VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY
INSTITUTE OF MATHEMATICS

Le Xuan Thanh

INTEGER PROGRAMMING: AN INTRODUCTION

Hanoi, 2020

VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY
INSTITUTE OF MATHEMATICS

Le Xuan Thanh

INTEGER PROGRAMMING: AN INTRODUCTION

Text book for master program

Hanoi, 2020

Contents

Chapter 1

Linear programming revisited

1.1 First concepts in optimization

Many problems from real life can be formulated in the following form

$$\text{minimize} \quad f(\mathbf{x}) \tag{1.1}$$
$$\text{subject to} \quad \mathbf{x} \in C, \tag{1.2}$$

in which $C \subset \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$. The problem (1.1)-(1.2) is called an *optimization problem*, in which the function f is called *objective function*, $\mathbf{x}$ consists of *decision variables*, and C is called *feasible set* or *constraint set*. If C is empty, we say that the problem is *infeasible*. Otherwise, we say that the problem is *feasible*.

Any $\mathbf{x} \in C$ is called a *feasible solution* of the problem. A feasible solution $\mathbf{x}^* \in C$ is called an *optimal solution* to problem (1.1)-(1.2) if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in C$. If $\mathbf{x}^*$ is an optimal solution, the value of $f(\mathbf{x}^*)$ is called *optimal value*. We say that $\mathbf{x}^*$ is a *local optimal solution* if there exist $\varepsilon > 0$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in C \cap B(\mathbf{x}^*, \varepsilon)$. Here, $B(\mathbf{x}^*, \varepsilon) \subset \mathbb{R}^n$ is an open ball of center $\mathbf{x}^*$ and radius ε, i.e.,

$$B(\mathbf{x}^*, \varepsilon) = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x} - \mathbf{x}^*\| < \varepsilon\},$$

in which $\|\cdot\|$ denotes the usual Euclidean norm in $\mathbb{R}^n$. Roughly speaking, a local optimal solution is optimal in a neighborhood of itself.

Often the feasible set C is given as the solution set of a system of the following form

$$\begin{cases} g_i(\mathbf{x}) \leq 0 & (i = 1, \ldots, m_1) \\ h_j(\mathbf{x}) = 0 & (j = 1, \ldots, m_2) \end{cases}$$

in which $g_1, \ldots, g_{m_1}$, $h_1, \ldots, h_{m_2}$ are real-valued functions defined on $\mathbb{R}^n$, so-called *constrained functions*. When the objective function and the constrained functions are affine, we have a *linear program* (abbreviated LP), so-called a *linear programming problem*.

1.2 Formulating linear programs

1.2.1 Some examples

1. A transportation problem

A company produces a single type of product. The company has m plants and n warehouses. After being produced at the plants, the company products are shipped to the warehouses for storage. Each plant has a given level of supply, and each warehouse has a given level of demand. We are also given the transportation costs from each plant to each warehouse, and these costs are assumed to be linear. More explicitly, the assumptions are:

- The total supply of the product from plant i is s_i, where $i = 1, \ldots, m$.
- The total demand for the product at warehouse j is d_j, where $j = 1, \ldots, n$.
- The cost of sending one unit of the product from plant i to warehouse j is equal to c_{ij}, where $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The total cost of a shipment is linear in the size of the shipment.

The problem of interest is to determine an optimal transportation scheme between the plants and the warehouses, subject to the specified supply and demand constraints. Graphically, a transportation problem is often visualized as a network with m source nodes, n sink nodes, and a set of $m \times n$ "directed arcs". Such a visualization with $m = 2$ and $n = 3$ is depicted in Figure 1.1.

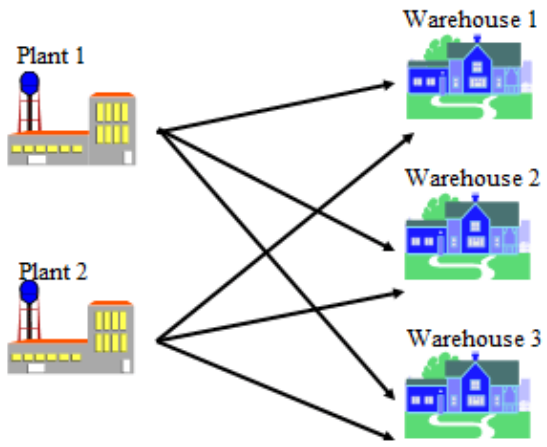


Figure 1.1: Visualization of a transportation problem.

We now formulate this problem in form of a linear program.

Decision variables

A transportation scheme is a complete specification of how many units of the product should be shipped from each plant to each warehouse. Therefore, the decision variables are

$$x_{ij} = \text{ the size of the shipment from plant } i \text{ to warehouse } j,$$

where $i = 1, \ldots, m$ and $j = 1, \ldots, n$. This is a set of mn variables.

Objective function

For each plant i and each warehouse j, the transportation cost per product unit is c_{ij}, and the size of the shipment is x_{ij}. Since we assume that the cost function is linear in the size of the shipment, the total cost of this shipment is given by $c_{ij}x_{ij}$. Summing over all plants and warehouses yields the overall transportation cost from all plant-warehouse combinations. That is, our objective function is

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

Constraints

Let us consider an arbitrary plant $i \in \{1, \ldots, m\}$. The total outgoing shipment from this plant to all warehouses is

$$x_{i1} + \ldots + x_{in} = \sum_{j=1}^{n} x_{ij}.$$

Since the total supply from plant i is s_i, the total outgoing shipment cannot exceed s_i. Therefore, we must require

$$\sum_{j=1}^{n} x_{ij} \leq s_i.$$

Now, let us consider an arbitrary warehouse $j \in \{1, \ldots, n\}$. The total incoming shipment at this warehouse from all plants is

$$x_{1j} + \ldots + x_{mj} = \sum_{i=1}^{m} x_{ij}.$$

Since the total demand at warehouse j is d_j, the total incoming shipment should not be less than b_j. That is, we must require

$$\sum_{i=1}^{m} x_{ij} \geq d_j.$$

Furthermore, as physical shipments, the x_{ij}'s should be nonnegative.

LP formulation

We come up with the following linear programming formulation of the transportation problem.

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

2. A production problem

A company produces n different goods from m different raw materials. To produce an unit of each good we need a certain amount of each material. Each raw material has a given level of supply. We are also given the revenue from selling each good, and these revenues are assumed to be linear. More explicitly, the assumptions are:

- The available amount of raw material i is b_i, where $i = 1,\dots,m$.
- The amount of material i to produce an unit of good j is a_{ij}, where $i = 1,\dots,m$ and $j = 1,\dots,n$.
- The revenue of selling one unit of good j is equal to c_j, where $j = 1,\dots,n$. The revenue of selling each good is linear in the amount of the good.

The problem of interest is to determine an optimal production plan subject to the specified constraints on used materials. In the following, we formulate this problem in form of a linear program.

Decision variables

A production plan is a complete specification of how many units of each good should be produced. Therefore, the decision variables are

$$x_j = \text{ the amount of good } j \text{ should be produced,}$$

where $j = 1,\dots,n$. This is a set of n variables.

Objective function

For each unit of good j, the revenue of selling is c_j. Since we assume that the revenue is linear in the amount of the good, the total revenue of selling good j is $c_j x_j$. Summing over all goods yields the total revenue of selling produced goods. Therefore, our objective function is

$$\max \quad \sum_{j=1}^{n} c_j x_j.$$

Constraints

Let us consider an arbitrary material $i \in \{1, \ldots, m\}$. The total amount of this material used to produce the goods is

$$a_{i1}x_1 + \ldots + a_{in}x_n = \sum_{j=1}^{n} a_{ij}x_j.$$

Since the available amount of material i is b_i, the total used amount of this material cannot exceed b_i. Therefore, we must require

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i.$$

Of course, as physical goods, the x_j's should be nonnegative.

LP formulation

We come up with the following linear programming formulation of the production problem.

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

1.2.2 General, standard, and canonical formulations

Generally, a linear program is a problem of optimizing a linear function subject to linear constraints. The *general formulation* of a linear program can be described explicitly as follows.

$$\begin{aligned} \max(\min) \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n & \\ \text{s.t.} \quad & a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq b_i & (i = 1, \ldots, m_1) \\ & a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \geq b_i & (i = m_1 + 1, \ldots, m_2) \\ & a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = b_i & (i = m_2 + 1, \ldots, m) \\ & x_j \leq 0 & (j = 1, \ldots, n_1) \\ & x_j \geq 0 & (j = n_1 + 1, \ldots, n_2) \\ & x_j \in \mathbb{R} & (j = n_2 + 1, \ldots, n) \end{aligned}$$

in which $x_j (j = 1, \ldots, n)$ are real-valued variables, $b_i, c_j, a_{ij} (i = 1, \ldots, m, j = 1, \ldots, m)$ are real-valued parameters.

A famous and efficient algorithm to solve linear programs is simplex method. As the starting point of this algorithm, a linear program is required to be in standard form. The *standard formulation* of a linear program can be described explicitly as follows.

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\ & \ldots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \\ & x_1, x_2, \ldots, x_n \geq 0 \end{aligned}$$

We have the following remarks on the standard formulation of a linear program.

- The objective is to *maximize* a linear function.
- All constraints are of the *less-than-or-equal* form.
- All variables are *nonnegative*.

The standard formulation of a linear program can be also written in matrix form as follows.

$$\begin{aligned} \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

in which $\mathbf{x} = (x_1, \ldots, x_n)^t$ is the column vector of decision variables, $\mathbf{c} = (c_1, \ldots, c_n)^t$, $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, $\mathbf{b} = (b_1, \ldots, b_m)^t$, $\mathbf{0} = (0, \ldots, 0)^t \in \mathbb{R}^n$. The inequalities in the formulation are component-wise.

Another form of linear programs that can be also used in the paradigm of simplex method is canonical form. The *canonical formulation* of a linear program is the following.

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ & \ldots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \\ & x_1, x_2, \ldots, x_n \geq 0 \end{aligned}$$

The canonical formulation is the same as the standard formulation, except that all constraints in the canonical formulation are equalities. In the matrix form, the canonical formulation can be written as follows.

$$\begin{aligned} \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \end{aligned}$$

$$\mathbf{x} \geq \mathbf{0}$$

We can easily convert a linear program from one formulation to another by using the following techniques.

- A minimization objective can be converted to a maximization objective, and vice versa, because of the following remark

$$f(\mathbf{x}^*) = \min\{f(\mathbf{x}) \mid \mathbf{x} \in C\} \quad \Leftrightarrow \quad -f(\mathbf{x}^*) = \max\{-f(\mathbf{x}) \mid \mathbf{x} \in C\}.$$

- An inequality constraint of the greater-than-or-equal form

$$a_1x_1 + \ldots + a_nx_n \geq b$$

can be converted to an inequality constraint of the less-than-or-equal form by multiplying both sides by -1:

$$-a_1x_1 - \ldots - a_nx_n \leq -b.$$

- An inequality constraint

$$a_1x_1 + \ldots + a_nx_n \leq b$$

can be converted to an equality constraint by adding a nonnegative variable w, which we call a *slack variable*:

$$a_1x_1 + \ldots + a_nx_n + w = b.$$

- An equality constraint

$$a_1x_1 + \ldots + a_nx_n = b$$

can be converted to inequality form by introducing two inequality constraints

$$a_1x_1 + \ldots + a_nx_n \leq b$$
$$a_1x_1 + \ldots + a_nx_n \geq b$$

- A variable x_j without any restriction on its sign can be replaced by the difference of two nonnegative variables:

$$x_j = x_j^+ - x_j^- \quad \text{with } x_j^+, x_j^- \geq 0.$$

1.3 Structures of feasible sets

In a nutshell, the feasible set of a linear program is a convex polyhedron. To have a deeper discovery on the structures of the feasible sets of linear programs, we need some concepts and results in convex analysis and theory of convex polytopes.

1.3.1 Some concepts and results in convex analysis

Definition 1.1. (Affine set).
A set $A \subset \mathbb{R}^n$ is affine if for any $\mathbf{x}^1, \mathbf{x}^2 \in A$ and $\theta \in \mathbb{R}$ we have $\theta\mathbf{x}^1 + (1-\theta)\mathbf{x}^2 \in A$.

One can easily verify that the following sets are affine.

- Let $\mathbf{x}^1, \mathbf{x}^2$ be two distinct points in $\mathbb{R}^n$. The line passing through $\mathbf{x}^1$ and $\mathbf{x}^2$ is defined by
$$\{\mathbf{y} \in \mathbb{R}^n \mid \mathbf{y} = \theta\mathbf{x}^1 + (1-\theta)\mathbf{x}^2 \text{ for some } \theta \in \mathbb{R}\}.$$
Figure 1.2 illustrates such a line in $\mathbb{R}^2$.

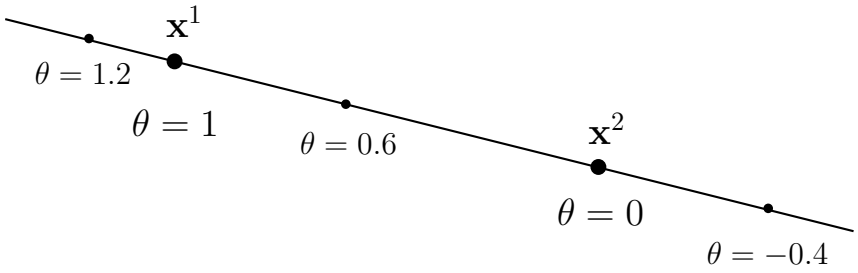


Figure 1.2: The line passes through two points $\mathbf{x}^1$ and $\mathbf{x}^2$ in $\mathbb{R}^2$.

- A hyperplane is a set of the form
$$H = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^t\mathbf{x} = b\}$$
where $\mathbf{a} \in \mathbb{R}^n \backslash \{\mathbf{0}\}$ and $b \in \mathbb{R}$.

- The solution set of a system of linear equations
$$C = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b}\}$$
where $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$.

In other words, a set $A \subset \mathbb{R}^n$ is affine if the line passing through any two distinct points of A is also in A.

Definition 1.2. (Affine combination).
An affine combination of the points $\mathbf{x}^1, \ldots, \mathbf{x}^k \in \mathbb{R}^n$ is a point of the form
$$\theta_1\mathbf{x}^1 + \ldots + \theta_k\mathbf{x}^k$$
where $\theta_1, \ldots, \theta_k \in \mathbb{R}$ satisfying $\theta_1 + \ldots + \theta_k = 1$.

Definition 1.3. (Affine hull).
The affine hull of a set $A \subset \mathbb{R}^n$ is the set of all affine combinations of points in A:
$$\text{aff}(A) = \{\theta_1\mathbf{x}^1 + \ldots + \theta_k\mathbf{x}^k \mid \mathbf{x}^1, \ldots, \mathbf{x}^k \in A, \theta_1 + \ldots + \theta_k = 1\}.$$

For example, any point on the line passing through two distinct points $\mathbf{x}^1, \mathbf{x}^2 \in \mathbb{R}^n$ is an affine combination of these two points.

Proposition 1.4. *The affine hull* $\text{aff}(A)$ *of a set* $A \subset \mathbb{R}^n$ *is the smallest affine set containing* A.

Definition 1.5. (Subspace associated with an affine set).
Let $A \subset \mathbb{R}^n$ *be an affine set, and* $\mathbf{x}^0 \in A$. *The subspace associated with* A *is the set*

$$V_A := A - \mathbf{x}^0 = \{\mathbf{v} \in \mathbb{R}^n \mid \mathbf{v} = \mathbf{x} - \mathbf{x}^0 \text{ for some } \mathbf{x} \in A\}.$$

As a remark, the subspace V_A associated with the affine set A is defined independently of the choice of $\mathbf{x}^0$.

Proposition 1.6. *Let* $A \subset \mathbb{R}^n$ *be an affine set, and* $\mathbf{x}^0 \in A$. *The subspace associated with* A *is indeed a subspace of the vector space* $\mathbb{R}^n$.

Definition 1.7. (Affine dimension).
The affine dimension of a set $A \subset \mathbb{R}^n$ *is the dimension of the subspace associated with* $\text{aff}(A)$.

For examples, the set consists of two distinct vectors in $\mathbb{R}^n$ has affine dimension 1, a hyperplane in the three-dimensional space has affine dimension 2, the solution set of a linear system $A\mathbf{x} = \mathbf{0}$ has affine dimension $n - \text{rank}(A)$, where n is the number of columns of matrix A.

Definition 1.8. (Relative interior).
The relative interior of a set $A \subset \mathbb{R}^n$ *is*

$$\text{relint}(A) = \{\mathbf{x} \in A \mid \exists r > 0 : B(\mathbf{x}, r) \cap \text{aff}(A) \subset A\},$$

where $B(\mathbf{x}, r) = \{\mathbf{y} \in \mathbb{R}^n \mid \|\mathbf{y} - \mathbf{x}\| < r\}$.

In other words, a relative interior point of a set $A \subset \mathbb{R}^n$ is an interior point with respect to the affine hull of A, not necessary an interior point of A in the whole space $\mathbb{R}^n$. As a remark, the relative interior of A is defined independently of the norm defining $B(\mathbf{x}, r)$, i.e., in this definition all norms define the same relative interior of A. As an example, if we have

$$A = \{\mathbf{x} \mid \mathbf{x} = \theta\mathbf{x}^1 + (1-\theta)\mathbf{x}^2, \theta \in [0, 1]\},$$

then

$$\text{relint}(A) = \{\mathbf{x} \mid \mathbf{x} = \theta\mathbf{x}^1 + (1-\theta)\mathbf{x}^2, \theta \in (0, 1)\}.$$

Figure 1.3 gives an illustration why in this example $\mathbf{x}^1$ and $\mathbf{x}^2$ are not in the relative interior of A.

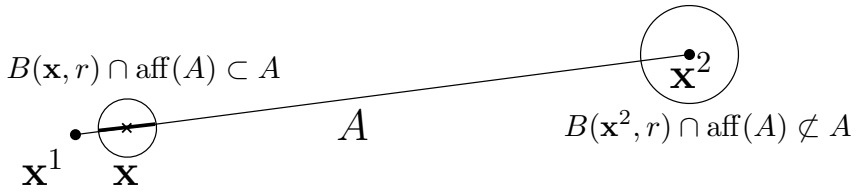


Figure 1.3: Relative interior of a line segment.

Definition 1.9. (Convex set).
A set $C \subset \mathbb{R}^n$ is convex if for any $\mathbf{x}^1, \mathbf{x}^2 \in C$ and $\theta \in [0,1]$ we have $\theta\mathbf{x}^1 + (1-\theta)\mathbf{x}^2 \in C$.

For example, the line segment between two distinct points $\mathbf{x}^1, \mathbf{x}^2 \in \mathbb{R}^n$, i.e. the set

$$\{\mathbf{y} \in \mathbb{R}^n \mid \mathbf{y} = \theta\mathbf{x}^1 + (1-\theta)\mathbf{x}^2 \text{ for some } \theta \in [0,1]\},$$

is convex. Figure 1.4 illustrates such a line in $\mathbb{R}^2$.

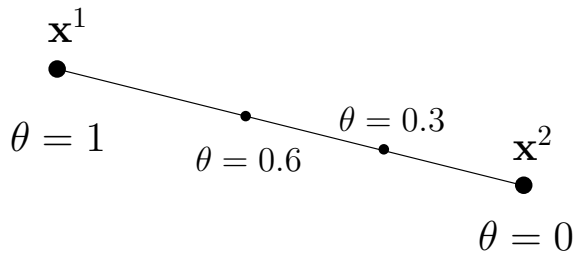


Figure 1.4: The line segment between two points $\mathbf{x}^1$ and $\mathbf{x}^2$ in $\mathbb{R}^2$.

In other words, a set $A \subset \mathbb{R}^n$ is convex if the line segment between two distinct points of A is also in A. In that manner, the sets in Figure 1.5 are convex, while the sets in Figures 1.6 are not.

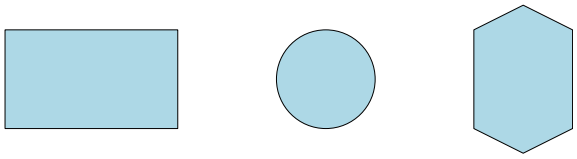


Figure 1.5: Some convex sets in $\mathbb{R}^2$.

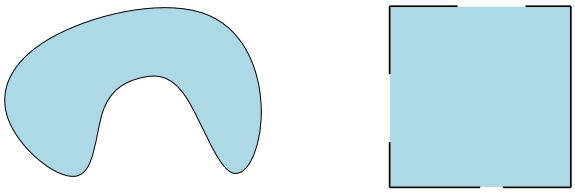


Figure 1.6: Some nonconvex sets in $\mathbb{R}^2$.

Definition 1.10. (Convex combination).
A convex combination of the points $\mathbf{x}^1, \ldots, \mathbf{x}^k \in \mathbb{R}^n$ is a point of the form

$$\theta_1\mathbf{x}^1 + \ldots + \theta_k\mathbf{x}^k$$

where $\theta_1, \ldots, \theta_k \in [0,1]$ satisfying $\theta_1 + \ldots + \theta_k = 1$

Definition 1.11. (Convex hull).
The convex hull of a set $C \subset \mathbb{R}^n$ is the set of all convex combinations of points in C:

$$\text{conv}(C) = \{\theta_1 \mathbf{x}^1 + \ldots + \theta_k \mathbf{x}^k \mid \mathbf{x}^1, \ldots, \mathbf{x}^k \in C, \theta_1, \ldots, \theta_k \geq 0, \theta_1 + \ldots + \theta_k = 1\}.$$

For example, any point in the line segment between two distinct points $\mathbf{x}^1, \mathbf{x}^2 \in \mathbb{R}^n$ is a convex combination of these two points.

Proposition 1.12. *The convex hull* $\text{conv}(C)$ *of a set $C \subset \mathbb{R}^n$ is the smallest convex set containing C.*

In spirit of Proposition 1.12, Figure 1.7 illustrates the convex hulls of two sets in $\mathbb{R}^2$.

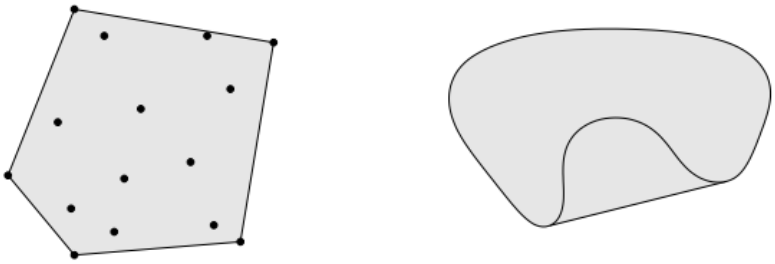


Figure 1.7: Convex hulls of two sets in $\mathbb{R}^2$.

Definition 1.13. (Cone).
A set $C \subset \mathbb{R}^n$ is called a cone if for any $\mathbf{x} \in C$ and $\theta \geq 0$ we have $\theta \mathbf{x} \in C$.

For examples, in $\mathbb{R}^n$ the following sets are cones.

- A line passing through origin.
- A ray based at origin (i.e., the set $\{\theta \mathbf{v} \mid \theta \geq 0\}$).
- The union of different rays based at origin.

Figure 1.8 illustrates these sets in $\mathbb{R}^2$.

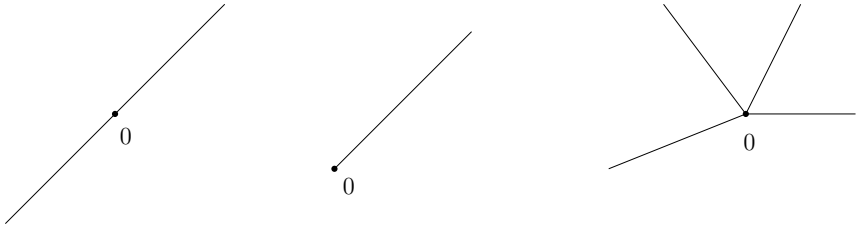


Figure 1.8: Some cones in $\mathbb{R}^2$.

Definition 1.14. (Convex cone).
A set $C \subset \mathbb{R}^n$ is a convex cone if for any $\mathbf{x}^1, \mathbf{x}^2 \in C$ and $\theta_1, \theta_2 \geq 0$ we have $\theta_1 \mathbf{x}^1 + \theta_2 \mathbf{x}^2 \in C$.

Proposition 1.15. *Any convex cone in $\mathbb{R}^n$ is convex and conic.*

Figure 1.9 illustrates a convex cone in $\mathbb{R}^2$.

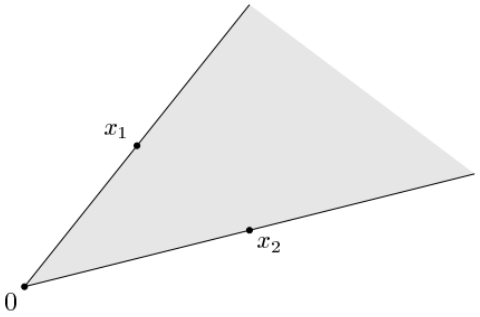


Figure 1.9: A convex cone in $\mathbb{R}^2$.

Definition 1.16. (Conic combination).
A conic combination of the points $\mathbf{x}^1, \ldots, \mathbf{x}^k \in \mathbb{R}^n$ is a point of the form

$$\theta_1 \mathbf{x}^1 + \ldots + \theta_k \mathbf{x}^k$$

where $\theta_1, \ldots, \theta_k \geq 0$.

Definition 1.17. (Conic hull).
The conic hull of a set $C \subset \mathbb{R}^n$ is the set of all conic combinations of points in C:

$$\text{cone}(C) = \{\theta_1 \mathbf{x}^1 + \ldots + \theta_k \mathbf{x}^k \mid \mathbf{x}^1, \ldots, \mathbf{x}^k \in C, \theta_1, \ldots, \theta_k \geq 0\}.$$

Proposition 1.18. *The conic hull* $\text{cone}(C)$ *of a set $C \subset \mathbb{R}^n$ is the smallest convex cone containing C.*

In spirit of Proposition 1.18, Figure 1.10 illustrates the conic hulls of two sets in $\mathbb{R}^2$.

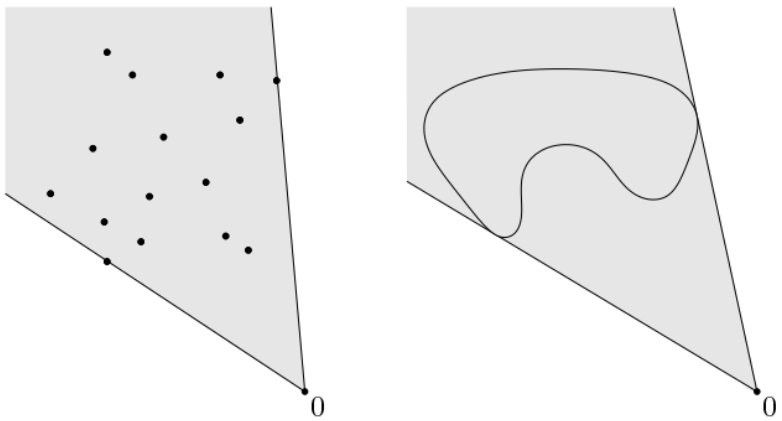


Figure 1.10: Conic hulls of two sets in $\mathbb{R}^2$.

Definition 1.19. (Polyhedral cone).
A polyhedral cone is the conic hull of a finite set of points in some $\mathbb{R}^n$*:*

$$\text{cone}(\mathbf{x}^1, \ldots, \mathbf{x}^k) = \{\theta_1 \mathbf{x}^1 + \ldots + \theta_k \mathbf{x}^k \mid \theta_1, \ldots, \theta_k \geq 0\}.$$

Figure 1.11 (left) illustrates a polyhedral cone induced by three points $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3$ in $\mathbb{R}^2$. Figure 1.11 (right) illustrates a polyhedral cone induced by three vectors $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3$ in $\mathbb{R}^3$, in which these vectors are not in the same hyperplane.

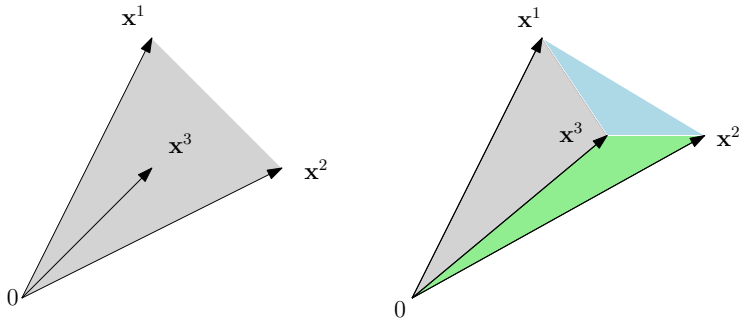


Figure 1.11: Two polyhedral cones in $\mathbb{R}^2$ and $\mathbb{R}^3$.

The ice-cream cone, so-called Lorentz cone, defined by

$$\left\{(x_1, x_2, x_3) \in \mathbb{R}^3 \mid 0 \leq x_3 \leq \sqrt{x_1^2 + x_2^2}\right\},$$

is a cone in $\mathbb{R}^3$, but is not a polyhedral cone. Figure 1.12 illustrates this cone.

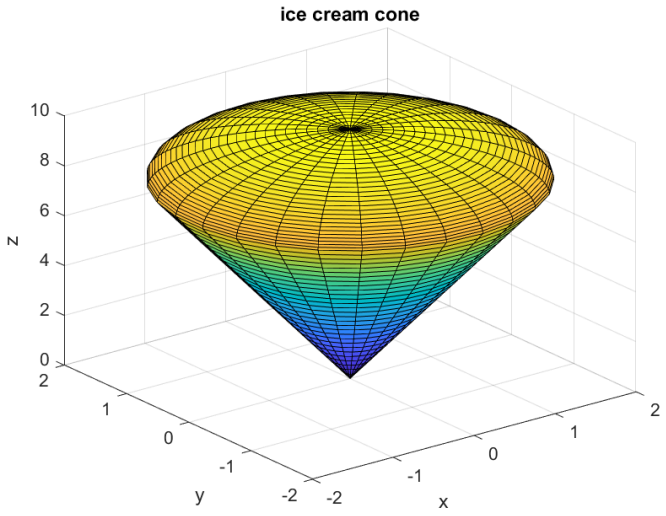


Figure 1.12: Lorentz cone is not a polyhedral cone.

Definition 1.20. (Recession cone).
The recession cone of a nonempty set $A \subset \mathbb{R}^n$ is defined by

$$\text{recc}(A) = \{\mathbf{v} \in \mathbb{R}^n \mid \mathbf{x} + \lambda\mathbf{v} \in A \ \forall \mathbf{x} \in A, \lambda \geq 0\}.$$

The following proposition explains why the term 'cone' appears in the name of the above concept.

Proposition 1.21. *Let $A \subset \mathbb{R}^n$ be a nonempty set. Then* $\text{recc}(A)$ *is a cone.*

In other words, the recession cone of a set A is a cone containing all vectors such that A recedes in that direction. That is, the set extends outward in all the directions given by the recession cone.

Definition 1.22. (Minkowski sum).
The Minkowski sum of two sets $P, Q \subset \mathbb{R}^n$ is defined by

$$P + Q = \{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in P, \mathbf{y} \in Q\}.$$

Figure 1.13 illustrates the Minkowski sum of two objects in $\mathbb{R}^2$.

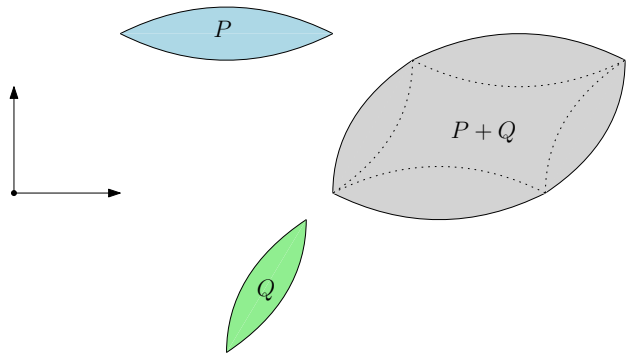


Figure 1.13: Minkowski sum of two objects in $\mathbb{R}^2$.

1.3.2 Algebraic and geometrical representations

Definition 1.23. ($\mathcal{V}$-polytope).
A $\mathcal{V}$-polytope $P \subset \mathbb{R}^n$ is a set of the form

$$P = \text{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^m),$$

in which $\mathbf{x}^1, \ldots, \mathbf{x}^m \in \mathbb{R}^n$.

In other words, a $\mathcal{V}$-polytope is the convex hull of a finite set of points some $\mathbb{R}^n$. Observe that a $\mathcal{V}$-polytope is always bounded in the sense that it does not contain any ray $\{\mathbf{x} + \lambda\mathbf{v} \mid \lambda \geq 0\}$ for any $\mathbf{v} \neq \mathbf{0}$. This definition of "bounded" has the advantages that it does not rely on a metric or a scalar product, and it is invariant under affine change of coordinates.

Definition 1.24. ($\mathcal{V}$-cone).
A $\mathcal{V}$-cone $C \subset \mathbb{R}^n$ is a set of the form

$$C = \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k),$$

in which $\mathbf{v}^1, \ldots, \mathbf{v}^k \in \mathbb{R}^n$.

It means that a $\mathcal{V}$-cone is the conic hull of a finite set of points in some $\mathbb{R}^n$, i.e., a polyhedral cone.

Definition 1.25. ($\mathcal{H}$-cone).
An $\mathcal{H}$-cone $C \subset \mathbb{R}^n$ is a set of the form

$$C = \{\mathbf{v} \in \mathbb{R}^n \mid A\mathbf{v} \leq \mathbf{0}\}$$

for some matrix $A \in \mathbb{R}^{m \times n}$.

It means that an $\mathcal{H}$-cone is a finite intersection of closed linear halfspaces in some $\mathbb{R}^n$.

Definition 1.26. ($\mathcal{V}$-polyhedron and $\mathcal{V}$-representation).
A $\mathcal{V}$-polyhedron $P \subset \mathbb{R}^n$ is a set of the form

$$P = \text{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^m) + \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k),$$

in which $\mathbf{x}^1, \ldots, \mathbf{x}^m, \mathbf{v}^1, \ldots, \mathbf{v}^k \in \mathbb{R}^n$. This Minkowski sum is called the $\mathcal{V}$-representation of P.

Roughly speaking, a $\mathcal{V}$-polyhedron is the Minkowski sum of a convex hull of a finite point set and the cone generated by a finite set of vectors. Figure 1.14 illustrates such a Minkowski sum. As a remark, it follows from the definition that a $\mathcal{V}$-polytope is a bounded $\mathcal{V}$-polyhedron.

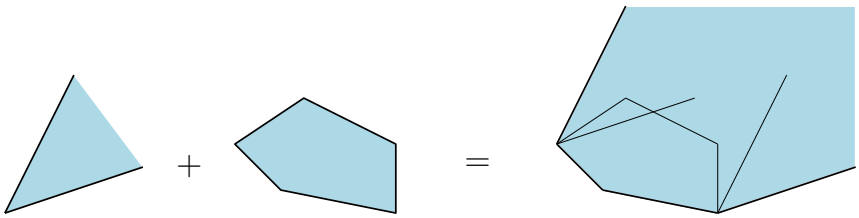


Figure 1.14: A $\mathcal{V}$-polyhedron is the Minkowski sum of a $\mathcal{V}$-polytope with a $\mathcal{V}$-cone.

Definition 1.27. ($\mathcal{H}$-polyhedron, $\mathcal{H}$-polytope, and $\mathcal{H}$-representation).
An $\mathcal{H}$-polyhedron $P \subset \mathbb{R}^n$ is a set of the form

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b} \text{ for some } A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m\}.$$

An $\mathcal{H}$-polytope is an $\mathcal{H}$-polyhedron that is bounded. The linear system $A\mathbf{x} \leq \mathbf{b}$ is called the $\mathcal{H}$-representation of P.

Roughly speaking, an $\mathcal{H}$-polyhedron is the intersection of a finite number of closed halfspaces. It follows immediately from the definition that the feasible set of a linear program is an $\mathcal{H}$-polyhedron.

We have seen that there are two ways to describe a polyhedron: the $\mathcal{V}$-representation gives us a geometrical description, while the $\mathcal{H}$-representation gives us an algebraic description. The central result in this subsection is Minkowski-Weyl theorem which states that these two representations are equivalent, i.e., any polyhedron that is described by one representation can also be described by the other.

Theorem 1.28. (Minkowski-Weyl theorem).
A subset $P \subseteq \mathbb{R}^n$ is a $\mathcal{V}$-polyhedron (i.e. the Minkowski sum of a convex hull of a finite set of points with a cone generated by a finite set of vectors)

$$P = \text{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^m) + \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k),$$

if and only if it is an $\mathcal{H}$-polyhedron (i.e. the intersection of a finite number of closed halfspaces)

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b} \text{ for some } A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m\}.$$

In a special case when the conic part is trivial (i.e., it consists of only $\{\mathbf{0}\}$), we obtain the following corollary, so-called main theorem for convex polytopes.

Theorem 1.29. (Main theorem for convex polytopes).
A subset $P \subseteq \mathbb{R}^n$ is a $\mathcal{V}$-polytope (i.e. the convex hull of a finite set of points)

$$P = \text{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^m),$$

if and only if it is an $\mathcal{H}$-polytope (i.e. a bounded intersection of a finite number of closed halfspaces)

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b} \text{ for some } A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m\}.$$

The proof of Minkowski-Weyl theorem bases on the main theorem for polyhedral cones stated as follows.

Theorem 1.30. (Main theorem for polyhedral cones).
A subset $P \subseteq \mathbb{R}^n$ is a $\mathcal{V}$-cone (i.e. a cone generated by a finite set of vectors)

$$P = \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k),$$

if and only if it is an $\mathcal{H}$-cone (i.e. a finite intersection of closed linear halfspaces)

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{0} \text{ for some } A \in \mathbb{R}^{m \times n}\}.$$

Let us see why Minkowski-Weyl theorem follows from the main theorem for polyhedral cones. The key idea here is to use a technique so-called homogenization. This technique associates with every polyhedron $P \subseteq \mathbb{R}^n$ a cone $C(P) \subseteq \mathbb{R}^{n+1}$ satisfying

$$C(P) = \left\{ \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \mid \mathbf{x} \in P \right\}.$$

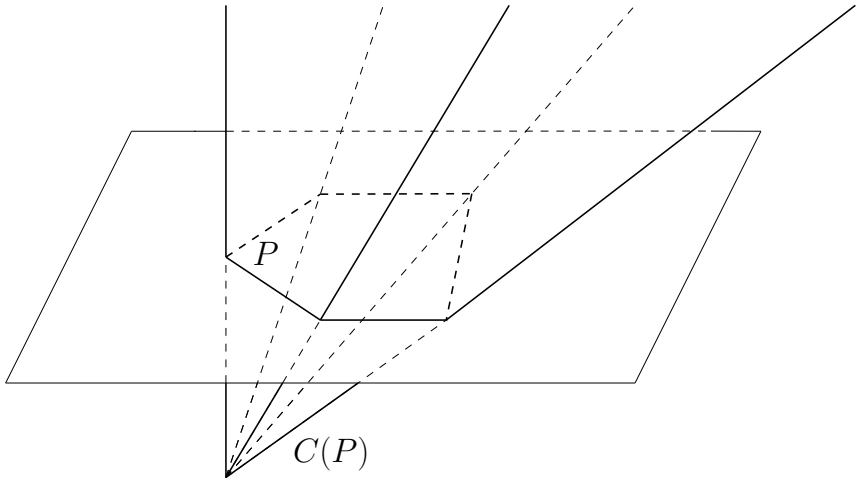


Figure 1.15: Homogenization of a polyhedron.

Proof of Theorem 1.28.

Assumed the main theorem for polyhedral cones (Theorem 1.30), we now prove the Minkowski-Weyl theorem. This proof composes of two parts.

A $\mathcal{V}$-polyhedron is also an $\mathcal{H}$-polyhedron.

Given a $\mathcal{V}$-polyhedron

$$P = \text{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^m) + \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k),$$

in which $\mathbf{x}^1, \ldots, \mathbf{x}^m, \mathbf{v}^1, \ldots, \mathbf{v}^k \in \mathbb{R}^n$. Consider the following cone in $\mathbb{R}^{n+1}$:

$$C(P) := \text{cone}\left(\begin{pmatrix}1\\ \mathbf{x}^1\end{pmatrix}, \ldots, \begin{pmatrix}1\\ \mathbf{x}^m\end{pmatrix}, \begin{pmatrix}0\\ \mathbf{v}^1\end{pmatrix}, \ldots, \begin{pmatrix}0\\ \mathbf{v}^k\end{pmatrix}\right).$$

We claim that

$$\mathbf{x} \in P \qquad \Leftrightarrow \qquad \begin{pmatrix}1\\ \mathbf{x}\end{pmatrix} \in C(P). \tag{1.3}$$

Indeed, let $\mathbf{x}$ be a vector in P, then by definition of P we have

$$\mathbf{x} = \alpha_1\mathbf{x}^1 + \ldots + \alpha_m\mathbf{x}^m + \beta_1\mathbf{v}^1 + \ldots + \beta_k\mathbf{v}^k$$

for some $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_k \geq 0, \alpha_1 + \ldots + \alpha_m = 1$. Hence we can express

$$\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix} = \alpha_1\begin{pmatrix}1\\ \mathbf{x}^1\end{pmatrix} + \ldots + \alpha_m\begin{pmatrix}1\\ \mathbf{x}^m\end{pmatrix} + \beta_1\begin{pmatrix}0\\ \mathbf{v}^1\end{pmatrix} + \ldots + \beta_k\begin{pmatrix}0\\ \mathbf{v}^k\end{pmatrix}$$

which belongs to $C(P)$ by definition of this cone. Conversely, assume that $\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix}$ is a vector in $C(P)$. Then by definition of $C(P)$ we have

$$\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix} = \lambda_1\begin{pmatrix}1\\ \mathbf{x}^1\end{pmatrix} + \ldots + \lambda_m\begin{pmatrix}1\\ \mathbf{x}^m\end{pmatrix} + \lambda_{m+1}\begin{pmatrix}0\\ \mathbf{v}^1\end{pmatrix} + \ldots + \lambda_{m+k}\begin{pmatrix}0\\ \mathbf{v}^k\end{pmatrix}$$

for some $\lambda_1, \ldots, \lambda_{m+k} \geq 0$. Looking at the first coordinate of the vectors in both sides of this equality, we obtain

$$\lambda_1 + \ldots + \lambda_m = 1,$$

while for the other coordinates we obtain

$$\mathbf{x} = \lambda_1 \mathbf{x}^1 + \ldots + \lambda_m \mathbf{x}^m + \lambda_{m+1} \mathbf{v}^1 + \ldots + \lambda_{m+k} \mathbf{v}^k. \tag{1.4}$$

Since $\lambda_1, \ldots, \lambda_m \geq 0$ and $\lambda_1 + \ldots + \lambda_m = 1$, we have

$$\lambda_1 \mathbf{x}^1 + \ldots + \lambda_m \mathbf{x}^m \in \text{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^m).$$

Since $\lambda_{m+1}, \ldots, \lambda_{m+k} \geq 0$, we have

$$\lambda_{m+1} \mathbf{v}^1 + \ldots + \lambda_{m+k} \mathbf{v}^k \in \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k).$$

Thus it follows from (1.4) that $\mathbf{x} \in \text{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^m) + \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k) = P$. This completes the proof of claim (1.3).

Now, looking again at the definition of $C(P)$ we see that it is a $\mathcal{V}$-cone. By the main theorem for polyhedral cones (Theorem 1.30), $C(P)$ is also an $\mathcal{H}$-cone, which means that we can represent $C(P)$ as

$$C(P) = \{\mathbf{z} \in \mathbb{R}^{n+1} \mid A\mathbf{z} \leq \mathbf{0}\}$$

for some matrix $A \in \mathbb{R}^{h \times (n+1)}$ with some $h \in \mathbb{N}$. By claim (1.3) we have

$$\begin{aligned} \mathbf{x} \in P \quad &\Leftrightarrow \quad \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \in C(P) \\ &\Leftrightarrow \quad A \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \leq \mathbf{0} \\ &\Leftrightarrow \quad \widetilde{A}\mathbf{x} \leq -\mathbf{a}^0 \end{aligned}$$

in which $\mathbf{a}^0$ is the first column of matrix A and $\widetilde{A}$ is the remain part of matrix A after removing the first column $\mathbf{a}^0$. This is an $\mathcal{H}$-representation of P, proving that P is an $\mathcal{H}$-polyhedron.

An $\mathcal{H}$-polyhedron is also a $\mathcal{V}$-polyhedron.

Given an $\mathcal{H}$-polyhedron

$$P_H = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b}\}$$

in which $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^n$. Consider the following cone in $\mathbb{R}^{n+1}$:

$$C(P_H) = \left\{ \mathbf{z} \in \mathbb{R}^{n+1} \mid \begin{pmatrix} -1 & 0 \\ -\mathbf{b} & A \end{pmatrix} \mathbf{z} \leq \begin{pmatrix} 0 \\ \mathbf{0} \end{pmatrix} \right\}.$$

We claim that

$$\mathbf{x} \in P_H \qquad \Leftrightarrow \qquad \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \in C(P_H). \tag{1.5}$$

Indeed, we have

$$\begin{aligned}
\mathbf{x} \in P_H \quad &\Leftrightarrow \quad A\mathbf{x} \leq \mathbf{b} \\
&\Leftrightarrow \quad \begin{cases} -1 & \leq 0 \\ A\mathbf{x} & \leq \mathbf{b} \end{cases} \\
&\Leftrightarrow \quad \begin{cases} -1 & \leq 0 \\ -\mathbf{b} + A\mathbf{x} & \leq \mathbf{0} \end{cases} \\
&\Leftrightarrow \quad \begin{pmatrix} -1 & 0 \\ -\mathbf{b} & A \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \leq \begin{pmatrix} 0 \\ \mathbf{0} \end{pmatrix} \\
&\Leftrightarrow \quad \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \in C(P_H).
\end{aligned}$$

Now, looking again at the definition of $C(P_H)$ we see that it is an $\mathcal{H}$-cone. By the main theorem for polyhedral cones (Theorem 1.30), $C(P_H)$ is also a $\mathcal{V}$-cone, which means that we can represent $C(P_H)$ as

$$C(P_H) = \text{cone}(\mathbf{z}^1, \ldots, \mathbf{z}^k)$$

for some $\mathbf{z}^1, \ldots, \mathbf{z}^k \in \mathbb{R}^{n+1}$. Let $\mathbf{x}$ be an arbitrary vector in P_H. Then by claim (1.5) we have $\begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \in C(P_H)$, which means that there exist $\mu_1, \ldots, \mu_k \geq 0$ such that

$$\begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} = \mu_1 \mathbf{z}^1 + \ldots + \mu_k \mathbf{z}^k. \tag{1.6}$$

For each $i = 1, \ldots, k$ we rewrite vector $\mathbf{z}^i$ as follows

$$\mathbf{z}^i = \begin{pmatrix} z_1^i \\ \mathbf{y}^i \end{pmatrix}$$

in which z_1^i is the first coordinate of vector $\mathbf{z}^i$, and $\mathbf{y}^i$ is the remaining part of $\mathbf{z}^i$ after removing its first coordinate. Let

$$I_0 = \{i \in \{1, \ldots, k\} \mid z_1^i = 0\}, \quad I_1 = \{i \in \{1, \ldots, k\} \mid z_1^i \neq 0\}.$$

At this point we take a pause and look again at the definition of $C(P_H)$. It follows from this definition that any vector $\mathbf{z} \in C(P_H)$ has non-negative first coordinate. Therefore, in fact we have

$$I_1 = \{i \in \{1, \ldots, k\} \mid z_1^i > 0\}.$$

Then (1.6) can be rewritten as follows.

$$\begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} = \mu_1 \begin{pmatrix} z_1^1 \\ \mathbf{y}^1 \end{pmatrix} + \ldots + \mu_k \begin{pmatrix} z_1^k \\ \mathbf{y}^k \end{pmatrix}$$

$$\Leftrightarrow \quad \begin{cases} 1 &= \mu_1 z_1^1 + \ldots \mu_k z_1^k \\ \mathbf{x} &= \mu_1 \mathbf{y}^1 + \ldots + \mu_k \mathbf{y}^k \end{cases}$$

$$\Leftrightarrow \quad \begin{cases} 1 &= \sum_{i \in I_1} \mu_i z_1^i \\ \mathbf{x} &= \sum_{i \in I_0} \mu_i \mathbf{y}^i + \sum_{i \in I_1} \mu_i \mathbf{y}^i \end{cases} \tag{1.7}$$

For convenience, for each $i \in I_1$ we denote $\alpha_i := \mu_i z_1^i$, and

$$\mathbf{w}^i := \frac{1}{z_1^i} \mathbf{y}^i.$$

This makes sense since $z_1^i > 0$ for all $i \in I_1$. Note that $\mu_i \geq 0$ for any index i, therefore we have $\alpha_i \geq 0$. We can now rewrite (1.7) as

$$\begin{cases} 1 &= \sum_{i \in I_1} \alpha_i \\ \mathbf{x} &= \sum_{i \in I_0} \mu_i \mathbf{y}^i + \sum_{i \in I_1} \alpha_i \mathbf{w}^i \end{cases}$$

Since $\mu_i \geq 0$ for all $i \in I_0$, $\alpha_i \geq 0$ for all $i \in I_1$, and $\sum_{i \in I_1} \alpha_i = 1$, the last equality above means that we can express $\mathbf{x}$ as the sum of a conic combination of $\mathbf{y}^i (i \in I_0)$ with a convex combination of $\mathbf{w}^i (i \in I_1)$. Since $\mathbf{x}$ is chosen arbitrarily in P_H, we obtain

$$P_H = \text{conv}(\mathbf{w}^i)_{i \in I_1} + \text{cone}(\mathbf{y}^i)_{i \in I_0}.$$

This is a $\mathcal{V}$-representation of P_H, proving that P_H is a $\mathcal{V}$-polyhedron. This completes the proof of Theorem 1.28. □

It is left to prove the main theorem for polyhedral cones (Theorem 1.30) now. The key technique used to prove this theorem is Fourier-Motzkin elimination, which is applied to a system of linear inequalities to projecting down the solution set one dimension at a time. For the detail proof of this theorem, we need some auxiliary results stated in the following lemmas.

Lemma 1.31.
The intersection of a $\mathcal{V}$-cone with a linear subspace is also a $\mathcal{V}$-cone.

Proof. Consider a $\mathcal{V}$-cone $C = \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^m) \subset \mathbb{R}^n$. Without loss of generality, we consider the linear subspace $H_k = \{\mathbf{x} \in \mathbb{R}^n \mid x_k = 0\}$. We will prove $C \cap H_k$ is a $\mathcal{V}$-cone by showing that $C \cap H_k = \text{cone}(V)$ with

$$V = \{\mathbf{v}^i \mid v_k^i = 0\} \cup \{v_k^i \mathbf{v}^j + (-v_k^j)\mathbf{v}^i \mid v_k^i > 0, v_k^j < 0\}.$$

As a remark, the vectors in the latter set in the definition of V are formed by Fourier-Motzkin elimination. These vectors are constructed from the original vectors $\mathbf{v}^1, \ldots, \mathbf{v}^m$ so that their k-th components equal 0.

Firstly, we will prove that $\text{cone}(V) \subset C \cap H_k$. Indeed, the k-th coordinate of every vector in V equals 0. Hence, $\text{cone}(V) \subset H_k$. Furthermore, since each vector in V is a conic combination of $\mathbf{v}^1, \ldots, \mathbf{v}^m$, it follows that $\text{cone}(V) \subset \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^m) = C$.

Now we prove that $C \cap H_k \subset \text{cone}(V)$. Take any vector $\mathbf{v} \in C \cap H_k$. Since $\mathbf{v} \in H_k$, we have $v_k = 0$. Since $\mathbf{v} \in C$, there exist $t_1, \ldots, t_m \geq 0$ such that

$$\mathbf{v} = t_1\mathbf{v}^1 + \ldots + t_m\mathbf{v}^m. \tag{1.8}$$

- If $t_i v_k^i = 0$ for all $i = 1, \ldots, m$, then it follows from (1.8) that

  $$\mathbf{v} = \sum_{i \in I} t_i \mathbf{v}^i$$

  in which $I = \{i \in \{1, \ldots, m\} \mid t_i \neq 0\}$. Note that for all $i \in I$, since $t_i \neq 0$, we must have $v_k^i = 0$. Hence the above expression means that $\mathbf{v} \in \text{cone}(\{\mathbf{v}^i \mid v_k^i = 0\}) \subset \text{cone}(V)$.

- Otherwise, there exists some $i \in \{1, \ldots, m\}$ such that $t_i v_k^i \neq 0$. Let

  $$\alpha = \sum_{i : v_k^i > 0} t_i v_k^i, \quad \beta = \sum_{j : v_k^j < 0} t_j (-v_k^j).$$

  Then we have $\alpha \geq 0$ and $\beta \geq 0$ since the coefficients t_i are non-negative. We see that $\alpha - \beta = v_k = 0$, i.e., $\alpha = \beta$. Furthermore, we must have $\alpha > 0$, since $\alpha = 0$ leads to $\beta = 0$ and consequently $t_j = 0$ for all j with $v_k^j \neq 0$, so $t_j v_k^j = 0$ for all $j = 1, \ldots, m$, contradicting our setting that $t_i v_k^i \neq 0$. Now we can express $\mathbf{v}$ as follows.

  $$\begin{aligned}
  \mathbf{v} &= \sum_{i : v_k^i = 0} t_i \mathbf{v}^i + \sum_{i : v_k^i > 0} t_i \mathbf{v}^i + \sum_{j : v_k^j < 0} t_j \mathbf{v}^j \\
  &= \sum_{i : v_k^i = 0} t_i \mathbf{v}^i + \frac{1}{\alpha} \sum_{i : v_k^i > 0} \alpha t_i \mathbf{v}^i + \frac{1}{\alpha} \sum_{j : v_k^j < 0} \alpha t_j \mathbf{v}^j \\
  &= \sum_{i : v_k^i = 0} t_i \mathbf{v}^i + \frac{1}{\alpha} \sum_{i : v_k^i > 0} \left(\sum_{j : v_k^j < 0} t_j (-v_k^j) \right) t_i \mathbf{v}^i + \frac{1}{\alpha} \sum_{j : v_k^j < 0} \left(\sum_{i : v_k^i > 0} t_i v_k^i \right) t_j \mathbf{v}^j \\
  &= \sum_{i : v_k^i = 0} t_i \mathbf{v}^i + \sum_{i : v_k^i > 0, j : v_k^j < 0} \frac{t_i t_j}{\alpha} (v_k^i \mathbf{v}^j + (-v_k^j) \mathbf{v}^i)
  \end{aligned}$$

  In the right hand side of the last equality, the first sum is a conic combination of vectors in the first set of V, while the second sum is a conic combination of vectors in the second set of V. Therefore, we have $\mathbf{v} \in \text{cone}(V)$.

In both cases considered above, we obtain $\mathbf{v} \in \text{cone}(V)$. Since $\mathbf{v}$ is chosen arbitrarily in $C \cap H_k$, we conclude that $C \cap H_k \subset \text{cone}(V)$. □

Lemma 1.32.
If C is an $\mathcal{H}$-cone in $\mathbb{R}^n$ and $k \in \{1, \ldots, n\}$, then so are

$$\text{elim}_k(C) := \{\mathbf{x} - t\mathbf{e}^k \mid \mathbf{x} \in C, t \in \mathbb{R}\}$$

and

$$\text{proj}_k(C) := \text{elim}_k(C) \cap \{\mathbf{x} \in \mathbb{R}^n \mid x_k = 0\}.$$

Here, $\mathbf{e}^k$ is the unit vector in $\mathbb{R}^n$ whose the k-th coordinate is 1 and the other coordinates are 0. Figure 1.16 illustrates the two sets $\text{elim}_k(C)$ and $\text{proj}_k(C)$ for a set $C \subset \mathbb{R}^2$.

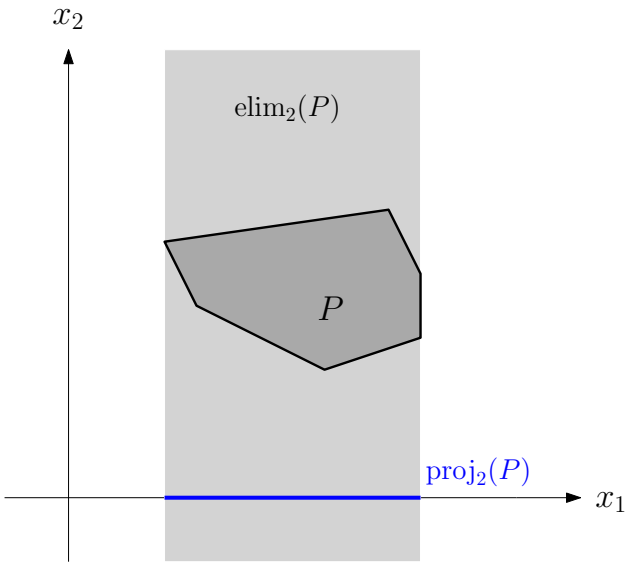


Figure 1.16: Elimination and projection of a set.

Proof. Let $A\mathbf{x} \leq \mathbf{0}$ be the $\mathcal{H}$-representation of the cone C. Let $\mathbf{a}^1, \ldots, \mathbf{a}^m \in \mathbb{R}_n$ be the row vectors of A. We will prove that $\text{elim}_k(C)$ is an $\mathcal{H}$-cone by showing explicitly its $\mathcal{H}$-representation

$$\text{elim}_k(C) = \{\mathbf{x} \in \mathbb{R}^n \mid \tilde{A}\mathbf{x} \leq \mathbf{0}\} =: P_{\tilde{A}},$$

in which $\tilde{A}$ is the matrix whose set of row vectors is

$$R_{\tilde{A}} = \{\mathbf{a}^i \mid \mathbf{a}^i_k = 0\} \cup \{a^i_k \mathbf{a}^j + (-a^j_k)\mathbf{a}^i \mid a^i_k > 0, a^j_k < 0\}.$$

Firstly, we will show that $\text{elim}_k(C) \subset P_{\tilde{A}}$. Indeed, it follows from the definition of $\tilde{A}$ that each row vector of this matrix is a conic combination of row vectors in A. Hence, if $\mathbf{x}$ satisfies $A\mathbf{x} \leq \mathbf{0}$, then $\mathbf{x}$ also satisfies $\mathbf{a}\mathbf{x} \leq 0$ with $\mathbf{a} \in R_{\tilde{A}}$. This implies that $\{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{0}\} \subset \{\mathbf{x} \in \mathbb{R}^n \mid \tilde{A}\mathbf{x} \leq \mathbf{0}\}$, i.e., $C \subset P_{\tilde{A}}$. Note furthermore that the k-th component of every vector in $\mathbb{R}_{\tilde{A}}$ equals 0, hence the variable x_k does not appear in the system $\tilde{A}\mathbf{x} \leq \mathbf{0}$, which implies $\text{elim}_k(C) \subset P_{\tilde{A}}$.

We now prove that $P_{\tilde{A}} \subset \text{elim}_k(C)$. Let $\mathbf{x} \in P_{\tilde{A}}$. We need to show that $\mathbf{x} \in \text{elim}_k(C)$. That is, we have to show that $\mathbf{x} = \tilde{\mathbf{x}} - t\mathbf{e}^k$, or equivalently, $\mathbf{x} + t\mathbf{e}^k = \tilde{\mathbf{x}}$ for some $t \in \mathbb{R}$ and $\tilde{\mathbf{x}} \in C = \{\mathbf{z} \in \mathbb{R}^n \mid A\mathbf{z} \leq \mathbf{0}\}$. Therefore, all we need to do now is to show that there exists $t \in \mathbb{R}$ such that $A(\mathbf{x} + t\mathbf{e}^k) \leq \mathbf{0}$.

Indeed, since $\mathbf{x} \in P_{\tilde{A}}$, we have $\mathbf{a}\mathbf{x} \leq 0$ for all $\mathbf{a} \in R_{\tilde{A}}$. Taking $\mathbf{a}$ in the first set composing $\mathbb{R}_{\tilde{A}}$, we have

$$\mathbf{a}^i\mathbf{x} \leq \mathbf{0} \quad \forall i : a^i_k = 0$$

which implies that for any $t \in \mathbb{R}$ we have

$$\mathbf{a}^i\mathbf{x} + ta^i_k \leq \mathbf{0} \quad \forall i : a^i_k = 0 \tag{1.9}$$

Taking $\mathbf{a}$ in the second set composing $\mathbb{R}_{\tilde{A}}$, we have

$$\begin{aligned}
& (a_k^i \mathbf{a}^j + (-a_k^j)\mathbf{a}^i)\mathbf{x} \le 0 && \forall i,j : a_k^i > 0, a_k^j < 0 \\
\Leftrightarrow \quad & \frac{1}{a_k^i(-a_k^j)}(a_k^i \mathbf{a}^j + (-a_k^j)\mathbf{a}^i)\mathbf{x} \le 0 && \forall i,j : a_k^i > 0, a_k^j < 0 \\
\Leftrightarrow \quad & \frac{1}{-a_k^j}\mathbf{a}^j\mathbf{x} + \frac{1}{a_k^i}\mathbf{a}^i\mathbf{x} \le 0 && \forall i,j : a_k^i > 0, a_k^j < 0 \\
\Leftrightarrow \quad & \frac{1}{a_k^i}\mathbf{a}^i\mathbf{x} \le \frac{1}{-a_k^j}(-\mathbf{a}^j)\mathbf{x} && \forall i,j : a_k^i > 0, a_k^j < 0 \\
\Leftrightarrow \quad & \max_{i:a_k^i>0} \frac{1}{a_k^i}\mathbf{a}^i\mathbf{x} \le \min_{j:a_k^j<0} \frac{1}{-a_k^j}(-\mathbf{a}^j)\mathbf{x}.
\end{aligned}$$

The last inequality above guarantees that we can choose $t \in \mathbb{R}$ such that

$$\max_{i:a_k^i>0} \frac{1}{a_k^i}\mathbf{a}^i\mathbf{x} \le -t \le \min_{j:a_k^j<0} \frac{1}{-a_k^j}(-\mathbf{a}^j)\mathbf{x}. \tag{1.10}$$

It follows from the first inequality in (1.10) that

$$\begin{aligned}
& \frac{1}{a_k^i}\mathbf{a}^i\mathbf{x} \le -t && \forall i : a_k^i > 0 \\
\Leftrightarrow \quad & \mathbf{a}^i\mathbf{x} \le -t a_k^i && \forall i : a_k^i > 0 \\
\Leftrightarrow \quad & \mathbf{a}^i\mathbf{x} + t a_k^i \le 0 && \forall i : a_k^i > 0
\end{aligned} \tag{1.11}$$

Similarly, it follows from the second inequality in (1.10) that

$$\begin{aligned}
& \frac{1}{-a_k^j}(-\mathbf{a}^j)\mathbf{x} \ge -t && \forall j : a_k^j < 0 \\
\Leftrightarrow \quad & -\mathbf{a}^j\mathbf{x} \ge (-t)(-a_k^j) && \forall j : a_k^j < 0 \\
\Leftrightarrow \quad & \mathbf{a}^j\mathbf{x} + t a_k^j \le 0 && \forall j : a_k^j < 0
\end{aligned} \tag{1.12}$$

Gathering (1.9), (1.11), and (1.12), for our chosen t we obtain

$$A\mathbf{x} + tA\mathbf{e}^k \le \mathbf{0}$$

which means $A(\mathbf{x} + t\mathbf{e}^k) \le \mathbf{0}$. This completes the proof of $P_{\tilde{A}} \subset \mathrm{elim}(C)$.

To prove $\mathrm{proj}_k(C)$ is an $\mathcal{H}$-cone, we see that

$$\begin{aligned}
\mathrm{proj}_k(C) &= \mathrm{elim}(C) \cap \{\mathbf{x} \in \mathbb{R}^n \mid x_k = 0\} \\
&= \{x \in \mathbb{R}^n \mid \tilde{A}\mathbf{x} \le \mathbf{0}, -x_k \le 0, x_k \le 0\}
\end{aligned}$$

which gives an $\mathcal{H}$-representation of $\mathrm{proj}_k(C)$. □

Proof of Theorem 1.30.

This proof consists of two parts.

A $\mathcal{V}$-cone is also an $\mathcal{H}$-cone.

Let $C = \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k)$ be a $\mathcal{V}$-cone with $\mathbf{v}^1, \ldots, \mathbf{v}^k \in \mathbb{R}^n$. We can represent C as follows.

$$\begin{aligned} C &= \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} = t_1\mathbf{v}^1 + \ldots + t_k\mathbf{v}^k \text{ for some } t_1, \ldots, t_k \geq 0\} \\ &= \text{proj}_{\mathbf{t}}\{(\mathbf{x}, \mathbf{t}) \in \mathbb{R}^{n+k} \mid \mathbf{x} = V\mathbf{t}, \mathbf{t} \geq \mathbf{0}\} \end{aligned}$$

in which V is the matrix whose column vectors are $\mathbf{v}^1, \ldots, \mathbf{v}^k$ and $\text{proj}_{\mathbf{t}}$ is the operation defined by

$$\text{proj}_{\mathbf{t}}(A) = \text{proj}_{t_1}(\ldots(\text{proj}_{t_{k-1}}(\text{proj}_{t_k}(A))))$$

Since $\{(\mathbf{x}, \mathbf{t}) \in \mathbb{R}^{n+k} \mid \mathbf{x} = V\mathbf{t}, \mathbf{t} \geq \mathbf{0}\} = \{(\mathbf{x}, \mathbf{t}) \in \mathbb{R}^{n+k} \mid \mathbf{x} - V\mathbf{t} \leq 0, V\mathbf{t} - \mathbf{x} \leq 0, -\mathbf{t} \leq \mathbf{0}\}$ is an $\mathcal{H}$-cone, we obtain that C is an $\mathcal{H}$-cone by applying Lemma 1.32 successively.

An $\mathcal{H}$-cone is also a $\mathcal{V}$-cone.

Let $C = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{0}\}$ be an $\mathcal{H}$-cone, in which A is a matrix in $\mathbb{R}^{m \times n}$. We can write it as

$$C \cong \left\{ \begin{pmatrix} \mathbf{x} \\ \mathbf{w} \end{pmatrix} \in \mathbb{R}^{n+m} \mid A\mathbf{x} \leq \mathbf{w} \right\} \bigcap \left\{ \begin{pmatrix} \mathbf{x} \\ \mathbf{w} \end{pmatrix} \in \mathbb{R}^{n+m} \mid \mathbf{w} = \mathbf{0} \right\}$$

Let

$$P = \left\{ \begin{pmatrix} \mathbf{x} \\ \mathbf{w} \end{pmatrix} \in \mathbb{R}^{n+m} \mid A\mathbf{x} \leq \mathbf{w} \right\}.$$

Since each vector in P can be represented as

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{w} \end{pmatrix} = \sum_{i=1}^{n} |x_i| \text{sign}(x_i) \begin{pmatrix} \mathbf{e}^i \\ A\mathbf{e}^i \end{pmatrix} + \sum_{j=1}^{m} (w_j - (A\mathbf{x})_j) \begin{pmatrix} \mathbf{0} \\ \mathbf{e}^j \end{pmatrix},$$

we have

$$P = \text{cone}\left(\left\{ \pm \begin{pmatrix} \mathbf{e}^i \\ A\mathbf{e}^i \end{pmatrix} \mid 1 \leq i \leq n \} \right\} \bigcup \left\{ \begin{pmatrix} \mathbf{0} \\ \mathbf{e}^j \end{pmatrix} \mid 1 \leq j \leq m \right\} \right),$$

which means that P is a $\mathcal{V}$-cone. Since

$$C \cong P \bigcap \left\{ \begin{pmatrix} \mathbf{x} \\ \mathbf{w} \end{pmatrix} \in \mathbb{R}^{n+m} \mid \mathbf{w} = 0 \right\},$$

by applying Lemma 1.31 we obtain that C is a $\mathcal{V}$-cone. □

We closes this subsection by showing some examples of polytopes, in which both $\mathcal{V}$-representation and $\mathcal{H}$-representation are provided.

- d-cube:

$$\begin{aligned} C_d &= \text{conv}(\{+1, -1\}^d) \\ &= \{\mathbf{x} \in \mathbb{R}^d \mid -1 \leq x_i \leq 1 \ \forall i = 1, \ldots, d\}. \end{aligned}$$

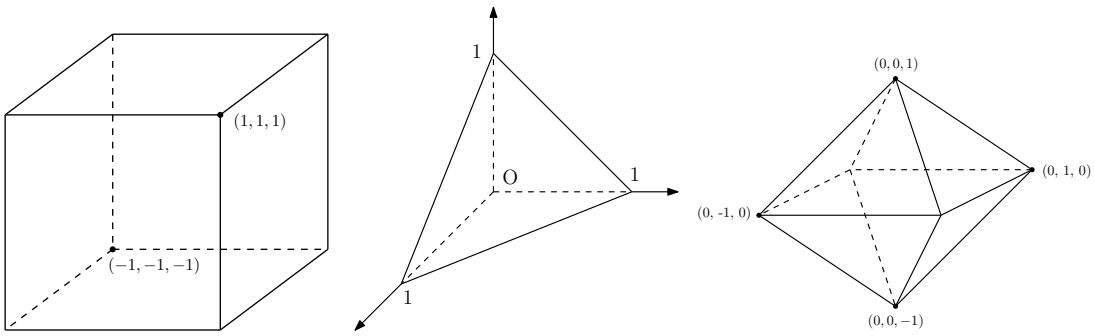
- Standard d-simplex

$$\begin{aligned}\Delta_d &:= \text{conv}(\{\mathbf{e}^1, \ldots, \mathbf{e}^{d+1}\}) \\ &= \{\mathbf{x} \in \mathbb{R}^{d+1} \mid \mathbf{1}^t\mathbf{x} = 1, \mathbf{x} \geq \mathbf{0}\}.\end{aligned}$$

- d-crosspolytope

$$\begin{aligned}C_d^{\Delta} &:= \text{conv}(\{\mathbf{e}^1, -\mathbf{e}^1, \ldots, \mathbf{e}^d, -\mathbf{e}^d\}) \\ &= \{\mathbf{x} \in \mathbb{R}^d \mid \textstyle\sum_{i=1}^d |x_i| \leq 1\}.\end{aligned}$$

Figure 1.17 illustrates these polytopes in $\mathbb{R}^3$.



(a) 3-cube (b) 3-simplex (c) 3-crosspolytope

Figure 1.17: d-cube, standard d-simplex, and d-crosspolytope with $d = 3$.

1.3.3 Faces, extreme points, and extreme rays

Definition 1.33. (Valid inequality).
Let $P \subseteq \mathbb{R}^n$ be a polyhedron. A linear inequality $\mathbf{c}^t\mathbf{x} \leq c_0$ is called a valid inequality for P if

$$\mathbf{c}^t\mathbf{u} \leq c_0 \quad \forall \mathbf{u} \in P.$$

Definition 1.34. (Face).
Let $P \subseteq \mathbb{R}^n$ be a polyhedron.

(i) A face of P is any set of the form $F = P \cap \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{c}^t\mathbf{x} \leq c_0\}$ where $\mathbf{c}^t\mathbf{x} \leq c_0$ is a valid inequality for P.

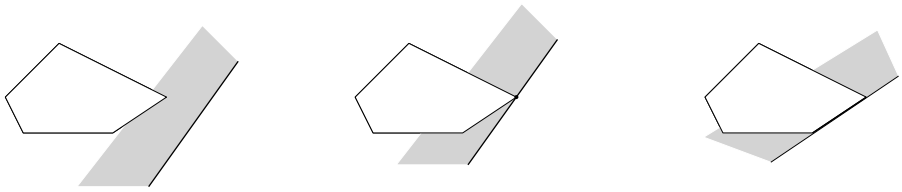
(ii) Any face F of P satisfying $F \subsetneq P$ is called a proper face of P.

(iii) Any face F of P with $\dim(\text{aff}(F)) = 0$ is called a vertex of P.

(iv) Any face F of P with $\dim(\text{aff}(F)) = 1$ is called an edge of P.

(v) Any face F of P with $\dim(\text{aff}(F)) = \dim(\text{aff}(P)) - 1$ is called a facet of P.

Note that by definition, P itself is a face of P by valid inequality $\mathbf{0}^t\mathbf{x} \leq 0$. Figure 1.18 illustrates some valid inequalities of a polytope in $\mathbb{R}^2$ and how they define different faces of the polytope.



(a) Whole polytope (b) Vertex (c) Edge

Figure 1.18: Valid inequalities define some faces of a polytope.

Definition 1.35. (Extreme points of convex sets).
Let $C \subseteq \mathbb{R}^n$ be a convex set. A point $\mathbf{x}^0 \in C$ is called an extreme point of C if there does not exist two distinct points $\mathbf{x}^1, \mathbf{x}^2 \in C$ such that $\mathbf{x}^0 = \alpha\mathbf{x}^1 + (1-\alpha)\mathbf{x}^2$ for some $\alpha \in (0,1)$.

In other words, x^0 is an extreme point of C if it is not a relative interior point of any line segment between two distinct points of C. For instances, if C is a ball in $\mathbb{R}^3$, then any point in its boundary is an extreme point. If C is a hyperplane in $\mathbb{R}^3$, then it has no extreme point. The following proposition shows that for convex polyhedron, the concepts of vertex and extreme point are equivalent.

Proposition 1.36.
Let $P \subset \mathbb{R}^n$ be a convex polyhedron. Then the set of vertices of P coincides the set of extreme points of P.

The following theorem gives us a closer view on the geometrical structure of convex polytopes: every polytope is the convex hull of its vertices.

Theorem 1.37.
Let $P = \text{conv}(V)$ be a polytope in which $V = \{\mathbf{x}^1, \ldots, \mathbf{x}^m\} \subset \mathbb{R}^n$. Let $\text{vert}(P)$ be the set of all vertices of P. Then we have
(i) $\text{vert}(P) \subseteq V$.
(ii) $P = \text{conv}(\text{vert}(P))$.

The following theorem gives us a closer view on the algebraic structure of vertices of convex polyhedra.

Theorem 1.38.
Let $P \subseteq \mathbb{R}^n$ be an $\mathcal{H}$-polyhedron given by

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

for some $A \in \mathbb{R}^{m\times n}$ with $\text{rank}(A) = m$, $\mathbf{b} \in \mathbb{R}^m$. Let $\mathbf{a}^1, \ldots, \mathbf{a}^n$ be column vectors of A. Then $\mathbf{x} = (x_1, \ldots, x_n)$ is a vertex of P if and only if $\{\mathbf{a}^j \mid x_j > 0\}$ is linearly independent.

1.4 Solution methods

1.4.1 Solution existence

It might happen that, for a linear program, there does not exist any solution satisfying all the constraints. In that case, the solution set of the linear program is an empty set, and we say that the linear program is infeasible. When the solution set of a linear program is nonempty, we say that the linear program is feasible. As we have seen from Section 1.3, the feasible set of a linear program, if nonempty, is a convex polyhedron.

It might also happen that, for a feasible linear program, we can find a ray in the solution set along which the objective value tends to infinity. In that case, we say that the linear program is unbounded. If a linear program has a finite optimal objective value, we say that the linear program is bounded. Examples for bounded and unbounded linear programs can be found in Section 1.4.2.

The following proposition gives us a condition in which an optimal solution to a linear program exists.

Proposition 1.39.
If the feasible set of a linear program is bounded (i.e., it is a polytope), then the optimal objective value of the linear program is finite, and an optimal solution to the linear program exists.

The following result shows that, to find an optimal solution to a linear program, we can focus on the vertices of the feasible set.

Theorem 1.40.
If the optimal objective value of a linear program exists, then the optimal objective value must be attained at some vertex of the feasible set.

The above results leads to the following procedure to solve linear programs. In fact, this procedure is the basic framework of simplex algorithm, and also gives us the geometric inside the algorithm.

- Step 1: Start at some vertex of the feasible set.
- Step 2: From the current vertex, move to an adjacent vertex of better objective value.
- Iteratively run Step 2 until:
  - No better vertex is found. In this case, the current vertex is an optimal solution.
  - An unbounded edge is visited, where along this edge, the objective function is unbounded. In this case, the linear program is unbounded.

1.4.2 2D linear programs

For linear programs with two decision variables, their feasible sets are polyhedra on plane, therefore we can easily graph them and following the procedure mentioned in Section 1.4.1. That are illustrated in the following examples.

Example 1.41.

Consider the following linear program

$$\begin{aligned} \max \quad & 5000x_1 + 4200x_2 \\ \text{subject to} & \\ x_1 + x_2 &\leq 40 \\ 200x_1 &\leq 6000 \\ 140x_2 &\leq 4200 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

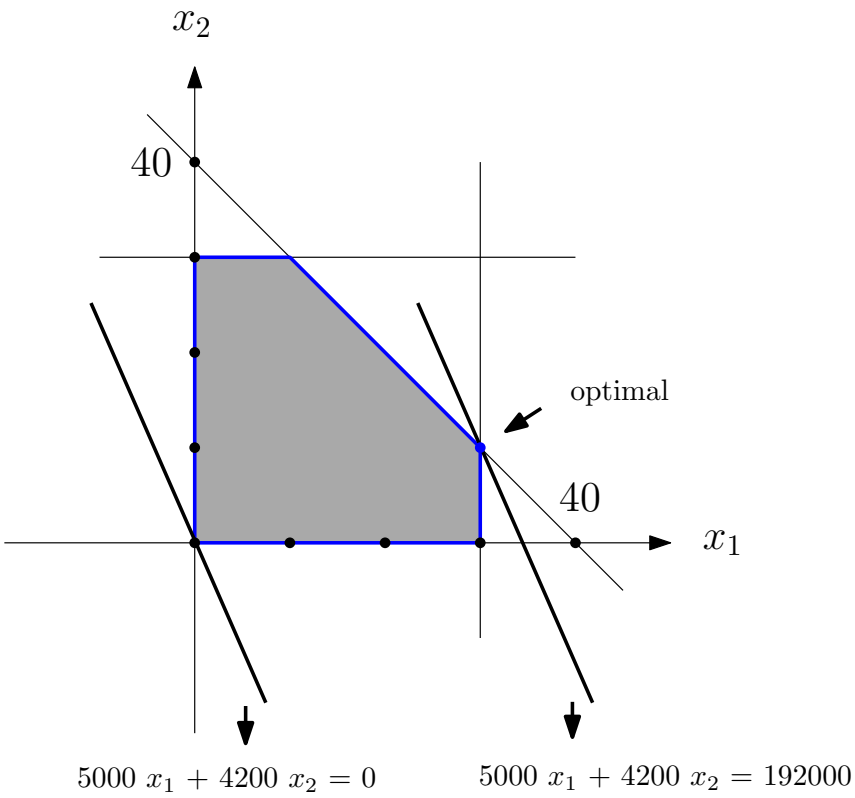


Figure 1.19: Graphical interpretation of the linear program in Example 1.41.

The feasible set of this linear program is visualized by the shaded region in Figure 1.19. The set of points (x_1, x_2) satisfying each constraint (including the nonnegativity

constraints on the variables) is a halfplane. We can determine these half-planes as follows: first, graphing the equation one obtained by replacing the inequality with an equality, and then asking whether or not some specific point satisfies the inequality constraint. Often $(0, 0)$ is chosen, of course any point that is not on the border of the halfplane (i.e., its coordinates do not satisfy the equality) can be used. The feasible set is just the intersection of these halfplanes.

Some level sets of the objective function are also shown in Figure 1.19. One of them passes through the origin which is also a vertex of the feasible set. The objective value corresponding to that level set is 0. The objective function value increases when the corresponding level set moves to the right. The level set corresponding to the case where the objective function equals 192000 is the last level set that touches the feasible set. This is the maximum value of the objective function over the feasible set. The optimal solution is the intersection of this level set with the feasible set, and as we can see from Figure 1.19 this optimal solution is $(30, 10)$.

Example 1.42.

Consider the following linear program

$$\begin{aligned} \max \quad & x_1 + 3x_2 \\ \text{s.t.} \quad & -x_1 - x_2 \leq -3 \\ & -x_1 + x_2 \leq -1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

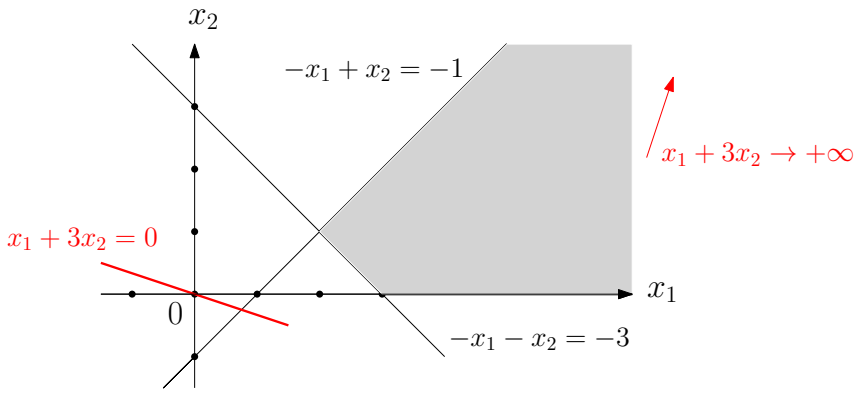


Figure 1.20: Graphical interpretation of the linear program in Example 1.42.

The feasible set of this linear program is visualized by the shaded region in Figure 1.19. It can be seen that this set is unbounded. As we move the level set of the objective function (the red line) to the right, the objective function value increases. Since the feasible set is unbounded, we can move the level set forever to the right without untouching the feasible set, and consequently the objective value increases to infinity. Thus, the linear program is unbounded.

1.4.3 Simplex algorithm

1. Simplex algorithm via an example

We first illustrate how the simplex method works as well as its underlying geometry on the following specific example.

$$\begin{aligned} \max \quad & g = x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 + x_2 \le 100 \\ & 2x_1 + 3x_2 \le 240 \\ & x_2 \le 60 \\ & x_1, x_2 \ge 0 \end{aligned}$$

Note that the linear program in this example is given in standard form, in which all constraints are less-than inequalities. We start by adding so-called slack variables to get equality constraints. For each of the less-than inequalities in the linear program, we introduce a new variable that represents the difference between the right-hand side and the left-hand side. For example, for the first constraint

$$x_1 + x_2 \le 100$$

we introduce the slack variable x_3 defined by

$$x_3 = 100 - x_1 - x_2.$$

Stipulating that $x_3 \ge 0$, the definition of this variable is equivalent to the original constraint. Apply this procedure to each of the less-than constraint, we obtain an equivalent linear program

$$\begin{aligned} \max \quad & x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 = 100 \\ & 2x_1 + 3x_2 + x_4 = 240 \\ & x_2 + x_5 = 60 \\ & x_1, x_2, x_3, x_4, x_5 \ge 0 \end{aligned}$$

To begin the iterative process of simplex method for this linear program, we need an initial feasible solution. To better see how to find such a solution, we first express some variables and the objective function in term of the other variables. Note that, for the linear program we are currently considering, the objective function concerns only the original variables. Therefore, we choose the original variables to express the other (slack) variables, and obtain the following linear program.

$$\begin{aligned} \max \quad & g = x_1 + 2x_2 \\ \text{s.t.} \quad & x_3 = 100 - x_1 - x_2 \\ & x_4 = 240 - 2x_1 - 3x_2 \\ & x_5 = 60 - x_2 \\ & x_1, x_2, x_3, x_4, x_5 \ge 0 \end{aligned}$$

The linear program in this form is called a *dictionary*. We call *basic variables* for the ones chosen to express, and *non-basic variables* the ones to express the others. In the above linear program, x_1 and x_2 are non-basic variables, while x_3, x_4, x_5 are basic ones. Hence, a dictionary is a formulation of a linear program in which the objective function as well as the basic variables are expressed in term of the non-basic variables. Now, we set value 0 to all non-basic variables and then compute the basic variables, and obtain the following solution

$$(x_1, x_2, x_3, x_4, x_5) = (0, 0, 100, 240, 60).$$

The obtained solution is called *basic solution*. Since it satisfies the non-negativity constraints on variables, it is feasible to the linear program we are considering, hence it is a *feasible basic solution*. The objective value at this solution is $g = g_0 = 0$. If we graph the feasible set of the linear program, this solution corresponds to the vertex $A = (0, 0)$ of the feasible set as illustrated in Figure 1.21.

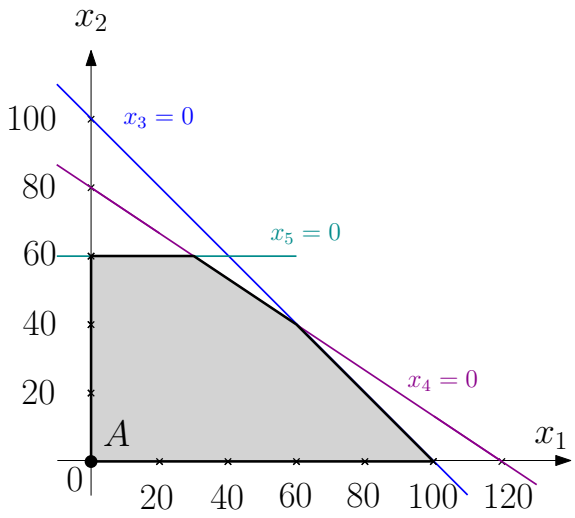


Figure 1.21: A is the vertex in the initial step of simplex algorithm for the example in Section 1.4.3.

Now we see that increasing x_1 or x_2 will bring about an increase in the objective value. Let us choose to increase x_2. Note that when we increase x_2, due to the constraints on the variables, the other variables may have to change their values. We must ensure that these variables remain non-negative. Therefore, we need to determine how much we can increase x_2 without violating the non-negativity of the other variables.

- Since $x_3 = 100 - x_1 - x_2$ and $x_1, x_3 \geq 0$, we must have $x_2 \leq 100$.
- Since $x_4 = 240 - 2x_1 - 3x_2$ and $x_1, x_4 \geq 0$, we must have $x_2 \leq 80$.
- Since $x_5 = 60 - x_2$ and $x_5 \geq 0$, we must have $x_2 \leq 60$.

Hence, we can increase x_2 from its current value 0 to at most 60 while keeping non-negativity of the other variables. Re-compute the values of the other variables, we obtain

the following feasible basic solution

$$(x_1, x_2, x_3, x_4, x_5) = (0, 60, 40, 60, 0).$$

The objective value at this solution is $g = g_1 = 60$, which is better than the previous value $g_0 = 0$. Graphing the feasible set of the linear program, we see that this solution corresponds to the vertex $B = (0, 60)$ of the feasible set as illustrated in Figure 1.22.

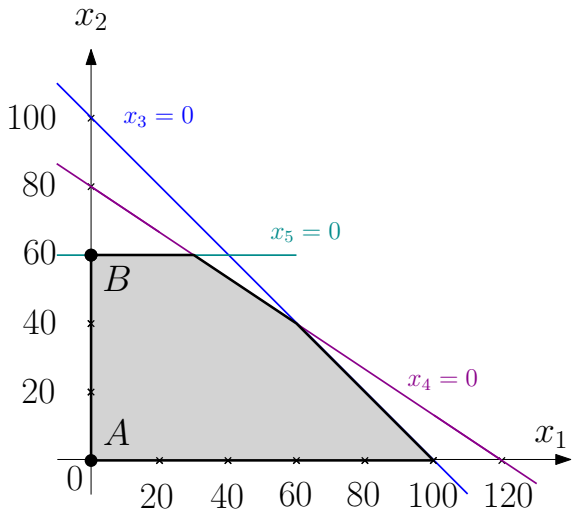


Figure 1.22: B is the vertex in the second step of simplex algorithm for the example in Section 1.4.3.

The variable x_2 was a non-basic variable and set to 0 before being chosen to increase. We will choose it to be a basic variable in the next step. We say that it is an *entering variable*. The variable x_5 was a basic variable and then decreased to 0. We will set it to be a non-basic variable in the next step. We say that it is a *leaving variable*. At this moment we have x_1 and x_5 are non-basic variables. In term of these non-basic variables we express the objective function as well as the other variables and obtain the following dictionary

$$\begin{aligned} \max \quad & g = x_1 + 2(60 - x_5) \\ \text{s.t.} \quad & x_3 = 100 - x_1 - (60 - x_5) \\ & x_4 = 240 - 2x_1 - 3(60 - x_5) \\ & x_2 = 60 - x_5 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

or shortly,

$$\begin{aligned} \max \quad & g = 120 + x_1 - 2x_5 \\ \text{s.t.} \quad & x_3 = 40 - x_1 + x_5 \\ & x_4 = 60 - 2x_1 + 3x_5 \end{aligned}$$

$$x_2 = 60 - x_5$$
$$x_1, x_2, x_3, x_4, x_5 \geq 0$$

For this dictionary we see that the only way to increase the objective value is to increase x_1. Of course when increasing x_1 we must ensure the non-negativity of the other variables. To determine how much we can increase x_1 without violating the non-negativity of the others, we observe that

- Since $x_3 = 40 - x_1 + x_5$ and $x_5 = 0, x_3 \geq 0$, we must have $x_1 \leq 40$.
- Since $x_4 = 60 - 2x_1 + 3x_5$ and $x_5 = 0, x_4 \geq 0$, we must have $x_1 \leq 30$.

Hence, we can increase x_1 from its current value 0 to at most 30 while keeping non-negativity of the other variables. Re-compute the values of the other variables, we obtain the following feasible basic solution

$$(x_1, x_2, x_3, x_4, x_5) = (30, 60, 10, 0, 0).$$

The objective value at this solution is $g = g_2 = 150$, which is better than the previous value $g_1 = 60$. Graphing the feasible set of the linear program, we see that this solution corresponds to the vertex $C = (30, 60)$ of the feasible set as illustrated in Figure 1.23.

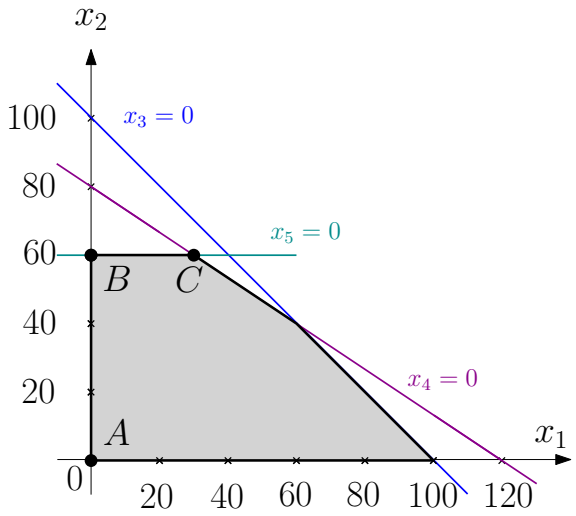


Figure 1.23: C is the vertex in the third step of simplex algorithm for the example in Section 1.4.3.

We have increased x_1 from 0 to 30, hence it is an entering variable and will be a basic variable in the next step. The variable x_4 has decreased from 60 to 0, hence it is a leaving variable and will be a non-basic variable in the next step. At this point, we have x_4 and x_5 are non-basic variables. Expressing the objective function as well as the other variables in term of these new non-basic variables, we obtain the following dictionary

$$\max \quad g = 120 + (30 - x_4 + \frac{3}{2}x_5) - 2x_5$$

$$\begin{aligned} \text{s.t.} \quad & x_2 = 60 - x_5 \\ & x_1 = 30 - x_4 + \frac{3}{2}x_5 \\ & x_3 = 40 - (30 - x_4 + \frac{3}{2}x_5) + x_5 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

or shortly,

$$\begin{aligned} \max \quad & g = 150 - x_4 - \frac{1}{2}x_5 \\ \text{s.t.} \quad & x_2 = 60 - x_5 \\ & x_1 = 30 - x_4 + \frac{3}{2}x_5 \\ & x_3 = 10 + x_4 - \frac{1}{2}x_5 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

Now we observe that there is no way to increase the objective value, since the objective function concerns only non-negative variables x_4, x_5 with negative coefficients. Thus, from the current solution $(x_1, x_2, x_3, x_4, x_5) = (30, 60, 10, 0, 0)$ we cannot improve the objective value, meaning that it is the optimal solution. In term of the original linear program, we have $(x_1, x_2) = (30, 60)$ is the optimal solution, with $g = 150$ is the optimal objective value.

Tracing the solution progress, we start from the vertex A of the feasible set, and then move to the adjacent vertex B with better objective value. From vertex B, we find an adjacent vertex C with better objective value and move to there. At vertex C, we find no adjacent vertices of better objective value, hence we stop there and conclude that it is the optimal solution. This progress illustrates the underlying geometry of the simplex method mentioned at the end of Section 1.4.1.

2. Simplex algorithm description

We have seen the process of simplex algorithm in the above example. Now we describe it in general.

We are given a linear program in standard form

$$\begin{aligned} \max \quad & g = \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i && \forall i = 1, \ldots, m \\ & x_j \geq 0 && \forall j = 1, \ldots, n \end{aligned}$$

Firstly, we add slack variables to the less-than inequalities in the linear program to get

equality constraints:

$$\begin{aligned} \max \quad & g = \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i && \forall i = 1, \ldots, m \\ & x_j \geq 0 && \forall j = 1, \ldots, n+m \end{aligned}$$

Each slack variable represents the difference between the right-hand side and the left-hand side of a constraint in the original linear program, therefore all introduced slack variables are non-negative. Now we express the slack variables in term of the original variables and get the starting dictionary

$$\begin{aligned} \max \quad & g = \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \quad & x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j && \forall i = 1, \ldots, m \\ & x_j \geq 0 && \forall j = 1, \ldots, n+m \end{aligned}$$

In this starting dictionary, we express m slack variables in term of n original variables. We call *basic variables* for the ones chosen to express, and *non-basic variables* for the ones used to express the others. We denote $\mathcal{B}$ the index set of basic variables, and $\mathcal{N}$ the index set of non-basic variables. Hence, for the starting dictionary, we have $\mathcal{N} = \{1, \ldots, n\}$ and $\mathcal{B} = \{n+1, \ldots, n+m\}$.

To obtain the initial solution, we set $x_j = 0$ for all $j \in \mathcal{N}$ and compute the corresponding x_j for $j \in \mathcal{B}$. If the computed values of these basic variables are all non-negative, then the obtained solution is feasible to the linear program, and hence becomes the initial solution for the search of simplex algorithm. At this moment we assume such the case. The situation in which not all computed values of these basic variables are non-negative will be discussed in the last part of this subsection.

The simplex algorithm searches for an optimal solution by moving from one dictionary to another. Each dictionary has m basic variables and n non-basic variables. The current dictionary looks like

$$\begin{aligned} \max \quad & g = \overline{g} + \sum_{j=1}^{n} \overline{c}_j x_j \\ \text{s.t.} \quad & x_{n+i} = \overline{b}_i - \sum_{j=1}^{n} \overline{a}_{ij} x_j && \forall i = 1, \ldots, m \\ & x_j \geq 0 && \forall j = 1, \ldots, n+m \end{aligned}$$

in which the bars over the coefficients indicate that they will change when the algorithm processes. In each iteration, we choose exactly one *entering variable*, (i.e. a variable goes

from non-basic to basic) and exactly one *leaving variable* (i.e. a variable goes from basic to non-basic). The entering variable is chosen with the aim of increasing the objective value when we increase that variable. That is, the entering variable must have positive coefficient in the objective function, or more precisely:

$$\text{“pick } k \text{ from } \{j \in \mathcal{N} \mid \bar{c}_j > 0\}\text{”}.$$

At this point, we have the following optimality condition:

If the set $\{j \in \mathcal{N} \mid \bar{c}_j > 0\}$ is empty, then the current solution is optimal.

If the set consists of more than one element, then we have more than one choice of which element to pick. Often we pick an index k having the largest coefficient $\bar{c}_j$. Once we have decided that x_k will be the entering variable, its value will be increased from 0 to a positive value. Due to the equality constraints of the dictionary, the increase of the entering variable will change the values of the basic variables in the following way

$$x_i = \overline{b}_i - \overline{a}_{ik} x_k \quad \forall i \in \mathcal{B}.$$

We must ensure that these variables remain non-negative by requiring

$$\overline{b}_i - \overline{a}_{ik} x_k \geq 0 \quad \forall i \in \mathcal{B}.$$

Since we are increasing x_k, we can restrict our attention to the indices $i \in \mathcal{B}$ for which $\overline{a}_{ik} > 0$. If none of such indices exists, i.e., $\overline{a}_{ik} \leq 0$ for all $i \in \mathcal{B}$, then we can increase x_k to infinity without violating the non-negativity of the current basic variables. The fact that x_k can increase to infinity leads to the consequence that the objective value is unbounded above, i.e., g can tend to $+\infty$. At this point, we obtain the following condition on the unboundedness of the linear program

If $\overline{a}_{ik} \leq 0$ for all $i \in \mathcal{B}$, then the linear program is unbounded.

Now we consider the case in which there exists some index $i \in \mathcal{B}$ such that $\overline{a}_{ik} > 0$. For such indices, the value of x_k is restricted by

$$x_k \leq \frac{\overline{b}_i}{\overline{a}_{ik}} \quad \forall i \in \mathcal{B}.$$

Hence, we can only increase x_k to the smallest of these values:

$$x_k = \min_{i \in \mathcal{B}: \overline{a}_{ik} > 0} \frac{\overline{b}_i}{\overline{a}_{ik}}.$$

When x_k is increased and reaches that value, the basic variables of the following indices will be reduced to 0:

$$\left\{ i \in \mathcal{B} \mid \overline{a}_{ik} > 0 \text{ and } \frac{\overline{b}_i}{\overline{a}_{ik}} \text{ is minimal} \right\}.$$

The leaving variable is chosen to preserve non-negativity of the current basic variables when increasing the chosen entering variable. Therefore, we choose a leaving variable as a member of the above set, i.e., the rule for selecting a leaving variable is

$$\text{“pick } l \text{ from } \left\{ i \in \mathcal{B} \mid \overline{a}_{ik} > 0 \text{ and } \frac{\overline{b}_i}{\overline{a}_{ik}} \text{ is minimal} \right\}\text{”}.$$

Once the leaving-basic and entering-nonbasic variables have been selected, we express the objective function as well as the new set of basic variables in term of the new set of non-basic variables. By this operation (so-called *pivot*), we move from the current dictionary to the new dictionary. For the new dictionary, we set value 0 to the updated non-basic variables and repeat the above procedure, until either the optimality or unboundedness condition is reached.

3. Simplex algorithm in matrix form

In this part we present the simplex algorithm in matrix form. In this form we can see how the coefficients of a linear program change between dictionaries.

Starting with a linear program in standard form

$$\begin{aligned} \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

by adding non-negative slack variables, we obtain the first dictionary

$$\begin{aligned} \max \quad & g(\mathbf{x}) = \mathbf{c}_B^t\mathbf{x}_B + \mathbf{c}_N^t\mathbf{x}_N \\ \text{s.t.} \quad & A_B\mathbf{x}_B = \mathbf{b} - A_N\mathbf{x}_N \\ & \mathbf{x}_B \geq \mathbf{0} \\ & \mathbf{x}_N \geq \mathbf{0} \end{aligned}$$

in which

- B is the index set of basic variables,
- N is the index set of non-basic variables,
- $\mathbf{x}_B$ is the column vector whose components are basic variables,
- $\mathbf{x}_N$ is the column vector whose components are non-basic variables,
- $\mathbf{c}_B$ is the column vector whose components are coefficients of the basic variables in the objective function,
- $\mathbf{c}_N$ is the column vector whose components are coefficients of the non-basic variables in the objective function,
- A_B is the matrix composed of column vectors $\{\mathbf{a}^i \mid i \in B\}$ of coefficient matrix of constraints,
- A_N is the matrix composed of column vectors $\{\mathbf{a}^i \mid i \in N\}$ of coefficient matrix of constraints.

In the first dictionary, we have

- $\mathbf{x}_N$ consists of original variables from the standard form of the linear program,
- $\mathbf{x}_B$ consists of slack variables added to get the canonical form of the linear program,
- $\mathbf{c}_N = \mathbf{c}$ which is the coefficient vector of the objective function in the standard form of the linear program,
- $\mathbf{c}_B = \mathbf{0}$,
- $A_N = A$ which is the coefficient matrix of the constraints in the standard form of the linear program,
- $A_B = I$ which is the identity matrix.

Since

$$A_B\mathbf{x}_B = \mathbf{b} - A_N\mathbf{x}_N$$

and A_B is invertible, we have

$$\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N. \tag{1.13}$$

To have the initial solution for the simplex algorithm, we set $\mathbf{x}_N = \overline{\mathbf{x}}_N = \mathbf{0}$ and obtain $\mathbf{x}_B = \overline{x}_B = \mathbf{A}_B^{-1}\mathbf{b}$. If $\overline{\mathbf{x}}_B \geq \mathbf{0}$, then $\overline{\mathbf{x}} = \begin{bmatrix}\overline{\mathbf{x}}_N \\ \overline{\mathbf{x}}_B\end{bmatrix}$ is a feasible basic solution to the linear program, and we choose it as an initial solution. If it is not the case, then we need to find another feasible solution to be initial one for the simplex algorithm. The next part of this subsection will discuss such case. At the moment, we assume the first computed value of $\overline{\mathbf{x}}_B$ is a non-negative vector.

Now, we see that

$$\begin{aligned} g(\mathbf{x}) &= \mathbf{c}_B^t\mathbf{x}_B + \mathbf{c}_N^t\mathbf{x}_N \\ &= \mathbf{c}_B^t(\mathbf{A}_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N) + \mathbf{c}_N^t\mathbf{x}_N \\ &= \mathbf{c}_B^t\mathbf{A}_B^{-1}\mathbf{b} + (\mathbf{c}_N^t - \mathbf{c}_B^tA_B^{-1}A_N)\mathbf{x}_N \\ &= g(\overline{x}) + \Delta^t\mathbf{x}_N \end{aligned} \tag{1.14}$$

in which $\Delta = (\mathbf{c}_N^t - \mathbf{c}_B^tA_B^{-1}A_N)^t = \mathbf{c}_N - (\mathbf{A}_B^{-1}A_N)^t\mathbf{c}_B$. There are two cases.

- Case 1: $\Delta \leq \mathbf{0}$.

  In this case, since $\mathbf{x}_N \geq \mathbf{0}$, we have $g(\mathbf{x}) \leq g(\overline{\mathbf{x}})$ for all feasible solution $\mathbf{x}$ of the linear program. That is, $\overline{\mathbf{x}}$ is an optimal solution to this program. Hence, we have the following optimality condition:

  If $\Delta = \mathbf{c}_N - (\mathbf{A}_B^{-1}A_N)^t\mathbf{c}_B \leq \mathbf{0}$*, then the current solution* $\overline{\mathbf{x}}$ *is optimal.*

- Case 2: There exists an index $k \in N$ such that $\Delta_k > 0$.

  In this case, by (1.14) we have the following observation: if we increase x_k while keeping zero value for other non-basic variables (i.e., $x_i = 0$ for all $i \in N\backslash\{k\}$), then the value of objective function $g(\mathbf{x})$ will increase from its current value $g(\overline{x})$. Let us increase x_k from its current value $\overline{x}_k = 0$ to a positive value $\theta > 0$. The variable x_k will be a basic variable in the next iteration, hence it is an entering variable. By (1.13) we have

  $$x_j = (\mathbf{A}_B^{-1}\mathbf{b})_j - (A_B^{-1}A_N)_{jk}\theta \quad \forall j \in B.$$

  By (1.14) we have

  $$g(\mathbf{x}) = g(\overline{\mathbf{x}}) + \Delta_k\theta.$$

  There are two subcases.

  - Subcase 2.1: $(A_B^{-1}A_N)_{jk} \leq 0$ for all $j \in B$.
    In this subcase, since $\mathbf{A}_B^{-1}\mathbf{b} \geq \mathbf{0}$ by our assumption, we can increase θ as we want without violating the non-negativity of basic variables $\mathbf{x}_j (j \in B)$. Observe that since $\Delta_k > 0$, we have $g(\mathbf{x}) \to +\infty$ when $\theta \to +\infty$. Hence, we have the following unboundedness condition:

    If $\Delta_k > 0$ for some $k \in N$, and $(A_B^{-1}A_N)_{jk} \leq 0$ for all $j \in B$, then the linear program is unbounded.

  - Subcase 2.2: There exists an index $j \in B$ such that $(A_B^{-1}A_N)_{jk} > 0$.
    In this subcase, to ensure the non-negativity of all basic variables, the largest acceptable value of θ is

    $$\theta^* = \min\left\{\frac{(\mathbf{A}_B^{-1}\mathbf{b})_j}{(A_B^{-1}A_N)_{jk}} \mid j \in B \text{ such that } (A_B^{-1}A_N)_{jk} > 0\right\}.$$

    Let us choose an index at which the above minimum is attained, i.e.,

    $$j_0 = \arg\min\left\{\frac{(\mathbf{A}_B^{-1}\mathbf{b})_j}{(A_B^{-1}A_N)_{jk}} \mid j \in B \text{ such that } (A_B^{-1}A_N)_{jk} > 0\right\}.$$

    The variable x_{j_0} will be a non-basic variable in the next iteration, hence it is a leaving variable. Then we update the current solution by

    $$\overline{x}_i = \begin{cases} \theta^* & \text{if } i = k \\ 0 & \text{if } i \in N\backslash\{k\} \\ 0 & \text{if } i = j_0 \\ (\mathbf{A}_B^{-1}\mathbf{b})_i - (A_B^{-1}A_N)_{ik}\theta^* & \text{if } i \in B\backslash\{j_0\} \end{cases}$$

    and then update the index sets of basic and non-basic variables as follows

    $$B := B\backslash\{j_0\} \cup \{k\},$$

$$N := N \backslash \{k\} \cup \{j_0\}.$$

With the updated solution as well as updated basic set and updated non-basic set, we come to the next iteration by repeating the arguments from (1.13).

4. Finding an initial feasible solution

As discussed in the description of simplex algorithm in the previous parts, to have an initial solution we set non-basic variables to 0, and compute the basic variables. If the computed values of basic variables are all non-negative, then we obtain a feasible basic solution which is used as an initial solution for the algorithm. What if they are not all non-negative? In this part we discuss how to handle with this difficulty.

We need two phases to solve the linear program in such situation. In Phase I, we solve the following auxiliary problem

$$\begin{aligned} & \max \quad -x_0 & \\ \text{s.t.} \quad & a_{i1}x_1 + \ldots + a_{in}x_n - x_0 \leq b_i & \forall i = 1, \ldots, m \\ & x_j \geq 0 & \forall j = 0, 1, \ldots, n \end{aligned}$$

For this auxiliary problem we will see that

- a feasible dictionary is easy to find, and
- the optimal dictionary provides a feasible dictionary for the original problem.

Indeed, it is easy to give a feasible solution to this auxiliary problem. We simply set $x_j = 0$ for all $j = 1, \ldots, n$ and then pick x_0 sufficiency large. Furthermore, the original problem has a feasible solution if and only if the auxiliary problem has optimal objective value zero. In other word, if the auxiliary problem has a non-zero optimal objecitve value, then the original linear program is infeasible. If the auxiliary problem in Phase I has an optimal solution with optimal objective value 0, then we come to Phase II in which we solve the original linear program with the initial feasible solution obtained by the optimal solution in Phase I.

Although it is clear to see that the auxiliary problem has feasible solutions, we have not yet shown that it has an easily obtained feasible dictionary. If the first dictionary of the auxiliary problem is infeasible (i.e., not all basic variables are non-negative when we assign 0 to non-basic variables), we can easily to convert it into a feasible dictionary by pivoting variable x_0 with the "most infeasible variable" (i.e. the basic variable with the smallest value of free parameter in its expression via non-basic variable). It is best to illustrate this technique with the following example.

Consider the following linear program

$$\begin{aligned} \max \quad & -2x_1 - x_2 \\ \text{s.t.} \quad & -x_1 + x_2 \leq -1 \\ & -x_1 - 2x_2 \leq -2 \\ & x_2 \leq 1 \end{aligned}$$

$$x_1, x_2 \geq 0$$

The auxiliary problem corresponds to this linear program is

$$\begin{aligned} & \max && -x_0 \\ \text{s.t.} \quad & && -x_1 + x_2 - x_0 \leq -1 \\ & && -x_1 - 2x_2 - x_0 \leq -2 \\ & && x_2 - x_0 \leq 1 \\ & && x_0, x_1, x_2 \geq 0 \end{aligned}$$

Introducing slack variables and then expressing them in term of original variables in the auxiliary problem, we obtain the first dictionary

$$\begin{aligned} \max \quad & -x_0 \\ \text{s.t.} \quad & x_3 = -1 + x_1 - x_2 + x_0 \\ & x_4 = -2 + x_1 + 2x_2 + x_0 \\ & x_5 = 1 - x_2 + x_0 \\ & x_0, \ldots, x_5 \geq 0 \end{aligned}$$

If we set the non-basic variable x_0, x_1, x_2 to 0, then the basic variables will obtain the value $(x_3, x_4, x_5) = (-1, -2, 1)$ which are not all non-negative. In this sense we say that this is an *infeasible dictionary*. In the obtained value for basic variables, we choose the variable of smallest value $x_4 = -2$ and make a pivot with the non-basic variable x_0, and obtain the following dictionary.

$$\begin{aligned} \max \quad & -2 + x_1 + 2x_2 - x_4 \\ \text{s.t.} \quad & x_0 = 2 - x_1 - 2x_2 + x_4 \\ & x_3 = 1 - 3x_2 + x_4 \\ & x_5 = 3 - x_1 - 3x_2 + x_4 \\ & x_0, \ldots, x_5 \geq 0 \end{aligned}$$

Now, if we set the non-basic variable x_1, x_2, x_4 to 0, then the basic variables will obtain the value $(x_0, x_3, x_5) = (2, 1, 3)$ which are all non-negative. Hence, this dictionary is feasible. The choice of basic variable of smallest value before pivoting ensures the feasibility of this dictionary.

1.5 Duality

1.5.1 Motivation

Given a linear program in its standard or canonical form. Each feasible solution provides a lower bound on the optimal objective value of the linear program. To evaluate this bound (how close is it to the optimal value?) we need to have upper bounds. Duality theory gives us a way to compute such upper bounds. To see the construction as well as the economic meaning of dual linear programs, we consider the following example.

Example 1.43.

Each day, a small company produces three kinds of goods $G1, G2, G3$ from two kinds of material $M1, M2$. The available amount of each material, the necessary amount of each material to produce each unit of goods, the revenue per unit of each goods are given in Table 1.1.

	M1 (kg)	M2 (kg)	Revenue (USD/unit)
G1	2	3	10
G2	4	1	8
G3	3	3	9
Available amount	30	20	

Table 1.1: Data for the problems in Example 1.43.

The company wants to have a production plan to maximize total revenue. Let x_j be the produced amount of goods Gj in the production plan ($j = 1, 2, 3$), then an optimal production plan of the company is an optimal solution to the following linear program.

$$\begin{aligned} (P) \qquad \max \quad & \alpha = 10x_1 + 8x_2 + 9x_3 \\ \text{s.t.} \quad & 2x_1 + 4x_2 + 3x_3 \leq 30 \\ & 3x_1 + x_2 + 3x_3 \leq 20 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

In this linear program, the objective function computes the total revenue of all produced goods. The first constraint ensures that the total amount of material $M1$ used to produce the goods does not exceed the available amount of this material. The second constraint ensures the similar condition but for material $M2$. The variable domain means that we cannot produce a negative unit of goods.

To see how the dual problem arises, we consider a broader context, in which there is a buyer who wants to buy all available materials of company in that day. From the perspective of a material seller, the company accepts to sell the materials only when the revenue of selling material amount in each goods must be no less than the revenue of selling the goods. From the perspective of a material buyer, the buyer wants to buy the materials with lowest cost. The seller and the buyer will agree about a price y_i for each kg of material Mi ($i = 1, 2$). The optimal decision of the buyer is an optimal solution of the following linear program.

$$\begin{aligned} (D) \qquad \min \quad & \beta = 30y_1 + 20y_2 \\ \text{s.t.} \quad & 2y_1 + 3y_2 \geq 10 \\ & 4y_1 + y_2 \geq 8 \\ & 3y_1 + 3y_2 \geq 9 \\ & y_1, y_2 \geq 0 \end{aligned}$$

In this linear program, the objective function computes the total cost the buyer has to pay to buy all the available materials. Each constraint of the linear program ensures that the revenue of selling material amount in each goods must be no less than the revenue of selling the goods. Naturally, the price of each material must be a non-negative value.

The linear program (D) is the dual program of the linear program (P). We have a useful observation that $\alpha \leq \beta$, meaning that the revenue of selling all completed goods must be less than or equal to the revenue of selling the materials. This is obvious due to the requirement of the company as the material seller.

1.5.2 Dual linear programs

In this subsection, we first show how to construct the dual to a linear program by using Lagrange function. Consider a linear program in its standard formulation

$$\begin{aligned} (P) \qquad \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

with $A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m$. We say (P) is the primal program.

Definition 1.44. (Lagrange function).
The function

$$L(\mathbf{x}, \mathbf{y}) = \mathbf{c}^t\mathbf{x} + (\mathbf{b} - A\mathbf{x})^t\mathbf{y},$$

that determines on $\mathbb{R}^n_+ \times \mathbb{R}^m_+$, is called Lagrange function of the linear program (P).

If $\mathbf{x}$ is a feasible solution to the primal program (P), then we have $\mathbf{b} - A\mathbf{x} \geq \mathbf{0}$ and therefore in this case we have

$$\min_{\mathbf{y}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}) = \mathbf{c}^t\mathbf{x},$$

in which the equality is attained at $\mathbf{y} = \mathbf{0}$. If $\mathbf{x} \geq \mathbf{0}$ but it is not a feasible solution to the primal program (P), then there exists an index $i \in \{1, \ldots, m\}$ such that $(\mathbf{b} - A\mathbf{x})_i < 0$. If we choose $\mathbf{y} = \theta\mathbf{e}^i \geq \mathbf{0}$ in which $\theta > 0$ and $\mathbf{e}^i$ is the unit vector in $\mathbb{R}^m$ with i-th component equals 1, then $L(\mathbf{x}, \mathbf{y}) = \mathbf{c}^t\mathbf{x} + \theta(\mathbf{b} - A\mathbf{x})_i$. Since $(\mathbf{b} - A\mathbf{x})_i < 0$, we have $L(\mathbf{x}, \mathbf{y}) \to -\infty$ when $\theta \to +\infty$. Hence

$$\min_{\mathbf{y}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}) = \begin{cases} \mathbf{c}^t\mathbf{x} & \text{if } \mathbf{x} \in \mathbb{R}^n_+ \text{ satisfying } A\mathbf{x} \leq \mathbf{b}, \\ -\infty & \text{otherwise,} \end{cases} \tag{1.15}$$

and therefore

$$(P) \quad \Leftrightarrow \quad \max_{\mathbf{x}\geq\mathbf{0}} \min_{\mathbf{y}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}). \tag{1.16}$$

The dual program (D) to the linear program (P) is obtained by interchanging the order of taking maximum and minimum in (1.16). That is

$$(D) \quad \Leftrightarrow \quad \min_{\mathbf{y}\geq\mathbf{0}} \max_{\mathbf{x}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}). \tag{1.17}$$

Note that we can rewrite

$$L(\mathbf{x}, \mathbf{y}) = \mathbf{c}^t\mathbf{x} + (\mathbf{b} - A\mathbf{x})^t\mathbf{y} = \mathbf{b}^t\mathbf{y} + (\mathbf{c} - A^t\mathbf{y})^t\mathbf{x},$$

therefore by similar arguments to (1.15) we have

$$\max_{\mathbf{x}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}) = \begin{cases} \mathbf{b}^t\mathbf{y} & \text{if } \mathbf{y} \in \mathbb{R}^m_+ \text{ satisfying } A^t\mathbf{y} \geq \mathbf{c}, \\ +\infty & \text{otherwise.} \end{cases}$$

So dual program (D) has the following form

$$\begin{aligned} (D) \qquad \min \quad & \mathbf{b}^t\mathbf{y} \\ \text{s.t.} \quad & A^t\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned}$$

More explicitly, when the primal linear program is of standard form

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\ & \qquad\qquad\qquad\qquad \ldots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \\ & x_1, x_2, \ldots, x_n \geq 0 \end{aligned}$$

its dual has the following explicit formulation

$$\begin{aligned} \min \quad & b_1y_1 + b_2y_2 + \ldots + b_my_m \\ \text{s.t.} \quad & a_{11}y_1 + a_{21}y_2 + \ldots + a_{m1}y_m \geq c_1 \\ & a_{12}y_1 + a_{22}y_2 + \ldots + a_{m2}y_m \geq c_2 \\ & \qquad\qquad\qquad\qquad \ldots \\ & a_{1n}y_1 + a_{2n}y_2 + \ldots + a_{mn}y_m \geq c_n \\ & y_1, y_2, \ldots, y_m \geq 0 \end{aligned}$$

The variables $y_1, \ldots, y_m$ in the dual program are called *dual variables*. One can see that each dual variable corresponds to a constraint in the primal program. The following proposition shows that taking the dual of the dual returns us to the primal.

Proposition 1.45. *The primal linear program is dual to its dual program.*

Proof. Let us consider a primal linear program in its standard form as above. We first must write the dual problem in standard form.

$$\begin{aligned} \max \quad & -b_1y_1 - b_2y_2 - \ldots - b_my_m \\ \text{s.t.} \quad & -a_{11}y_1 - a_{21}y_2 - \ldots - a_{m1}y_m \leq -c_1 \end{aligned}$$

$$\begin{aligned} -a_{12}y_1 - a_{22}y_2 - \ldots - a_{m2}y_m &\leq -c_2 \\ &\ldots \\ -a_{1n}y_1 - a_{2n}y_2 - \ldots - a_{mn}y_m &\leq -c_n \\ y_1, \ldots, y_m &\geq 0 \end{aligned}$$

Now, following the previous discussion, we can take the dual of the dual program which is currently in its standard form:

$$\begin{aligned} \min \quad & -c_1x_1 - c_2x_2 - \ldots - c_nx_n \\ \text{s.t.} \quad & -a_{11}x_1 - a_{12}x_2 - \ldots - a_{1n}x_n \geq -b_1 \\ & \ldots \\ & -a_{m1}x_1 - a_{m2}x_2 - \ldots - a_{mn}x_n \geq -b_m \\ & x_1, \ldots, x_n \geq 0 \end{aligned}$$

By changing the sign of the objective as well as constraints, we obtain the equivalent program

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\ & \ldots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \\ & x_1, x_2, \ldots, x_n \geq 0 \end{aligned}$$

which is clearly the primal program. □

We have seen the construction of the dual program when the primal one is given in its standard form. What if the primal program is given in general form? Naturally, we can transfer the primal program to its standard form and then taking the dual program by following the above mentioned process. We then come up with Table 1.2 which summarizes the rules of constructing the dual to a linear program.

Primal	Dual
max $\mathbf{c}^t\mathbf{x}$	min $\mathbf{b}^t\mathbf{y}$
min $\mathbf{c}^t\mathbf{x}$	max $\mathbf{b}^t\mathbf{y}$
$x_i \geq 0$	$a_{1i}y_1 + \ldots + a_{mi}y_m \geq c_i$
$x_i \leq 0$	$a_{1i}y_1 + \ldots + a_{mi}y_m \leq c_i$
$x_i \in \mathbb{R}$	$a_{1i}y_1 + \ldots + a_{mi}y_m = c_i$
$a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i$	$y_i \geq 0$
$a_{i1}x_1 + \ldots + a_{in}x_n \geq b_i$	$y_i \leq 0$
$a_{i1}x_1 + \ldots + a_{in}x_n = b_i$	$y_i \in \mathbb{R}$

Table 1.2: Rules of constructing the dual to a linear program.

As an example, if we are given a primal linear program

$$\begin{aligned} \min \quad & 5x_1 - x_2 + 2x_3 + 3x_4 \\ \text{s.t.} \quad & 3x_1 - x_2 + 2x_3 + 3x_4 = 80 \\ & x_1 + x_2 + x_3 + x_4 \geq -10 \\ & x_1 + x_3 \leq 6 \\ & x_1 \leq 0 \\ & x_2, x_3 \geq 0 \\ & x_4 \in \mathbb{R} \end{aligned}$$

then its dual is

$$\begin{aligned} \max \quad & 80y_1 - 10y_2 + 6y_3 \\ \text{s.t.} \quad & 3y_1 + y_2 + y_3 \geq 5 \\ & -y_1 + y_2 \leq -1 \\ & 2y_1 + y_2 + y_3 \leq 2 \\ & 3y_1 + y_2 = 3 \\ & y_1 \in \mathbb{R} \\ & y_2 \geq 0 \\ & y_3 \leq 0 \end{aligned}$$

1.5.3 Weak duality

As discussed in Section 1.5.1, the dual program provides upper bounds for the primal objective function value. We will state this fact more mathematically as follows.

Let us consider a primal linear program in its standard form

$$\begin{aligned} (P) \qquad \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

and its dual program

$$\begin{aligned} (D) \qquad \min \quad & \mathbf{b}^t\mathbf{y} \\ \text{s.t.} \quad & A^t\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned}$$

Theorem 1.46. (Weak duality theorem).
For any feasible solution $\mathbf{x}$ *of the primal program and for any feasible solution* $\mathbf{y}$ *of the dual program, we have*

$$\mathbf{c}^t\mathbf{x} \leq \mathbf{b}^t\mathbf{y}.$$

Proof. Here we provide a proof based on the use of Lagrange function associated with the primal program

$$L(\mathbf{x}, \mathbf{y}) = \mathbf{c}^t\mathbf{x} + (\mathbf{b} - A\mathbf{x})^t\mathbf{y} = \mathbf{b}^t\mathbf{y} + (\mathbf{c} - A^t\mathbf{y})^t\mathbf{x}.$$

For any $\mathbf{y}^* \geq \mathbf{0}$ we have

$$\min_{\mathbf{y}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}) \leq L(\mathbf{x}, \mathbf{y}^*) \quad \forall \mathbf{x} \geq \mathbf{0},$$

which implies

$$\max_{\mathbf{x}\geq\mathbf{0}} \min_{\mathbf{y}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}) \leq \max_{\mathbf{x}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}^*).$$

Since this inequality holds for arbitrary $\mathbf{y}^* \geq \mathbf{0}$, we have

$$\max_{\mathbf{x}\geq\mathbf{0}} \min_{\mathbf{y}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}) \leq \min_{\mathbf{y}^*\geq\mathbf{0}} \max_{\mathbf{x}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}^*) = \min_{\mathbf{y}\geq\mathbf{0}} \max_{\mathbf{x}\geq\mathbf{0}} L(\mathbf{x}, \mathbf{y}).$$

Keeping in mind (1.16) and (1.17), the weak duality theorem follows immediately from the above inequality. □

An advantage of the above proof is that it can be applied to other Lagrange functions that are not neccessarily associated with a linear program. One can have the following direct proof for the theorem

$$\mathbf{c}^t\mathbf{x} \leq (A^t\mathbf{y})^t\mathbf{x} = \mathbf{y}^tA\mathbf{x} \leq \mathbf{y}^t\mathbf{b} = \mathbf{b}^t\mathbf{y},$$

where the first inequality follows from the fact that $\mathbf{x} \geq \mathbf{0}$ and $A^t\mathbf{y} \geq \mathbf{c}$, while the second inequality follows from the fact that $\mathbf{y} \geq \mathbf{0}$ and $A\mathbf{x} \leq \mathbf{b}$.

The following proposition can be seen as an immediate corollary of the weak duality theorem.

Proposition 1.47.

(i) The primal problem is unbounded if and only if the dual problem is infeasible.

(ii) The dual problem is unbounded if and only if the primal problem is infeasible.

To illustrate the result (i) in Proposition 1.47, let us consider the following primal linear program

$$\begin{aligned} \max \quad & -3x_1 + 6x_2 + 6x_3 \\ \text{s.t.} \quad & -x_1 + 2x_2 - x_3 \leq 4 \\ & -x_1 - x_2 + 2x_3 \leq 7 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

It is easy to see that this primal program is unbounded. Indeed, any point of the form $(x_1, x_2, x_3) = (t, t, t)$ with $t \geq 0$ clearly satisfies the constraints, while its objective value is $9t$. Taking $t \to +\infty$ leads to unboundedness of the program. Now, taking a look at the dual program

$$\min \quad 4y_1 + 7y_2$$

$$\begin{aligned} \text{s.t.} \quad & -y_1 - y_2 \geq -3 \\ & 2y_1 - y_2 \geq 6 \\ & -y_1 + 2y_2 \geq 6 \\ & y_1, y_2 \geq 0 \end{aligned}$$

This program is obviously infeasible, since if we sum up the last two constraints, then we obtain $y_1 + y_2 \geq 12$, which contradicts the first constraint.

To illustrate the result (ii) in Proposition 1.47, let us consider the following primal linear program

$$\begin{aligned} \max \quad & 4x_1 + 7x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 3 \\ & -2x_1 + x_2 \leq -6 \\ & x_1 - 2x_2 \leq -6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

This program is infeasible, since if sum up the last two constraints, then we obtain $-x_1 - x_2 \leq -12$, which contradicts the first constraint. Now, taking a look at the dual program

$$\begin{aligned} \min \quad & 3y_1 - 6y_2 - 6y_3 \\ \text{s.t.} \quad & y_1 - 2y_2 + y_3 \geq 4 \\ & y_1 + y_2 - 2y_3 \geq 7 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

It can be seen that this program is unbounded. Indeed, any point of the form $(y_1, y_2, y_3) = (2t, t, t)$ with $t \geq 7$ is feasible to this program, while its objective value is $-6t$. Taking $t \to +\infty$ leads to unboundedness of the program.

1.5.4 Strong duality

For a linear program of standard form, the weak duality theorem tells us that the set of primal values lies entirely to the left of the set of dual values on the real line. One may think that there is a gap between these two sets. However, for linear programs, we have a stronger result which states that there is no gap between the set of primal values and the set of dual values. The following theorem states this result mathematically.

Let us consider a primal linear program in its standard form

$$\begin{aligned} (P) \qquad \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

and its dual program

$$(D) \qquad \min \quad \mathbf{b}^t\mathbf{y}$$

$$\begin{aligned} \text{s.t.} \quad & A^t\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned}$$

Theorem 1.48. (Strong duality theorem).
If the primal program has an optimal solution $\mathbf{x}^*$*, then the dual program also has an optimal solution* $\mathbf{y}^*$ *such that* $\mathbf{c}^t\mathbf{x}^* = \mathbf{b}^t\mathbf{y}^*$.

The following immediate corollary is very useful in the sense that it provides a certificate of optimality for solving linear programs.

Corollary 1.49. (A certificate of optimality).
The necessary and sufficient condition for a pair of solutions $(\mathbf{x}^*, \mathbf{y}^*)$ *to be the optimal solutions of the pair of dual problems* (P) *and* (D) *is that* $\mathbf{c}^t\mathbf{x}^* = \mathbf{b}^t\mathbf{y}^*$.

In the following we will see how useful is the certificate of optimality, provided that we are given a primal solution $\mathbf{x}^*$ to the primal program and a dual solution $\mathbf{y}^*$ to the dual program. All we need to do are the following easy tasks:

- check that the primal solution is feasible for the primal problem,
- check that the dual solution is feasible for the dual problem,
- and the primal and dual objective values agree.

If it is the case, then we know that these solutions are optimal without using any solution approach such as simplex method. That dramatically reduces the amount of solving time! As an illustration for this certificate, we consider the following primal linear program

$$\begin{array}{rrcl} \max & x_1 + 2x_2 + 3x_3 - 4x_4 & & \\ \text{s.t.} & x_1 - x_2 + x_3 & \leq & 1 \\ & x_1 - x_3 - x_4 & \leq & -1 \\ & x_2 + x_3 + x_4 & \leq & 1 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

and its dual program

$$\begin{array}{rrcl} \min & y_1 - y_2 + y_3 & & \\ \text{s.t.} & y_1 + y_2 & \geq & 1 \\ & -y_1 + y_3 & \geq & 2 \\ & y_1 - y_2 + y_3 & \geq & 3 \\ & -y_2 + y_3 & \geq & -4 \\ & y_1, y_2, y_3 & \geq & 0 \end{array}$$

Given a primal solution $\mathbf{x} = (0, 0, 1, 0)$ and a dual solution $\mathbf{y} = (1, 1, 3)$. The readers will find no difficulty to check that

- $\mathbf{x}$ is a feasible solution of primal problem with objective value 3,
- $\mathbf{y}$ is a feasible solution of dual problem with objective value 3.

Since the objective values of these solutions agree, by the certificate of optimality we conclude that $\mathbf{x}$ is an optimal solution of the primal problem and $\mathbf{y}$ is an optimal solution of the dual problem.

1.5.5 Complementary slackness

This subsection concerns the following context. Consider a primal linear program

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \le b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \le b_2 \\ & \qquad\qquad \ldots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \le b_m \\ & x_1, x_2, \ldots, x_n \ge 0 \end{aligned}$$

and an optimal solution $\mathbf{x}^*$ to this program. Its dual program is

$$\begin{aligned} \min \quad & b_1y_1 + b_2y_2 + \ldots + b_my_m \\ \text{s.t.} \quad & a_{11}y_1 + a_{21}y_2 + \ldots + a_{m1}y_m \ge c_1 \\ & a_{12}y_1 + a_{22}y_2 + \ldots + a_{m2}y_m \ge c_2 \\ & \qquad\qquad \ldots \\ & a_{1n}y_1 + a_{2n}y_2 + \ldots + a_{mn}y_m \ge c_n \\ & y_1, y_2, \ldots, y_m \ge 0 \end{aligned}$$

Of course one can use simplex method to solve the dual program. However, in that way the information of the primal optimal solution $\mathbf{x}^*$ plays no role. Can we take into account this information to compute an optimal solution of the dual problem? In this regard, the following theorem gives a help.

Theorem 1.50. (Complementary slackness theorem).
Let $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$ be a feasible solution of primal problem. Let $\mathbf{y}^* = (y_1^*, \ldots, y_m^*)$ be a feasible solution of dual problem. Then $\mathbf{x}^*$ and $\mathbf{y}^*$ are optimal solutions of their corresponding problems if and only if

$$(b_i - a_{i1}x_1^* - a_{i2}x_2^* - \ldots - a_{in}x_n^*)y_i^* = 0 \qquad (i = 1, \ldots, m) \tag{1.18}$$

$$(a_{1j}y_1^* + a_{2j}y_2^* + \ldots + a_{mj}y_m^* - c_j)x_j^* = 0 \qquad (j = 1, \ldots, n) \tag{1.19}$$

Proof. Let

$$\begin{aligned} u_i &= (b_i - a_{i1}x_1^* - a_{i2}x_2^* - \ldots - a_{in}x_n^*)y_i^* && (i = 1, \ldots, m), \\ v_j &= (a_{1j}y_1^* + a_{2j}y_2^* + \ldots + a_{mj}y_m^* - c_j)x_j^* && (j = 1, \ldots, n). \end{aligned}$$

Since $\mathbf{x}^*$ and $\mathbf{y}^*$ are respectively feasible solutions of the primal problem and dual problem, we have $u_i \ge 0, v_j \ge 0$ for all $i = 1, \ldots, m$ and for all $j = 1, \ldots, n$. Furthermore, we observe that

$$\sum_{i=1}^{m} u_i + \sum_{j=1}^{n} v_j = \sum_{i=1}^{m} \left(b_i - \sum_{j=1}^{n} a_{ij}x_j^* \right) y_i^* + \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij}y_i^* - c_j \right) x_j^*$$

$$= \sum_{j=1}^{n} c_j x_j^* - \sum_{i=1}^{m} b_i y_i^*$$
$$= \mathbf{c}^t\mathbf{x} - \mathbf{b}^t\mathbf{y}. \tag{1.20}$$

By strong duality theorem for linear programs, $\mathbf{x}^*$ and $\mathbf{y}^*$ are optimal solutions of their corresponding problems if and only if $\mathbf{c}^t\mathbf{x} - \mathbf{b}^t\mathbf{y} = 0$, or equivalently,

$$\sum_{i=1}^{m} u_i + \sum_{j=1}^{n} v_j = 0$$

due to (1.20). Note that all summands on the left of this equality are non-negative, therefore this condition in turn is equivalent to (1.18) and (1.19). □

The following concept makes use of the complementary slackness theorem in computing an optimal solution to the dual program. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a feasible solution of primal problem. A *dual solution* associated with $\mathbf{x}$ is a solution $\mathbf{y} = (y_1, \ldots, y_m)$ of system

$$\begin{aligned} (b_i - a_{i1}x_1 - a_{i2}x_2 - \ldots - a_{in}x_n)y_i &= 0 \qquad (i = 1, \ldots, m) \\ (a_{1j}y_1 + a_{2j}y_2 + \ldots + a_{mj}y_m - c_j)x_j &= 0 \qquad (j = 1, \ldots, n) \end{aligned}$$

If such $\mathbf{y}$ is feasible to the dual problem, then it follows from the complementary slackness theorem that it is also an optimal solution of dual problem, and moreover, the given solution $\mathbf{x}$ is an optimal solution of primal problem. The following small example illustrates this solution approach.

Let us consider the following primal linear program

$$\begin{aligned} \max \quad & x_1 + 2x_2 \\ \text{s.t.} \quad & 2x_1 + 3x_2 \le 12 \\ & 6x_1 + 5x_2 \le 30 \\ & x_2 \le 3 \\ & x_1, x_2 \ge 0 \end{aligned}$$

and $\mathbf{x} = (x_1, x_2) = \left(\frac{3}{2}, 3\right)$, which can be easily checked that it is a feasible solution to this primal program. The dual program is

$$\begin{aligned} \min \quad & 12y_1 + 30y_2 + 3y_3 \\ \text{s.t.} \quad & 2y_1 + 6y_2 \ge 1 \\ & 3y_1 + 5y_2 + y_3 \ge 2 \\ & y_1, y_2, y_3 \ge 0 \end{aligned}$$

Using the complementary slackness theorem, we can construct a dual solution $\mathbf{y} = (y_1, y_2, y_3)$ associated with the primal feasible solution $\mathbf{x}$ as follows.

- Since $x_1 = \frac{3}{2} > 0$, we must have $2y_1 + 6y_2 = 1$.
- Since $x_2 = 3 > 0$, we must have $3y_1 + 5y_2 + y_3 = 2$.

- Since $6x_1 + 5x_2 = 24 < 30$, we must have $y_2 = 0$.

Solving the obtained system of linear equalities on (y_1, y_2, y_3), we obtain $\mathbf{y} = \left(\frac{1}{2}, 0, \frac{1}{2}\right)$. It is not hard to see that this solution is feasible to the dual program. Hence, by the above discussion, we can conclude that the given $\mathbf{x} = \left(\frac{3}{2}, 3\right)$ is an optimal solution to the primal program, and $\mathbf{y} = \left(\frac{1}{2}, 0, \frac{1}{2}\right)$ is an optimal solution to the dual program.

Chapter 2

Formulating integer programs

2.1 What is an integer program?

In many applications, natural indivisibilities of the problem under study are reflected by integrality restrictions. For example, when deciding how many containers can some ship carry or how many trains can be used in a train schedule, fractional solutions clearly are meaningless. In these situations, by the nature of the decision-making problem, the decision variables are inherently integral. That raises the need of considering integer programs.

2.1.1 Formulations

Roughly speaking, an integer program (abbreviated IP) is a problem of optimizing a linear function subject to linear constraints on integer-valued variables. It differs from the linear programs by the restriction on the variable domain, which requires integer values for all variables. An integer program in its general formulation can be described explicitly as follows.

$$
\begin{aligned}
\max(\min) \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n & & \\
\text{s.t.} \quad & a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq b_i & & (i = 1, \ldots, m_1) \\
& a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \geq b_i & & (i = m_1 + 1, \ldots, m_2) \\
& a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = b_i & & (i = m_2 + 1, \ldots, m) \\
& x_j \in \mathbb{Z} & & (j = 1, \ldots, n)
\end{aligned}
$$

in which $x_j (j = 1, \ldots, n)$ are variables, $b_i, c_j, a_{ij} (i = 1, \ldots, m, j = 1, \ldots, m)$ are real-valued parameters. An integer program may also be described in its standard formulation

$$
\begin{aligned}
\max \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
\text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\
& \ldots
\end{aligned}
$$

$$\begin{aligned} a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &\leq b_m \\ x_1, x_2, \ldots, x_n &\in \mathbb{Z}_+ \end{aligned}$$

That is, in the standard formulation of an integer program, the objective is to maximize a linear function, all constraints are in the less-than-or-equal form, and all variables are nonnegative integers. For convenience of description, one often writes an integer program in the matrix form of its standard formulation

$$\begin{aligned} \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}_+^n \end{aligned}$$

in which $\mathbf{x} = (x_1, \ldots, x_n)^t$ is the column vector of decision variables, $\mathbf{c} = (c_1, \ldots, c_n)^t$, $A = (a_{ij}) \in \mathbb{R}^{m\times n}$, $\mathbf{b} = (b_1, \ldots, b_m)^t$. The inequalities in the formulation are componentwise.

A special case of integer program is *binary program* (or more precisely, binary linear program, abbreviated BLP), in which all variables are restricted to obtain binary values. That is, a binary program can be described explicitly in its standard formulation as follows

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\ & \qquad\qquad \ldots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \\ & x_1, x_2, \ldots, x_n \in \{0, 1\} \end{aligned}$$

or in the following matrix form

$$\begin{aligned} \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \{0, 1\}^n \end{aligned}$$

If some but not all variables are integer, we have a *mixed integer program* (abbreviated MIP), which is written in matrix form as follows

$$\begin{aligned} \max \quad & \mathbf{c}^t\mathbf{x} + \mathbf{d}^t\mathbf{y} \\ \text{s.t.} \quad & A\mathbf{x} + B\mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{y} \geq \mathbf{0} \text{ and integer} \end{aligned}$$

2.1.2 Some introductory examples

As in linear programming, translating a problem description into an integer program formulation should be done systematically. One should follow the following modeling framework:

(i) Define what appear to be necessary variables, and what are data of problem instances.

(ii) Use these variables to define the objective function of the problem.

(iii) Use these variables to define problem constraints (so that feasible points correspond to feasible solutions of the problem).

Sometimes, if it is difficult to model a problem by using the current set of variables, one needs to define additional or alternative variables and then iterate the above framework. To illustrate this modeling framework, we present here some introductory examples of translating a problem description into an integer program.

1. Knapsack problem

There is a budget b available for investment in n projects, each project j has outlay a_j and expected return c_j. The goal is to find a portfolio (i.e. a set of projects) of maximum expected return, provided that the budget is not exceeded. This problem can be formulated in form of a binary program as follows.

Decision variables

A portfolio is a complete specification of which projects are chosen to invest and which projects are not. Therefore, the decision variables are

$$x_j = \begin{cases} 1 & \text{if project } j \text{ is chosen to invest,} \\ 0 & \text{otherwise,} \end{cases}$$

where $j = 1, \ldots, n$. This is a set of n binary variables.

Objective function

If we choose project j to invest (i.e., $x_j = 1$), then the expected return from investment to this project is c_j $(= c_j x_j)$. Otherwise, if we do not invest to this project (i.e., $x_j = 0$), then we of course receive no return from it, or in other words, the expected return from this project is 0 $(= c_j x_j)$. Therefore, the contribution of project j into the total expected return is $c_j x_j$. Summing over all projects yields overall expected return of the portfolio. That is, our objective is

$$\max \quad \sum_{j=1}^{n} c_j x_j.$$

Constraint

If we choose project j to invest (i.e., $x_j = 1$), then it takes an outlay of a_j $(= a_j x_j)$ from our budget. Otherwise, if this project is not in our portfolio (i.e., $x_j = 0$), then of course it takes no outlay from our budget, or in other words, the outlay of this project will be 0 $(= a_j x_j)$. Therefore, each project j takes an outlay $a_j x_j$ from the budget. The total outlay of the portfolio must not be exceed the given budget, leading to the following constraint.

$$\sum_{j=1}^{n} a_j x_j \leq b.$$

IP formulation

We come up with the following integer programming formulation of the knapsack problem.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_j x_j \leq b \\ & x_j \in \{0,1\} \qquad\qquad \forall j = 1, \ldots, n \end{aligned}$$

2. Production plan

A company is attempting to decide the mix of products which it should produce next week. It has n products $\{1, \ldots, n\}$, each unit of product j has an outlay a_j, a profit p_j and production time t_j man-hours. The company has H man-hours and a budget B available next week. Assume that no fraction of products is accepted, i.e., the number of units of each product to be produced must be a non-negative integer. The goal is to decide how many units (if any) of each product to produce next week in order to get maximum profit. This problem can be formulated as an integer program as follows.

Decision variables

Since we need to decide how many units of each product to produce, the decision variables are

$$x_j = \text{the number of units of product } j \text{ to produce next week,}$$

where $j = 1, \ldots, n$. Since we are not allowed to have fraction of products, we have $x_j \in \mathbb{Z}_+$ for all products $j = 1, \ldots, n$.

Objective function

The profit from producing x_j units of product j is $p_j x_j$. Summing over all products yields the total profit from the production plan. Hence, our objective is

$$\max \quad \sum_{j=1}^{n} p_j x_j.$$

Constraint

The outlay of producing x_j units of product j is $a_j x_j$. Summing over all products yields the total outlay of the production plan, and this must not exceed the given budget B. Therefore, we have the following constraints

$$\sum_{j=1}^{n} a_j x_j \leq B.$$

The production time of producing x_j units of product j is $t_j x_j$. Summing over all products yields the total production time of the production plan. Since the company has H man-hours available, we must have

$$\sum_{j=1}^{n} t_j x_j \leq H.$$

IP formulation

We come up with the following integer programming formulation of the problem.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_j x_j \leq B \\ & \sum_{j=1}^{n} t_j x_j \leq H \\ & x_j \in \mathbb{Z}_+ \qquad\qquad \forall j = 1, \ldots, n \end{aligned}$$

2.2 Basic modeling techniques

When formulating a problem description as an integer program, integer variables may arise from either logical implications or non-linearities. This section is devoted to analyzing and consolidating some simple techniques to express these implications as linear constraints involving binary variables. The modeling techniques concerning non-linearities will be discussed in the next section.

2.2.1 Implicit 0-1 variables

When we have to determine whether or not to engage in some activity, we must make a *yes-no decision*. The choice is easily modeled by introducing a *binary variable* x, then let $x = 1$ if we engage in the activity and $x = 0$ otherwise. Binary variables are of great importance because they occur regularly in many model formulations.

Suppose that there is a finite set $\mathcal{I}$ of activities, and management has to decide that *at most* k of these activities can be pursued. This situation is easily modeled by introducing binary variables $x_i (i \in \mathcal{I})$ in which $x_i = 1$ if activity i is chosen and $x_i = 0$ otherwise. Then, the following constraint is appropriate:

$$\sum_{i \in \mathcal{I}} x_i \leq k.$$

Similarly, consider the setting in which $x_i (i \in \mathcal{I})$ are binary variables, with $\mathcal{I}$ is a finite index set. Then, the constraint that *exactly* k of these variables are equal to 1 is easily

modeled by

$$\sum_{i \in \mathcal{I}} x_i = k.$$

In the same setting, if we are required to have *at least* k of these variables equal 1, then the following constraint is appropriate:

$$\sum_{i \in \mathcal{I}} x_i \geq k.$$

Example 2.1. (Assignment problem).

There are n people available to carry out n jobs. Each person is assign to carry out exactly one job. There is an estimated cost c_{ij} if person i is assigned to job j, since some people are better suited to some jobs. The assignment problem aims to find an assignment of minimum cost.

We define the following variables

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ does job } j, \\ 0 & \text{otherwise,} \end{cases}$$

with $i, j \in \{1, \ldots, n\}$. The assignment problem can be formulated as the following binary linear program:

$$\min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1, \ldots, n \tag{2.3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j = 1, \ldots, n \tag{2.4}$$

The sum in (2.1) is the cost of the assignment, which is required to be minimized. Constraints (2.2) mean that each person is assigned to carry out exactly one job. Constraints (2.3) ensure that each job is done by exactly one person. The binary domain of the variables is given in (2.4).

Example 2.2. (Set covering problem).

Given a set $M = \{1, \ldots, m\}$ of regions, the problem is to decide where to install a set of emergency service centers from a set $N = \{1, \ldots, n\}$ of potential centers. For each possible center $j \in N$, the cost c_j of installing a service center is known, and the set $S_j \subset M$ of regions it can service is also known. The goal is to choose a minimum cost set of service centers so that all regions are covered.

We construct an incident matrix $A = (a_{ij})_{m\times n}$ in which

$$a_{ij} = \begin{cases} 1 & \text{if } i \in S_j \text{ (i.e., region } i \text{ is serviced by center } j), \\ 0 & \text{otherwise.} \end{cases}$$

This is nothing but processing the data for facilitating the problem description. We define the following variables

$$x_j = \begin{cases} 1 & \text{if center } j \text{ is selected to install an emergency service center}), \\ 0 & \text{otherwise,} \end{cases}$$

with $j \in \{1, \ldots, n\}$. The set covering problem can be formulated as the following binary linear program:

$$\min \quad \sum_{j=1}^{n} c_j x_j \tag{2.5}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad \forall i = 1, \ldots, m \tag{2.6}$$

$$x_j \in \{0, 1\} \qquad \forall j = 1, \ldots, n \tag{2.7}$$

The sum in (2.5) is the cost of installing the selected emergency service centers, that is required to be minimized. Constraints (2.6) guarantee that each region can be serviced by at least one emergency service center. The binary domain of the variables is given in (2.7).

2.2.2 Logical implications involving binary variables

Many technical constraints of practical problems are stated in form of *logical implications*, that are properties expressed as

"If A holds, then B holds."

Here we consider the situations in which A and B are expressed as particular settings of some binary variables, and we want to formulate this implication by way of linear constraints. The following principle gives us a basic way to make such formulation.

Proposition 2.3. (Logical implication principle).
Consider the setting in which we are given

- *a set of binary variables $x_i (i \in \mathcal{I})$ with $\mathcal{I}$ is a finite index set, and*
- *a real-value variable y satisfying $0 \leq y \leq c$, where c is a positive constant.*

The logical implication

"if $x_i = 0$ for all $i \in \mathcal{I}$, then $y = 0$"

is exactly expressed by

$$y \leq c \sum_{i \in \mathcal{I}} x_i. \tag{2.8}$$

Proof. In case that $x_i = 0$ for all $i \in \mathcal{I}$, the right hand side of (2.8) equals 0, hence (2.8) becomes

$$y \leq 0.$$

Since y is non-negative, this means $y = 0$ as desired.

In case that some $x_i = 1$, we have

$$\sum_{i \in \mathcal{I}} x_i \geq 1,$$

and consequently,

$$c \sum_{i \in \mathcal{I}} x_i \geq c.$$

Since $y \leq c$, constraint (2.8) is always satisfied. □

The logical implication principle in Proposition 2.3 can be seen geometrically. For simplicity, let us consider the case in which $\mathcal{I} = \{1, 2\}$ and $c = 1$. In this case, we have the following logical implication

$$\text{“if } x_1 = 0 \text{ and } x_2 = 0, \text{ then } y = 0\text{”}$$

given the binary variables x_1, x_2 and a real-value variable y satisfying $0 \leq y \leq 1$. The set of all possible values of variables (x_1, x_2, y) consists of four line segments

$$\{0, 1\} \times \{0, 1\} \times [0, 1].$$

The logical implication means that we have to exclude $\{(0, 0, y) \mid 0 < y \leq 1\}$ from the above set. The expression (2.8) in this case becomes

$$y \leq x_1 + x_2.$$

As visualized in Figure 2.1, the halfspace defined by this inequality nicely cuts off the set to be excluded, and the remaining feasible set consists of three vertical thick line segments and the origin. It does so in the best way possible, since the inequality defines a facet of the remaining feasible set.

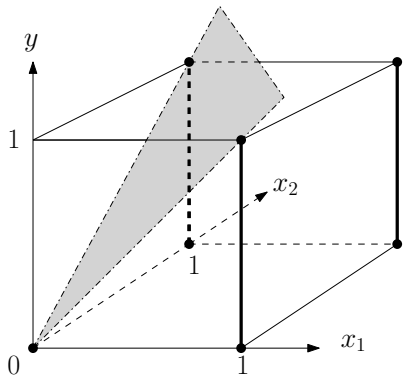


Figure 2.1: Feasible set (thick black) and cutting plane.

The logical implication principle mentioned in Proposition 2.3 is more widely valid by using complementarity in sense of the following proposition.

Proposition 2.4.
Consider the setting in which we are given

- *a set of binary variables* $x_i (i \in \mathcal{I})$ *with* $\mathcal{I}$ *is a finite index set,*
- $\mathcal{I} = \mathcal{I}_0 \cup \mathcal{I}_1$ *is the disjoint union of two subsets, and*
- *a real-value variable* y *satisfying* $0 \leq y \leq 1$.

The logical implication

$$\text{“if } x_i = 0 \text{ for all } i \in \mathcal{I}_0 \text{ and } x_j = 1 \text{ for all } j \in \mathcal{I}_1, \text{ then } y = 0\text{”}$$

is exactly expressed by

$$y \leq \sum_{i \in \mathcal{I}_0} x_i + \sum_{j \in \mathcal{I}_1} (1 - x_j). \tag{2.9}$$

Proof. This is a direct application of the logical implication principle (2.8) to the binary variables $x_i (i \in \mathcal{I}_0)$ and $1 - x_j (j \in \mathcal{I}_1)$. □

Note that we often see (2.9) in the following reformulated form, which is more standard looking version but much less easy to interpret correctly.

$$\sum_{j \in \mathcal{I}_1} x_j - \sum_{i \in \mathcal{I}_0} x_i + y \leq |\mathcal{I}_1|.$$

The following example gives an application of the logical implication principles above.

Example 2.5. (n-queen problem).

The n-queen problem is to place n queens on a $n \times n$ chessboard so that no two queens are on the same row, column, or diagonal. To formulate this problem as an IP, we introduce binary variables

$$x_{ij} = \begin{cases} 1 & \text{if a queen is placed at cell of row } i \text{ column } j, \\ 0 & \text{otherwise.} \end{cases}$$

Instead of finding a solution of placing n queens on the chessboard, we will find a solution of maximum number of queens and observe that the optimal objective value will coincide n. For that, with each square we compute in advance which other squares are blocked if a queen is placed on this particular square:

$$T(i,j) = \{(h,k) \in \{1,\ldots,n\}^2 \mid h = i \text{ or } k = j \text{ or } |h-i| = |k-j|\} \backslash \{(i,j)\}.$$

The set $T(i,j)$ for each cell (i,j) is computed beforehand in a pre-processing step. Figure 2.2 represents a chosen cell (in blue color) in an 8×8 chessboard, and its corresponding set $T(i,j)$ which consists of cells in other colors except for white.

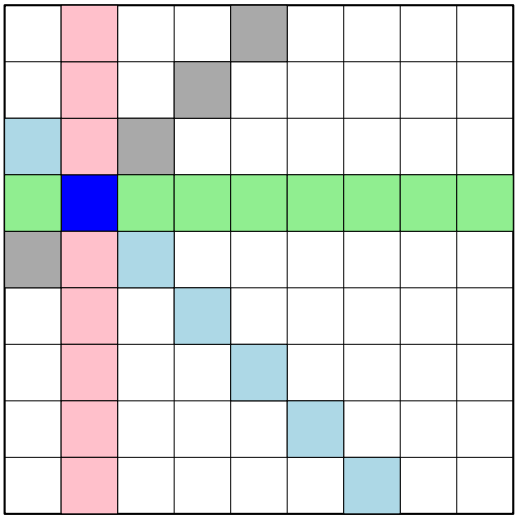


Figure 2.2: Feasible set (thick black) and cutting plane.

Using the binary variables x_{ij} indicating the positions of the queens, the objective of the problem is

$$\max \sum_{i,j=1}^{n} x_{ij}$$

subject to the constraints that hold for all cell $(i,j) \in \{1,\ldots,n\}^2$:

"If a queen is placed at cell (i,j),
then no queen can be placed in any cell of $T(i,j)$".

The logical implication corresponds to cell $(i,j) \in \{1,\ldots,n\}^2$ can be expressed as follows:

$$\text{"If } x_{ij} = 1, \text{ then } \sum_{(h,k)\in T(i,j)} x_{hk} = 0\text{"}.$$

Keeping in mind that $0 \leq \sum_{(h,k)\in T(i,j)} x_{hk} \leq |T(i,j)|$, this can be linearly represented by

$$\sum_{(h,k)\in T(i,j)} x_{hk} \leq |T(i,j)|(1 - x_{ij}).$$

2.2.3 Logical implications involving integer variables

In this part we consider the logical implications of the form

"if A holds, then B holds"

in which A and B are expressed as particular settings of some integer variables. In spirit of this chapter, we want to formulate such implications as linear constraints. A standard

way to represent a non-negative variable $x \in \mathbb{N}$ whose value is limited by some known bound c_x is to use its binary decomposition

$$x = \sum_{j=0}^{\lfloor \log_2 c_x \rfloor} 2^j x_j$$

where $x_j \in \{0, 1\}$ represents the j^{th} binary digit of x. However, this representation does not allow to formulate implications like

"if $1 \leq x \leq 3$ and $y \in \{1, 2, 5\}$, then $z = 2$ or $4 \leq z \leq 7$."

This becomes easier if we use binarizing approach. The key idea of this approach is to introduce, for each integer variable x whose value is limited below by some known bound l_x and limited above by some known bound u_x, binary variables $x_i (i = l_x, \ldots, u_x)$, in which x_i answers the question "Is $x = i$?". Any property like $x \in C \subset \{l_x, \ldots, u_x\}$ is then represented by the implicit binary variable $\sum_{i \in C} x_i$. Hence, we can reformulate the given implication using the introduced binary variables, and afterward express the given implication as linear constraints on these binary variables. To illustrate this process, the above implication may be reformulated as

$$\text{"if } \sum_{i=1}^{2} x_i = 1 \text{ and } y_1 + y_2 + y_5 = 1, \text{ then } z_2 + \sum_{i=4}^{7} z_i = 1\text{"},$$

which is immediately formulated as the following linear constraint

$$\left(1 - \sum_{i=1}^{3} x_i\right) + (1 - (y_1 + y_2 + y_5)) \geq 1 - \left(z_2 + \sum_{i=4}^{7} z_i\right).$$

2.3 Advanced modeling techniques

In this section, we analyze and consolidate some simple techniques to express non-linearity as linear constraints involving integer variables.

2.3.1 Linearization of quadratic binary variables

If quadratic terms like $x_i x_j$ with $x_i, x_j \in \{0, 1\}$ are involved in the formulation of our interested problem, we can obtain a linear equivalent description by introducing new binary variables y_{ij} in spirit of

$$y_{ij} = x_i x_j.$$

Since all related variables in this equality are binary, this means exactly

"$y_{ij} = 1$ if and only if $x_i = 1$ and $x_j = 1$.

This in turn can be described equivalently by the following three logical implications:

$$\begin{aligned}&\text{if } y_{ij} = 1, \text{ then } x_i = 1,\\ &\text{if } y_{ij} = 1, \text{ then } x_j = 1,\\ &\text{if } x_i = 1 \text{ and } x_j = 1, \text{ then } y_{ij} = 1.\end{aligned}$$

By applying the logical implication principle in Proposition 2.4, these logical implications can be respectively formulated as the following linear constraints:

$$\begin{aligned}1 - x_i &\le y_{ij}\\ 1 - x_j &\le y_{ij}\\ 1 - y_{ij} &\le 1 - x_i + 1 - x_j\end{aligned}$$

or in more usual form

$$\begin{aligned}y_{ij} - x_i &\le 0\\ y_{ij} - x_j &\le 0\\ x_i + x_j - y_{ij} &\le 1\end{aligned}$$

Figure 2.3 shows the geometric behind this linearization of the quadratic binary expression. All possible values of the triplet of binary variables (x_i, x_j, y_{ij}) satisfying quadratic relationship $y_{ij} = x_i x_j$ are

$$(0,0,0), (0,1,0), (1,0,0), (1,1,1).$$

The three inequalities in the linearization form above constitute exactly the nontrivial facets of the convex hull of these points. The remaining (trivial) facet of this polytope is $y_{ij} \ge 0$. Therefore, this linearization form is the tightest one for relationship.

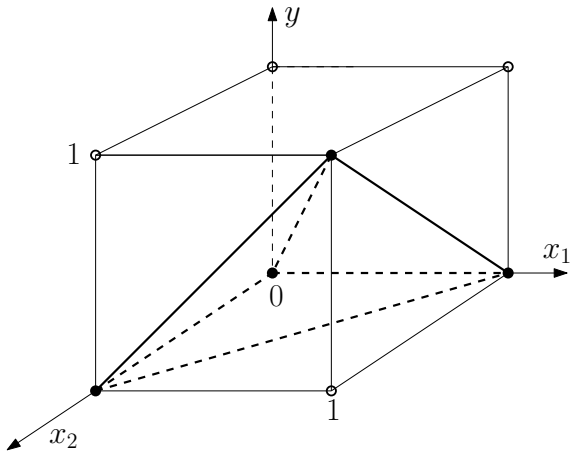


Figure 2.3: Quadratic binary polytope.

The following example shows an application of this linearization technique.

Example 2.6. (Maximal clique).

Given an undirected graph $G = (V, E)$, a clique of G is a complete subgraph. We want to find a maximal clique of G, i.e. a clique of maximum number of vertices. To formulate this problem as an integer program, we number the vertices in V by $1, \ldots, n$, and for each vertex $i \in V$ we introduce a binary variable

$$x_i = \begin{cases} 1 & \text{if vertex } i \text{ is chosen to the clique,} \\ 0 & \text{otherwise.} \end{cases}$$

To recognize which edges of G belong to the desired clique, for each edge $i, j \in E$ we introduce a binary variable

$$y_{ij} = \begin{cases} 1 & \text{if edge } \{i, j\} \text{ is chosen to the clique,} \\ 0 & \text{otherwise.} \end{cases}$$

Then the number of vertices in our chosen clique is

$$\sum_{i=1}^{n} x_i$$

which should be maximized. To be a clique, the subgraph must satisfy the following requirements.

- If $\{i, j\}$ is not an edge of G, then i and j cannot be chosen simultaneously. This means the following implication:

  "For $\{i, j\} \notin E$, if $x_i = 1$, then $x_j = 0$"

  which can be represented as

  $$x_j \leq 1 - x_i \quad \forall \{i, j\} \notin E,$$

  or in more usual form

  $$x_i + x_j \leq 1 \quad \forall \{i, j\} \notin E.$$

- If $\{i, j\}$ is an edge of G, then this edge is chosen to the clique if and only if both its vertices were chosen. This means the following implication:

  "For $\{i, j\} \in E$, $y_{ij} = 1$ if and only if $x_i = 1$ and $x_j = 1$."

  As discussed above, this implication can be represented by

  $$\begin{aligned} y_{ij} - x_i &\leq 0 && \forall \{i, j\} \in E \\ y_{ij} - x_j &\leq 0 && \forall \{i, j\} \in E \\ x_i + x_j - y_{ij} &\leq 1 && \forall \{i, j\} \in E \end{aligned}$$

2.3.2 Linearization of polynomials of binary variables

Polynomials of binary variables appears in zero-one polynomial programming problems, that belong to a special subclass of integer programming. These problems ask

$$\begin{aligned} \min \mid \max \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 \quad \forall i = 1, \ldots, m \\ \mathbf{x} = (x_1, \ldots, x_n) & \in \{0,1\}^n \end{aligned}$$

where $f(\mathbf{x})$ and $g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})$ are polynomials. Since the polynomials are nonlinear, a way to approach such problems is to transform into binary linear programs. The key idea in this transformation is to linearize a polynomial function of binary variables. For that we consider a polynomial function $f(\mathbf{x})$ of binary variables $\mathbf{x} = (x_1, \ldots, x_n)$. Note that for every binary variable x_j we have $x_j^k = x_j$ with k is an arbitrary positive integer. Furthermore, the product of binary variables equals 1 if and only if all of the related binary variables obtain value 1. Therefore we can apply the following framework to linearize $f(\mathbf{x})$.

- Replace each x_j^k by x_j, and then
- replace each $\prod_{j \in Q} x_j$ by a new variable x_Q satisfying

  "$x_Q = 1$ if and only if $x_j = 1 \ \forall j \in Q$", and
  "$x_Q = 0$ if at least one $x_j = 0$".

Now, to express these logical implications as linear constraints, we can use one of the following representations.

- *Representation 1:* $x_Q \in \{0,1\}$ satisfying

$$\begin{aligned} \sum_{i \in Q} x_j &\leq x_Q + |Q| - 1 \\ |Q| x_Q &\leq \sum_{i \in Q} x_j. \end{aligned}$$

  To illustrate this representation, let us consider an example in which $f(x) = x_1^2 x_2^3 x_3$ with $x_1, x_2, x_3 \in \{0,1\}$. Using this representation, we can replace

$$f(\mathbf{x}) = x_1 x_2 x_3 =: x_*$$

  with x_* satisfies the following linear constraints:

$$\begin{aligned} x_1 + x_2 + x_3 &\leq x_* + 2 \\ 3x_* &\leq x_1 + x_2 + x_3. \end{aligned}$$

- *Representation 2:* $x_Q \geq 0$ satisfying

$$\sum_{i \in Q} x_j \leq x_Q + |Q| - 1$$
$$x_Q \leq x_j \ \forall j \in Q.$$

To illustrate this representation, let us consider the same example as above. Using this representation, we can replace

$$f(\mathbf{x}) = x_1 x_2 x_3 =: x_*$$

with

$$x_1 + x_2 + x_3 \leq x_* + 2$$
$$x_* \leq x_1$$
$$x_* \leq x_2$$
$$x_* \leq x_3.$$

- *Representation 3:* $x_Q \geq 0$ satisfying

$$\sum_{i \in Q} x_j \leq x_Q + |Q| - 1$$
$$\sum_{Q \in \mathcal{S}_j} x_Q \leq |\mathcal{S}_j| x_j \quad \forall j \in Q.$$

where $\mathcal{S}_j$ is the collection of sets Q that contain j. To illustrate this representation, let us consider an example in which $f(x) = x_1^2 x_2^3 x_3 + x_1 x_3^2 x_4^3$ with $x_1, x_2, x_3, x_4 \in \{0, 1\}$. Using this representation, we can express

$$f(\mathbf{x}) = x_1 x_2 x_3 + x_1 x_3 x_4 = x_{123} + x_{134}$$

subject to

$$x_1 + x_2 + x_3 \leq x_{123} + 2$$
$$x_1 + x_3 + x_4 \leq x_{134} + 2$$
$$x_{123} + x_{134} \leq 2x_1$$
$$x_{123} \leq x_2$$
$$x_{123} + x_{134} \leq 2x_3$$
$$x_{124} \leq x_4.$$

2.3.3 Linearization of piecewise linear functions

Suppose that we have a piecewise linear function $f(x)$ specified by the points $\{(a_i, f(a_i))\}$ for $i = 1, \ldots, r$. That means $f(x)$ is linear in each segment $[a_i, a_{i+1}]$. Note that an arbitrary continuous function of one variable can be approximated by a piecewise linear

function, with the quality of the approximation being controlled by the size of the linear segments. Figure 2.4 illustrates a piecewise linear function specified by 5 points.

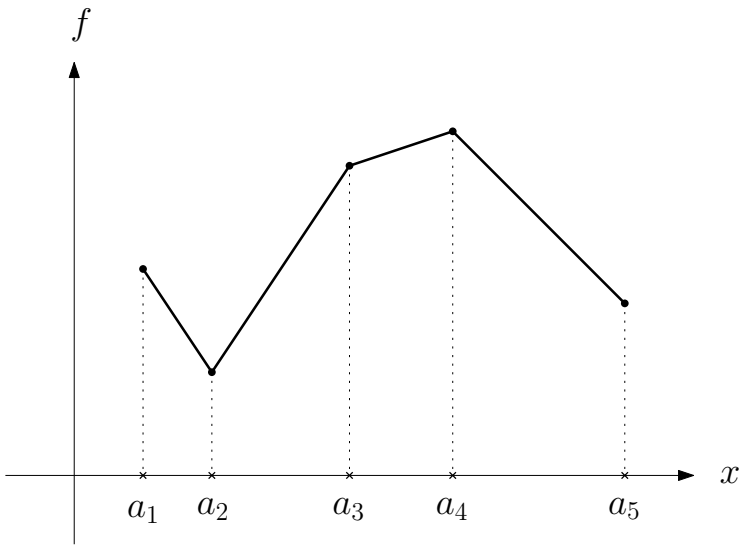


Figure 2.4: A piecewise linear function.

Given the setting of $f(x)$, any $x \in [a_1, a_r]$ can be expressed as a convex combination of $a_1, \ldots, a_r$, or more precisely,

$$x = \lambda_1 a_1 + \ldots + \lambda_r a_r \tag{2.10}$$

with

$$\lambda_1 + \ldots + \lambda_r = 1$$
$$\lambda = (\lambda_1, \ldots, \lambda_r) \in \mathbb{R}^r_+.$$

The coefficients of the convex combination (2.10) are not unique. However, if $x \in [a_i, a_{i+1}]$ and λ is chosen so that $x = \lambda_i a_i + \lambda_{i+1} a_{i+1}$ and $\lambda_i + \lambda_{i+1}$, then we obtain

$$f(x) = \lambda_i f(a_i) + \lambda_{i+1} f(a_{i+1}).$$

So we have a linear representation

$$f(x) = \lambda_1 f(a_1) + \ldots + \lambda_r f(a_r)$$

with $\lambda_1, \ldots, \lambda_r \geq 0$ satisfying $\lambda_1 + \ldots + \lambda_r = 1$, at most two of the coefficients are positive, and the positive coefficients have consecutive indices. This condition can be modeled by introducing binary variables

$$y_i = \begin{cases} 1 & \text{if } x \in [a_i, a_{i+1}] \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, r-1)$$

and constraints

$$\begin{aligned} &\lambda_1 \le y_1 \\ &\lambda_i \le y_{i-1} + y_i \qquad (i = 2, \ldots, r-1) \\ &\lambda_r \le y_{r-1} \\ &\sum_{i=1}^{r-1} y_i = 1. \end{aligned}$$

These constraints ensure that if $y_i = 1$, then $\lambda_j = 0$ for $j \notin \{i, i+1\}$.

2.3.4 Disjunctive constraints

A disjunctive set of constraints aims to find a feasible solution satisfying at least k of m sets of linear constraints. Disjunctive constraints arise naturally in many models. A simple example is when we need to define a variable that is equal to the minimum of two other variables:

$$y = \min\{x_1, x_2\}.$$

This can be done with the two constraints

$$y \le x_1 \quad \text{and} \quad y \le x_2,$$

and the following disjunctive set of constraints

$$y \ge x_1 \quad \text{or} \quad y \ge x_2.$$

To illustrate the technique of transforming a disjunctive set of linear constraints into a system of linear constraints, let us consider the following problem

"Find $\mathbf{x} \in \mathbb{R}^n$ satisfying $\mathbf{0} \le \mathbf{x} \le \mathbf{u}$ and $\mathbf{a}^t\mathbf{x} \le b$ or $\mathbf{c}^t\mathbf{x} \le d$".

Let

$$M = \max_{\mathbf{0} \le \mathbf{x} \le \mathbf{u}} \{\mathbf{a}^t\mathbf{x} - b, \mathbf{c}^t\mathbf{x} - d\}.$$

Since $\mathbf{x}$ belongs to a compact set, that M exists finitely. Now we introduce two binary variables y_1, y_2 with the following meaning

$$y_1 = \begin{cases} 1 & \text{if } \mathbf{a}^t\mathbf{x} \le b, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_2 = \begin{cases} 1 & \text{if } \mathbf{c}^t\mathbf{x} \le d, \\ 0 & \text{otherwise.} \end{cases}$$

Then the following linear system expresses the disjunctive constraints we are considering.

$$\mathbf{a}^t\mathbf{x} - b \le My_1$$

$$\begin{aligned}
\mathbf{c}^t\mathbf{x} - d &\leq My_2 \\
y_1 + y_2 &\geq 1 \\
\mathbf{x} &\leq \mathbf{u} \\
\mathbf{x} &\in \mathbb{R}^n_+.
\end{aligned}$$

This follows since $y_1 = 1$ yields the constraint $\mathbf{a}^t\mathbf{x} \leq b$, while $y_1 = 0$ yields the redundant constraint $\mathbf{a}^t\mathbf{x} \leq b + M$. Similarly, $y_2 = 1$ yields the constraint $\mathbf{c}^t\mathbf{x} \leq d$, while $y_2 = 0$ yields the redundant constraint $\mathbf{c}^t\mathbf{x} \leq d + M$. The condition $y_1 + y_2 \geq 1$ ensures that at least one of the constraints $\mathbf{a}^t\mathbf{x} \leq b$ and $\mathbf{c}^t\mathbf{x} \leq d$ holds.

Now we consider a more general problem. Suppose $P_i = \{\mathbf{x} \in \mathbb{R}^n_+ \mid A^i\mathbf{x} \leq \mathbf{b}^i\}$ for $i = 1, \ldots, m$, with $\mathbf{0} \leq x \leq \mathbf{u}$, and the disjunctive constraint is to find a feasible $\mathbf{x}$ contained in at least k of the sets $P_1, \ldots, P_m$. Similar to the modeling process above, we introduce binary variables $y_1, \ldots, y_m$ in which each binary variable y_i indicates that $\mathbf{x} \in P_i$. Note that there exists a vector $\mathbf{w}$ such that $A^i\mathbf{x} - \mathbf{b}^i \leq \mathbf{w}$ for all $i = 1, \ldots, m$ and for all $\mathbf{x} \in [\mathbf{0}, \mathbf{u}]$. By similar arguments as above, the disjunctive constraint is equivalently expressed by

$$\begin{aligned}
A^i\mathbf{x} &\leq \mathbf{b}^i + \mathbf{w}(1 - y_i) \quad \forall i = 1, \ldots, m \\
\sum_{i=1}^{m} y_i &\geq k \\
\mathbf{x} &\leq \mathbf{u} \\
\mathbf{x} &\in \mathbb{R}^n_+ \\
y_i &\in \{0, 1\} \quad \forall i = 1, \ldots, m.
\end{aligned}$$

In case of $k = 1$, an alternative formulation is

$$\begin{aligned}
A^i\mathbf{x}^i &\leq y_i\mathbf{b}^i && \forall i = 1, \ldots, m \\
\mathbf{x}^i &\leq y_i\mathbf{u} && \forall i = 1, \ldots, m \\
\sum_{i=1}^{m} y_i &= 1 \\
\sum_{i=1}^{m} \mathbf{x}^i &= \mathbf{x} \\
y_i &\in \{0, 1\} && \forall i = 1, \ldots, m \\
\mathbf{x}^i &\in \mathbb{R}^n_+ \\
\mathbf{x} &\in \mathbb{R}^n_+.
\end{aligned}$$

Indeed, given that $\mathbf{x} \in \cup_{i=1}^m P_i$, without loss of generality we can suppose that $\mathbf{x} \in P_1$. Then a solution to the above system is $y_1 = 1$, $y_i = 0 (i = 2, \ldots, m)$, $\mathbf{x}^1 = \mathbf{x}$, and $\mathbf{x}^i = \mathbf{0} (i = 2, \ldots, m)$. Conversely, suppose that the above system has a solution. Without loss of generality, we can assume in that solution $y_1 = 1$, and hence $y_i = 0 (i = 2, \ldots, m)$.

Then we obtain $\mathbf{x}^i = \mathbf{0}$ for $i = 2, \ldots, m$ and $\mathbf{x}^1 = \mathbf{x}$. Since $A^1\mathbf{x}^1 \leq y_1\mathbf{b}^1 = \mathbf{b}^1$, we have $\mathbf{x} = \mathbf{x}^1 \in P_1$, which means $\cup_{i=1}^m P_i \neq \emptyset$.

The following example shows an application of these linearization techniques for disjunctive constraints.

Example 2.7. (Sudoku).

Sudoku is a famous puzzle. In its well-known format, a Sudoku puzzle can be described as follows. Given a grid consisting of 81 cells arranged in 9 rows and 9 columns, which can also be viewed as a composition of nine 3×3 blocks. Players must follow the following puzzle rules:

(S1) Each cell of grid must be filled by exactly one digit from 1 to 9.

(S2) Each digit from 1 to 9 appears exactly once in each row of the grid.

(S3) Each digit from 1 to 9 appears exactly once in each column of the grid.

(S4) Each digit from 1 to 9 appears exactly once in each of the 3×3 blocks of the grid.

A Sudoku puzzle is usually provided with a partially complete grid, and the objective is to completely fill the grid regarding the puzzle rules. Often a well-posed Sudoku puzzle is given (in the sense that it has exactly one solution). Figure 2.5 gives an example of a Sudoku puzzle and a solution for the puzzle.

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

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

Figure 2.5: A Sudoku puzzle (left) and a solution for it (right).

We equip a unique coordinate system to each Sudoku puzzle so that the coordinate of the top left cell of the grid is $(1, 1)$, and the one of the bottom right cell is $(9, 9)$. Furthermore, we number the 3×3 blocks of Sudoku grid from left to right and from top to bottom, so the top left 3×3 block is numbered by 1, and the right bottom 3×3 subgrid

is numbered by 9. The coordinate setting is illustrated in Figure 2.6. For convenience, we denote $S := \{1, \ldots, 9\}$, and let

$$B := \{(1,1), (1,4), (1,7), (4,1), (4,4), (4,7), (7,1), (7,4), (7,7)\}$$

be the set of coordinates of the top left cells of the blocks.

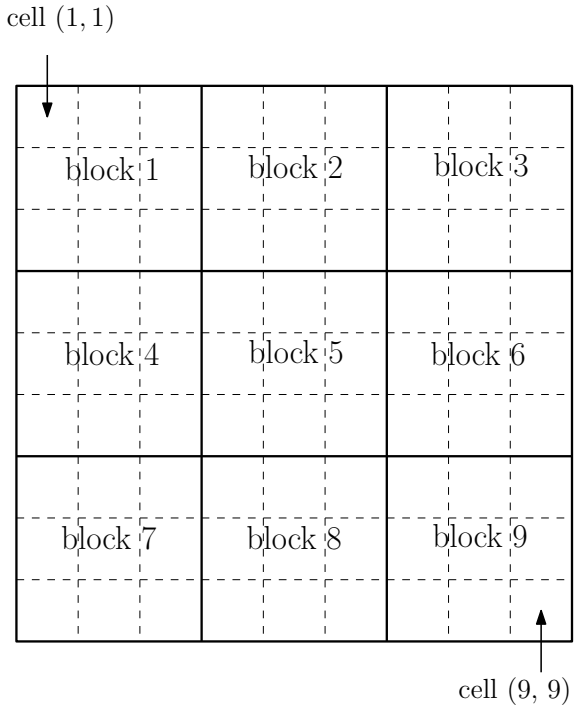


Figure 2.6: Coordinate setting for Sudoku puzzles.

The Sudoku problem can be stated using constraint programming as a collection of *all-different* constraints. This formulation uses the integer variables $x_{ij} (i, j \in S)$ with values in S. The value of x_{ij} is what we fill in the cell (i, j) of the grid in the solution. Let F be the set of indices (i, j) of the already-filled cells of the grid. For each cell $(i, j) \in F$, let $g(i, j)$ be the value that is already given inside that cell. Then a solution to the Sudoku puzzle can be found by finding a feasible solution to the following (nonlinear) integer program

$$(SUDOKU - NLIP) \qquad \min \sum_{i,j \in S} 0 \cdot x_{ij} \tag{2.11}$$

$$\text{subject to} \qquad x_{ij} = g(i,j) \qquad \forall (i,j) \in F \tag{2.12}$$

$$|x_{ij} - x_{ik}| \geq 1 \qquad \forall i, j, k \in S, j < k \tag{2.13}$$

$$|x_{ij} - x_{kj}| \geq 1 \qquad \forall i, j, k \in S, i < k \tag{2.14}$$

$$|x_{p+i,q+j} - x_{p+k,q+l}| \geq 1 \qquad \forall (p,q) \in B,$$
$$i, j, k, l \in \{0, 1, 2\} : 3i + j < 3k + l \tag{2.15}$$

$$x_{ij} \in S \qquad \forall i, j \in S \tag{2.16}$$

The already-filled cells in the grid are taken care by constraints (2.12). Keeping in mind that by constraints (2.16) the used variables are integers, constraints (2.13) mean that different cells of the same row of the grid have different assigned values. Since there are 9 cells in each row of the grid and also 9 values are possible to assign to these cells, constraints (2.13) ensure that each element of S appears exactly once in each row of the grid. Similarly, constraints (2.14) ensure that each element of S appears exactly once in each column of the grid, and constraints (2.15) ensure that each element of S appears exactly once in each subgrid. We use objective function (2.11) for the manner of solving Sudoku problems by using IP solvers, although the objective function is in fact not needed.

The nonlinearity of $(NLIP)$ is due to the absolute terms in the left hand sides of constraints (2.13)-(2.15). Due to (2.16), the values of these absolute terms are at most 8. The next step is to linearize these nonlinear constraints. Note that each of such constraints is of the form $|y| \geq 1$, and we know furthermore that y is an integer variable and $|y| \leq 8$.

In order to linearize such constraint, we introduce two additional non-negative variables y^+ and y^- to hold the positive and negative parts of y. Furthermore, two additional binary variables a^+ and a^- are introduced as indicators for whether y is positive or negative. Detail description of these additional variables is as follows.

$$y^+ = \begin{cases} y & \text{if } y > 0, \\ 0 & \text{if } y \leq 0, \end{cases} \qquad y^- = \begin{cases} 0 & \text{if } y \geq 0, \\ -y & \text{if } y < 0, \end{cases}$$

$$a^+ = \begin{cases} 1 & \text{if } y > 0, \\ 0 & \text{if } y \leq 0, \end{cases} \qquad a^- = \begin{cases} 1 & \text{if } y < 0, \\ 0 & \text{if } y \geq 0. \end{cases}$$

Since $|y| = y^+ + y^-$, our considering nonlinear constraint

$$1 \leq |y| \leq 8, y \in \mathbb{Z}$$

can be linearized by the following integer program

$$\begin{aligned} y^+ - y^- &= y \\ y^+ + y^- &\geq 1 \\ a^+ \leq y^+ &\leq 8a^+ \\ a^- \leq y^- &\leq 8a^- \\ a^+ + a^- &= 1 \\ y^+, y^- &\in \mathbb{Z} \\ a^+, a^- &\in \{0, 1\} \end{aligned}$$

Thus, each nonlinear constraint in (2.13)-(2.15) can be linearized by using 4 additional variables and 7 linear constraints. We come up with the following integer programming model for the Sudoku program, in which for convenience we denote $P := \{(i, j, k, l) \mid i, j, k, l \in \{0, 1, 2\} : 3i + j < 3k + l\}$.

$$(SUDOKU - IP) \qquad \min \sum_{i,j \in S} 0 \cdot x_{ij} \tag{2.17}$$

$$
\begin{aligned}
\text{s.t.} \quad x_{ij} &= g(i,j) && \forall (i,j) \in F && (2.18)\\
x_{ij} - x_{ik} &= u^+_{ijk} - u^-_{ijk} && \forall i,j,k \in S, j<k && (2.19)\\
u^+_{ijk} + u^-_{ijk} &\geq 1 && \forall i,j,k \in S, j<k && (2.20)\\
u^+_{ijk} &\geq b^+_{ijk} && \forall i,j,k \in S, j<k && (2.21)\\
u^+_{ijk} &\leq 8b^+_{ijk} && \forall i,j,k \in S, j<k && (2.22)\\
u^-_{ijk} &\geq b^-_{ijk} && \forall i,j,k \in S, j<k && (2.23)\\
u^-_{ijk} &\leq 8b^-_{ijk} && \forall i,j,k \in S, j<k && (2.24)\\
b^+_{ijk} + b^-_{ijk} &= 1 && \forall i,j,k \in S, j<k && (2.25)\\
u^+_{ijk}, u^-_{ijk} &\in \mathbb{Z} && \forall i,j,k \in S, j<k && (2.26)\\
b^+_{ijk}, b^-_{ijk} &\in \{0,1\} && \forall i,j,k \in S, j<k && (2.27)\\
x_{ij} - x_{kj} &= v^+_{ijk} - v^-_{ijk} && \forall i,j,k \in S, i<k && (2.28)\\
v^+_{ijk} + v^-_{ijk} &\geq 1 && \forall i,j,k \in S, i<k && (2.29)\\
v^+_{ijk} &\geq c^+_{ijk} && \forall i,j,k \in S, i<k && (2.30)\\
v^+_{ijk} &\leq 8c^+_{ijk} && \forall i,j,k \in S, i<k && (2.31)\\
v^-_{ijk} &\geq c^-_{ijk} && \forall i,j,k \in S, i<k && (2.32)\\
v^-_{ijk} &\leq 8c^-_{ijk} && \forall i,j,k \in S, i<k && (2.33)\\
c^+_{ijk} + c^-_{ijk} &= 1 && \forall i,j,k \in S, i<k && (2.34)\\
v^+_{ijk}, v^-_{ijk} &\in \mathbb{Z} && \forall i,j,k \in S, i<k && (2.35)\\
c^+_{ijk}, c^-_{ijk} &\in \{0,1\} && \forall i,j,k \in S, i<k && (2.36)\\
x_{p+i,q+j} - x_{p+k,q+l} &= w^+_{ijkl} - w^-_{ijkl} && \forall (p,q) \in B, (i,j,k,l) \in P && (2.37)\\
w^+_{ijkl} + w^-_{ijkl} &\geq 1 && \forall (p,q) \in B, (i,j,k,l) \in P && (2.38)\\
w^+_{ijkl} &\geq d^+_{ijkl} && \forall (p,q) \in B, (i,j,k,l) \in P && (2.39)\\
w^+_{ijkl} &\leq 8d^+_{ijkl} && \forall (p,q) \in B, (i,j,k,l) \in P && (2.40)\\
w^-_{ijkl} &\geq d^-_{ijkl} && \forall (p,q) \in B, (i,j,k,l) \in P && (2.41)\\
w^-_{ijkl} &\leq 8d^-_{ijkl} && \forall (p,q) \in B, (i,j,k,l) \in P && (2.42)\\
d^+_{ijkl} + d^-_{ijkl} &= 1 && \forall (p,q) \in B, (i,j,k,l) \in P && (2.43)\\
w^+_{ijkl}, w^-_{ijkl} &\in \mathbb{Z} && \forall (p,q) \in B, (i,j,k,l) \in P && (2.44)\\
d^+_{ijkl}, d^-_{ijkl} &\in \{0,1\} && \forall (p,q) \in B, (i,j,k,l) \in P && (2.45)\\
x_{ij} &\in S && \forall i,j \in S && (2.46)
\end{aligned}
$$

Constraints (2.19)-(2.27) are for linearizing constraints (2.13). Similarly, constraints (2.28)-(2.36) correspond to constraints (2.14), while constraints (2.37)-(2.45) linearize constraints (2.15).

2.4 Good and ideal formulations

As we have shown in the previous sections, most integer programming problems can be formulated in several ways. From both theoretical and practical points of view:

> *In integer (and mixed integer) programming, formulating a "good" model is of crucial importance to solving the model.*

What does 'a good model' means? We now define it precisely for the general case of mixed integer programs.

Recall that a mixed integer program (MIP) is an optimization problem of the form

$$\begin{aligned} \max \quad & \mathbf{c}^t\mathbf{x} \\ \text{s.t.} \quad & \mathbf{x} \in X \subset \mathbb{Z}^n \times \mathbb{R}^p, \end{aligned}$$

where $X = \{\mathbf{x} \in \mathbb{R}^{n+p} \mid A\mathbf{x} \leq \mathbf{b}\} \cap (\mathbb{Z}^n \times \mathbb{R}^p)$, for some matrix A and column vector $\mathbf{b}$. The set $P = \{\mathbf{x} \in \mathbb{R}^{n+p} \mid A\mathbf{x} \leq \mathbf{b}\}$ forms a polyhedron, and it is called a *formulation* of the set X (and of the program). A MIP (or a set) may have many formulations. To illustrates this claim, take

$$X = \{(1,1), (2,1), (3,1), (1,2), (2,2), (3,2), (2,3)\}.$$

as an example. Figure 2.7 visualizes three different formulations for this set, including P_1 (the polygon with blue boundary), P_2 (the polygon with red boundary), and P_3 (the polygon with black boundary).

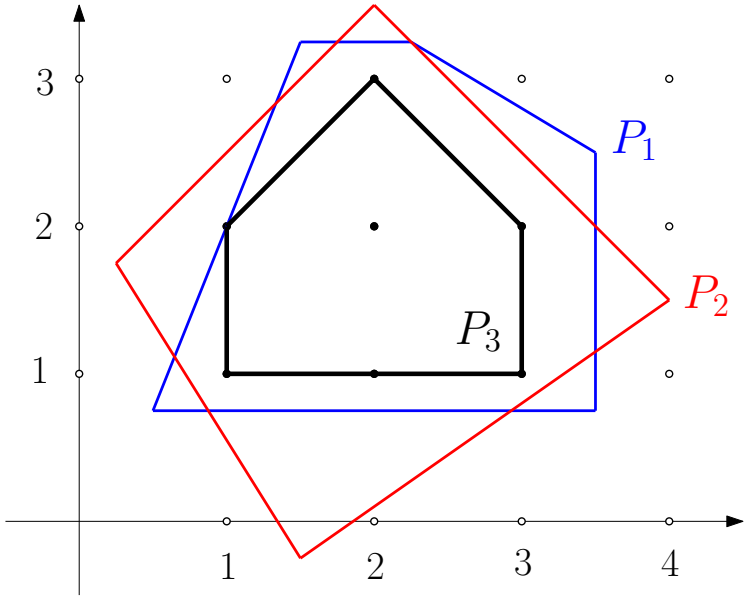


Figure 2.7: Three different formulations for a set of integer points.

Geometrically, for a set or a MIP, we can see that there must be an infinite number of formulations. How we can choose between them? The geometry again helps us to make our choice.

Definition 2.8. (Good formulation).
Given a set $X \subseteq \mathbb{R}^k$ and two formulations P, Q for X. We say that formulation P is better (or stronger) than formulation Q if $P \subsetneq Q$.

Back to the example illustrated in Figure 2.7, formulation P_3 is better than P_1 and P_2. However, we cannot say which one is better between the two formulations P_1 and P_2. Furthermore, we can see that formulation P_3 is better than any possible formulation, since it is the convex hull of the set X and therefore is contained in any convex polyhedron containing X. Formulation P_3 has another special property: all of its vertices are integral points, and hence if we solve a linear program over P_3, then optimal solution is integral. In that sense, P_3 is an *ideal formulation*. We define more precisely this concept in the context of mixed integer programs as follows.

Definition 2.9. (Ideal formulation).
Let P be a formulation of a set $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$. We say that P is ideal if it is better than any of other formulations of X, and the first n components of all of extreme points of P are integer.

We give an example for these concepts in case of integer programs.

Example 2.10. (Ideal formulation for an 0-1 Knapsack set).

Consider

$$X = \{(0,0,0,0), (1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1), (0,1,0,1), (0,0,1,1)\}.$$

Let

$$\begin{aligned}
P_1 &= \{x \in \mathbb{R}^4 \mid 0 \leq x_1, x_2, x_3, x_4 \leq 1, 83x_1 + 61x_2 + 49x_3 + 20x_4 \leq 100\},\\
P_2 &= \{x \in \mathbb{R}^4 \mid 0 \leq x_1, x_2, x_3, x_4 \leq 1, 4x_1 + 3x_2 + 2x_3 + x_4 \leq 4\},\\
P_3 &= \{x \in \mathbb{R}^4 \mid 0 \leq x_1, x_2, x_3, x_4 \leq 1, 4x_1 + 3x_2 + 2x_3 + x_4 \leq 4,\\
&\qquad\qquad x_1 + x_2 + x_3 \leq 1, x_1 + x_4 \leq 1\}.
\end{aligned}$$

It is trivial to check that

- all elements of X satisfy the inequalities defining the sets P_1, P_2, P_3, and
- all elements of $\{0,1\}^4 \backslash X$ do not satisfy the inequalities defining the sets P_1, P_2, P_3.

Therefore, P_1, P_2, P_3 are formulations of X. We now show that

$$P_3 \subsetneq P_2 \subsetneq P_1.$$

Indeed, the formulation P_3 is formed by adding to the formulation P_2 two more inequalities $x_1+x_2+x_3 \leq 1$ and $x_1+x_4 \leq 1$. Thus, $P_3 \subset P_2$. One can easily check that $(\frac{1}{2}, 0, 0, 1) \in P_2$ but this point does not belong to P_3, therefore $P_3 \subsetneq P_2$. To prove the second inclusion, let $\mathbf{x} = (x_1, x_2, x_3, x_4)$ be an arbitrary point in P_2. From the definition of P_2, we have

$$\begin{aligned}
& 4x_1 + 3x_2 + 2x_3 + x_4 && \leq 4 \\
\Leftrightarrow \quad & \frac{83}{4}(4x_1 + 3x_2 + 2x_3 + x_4) && \leq 83 \\
\Leftrightarrow \quad & 83x_1 + 62.25x_2 + 41.5x_3 + 20.75x_4 && \leq 83 \\
\Rightarrow \quad & 83x_1 + 62.25x_2 + 49x_3 + 20.75x_4 && \leq 83 + 7.5x_3 \leq 90.5 < 100 \quad (\text{since } x_3 \leq 1).
\end{aligned}$$

Since x_1, x_2, x_3, x_4 are non-negative, we have

$$83x_1 + 62.25x_2 + 49x_3 + 20.75x_4 \geq 83x_1 + 61x_2 + 49x_3 + 20x_4.$$

Consequently, we obtain

$$83x_1 + 61x_2 + 49x_3 + 20x_4 < 100,$$

which means $\mathbf{x} \in P_1$. Since $\mathbf{x}$ is chosen arbitrarily in P_2, we conclude that $P_2 \subset P_1$. Furthermore, one can easily check that $(1, 0.1, 0, 0) \in P_1$ but this point does not belong to P_2. Hence $P_2 \subsetneq P_1$. To sum up, formulation P_3 is stronger than formulation P_2, and in turn formulation P_2 is better than formulation P_1.

It can be checked (non-trivially) that $P_3 = \text{conv(X)}$, thus it is an ideal formulation of X.

2.5 Extended formulation

In Section 2.4 we discuss an approach to formulate the feasible sets of mixed integer programs. In that approach, a set may have different formulations using the same set of variables, and we just need to add or modify constraints in each formulation to get another one. In this section, we discuss another approach, in which we add or choose different variables to formulate the feasible sets of mixed integer programs. For this approach we talk of *extended formulations*.

Let n be a positive integer, $N = \{1, \ldots, n\}$, and $M \subset N$. Consider a set $S \subset \mathbb{R}^N$. We denote

$$\text{proj}_{\mathbf{x}}(S) := \left\{ \mathbf{x} \in \mathbb{R}^M \mid \exists\ \mathbf{z} \in \mathbb{R}^{N \setminus M} \text{ such that } \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix} \in S \right\}.$$

This set is the orthogonal projection of S onto the subspace

$$L = \{\mathbf{x} \in \mathbb{R}^N \mid x_i = 0\ \forall i \in N \setminus M\}.$$

The motivation of the concept of *extended formulation* comes from the fact that the two programs

$$\max\{f(\mathbf{x}) \mid \mathbf{x} \in \text{proj}_{\mathbf{x}}(S)\}$$

and

$$\max \left\{ f(\mathbf{x}) + \mathbf{0}^t\mathbf{z} \mid \mathbf{x} \in \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix} \in S \right\}$$

are equivalent. However, sometimes the set S is easier to describe than its projection $\text{proj}_{\mathbf{x}}(S)$, so studying the second problem is easier. In the context of integer programming, we study the case in which the first problem is the linear relaxation of a mixed integer program. For that, we consider sets of the type

$$X = \{\mathbf{x} \in \mathbb{R}^M \mid A'\mathbf{x} \leq \mathbf{b}', x_i \in \mathbb{Z}(i \in I \subseteq M)\}, \tag{2.47}$$

that are of form of feasible sets of mixed integer programs. If we only use variables defining the set X, then the best formulation of X would be $\text{conv}(X)$. We aim to find an external description of the polyhedron $\text{conv}(X)$ as the projection of a polyhedron in a higher dimensional space, i.e., we want to have a polyhedron S such that $S = \text{conv}(X)$. This motivates the following definition.

Definition 2.11. (Extended formulation).
Let X be a set of the form (2.47). An extended formulation of X is a polyhedron

$$S = \left\{ \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix} \in \mathbb{R}^N \mid A\mathbf{x} + B\mathbf{z} \leq \mathbf{b} \right\}$$

such that $\text{proj}_{\mathbf{x}}(S) = \text{conv}(X)$. *If the size of $(A \mid B \mid \mathbf{b})$ is polynomially bounded in the size of $(A' \mid \mathbf{b}')$, then we say that S is a compact extended formulation of X.*

To give a simple example of the extended formulation concept, let us revisit the Minkowski-Weyl theorem (Theorem 1.28). It is stated in the theorem that if P is a polyhedron of form

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b}\},$$

then there exist $\mathbf{x}^1, \ldots, \mathbf{x}^m \in \mathbb{R}^n$ and $\mathbf{v}^1, \ldots, \mathbf{v}^k \in \mathbb{R}^n$ such that

$$P = \text{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^m) + \text{cone}(\mathbf{v}^1, \ldots, \mathbf{v}^k).$$

This means that

$$P = \text{proj}_{\mathbf{x}}(Q),$$

with

$$Q = \left\{ (\mathbf{x}, \lambda, \mu) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^k \mid \mathbf{x} = \sum_{i=1}^{m} \lambda_i \mathbf{x}^i + \sum_{i=1}^{k} \mu_i \mathbf{v}^i, \sum_{i=1}^{m} \lambda_i = 1, \lambda \geq \mathbf{0}, \mu \geq \mathbf{0} \right\}.$$

This provides Q as an extended formulation of P.

Chapter 3

Rapid programming and solvers

In Chapter 2 we have introduced various techniques to translate a problem description into a linear program (LP), an integer program (IP) in particular, or a mixed integer program (MIP) in general. The modeling is the first step to solve the given problem. To obtain numerically the solutions to the problem instances, especially in case of large-scale problem instances, one needs the help of computers. In order to make the computers 'understand' our problem instances, we need to implement the obtained models by using some programming languages. Once computers understand the models, we need to use some solvers to solve the problem instances to final solutions. This chapter presents a brief guide to use MATLAB for solving LPs and MIPs.

MATLAB (an abbreviation of 'matrix laboratory') is a proprietary multi-paradigm programming language and numerical computing environment, developed by MathWorks company. The first version of MATLAB was released in 1984, and now users can have MATLAB R2020a (version 9.8) which is released in 2020. More on MATLAB can be found on its official website

https://www.mathworks.com/products/matlab.html.

To use MATLAB for solving LPs and MIPs, users need to include the option of Optimization Toolbox when installing MATLAB on computers. Alternatively, users can use MATLAB Online which is available at

https://www.mathworks.com/products/matlab-online.html.

With MATLAB Online, no downloads or installations of MATLAB packages on computers is needed. MATLAB Online provides access to MATLAB from any standard web browser wherever users have internet access, all we need is to create an account and then just sign in a created account.

The MATLAB Optimization Toolbox provides MATLAB functions for finding solutions that optimize objectives while satisfying constraints. It includes solvers for linear programs, mixed integer programs, quadratic programs, second-order cone programs, nonlinear programs, constrained linear least squares, nonlinear least squares, and nonlinear equations. For the scope of this text book, we only focus on solving linear programs (LPs) and mixed integer programs (MIPs) with MATLAB.

3.1 Solve LPs by linprog solver of MATLAB

MATLAB provides linprog function as a linear programming solver. Details on this function can be found at

https://www.mathworks.com/help/optim/ug/linprog.html.

In this section we give only basic instructions about this MATLAB function. There are two ways of calling this function. In the first way, the parameters of a linear program are directly put in the syntax of the linprog function. In the second way, one needs to set up a linear program in form of a problem structure, and then put the problem structure to the input of the function syntax. The former way is discussed in Section 3.1.1, and the latter way is introduced in Section 3.1.2 and Section 3.1.3

3.1.1 Problem parameters as input

Often we are interested in finding an optimal solution and the optimal objective value of a given linear program. The syntax

```
[x,fval] = linprog(f,A,b,Aeq,beq,lb,ub)
```

is suitable for that purpose. This return `x` as the optimal solution and `fval` as the optimal objective value of the linear program

$$\begin{aligned} \min \quad & \mathbf{f}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & A_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub}. \end{aligned}$$

Here, $\mathbf{f}$ is the column vector of coefficients in objective function, $\mathbf{x}$ is the column vector of variables, $\mathbf{b}$ and $\mathbf{b}_{eq}$ are column vectors relating to the constraints, A and A_{eq} are matrices, $\mathbf{lb}$ and $\mathbf{ub}$ are column vectors that respectively define the lower bound and upper bound on the value of variable vector $\mathbf{x}$. These problem parameters are used as input of the function in the syntax. Note that by this syntax we solve the **minimization problem**, not the maximization problem. If we are only interested in finding an optimal solution, not the optimal objective value, of the linear program, we can use the following alternative syntax

```
x = linprog(f,A,b,Aeq,beq,lb,ub).
```

In this way of using linprog function, a linear program must be transformed into the above form in order to process further. More precisely, one has to decide the coefficient vector $\mathbf{f}$ of the objective function, the matrix A and vector $\mathbf{b}$ of the inequality constraints, the matrix A_{eq} and vector $\mathbf{b}_{eq}$ of the equality constraints, and the vectors $\mathbf{lb}$ and $\mathbf{ub}$ defining the range of values for variable vector. There are several special cases as follows.

- If a variable, say x_i, is unbounded from below, then we set the corresponding component of vector $\mathbf{lb}$ to $-\infty$ by the following command.

```
lb(i) = -inf
```

Similarly, if x_i is unbounded from above, then we set the corresponding component of vector **ub** to $+\infty$ by the following command.

```
ub(i) = inf
```

- Set `ub = []` if the whole variable vector **x** is unbounded from above. Set `lb = []` if the whole variable vector **x** is unbounded from below. If **x** is unbounded from above and below, then one can use the following reduced syntax

```
[x,fval] = linprog(f,A,b,Aeq,beq)
```

or

```
x = linprog(f,A,b,Aeq,beq)
```

if the information about optimal objective value is not necessary.

- If the specified input bounds for a problem are inconsistent (i.e., no value for **x** satisfies $\mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub}$), then MATLAB return `fval = []`.

- If no equalities exist in the constraints of the problem, then we set

`Aeq = []` and `beq = []`.

- If only inequalities take part in the constraints of the problem, i.e., if

```
Aeq = [], beq = [], lb = [], ub = [],
```

then one can use the following reduced syntax

```
[x,fval] = linprog(f,A,b)
```

or

```
x = linprog(f,A,b)
```

if we only want to find an optimal solution.

- Set

```
A = [], b = []
```

if no inequalities exist in the constraints of the problem.

For illustration purpose, we present here some examples.

Example 3.1.

Consider the following linear program

$$\begin{array}{rrl} \min & 2x_1 + 5x_2 + 4x_3 + x_4 - 5x_5 & \\ \text{s.t.} & x_1 - 6x_2 - 2x_4 - 9x_5 = 32 & \\ & 2x_2 + x_3 + \frac{1}{2}x_4 + \frac{3}{2}x_5 = 30 & \\ & 3x_2 + x_5 \geq 36 & \\ & 4x_1 + 2x_3 - 2x_4 + 3x_5 \geq 20 & \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 & \end{array}$$

Transforming this linear program to the valid form of linprog function

$$\begin{aligned} \min \quad & \mathbf{f}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & A_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub}. \end{aligned}$$

we have

$$\begin{aligned} \mathbf{f} &= (2, 5, 4, 1, -5)^t, \\ A &= \begin{bmatrix} 0 & -3 & 0 & 0 & -1 \\ -4 & 0 & -2 & 2 & -3 \end{bmatrix}, \\ \mathbf{b} &= \begin{bmatrix} -36 \\ -20 \end{bmatrix}, \\ A_{eq} &= \begin{bmatrix} 1 & -6 & 0 & -2 & -9 \\ 0 & 2 & 1 & \frac{1}{2} & \frac{3}{2} \end{bmatrix}, \\ \mathbf{b}_{eq} &= \begin{bmatrix} 32 \\ 30 \end{bmatrix}, \\ \mathbf{lb} &= (0, 0, 0, 0, 0)^t, \\ \mathbf{ub} &= [\,]. \end{aligned}$$

To import these parameters, in the command window of MATLAB, type each of the following commands (a command starts after the sign `>>` and ends with an enter from keyboard):

```
>> f = [2 5 4 1 -5]'
>> A = [0 -3 0 0 -1; -4 0 -2 2 -3]
>> b = [-36 -20]'
>> Aeq = [1 -6 0 -2 -9; 0 2 1 1/2 3/2]
>> beq = [32 30]'
>> lb = zeros(5,1)
>> ub = []
```

and then type the syntax of linprog function

```
>> [x,fval] = linprog(f, A, b, Aeq, beq, lb, ub)
```

After a while, MATLAB announces on the command window

```
Optimal solution found.

x =

104.0000
12.0000
6.0000
0
0

fval =

292.0000
```

which means that the given linear program has an optimal solution whose value is

$$(x_1, x_2, x_3, x_4, x_5) = (104, 12, 6, 0, 0)$$

and the optimal objective value is 292.

Example 3.2.

Consider the following linear program

$$\begin{array}{rrrrrrrl} \max & 2x_1 & + 4x_2 & + & x_3 & + & x_4 & \\ \text{s.t.} & x_1 & + 3x_2 & + & x_3 & + & x_4 & \leq 1 \\ & & - 5x_2 & & & - & 2x_4 & \leq 3 \\ & & & & x_2 & + 4x_3 & + x_4 & \leq 3 \\ & & & & & & x_1, x_2 & \geq 0 \\ & & & & & & x_3 & \leq 15 \\ & & & & & & x_4 & \in \mathbb{R} \end{array}$$

This is a maximization problem. Transforming this linear program to the valid form of linprog function

$$\begin{array}{rr} \min & \mathbf{f}^t\mathbf{x} \\ \text{s.t.} & A\mathbf{x} \leq \mathbf{b} \\ & A_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub}. \end{array}$$

we have

$$\mathbf{f} = (-2, 4, 1, 1)^t,$$

$$A = \begin{bmatrix} 1 & 3 & 1 & 1 \\ 0 & -5 & 0 & -2 \\ 0 & 1 & 4 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix},$$
$$A_{eq} = [\,], \quad \mathbf{b}_{eq} = [\,],$$
$$\mathbf{lb} = (0, 0, -\infty, -\infty)^t,$$
$$\mathbf{ub} = (+\infty, +\infty, 15, +\infty)^t.$$

In the following we show the commands to import these parameters and how MATLAB shows the obtained values on the command window.

```
>> f = [-2 4 1 1]'

f =

-2
4
1
1

>> A = [1 3 1 1; 0 -5 0 -2; 0 1 4 1]

A =

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

>> b = [1 3 3]'

b =

1
3
3

>> lb = [0 0 -inf -inf]'

lb =

0
0
-Inf
-Inf
```

```
>> ub = [inf inf 15 inf]'

ub =

Inf
Inf
15
Inf
```

Now, if we only want to find an optimal solution to the given linear program, we can type the following command

```
>> x = linprog(f, A, b, [], [], lb, ub)
```

in which the two notations `[]` mean that `Aeq = []` and `beq = []`, since they appear at the positions of this parameters in the syntax of linprog function. The answer we get on the command window is

```
Problem is unbounded.
```

That means the given linear program is unbounded.

3.1.2 Problem structure as input

MATLAB provides problem-based approach in which users can set up a linear program step by step as follows. First, create an OptimizationProblem object to represent the given linear program. Then, define each component of this object, including the problem variables, the problem objective, and afterward the problem constraints. The created object then is converted to a problem structure, which is put as the input of linprog function. An advantage of this way is that we can import a given linear program as it is, and do not need to transform it into the format described in the above part.

The following example illustrates how to solve a linear program using this problem-based approach for linprog solver.

Example 3.3.

Consider the following linear program

$$\begin{array}{rrcl} \max & 2x_1 - x_2 \quad - x_4 & & \\ \text{s.t.} & x_1 + x_2 - x_3 & \leq & 15 \\ & x_1 + x_2 + x_3 + x_4 & = & 27 \\ & -2x_1 + x_2 + x_3 & \geq & 18 \\ & x_1 & \geq & 0 \\ & x_1 & \leq & 10 \\ & x_2 & \geq & -5 \\ & x_2 & \leq & 5 \\ & x_3, x_4 & \geq & 0 \end{array}$$

Following the problem-based approach, we create an OptimizationProblem object named prob to represent the given linear program. For that, we need to define the problem variables of this linear program by the following commands.

```
>> x1 = optimvar('x1','LowerBound',0,'UpperBound',10);
>> x2 = optimvar('x2','LowerBound',-5,'UpperBound',5);
>> x3 = optimvar('x3','LowerBound',0,'UpperBound',inf);
>> x4 = optimvar('x4','LowerBound',0,'UpperBound',inf);
```

These commands use the key word `optimvar` to tell MATLAB that they define the variables of the given problem. The first command creates variable `x1` whose lower bound is 0 and upper bound is 10. This variable is exactly the variable x_1 in the given linear program. Similarly, the remaining commands creates the other variables of the problem. Note that the semicolon sign after each command tells MATLAB that does not show information imported from the command. Without the semicolon, MATLAB will show on the command window the information of variables like

```
x1 =

OptimizationVariable with properties:

Name: 'x1'
Type: 'continuous'
IndexNames: {{}  {}}
LowerBound: 0
UpperBound: 10

See variables with show.
See bounds with showbounds.
```

Now, we are ready to create the problem object, and the command is

```
>> prob = optimproblem('Objective',2*x1 - x2 - x4,'ObjectiveSense','max');
```

The part `'Objective',2*x1 - x2 - x4` tells MATLAB that the objective function is $2x_1 - x_2 - x_4$, while the part `'ObjectiveSense','max'` tells MATLAB that we aim to solve a maximization problem. Except for the constraints on the ranges of variables, the given linear program has three constraints, and they are imported to the object prob by the following commands:

```
>> prob.Constraints.c1 = x1 + x2 - x3 <= 15;
>> prob.Constraints.c2 = x1 + x2 + x3 + x4 == 27;
>> prob.Constraints.c3 = -2 * x1 + x2 + x3 >= 18;
```

These commands assign the name `c1, c2, c3` for the constraint field of the object prob. Now, the object prob is fully defined, and we need to convert the problem object to a problem structure by the following command:

```
>> problem = prob2struct(prob);
```

The last step is to put the problem structure as the input of linprog function as follows.

```
>> [sol,fval] = linprog(problem)
```

This command gives `sol` as the optimal solution of the given program, and `fval` as the optimal objective value. For our considering example, MATLAB returns

```
Optimal solution found.

sol =

3.0000
-5.0000
29.0000
0

fval =

-11
```

which means that the optimal solution to the given linear program is (3, -5, 29, 0) and the optimal objective value is -11. □

3.1.3 MPS files for mpsread

There is another way to create an OptimizationProblem object to represent a linear program in MATLAB. The object contents can be imported from a text file. Unlike the way described in Section 3.1.2 in which we have to use MATLAB commands to define a linear program, in the way discussed in the present section we save a linear program into a file and then import to MATLAB. Of course, in order to be recognized by MATLAB, the text file must be given in a prescribed format. MATLAB provides function 'mpsread' to read data of a linear program from a file of MPS (Mathematical Programming System) format. The files with this format must have the extension .mps. Detail introduction about MPS format is available at

http://lpsolve.sourceforge.net/5.5/mps-format.htm.

MPS input format was originally introduced by IBM to express linear and integer programs in a standard way. Horizontally, a MPS file is divided in sections, each section is marked by a so-called header card. The header cards must be written in upper-case, while mixture of lower-case and upper-case can be used anywhere else. Vertically, a MPS file is divided in fields, and the fields start in column 2, 5, 15, 25, 40 and 50. In each line, each character occupies one space corresponding to one column. Header cards are distinguished by their starting in column 1. The precise template for the most common use of MPS format is as follows.

Field:	1	2	3	4	5	6
Columns:	2−3	5−12	15−22	25−36	40−47	50−61
	NAME		problem name			
	ROWS					
	type	name				
	COLUMNS					
		column name	row name	value	row name	value
	RHS					
		rhs name	row name	value	row name	value
	RANGES					
		range name	row name	value	row name	value
	BOUNDS					
	type	bound name	column name	value		
	ENDATA					

There are some remarks before going to the details of the template.

- The contents between the first and the second horizontal lines in the template, as well as the three horizontal lines, are just for illustration and explanation purposes. They do not occur in any MPS file.
- MPS format is a fixed column format, so one must be careful when placing all information in the correct columns.
- Everything in the linear program encoded in a MPS file gets a name. The names that you choose for the individual entities (objective, constraints, or variables of the program) are not important to the solver. However, you should set names that are meaningful to you, or will be easy for a post-processing code to read.
- Field 5 and field 6 are optional, as are RANGES and BOUNDS sections. Everything else is required.
- There is nothing in MPS format that specifies the direction of optimization. By

default, the linprog solver in MATLAB solves linear programs encoded in the MPS files **in the direction of minimization**.

In the following we will explain this template via a small example. Assume that we want to represent in MPS format the following linear program

$$\begin{array}{rrl} \min & 3x_1 + 5x_2 + 8x_3 & \\ \text{s.t.} & x_1 + 2x_2 \qquad\quad \leq & 6 \\ & x_1 \qquad + \;\; x_3 \geq & -8 \\ & -\,2x_2 + 3x_3 = & 2 \\ & x_1 \leq & 3 \\ & x_2 \geq & -7 \\ & x_2 \leq & 10 \\ & x_3 \in & \mathbb{R} \end{array}$$

For convenience, we set the name TestProg for this program. The MPS file representing this program start with the NAME section. The header card NAME is written on the first line of the file, and start at column 1. This section starts in column 15 of the same line. Although it can have anything the users want, its role is to tell the problem name to readers. Therefore, the first line of the MPS file looks like

```
NAME          TestProg
```

Next to the NAME section is the ROWS section. This section defines the names of all the constraints. The header card ROWS is written in a single line, starts at column 1. Then, each line in this section tells us some information about the objective function and constraints of the program. More precisely, in ROWS section, each row of the constraint matrix must have a row type and a row name specified. The codes for indicating row types are given in Table 3.1.

Row type	Meaning
N	objective
E	equality constraint
G	greater-than-or-equal constraint
L	less-than-or-equal constraint
N	no restriction

Table 3.1: Row types and meaning in MPS format.

Except for the constraints on domain of variables, our test program has an objective function and three constraints. We set the name ObjF for the objective function, and respectively set the names Con1, Con2, Con3 for the first three constraints. Constraint Con1 is

$$x_1 + 2x_2 \leq 6,$$

which is of the less-than-or-equal form, hence it is of row type L. Constraint Con2 is

$$x_1 + x_3 \geq 8,$$

which is of the greater-than-or-equal form, hence it is of row type G. Constraint Con3 is

$$-2x_2 + 3x_3 = 5,$$

which is an equality, hence it is of row type E. So the ROWS section of the file is written as

```
ROWS
 N  ObjF
 L  Con1
 G  Con2
 E  Con3
```

Note that the codes for row types are written in the first field (starting from column 2), while the row names are written in the second fields (starting from column 5).

The largest part of the file is in the COLUMNS section. Similar to the other header cards, the header card COLUMNS is written in a single line, starts at column 1. Then, in the following lines of this section, the names of the variables are defined along with the **nonzero** coefficients of the objective and constraints. More precisely, in each line of this section, 'column name' is a variable name, 'row name' is a constraint name, and 'value' is the coefficient of corresponding variable in the corresponding constraint. Note that the column names must be in second field which starts from column 5. Note furthermore that, all entries for a given column name (i.e., a variable) must be placed consecutively in the next fields, before we come to another column name.

To illustrate this structure, let us come back to our TestProg above. For representing variables x_1, x_2, x_3 of the program in the file, we respectively set the name `x1, x2, x3` for these variables. Consider the variable x_1 of the program, which has the name `x1` in the file.

(i) Its coefficient in the objective function (named `ObjF` in the file) is 3.

(ii) Its coefficient in the first constraint (named `Con1` in the file) is 1.

(iii) Its coefficient in the second constraint (named `Con2` in the file) is 1.

(iv) Its coefficient in the third constraint (named `Con3` in the file) is 0, and therefore we do not need to write down the information of this coefficient to the file.

So the information about the coefficients of variables x_1 in the objective and constraints is given in COLUMNS section of the file as follows.

```
    x1        ObjF       3              Con1       1
    x1        Con2       1
```

In the above way, we use field 5 and field 6 to enter coefficients of variable `x1`. Since these fields are optional, we can use another way to enter these information, which spends up to field 4 as follows.

```
    x1        ObjF      3
    x1        Con1      1
    x1        Con2      1
```

Similarly, for variable x_2 of the program, which has the name `x2` in the file, we see that its coefficients in objective function and the first three constraints are respectively 5, 2, 0, -2. Since only nonzero coefficients take part in the file, the part concerning the variable x_2 in COLUMNS section of the file should be

```
    x2        ObjF      5              Con1      2
    x2        Con3      -2
```

Following the same paradigm, for variable x_3 of the program, its part in COLUMNS section of the file is

```
    x3        ObjF      8              Con2      1
    x3        Con3      3
```

The RHS section starts with the card header RHS written in the first field of a single line. This section contains information for the right-hand side of the program. One can gather the free coefficients in the right hand sides of the constraints into a single vector, and then set a name for this vector in the MPS file. One can also partition the set of these coefficients into different vectors and set different names for these vectors. The latter option is often used when some constraints have the same purpose, and in this case the right-hand-side coefficient of these constraints belong to a same vector. The RHS section has similar structure as the COLUMNS section, except that the 'rhs name' refers to the name for a right-hand-side vector. As an example, for our TestProg, the right-hand-side coefficients of the constraints are 6, 8, 5 respectively. We gather them into a single vector named `Rhs1`. The RHS section of the file should be

```
RHS
     Rhs1      Con1      6              Con2      -8
     Rhs1      Con3      2
```

Note that the objective may also have a constant. This can also be specified in RHS section by using the object name as constraint name and then specifying the constant. However, we can neglect this issue, since we can omit the constant in objective to obtain an equivalent program.

The RANGES section is used for constraints of the form

$$\ell \leq \text{ constraint } \leq u.$$

For such a constraint, its range is $r = u - \ell$. The value of r is specified in the RANGES section, and the value of u or ℓ is specified in the RHS section. Since this section is optional, we skip discussing about this section here.

Bounds on the variables are specified in the optional BOUNDS section. As other sections, this section starts with card header BOUNDS in the first field of a single line. Then,

in the following line of this section, one can put lower and upper bounds on individual variables, instead of having to define extra rows in the coefficient matrix of constraints. When bounds are not indicated, the default bounds $0 \leq x < +\infty$ are assumed.

For setting the bounds of each variable, one first needs to specify the bound type. The codes for indicating bound types is given in Table 3.2. The bound type code of a variable is written in the first field of each line in this section. Then, one needs to set a bound name in the second field, before put the column name (i.e. the name of a variable) in the third field. One can use a bound name for all variables, or can use different bound names for different subsets of variables. In the latter situation, the bound name of each subset of variables should represent the common meaning of the variables in the subset.

Bound type	Meaning	In formula
LO	lower bound	$b \leq x$, b will be specified
UP	upper bound	$x \leq b$, b will be specified
FX	fixed variable	$x = b$, b will be specified
FR	free variable	$x \in \mathbb{R}$
MI	no lower bound	$x > -\infty$
PL	no upper bound	$x < +\infty$

Table 3.2: Bound types and meaning in MPS format for linear programs.

As an illustrative example, following these rules of setting up the bounds for variables in MPS format, the BOUNDS section for our TestProg should be

```
BOUNDS
 UP  Bnd1      x1      3
 LO  Bnd1      x2      -7
 UP  Bnd1      x2      10
 FR  Bnd1      x3
```

Here, we set the name `Bnd1` for all of the related bounds in the section. The first information line of this section tells that the bound for variable `x1` is of type UP with bound value 3, i.e., $x_1 \leq 3$. The second line tells that the bound for variable `x2` is of type LO with bound value -7, i.e., $x_2 \geq -7$. Similarly, the third line means that $x_2 \leq 10$. The last line means that x_3 is a free variable, i.e. $x_3 \in \mathbb{R}$.

The final card must be ENDATA, which is put in the first field of the last line of the file. This announces the end of the encoded data. To sum up, we recall our TestProg here

$$
\begin{aligned}
\min\ & 3x_1 + 5x_2 + 8x_3 & & \\
\text{s.t.}\ & x_1 + 2x_2 & \leq\ & 6 \\
& x_1 \quad + \quad x_3 & \geq\ & -8 \\
& -2x_2 + 3x_3 & =\ & 2 \\
& x_1 & \leq\ & 3 \\
& x_2 & \geq\ & -7 \\
& x_2 & \leq\ & 10 \\
& x_3 & \in\ & \mathbb{R}
\end{aligned}
$$

and the MPS file encoding this linear program is given below.

```
NAME          TestProg

ROWS
 N  ObjF
 L  Con1
 G  Con2
 E  Con3

COLUMNS
    x1        ObjF      3          Con1      1
    x1        Con2      1
    x2        ObjF      5          Con1      2
    x2        Con3      -2
    x3        ObjF      8          Con2      1
    x3        Con3      3

RHS
    Rhs1      Con1      6          Con2      -8
    Rhs1      Con3      2

BOUNDS
 UP Bnd1      x1        3
 LO Bnd1      x2        -7
 UP Bnd1      x2        10
 FR Bnd1      x3

ENDATA
```

Let us set the name TestProg.mps for the MPS file encoding our TestProg. We now need to import the linear program in the MPS file into a problem object in MATLAB. Assume that the file is located in the directory `C:\Users\` on your computer, then we use the following MATLAB command to import the problem data

```
>> problem = mpsread('C:\Users\TestProg.mps')
```

After this command, the data in the MPS file is transformed to the problem structure `problem`, and it is ready to be solved by linprog function. The last step is to enter the command

```
>> [x, fval] = linprog(problem)
```

MATLAB returns the solution information for our TestProg as follows.

```
Optimal solution found.
```

```
x =

    0
    -7.0000
    -4.0000

fval =

    -67
```

That is, the optimal solution to our TestProg is $(x_1, x_2, x_3) = (0, -7, -4)$ with the optimal objective value is -67.

3.2 Solve MIPs by intlinprog solver of MATLAB

MATLAB provides intlinprog function as a mixed integer linear programming solver. Details on this function can be found at

https://www.mathworks.com/help/optim/ug/intlinprog.html.

The use of this MATLAB function is similar to the use of linprog function described in Section 3.1, except for some remarks concerning integer variables in MIPs.

3.2.1 MPS format for MIPs

The MPS format for MIPs is the same as the one for LPs introduced in Section 3.1.3, with additional specification of integer variables. Markers are used to indicate the start and end of a group of integer variables. The start marker has its name in field 2, 'MARKER' in field 3, and 'INTORG' in field 5. The end marker has its name in field 2, 'MARKER' in field 3, and 'INTEND' in field 5. These markers are placed in the COLUMNS section. When there are BOUNDS on the variables, then these are used as lower and upper bound of these integer variables and there is no confusion possible. Even a lower bound of 0 is already enough. In that case, if there is no upper bound, infinite is used. In addition to the bound types given in Table 3.2 for linear programs, with mixed integer programs there are more bound types given in Table 3.3.

Bound type	Meaning	In formula
BV	binary variable	$x \in \{0, 1\}$
LI	lower bound for integer variable	$b \leq x$, b will be specified
UI	upper bound for integer variable	$x \leq b$, b will be specified

Table 3.3: Bound types and meaning in MPS format for mixed integer programs (in addition to the ones in Table 3.2).

Once the data of a MIP in the MPS file is transformed to a problem structure `problem` in MATLAB, we are ready to solve the MIP by intlinprog function by entering the following command

```
>> [x, fval] = intlinprog(problem)
```

3.2.2 Problem parameters as input

The syntax

```
[x,fval] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub)
```

is the most common used when solving mixed integer programs by intlinprog solver of MATLAB. This return `x` as the optimal solution and `fval` as the optimal objective value of the linear program

$$\begin{aligned} \min \quad & \mathbf{f}^t\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & A_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \\ & x_i \in \mathbb{Z} \quad \forall i \in \text{intcon} \end{aligned}$$

Here, in the syntax, `intcon` is a vector of indices of integer variables in the input MIP. If we need to impose a variable as binary, then in addition to putting the corresponding index of the variable to the vector `intcon`, we need to set the that index in `lb` as 0, and that index of the variable in `ub` as 1. In the following example, we give an application of the intlinprog solver of MATLAB in solving Sudoku puzzles.

Example 3.4.

Sudoku puzzles and a mixed integer programming formulation for the problem of solving these puzzles are introduced in Example 2.7. In this example, we introduce another formulation for this problem, based on the use of the following binary variables

$$x_{ijk} = \begin{cases} 1 & \text{if } k \text{ is filled in the cell } (i,j) \text{ of the grid,} \\ 0 & \text{otherwise.} \end{cases}$$

Given a Sudoku puzzle, let G be the set representing the clues given on the grid, i.e.,

$$G = \{(i,j,k) \mid k \text{ is already filled in the cell } (i,j)\}.$$

We come up with the following binary integer linear program:

$$(SUDOKU - BLP) \qquad \min \sum_{i,j,k \in S} 0 \cdot x_{ijk} \tag{3.1}$$

$$\text{subject to} \qquad x_{ijk} = 1 \qquad \forall (i,j,k) \in G \tag{3.2}$$

$$\sum_{k \in S} x_{ijk} = 1 \qquad \forall i \in S, j \in S \tag{3.3}$$

$$\sum_{i \in S} x_{ijk} = 1 \qquad \forall j \in S, k \in S \tag{3.4}$$

$$\sum_{j \in S} x_{ijk} = 1 \qquad \forall i \in S, k \in S \tag{3.5}$$

$$\sum_{i=p}^{p+2} \sum_{j=q}^{q+2} x_{ijk} = 1 \qquad \forall k \in S, (p,q) \in B \tag{3.6}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i \in S, j \in S, k \in S \tag{3.7}$$

Constraints (3.2) take care of the already-filled cells in the grid. Constraints (3.3) ensure that each cell in the grid is filled by exactly one value of $k \in S$. Constraints (3.4) mean that in each column of the grid each value of $k \in S$ appears exactly once. Similarly, constraints (3.5) mean that in each row of the grid each value of $k \in S$ appears exactly once. Each pair of (p, q) in constraints (3.6) corresponds to an 3×3 block in the puzzle, therefore these constraints ensure that each value of $k \in S$ appears exactly once in each of the blocks. The domains of variables are given in constraints (3.7).

Note that the Sudoku problem is in fact a feasibility problem, therefore no objective function is needed. However, for the manner of solving Sudoku problems by using MIP solvers, an objective function is required. In our model, for simplicity we choose 0 for coefficients in the objective function, but other values for the coefficients can also be chosen. Different choices of the coefficients' values may lead to different solutions in case that the Sudoku problem has more than one solution. This model uses 729 binary variables and $324 + |G|$ constraints.

The formulation can be implemented in MATLAB and solved by intlinprog function as in the following code.

```
function [SudokuSolution, exitFlag] = SudokuBLP(n, G)
% This MATLAB function is to solve a Sudoku puzzle
% by using binary linear program.

% INPUT:
% n: the integer telling the size of the puzzle (n^2 x n^2).
% G: the matrix expresses the given clues in the puzzle.
% The matrix G must have exactly 3 columns.
% The first two elements in each row are the i, j coordinates of a clue
% and the third element is the value of the corresponding clue.

% SOLUTION ALGORITHM:
% express the puzzle in form of a binary linear program
% and call MATLAB function "intlinprog" to solve the puzzle.

% OUTPUT:
```

```
% SudokuSolution: solution to the puzzle in form of a matrix.
% exitFlag: status of solution existence.

%% Presolve
% Analyze the input value of n and the input matrix G
% to ensure its validity

% n must be an integer and at least 2
if (n ~= round(n)) || (n < 2)
    error('The input value of n is not valid')
end

% The matrix G must have exactly 3 columns
if size(G, 2) ~= 3
    error('The input matrix must have three columns')
end

% All entries of G must be integers from 1 to n^2
if sum([any(G ~= round(G)), any(G < 1), any(G > n^2)])
    error(['Entries must be integers from 1 to ' int2str(n^2)])
end

%% Parameters to be used
% The constraints of the binary linear program modeling the puzzle
% will be represented by linear system Aeq * x = beq,
% where Aeq is a matrix, beq is a vector, and x is vector of variables

% Number of variables, which is also the number of columns of Aeq
nVar = n^6;

% Number of constraints, which is also the number of rows of Aeq
nCon = 4*n^4;   % see construction of Aeq below

% Initialize values for coefficients in the system of constraints
Aeq = zeros(nCon, nVar);
beq = ones(nCon, 1);

% Range of values of variables
% (binary variables are relaxed to real variables in [0, 1],
% the optimal solution still has binary entries)
lb = zeros(n^2, n^2, n^2);
ub = lb + 1;

% Coefficients of objective function
```

```
% (they can be anything,
% but having non-zero values may speed up the solver)
f = zeros(1, nVar)';

%% Constraints of Sudoku puzzle

% For each entry of the array lb that corresponds to a clue, reset it to 1.
% This forces the corresponding variable x(i, j, k) to be 1.
for i = 1 : size(G, 1)
    lb(G(i, 1), G(i, 2), G(i, 3)) = 1;
end

% Use a counter variable to count the number of constraints
counter = 1;

% The system Aeq*x=beq is the union of constraints in the following sets:
% Each cell in the grid is filled by exactly one valid number
for i = 1 : n^2
    for j = 1 : n^2
        Acell = zeros(n^2, n^2, n^2);
        Acell(i, j, 1 : end) = 1;
        Aeq(counter, :) = Acell(:)';    % one row in Aeq*x = beq
        counter = counter + 1;
    end
end

% Each valid number appears exactly once in each column of the grid
for j = 1 : n^2
    for k = 1 : n^2
        Acolumn = zeros(n^2, n^2, n^2);
        Acolumn(1 : end, j, k) = 1;
        Aeq(counter, :) = Acolumn(:)';  % one row in Aeq*x = beq
        counter = counter + 1;
    end
end

% Each valid number appears exactly once in each row of the grid
for i = 1 : n^2
    for k = 1 : n^2
        Arow = zeros(n^2, n^2, n^2);
        Arow(i, 1 : end, k) = 1;
        Aeq(counter, :) = Arow(:)';  % one row in Aeq*x = beq
        counter = counter + 1;
    end
```

```
end

% Each valid number appears exactly once in each block of the grid
for U = 0 : n : n^2-n
    for V = 0 : n : n^2-n
        for k = 1 : n^2
            Ablock = zeros(n^2, n^2, n^2);
            Ablock(U + (1 : n), V + (1 : n), k) = 1;
            Aeq(counter,:) = Ablock(:)';    % one row in Aeq*x = beq
            counter = counter + 1;
        end
    end
end

%% Solve the binary linear program
% The Sudoku constraints are represented in system Aeq * x = beq,
% the clues are ones in array lb,
% Binary variables are represented by setting intcon argument to 1 : nVar
% together with vectors lb and ub.
% Solve the problem by calling intlinprog.

intcon = 1 : nVar;
[x, ~, exitFlag] = intlinprog(f, intcon, [], [], Aeq, beq, lb, ub);
% The apperance of [] in the third and fourth arguments of intlinprog
% means that no inequality appears in the set of constraints.

%% Convert the solution of the binary linear program to Sudoku form
% First, multiply each element x(i, j, k) with k.
% Then, add up the elements having the same (i, j)-coordinate.

% If the binary linear program has an optimal solution, then convert it
if exitFlag > 0
    % Change x back to $n^2$-by-$n^2$-by-$n^2$ array
    x = reshape(x, n^2, n^2, n^2);
    % Clean up non-integer elements
    x = round(x);
    % Multiply each element x(i, j, k) with k,
    % then save into an auxilary array to convert to Sudoku form later
    y = ones(size(x));
    for k = 2 : n^2
        y(:,:,k) = k;
    end
    SudokuSolution = x.*y;  % element-wise multiplication
    % Convert the obtain array to Sudoku form by taking
```

```
    % the sum along dimension 3 (the last coordinate) of the array.
    % It results in an $n^2$-by-$n^2$-array holding the solved puzzle
    SudokuSolution = sum(SudokuSolution, 3);
else
    % If no optimal solution to the binary linear program exits,
    % then there is no solution to the given Sudoku puzzle
    SudokuSolution = [];
end
```