$	0	0	$-4M$	z
0	2	1	1	0	4	a_1
0	1	2	0	1	5	s_2

Here, x_1 should be the entering variable because $-5-2M < -10-M$ for sufficiently large M. By the ratio test, a_1 should be leaving.

- *Iteration 1*

z	x_1	x_2	a_1	s_2	rhs	$basis$
1	0	$-15/2$	$5/2+M$	0	10	z
0	1	$1/2$	$1/2$	0	2	x_1
0	0	$\boxed{3/2}$	$-1/2$	1	3	s_2

Now x_2 enters and s_2 leaves the basis.

- *Iteration 2*

z	x_1	x_2	a_1	s_2	rhs	$basis$
1	0	0	M	5	25	z
0	1	0	$2/3$	$-1/3$	1	x_1
0	0	1	$-1/3$	$2/3$	2	x_2

We terminate the simplex method as all coefficients in the z-row are nonnegative.

We have found an optimal solution $(x_1^*, x_2^*) = (1, 2)$. It is feasible to the original problem because $a_1^* = 0$ is nonbasic. The optimal value is $z^* = 25$.

4.4 Simplex Method in the Matrix Form

It may be conceptually simpler to consider the simplex method in the following matrix form. The standard form can be written as

$$\begin{aligned} \max \quad & c^\mathsf{T} x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0. \end{aligned} \tag{4.9}$$

By using the index sets B (for basic variables) and N (for nonbasic variables), we use the subscript notation to indicate the data associated with these variables. For example, if $B = \{1, 3, 4\}$, then $c_B = (c_1, c_3, c_4)$ is a vector consisting of components of the vector c with indices in B. Thus given B and N, the standard form can be rewritten as

$$\begin{aligned} \max \quad & [c_B^\mathsf{T}, c_N^\mathsf{T}] \begin{bmatrix} x_B \\ x_N \end{bmatrix} \\ \text{s.t.} \quad & [A_B, A_N] \begin{bmatrix} x_B \\ x_N \end{bmatrix} = b, \\ & \begin{bmatrix} x_B \\ x_N \end{bmatrix} \geq 0. \end{aligned} \iff \begin{aligned} \max \quad & c_B^\mathsf{T} x_B + c_N^\mathsf{T} x_N \\ \text{s.t.} \quad & A_B x_B + A_N x_N = b, \\ & x_B \geq 0,\ x_N \geq 0. \end{aligned} \tag{4.10}$$

Then each iteration of the simplex method can be written in the *matrix form*:

$$\begin{array}{rcccc} z & = & c_B^\mathsf{T} A_B^{-1} b & + & r^\mathsf{T} x_N \\ \hline x_B & = & p & + & Q x_N \end{array} \tag{4.11}$$

where $p = A_B^{-1} b$, $Q = [q_{ij}] = -A_B^{-1} A_N$, and the vector $r := c_N - (c_B^\mathsf{T} A_B^{-1} A_N)^\mathsf{T}$ is sometimes called the *reduced cost*. Recall that the definition of a basis requires the submatrix A_B to be nonsingular, which ensures that the inverse A_B^{-1} is well-defined. Equivalently, we can write the tableau in each iteration as

z	x_B	x_N	rhs
1	0	$-r$	$c_B^\mathsf{T} A_B^{-1} b$
0	I	$-Q$	p

(4.12)

where I in the tableau is the m-by-m identity matrix.

The optimality criterion for the simplex method (for a LO maximization problem) can be restated as $r \leq 0$, and if any $r_k > 0$, then x_k should be the entering variable. The leaving variable is determined by the ratio test, i.e., any x_j such that

$$q_{jk} < 0 \quad \text{and} \quad -\frac{p_j}{q_{jk}} = \min \left\{ -\frac{p_i}{q_{ik}} : q_{ik} < 0,\ i = 1, \ldots, m \right\}. \tag{4.13}$$

The unboundedness can be detected if for all $i = 1, \ldots, m$, $q_{ik} \geq 0$. Otherwise, we set $B \leftarrow B \cup \{j\} \setminus \{k\}$ and continue. We use the data in Example 4.3 to illustrate the procedure in the matrix form.

Example 4.10. *We write the standard form of the LO problem*

$$\begin{array}{rrcrcrcl} \max & 5x_1 & + & 5x_2 & + & 3x_3 & & \\ \text{s.t.} & x_1 & + & 3x_2 & + & x_3 & \leq & 3, \\ & -x_1 & & & + & 3x_3 & \leq & 2, \\ & 2x_1 & - & x_2 & + & 2x_3 & \leq & 4, \\ & 2x_1 & + & 3x_2 & - & x_3 & \leq & 2, \\ & & & & & x_1, x_2, x_3 & \geq & 0. \end{array}$$

with slack variables $x_4, x_5, x_6, x_7 \geq 0$ through the matrix and the vectors

$$A = \begin{bmatrix} 1 & 3 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 3 & 0 & 1 & 0 & 0 \\ 2 & -1 & 2 & 0 & 0 & 1 & 0 \\ 2 & 3 & -1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 2 \\ 4 \\ 2 \end{bmatrix}, \quad c = \begin{bmatrix} 5 \\ 5 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Initially, we set $B \leftarrow B_0 = \{4,5,6,7\}$ and $N \leftarrow N_0 = \{1,2,3\}$, then we have

$$A_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_N = \begin{bmatrix} 1 & 3 & 1 \\ -1 & 0 & 3 \\ 2 & -1 & 2 \\ 2 & 3 & -1 \end{bmatrix}, \quad c_B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad c_N = \begin{bmatrix} 5 \\ 5 \\ 3 \end{bmatrix}.$$

In particular,

$$p = A_B^{-1}b = \begin{bmatrix} 3 \\ 2 \\ 4 \\ 2 \end{bmatrix}, \quad Q = -A_B^{-1}A_N = \begin{bmatrix} -1 & -3 & -1 \\ 1 & 0 & -3 \\ -2 & 1 & -2 \\ -2 & -3 & 1 \end{bmatrix}, \quad r = c_N - (c_B^{\mathsf{T}} A_B^{-1} A_N)^{\mathsf{T}} = \begin{bmatrix} 5 \\ 5 \\ 3 \end{bmatrix}.$$

Iteration 1. *We pick x_1 as an entering variable, and the ratio test determines that x_7 is a leaving variable, Thus $B_1 = B_0 \cup \{1\} \setminus \{7\} = \{1,4,5,6\}$, and consequently $N_1 = \{2,3,7\}$. Now setting $B \leftarrow B_1$ and $N \leftarrow N_1$, we get*

$$p = A_B^{-1}b = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \end{bmatrix}, \quad Q = -A_B^{-1}A_N = \begin{bmatrix} -\frac{3}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ -\frac{3}{2} & -\frac{5}{2} & -\frac{1}{2} \\ 4 & -3 & 1 \end{bmatrix}, \quad r = c_N - (c_B^{\mathsf{T}} A_B^{-1} A_N)^{\mathsf{T}} = \begin{bmatrix} -\frac{5}{2} \\ \frac{11}{2} \\ -\frac{5}{2} \end{bmatrix}.$$

Iteration 2. *Now x_3 is the new entering variable, and by the ratio test, x_6 is leaving. Thus $B_2 = B_1 \cup \{3\} \setminus \{6\} = \{1,3,4,5\}$, and consequently $N_2 = \{2,6,7\}$. Now setting $B \leftarrow B_2$ and $N \leftarrow N_2$, we get*

$$p = A_B^{-1}b = \begin{bmatrix} \frac{2}{3} \\ \frac{4}{3} \\ 1 \\ \frac{4}{3} \end{bmatrix}, \quad Q = -A_B^{-1}A_N = \begin{bmatrix} \frac{4}{3} & -\frac{1}{3} & \frac{1}{3} \\ -\frac{5}{6} & -\frac{1}{6} & -\frac{1}{3} \\ -\frac{7}{2} & +\frac{1}{2} & 0 \\ -\frac{29}{6} & \frac{5}{6} & -\frac{4}{3} \end{bmatrix}, \quad r = c_N - (c_B^\mathsf{T} A_B^{-1} A_N)^\mathsf{T} = \begin{bmatrix} \frac{29}{6} \\ -\frac{11}{6} \\ -\frac{2}{3} \end{bmatrix}.$$

Iteration 3. *The variable x_2 is entering and by the ratio test x_5 is leaving. Thus $B_3 = B_2 \cup \{2\} \setminus \{5\} = \{1,2,3,4\}$, and consequently $N_2 = \{5,6,7\}$. Now setting $B \leftarrow B_3$ and $N \leftarrow N_3$, we get*

$$p = A_B^{-1}b = \begin{bmatrix} \frac{8}{29} \\ \frac{30}{29} \\ \frac{32}{29} \\ \frac{1}{29} \end{bmatrix}, \quad Q = -A_B^{-1}A_N = \begin{bmatrix} -\frac{6}{29} & \frac{5}{29} & -\frac{8}{29} \\ -\frac{8}{29} & -\frac{3}{29} & -\frac{1}{29} \\ \frac{5}{29} & -\frac{9}{29} & -\frac{3}{29} \\ \frac{21}{29} & -\frac{3}{29} & \frac{28}{29} \end{bmatrix}, \quad r = c_N - (c_B^\mathsf{T} A_B^{-1} A_N)^\mathsf{T} = \begin{bmatrix} -1 \\ -1 \\ -2 \end{bmatrix}.$$

As $r \leq 0$, we have found an optimal solution $x^* = (32/29, 8/29, 30/29, 0, 0, 0, 0)$ with the optimal value $z^* = 10$.

5 Linear Optimization IV: Duality and Sensitivity

5.1 Dual Linear Optimization Formulation

We now discuss *duality*, which is perhaps one of the most important concepts in mathematical optimization. The following example gives a motivation for our discussion.

Example 5.1. *A carpenter makes tables and chairs for sale. A total of 150 board ft of oak and 250 board ft of pine are available. A table requires 16 board ft of oak and 30 board ft of pine, while a chair requires 5 board ft of oak and 12 board ft of pine. Each table can be sold for \$40, and each chair for \$15. Assuming that the tables and chairs can be produced in fractions, we can define x_1,x_2 to be the number of tables and chairs to make, respectively, and write a LO model to maximize the revenue for the carpenter*

$$\begin{aligned} \max \quad & 40x_1 + 15x_2 \\ \text{s.t.} \quad & 16x_1 + 5x_2 \leq 150, \quad \textit{(oak constraint)} \\ & 30x_1 + 12x_2 \leq 250, \quad \textit{(pine constraint)} \\ & x_1, x_2 \geq 0. \end{aligned}$$

Now suppose the carpenter can sell any unused oak and pine to a wood company, and also buy

additional oak and pine from it at the price of y_1 and y_2, respectively. How should the wood company set the prices such that the carpenter would have the same revenue as before? To answer this question, we can formulate this as a min-max problem

$$\min_{y_1,y_2\geq 0} \quad \max_{x_1,x_2\geq 0} \quad 40x_1 + 15x_2 + (150 - 16x_1 - 5x_2)y_1 + (250 - 30x_1 - 12x_2)y_2.$$

The min-max (or sometimes minimax*) problem lets the wood company determine the price first, and then let carpenter decides the production decisions, which imply the buying/selling strategies of the oak and pine. In game theory terminology, we are trying to find the* equilibrium *prices between the carpenter and the wood company. Note that the inner maximization problem can be written as*

$$\max_{x_1,x_2\geq 0} \quad 150y_1 + 250y_2 + (40 - 15y_1 - 30y_2)x_1 + (15 - 5y_1 - 12y_2)x_2.$$

If the either of the coefficients $40 - 15y_1 - 30y_2$ and $15 - 5y_1 - 12y_2$ is positive, then we can simply increase the value of x_1 or x_2 to make the objective value arbitrarily large. Thus we should have both coefficients being nonpositive. Consequently, we have an obvious solution to the inner maximization problem $x_1^* = x_2^* = 0$, so the min-max problem reduces to

$$\begin{aligned} \min \quad & 150y_1 + 250y_2 \\ \text{s. t.} \quad & 15y_1 + 30y_2 \geq 40, \quad \textit{(nonpositive coefficient of } x_1\textit{)} \\ & 5y_1 + 12y_2 \geq 15, \quad \textit{(nonpositive coefficient of } x_2\textit{)} \\ & y_1, y_2 \geq 0. \end{aligned}$$

This minimization problem is called the dual *problem to the original maximization problem. We will see below that solving the dual problem gives us prices that let the revenue of the carpenter stays the same as before.*

In general, we can formulate dual linear optimization (LO) problems based on the idea of *relaxation*. Consider a LO problem with some given $m' \leq m$ and $n' \leq n$:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s. t.} \quad & \sum_{i=1}^{n} a_{ji} x_i \geq b_j, \quad \forall\, j = 1, \ldots, m', \\ & \sum_{i=1}^{n} a_{ji} x_i = b_j, \quad \forall\, j = m'+1, \ldots, m, \\ & x_i \geq 0, \quad i = 1, \ldots, n', \\ & x_i \in \mathbb{R}, \quad i = n'+1, \ldots, n. \end{aligned} \tag{5.1}$$

We can relax all the constraints (both equalities and inequalities) and penalize the violation with a price vector $y \in \mathbb{R}^m$ as

$$\min \quad \sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{m} \left(b_j - \sum_{i=1}^{n} a_{ji} x_i \right) y_j. \tag{5.2}$$

The first part here is the original objective function, while the second part is given by the penalty on any constraint violation. Since we are minimizing the objective value, we would like to have a positive penalty $(b_j - \sum_{i=1}^{n} a_{ji} x_i) y_j > 0$ only when there is violation of this constraint, i.e., $\sum_{i=1}^{n} a_{ji} x_i < b_j$ for any $j \le m'$, or $\sum_{i=1}^{n} a_{ji} x_i \neq b_j$ for any $j \ge m' + 1$. Thus we should set y_j to be nonnegative for each $j \le m'$ and unrestricted in sign for $j \ge m' + 1$. Now we want to find a *tightest* relaxation, in the sense that the objective value should be as close to the original objective value as possible. Naturally this leads to maximization in the variables $y_1, \ldots, y_m$. By rearranging terms in (5.2), we can write the tightest relaxation problem as

$$\max_{\substack{y_1,\ldots,y_{m'} \ge 0 \\ y_{m'+1},\ldots,y_m \in \mathbb{R}}} \quad \min_{\substack{x_1,\ldots,x_{n'} \ge 0 \\ x_{n'+1},\ldots,x_n \in \mathbb{R}}} \quad \sum_{i=1}^{n} \left(c_i - \sum_{j=1}^{m} a_{ji} y_j \right) x_i + \sum_{j=1}^{m} b_j y_j. \tag{5.3}$$

Note that if $c_i - \sum_{j=1}^{m} a_{ji} y_j < 0$ for some $i \le n'$, then the inner minimization would be unbounded (as we can take x_i to be arbitrarily large). The same behavior happens if $c_i - \sum_{j=1}^{m} a_{ji} y_j \neq 0$ for some $i \ge n' + 1$, because x_i is not restricted in sign. Therefore, we can impose the constraints

$$\begin{aligned} \sum_{j=1}^{m} a_{ji} y_j &\le c_i, \quad i = 1, \ldots, n', \\ \sum_{j=1}^{m} a_{ji} y_j &= c_i, \quad i = n' + 1, \ldots, n, \end{aligned} \tag{5.4}$$

in the outer maximization without any compromise of its optimality. Consequently, the first summation in (5.3) vanishes and the inner minimization becomes trivial. We

have therefore derived our *dual* problem, i.e., finding the tightest relaxation, as

$$\begin{aligned} \max \quad & \sum_{j=1}^{m} b_j y_j \\ \text{s.t.} \quad & \sum_{j=1}^{m} a_{ji} y_j \le c_i, \quad i = 1, \dots, n', \\ & \sum_{j=1}^{m} a_{ji} y_j = c_i, \quad i = n'+1, \dots, n, \\ & y_j \ge 0, \quad j = 1, \dots, m', \\ & y_j \in \mathbb{R}, \quad j = m'+1, \dots, m. \end{aligned} \tag{5.5}$$

To distinguish the problems, we also call the original problem (5.1) the *primal* problem. The derivation above is known as *Lagrangian duality* and has been used beyond LO problems, e.g., in some nonlinear optimization or integer optimization problems as well. To save the effort of deriving the dual formulation from scratch each time, we summarize the correspondence of the sign restrictions and inequality directions for LO problems in Table 1.

Table 1: Dual correspondence of constraints and variables

Minimization		Maximization
$=$ constraint	$\longleftrightarrow$	free variable
$\ge$ constraint	$\longleftrightarrow$	nonnegative variable
$\le$ constraint	$\longleftrightarrow$	nonpositive variable
free variable	$\longleftrightarrow$	$=$ constraint
nonnegative variable	$\longleftrightarrow$	$\le$ constraint
nonpositive variable	$\longleftrightarrow$	$\ge$ constraint

5.2 Weak and Strong Duality

For notational convenience, we consider matrix forms of the primal and the dual problems:

$$\begin{aligned} \min \quad & c^\mathsf{T} x \\ \text{s.t.} \quad & Ax = b, \\ & x \ge 0, \end{aligned} \tag{P}$$

and

$$\begin{aligned} \max \quad & b^\mathsf{T} y \\ \text{s.t.} \quad & A^\mathsf{T} y \le c, \\ & y \in \mathbb{R}^m. \end{aligned} \tag{D}$$

We can see that for any primal feasible solution $x = (x_1, \dots, x_n)$ and dual feasible

solution $y = (y_1, \ldots, y_m)$, we have

$$b^\mathsf{T} y = (Ax)^\mathsf{T} y = x^\mathsf{T}(A^\mathsf{T} y) \leq c^\mathsf{T} x. \tag{5.6}$$

Thus if the primal problem (P) admits an optimal objective value z^* with an optimal solution x^*, then

$$b^\mathsf{T} y \leq c^\mathsf{T} x^* = z^*, \tag{5.7}$$

for any dual feasible solution $y \in \mathbb{R}^m$, which means the dual problem (D) is bounded. Similarly, if the dual problem (D) admits an optimal objective value w^* with an optimal solution y^*, then

$$w^* = b^\mathsf{T} y^* \leq c^\mathsf{T} x, \tag{5.8}$$

for any primal feasible solution $x \geq 0$, which means the primal problem (P) is bounded. In the case where both the primal and the dual problems have optimal solutions, x^* and y^*, respectively, we have the inequality

$$w^* = b^\mathsf{T} y^* \leq c^\mathsf{T} x^* = z^*. \tag{5.9}$$

This is called the *weak duality* of LO problems. We may further extend our discussion to some infeasible or unbounded LO problems. Recall that we can set $z^* = +\infty$ (resp. $z^* = -\infty$) if the primal minimization problem (D) is infeasible (resp. unbounded), and $w^* = -\infty$ (resp. $w^* = +\infty$) if the dual maximization problem (D) is infeasible (resp. unbounded). If the primal problem is unbounded, then for any dual feasible solution y, there exists a primal feasible solution x such that

$$b^\mathsf{T} y > c^\mathsf{T} x,$$

which contradicts with the inequality (5.9). Thus the dual problem must be infeasible, in which case we may write $z^* = w^* = -\infty$. Alternatively, if the dual problem is unbounded, then by the same argument we must have an infeasible primal problem, in which case we may write $z^* = w^* = +\infty$. In fact, the equality $z^* = w^*$ holds generally for feasible primal and dual problems as well. This is known as the *strong duality* result for LO problems.

Proposition 5.2. *If the primal problem* (P) *has an optimal solution x^*, then the dual problem* (D) *has an optimal solution y^* with $c^\mathsf{T} x^* = b^\mathsf{T} y^*$.*

Proof. By assumption, the primal problem (P) is feasible and bounded. By the simplex method (with Bland's pivot rule), there exists a basic feasible solution $x = (x_B, x_N)$, for some basis $B \subset \{1, \ldots, n\}$ and $N := \{1, \ldots, n\} \setminus B$, such that $x_B = A_B^{-1} b$ and $x_N = 0$. Obviously, $c^\mathsf{T} x^* = c^\mathsf{T} x$ as both are optimal primal solutions. Let $y^* := (c_B^\mathsf{T} A_B^{-1})^\mathsf{T}$. We

first check that y^* is a feasible solution to the dual problem (D):

$$A^\mathsf{T} y^* = \begin{bmatrix} A_B^\mathsf{T} \\ A_N^\mathsf{T} \end{bmatrix} (c_B^\mathsf{T} A_B^{-1})^\mathsf{T} = \begin{bmatrix} c_B \\ (c_B^\mathsf{T} A_B^{-1} A_N)^\mathsf{T} \end{bmatrix} \leq \begin{bmatrix} c_B \\ c_N \end{bmatrix} = c.$$

Here the last inequality is ensured by optimality of x, where the reduced cost vector $r = c_N - (c_B^\mathsf{T} A_B^{-1} A_N)^\mathsf{T} \geq 0$ (note that it is minimization). Then $b^\mathsf{T} y^* = c_B^\mathsf{T} A_B^{-1} b = c_B^\mathsf{T} x_B = c^\mathsf{T} x$, which implies that y^* is an optimal solution to the dual problem (D) by the weak duality (5.9). □

Using the symmetry between the primal and the dual problems, Proposition 5.2 tells us both have the same optimal value as long as one of them has an optimal solution. It remains to ask the possible outcomes of the primal and the dual problems if we know one of them is infeasible. The next example shows that it is possible for both of the problems to be infeasible.

Example 5.3. *Let*

$$A = \begin{bmatrix} 2 & -1 \\ -2 & 1 \end{bmatrix}, b = \begin{bmatrix} 2 \\ -3 \end{bmatrix}, c = \begin{bmatrix} -5 \\ 2 \end{bmatrix}.$$

Then the primal problem (P) *becomes*

$$\begin{aligned} \min \quad & -5x_1 + 2x_2 \\ \text{s.t.} \quad & 2x_1 - x_2 = 2, \\ & -2x_1 + x_2 = -3, \\ & x_1, x_2 \geq 0, \end{aligned}$$

which is obviously infeasible due to the conflicting constraints. The dual problem (D) *can be written as*

$$\begin{aligned} \max \quad & 2y_1 - 3y_2 \\ \text{s.t.} \quad & 2y_1 - 2y_2 \leq -5, \\ & -y_1 + y_2 \leq 2, \\ & y_1, y_2 \in \mathbb{R}, \end{aligned}$$

which is also infeasible because the second constraint implies $2y_1 - 2y_2 \geq -4$*, which contradicts with the first constraint.*

We summarize the possible outcomes of the primal and the dual problems in the following table, where the columns and the rows correspond to the primal and the dual LO problem, respectively, and a ✓ means possible while ✗ means impossible.

A very useful implication of the strong duality (Proposition 5.2) is a relation between any primal and dual solutions, which is called *complementary slackness*.

Table 2: Possible outcomes of the primal and the dual LO problems

	Optimal	Infeasible	Unbounded
Optimal	✓	✗	✗
Infeasible	✗	✓	✓
Unbounded	✗	✓	✗

Corollary 5.4. *Let x denote a feasible solution to the primal problem (P) and y a feasible solution to the dual problem (D). Then x and y are simultaneously optimal solutions to their LO problems if and only if the following condition hold:*

$$(c - A^\mathsf{T} y)^\mathsf{T} x = 0.$$

Proof. First, suppose x and y are optimal solutions. Then by Proposition 5.2, we know that

$$0 = c^\mathsf{T} x - b^\mathsf{T} y = (c - A^\mathsf{T} y)^\mathsf{T} x.$$

Conversely, by the same inequality we know that $c^\mathsf{T} x = b^\mathsf{T} y$. Now apply the weak duality (5.9), we know that both x and y are optimal solutions. □

Corollary 5.4 tells us the followings: for an optimal primal solution x^* and an optimal dual solution y^*,

(i) if $x_i^* > 0$ for some $i = 1, \ldots, n$, then we must have $\sum_{j=1}^m a_{ji} y_j^* = c_i$; or

(ii) if $\sum_{j=1}^m a_{ji} y_j^* < c_i$ for some $i = 1, \ldots, n$, then $x_i^* = 0$.

While our proof here is based on the particular format used in the primal (P) and the dual (D) problems, it is straightforward to check that the complementary slackness holds for any general LO forms. We may interpret it as the following simple rule:

> Either a variable is at its bound zero, or the corresponding dual constraint must hold as an equality.

We illustrate how we can use the complementary slackness to find an optimal solution of a pair of LO problems below.

Example 5.5. *Consider the LO problem:*

$$\begin{aligned} \min \quad & 3x_1 + 4x_2 + 2x_3 \\ \text{s.t.} \quad & x_1 + 2x_2 + x_3 = 5, \\ & 2x_1 + 3x_2 + x_3 = 8, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

The dual LO problem is

$$\begin{aligned} \max \quad & 5y_1 + 8y_2 \\ \text{s.t.} \quad & y_1 + 2y_2 \leq 3, \\ & 2y_1 + 3y_2 \leq 4, \\ & y_1 + y_2 \leq 2, \\ & y_1, y_2 \in \mathbb{R}. \end{aligned}$$

By the complementary slackness, we must have two out of the three constraints binding *at an optimal dual solution y^*.*

(i) *If the second and the third constraints are binding, then we have $y^{(1)} = (2, 0)$, which gives an objective value $w^{(1)} = 5y_1^{(1)} + 8y_2^{(1)} = 10$.*

(ii) *If the first and the third constraints are binding, then we have $y^{(2)} = (1, 1)$, but this is not feasible as it violates the second constraint.*

(iii) *If the first and the second constraints are binding, then we have $y^{(3)} = (-1, 2)$, which gives an objective value $w^{(3)} = 5y_1^{(3)} + 8y_2^{(3)} = 11$.*

Comparing these three possibilities, we see that $y^* = y^{(3)} = (-1, 2)$, which means that $x_3^* = 0$ for any optimal primal solution x^*. This allows us to solve a system of equations for x_1^* and x_2^*, which gives us $x^* = (1, 2, 0)$.

Sometimes it is more efficient to solve the dual problem and use the complementary slackness to recover the primal solution.

Example 5.6. *Consider the following LO problem*

$$\begin{aligned} \max \quad & 5y_1 + 8y_2 \\ \text{s.t.} \quad & y_1 + 2y_2 \leq 3, \\ & 2y_1 + 3y_2 \leq 4, \\ & y_1 + y_2 \leq 2, \\ & y_2 \leq 1, \\ & y_1, y_2 \in \mathbb{R}. \end{aligned}$$

Note that this is the maximization LO problem in Example 5.5 with one additional constraint $y_2 \leq 1$. As a result, the solution $y = (-1, 2)$ is no longer feasible for the new maximization problem. Nevertheless, if we write out the dual LO problem

$$\begin{aligned} \min \quad & 3x_1 + 4x_2 + 2x_3 + x_4 \\ \text{s.t.} \quad & x_1 + 2x_2 + x_3 = 5, \\ & 2x_1 + 3x_2 + x_3 + x_4 = 8, \\ & x_1, x_2, x_3, x_4 \geq 0, \end{aligned}$$

we see that the optimal solution to the minimization problem in Example 5.5 can be extended to a basic feasible solution $x = (1,2,0,0)$ to the new minimization problem. Thus instead of using the big-M or the two-phase simplex method on the maximization LO problem, we can use the known solution to "warm start" our minimization problem. We convert our problem into a maximization problem with the objective function

$$z = -3x_1 - 4x_2 - 2x_3 - x_4,$$

and write the raw tableau as follows.

z	x_1	x_2	x_3	x_4	*rhs*	*basis*
1	3	4	2	1	0	z
0	1	2	1	0	5	x_1
0	2	3	1	1	8	x_2

Through elementary row operations, the standard initial tableau should be as follows.

z	x_1	x_2	x_3	x_4	*rhs*	*basis*
1	0	0	1	-1	-11	z
0	1	0	-1	2	1	x_1
0	0	1	1	-1	2	x_2

By choosing x_4 as the entering variable and x_1 as the leaving variable, we get the new tableau below.

z	x_1	x_2	x_3	x_4	*rhs*	*basis*
1	1/2	0	1/2	0	$-21/2$	z
0	1/2	0	$-1/2$	1	1/2	x_4
0	1/2	1	1/2	0	5/2	x_2

This tableau gives an optimal solution $x^* = (0,5/2,0,1/2)$ with an optimal value $z^* = -21/2$. By the complementary slackness, we know that for an optimal dual solution y^*, the second and the fourth constraints must be binding:

$$\begin{aligned} 2y_1^* + 3y_2^* &= 4, \\ y_2^* &= 1. \end{aligned}$$

Then it is straightforward to see that $y^* = (1/2,1)$, with the objective value $w^* = 5y_1^* + 8y_2^* = 21/2 = -z^*$, which confirms that we have found an optimal solution.

The procedure in Example 5.6 is a simple illustration of what is known as the *dual simplex method*. The high-level idea behind the dual simplex method is that when a LO problem is modified, we may still be able to reuse some of the found information to

make assertions about or accelerate the solution to the modified problem. This idea leads to *sensitivity analysis*, which is discussed below.

5.3 Sensitivity Analysis

We begin with the assumption that we have found an optimal solution $\bar{x}$ to the primal LO problem (P) with the basis $B \subset \{1, \ldots, n\}$, such that the simplex tableau can be written as

z	x_B	x_N	rhs
1	0	$-(c_N - c_B^\mathsf{T} A_B^{-1} A_N)$	$c_B^\mathsf{T} A_B^{-1} b$
0	I	$A_B^{-1} A_N$	$A_B^{-1} b$

(5.10)

where $N := \{1, \ldots, n\} \setminus B$, and $r = c_N - c_B^\mathsf{T} A_B^{-1} A_N \geq 0$ due to the minimization in (P). We discuss some possible changes to the problem data c, b, and A, respectively, in the following.

Changing the objective coefficients

We consider a new objective function $z = (c + d)^\mathsf{T} x$ for some $d \in \mathbb{R}^n$, which can be partitioned into d_B and d_N in the same way c is. Note that the solution $\bar{x}$ remains feasible to the modified problem as no constraint is changed. The objective value associated with the solution $\bar{x}$ is changed to

$$(c_B + d_B)^\mathsf{T} A_B^{-1} b = c_B^\mathsf{T} A_B^{-1} b + d_B^\mathsf{T} A_B^{-1} b.$$

The difference here is the product of the change in the basic variable coefficient d_B and the constraint right-hand side $A_B^{-1} b$. To check the optimality of the incumbent solution $\bar{x}$, we need to calculate the modified reduced cost

$$r' := (c_N + d_N) - (c_B + d_B)^\mathsf{T} A_B^{-1} A_N = r + (d_N - d_B^\mathsf{T} A_B^{-1} A_N).$$

The difference here can be calculated by the change of objective coefficients d_B and d_N, and the constraint coefficients in the tableau $A_B^{-1} A_N$. The solution $\bar{x}$ remains optimal if and only if $r' \geq 0$.

Changing the constraint coefficients

If any constraint coefficient associated with the basic variables A_B is changed, then the inverse A_B^{-1} is also changed (in a nonlinear way), so that we need to recompute it to get the coefficients for x_N and the right-hand sides before we can verify feasibility or optimality of $\bar{x}$. However, if only the coefficients associated with the nonbasic variables A_N are changed, for example, to $A_N + D_N$, then $\bar{x}$ remains feasible with the constraint

coefficients $A_B^{-1}A_N$ changed to $A_B^{-1}(A_N + D_N)$. To check the optimality, note that the reduced cost is modified to

$$r' := c_N - c_B^\mathsf{T} A_B^{-1}(A_N + D_N) = r - c_B^\mathsf{T} A_B^{-1} D_N.$$

If we have a dual optimal solution $\bar{y} = (c_B^\mathsf{T} A_B^{-1})^\mathsf{T}$, then the difference in the reduced cost can be calculated more conveniently by $\bar{y}^\mathsf{T} D_N$.

Changing the constraint right-hand sides

Suppose the constraint right-hand side is changed from b to $b + d$ for some vector $d \in \mathbb{R}^m$. Then the constraint right-hand side is changed to

$$A_B^{-1}(b + d) = A_B^{-1}b + A_B^{-1}d.$$

In general, we would need to compute the inverse A_B^{-1} to obtain the difference $A_B^{-1}d$. If the new right-hand side $A_B^{-1}(b+d)$ is nonnegative, then the solution $\bar{x}$ remains feasible and optimal, because the reduced cost r is unchanged. If the right-hand side $A_B^{-1}(b+d)$ is no longer nonnegative, or if the inverse A_B^{-1} is challenging to compute, while the dual solution $\bar{y}$ is available, then we can use the dual simplex method (as described in Example 5.6) to find an optimal dual solution to the problem

$$\begin{aligned} \max \quad & (b+d)^\mathsf{T} y \\ \text{s.t.} \quad & A^\mathsf{T} y \le c, \\ & y \in \mathbb{R}^m, \end{aligned}$$

and recover a primal optimal solution using complementary slackness. Moreover, if the tableau associated with $\bar{y}$ is known, then the change of the optimal value can be seen from the dual problem with a changed objective coefficient vector.

Adding a new variable or constraint

If a new variable x_{n+1} is added into our problem (P), then the solution $(x_1, \ldots, x_n, x_{n+1}) = (\bar{x}_1, \ldots, \bar{x}_n, 0)$ is feasible. We can set B to be the same basis, while $N \leftarrow N \cup \{n+1\}$, so we can start our simplex method from the solution $(\bar{x}, 0)$. Given $a_{n+1} \in \mathbb{R}^n$ and $c_{n+1} \in \mathbb{R}$, we can set

$$D_N = \begin{bmatrix} 0 & \ldots & 0 & a_{n+1,1} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{n+1,m} \end{bmatrix}$$

and calculate the modified reduced cost by

$$r' = (c_N, c_{n+1}) - c_B^\mathsf{T} A_B^{-1} D_N.$$

The solution $\bar{x}$ remains optimal if and only if $r' \geq 0$, or equivalently, $c_{n+1} \geq c_B^\mathsf{T} A_B^{-1} a_{n+1}$.

If a new constraint is added, then the primal solution $\bar{x}$ may not be feasible any more. However, on the dual side we are basically adding a new variable to the problem (D), so the solution $(\bar{y}, 0)$ remains feasible. Thus we can start our dual simplex method as we did in Example 5.6.

6 Overview of Mixed-integer Linear Optimization

6.1 Mixed-integer Linear Optimization and Computer Tools

Given problem data $c = (c_1, \ldots, c_n) \in \mathbb{R}^n$, $b = (b_1, \ldots, b_m) \in \mathbb{R}^m$, and

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n},$$

a (mixed-)integer linear optimization (MILO) model can be defined as

$$\begin{aligned} \min \text{ / } \max \quad & c^\mathsf{T} x \\ \text{s.t.} \quad & Ax \leq b, \\ & x \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}, \end{aligned} \tag{6.1}$$

where $n_1 + n_2 = n$. If $n = n_1$ and $n_2 = 0$, then the problem (6.1) reduces to the usual linear optimization (LO) model. If $n_2 = n$, then we say it is a (pure) integer linear optimization, and otherwise a mixed-integer linear optimization when $n_2 < n$. While integer and mixed-integer linear optimization models may have different algorithmic aspects, we do not distinguish them in this course because we focus mainly on the modeling part. We see a simple example of a MILO model as follows.

Example 6.1. *A company produces two types of baby carriers, non-reversible and reversible.*

- *Each non-reversible carrier sells for \$22, requires 2 linear yards of a solid color fabric, and costs \$7 to manufacture.*
- *Each reversible carrier sells for \$35, requires 2 linear yards of a printed fabric as well as 2 linear yards of a solid color fabric, and costs \$10 to manufacture.*

The company has 900 linear yards of solid color fabrics and 600 linear yards of printed fabrics available for its new carrier collection. It can spend up to \$4,000 on manufacturing the carriers. The demand is such that all reversible carriers made are projected to sell, whereas at most 350

non-reversible carriers can be sold. The goal of the company is to maximize its profit (e.g., the difference of revenues and expenses) resulting from manufacturing and selling the new carrier collection.

We define $x_1, x_2 \geq 0$ to be the numbers of non-reversible and reversible carriers to manufacture. If we do not require x_1, x_2 to take integer values, then we have a LO model as

$$\begin{array}{rrcrcll}
\max & 15x_1 & + & 25x_2 & & & \textit{(profit)} \\
\text{s.t.} & x_1 & + & x_2 & \leq & 450 & \textit{(solid color fabric constraint)} \\
 & & & x_2 & \leq & 300 & \textit{(printed fabric constraint)} \\
 & 7x_1 & + & 10x_2 & \leq & 4,000 & \textit{(budget constraint)} \\
 & x_1 & & & \leq & 350 & \textit{(demand constraint)} \\
 & & & x_1, x_2 & \geq & 0. & \textit{(nonnegativity constraints).}
\end{array}$$

We can use `OR-Tools` *and* `GLOP` *to find the solution, as coded in* `lin_model_production.py`*, and get the following result.*

```
The maximum profit for the baby carrier production is 9642.9.
x1 = 142.86 (non-reversible carriers)
x2 = 300.00 (reversible carriers)
```

We can see that the solution is not integral. In fact, since the objective function consists of integer coefficients and the optimal value here is non-integer, there does not exist an integer solution. This example suggests that it is not always practical to relax the integrality requirements and solve the LO model as an substitute.

We can still use the `Python` module `OR-Tools` for MILO modeling. As usual, we import it using the following command.

```
from ortools.linear_solver import pywraplp
```

Next we declare a MILO solver `SCIP` instead of the LO solver `GLOP` that we used for LO models.

```
solver = pywraplp.Solver.CreateSolver("SCIP")
```

For Example 6.1, we can define our integer variables using the `IntVar` functions, the argument for which is the same as the function `NumVar` for continuous variables.

```
x1 = solver.IntVar(0.0, solver.infinity(), 'x1')
x2 = solver.IntVar(0.0, solver.infinity(), 'x2')
```

To be more specific, the first argument specifies the lower bound, the second one specifies the upper bound (where `solver.infinity()` gives ∞), and the last one gives the variable a name used in model printing or exporting. After adding the variables, each linear constraint can be added similar to what we did for LO models.

```
# define the solid color fabric constraint
solver.Add(x1 + x2 <= 450)

# define the printed fabric constraint
solver.Add(x2 <= 300)

# define the budget constraint
solver.Add(7*x1 + 10*x2 <= 4000)

# define the demand constraint
solver.Add(x1 <= 350)
```

We set linear objective function.

```
# create the objective of maximizing the profit
solver.Maximize(15*x1 + 25*x2)
```

And now we may invoke the solver.

```
# call the solver
status = solver.Solve()
```

The result (from the script `int_model_production.py`) is printed below.

```
The maximum profit for the baby carrier production is 9635.0.
x1 = 144.00 (non-reversible carriers)
x2 = 299.00 (reversible carriers)
```

By comparing the MILO and LO solutions in Example 6.1, we see that the MILO solution is not just a rounded solution of the LO model. The following example further illustrates that a rounded solution may not be feasible to the MILO problem.

Example 6.2. *A hospital uses a 12-hour shift schedule for its nurses, with each nurse working either day shifts (7:00 am-7:00 pm) or night shifts (7:00 pm-7:00 am). Each nurse works 3*

Table 3: Number of Required Nurses for Day Shifts

Day of week/shift	Nurses required
Monday (Mo)	16
Tuesday (Tu)	12
Wednesday (We)	18
Thursday (Th)	13
Friday (Fr)	15
Saturday (Sa)	9
Sunday (Su)	8

consecutive day shifts or 3 consecutive night shifts and then has 4 days off. The minimum

number of nurses required for each day shift during a week is given in the following table: In addition, it is required that at least two thirds of the day-shift nurses have weekends (Saturday and Sunday) off. The hospital is aiming to design a schedule for day-shift nurses that minimizes the total number of nurses employed.

To formulate a MILO model for the hospital scheduling problem, we define the decision variables ($\mathbb{Z}_{\geq 0}$ means nonnegative integers) as follows.

$x_1 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Mo-Tu-We schedule*
$x_2 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Tu-We-Th schedule*
$x_3 \in \mathbb{Z}_{\geq 0}$: *number of nurses on We-Th-Fr schedule*
$x_4 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Th-Fr-Sa schedule*
$x_5 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Fr-Sa-Su schedule*
$x_6 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Sa-Su-Mo schedule*
$x_7 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Su-Mo-Tu schedule*

On Monday, there are $x_1 + x_6 + x_7$ nurses working, so by requirement we should have

$$x_1 + x_6 + x_7 \geq 16.$$

Similarly, for the other days of the week, we have constraints

$$\begin{aligned} x_1 + x_2 + x_7 &\geq 12, \\ x_1 + x_2 + x_3 &\geq 18, \\ x_2 + x_3 + x_4 &\geq 13, \\ x_3 + x_4 + x_5 &\geq 15, \\ x_4 + x_5 + x_6 &\geq 9, \\ x_5 + x_6 + x_7 &\geq 8. \end{aligned}$$

Clearly $\sum_{i=1}^{7} x_i \geq 1$. Thus the requirement that two thirds of the day-shift nurses have weekends off can be expressed as

$$\frac{x_1 + x_2 + x_3}{\sum_{i=1}^{7} x_i} \geq \frac{2}{3}.$$

This can be transformed into a linear *constraint*

$$x_1 + x_2 + x_3 - 2x_4 - 2x_5 - 2x_6 - 2x_7 \geq 0.$$

The objective is to minimize the total number of nurses $\sum_{i=1}^{7} x_i$, so the model can be written as

$$\begin{aligned}
\min \quad & \sum_{i=1}^{7} x_i \\
\text{s.t.} \quad & x_1 + x_6 + x_7 \geq 16, \\
& x_1 + x_2 + x_7 \geq 12, \\
& x_1 + x_2 + x_3 \geq 18, \\
& x_2 + x_3 + x_4 \geq 13, \\
& x_3 + x_4 + x_5 \geq 15, \\
& x_4 + x_5 + x_6 \geq 9, \\
& x_5 + x_6 + x_7 \geq 8, \\
& \sum_{i=1}^{3} x_i - 2 \sum_{i=4}^{7} x_i \geq 0, \\
& x_i \in \mathbb{Z}_{\geq 0}, \quad i = 1, \ldots, 7.
\end{aligned}$$

If we relax the integrality constraints, the result (from the script `lin_model_scheduling.py`*) is printed below.*

```
The minimum number of nurses is 31.3.
x1 = 10.29
x2 = 0.29
x3 = 10.29
x4 = 2.43
x5 = 2.29
x6 = 4.29
x7 = 1.43
```

If we round them to a nearest integer solution $\bar{x} = (10, 0, 10, 2, 2, 4, 1)$, we see that

$$\begin{aligned}
\bar{x}_1 + \bar{x}_6 + \bar{x}_7 &= 15 \not\geq 16, \\
\bar{x}_1 + \bar{x}_2 + \bar{x}_7 &= 11 \not\geq 12, \\
\bar{x}_1 + \bar{x}_2 + \bar{x}_3 &= 20 \geq 18, \\
\bar{x}_2 + \bar{x}_3 + \bar{x}_4 &= 12 \not\geq 13, \\
\bar{x}_3 + \bar{x}_4 + \bar{x}_5 &= 14 \not\geq 15, \\
\bar{x}_4 + \bar{x}_5 + \bar{x}_6 &= 8 \not\geq 9, \\
\bar{x}_5 + \bar{x}_6 + \bar{x}_7 &= 7 \not\geq 8.
\end{aligned}$$

In other words, this rounded solution $\bar{x}$ is infeasible*. Instead, we should directly solve the MILO model in the script* `int_model_scheduling.py`*, the result for which is printed below.*

```
The minimum number of nurses is 32.0.
x1 = 11.00
x2 = 0.00
x3 = 11.00
x4 = 2.00
x5 = 3.00
x6 = 4.00
x7 = 1.00
```

One of the most important features of MILO modeling is the use of $0/1$ variables (or sometimes called *binary* variables). An example is the following location covering problem.

Example 6.3. *A city is planning to set up new emergency centers at different possible locations. Due to distances and one-way streets, an emergency center at*

- *location 1 can cater to patients in locations 1, 2, 4, 7;*
- *a center at location 2 can cater to patients in locations 2, 3, 5;*
- *a center at location 3 can cater to patients in locations 1, 3, 6;*
- *a center at location 4 can cater to patients in locations 2, 3, 4, 5;*
- *a center at location 5 can cater to patients in locations 1, 5, 6;*
- *a center at location 6 can cater to patients in locations 3, 4, 6;*
- *a center at location 7 can cater to patients in locations 2, 3, 7.*

We need to cater to patients at all locations and would like to set up the minimum number of emergency centers.
To model this problem, we can define our decision variables for $i = 1, \ldots, 7$ by

$$x_i = \begin{cases} 1 & \text{if an emergency center is set up at location } i, \\ 0 & \text{otherwise.} \end{cases}$$

To cover patients at the location 1, there are only three possible locations: 1, 3, and 5. Thus we need

$$x_1 + x_3 + x_5 \geq 1.$$

Similarly, we can write the covering constraints at other locations as

$$\begin{aligned} x_1 + x_2 + x_4 + x_7 &\geq 1, \\ x_2 + x_3 + x_4 + x_6 + x_7 &\geq 1, \\ x_1 + x_4 + x_6 &\geq 1, \\ x_2 + x_4 + x_5 &\geq 1, \\ x_3 + x_5 + x_6 &\geq 1, \\ x_1 + x_7 &\geq 1. \end{aligned}$$

The goal is to minimize the sum

$$\min \quad \sum_{i=1}^{7} x_i.$$

We can code this model in the script `model_covering.py` *and get the following result.*

```
The minimum number of emergency centers is 3.0.
x1 = 1.0
x2 = 1.0
x3 = 1.0
x4 = 0.0
x5 = 0.0
x6 = 0.0
x7 = 0.0
```

The following *knapsack problem* is a very well-known problem in integer optimization.

Example 6.4. *One would like to carry different items to the camping ground in my knapsack. They have a choice of n items. Item i produces an utility of u_i for them. The volume of item i is v_i. The volume of the knapsack is V. The goal is to maximize the total utility of items put in the knapsack.*
To formulate a MILO model, we define our variables for each item $i = 1, \ldots, n$ as

$$x_i \in \{0, 1\} : \text{ whether or not item } i \text{ is chosen.}$$

Then the knapsack volume constraint can be written as

$$\sum_{i=1}^{n} v_i x_i \leq V.$$

The objective function is to maximize the total utility

$$\max \quad \sum_{i=1}^{n} u_i x_i.$$

6.2 Logical Constraints on 0/1 Variables

In MILO models, $0/1$ variables are often used to indicate whether a certain condition happens or not. For example, for a variable z_i, we say that condition i is satisfied if $z_i = 1$, and $z_i = 0$ otherwise, for some $i = 1, 2$, or 3. We can impose logical constraints on these $0/1$ as follows.

- "If-then" constraint: if condition 1 is satisfied, then we also need to satisfy condition 2

$$z_2 \geq z_1, \quad z_1, z_2 \in \{0, 1\}.$$

- Nonexclusive "either-or" constraint: either condition 1 is satisfied, or condition 2 is satisfied, and both of them can be satisfied at the same time

$$z_1 + z_2 \geq 1, \quad z_1, z_2 \in \{0, 1\}.$$

- Exclusive "either-or" constraint: either condition 1 is satisfied, or condition 2 is satisfied, but they cannot be satisfied at the same time

$$z_1 + z_2 = 1, \quad z_1, z_2 \in \{0, 1\}.$$

Sometimes we may compose our conditions with logical operations before in "if-then" statement. The following cases are some examples on how we can do this.

- "Not" operation: to say that condition 2 is satisfied if condition 1 is *not* satisfied, we can use

$$z_2 \geq 1 - z_1, \quad z_1, z_2 \in \{0, 1\}.$$

- "And" operation: to say that condition 3 is satisfied if conditions 1 *and* 2 are satisfied, we can use

$$z_3 \geq z_1 + z_2 - 1, \quad z_1, z_2, z_3 \in \{0, 1\}.$$

- "Or" operation: to say that condition 3 is satisfied if conditions 1 *or* 2 is satisfied, we can use

$$z_3 \geq z_1, \quad z_3 \geq z_2, \quad z_1, z_2, z_3 \in \{0, 1\}.$$

- "Xor" operation: to say that condition 3 is satisfied if conditions 1 *xor* 2 is satisfied

(i.e., exactly one of conditions 1 and 2 is satisfied), we can use

$$z_3 \geq z_3', \quad z_1 + z_2 + z_3' = 2z_3'', \quad z_1, z_2, z_3, z_3', z_3'' \in \{0, 1\}.$$

Here, z_3' and z_3'' are auxiliary variables introduced to model the "xor" operation.

The following resource allocation problem is an example of the knapsack problem with logical constraints.

Example 6.5. *There are 6 projects considered for potential investment of the \$100,000 budget for the upcoming year. The required investment and end-of-year payout amounts are described in the following table. We assume that partial investments are not allowed, that is, if a project*

	Project					
	1	2	3	4	5	6
Investment (\$·1000)	10	25	35	45	50	60
Payout (\$·1000)	12	30	41	55	65	77

is selected, then we must invest the full amount in it. Additionally, the following requirements need to be satisfied.

- *No more than three projects can be selected.*
- *If project 6 is chosen, then project 1 must also be chosen.*
- *Project 5 can be chosen only if project 2 is chosen.*
- *If project 3 is chosen, then project 4 cannot be chosen.*

The objective is to maximize the total end-of-year payout from the investment.
To define our decision variables, for each project $i = 1, \ldots, 6$, let

$$x_i = \begin{cases} 1, & \text{if we choose project } i, \\ 0, & \text{otherwise.} \end{cases}$$

Without the additional requirements, we only have the budget constraint

$$10x_1 + 25x_2 + 35x_3 + 45x_4 + 50x_5 + 60x_6 \leq 100.$$

The objective can be written as

$$\max \quad 12x_1 + 30x_2 + 41x_3 + 55x_4 + 65x_5 + 77x_6.$$

Each of the requirements can be written as follows.

- *No more than three projects can be selected.*

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3.$$

- *If project 6 is chosen, then project 1 must also be chosen.*

$$x_6 \leq x_1.$$

- *Project 5 can be chosen only if project 2 is chosen.*

$$x_5 \leq x_2.$$

- *If project 3 is chosen, then project 4 cannot be chosen.*

$$x_4 \leq 1 - x_3.$$

We code this MILO model in `model_allocation.py` *and use the solver* `SCIP` *to get the following result.*

```
The maximum payout is $119000.00.
The investment on each project is shown below.
x1: 1.0
x2: 1.0
x3: 0.0
x4: 0.0
x5: 0.0
x6: 1.0
```

The logical constraints can appear even when there are continuous variables, as shown in the next example.

Example 6.6. *Suppose that in Example 6.5, we now allow fractional investment but with a minimum for each project. The updated project information is listed in the table below. To up-*

	Project					
	1	2	3	4	5	6
Investment ($·1000)	10	25	35	45	50	60
Payout ($·1000)	12	30	41	55	65	77
Min. Amount ($·1000)	2	5	4	9	6	7

date the model, we define the following continuous decision variables in addition to $x_1, \ldots, x_6$,

$$y_i \geq 0 : \text{ the amount invested in the project } i, \quad i = 1, \ldots, 6.$$

We can impose constraints

$$m_i x_i \leq y_i \leq v_i x_i, \quad i = 1, \ldots, 6,$$

where $m_i > 0$ is the minimum investment ammount for project i, while v_i is the full investment amount. In this way, we invest in the project i if and only if $x_i = 1$. Accordingly, the budget constraint becomes

$$y_1 + y_2 + y_3 + y_4 + y_5 + y_6 \leq 100.$$

The objective can be written as

$$\max \quad \frac{12}{10}y_1 + \frac{30}{25}y_2 + \frac{41}{35}y_3 + \frac{55}{45}y_4 + \frac{65}{50}y_5 + \frac{77}{60}y_6.$$

We code this MILO model in the script `model_fractional_allocation.py` *and get the following result.*

```
The maximum payout is $126000.00.
The investment on each project is shown below.
x1: 0.0
y1: $0.00
x2: 1.0
y2: $5000.00
x3: 0.0
y3: $0.00
x4: 1.0
y4: $45000.00
x5: 1.0
y5: $50000.00
x6: 0.0
y6: $0.00
```

We see that the total payout increases compared with Example 6.5 because we have more flexible investment options for each project. We are now investing in projects 2, 4, 5, which are different from the projects 1, 2, 6 previously.

In Example 6.6, we are enforcing the continuous variables y_i to be zero if the corresponding 0/1 variable x_i is zero. The modeling of such optional variable bound/-linear constraint can be done in a more general setting. For some linear constraint $j = 1, \ldots, m$,

$$\sum_{i=1}^{n} a_{ji} x_i \leq b_j,$$

we can "switch on or off" this constraint by an additional integer variable $z_j \in \{0, 1\}$ by

$$\sum_{i=1}^{n} a_{ji} x_i \leq b_j + M z_j,$$

where the parameter M is a "sufficiently large" number. Theoretically speaking, we

should choose M such that during any algorithmic step (e.g., an simplex method iteration), the sign of any linear expression involving M only depends on the sign of M. In practice, assuming that our problem is bounded, we may adaptively increase the value of M by a fixed multiplicative factor each time, until our objective value does not change any more. There are also many problems where we can naturally get a good choice of M from the problem data, as shown in the following example.

Example 6.7. *A wholesale company specializing in one product considers the possibility of opening up to m warehouses W_i of capacity b_i, for $i = 1, \ldots, m$, to serve n retail locations R_j, for $j = 1, \ldots, n$. The fixed cost of opening warehouse W_i is f_i. The capacity of b_i means that we can ship up to b_i units of product from warehouse W_i. Transporting one unit of the product from W_i to R_j costs c_{ij} dollars. To satisfy the demand, at least d_j units of the product must be delivered to R_j. The goal is to decide which of the m warehouses to open and how many units of the product should be shipped from each opened warehouse to each retail location, minimizing the overall cost for the company.*
As in the usual transportation problem, we define the continuous variables for each $i = 1, \ldots, m$ and $j = 1, \ldots, n$

$$x_{ij} \geq 0 : \text{ the product quantity shipped from } W_i \text{ to } R_j.$$

To model the fixed cost, we introduce for each $i = 1, \ldots, m$

$$y_i = \begin{cases} 1, & \text{if warehouse } i \text{ is open,} \\ 0, & \text{otherwise.} \end{cases}$$

The objective is to minimize the total cost of transportation plus the fixed charges of opening warehouse:

$$\min \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i.$$

To satisfy the demand at R_j, we can write

$$\sum_{i=1}^{m} x_{ij} \geq d_j, \quad j = 1, \ldots, n.$$

We also need to make sure that the number of units shipped out of W_i does not exceed the capacity:

$$\sum_{j=1}^{n} x_{ij} \leq b_i, \quad i = 1, \ldots, m.$$

In addition, if a warehouse is not opened, then no units can be shipped out of it:

$$\text{if } y_i = 0, \quad \text{then } x_{ij} = 0, \quad j = 1, \ldots, m.$$

Note that this is an optional linear constraint, and b_i is a natural choice for the big-M for every x_{ij}, $j = 1, \ldots, n$. Thus we can write our MILO model as

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i, \\
\text{s. t.} \quad & \sum_{i=1}^{m} x_{ij} \geq d_j, \quad j = 1, \ldots, n, \\
& \sum_{j=1}^{n} x_{ij} \leq b_i, \quad i = 1, \ldots, m, \\
& x_{ij} \leq b_i y_i, \quad i = 1, \ldots, m,\ j = 1, \ldots, n, \\
& x_{ij} \geq 0,\ y_i \in \{0, 1\}, \quad i = 1, \ldots, m,\ j = 1, \ldots, n.
\end{aligned}$$

An important application of MILO is to model piecewise linear functions that are not convex nor concave. For simplicity, we focus on modeling continuous piecewise linear functions on an interval I of finite length. Recall that a function f is piecewise linear on $I := \{x \in \mathbb{R} : \underline{a} \leq x \leq \bar{a}\}$ if we can find points $\underline{a} = a_0 < a_1 < \cdots < a_l = \bar{a}$, such that on each subinterval $I_k := \{x \in \mathbb{R} : a_{k-1} < x < a_k\}$, $k = 1, \ldots, l$, $f(x)$ is an affine linear function, i.e., there exist $b_k, c_k \in \mathbb{R}$ such that

$$f(x) = b_k x + c_k, \quad \forall\, x \in I_k, \quad k = 1, \ldots, l.$$

The continuity assumption then translates into conditions

$$f(a_k) = a_k b_k + c_k = a_k b_{k+1} + c_{k+1}, \quad \forall\, k = 1, \ldots, l-1.$$

Suppose we would like to model the equation $y = f(x)$, $x \in I$. We can define auxiliary variables

$$z_k = \begin{cases} 1, & \text{if } x \text{ lies in the interval } I_k, \\ 0, & \text{otherwise.} \end{cases}$$

By definition, we should have

$$\sum_{k=1}^{l} z_k = 1.$$

Note that if $x = a_k$ for some $k = 1, \ldots, k-1$, then we can allow either $z_k = 1$ or $z_{k+1} = 1$. We need to enforce the linear constraints $y = b_k x + c_k$ if $z_k = 1$, which can be written as

$$\begin{aligned}
y &\leq b_k x + c_k + M(1 - z_k), \\
y &\geq b_k x + c_k - M(1 - z_k),
\end{aligned}$$

for some sufficiently large $M > 0$, e.g., $M = \max\{\sup_{x \in I} f(x), -\inf_{x \in I} f(x)\}$. In practice, using this big-M model can sometimes lead to inefficiency in the solution step

(see the next section). Thus we describe an alternative way to model piecewise linear functions.

We observe that if f is a continuous piecewise linear function, then on each subinterval I_k, the value y should be a convex combination of $f(a_{k-1}), f(a_k)$ in the same way x is a convex combination of a_{k-1}, a_k. Thus we define more auxiliary variables $0 \leq w_0, w_1, \ldots, w_k \leq 1$ as convex combination coefficients.

- If $x \notin I_1$, then $w_0 = 0$, which can be expressed as

$$w_0 \leq z_1.$$

- If $x \notin I_k$ *and* $x \notin I_{k+1}$, $k \leq l-1$, then $w_k = 0$, which can be expressed as

$$w_k \leq z_k + z_{k+1}, \quad k = 1, \ldots, l-1.$$

- If $x \notin I_l$, then $w_l = 0$, which can be expressed as

$$w_l \leq z_l.$$

Then we can write x, y as convex combinations

$$\begin{aligned} 1 &= \sum_{k=0}^{l} w_k, \\ x &= \sum_{k=0}^{l} w_k a_k, \\ y &= \sum_{k=0}^{l} w_k f(a_k). \end{aligned}$$

Example 6.8. *A large-scale grocery retailer must purchase onions for two of their stores. Onions can be purchased from three farms. Here are the relevant details: Store 1 requires at least 1000 units and store 2 requires at least 2000 units of onions. Farm 1 sells onions at \$3 per unit and farm 2 sells onions at \$4 per unit. Farm 3 sells onions in the following fashion. The first 300 units are sold for \$3 per unit, the next 400 units are sold at a discounted rate of \$2.5 per unit. However, the price goes up after that to \$5, as the farm believes that there may be more demand than supply. For example, if 800 units are purchased from farm 3, then the cost is*

$$\$3 \cdot 300 + \$2.5 \cdot 400 + \$5 \cdot 100 = \$2400.$$

The transportation cost per unit from the farms to the stores are given below.

	Farm 1	*Farm 2*	*Farm 3*
Store 1	*\$1*	*\$1*	*\$2*
Store 2	*\$2*	*\$1*	*\$1*

The goal is to determine how many units of onions to purchase from each farm such that the total cost is minimized. For $i = 1, 2, 3$ and $j = 1, 2$, let

$$x_{ij} \geq 0 : \textit{number of units of onions to purchase from farm } i \textit{ for store } j.$$

Let $y \in \mathbb{R}$ denote the cost of purchase from farm 3. The store demand constraints are

$$\begin{aligned} x_{11} + x_{12} + x_{13} &\geq 1000, \\ x_{21} + x_{22} + x_{23} &\geq 2000. \end{aligned}$$

The objective is to minimize the total purchase and transportation cost

$$\begin{aligned} \min \quad & 3(x_{11} + x_{21}) + 4(x_{12} + x_{22}) + y \\ & + x_{11} + x_{12} + 2x_{13} + 2x_{21} + x_{22} + x_{23}. \end{aligned}$$

To model the function $y = f(x_{13} + x_{23})$, note that

$$f(0) = 0, \quad f(300) = 900, \quad f(700) = 1900, \quad f(3000) = 13400.$$

For $k = 1, 2, 3$, let

$$z_k \in \{0, 1\} \textit{ denote whether } x_{13} + x_{23} \textit{ lies in the interval } I_k,$$

and for $k = 0, 1, 2, 3$, let

$$0 \leq w_k \leq 1 \textit{ be the kth convex combination coefficient.}$$

We can write the constraint $y = f(x_{13} + x_{23})$ through the following ones:

$$\begin{aligned} w_0 &\leq z_1, \\ w_1 &\leq z_1 + z_2, \\ w_2 &\leq z_2 + z_3, \\ w_3 &\leq z_3, \\ z_1 + z_2 + z_3 &= 1, \\ w_0 + w_1 + w_2 + w_3 &= 1, \\ 300w_1 + 700w_2 + 3000w_3 &= x_{13} + x_{23}, \\ 900w_1 + 1900w_2 + 13400w_3 &= y. \end{aligned}$$

We code the MILO model in the script `model_purchase.py` *and use* `SCIP` *to solve it. The result is printed below.*

```
The minimum total cost for onion purchase is 13100.000000000004.
x11 = 1000.0000000000001
x21 = 0.0
x31 = 0.0
x12 = 0.0
x22 = 1300.0000000000002
x32 = 700.0
```

6.3 A Glance at MILO Solution Methods

There are two major families of solution methods in solving MILO problems: branch-and-bound methods and cutting plane methods. Each of them deserve much more than what we can spend in this course. Our goal here is modest: to give an idea of how MILO problems can be solved and how the solution methods are related to the simplex method we used for LO problems. For any MILO problem (6.1), we define its LO relaxation as

$$\begin{aligned} \min \text{ / } \max \quad & c^\mathsf{T} x \\ \text{s. t.} \quad & Ax \leq b, \\ & x \in \mathbb{R}^{n_1+n_2}. \end{aligned} \tag{6.2}$$

6.3.1 The Branch-and-bound Methods

Given our MILO problem, suppose we found a solution $\bar{x}$ to the LO relaxation.

- If $\bar{x} \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}$, then $\bar{x}$ is feasible to the MILO problem;
- otherwise there exists $n_1 < i \leq n_1 + n_2$ such that $\bar{x}_i$ is fractional. In this case, we need to solve the problem with either of the two constraints
  - $x_i \geq \lceil \bar{x}_i \rceil$,
  - $x_i \leq \lfloor \bar{x}_i \rfloor$,

Note that at least one of them contains the true optimal solution to our MILO. This *branching* procedure leads to a *tree* of LO subproblems. For example, for the LO relaxation of our MILO problem

$$\text{LO}_0 : \max_{x \in \mathbb{R}^{n_1+n_2}} \quad c^\mathsf{T} x \quad \text{s. t. } Ax \leq b,$$

if we get a fractional solution $\bar{x}$ such that $\bar{x}_i \notin \mathbb{Z}$, we then get two LO problems

$$\text{LO}_1 : \max_{x \in \mathbb{R}^{n_1+n_2}} \quad c^\mathsf{T} x \quad \text{s. t. } Ax \leq b, x_i \geq \lceil \bar{x}_i \rceil,$$

and

$$\text{LO}_2 : \max_{x \in \mathbb{R}^{n_1+n_2}} \quad c^\mathsf{T} x \quad \text{s. t. } Ax \leq b, x_i \leq \lfloor \bar{x}_i \rfloor.$$

Then again if we get a fractional solution $\tilde{x}$ to LO_1, with $\tilde{x}_j \notin \mathbb{Z}$, we can continue the *branching* procedure and get

$$\text{LO}_3 : \max_{x \in \mathbb{R}^{n_1+n_2}} \quad c^\mathsf{T} x \quad \text{s. t. } Ax \leq b, x_i \geq \lceil \bar{x}_i \rceil, x_j \geq \lceil \tilde{x}_j \rceil,$$

and

$$\text{LO}_4 : \max_{x \in \mathbb{R}^{n_1+n_2}} \quad c^\mathsf{T} x \quad \text{s. t. } Ax \leq b, x_i \geq \lceil \bar{x}_i \rceil, x_j \leq \lfloor \tilde{x}_j \rfloor.$$

While it seems that we may need to enumerate all possible integers for $x_{n_1+1}, \ldots, x_{n_1+n_2}$ in our feasible region, many of the LO problems in this procedure do not need any further branching. We can *prune* the problem LO_i if

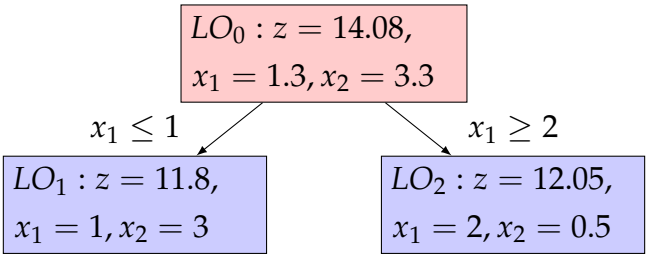
- we find a solution $x \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}$ to the LO problem LO_i;
- the LO problem LO_i is infeasible; or
- the optimal value of LO_i is worse than the best *bound* (the objective value of the best solution we have found).

The pruning step, especially when we already obtained a high-quality bound, would often greatly save our computational effort. We illustrate the branch-and-bound method by the following simple example.

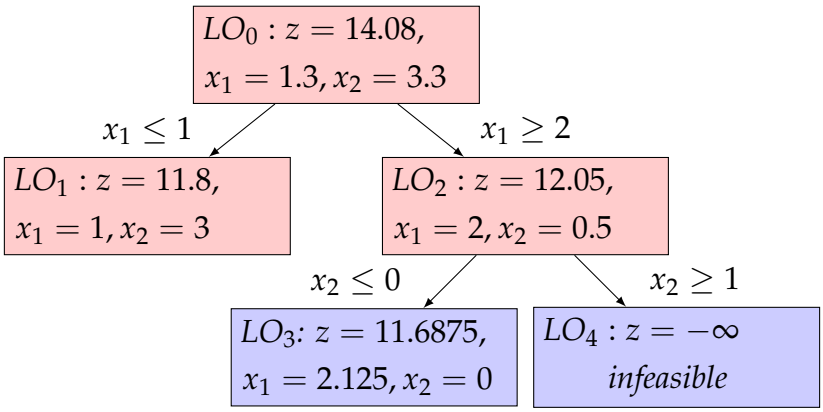
Example 6.9. *Consider the MILO problem*

$$\begin{aligned} \max \quad & 5.5x_1 + 2.1x_2 \\ \text{s. t.} \quad & -x_1 + x_2 \leq 2, \\ & 8x_1 + 2x_2 \leq 17, \\ & x_1, x_2 \in \mathbb{Z}_{\geq 0}. \end{aligned}$$

The solution of the LO relaxation is $x_1 = 1.3, x_2 = 3.3$, while the LO optimal value is $z = 14.08$, which gives an upper bound on our maximization problem. We can branch on the variable x_1: $x_1 \leq 1$ and $x_2 \geq 2$.



Here, the branch LO_1 is pruned by integrality. From LO_2 we branch on x_2: $x_2 \leq 0$ and $x_2 \geq 1$.



Now the branch LO_3 is pruned by bound, and the branch LO_4 is pruned by infeasibility. The optimal value of the MILO problem is thus $z = 11.8$ with an optimal solution $x_1 = 1$, $x_2 = 3$.

The numerical performance of the branch-and-bound methods will often depend on the *branching rule*, i.e., which LO problem to solve next. There are numerous branching rules and heuristics for high-quality solutions for pruning based on the structure of the MILO problem. We remark that in each branching step, we are simply resolving the LO problem with one additional inequality constraint. Thus we may use the current primal-dual solution information in our simplex tableau to warm start the dual simplex method.

6.3.2 The Cutting Plane Methods

Recall that the feasible regions of LO problems are polyhedra. If all of the vertices of the feasible region of the LO relaxation (6.2) are integer points for the last n_2 coordinates, then the simplex method will find an optimal solution to our MILO problem (6.1). Such polyhedra are called *integral*, but they are uncommon in practice. The main idea of the cutting plane methods is to artificially add linear constraints that are *valid* for all (mixed-)integer points while being able to "cut off" non-integer vertices.

As an simple illustration, we consider the following discrete (pure-integer) optimization case (i.e., $n_1 = 0$, $n = n_2$). Suppose that we have found a basic optimal solution $\bar{x}$ to the standard form of the LO relaxation (6.2), where $\bar{x}_i = \bar{b}_i \notin \mathbb{Z}$ for some $i \in B$. The corresponding constraint in the tableau is

$$x_i + \sum_{j \in N} \bar{a}_{ij} x_j = \bar{b}_i. \tag{6.3}$$

We claim that the following inequality is valid for all feasible solutions to the MILO problem (6.1)

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_{j \in N} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j \leq 0. \tag{6.4}$$

To see this, note that since $x_j \in \mathbb{Z}$ for each $j \in N$, we have

$$\lfloor \bar{b}_i \rfloor = \left\lfloor \sum_{j \in N} \bar{a}_{ij} x_j \right\rfloor \geq \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j.$$

By substituting (6.3) in the above inequality, we obtain the Gomory fractional cut (6.4).

Example 6.10. *Consider the same MILO problem*

$$\begin{aligned} \max \quad & 5.5x_1 + 2.1x_2 \\ \text{s. t.} \quad & -x_1 + x_2 \leq 2, \\ & 8x_1 + 2x_2 \leq 17, \\ & x_1, x_2 \in \mathbb{Z}_{\geq 0}. \end{aligned}$$

We may introduce slack variables $x_3, x_4 \in \mathbb{Z}_{\geq 0}$ for the inequality constraints, due to the integrality of the left-hand side coefficients and the right-hand sides.

$$\begin{aligned} \max \quad & 5.5x_1 + 2.1x_2 \\ \text{s. t.} \quad & -x_1 + x_2 + x_3 = 2, \\ & 8x_1 + 2x_2 + x_4 = 17, \\ & x_1, x_2, x_3, x_4 \in \mathbb{Z}_{\geq 0}. \end{aligned}$$

Suppose we have found the simplex tableau associated with the optimal solution $x_1 = 1.3, x_2 = 3.3$ to its LO relaxation.

z	x_1	x_2	x_3	x_4	*rhs*	*basis*
1	0	0	0.58	0.76	14.08	z
0	0	1	0.8	0.1	3.3	x_2
0	1	0	−0.2	0.1	1.3	x_1

In particular, the second row can be written as

$$x_2 + 0.8x_3 + 0.1x_4 = 3.3.$$

The Gomory fractional cut can be applied with $i = 2$, $N = \{3, 4\}$, $\bar{b}_2 = 3.3$, $\bar{a}_{23} = 0.8$, and $\bar{a}_{24} = 0.1$

$$0.8x_3 + 0.1x_4 \geq 0.3.$$

Since $x_3 = 2 + x_1 - x_2$ and $x_4 = 17 - 8x_1 - 2x_2$, this yields

$$x_2 \leq 3.$$

We plot this cut in Figure 6.1.

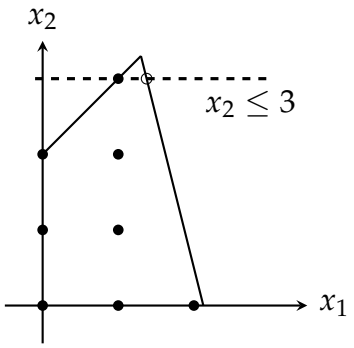


Figure 6.1: Illustration of a Gomory fractional cut

The idea of rounding coefficients can be used to generate cuts for the mixed-integer case, which is known as the *Gomory mixed-integer cuts*. Suppose that we have found a basic feasible solution $\bar{x}$ to the LO relaxation (6.2), with the corresponding row (6.3) where $i \in B \cap \{n_1 + 1, \ldots, n\}$. Let $N_1 := N \cap \{1, \ldots, n_1\}$, $N_2 := N \cap \{n_1 + 1, \ldots, n_1 + n_2\}$, $f_0 := \bar{b}_i - \lfloor \bar{b}_i \rfloor$, and $f_j := \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$, for $j \in N$.
The Gomory mixed integer cut can be written as

$$\sum_{\substack{j \in N_2: \\ f_j \leq f_0}} \frac{f_j}{f_0} x_j + \sum_{\substack{j \in N_2: \\ f_j > f_0}} \frac{1 - f_j}{1 - f_0} x_j + \sum_{\substack{j \in N_1: \\ \bar{a}_{ij} \geq 0}} \frac{\bar{a}_{ij}}{f_0} x_j - \sum_{\substack{j \in N_1: \\ \bar{a}_{ij} < 0}} \frac{\bar{a}_{ij}}{1 - f_0} x_j \geq 1. \tag{6.5}$$

Example 6.10 (continued). *Here we have $f_0 = 0.3$, $f_3 = 0.8$, and $f_4 = 0.1$. Note that we do not have any continuous variable $N_1 = \varnothing$, so the formula (6.5)*

$$\sum_{\substack{j \in N_2: \\ f_j \leq f_0}} \frac{f_j}{f_0} x_j + \sum_{\substack{j \in N_2: \\ f_j > f_0}} \frac{1 - f_j}{1 - f_0} x_j \geq 1$$

then becomes

$$\frac{1 - 0.8}{1 - 0.3} x_3 + \frac{0.1}{0.3} x_4 \geq 1 \iff 6x_3 + 7x_4 \geq 21.$$

Using $x_3 = 2 + x_1 - x_2$ and $x_4 = 17 - 8x_1 - 2x_2$, we can write the cut in terms of x_1 and x_2 as

$$5x_1 + 2x_2 \leq 11.$$

We plot this cut in Figure 6.2. Compared with Figure 6.1, it shows that Gomory mixed-integer cut is stronger, *in the sense that it "cuts off" more non-integer points in the LO relaxation.*

The proof of the Gomory mixed-integer cuts (6.5) consists of a sequence of mixed-integer cuts, which are presented below.

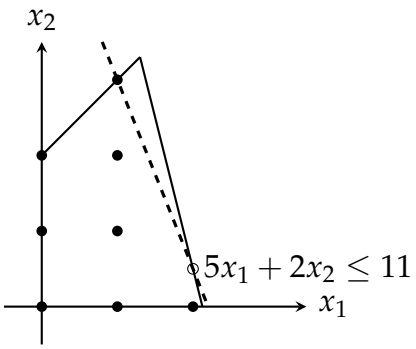


Figure 6.2: Illustration of a Gomory mixed-integer cut

Lemma 6.11. *If* $x_1 + \sum_{j=2}^{n} x_j \geq b$, $x_1 \geq 0$, $x_2, \ldots, x_n \in \mathbb{Z}$, *and* $f := b - \lfloor b \rfloor > 0$, *then*

$$\frac{x_1}{f} + \sum_{j=2}^{n} x_j \geq \lceil b \rceil.$$

Proof. Note that

$$f\left(\lceil b \rceil - \sum_{j=2}^{n} x_j\right) = f + f\left(\lfloor b \rfloor - \sum_{j=2}^{n} x_j\right) \leq f + \left(\lfloor b \rfloor - \sum_{j=2}^{n} x_j\right) = b - \sum_{j=2}^{n} x_j \leq x_1.$$

Divide both sides by f and we are done. □

Lemma 6.12. *If* $\sum_{j=2}^{n} x_j \leq b + x_1$, $x_1 \geq 0$, $x_2, \ldots, x_n \in \mathbb{Z}$, *and* $f := b - \lfloor b \rfloor > 0$, *then*

$$\sum_{j=2}^{n} x_j \leq \lfloor b \rfloor + \frac{x_1}{1-f}.$$

Proof. Apply Lemma 6.11 with $-b$ and $-x_j$ for $j = 2, \ldots, n$. Note that $\lfloor -b \rfloor = -\lceil b \rceil$ and $-b + \lfloor -b \rfloor = 1 - f$. □

Lemma 6.13. *Consider* $\sum_{j=2}^{n} a_j x_j \leq b + x_1$, *where* $x_1 \geq 0$ *and* $x_j \in \mathbb{Z}_{\geq 0}$ *for* $j = 2, \ldots, n$. *Let* $f := b - \lfloor b \rfloor$ *and* $f_j := a_j - \lfloor a_j \rfloor$ *for* $j = 2, \ldots, n$. *Then*

$$\sum_{j: f_j \leq f} \lfloor a_j \rfloor x_j + \sum_{j: f_j > f} \left(\lfloor a_j \rfloor + \frac{f_j - f}{1-f}\right) x_j \leq \lfloor b \rfloor + \frac{x_1}{1-f}.$$

Proof. Note that

$$\sum_{j: f_j \leq f} \lfloor a_j \rfloor x_j + \sum_{j: f_j > f} \lceil a_j \rceil x_j \leq b + x_1 + \sum_{j: f_j > f} (1 - f_j) x_j.$$

The left-hand side are all integers, so we can apply Lemma 6.12 with $x_1 + \sum_{j: f_j > f} (1 - f_j) x_j \geq 0$ regarded as the continuous value. □

Proposition 6.14 (Gomory mixed-integer cuts). *If $x \in \mathbb{R}^{n_1}_{\geq 0} \times \mathbb{Z}^{n_2}_{\geq 0}$ satisfies (6.3), then it also satisfies (6.5). Moreover, the point $x_i = \bar{b}_i$ with $x_j = 0$ for all $j \in N$ does not.*

Proof. Note that the equation (6.3) implies

$$x_i + \sum_{j \in N_2} \bar{a}_{ij} x_j \leq \bar{b}_i + \sum_{\substack{j \in N_1: \\ \bar{a}_{ij} < 0}} -\bar{a}_{ij} x_j.$$

Now apply Lemma 6.13 and we get

$$x_i + \sum_{\substack{j \in N_2: \\ f_j \leq f}} \lfloor \bar{a}_{ij} \rfloor x_j + \sum_{\substack{j \in N_2: \\ f_j > f}} \left(\lfloor \bar{a}_{ij} \rfloor + \frac{f_j - f_0}{1 - f_0} \right) \leq \lfloor \bar{b}_i \rfloor + \sum_{\substack{j \in N_1: \\ \bar{a}_{ij} < 0}} \frac{-\bar{a}_{ij}}{1 - f_0} x_j.$$

Now substitute x_i using the equation (6.3) and we are done. The last claim follows from $f_0 > 0$. □

We remark that there are many more types cuts used in MILO problems, most of which exploits some further structure of the data A and b. Due to the limit of this course, we refer any interested reader to the textbook **[ConfortiCornuéjolsZambelli2014]** for further study.

6.4 More on Integer Modeling

Example 6.15. *A salesperson wants to visit n cities. The distance between the different cities $i, j \in \{1, \ldots, n\}$ is denoted as d_{ij}. Starting at city 1, the salesperson must visit each city* exactly once *and then return to city 1 (see Figure 6.3). The goal is to minimize the total distance traveled. To model this problem, we can define variables for each $i \neq j$, $i, j \in \{1, \ldots, n\}$*

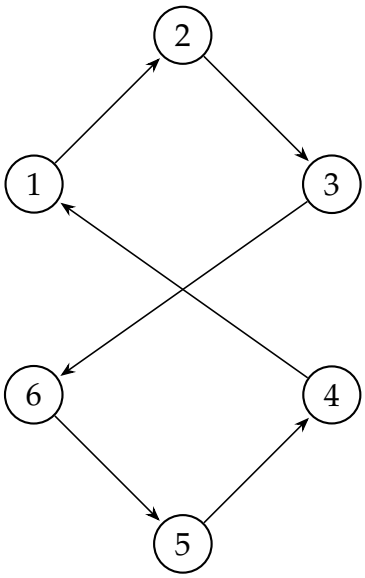
$$x_{ij} = \begin{cases} 1, & \text{if the salesperson travels from city } i \text{ to city } j, \\ 0, & \text{otherwise.} \end{cases}$$

The objective is to minimize the total distance

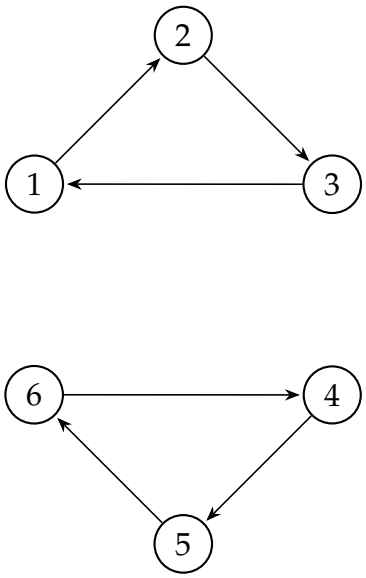
$$\min \quad \sum_{i=1}^{n} \sum_{j \neq i} d_{ij} x_{ij}.$$

Note that the salesperson should arrive at and leave from each city exactly once, *which can be written as the constraints*

$$\sum_{j \neq i} x_{ij} = 1, \quad i = 1, \ldots, n,$$
$$\sum_{i \neq j} x_{ij} = 1, \quad j = 1, \ldots, n.$$



(a) a feasible travel plan



(b) an infeasible travel plan

Figure 6.3: Traveling plans of visiting 6 cities

The travel plan in Figure 6.3b satisfies all these constraints, but is still infeasible as we go back to city 1 (where we started) before we visited all of the 6 cities. We call the paths such as $1 \to 2 \to 3 \to 1$ subtours, *which need to be eliminated by additional constraints*

$$\sum_{i \in S, j \notin S} x_{ij} \geq 1, \quad \text{for all subsets } S \subsetneq \{1, \ldots, n\}, \quad 2 \leq |S| \leq n-2.$$

The number of the subtour elimination constraints in Example 6.15 is typically huge: for $n \geq 4$, we would have $2^n - 2 - 2n$ possible subtours, which is over 1 million for a mere number of 20 cities! People thus often use a technique called *constraint generation* to solve this problem. The idea is to solve the problem with only a subset of the constraints and dynamically add the rest as needed. In Example 6.15, once we get a solution $\bar{x}_{ij}$, we can start out tour from city 1 and move to the city j such that $\bar{x}_{1j} = 1$. By repeating the procedure, we would either visit all the cities before going back to city 1, or find a subtour $\bar{S}$. In the former case, we have successfully solved the traveling salesperson problem, while in the latter we need to add the subtour elimination constraint for $\bar{S}$ and resolve the problem.

Example 6.16. *A paper mill produces large rolls of paper of width W, which are then cut into rolls of various smaller widths in order to meet demand. Let m be the number of different widths that the mill produces. The mill receives an order for b_i rolls of width w_i for $i = 1, \ldots, m$, where $w_i \leq W$. The goal is to find the smallest number of large rolls needed to meet the demand.*
One way to formulate our MILO model is as follows. Suppose p is an upper bound on the number of paper rolls, such as $p = \sum_{i=1}^{m} b_i$. We can define our decision variables for $j =$

$1, \ldots, n$ *as*

$$y_j = \begin{cases} 1, & \text{if the large roll } j \text{ is used,} \\ 0, & \text{otherwise,} \end{cases}$$

and for $i = 1, \ldots, m$, $j = 1, \ldots, p$,

$$z_{ij} \in \mathbb{Z}_{\geq 0} : \text{ the number of rolls of width } w_i \text{ to be cut out of roll } j.$$

Then our MILO model can be written as

$$\begin{aligned} \min \quad & \sum_{j=1}^{p} y_j \\ \text{s. t.} \quad & \sum_{i=1}^{m} w_i z_{ij} \leq W y_i, \quad i = 1, \ldots, p, \\ & \sum_{j=1}^{p} z_{ij} \geq b_i, \quad i = 1, \ldots, m, \\ & y_j \in \{0, 1\}, \quad j = 1, \ldots, p, \\ & z_{ij} \in \mathbb{Z}_{\geq 0}, \quad i = 1, \ldots, m, \ j = 1, \ldots, p. \end{aligned} \tag{6.6}$$

This model has some disadvantages: its LO relaxation is usually weak, and there is no easy way to round fractional solutions from the LO relaxation into feasible integer solutions because we have inequality constraints of both directions. Alternatively, people consider the following formulation. Let $s \in \mathbb{Z}^m$ *denote a cutting pattern, where* s_i *rolls of width* w_i *are cut out of the large paper roll. The set of all cutting patterns is*

$$\mathcal{S} := \left\{ s \in \mathbb{Z}^m_{\geq 0} : \sum_{i=1}^{m} w_i s_i \leq W \right\}.$$

Now we can define for each $s \in \mathcal{S}$,

$$x_s \in \mathbb{Z}_{\geq 0} : \text{ the number of rolls cut according to the pattern } s.$$

The only constraints are

$$\sum_{s \in \mathcal{S}} s_i x_s \geq b_i, \quad i = 1, \ldots, m.$$

Thus our alternative MILO model is

$$\begin{aligned} \min \quad & \sum_{s \in \mathcal{S}} x_s \\ \text{s. t.} \quad & \sum_{s \in \mathcal{S}} s_i x_s \geq b_i, \quad i = 1, \ldots, m, \\ & x_s \in \mathbb{Z}_{\geq 0}, \quad s \in \mathcal{S}. \end{aligned} \tag{6.7}$$

The number of variables $|\mathcal{S}|$ can be potentially large, which motivates people to use the column generation *algorithm, as outlined below. The dual of its LO relaxation is*

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} b_i u_i \\ \text{s. t.} \quad & \sum_{i=1}^{m} s_i u_i \le 1, \quad \forall\, s \in \mathcal{S}, \\ & u_1, \dots, u_m \ge 0. \end{aligned} \tag{6.8}$$

Suppose we use a subset $\mathcal{S}'$ of $\mathcal{S}$ and find a primal optimal solution $\bar{x}_s$, $s \in \mathcal{S}'$ and a dual optimal solution $\bar{u}_1, \dots, \bar{u}_m$. We can extend $\bar{x}_s$ to all of $\mathcal{S}$ by setting $\bar{x}_s = 0$ for all $s \in \mathcal{S} \setminus \mathcal{S}'$. Thus when the dual solution $\bar{u}$ is feasible, we know that $\bar{x}$ is optimal by the weak duality. This is equivalent to check that the constraints

$$\sum_{i=1}^{m} s_i \bar{u}_i \le 1$$

are satisfied for all $s \in \mathcal{S}$. Or equivalently, if the following problem

$$\max \quad \sum_{i=1}^{m} \bar{u}_i s_i, \quad s \in \mathcal{S} \tag{6.9}$$

has an optimal value is no more than 1, then $\bar{u}$ is feasible. Otherwise any $s \in \mathcal{S}$ such that $\sum_{i=1}^{m} \bar{u}_i s_i > 1$ can be added as a new variable $S' \leftarrow S' \cup \{s\}$. The problem (6.9) is in fact a knapsack problem (cf. Example 6.4) by the definition of $\mathcal{S}$.

Example 6.17. *The famous puzzle game* sudoku *can be formulated as an integer optimization problem. The goal is to fill numbers $\{1, 2, \dots, 9\}$ in the empty locations and the rule is that every number should appear only once in each row, in each column, and in each 3×3 square. Some numbers are provided in the beginning of the puzzle that cannot be changed. We can define variables for $i, j, k \in \{1, 2, \dots, 9\}$,*

$$x_{i,j,k} = \begin{cases} 1, & \text{if the number } k \text{ is selected at the location } (i, j), \\ 0, & \text{otherwise.} \end{cases}$$

The constraints consist of the following. Exactly one number should be selected at each location:

$$\sum_{k=1}^{9} x_{i,j,k} = 1, \quad i, j = 1, \dots, 9.$$

For the given numbers, we fix the corresponding variables to 1:

$$x_{i,j,k} = 1, \quad \text{if } S_{ij} = k.$$

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

Figure 6.4: A sudoku puzzle

For each row and column, every number should appear only once:

$$\sum_{j=1}^{9} x_{i,j,k} = 1, \quad i,k = 1,\ldots,9,$$

and

$$\sum_{i=1}^{9} x_{i,j,k} = 1, \quad j,k = 1,\ldots,9.$$

For each block, every number should appear only once:

$$\sum_{i''=1}^{3}\sum_{j''=1}^{3} x_{3(i'-1)+i'',3(j'-1)+j'',k} = 1, \quad i',j' = 1,\ldots,3,\ k = 1,\ldots,9.$$

The objective is not needed, or we can set trivially

$$\min \quad 0.$$

We can code this model in `model_sudoku.py` *and get the following solution to the puzzle shown in Figure 6.4.*

```
Found a solution to the sudoku puzzle:
2 5 8 7 3 6 9 4 1
```

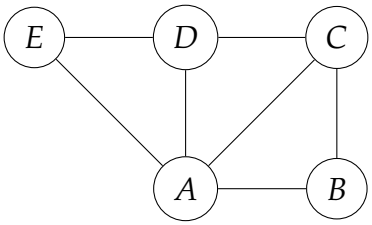
```
6 1 9 8 2 4 3 5 7
4 3 7 9 1 5 2 6 8
3 9 5 2 7 1 4 8 6
7 6 2 4 9 8 1 3 5
8 4 1 6 5 3 7 2 9
1 8 4 3 6 9 5 7 2
5 7 6 1 4 2 8 9 3
9 2 3 5 8 7 6 1 4
```

7 Graphs and Network Optimization

7.1 Graphs and Networks

A graph $G = (N, E)$ consists of a set of *nodes* N and a set of *edges* E, which are represented by pairs of nodes. For example, $N = \{1, 2, \ldots, n\}$ and $E = \{(1,2), (1,3), \ldots, (1,n)\}$. We remark that there could be multiple edges represented by the same pair of nodes. If there is at most one edge between a pair of nodes, then G is called a *simple graph*.

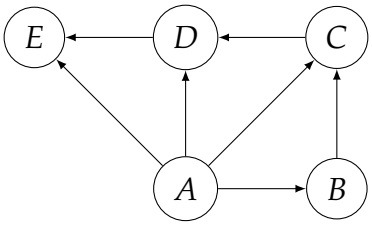
Example 7.1. *Here is a graph with 5 nodes.*



The nodes are denoted by A, B, C, D, E, and the edges are represented by the unordered pairs (A, B), (A, C), (A, D), (A, E), (B, C), (C, D), (D, E).

A graph is undirected if the edges have no "directions," i.e., (i, j) is the same as (j, i) for any $i, j \in N$. Otherwise the graph is directed and sometimes called a *digraph*. The nodes and edges in directed graphs are often called *vertices* and *arcs*.

Example 7.2. *Here is a directed graph with 5 vertices.*



The vertices are denoted by A, B, C, D, E, and the arcs are represented by the ordered pairs (A, B), (A, C), (A, D), (A, E), (B, C), (C, D), (D, E).

An undirected graph can be viewed as a directed graphs where we have arcs with both directions for each edge.

We say that a node $j \in N$ is *adjacent* to $i \in N$ if $(i, j) \in E$. An edge (or arc) $(i, j) \in E$ is said to be *incident* to nodes $i, j \in N$. When the graph is directed, we also say that it *emanates* from node i and *terminates* at node j; it is an *outgoing* arc of i and *incoming* arc of j. The *degree* of a node is the number of its incident edges. The *in-degree* of a vertex is the number of its incoming arcs, and the *out-degree* of a vertex is the number of its outgoing arcs.

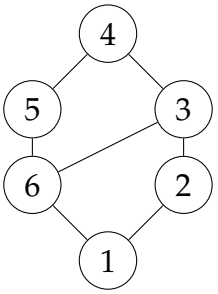
A *walk* is a sequence of alternating nodes and edges

$$i_1, e_1, i_2, e_2, \ldots, i_{k-1}, e_k, i_k,$$

where $i_1, \ldots, i_k \in N$ and $e_j = (i_j, i_{j+1}) \in E$ for $j = 1, \ldots, k-1$. It is called *closed* if $i_1 = i_k$. We also denote a walk by $i_1, i_2, \ldots, i_k$ or $e_1, \ldots, e_k$. A *trail* is a walk where $e_1, \ldots, e_k$ are distinct. A *path* is a walk where $i_1, \ldots, i_k$ are distinct. A path is always a trail, but the converse is not necessarily true. These definitions apply to directed graphs as well.

A *circuit* is a closed trail. A *cycle* is a circuit $i_1, i_2, \ldots, i_k, i_1$ where $i_1, \ldots, i_k$ are distinct. A graph without any cycles is called *acyclic*. A graph is *connected* if for any pair of nodes, there is a path connecting them. A connected, acyclic graph is called a *tree*.

Example 7.3. *In the following graph,*



$1, 2, 1$ *is a walk but not a trail;* $3, 2, 1, 6, 5, 4, 3, 6$ *is a trail but not a path;* $1, 2, 3, 4, 5, 6, 1$ *is a circuit and a cycle. The graph is connected but not a tree.*

An *Eulerian trail* is a trail $i_1, e_1, i_2, \ldots, i_{k-1}, e_{k-1}, i_k$ such that each edge e_j appears only once in the trail for $j = 1, \ldots, k-1$. An *Eulerian circuit* is an Eulerian trail that is also a circuit. Historically, one of the first graph theory problems studied was the Eulerian trail/circuit problem, motivated by the Königsberg seven bridge problem. It can be formulated as the existence of an Eulerian trail/path in the following (nonsimple) graph (Figure 7.1).

The seven bridge problem has a simple answer using the above definitions: a connected undirected graph admits an Eulerian trail (resp. circuit) if at most two (resp. none) of the nodes have odd degrees. This is because in an Eulerian trail, except for

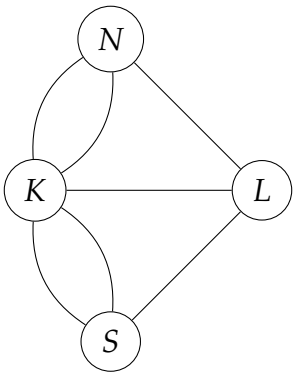


Figure 7.1: Königsberg seven bridge problem

the starting and ending nodes, we must arrive at and leave from each node the same amount of times, so the degree must be even. Conversely, if none of the nodes has an odd degree, then each time we arrive at the node, there is an edge we can use to leave from this node. The same argument applies to Eulerian trail by connecting the two odd-degree nodes (if there are any). Therefore, we can say that the Eulerian trail/circuit problem is easy as we only need to count the degrees of the nodes.

The following *vertex coloring* problem can be much more challenging: given a graph $G = (V, E)$, we want to color the vertices (nodes) such that no adjacent vertices share the same color. The minimum number of colors we need is called the *chromatic number* of the graph. A *heuristic* method to find a vertex coloring is as follows.

(i) Start with a vertex and use color $C = \{1\}$.

(ii) Go to a vertex $i \in N$.

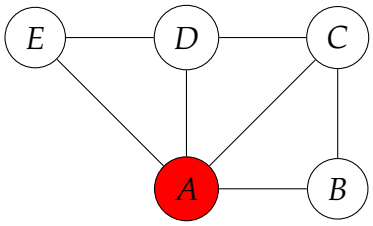
  (a) If there is a color $c \in C$ that is not used by any adjacent node j of i, color i with c;

  (b) otherwise add a new color to c' to C and color i with c'.

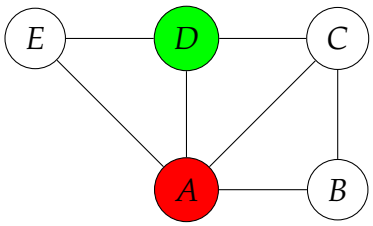
(iii) Repeat Step 2 until all vertices are colored.

Example 7.4. *The following example illustrates the heuristic way of coloring the vertices.*

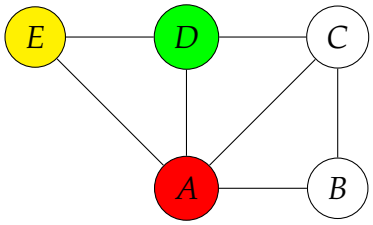
Step 1. *Used color: red*



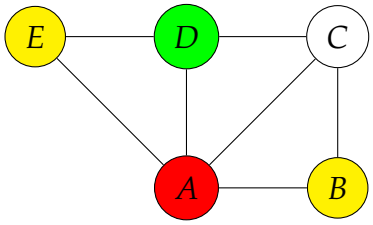
Step 2. *Used colors: red, green*



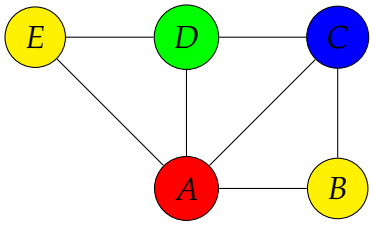
Step 3. *Used color: red, green, yellow*



Step 4. *We can color node B with the yellow color.*

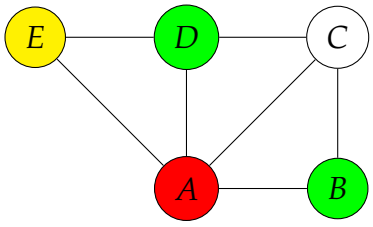


Step 5. *Used color: red, green, yellow, blue*

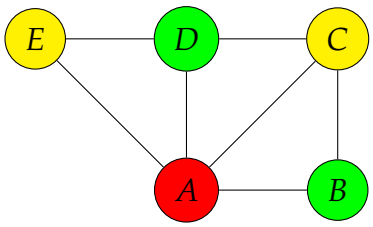


In this way, we can color the graph with 4 colors. However, depending on the ordering of the colors, we may be able to use fewer colors. For example, in step 4, if we color node B with the green color, then we would get the following.

Step 4. *Used color: red, green, yellow*



Step 5. *Used color: red, green, yellow*



This means that we only need at most 3 colors for this graph.

The chromatic number for the graph in Example 7.4 is indeed 3. To see this, note that the graph contains triangles, such as $\{A, D, E\}$ and $\{A, B, C\}$. Each of these triangles would require 3 different colors, as each vertex is adjacent to the other two. Therefore, the graph would require at least 3 colors. This argument can be generalized by the notion of *cliques* or *complete subgraphs*, meaning a subset of the vertices that are all adjacent to each other.

We are fortunate in finding the chromatic number in Example 7.4 by the above heuristic. In general, such heuristic can only be used for an upper bound on the chromatic number, which may not equal the lower bound provided by cliques. Nevertheless, it can be used to formulate the vertex coloring problem as a integer linear optimization as follows. Suppose we need at most c colors, which can be found by the above heuristic. For $i \in N$ and $k = 1, \ldots, c$, let

$$x_{ik} = \begin{cases} 1, & \text{if the vertex } i \text{ is colored by } k, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$y_k = \begin{cases} 1, & \text{if the color } k \text{ is used,} \\ 0, & \text{otherwise.} \end{cases}$$

The vertex coloring problem can be formulated as

$$\begin{aligned} \min \quad & \sum_{k=1}^{c} y_k \\ \text{s. t.} \quad & \sum_{k=1}^{c} x_{ik} = 1, && i \in N, \\ & x_{ik} + x_{jk} \le 1, && (i,j) \in E,\ k = 1, \ldots, c, \\ & x_{ik} \le y_k, && i \in N,\ k = 1, \ldots, c, \\ & x_{ik}, y_k \in \{0, 1\}, && i \in N,\ k = 1, \ldots, c. \end{aligned}$$

The first constraint ensures that we are coloring each vertex with exactly one color; the second constraint ensures that adjacent vertices do not use the same color; the third constraint checks if a color is used on any of the vertices. The objective function is the number of colors used on the graph.

7.2 Network Flow Problems

A directed graph $G = (N, A)$ can be used to represent pipelines or transportation networks. Each arc $(i, j) \in A$ is associated with a *flow* variable x_{ij}. At each node $i \in N$,

we have the *flow balance* constraint

$$\sum_{j:(j,i)\in A} x_{ji} - \sum_{j:(i,j)\in A} x_{ij} = f_i,$$

where f_i is the extraction/injection of the flow at node $i \in N$.

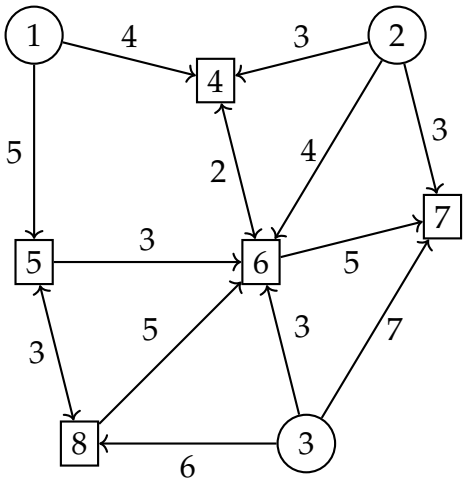
Example 7.5. *A water pipeline is built to deliver water to residential locations in a village. The water sources are listed below*

source	*1*	*2*	*3*
capacity	*100*	*100*	*80*

and the residential demands are listed below.

residence	*4*	*5*	*6*	*7*	*8*
demand	*50*	*60*	*40*	*30*	*70*

Each pipeline has its unit cost of delivering water, due to geographical differences. Below is the water network, where circle nodes are water sources and square nodes are residential locations. The arrows indicate the directions and the numbers indicate the unit costs for the pipelines. The



goal is to find the minimum total water delivery cost while satisfying the residential demands.

Let c_{ij} denote the cost on the arc $(i,j) \in A$, d_i denote the supply/demand at the node $i \in N$. We define variables

$$x_{ij} \geq 0 : \text{ water flow in the directed pipeline } (i,j) \in A,$$

and

$$y_i \in \mathbb{R} : \text{ water flowing out of the network at the node } i \in N.$$

Note that it is possible to have both x_{ij} and x_{ji}. If $y_i < 0$, then it means water flows into the network at the node $i \in N$. We have the water flow balance constraint

$$\sum_{j:(j,i)\in A} x_{ji} - \sum_{j:(i,j)\in A} x_{ij} = y_i, \forall\, i \in N.$$

At water sources, we have

$$d_i \leq y_i \leq 0, \quad i = 1, 2, 3.$$

At residential locations, we have

$$y_i \geq d_i, \quad i = 4, 5, \ldots, 8.$$

Then objective function is

$$\min \quad \sum_{(i,j)\in A} c_{ij} x_{ij}.$$

We code this LO model in `model_water_network.py` *and get the following results.*

```
The minimum cost is 1120.00.
The flow in each pipeline is shown below.
x(1, 4) = 10.000000000000004
x(1, 5) = 60.0
x(5, 6) = 0.0
x(8, 5) = 0.0
x(5, 8) = 0.0
x(8, 6) = 0.0
x(3, 8) = 70.0
x(3, 6) = 9.999999999999991
x(3, 7) = 0.0
x(6, 4) = 0.0
x(4, 6) = 0.0
x(2, 4) = 40.0
x(2, 6) = 30.000000000000007
x(2, 7) = 30.0
x(6, 7) = 0.0
```

If we have limits on the flows, then it may not be possible to transport any amount from the source to the target (or *sink*). We only inject flow at the source $s \in N$ and extract flow at the sink $t \in N$ and solve the above network flow problem. To simply the notation, we can create an artificial arc (t, s) and

$$\begin{aligned} \max \quad & x_{ts} \\ \text{s. t.} \quad & \sum_{j:(j,i)\in A'} x_{ji} - \sum_{j:(i,j)\in A'} x_{ij} = 0, \quad \forall\, i \in N, \\ & 0 \leq x_{ij} \leq u_{ij}, \qquad\qquad \forall\, (i,j) \in A, \end{aligned} \tag{7.1}$$

where $A' = A \cup \{(t,s)\}$ and u_{ij} is the given limit for arc $(i,j) \in A$. The dual problem

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} u_{ij} y_{ij} \\ \text{s.t.} \quad & z_t - z_s \geq 1, \\ & z_i - z_j + y_{ij} \geq 0, \qquad \forall\, (i,j) \in A, \\ & y_{ij} \geq 0, \qquad \forall\, (i,j) \in A, \end{aligned} \tag{7.2}$$

has a nice interpretation, known as the *minimum cut* problem. Given a graph $G = (N, A)$, an *s-t cut* is a partition of $N = S \cup T$, $S \cap T = \varnothing$, such that $s \in S$ and $t \in T$. Given the capacity of each arc u_{ij}, the capacity of the cut S, T is the sum

$$\sum_{i\in S, j\in T, (i,j)\in A} u_{ij}.$$

To find a cut with minimum capacity, we can formulate it as an integer linear optimization. For each $i \in N$, let

$$z_i = \begin{cases} 1 & \text{if } i \in T, \\ 0 & \text{if } i \in S, \end{cases}$$

and for each $(i,j) \in A$, let

$$y_{ij} = \begin{cases} 1 & \text{if } i \in S \text{ and } j \in T, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly we must have constraints

$$y_{ij} \geq z_j - z_i, \quad \forall\, (i,j) \in A.$$

Since we are looking for an *s-t* cut, we should have

$$z_t = 1,\ z_s = 0 \iff z_t - z_s = 1.$$

The objective is to minimize the sum

$$\min \quad \sum_{(i,j)\in A} u_{ij} y_{ij}.$$

To summarize, the minimum cut problem can be written as

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} u_{ij} y_{ij} \\ \text{s.t.} \quad & y_{ij} \geq z_j - z_i, \qquad \forall\, (i,j) \in A, \\ & z_t - z_s = 1, \\ & y_{ij} \in \{0,1\}, \qquad \forall\, (i,j) \in A, \\ & z_i \in \{0,1\}, \qquad \forall\, i \in N. \end{aligned} \tag{7.3}$$

In fact, it can be proved that the MILO problem (7.3) is equivalent to the LO problem (7.2). The main idea is that all of the *minors* of the constraint coefficient matrix in (7.2) is ± 1 and therefore the simplex tableau will only consist of coefficients ± 1. Hence, the basic variable values will be either 0 or 1, as the right-hand side constants.

Theorem 7.6. *For any network $G = (N, A)$, the maximum flow in (7.1) equals the minimum cut (7.3).*

The theorem is connected to the *Ford-Fulkerson* algorithm, that directly calculates the maximum flow by iterative improvement on the current flows until reaching the capacities in some arcs. To be precise, we introduce two basic operations in the algorithm.

- Labeling path:
  - Label the source.
  - Given a labeled node i, label the node j if either
    - the arc (i,j) is a *forward arc*: $0 \leq x_{ij} < u_{ij}$; or
    - the arc (j,i) is a *backward arc*: $0 < x_{ij} \leq u_{ij}$.
  - Repeat the previous step until reaching the sink or no more nodes can be labeled.
- Augmenting flow:
  - Decrease the flow that would be feasible for all backward arcs:

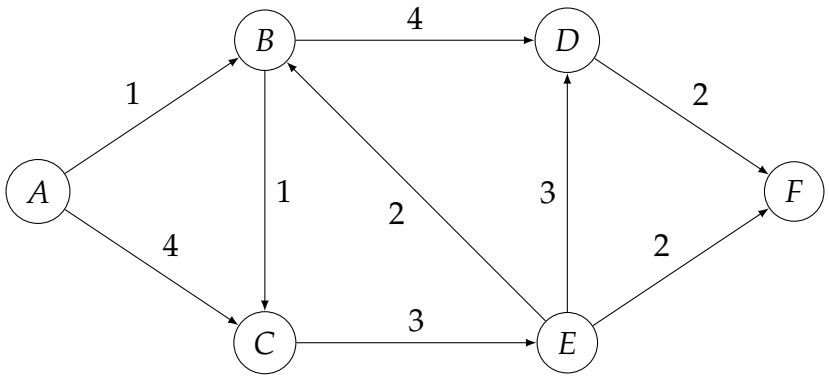
$$\Delta_{\mathrm{b}} := \min\{x_{ij} : (i,j) \text{ is a backward arc}\}.$$

  - Increase the flow that would be feasible for all forward arcs:

$$\Delta_{\mathrm{f}} := \min\{u_{ij} - x_{ij} : (i,j) \text{ is a forward arc}\}.$$

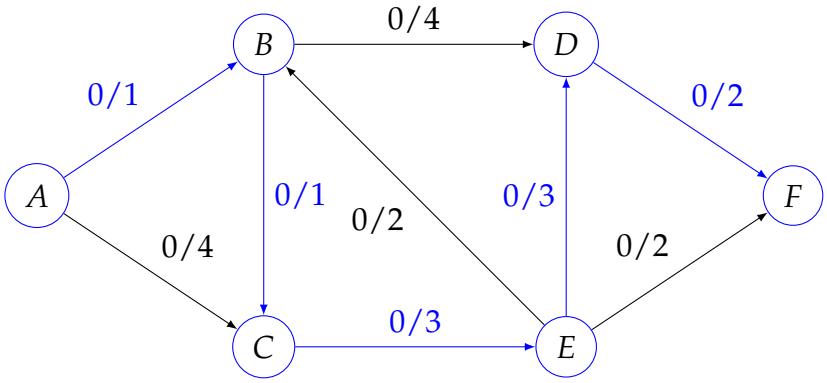
  - Let $\Delta := \min\{\Delta_{\mathrm{b}}, \Delta_{\mathrm{f}}\}$. We can then improve the current solution by sending Δ units of flow from source to sink via the augmenting path:
    - increase the flow for all forward arcs in the path by Δ, and
    - decrease the flow for all backward arcs in the path by Δ.

Example 7.7. *We want to find the maximum flow from A to F, where the capacities are labeled on the arcs.*



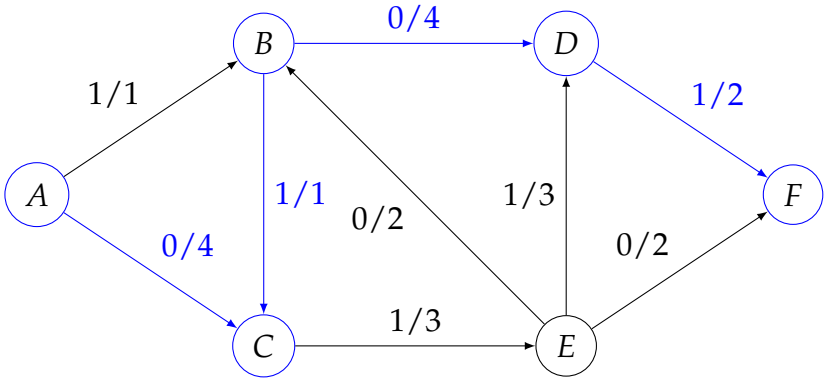
The iterations of Ford-Fulkerson algorithm are executed as follows.

Iteration 1: Labeling path A,B,C,E,D,F



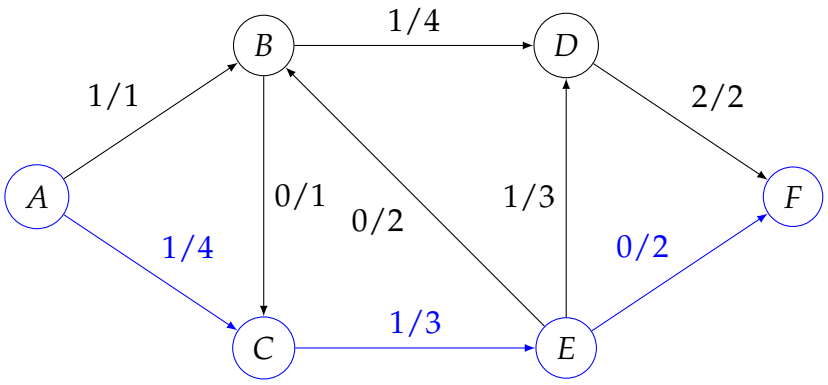
Augmenting flow $\Delta = \min\{1, 1, 3, 3, 2\} = 1$

Iteration 2: Labeling path A,C,B,D,F



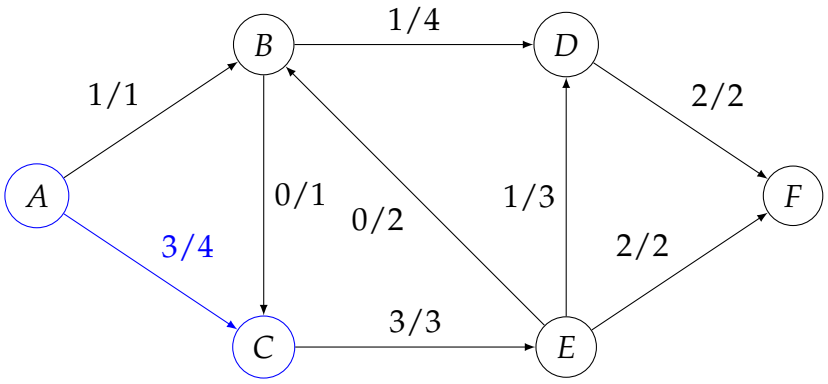
Augmenting flow $\Delta = \min\{4, 1, 4, 1\} = 1$

Iteration 3: Labeling path A,C,E,F



Augmenting flow $\Delta = \min\{3, 2, 2\} = 2$

Iteration 4: Labeling path A,C, and no more nodes can be labeled



Terminate the algorithm. The maximum flow from A to F is 4.

To see the connection to the minimum cut problem, we construct the cut S, T when the algorithm terminates. Let S be all the nodes that is still connected to the source with the remaining capacities. Clearly, S does not contain the sink. All arcs from S to T contain all the flows from the source to the sink, the sum of which equal to the capacity of this cut. Thus by LO duality, we know that S, T is a minimum cut.

7.3 Shortest Path Problem

Given a directed graph $G = (N, A)$ with each arc (i, j) associated with some *costs* c_{ij}, we want to find a path from a node $s \in N$ to a node $t \in N$ with the minimum total cost. Here we use the term "cost" instead of just distance because the shortest path problem can be used for modeling some general decision problems.

Example 7.8. *Suppose we have purchases a new machine for \$24,000 at time 0. The cost of maintaining the machine during a year and its trade-in price are given in the table below.*

Age	*Annual Maintenance Cost ($)*	*Age*	*Trade-in Price ($)*
0	*4,000*	*1*	*14,000*
1	*8,000*	*2*	*12,000*
2	*10,000*	*3*	*4,000*
3	*18,000*	*4*	*2,000*
4	*24,000*	*5*	*0*

Assume that at any time it costs $24,000 to purchase a new machine. Our goal is to minimize the net cost incurred during the next five years. We can model this problem using a graph.

- *Our network will have six nodes corresponding to the beginning of years 1-6.*
- *Node i is the beginning of year i and for $i < j$, an arc (i, j) corresponds to purchasing a new machine at the beginning of year i and keeping it until the beginning of year j.*
- *The cost on the arc (i, j) is the total net cost incurred from years i to j:*

$$\begin{aligned} c_{ij} = {} & \textit{maintenance cost incurred during years } i, i+1, \ldots, j-1 \\ & + \textit{ cost of purchasing a machine at the beginning of year } i \\ & - \textit{ trade-in value received at the beginning of year } j. \end{aligned}$$

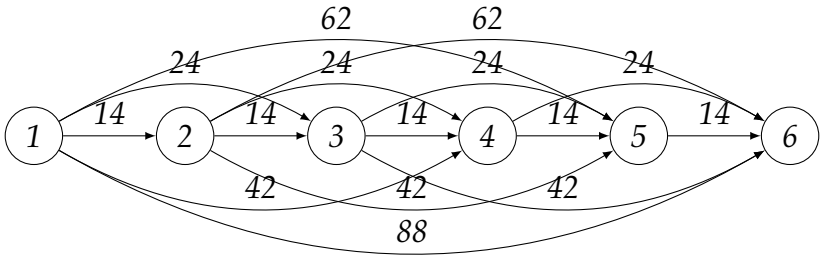
For example, (in $1,000)

$$c_{12} = 4 + 24 - 14 = 14,$$

and

$$c_{26} = 4 + 8 + 10 + 18 + 24 - 2 = 62.$$

The network is then plotted as follows.



Then the maintenance problem reduces to finding a shortest/cheapest path from node 1 *to node* 6.

Any shortest path problem can be formulated as a MILO model. Define for each $(i, j) \in A$,

$$x_{ij} = \begin{cases} 1 & \text{if the arc } (i, j) \text{ is selected in the path,} \\ 0 & \text{otherwise,} \end{cases}$$

and for each $i \in N \setminus \{s, t\}$

$$y_i = \begin{cases} 1 & \text{if the node } (i, j) \text{ is visited in the path,} \\ 0 & \text{otherwise.} \end{cases}$$

There is one outgoing arc for the starting node

$$\sum_{j:(s,j)\in A} x_{sj} = 1,$$

one incoming arc for the terminating node

$$\sum_{j:(j,t)\in A} x_{jt} = 1,$$

and one incoming arc and one outgoing arc for each visited node

$$\sum_{j:(j,i)\in A} x_{ji} = y_i,$$
$$\sum_{j:(i,j)\in A} x_{ij} = y_i.$$

The objective is to minimize the total cost

$$\min \quad \sum_{(i,j)\in A} c_{ij} x_{ij}.$$

Alternatively, we can also solve the shortest path problem efficiently through specialized algorithms.

7.3.1 Dijkstra's Algorithm

The algorithm can be described as follows.

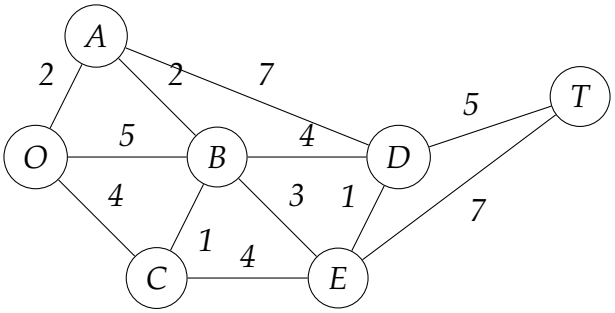
(i) Mark s as solved with cost 0, and all other nodes as unsolved.

(ii) For any node adjacent to the last solved node, calculate the sum of the edge cost and the minimum cost of the last solved node, and
- if it is not a candidate, set it to be a candidate with tentative cost being the sum;
- otherwise, if the sum is smaller than the incumbent tentative cost, update the tentative cost and predecessor to be the sum and the last solved node.

(iii) Pick a candidate node with the smallest tentative cost and mark it as solved.
- If the destination is solved or if there is no more candidate node, terminate the algorithm.

- Otherwise, go back to Step 2.

Example 7.9. *We want to find a shortest path from node O to node T. The cost of each edge is marked on the graph.*



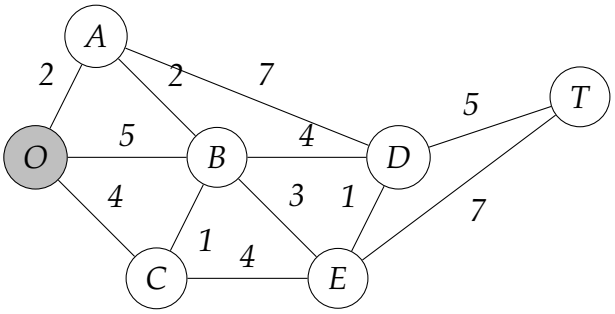
The iterations are illustrated as follows.

Iteration 1: solved nodes (with min. cost, predecessor)

- *O*, $(0, N/A)$

candidate nodes (with tentative cost, predecessor)

- *A (2, O)*
- *B (5, O)*
- *C (4, O)*

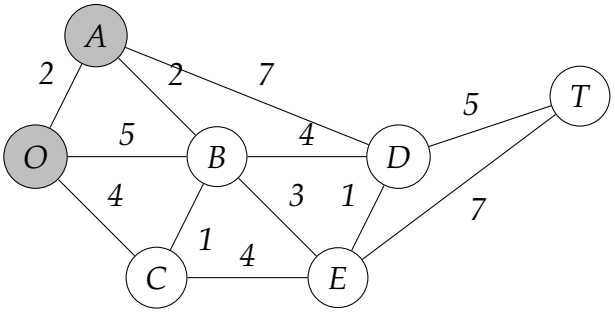


Iteration 2: solved nodes

- *O (0, N/A)*
- *A (2, O)*

candidate nodes

- *B (5, O)* $\rightarrow$ *(4, A)*
- *C (4, O)*
- *D (9, A)*

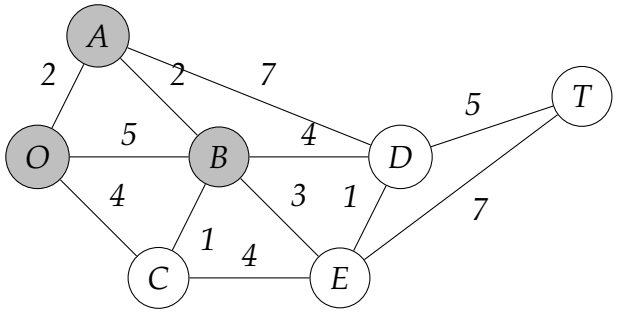


Iteration 3: *solved nodes*

- *O (0, N/A)*
- *A (2, O)*
- *B (4, A)*

candidate nodes

- *C (4, O)*
- *D (9, A)* $\rightarrow$ *(8, B)*
- *E (7, B)*

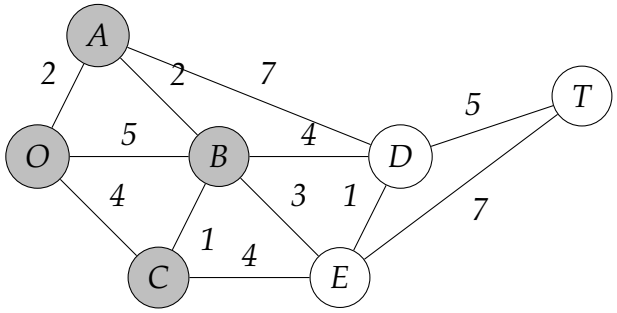


Iteration 4: *solved nodes*

- *O (0, N/A)*
- *A (2, O)*
- *B (4, A)*
- *C (4, O)*

candidate nodes

- *D (8, B)*
- *E (7, B)*



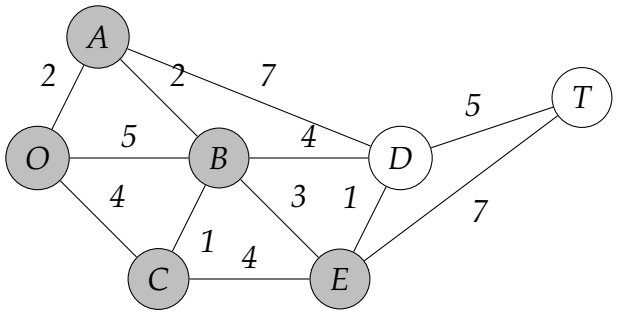
Iteration 5: *solved nodes*

- *O (0, N/A)*
- *A (2, O)*
- *B (4, A)*
- *C (4, O)*
- *E (7, B)*

candidate nodes

- *D (8, B)*

- *T (14, E)*

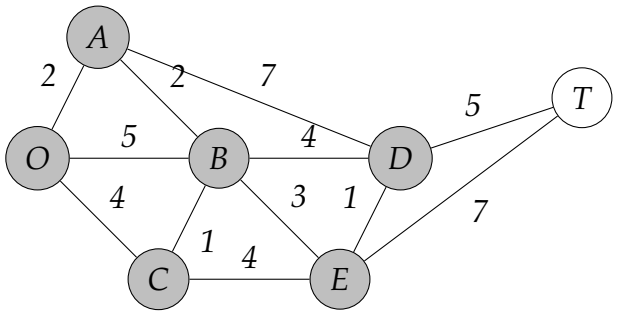


Iteration 6: solved nodes

- *O (0, N/A)*
- *A (2, O)*
- *B (4, A)*
- *C (4, O)*
- *E (7, B)*
- *D (8, B)*

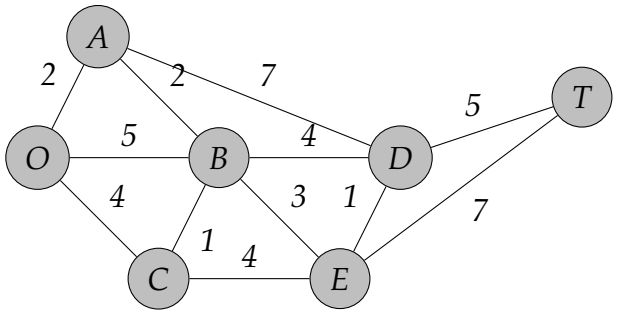
candidate nodes

- *T (14, E) → (13, D)*



Iteration 7: solved nodes

- *O (0, N/A)*
- *A (2, O)*
- *B (4, A)*
- *C (4, O)*
- *E (7, B)*
- *D (8, B)*
- *T (13, D)*



Terminate the algorithm as the target node is solved.

We remark that when there is negative cost on the edges/arcs, Dijkstra's algorithm may terminate prematurely and fail to give the correct answer. Thus we need another algorithm to handle the more general case.

7.3.2 Bellman-Ford Algorithm

The algorithm can be described as follows.

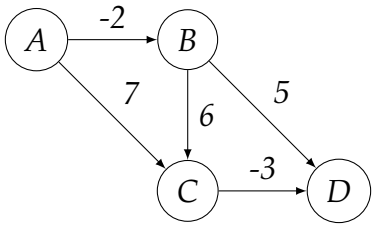
(i) Mark the cost of s as 0, and those of other nodes as $+\infty$.

(ii) Repeat the following step for $|V| - 1$ times.

- For each $(i, j) \in A$, if the current total cost of j is greater than the sum of the current total cost of i plus the cost on the arc (i, j),
  - update the cost of j to be the sum and set its predecessor to be i.

The cost we calculate in each step k is the minimum cost of node of a path from s to i connected with at most k arcs.

Example 7.10. *We want to find a cheapest path from node A to node D in the following directed graph.*



Iteration 0:

Node	*Cost*	*Predecessor*
A	0	N/A
B	$+\infty$	N/A
C	$+\infty$	N/A
D	$+\infty$	N/A

Iteration 1:

Node	*Cost*	*Predecessor*
A	0	N/A
B	-2	A
C	7	A
D	$+\infty$	N/A

Iteration 2:

Node	*Cost*	*Predecessor*
A	0	N/A
B	-2	A
C	4	B
D	3	B

Iteration 3:

Node	*Cost*	*Predecessor*
A	0	N/A
B	-2	A
C	4	B
D	1	C

We see that the minimum cost from A to D is 1, and the path is A, B, C, D. Note that the Dijkstra's algorithm, if applied in this case, would falsely terminate at the second iteration with a total cost 3.

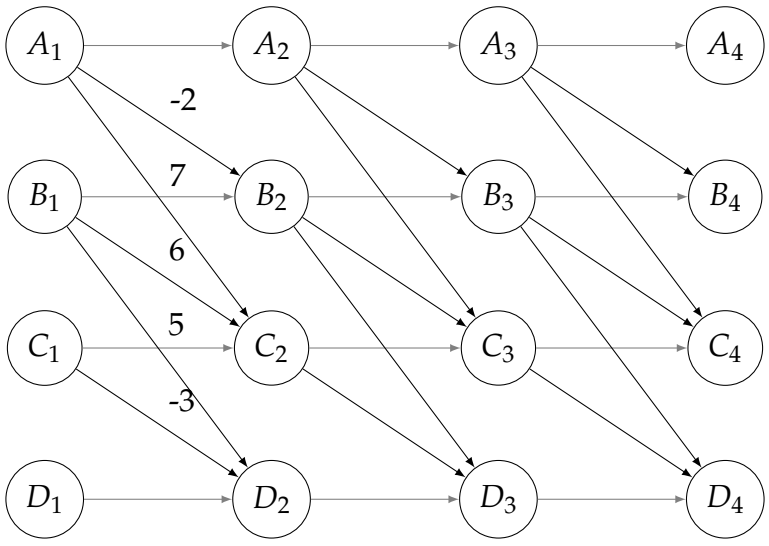
In general, the Bellman-Ford algorithm can be applied to any (directed) graph without a negative-cost cycle. We see below an alternative explanation why we need to execute the calculation step for $|V| - 1$ times.

7.4 Dynamic Programming

The shortest path problem can be reformulated on a *directed acyclic graph* (DAG). Let $\tilde{G} = (\tilde{V}, \tilde{A})$ such that

- $\tilde{V}$ consists of v_k for each $v \in V$ and $k = 1, \ldots, K := |V|$;
- the arcs $\tilde{A}$ consists of
  - (v_{k-1}, v_k) with cost 0 for each $v \in V$ and $k = 2, \ldots, K$,
  - and (v_{k-1}, u_k) with the cost c_{uv} for each $(u, v) \in A$ and $k = 2, \ldots, K$.

For example, for Example 7.10, the DAG can be plotted as follows.



We now want to find a path with minimum cost from s_1 to t_K in $\tilde{G}$. To do this, notice that the vertices are now grouped into *stages* $k = 1, \ldots, K$, such that arcs emanating from stage k vertices terminate in stage $k+1$. For each vertex v_k in stage $k = 2, \ldots, K$, we can write

$$c(v_k) = \min\{c_{uv} + c(u_{k-1}) : u \in V\}.$$

This is known as the *Bellman equation of dynamic programming* and corresponds to exactly what we did in the Bellman-Ford algorithm.

Dynamic programming (DP) is a very useful algorithmic framework for many optimization problems, where

- the decisions are made in different *stages* $t = 1, \ldots, T$;
- in each stage there are multiple *states* $v \in V_t$ that contain all information impacting our decision in that stage;
- once a decision is made, the transition into the next stage is known $(u, v) \in A_t$;
- the cumulative cost f_t starting from stage t can be described by recursions

$$f_t(u) = \min_{v \in V_{t+1}} \{c_{uv} + f_{t+1}(v)\}.$$

The Bellman equation allows us to solve many discrete optimization problems recursively.

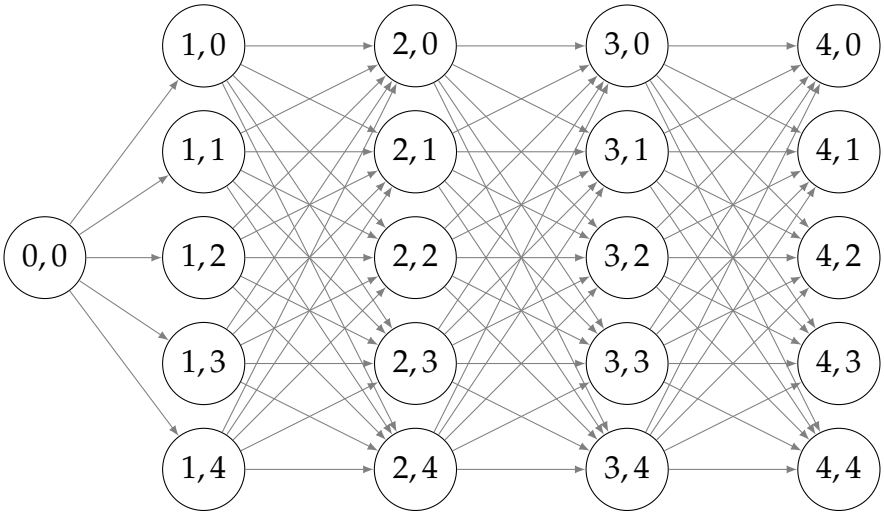
Example 7.11. *A company knows that the demand for its product during each of the next four months will be as follows:*

Month	*Demand*
1	*1*
2	*3*
3	*2*
4	*4*

At the beginning of each month, the company must determine how many units should be produced during the current month. During a month in which any units are produced, a setup cost of \$3 is incurred. In addition, there is a variable cost of \$1 for every unit produced. At the end of each month, a holding cost of \$0.5 per unit on hand is incurred. Capacity limitations allow a maximum of 5 units to be produced during each month. The size of the companys warehouse restricts the ending inventory for each month to 4 units at most. The company wants to determine a production schedule that will meet all demands on time and will minimize the sum of production and holding costs during the four months. Assume that 0 units are on hand at the beginning of the first month.

Here, we can index the stages by 1, 2, 3, 4 (one for each month), and the state by 0, 1, 2, 3, 4, which represents the inventory at the end of the month. The transition is given by the fact that the inventory at the end of month t equals the inventory at the end of month $t-1$ plus

the production in month t minus the demand in month t. We may use a DAG to represent the problem, where every vertex is denoted by a stage-state pair, and an additional vertex $(0,0)$ *is added to denote the initial inventory at hand.*



To solve the problem through DP recursion (Bellman equation), note that

$$f(4,0) = \cdots = f(4,4) = 0,$$

and

$$f(3,1) = \min_{p=3,4,5}\{f(4,1+p-4) + 3 \cdot \mathbb{1}(p>0) + p + 0.5\},$$

because the demand in month 4 is 4. Similarly, this gives us

$$\begin{aligned}
f(3,0) &= \min\{3+4, 3+5\} = 7, \text{ with } p = 4,\\
f(3,1) &= \min\{3+3+0.5, 3+4+0.5, 3+5+0.5\} = 6.5, \text{ with } p = 3,\\
f(3,2) &= \min\{3+2+1, \ldots, 3+5+1\} = 6, \text{ with } p = 2,\\
f(3,3) &= \min\{3+1+1.5, \ldots, 3+5+1.5\} = 5.5, \text{ with } p = 1,\\
f(3,4) &= \min\{2, 3+1+2, \ldots, 3+4+2\} = 2, \text{ with } p = 0.
\end{aligned}$$

We can now calculate $f(2,i)$ *for* $i = 0,1,\ldots,4$ *by*

$$f(2,i) = \min_{p\le 5}\{f(3,i+p-2) + 3 \cdot \mathbb{1}(p>0) + p + 0.5i : 0 \le i+p-2 \le 4\}.$$

Plugging in the values, we get

$$f(2,0) = \min\{7+3+2, 6.5+3+3, \ldots, 5.5+3+5\} = 12, \text{ with } p = 2,$$
$$f(2,1) = \min\{7+3+1+0.5, \ldots, 5.5+3+4+0.5, 2+3+5+0.5\} = 10.5, \text{ with } p = 5,$$
$$f(2,2) = \min\{7+1, \ldots, 5.5+3+3+1, 2+3+4+1\} = 8, \text{ with } p = 0,$$
$$f(2,3) = \min\{6.5+1.5, 6+3+1+1.5, \ldots, 2+3+3+1.5\} = 8, \text{ with } p = 0,$$
$$f(2,4) = \min\{6+2, 5.5+3+1+2, 2+3+2+2\} = 8, \text{ with } p = 0.$$

Then, we can now calculate $f(1,i)$ for $i = 0, 1, \ldots, 4$ by

$$f(1,i) = \min_{p \le 5}\{f(2, i+p-3) + 3 \cdot \mathbb{1}(p > 0) + p + 0.5i : 0 \le i+p-3 \le 4\}.$$

Plugging in the values, we get

$$f(1,0) = \min\{18, 17.5, 16\} = 16, \text{ with } p = 5,$$
$$f(1,1) = \min\{17.5, 17, 15.5, 16.5\} = 16.5, \text{ with } p = 4,$$
$$f(1,2) = \min\{17, 16.5, 15, 16, 17\} = 15, \text{ with } p = 3,$$
$$f(1,3) = \min\{13.5, 16, 14.5, 15.5, 16.5\} = 13.5, \text{ with } p = 0,$$
$$f(1,4) = \min\{12.5, 14, 15, 16\} = 12.5, \text{ with } p = 0,$$

We can finally calculate $f(0,0)$ by

$$f(0,0) = \min_{p \le 5}\{f(1, p-1) + 3 \cdot \mathbb{1}(p > 0) + p : 0 \le p-1 \le 4\}.$$

This gives us

$$f(0,0) = \min\{20, 21.5, 21, 20.5, 20.5\} = 20, \text{ with } p = 1.$$

Retrieving the solutions, we should produce 1 unit in month 1, 5 units in month 2, 0 units in month 3, and 4 units in month 4. The total cost is \$20.

Sometimes DP can be applied to problems that do not have a clear stage structure. We illustrate this by the following example of a *knapsack problem* that has been introduced in MILO model and the cutting stock problem in the column generation step.

Example 7.12. *Suppose we would like to fill a knapsack with capacity of* 10 *kilograms. The items of each type are listed below with their values.*

Type	*Weight (kg)*	*Value ($)*
1	*4*	*11*
2	*3*	*7*
3	*5*	*12*

The goal is to maximize the total value of the items in the knapsack.

Here, we can set the stages to be $t = 1, 2, 3$*, representing the items of types* $t, t+1, \ldots, 3$ *to be filled. Then the state in each stage is denoted by* x*, the remaining capacity in the knapsack. Since we are maximizing the total value, we define our value function in stage* t *as* $f_t(x)$*, which is the maximum value that can be filled in the knapsack with items* $t, t+1, \ldots, 3$*. We have the recursion*

$$f_t(x) = \max_y \{v_t \cdot y + f_{t+1}(x - w_t \cdot y) : x - w_t \cdot y \geq 0\},$$

where v_t, w_t *are the value and the weight of item* t*.*

In stage 3, clearly we have

$$\begin{aligned} &f_3(10) = 24, \text{ with } y_3^* = 2, \\ &f_3(x) = 12, \text{ for } 5 \leq x \leq 9, \text{ with } y_3^* = 1, \\ &f_3(x) = 0, \text{ for } 0 \leq x \leq 4, \text{ with } y_3^* = 0. \end{aligned}$$

In stage 2, $f_2(x) = \max\{7y + f_3(x - 3y) : x - 3y \geq 0\}$*, which gives us*

$$\begin{aligned} f_2(10) &= \max\{24, 19, 14, 21\} = 24, \text{ with } y_2^* = 0, \\ f_2(9) &= \max\{12, 19, 14, 21\} = 21, \text{ with } y_2^* = 3, \\ f_2(8) &= \max\{12, 19, 14\} = 19, \text{ with } y_2^* = 1, \\ f_2(7) &= \max\{12, 7, 14\} = 14, \text{ with } y_2^* = 2, \\ f_2(6) &= \max\{12, 7, 14\} = 14, \text{ with } y_2^* = 2, \\ f_2(5) &= \max\{12, 7\} = 12, \text{ with } y_2^* = 0, \\ f_2(4) &= \max\{0, 7\} = 7, \text{ with } y_2^* = 1, \\ f_2(3) &= \max\{0, 7\} = 7, \text{ with } y_2^* = 1, \\ f_2(x) &= 0, \text{ for } x = 2, 1, 0, \text{ with } y_2^* = 0. \end{aligned}$$

Finally for stage 1, we only need to calculate

$$f_1(10) = \max\{24, 25, 22\} = 25, \text{ with } y_1^* = 1.$$

The optimal knapsack consists of 1 type 1 item and then 2 type 2 items, with the total value being 25.