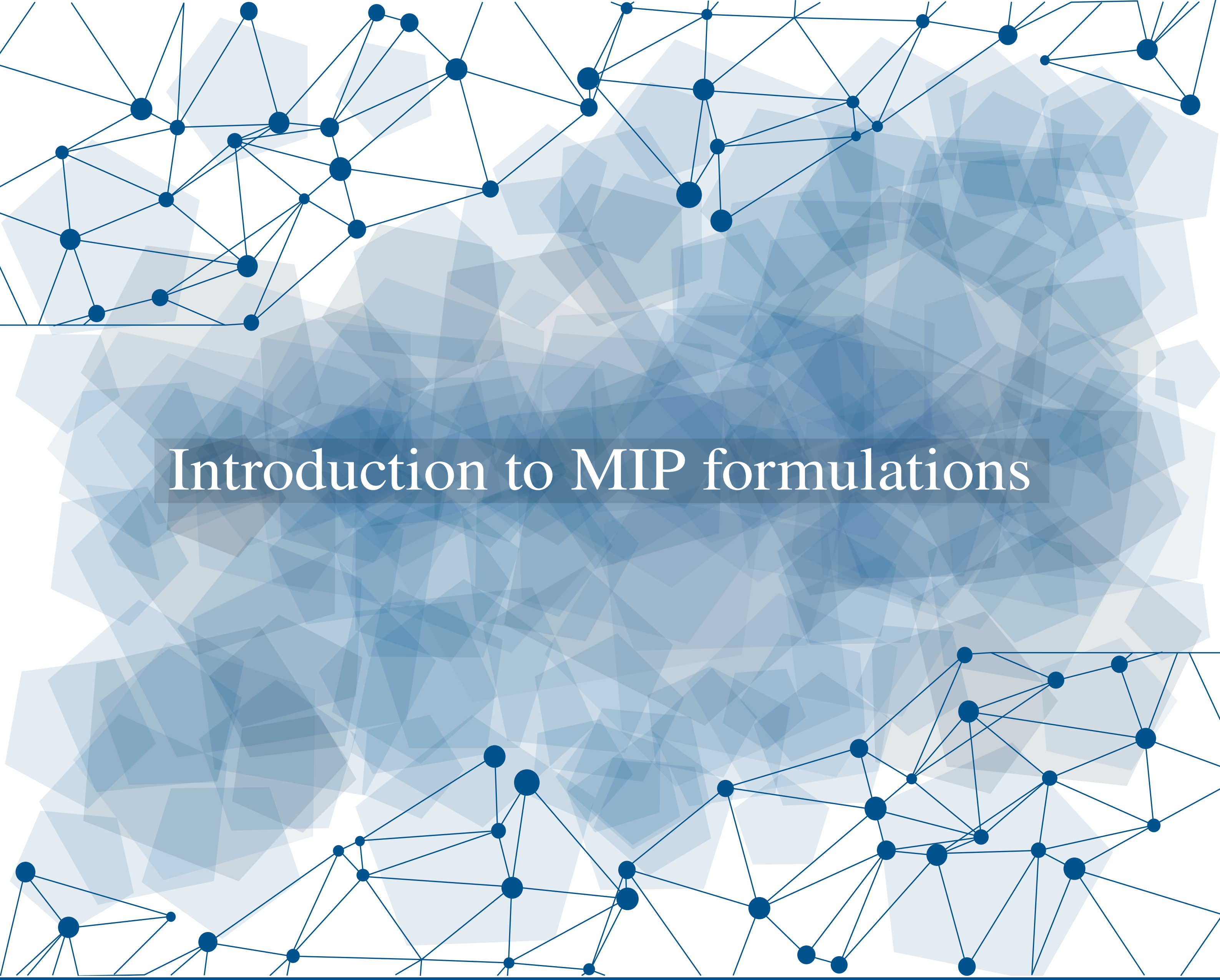


Introduction to MIP formulations

Author: R Luies & SE Terblanche

Date: April 9, 2022

Version: 1.0

Ideas are cheap, implementation and verification determine the quality of an idea.

Contents

Chapter 1 Introduction

Introduction

- ❏ *General Optimisation*
- ❏ *Linear Programming*
- ❏ *Duality*
- ❏ *Simplex Method*

We give a brief introduction to optimisation, which include the linear programming, theory of duality and the Simplex Method.

1.1 Optimisation

Optimisation is a mathematical concept where we maximise or minimise some function. A simple case is to determine a stationary point x^* for a univariate function (shown in Figure 1.1), this can be achieved when $f'(x) = 0$, and

- if $f''(x^*) < 0$, then $f(x)$ has a local maxima at x^*,
- if $f(x)$ is a concave function, then $f(x)$ has a global maximum at x^*.

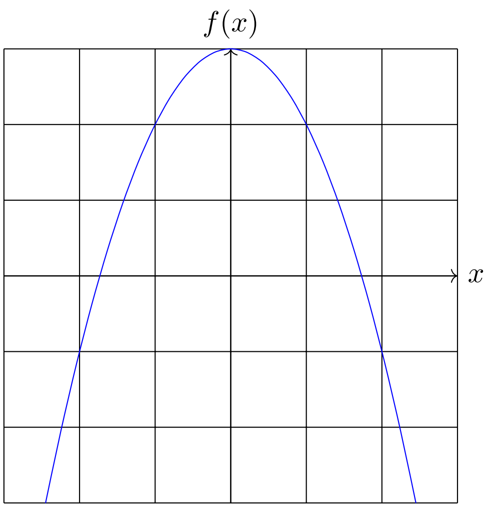


Figure 1.1: Univariate function $f(x)$

Difficult cases require a numerical or an algorithmic approach such as,

- general non-linear functions with multiple local optima,
- multivariate function,
- optimisation with constraints,
- and optimisation with integrality requirements.

A broad definition of optimisation consists of minimising or maximising a real function by systematically selecting from an allowed set of inputs $\mathcal{I}$ and choosing the best input. Given a function $f : \mathcal{I} \to \mathbb{R}$ we seek an element $\boldsymbol{x}^* \in \mathcal{I}$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{I}$ in minimisation problems, and $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{I}$ in maximisation problems. The corresponding minimisation mathematical program is:

$$\begin{aligned} &\min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ &\text{subject to } \boldsymbol{x} \in \mathcal{I}. \end{aligned} \tag{1.1}$$

It is important to distinguish between local- and global optima since some algorithms cannot guarantee optimality. A point $\boldsymbol{x}^*$ is considered a local minimum if there exists some $\epsilon > 0$ such that, for all $\boldsymbol{x} \in \mathcal{I}$ within a distance ϵ of x^*, $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$. With the latter statement, when $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$, the point $\boldsymbol{x}^*$ is considered a local maximum.

When complete search space is explored, the global optima can be determined. For a minimisation problem, a point $\boldsymbol{x}^*$ is considered the global minimum if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{I}$. For a maximisation problem, a point $\boldsymbol{x}^*$ is considered the global maximum if $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{I}$. Figure 1.2 is a graphical representation of these statements.

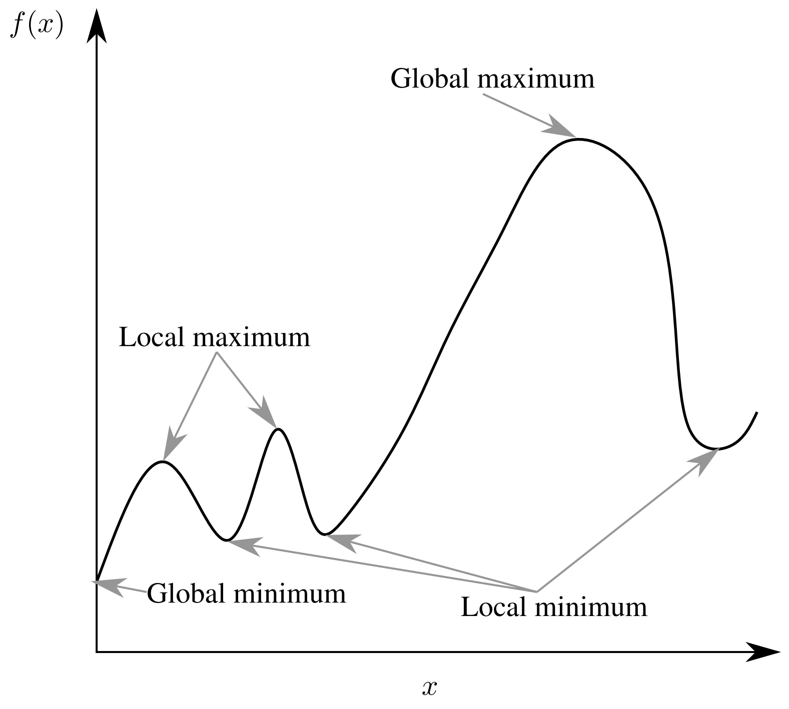


Figure 1.2: Local- and global optima for $f(x)$.

Example 1.1 Optimisation with constraints Consider the mathematical program,

$$\begin{array}{lrcrcr} \max_{x,y} & 4x & + & 6y & & \\ \text{s.t.} & -x & + & y & \leq & 11 \\ & x & + & y & \leq & 27 \\ & 2x & + & 5y & \leq & 90 \\ & & & y & \geq & 0 \\ & x & & & \geq & 0 \end{array} \tag{1.2}$$

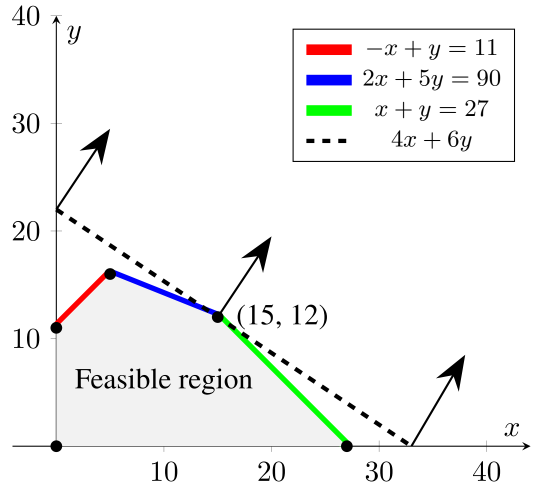
we can find the optimal solution using the graphical method shown in Figure 1.3a.

Example 1.2 In some cases, there can exist multiple optimal solutions, but the optimal objective value stays the same. If the objective function of the mathematical program (1.2) was,

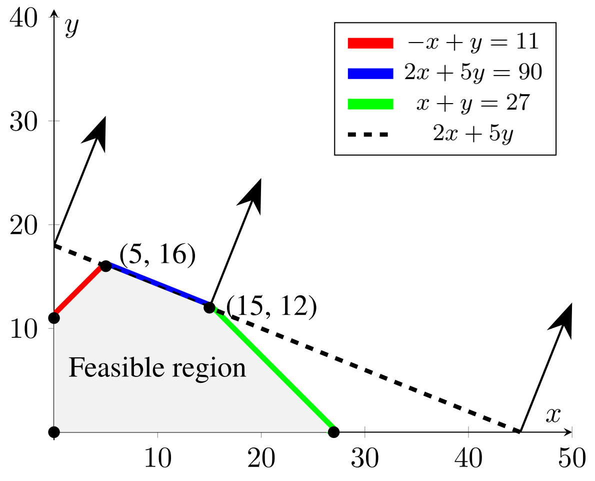
$$\max_{x,y}\ 2x + 5y, \tag{1.3}$$

the objective function would be parallel to $2x + 5y = 90$, this is shown in Figure 1.3b.

Example 1.1 is also known as a linear program, which is a special case of optimization an form an important part of formulating Mixed Integer Linear Programming (MILP) models. In the next sections we discuss linear and integer programs in more depth.



(a) Graphical Approach for the mathematical program (1.2).



(b) Example 1.2 solution with multiple optimal solutions.

Figure 1.3: Example 1.1 and Example 1.2 solutions.

1.2 Linear programming

A linear program maximises or minimises a linear objective function with linear inequality or equality constraints.

Definition 1.1 (Linear Program)

A linear program with n variables and m constraints is specified by the following,

- *a linear objective function $f(\boldsymbol{x})$, with a direction of optimization, either maximize or minimize,*
- *for a maximization problem, profit function $f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$, $\boldsymbol{c}$ is called the profit vector,*
- *for a minimization problem, cost function $f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$, $\boldsymbol{c}$ is called the cost vector,*
- *m constraints of the form $\boldsymbol{a}_i\boldsymbol{x} \bowtie_i \boldsymbol{b}_i$, where $\bowtie_i \in \{\leq, \geq, =\}$, $\mathcal{L} = \{\boldsymbol{a}_i\boldsymbol{x} \bowtie_i b_i : \boldsymbol{x} \in \mathbb{R}^n_+, \forall i \in \{1, \ldots, m\}\}$, where $\boldsymbol{a}_i \in \mathbb{R}^n$ is the i-th row vector of the $m \times n$ matrix A.*

♣

The general linear program can be described as,

$$p_{LP} = \max\{\boldsymbol{c}^T\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n_+\} \tag{1.4}$$

where A is an $m \times n$ matrix, $\boldsymbol{c}^T$ is a row vector with n elements, $\boldsymbol{b}$ is a column vector with dimension m and $\boldsymbol{x}$ is a column vector with dimension n.

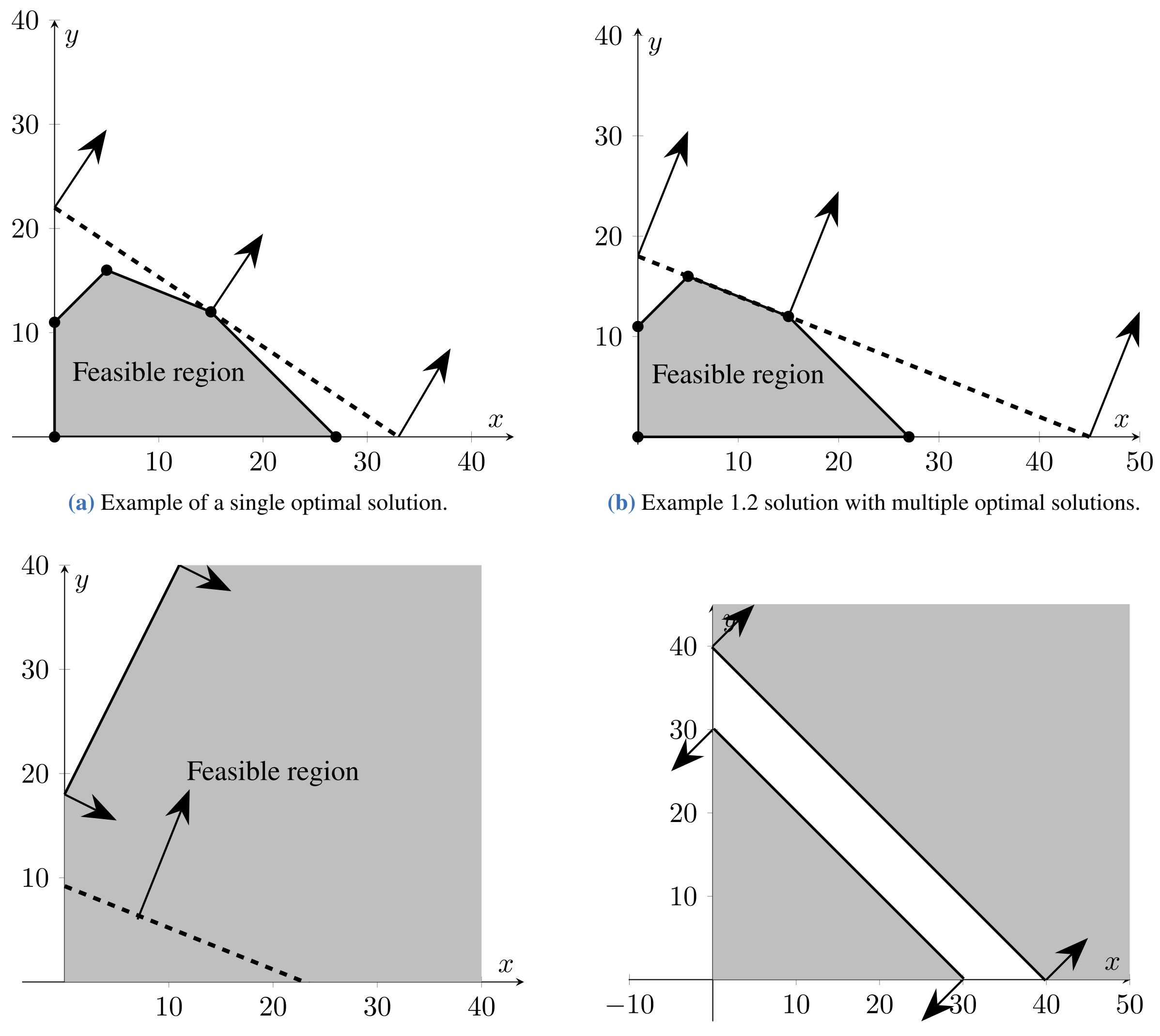
Any linear equality can be replaced with two linear inequalities, e.g.

$$c_1x_1 + c_2x_2 = b_1 \tag{1.5}$$

can be replaced with

$$\begin{aligned} -c_1x_1 - c_2x_2 &\leq -b_1, \\ c_1x_1 + c_2x_2 &\leq b_1, \\ x_1, x_2 &\in \mathbb{R}_+. \end{aligned} \tag{1.6}$$

In Example 1.1 and Example 1.2 we showed linear programs with a single objective value and multiple objective values. Figure 1.4 shows the different Linear Programming (LP) solution outcomes. These examples did not



(a) Example of a single optimal solution.
(b) Example 1.2 solution with multiple optimal solutions.
(c) Example of an unbounded solution.
(d) Example of constraints with no feasible region.

Figure 1.4: Different LP solution outcomes.

solve any specific real world problem, let us consider a more realistic example.

Example 1.3 Problem Formulation Consider the following information on nutrient content of different food types, in Table 1.1.

	Bread	Milk	Cheese	Potato	Fish	Yoghurt
Cost	2.0	3.5	8.0	1.5	11.0	1.0
Protein, g	4.0	8.0	7.0	1.3	8.0	9.2
Fat, g	1.0	5.0	9.0	0.1	7.0	1.0
Carbohydrates, g	15.0	11.7	0.4	22.6	0.0	17.0
Calories	90	120	106	97	130	180

Table 1.1: Diet problem input data.

The optimisation problem is to determine a minimum cost diet that contains at least 300 calories, no more than 10 grams of protein, no less than 10 grams of carbohydrates and no less than 8 grams of fat. In addition, the diet should contain at least 0.5 units of fish and no more than 1 unit of milk.

Step 1: Identify the decision variables

- $x_B \geq 0$ the units of Bread to be included in diet,

- $x_M \geq 0$ the units of Milk to be included in diet,
- $x_C \geq 0$ he units of Cheese to be included in diet,
- $x_P \geq 0$ he units of Potatoes to be included in diet,
- $x_F \geq 0$ he units of Fish to be included in diet,
- $x_Y \geq 0$ he units of Yoghurt to be included in diet,

Step 2: Determine the objective function
The objective is to minimise the total cost, e.g. $2x_B + 3.5x_M + 8x_c + 1.5x_P + 11x_F + x_Y$

Step 3: Formulate the constraints

- At least 300 calories, $90x_B + 120x_M + 106x_c + 97x_P + 130x_F + 180x_Y \geq 300$,
- no more than 10 grams of protein, $4x_B + 8x_M + 7x_c + 1.3x_P + 8x_F + 9.2x_Y \leq 10$,
- no less than 10 grams of carbohydrates, $15x_B + 11.7x_M + 0.4x_c + 22.6x_P + 17x_Y \geq 10$,
- no less than 8 grams of fat, $x_B + 5x_M + 9x_c + 0.1x_P + 7x_F + x_Y \geq 8$,
- at least 0.5 unit of fish, $x_F \geq 0.5$,
- no more than 1 unit of milk, $x_M \leq 1$,

Step 4: Find a solution Use Algorithm 1 to solve the LP.

1.3 Duality

Duality theory applies to general linear programs and explores the relationship between the solutions of paired linear programs. One problem is called the primal, and the other the dual. It does not matter which problem is called the primal, since the dual of a dual is the primal, but for demonstration, we use the symmetrical linear program (1.4) as the primal problem. The dual is then defined as

$$d_{LP} = \min\{\boldsymbol{b}^T\boldsymbol{y} : A^T\boldsymbol{y} \geq c^T, \boldsymbol{y} \in \mathbb{R}_+^m\}, \tag{1.7}$$

where $\boldsymbol{y}$ is a column vector with dimension m.

Example 1.4 Simple dual Find the maximum value in a collection of positive real values, $v_i \in \mathbb{R}_+, \ \forall i \in \mathcal{I}$ as a linear program,

$$\begin{aligned} &\text{minimize } r \\ &\text{subject to } r \geq v_i, \quad \forall i \in \mathcal{I}, \\ &\qquad r \in \mathbb{R}_+. \end{aligned} \tag{1.8}$$

The decision variable r will result in the largest positive real value for all $v_i, \ i \in \mathcal{I}$. The idea of a dual linear program is to use the constraints of the primal to calculate bounds on the objective function. Associate each constraint with a decision variable $y_i, \forall i \in \mathcal{I}$. Multiply each constraint with it's associated decision variable y_i to get

$$y_i r \geq y_i v_i, \quad \forall i \in \mathcal{I}, \tag{1.9}$$

and add them together to get,

$$\sum_{i\in\mathcal{I}} y_i r \geq \sum_{i\in\mathcal{I}} y_i v_i, \tag{1.10}$$

we know that

$$\sum_{i\in\mathcal{I}} y_i v_i \leq \sum_{i\in\mathcal{I}} y_i r \leq r, \tag{1.11}$$

and we want to get $\sum_{i\in\mathcal{I}} y_i v_i$ as close as possible to r, and thus it forms the objective function for the dual, and by eliminating r from the right side of constraint (1.11) we get,

$$\sum_{i \in \mathcal{I}} y_i \leq 1, \tag{1.12}$$

thus the dual of the primal (1.8),

$$\begin{aligned} \text{maximize } & \sum_{i \in \mathcal{I}} y_i v_i \\ \text{subject to } & \sum_{i \in \mathcal{I}} y_i \leq 1, \\ & y_i \in \mathbb{R}_+, \forall i \in \mathcal{I}. \end{aligned} \tag{1.13}$$

Feasible solutions to the dual problem provide an upper-bound to the primal problem p_{LP} and feasible solutions to the primal give lower-bounds to the dual d_{LP}. If the primal has an unbounded optimal value, the dual is infeasible. These properties are described by the following well-known theorem:

Theorem 1.1 (Duality Theorem)

Duality theorem states that:

1. *if the primal has an optimal solution, then so does the dual, and $p_{LP} = d_{LP}$;*
2. *if the primal is unbounded, $p_{LP} = \infty$, then the dual is infeasible;*
3. *if the primal is infeasible, then the dual is either infeasible or unbounded.*

♡

Theorem 1.2 (Complementary Slackness)

Let $\boldsymbol{x}$ be feasible for the primal LP and $\boldsymbol{y}$ be feasible for the dual. Then we have that $\boldsymbol{x}$ and $\boldsymbol{y}$ are optimal solution of their respective LPs if and only if complementary slackness holds:

1. $y_i = 0$, *or* $b_i - \sum_{j=1}^{n} a_{ij} y_i = 0 \qquad \forall i = 1, 2, \ldots, m$, *and*
2. $x_j = 0$, *or* $\sum_{i=1}^{m} a_{ij} x_j - c_j = 0 \qquad \forall j = 1, 2, \ldots, n$,

where a_{ij} is the element in the i-th row and j-th column of the matrix A.

♡

1.4 Simplex method

The idea of the simplex method is to move a basic feasible solution to an adjacent vertex to improve the objective value [3]. If the admissible set is a convex polytope, at least one of the vertices must be an optimal solution, given a linear objective function. A graphical representation of the simplex method is shown in Figure 1.5 to demonstrate the operations of the simplex method.

Let the linear program be in the form $\min\{c\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}_+^n\}$. Denote the set of indices for the basic variables as $\mathcal{B}$ and the set of indices for non-basic variables as $\mathcal{N}$. Add slack variables to each inequality to convert the inequality to an equality, e.g. replace (1.14) with (1.15)

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq b_1 \tag{1.14}$$

$$\begin{aligned} a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + s = b_1 \\ s \geq 0 \end{aligned} \tag{1.15}$$

The linear program is now in the form

$$\min\{\boldsymbol{c}^T \boldsymbol{x} : A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}_+^n\}. \tag{1.16}$$

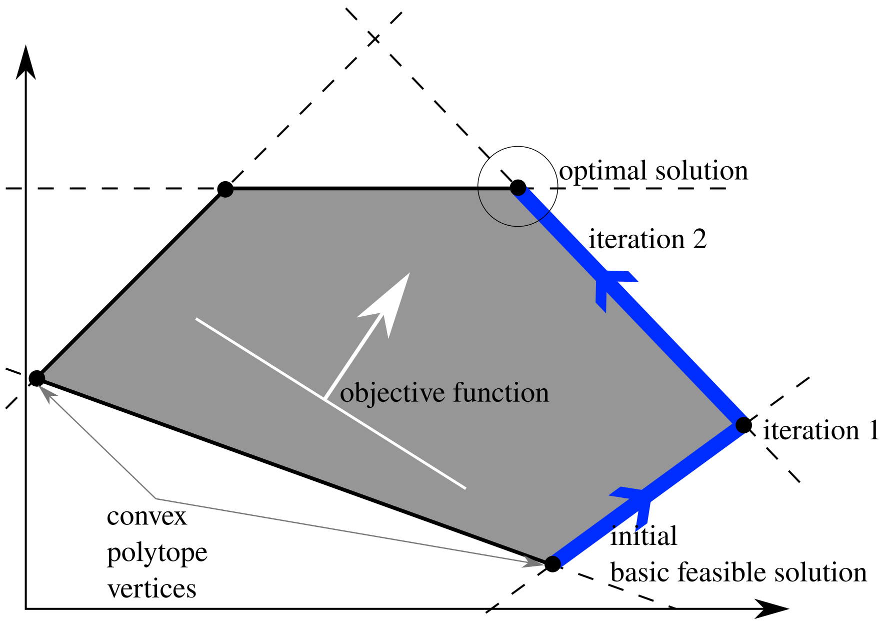


Figure 1.5: Graphical representation of the simplex method.

Since the simplex method moves from one basic feasible solution to another, it requires increasing a non-basic variable. We need to track how such a change affects the basic variables. To understand this effect, rewrite $A\boldsymbol{x} = \boldsymbol{b}$ in basic and non-basic components, e.g.

$$\begin{aligned} A\boldsymbol{x} &= A_{\mathcal{B}}\boldsymbol{x}_{\mathcal{B}} + A_{\mathcal{N}}\boldsymbol{x}_{\mathcal{N}} = \boldsymbol{b} \\ \Leftrightarrow \boldsymbol{x}_{\mathcal{B}} &= A_{\mathcal{B}}^{-1}\boldsymbol{b} - A_{\mathcal{B}}^{-1}A_{\mathcal{N}}\boldsymbol{x}_{\mathcal{N}} \end{aligned} \tag{1.17}$$

Rewrite (1.17) as components of the non-basic variables,

$$\boldsymbol{x}_{\mathcal{B}} = A_{\mathcal{B}}^{-1}\boldsymbol{b} - \sum_{j\in\mathcal{N}} A_{\mathcal{B}}^{-1}A_j x_j. \tag{1.18}$$

To see the effect of a non-basic variable, we derive $\boldsymbol{x}_{\mathcal{B}}$ with respect to each non-basic variable,

$$\frac{\partial \boldsymbol{x}_{\mathcal{B}}}{\partial x_j} = -A_{\mathcal{B}}^{-1}A_j, \qquad \forall j \in \mathcal{N}, \tag{1.19}$$

it is clear that an increase of non-basic variable x_j will decrease the basic variables $\boldsymbol{x}_{\mathcal{B}}$ by the vector $A_{\mathcal{B}}^{-1}A_j$. To improve the objective value, the effect on the objective function needs to be determined, let the objective function from (1.16) be

$$z = \boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{c}_{\mathcal{B}}^T\boldsymbol{x}_{\mathcal{B}} + \boldsymbol{c}_{\mathcal{N}}^T\boldsymbol{x}_{\mathcal{N}}. \tag{1.20}$$

Rewrite (1.20) as a function of non-basic variables $\boldsymbol{x}_{\mathcal{N}}$,

$$\begin{aligned} z(\boldsymbol{x}_{\mathcal{N}}) &= \boldsymbol{c}_{\mathcal{B}}^T(A_{\mathcal{B}}^{-1}\boldsymbol{b} - \sum_{j\in\mathcal{N}} A_{\mathcal{B}}^{-1}A_j\boldsymbol{x}_j) + \boldsymbol{c}_{\mathcal{N}}^T\boldsymbol{x}_{\mathcal{N}} \\ \Leftrightarrow z(\boldsymbol{x}_{\mathcal{N}}) &= \boldsymbol{c}_{\mathcal{B}}^T A_{\mathcal{B}}^{-1}\boldsymbol{b} - \sum_{j\in\mathcal{N}} (c_j - \boldsymbol{c}_{\mathcal{B}}^T A_{\mathcal{B}}^{-1}A_j)x_j. \end{aligned} \tag{1.21}$$

To determine the rate at which the objective function will change for a given non-basic variable x_j, we derive the objective function z with respect to each non-basic variable x_j,

$$r_j = \frac{\partial z}{\partial x_j} = c_j - \boldsymbol{c}_{\mathcal{B}}^T A_{\mathcal{B}}^{-1}A_j, \qquad \forall j \in \mathcal{N}, \tag{1.22}$$

this is known as the reduced cost r_j. Since we are minimising, one of the non-basic variables with a negative reduced cost is increased, this variable enters the basis. When all the reduced cost associated with non-basic variables are greater or equal to zero, the current basic feasible solution is optimal since the objective value can no longer improve. The value of the chosen non-basic variable is increased as long as all basic variables stay non-negative. From equation (1.19) it is obvious that an increase in x_j, will decrease $\boldsymbol{x}_{\mathcal{B}}$ at a rate of

$$\boldsymbol{d}^{\boldsymbol{j}}_{\boldsymbol{\mathcal{B}}} = A_{\mathcal{B}}^{-1}A_j. \tag{1.23}$$

When $d_i^j < 0$, the basic variable x_i will remain non-negative. From equation (1.18) we see if d_i^j is positive, x_i is only non-negative when

$$x_j \leq \frac{(A_{\mathcal{B}}^{-1}\boldsymbol{b})_i}{d_i^j}, \tag{1.24}$$

this suggests that the value of the non-basic variable entering the basis should be

$$x_j = \min\left\{\frac{x_i}{d_i^j} : i \in \mathcal{B},\ d_i^j > 0\right\}. \tag{1.25}$$

Pseudocode for the simplex method is given in algorithm 1, where the input is a standard linear program, and the output may be an optimal solution, an infeasible flag or an unbounded flag. When at least one variable

Algorithm 1 Pseudocode for the Simplex method [3].

```
Input: A standard linear program.
Output: Maybe an optimal solution.
 1: Initialise with a basic feasible solution $\boldsymbol{x}^0$ with $A_{\mathcal{B}}$ the associated basis.
 2: if no basic feasible solution exists then
 3:     return infeasible
 4: end if
 5: while basic feasible solution is not optimal do
 6:     for all $j \in N$ do
 7:         $r_j \leftarrow c_j - \boldsymbol{c}_{\mathcal{B}}^T A_{\mathcal{B}}^{-1} A_j$
 8:     end for
 9:     if $r_j \geq 0, \forall j \in \mathcal{N}$ then
10:         return current basic feasible solution (optimal).
11:     else
12:         for all $j \in \mathcal{N}$ do
13:             if $r_j < 0$ then
14:                 $\boldsymbol{d}_{\mathcal{B}}^j \leftarrow \boldsymbol{A}_{\mathcal{B}}^{-1} A_j$
15:             end if
16:         end for
17:     end if
18:     if $\boldsymbol{d}_{\mathcal{B}}^j \leq \boldsymbol{0},\ \forall r_j < 0,\ j \in \mathcal{N}$ then
19:         return unbounded
20:     else
21:         for all $i \in B$ do
22:             $x_i \leftarrow (A_{\mathcal{B}}^{-1}\boldsymbol{b})_i$
23:         end for
24:         $x_j \leftarrow \min\{\frac{x_i}{d_i^j} : d_i^j > 0\}$
25:         let $x_j$ enter the basis and the corresponding $x_i$ exit the basis.
26:     end if
27: end while
28: return $\boldsymbol{x}$
```

in the basic feasible solution is zero, the solution is degenerate and the next iteration may not improve the objective value. When a basic feasible solution is not degenerate, the objective value strictly improves the next iteration. When the same basic feasible solution occurs more than once, the simplex method will cycle, and fail to terminate. Multiple methods exist to prevent cycling, such as Bland's rule [1] and the criss-cross method [7, 11]. Other methods avoid the appearance of degenerate solutions by adding small positive constant values to the right hand side, so basic feasible solutions are never zero [9].

Chapter 1 Exercise

1. A company manufactures two products (A and B) and the profit per unit sold is £3 and £5 respectively. Each product has to be assembled on a particular machine, each unit of product A taking 12 minutes of assembly time and each unit of product B 25 minutes of assembly time. The company estimates that the machine used for assembly has an effective working week of only 30 hours (due to maintenance/breakdown). Technological constraints mean that for every five units of product A produced at least two units of product B must be produced. Formulate the problem of how much of each product to produce as a linear program.
2. A company makes two products (X and Y) using two machines (A and B). Each unit of X that is produced requires 50 minutes processing time on machine A and 30 minutes processing time on machine B. Each unit of Y that is produced requires 24 minutes processing time on machine A and 33 minutes processing time on machine B. At the start of the current week there are 30 units of X and 90 units of Y in stock. Available processing time on machine A is forecast to be 40 hours and on machine B is forecast to be 35 hours. The demand for X in the current week is forecast to be 75 units and for Y is forecast to be 95 units. Company policy is to maximise the combined sum of the units of X and the units of Y in stock at the end of the week. Formulate the problem of deciding how much of each product to make in the current week as a linear program.
3. A farmer can grow Wheat, Corn or Beans on his 500 acres farm. He requires 200 tons of wheat and 240 tons of corn to feed his cattle. The costs are \$150/acre for planting wheat, \$230/acre for planting corn and \$260/acre for planting beans. Any production in excess of these amounts can be sold for \$170/ton (wheat) and \$150/ton (corn). Any shortfall must be bought from the wholesaler at a cost of \$238/ton (wheat) and \$210/ton (corn). The farmer can also grow beans which may sell at \$36/ton for the first 6000 tons. Beans in excess of 6000 tons can only be sold at \$10/ton, due to economic quotas. If the expected yield is 2.5 tons/acre for wheat, 3 tons/acre for corn and 20 tons/acre for beans, write down the linear programming problem which determines how many acres of each the farmer has to plant in order to maximise his profit.
4. A manufacturer of plastics is planning to blend a new product from four chemical compounds. These compounds are mainly composed of three elements: A, B, and C. The composition and unit cost of these chemicals are shown in Table 1.2. The new product consists of 20 percent element A, at least 30 percent

Chemical Compound	1	2	3	4
Percentage A	30	20	40	20
Percentage B	20	60	30	40
Percentage C	40	15	25	30
Cost/Kilogram	20	30	20	15

Table 1.2: Chemical composition dataset

element B, and at least 20 percent element C. Owing to side effects of compounds 1 and 2, they must not exceed 30 percent and 40 percent of the content of the new product, respectively. Formulate the problem of finding the least costly way of blending as a linear programming problem.

5. The demand for two products in each of the last four weeks is shown in Table 1.3. Apply exponential

Demand	Week 1	Week 2	Week 3	Week 4
Product 1	23	27	34	40
Product 2	11	13	15	14

Table 1.3: Chemical composition dataset

smoothing with a smoothing constant of 0.7 to generate a forecast for the demand for these products in week 5. These products are produced using two machines, X and Y. Each unit of product 1 that is produced requires 15 minutes processing on machine X and 25 minutes processing on machine Y. Each unit of product 2 that is produced requires 7 minutes processing on machine X and 45 minutes processing on machine Y. The available time on machine X in week 5 is forecast to be 20 hours and on machine Y in week 5 is forecast to be 15 hours. Each unit of product 1 sold in week 5 gives a contribution to profit of £10 and each unit of product 2 sold in week 5 gives a contribution to profit of £4. It may not be possible to produce enough to meet your forecast demand for these products in week 5 and each unit of unsatisfied demand for product 1 costs £3, each unit of unsatisfied demand for product 2 costs £1. Formulate the problem of deciding how much of each product to make in week 5 as a linear program.

Chapter 2 Graph Theory

Introduction

- *Graph Definitions*
- *Describing problems as graphs*
- *Using a graph in a linear program*
- *Implementing Dijkstra's shortest path*

A transportation network can be modelled as a graph, this allows the use of graph algorithms such as single-source shortest path algorithms [12]. A graph includes points called *vertices* or *nodes* and the lines connecting them are called *edges*. A weight is usually associated with each edge and for the shortest path problem it represents the length of an edge.

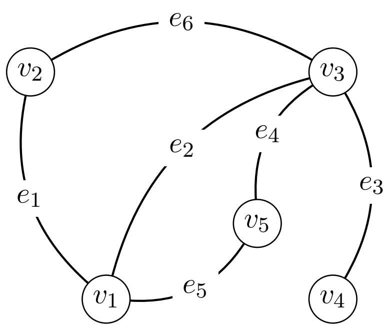
Definition 2.1

*A **graph** $\mathcal{G}$ is composed of two finite sets, a set of **vertices** $\mathcal{V}$, and a set of connecting lines $\mathcal{E}$ called **edges**, such that each edge connects two vertices. A graph is written as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.* ♣

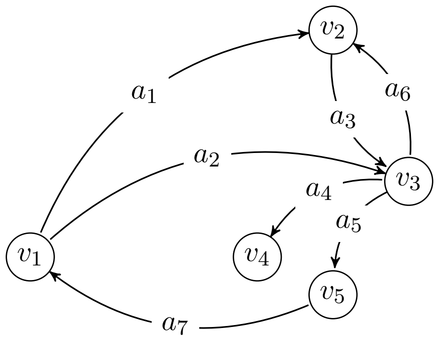
Definition 2.2

*A **digraph** $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ is a graph where each arc $a = (v, w) \in \mathcal{A}$ has a direction from vertex v to w with $v \in V$ and $w \in \mathcal{V}$. **Arcs** are used instead of **edges** when using **digraphs**.* ♣

Example 2.1 Figure 2.1a shows an example of a graph with 5 vertices and 6 edges, every edge in the graph connects two vertices for example, e_6 connects v_2 and v_3. Figure 2.1b shows an example of a digraph with 5 vertices and 7 arcs, every arc in the graph connects two vertices in a directed fashion for example, a_7 connects v_5 and v_1.



(a) Graph with 5 vertices and 6 edges.



(b) Digraph with 5 vertices and 7 arcs.

Figure 2.1: Graph and Digraph.

Definition 2.3

*A **tree** is an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where any two vertices are connected by exactly one path.* ♣

Example 2.2 There exists only one path between any two vertices in Figure 2.2 , e.g. the path between v_1 and v_2 is the sequence of edges e_4, e_2, e_3.

Definition 2.4

*The **degree** of a vertex in a graph is the number of edges that contain a vertex in the pair $(a, b) \in \mathcal{E}$.* ♣

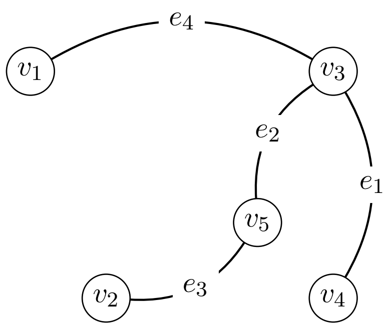
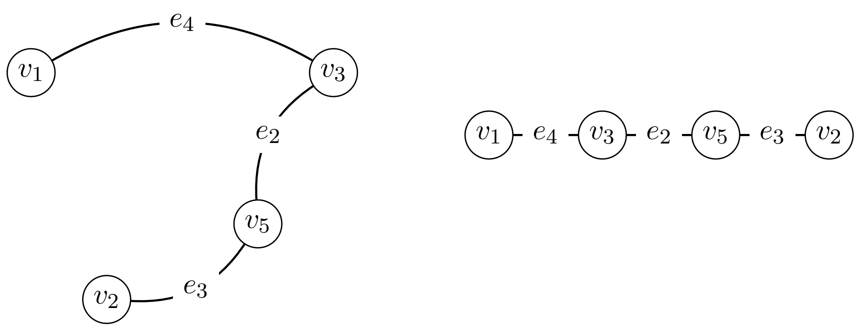


Figure 2.2: Example of a tree graph.

Definition 2.5

*The **path graph** $\mathcal{P}_n$ is a tree with two nodes of vertex degree one, and the other $n-2$ nodes of vertex degree two.* ♣

Example 2.3 Figure 2.3a shows an example of a path graph that corresponds with the path between v_1 and v_2 of Figure 2.2. A path graph is a graph that can be drawn so that all of its vertices and edges lie on a single straight line [8]. Figure 2.3b is an example where the path graph in Figure 2.3a is drawn in a straight line.



(a) Example of a path graph. **(b)** Vertices rearranged in a line.

Figure 2.3: Path graph (left) rearranged into a horizontal line (right).

Definition 2.6

*A **directed** path is a finite sequence of edges directed in the same direction which joins a sequence of vertices, where all vertices and edges are distinct.* ♣

Definition 2.7

*Two subgraphs are edge **disjoint** if they share no edges, and vertex disjointed if they share no vertices.* ♣

2.1 Mathematical Representation

A graph can be represented with a $|\mathcal{V}| \times |\mathcal{V}|$ matrix M, where $\mathcal{V} = \{1, 2, \ldots, n\}$ is a set of vertices of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. Each non-zero entry in matrix M is an arc (r, c) where r and c corresponding to each row and each column in M, e.g.

$$M = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} & a_n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix},$$

where $a_{r,c} = 1$ is (r, c) is in $\mathcal{A}$, otherwise $a_{r,c} = 0$.

2.2 Transportation problem

Consider m origin points, where origin i has a supply of s_i units of a particular item, and n destination points, where destination j requires d_j units of an item. The objective of the problem is to determine the minimum transportation cost by taking into account the unit cost c_{ij} of shipping an item from origin i to destination j.

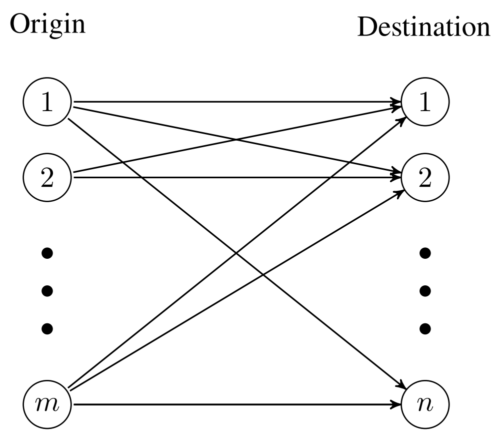


Figure 2.4: Graph representation of the transportation problem.

Minimise

$$\sum_{j\in D}\sum_{i\in S} x_{i,j}c_{i,j}; \tag{2.1}$$

subject to

$$\sum_{j\in D} x_{i,j} \leq s_i, \forall i \in S, \tag{2.2}$$

$$\sum_{i\in S} x_{i,j} = d_j, \forall j \in D, \tag{2.3}$$

$$x_{i,j} \in \mathbb{R}_+, \forall i \in S, \forall j \in D. \tag{2.4}$$

Example 2.4 Transportation problem Consider the transportation problem with the input data in Table 2.1, and implement it in an modelling environment with a capable solver. The model implemented on the elytica platform (with python) is given in Listing 2.1 with the resulting output in Listing 2.2. The steps to implement and evaluate the problem is as follow:

1. Add `import elytica` on top of the script,
2. create an empty model with the `model "transportation"` on one line and `end` on the next,

		Destination			
		1	2	3	s_i
Origin	1	$c_{11} = 4$	$c_{12} = 7$	$c_{13} = 5$	30
	2	$c_{21} = 2$	$c_{22} = 4$	$c_{23} = 3$	20
	d_j	15	10	25	

Table 2.1: Input data for the transportation problem.

3. define the data for the model using either the `set` or `const` keywords between the model definition, e.g. `set D = {1, 2, 3};` and `const d = {15, 10, 25}, forall j in D;`,
4. create the decision variables with keyword `var` and specify their domain, e.g. `var 0 <= x <= 1000, forall i in D;`
5. implement the model with the keywords, `min`, `max`, `constr`, e.g.
   `min sum_{j in D}{sum_{i in S}{x_{i,j} * c_{i,j}}};`
   `constr sum_{j in D}{x_{i, j}} <= s_{i}, forall i in S;`
   `constr sum_{i in S}{x_{i, j}} = d_{j}, forall j in D;`
   **using $\sum_{i \in I}$ with the modelling language has the notation:
   `sum_{i in I}{ <some linear expression related to the data> }`
6. create a main function with `def main():` then `init` and `run` the model, make sure `elytica.init("transportation")` is called before `elytica.run("transportation")` to interpret and initialise the model,
7. print the resulting variables to the console with:
   `elytica.get_variable_value("transportation', "x1,1"),`
   replacing `x1,1` with the desired variable.

Listing 2.1: elytica implementation of the Transportation Problem

```
model transportation
  set D = {1, 2, 3};
  set S = {1, 2};
  const d = {15, 10, 25}, forall j in D;
  const s = {30, 20}, forall i in S;
  const c = {{4, 7 ,5},{2, 4, 3}}, forall i in S, forall j in D;
  var 0<=x<=infty, forall i in S, forall j in D;
  min sum_{j in D}{sum_{i in S}{x_{i,j} * c_{i,j}}};
  constr sum_{j in D}{x_{i, j}} <= s_{i}, forall i in S;
  constr sum_{i in S}{x_{i, j}} = d_{j}, forall j in D;
end

import elytica

def main():
  elytica.init_model("transportation")
  elytica.run_model("transportation")
  print("x_{1,1}=" + str(elytica.get_variable_value("transportation", "x1,1")));
  print("x_{1,2}=" + str(elytica.get_variable_value("transportation", "x1,2")));
  print("x_{1,3}=" + str(elytica.get_variable_value("transportation", "x1,3")));
  print("x_{2,1}=" + str(elytica.get_variable_value("transportation", "x2,1")));
  print("x_{2,2}=" + str(elytica.get_variable_value("transportation", "x2,2")));
  print("x_{2,3}=" + str(elytica.get_variable_value("transportation", "x2,3")));
  return 0
```

Listing 2.2: Output for the elytica implementation of the Transportation Problem

```
x_{1,1}=15.0
x_{1,2}=0.0
x_{1,3}=15.0
x_{2,1}=0.0
x_{2,2}=10.0
x_{2,3}=10.0
```

2.3 Shortest path

One of the most common shortest path algorithm used is Dijkstra's Algorithm [4]. Using abstract data types, such as minimum priority queues, can lead to improved computation time [2, 5, 6]. A minimum priority queue is an abstract data-type that consists of 3 basic operations: get minimum value from the queue, add to queue with priority, and decrease the priority of an element in the queue [10]. These operations are added to Dijkstra algorithm in Algorithm 2 with Q, the priority queue.

The input for Algorithm 2 is a finite weighted graph, a source node s and a target node t. The output is the minimum distance, and the predecessor node of each explored node. Once a node is considered by the algorithm as in line 11, the minimum distance in the set D is updated. The variable d_v presents the distance to the node v from the source node. The variable p_v represent the predecessor vertex of node v from the source node s.

The source node s gets the distance to itself as zero, which line 1 describes. Initially, the queue of vertices the algorithm needs to explore is empty. The algorithm adds each vertex in V to the queue Q. The priority of each vertex in the queue is initially ∞, except the source node s which is first in the queue. The initialisation of the queue is done in lines 3 - 9.

As long as the queue Q is not empty, the first element with minimum priority is removed and explored. The priority of the adjacent vertices of the selected element in the priority queue is updated, and the process repeats until the queue is empty or the algorithm reaches the target vertex.

2.3.1 Linear programming formulation

To formulate the shortest path problem as a linear program, there are two well-know approaches, a flow conservation and path-based formulation. The flow conservation strongly relates to Kirchhoff's current law, where the sum of incoming flows are equal to the sum of all outgoing flows. For the shortest path problem we want a directed path graph as a result and not a Hamiltonian circuit, therefore the source and target vertices should create and absorb flow respectively. We need to make sure that flows are conserved for all the vertices except the source and target vertex, mathematically we can describe it as,

$$\sum_{a\in\sigma(i)} f_a - \sum_{a\in\delta(i)} f_a = 0,\ \forall i \in \mathcal{V} \setminus \{s,t\}, \tag{2.5}$$

where the graph $G = (\mathcal{V}, \mathcal{A})$, $s, t \in \mathcal{V}$, s is the source vertex, t is the target vertex, $\sigma(i)$ is a function that returns the set of all incoming arcs for vertex i and $\delta(i)$ is a function that returns the set of all outgoing arcs for vertex i.

Algorithm 2 Dijkstra's algorithm with min-priority queues.

```
Input:  Finite weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, source node $s$, target node $t$
Output: $D$ a set of minimum distances, $P$ a set of predecessor nodes
 1: $d_s \leftarrow 0$
 2: $Q \leftarrow \emptyset$
 3: for each $v \in V$ do
 4:     if $v \neq s$ then
 5:         $d_v \leftarrow \infty$
 6:         $p_v \leftarrow \varnothing$
 7:     end if
 8:     Add $v$ to $Q$ with priority $d_v$
 9: end for
10: while $Q$ is not empty do
11:     $u \leftarrow$ extract minimum from $Q$
12:     if $u = t$ then
13:         break
14:     end if
15:     for each adjacent vertex $v \in Q$ of $u$ do
16:         $a \leftarrow d_u + w(e = (u, v))$
17:         if $a < d_v$ then
18:             $d_v \leftarrow a$
19:             $p_v \leftarrow u$
20:             Decrease priority of $v$ in $Q$ to $a$
21:         end if
22:     end for
23: end while
24: $D \leftarrow D \bigcup d_v \qquad \forall v \in V$
25: $P \leftarrow P \bigcup p_v \qquad \forall v \in V$
```

The flow variables $f_a \in \mathbb{R}_+, \forall a \in \mathcal{A}$ are used to describe the resulting path. For Equation (2.5) a valid solution is to set all the flows to zero, thus we need to account for the source and target vertices. The source vertex must create an outgoing flow,

$$\sum_{a \in \sigma(s)} f_a = 1, \tag{2.6}$$

and the target vertex must absorb and incoming flow,

$$\sum_{a \in \delta(t)} f_a = 1. \tag{2.7}$$

Equation (2.5) - (2.7) allow the resulting flows to relate to a directed path graph, but the length of the path is arbitrary and there can exist multiple solutions. Next we want to get determine which combination of flows minimise the path length of the resulting directed path graph and adhere to Equation (2.5) - (2.7),

$$\begin{aligned} \text{minimise} \quad & \sum_{a\in\mathcal{A}} f_a \\ \text{subject to} \quad & \sum_{a\in\sigma(i)} f_a - \sum_{a\in\delta(i)} f_a = 0, \ \forall i \in \mathcal{V}\setminus\{s,t\}, \\ & \sum_{a\in\sigma(s)} f_a = 1, \\ & \sum_{a\in\delta(t)} f_a = 1, \\ & f_a \in \mathbb{R}_+, \forall a \in \mathcal{A}. \end{aligned} \tag{2.8}$$

✍ **Exercise 2.1** Determine the shortest path between v_2 and v_4 using a flow conservation model, with $G = (\{v_1, v_2, v_3, v_4, v_5, v_6\}, \{(v_2, v_1), (v_1, v_6), (v_6, v_5), (v_2, v_6), (v_4, v_2), (v_5, v_3), (v_3, v_4)\})$. Represent G and the resulting path graph graphically and write the complete mathematical model out.

Chapter 2 Exercise

1. Will the shortest path flow conservation model with incoming target and outgoing source flows as one always result in an integer solution when using the Simplex method? Explain your answer.
2. What properties does a directed path graph have?
3. Implement the shortest path flow conservation model using the elytica modelling language using the following data: $G = (V, A) = (\{1, 2, 3, 4, 5\}, \{(1, 2), (1, 3), (2, 3), (3, 2), (3, 4), (3, 5), (4, 5), (5, 1)\})$ with $s = 1$ and $t = 5$, and draw the graph to verify the results. Assume the weight of all arcs are 1.
4. How will the objective function and constraints change for the shortest path flow conservation model when the weights of arcs are not 1?
5. Implement a model that accounts for weighted arcs and solve the model using the data from Exercise 3. and weights:
   $w_{1,2} = 0.1$
   $w_{1,3} = 4.1$
   $w_{2,3} = 0.45$
   $w_{3,2} = 0.01$
   $w_{3,4} = 1.24$
   $w_{3,5} = 1.01$
   $w_{4,5} = 0.93$
   $w_{5,1} = 0.001$
   Write down the objective value and explain why the solution is different from the model in Exercise 3.

Bibliography

[1] David Avis and Vasek Chvátal. “Notes on Bland’s pivoting rule”. In: *Polyhedral Combinatorics*. Springer, 1978, pp. 24–34.

[2] Gerth Stølting Brodal et al. “Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths”. In: *Scandinavian Workshop on Algorithm Theory*. Springer. 2004, pp. 480–492.

[3] George Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

[4] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[5] Raphael A Finkel and Jon Louis Bentley. “Quad trees a data structure for retrieval on composite keys”. In: *Acta informatica* 4.1 (1974), pp. 1–9.

[6] Michael L Fredman and Robert Endre Tarjan. “Fibonacci heaps and their uses in improved network optimization algorithms”. In: *Journal of the ACM (JACM)* 34.3 (1987), pp. 596–615.

[7] Komei Fukuda and Tamás Terlaky. “Criss-cross methods: A fresh view on pivot algorithms”. In: *Mathematical Programming* 79.1-3 (1997), pp. 369–395.

[8] Jonathan L Gross and Jay Yellen. *Handbook of graph theory*. CRC press, 2004.

[9] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[10] Steven S Skiena. *The algorithm design manual*. Vol. 1. Springer Science & Business Media, 1998.

[11] Tamas Terlaky. “A convergent criss-cross method”. In: *Optimization* 16.5 (1985), pp. 683–690.

[12] Douglas Brent West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, 2001.