MATTHIAS GERDTS

OPERATIONS RESEARCH
SoSe 2009

Address of the Author:

Matthias Gerdts

Institut für Mathematik
Universität Würzburg
Am Hubland
97074 Würzburg

E-Mail: gerdts@mathematik.uni-wuerzburg.de
WWW: www.mathematik.uni-wuerzburg.de/∼gerdts

Preliminary Version: July 23, 2009

Contents

Lecture plan

Lectures:

Date	Hours	Pages
21.4.2009	15:15-16:45	VL, 2–6, 13
23.4.2009	13:30-15:00	VL, 7–9, 15–23
28.4.2009	15:15-16:45	Üb,
30.4.2009	13:30-15:00	VL, 24–31
05.5.2009	15:15-16:45	VL, 31–36
07.5.2009	13:30-15:00	VL, 36–40
12.5.2009	15:15-16:45	VL, 41–46
14.5.2009	13:30-15:00	Üb,
19.5.2009	15:15-16:45	VL, 46–52,
21.5.2009	13:30-15:00	frei
26.5.2009	15:15-16:45	VL, 52–59,
28.5.2009	13:30-15:00	VL, 59–64
02.6.2009	15:15-16:00	frei
04.6.2009	13:30-15:00	Üb
09.6.2009	15:15-16:45	VL, 64–71
11.6.2009	13:30-15:00	frei
16.6.2009	15:15-16:45	Üb,
18.6.2009	13:30-15:00	VL, 71–79
23.6.2009	15:15-16:45	VL, 79–86
25.6.2009	13:30-15:00	VL, 87–94
30.6.2009	15:15-16:45	Üb,
02.7.2009	13:30-15:00	VL, 95–102
07.7.2009	15:15-16:45	VL, 103–110
09.7.2009	13:30-15:00	VL, 110–114
14.7.2009	15:15-16:45	Üb,
16.7.2009	13:30-15:00	VL, 115–120, 121–124 ausgelassen
21.7.2009	15:15-16:45	VL, 125–131
23.7.2009	13:30-15:00	VL, 131–137

Chapter 1

Introduction

This course is about operations research. Operations research is concerned with all kinds of problems related to decision making, for instance in the operation of companies, public administration, or transportation networks. Particularly, operations research addresses modeling issues, qualitative and quantitative analysis, and algorithmic solution of decision making problems. The main fields of application are analysis and optimisation of networks in companies, transportation, economy, and engineering. Common questions are how to minimise operational costs or how to maximise profit subject to constraints on budget or resources.

Naturally, operations research consists of very many different problem classes, mathematical theories and algorithms for solving such problems. In this lecture we can only discuss a collection of problems from the vast area of operations research.

In particular, we will discuss

- linear optimisation problems (linear programming)
- integer linear programming
- network optimisation problems (shortest path problems, maximum flow problems, network flow problems)
- dynamic optimisation problems (dynamic programming)

Beyond these problem classes there are many more fields that belong to operations research, among them are nonlinear optimisation (nonlinear programming), combinatorial optimisation, games, optimal control. All these topics involve optimisation. The knowledge of an optimal solution with regard to some optimality measure may be used to support the decision maker to find a good strategy for a given problem.

Optimisation plays a major role in many applications from economics, operations research, industry, engineering sciences, and natural sciences. In the famous book of Nocedal and Wright [NW99] we find the following statement:

People optimise: Airline companies schedule crews and aircraft to minimise cost. Investors seek to create portfolios that avoid risks while achieving a high rate of return. Manufacturers aim for maximising efficiency in the design and operation of their production processes. Nature optimises: Physical systems tend to a state of minimum energy. The molecules in an isolated chemical system react with each other until the total potential energy of their electrons is minimised, Rays of light follow paths that minimise their travel time.

All of the problems discussed in this course eventually lead to an optimisation problem and can be cast in the following very general form.

Definition 1.0.1 (General optimisation problem)
For a given function $f : \mathbb{R}^n \to \mathbb{R}$ and a given set $M \subseteq \mathbb{R}^n$ find $\hat{x} \in M$ which minimises f on M, that is $\hat{x}$ satisfies

$$f(\hat{x}) \leq f(x) \qquad \textit{for every } x \in M.$$

We need some terminology that will be used throughout this course:

- f is called objective function.
- M is called feasible set (or admissible set).
- $\hat{x}$ is called minimiser (or minimum) of f on M.
- Any $x \in M$ is called feasible or admissible.
- The components x_i, $i = 1, \ldots, n$, of the vector x are called optimisation variables. For brevity we likewise call the vector x optimisation variable or simply variable.

Remarks:

- Without loss of generality, it suffices to consider minimisation problems. For, if the task were to maximise f, an equivalent minimisation problem is given by minimising $-f$.
- The function f often is associated with costs.
- The set M is often associated with resources, e.g. available budget, available material in a production, etc.

Typical applications (there are many more!):

- nonlinear programming: portfolio optimisation, shape optimisation, parameter identification, topology optimisation, classification
- linear programming: allocation of resources, transportation problems, transshipment problems, maximum flows
- integer programming: assignment problems, travelling salesman problems, VLSI design, matchings, scheduling, shortest paths, telecommunication networks, public transportation networks
- games: economical behaviour, equilibrium problems, electricity markets
- optimal control: path planning for robots, cars, flight systems, crystal growth, flows, cooling of steel, economical strategies, inventory problems

The following questions will be investigated throughout this course:

- How to model a given problem?
- Existence and uniqueness of optimal solutions?
- How does an optimal solution depend on problem data?
- Which algorithms can be used to compute an optimal solution?
- Which properties do these algorithms possess (finiteness, complexity)?

1.1 Examples

Several typical examples from operations research are discussed. We start with a typucal linear program.

Example 1.1.1 (Maximisation of Profit)
A farmer intends to plant 40 acres with sugar beets and wheat. He can use 2400 pounds and 312 working days to achieve this. For each acre his cultivation costs amount to 40 pounds for sugar beets and to 120 pounds for wheat. For sugar beets he needs 6 working days per acre and for wheat 12 working days per acre. The profit amounts to 100 pounds per acre for sugar beets and to 250 pounds per acre for wheat. Of course, the farmer wants to maximise his profit.
Mathematical formulation: Let x_1 denote the acreage which is used for sugar beets and x_2 those for wheat. Then, the profit amounts to

$$f(x_1, x_2) = 100x_1 + 250x_2.$$

The following restrictions apply:

$$\begin{array}{ll} \textit{maximum size:} & x_1 + x_2 \leq 40 \\ \textit{money:} & 40x_1 + 120x_2 \leq 2400 \\ \textit{working days:} & 6x_1 + 12x_2 \leq 312 \\ \textit{no negative acreage:} & x_1, x_2 \geq 0 \end{array}$$

In matrix notation we obtain

$$\max \quad c^\top x \quad s.t. \quad Ax \leq b,\ x \geq 0,$$

where

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad c = \begin{pmatrix} 100 \\ 250 \end{pmatrix}, \quad b = \begin{pmatrix} 40 \\ 2400 \\ 312 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 1 \\ 40 & 120 \\ 6 & 12 \end{pmatrix}.$$

Example 1.1.2 (The Diet Problem by G.J.Stigler, 1940ies)

A human being needs vitamins S1, S2, and S3 for healthy living. Currently, only 4 medications A1,A2,A3,A4 contain these substances:

	S1	*S2*	*S3*	*cost*
A1	*30*	*10*	*50*	*1500*
A2	*5*	*0*	*3*	*200*
A3	*20*	*10*	*50*	*1200*
A4	*10*	*20*	*30*	*900*
need per day	*10*	*5*	*5.5*	

Task: Find combination of medications that satisfy the need at minimal cost!
Let x_i denote the amount of medication A_i, $i = 1, 2, 3, 4$. The following linear program solves the task:

$$\begin{array}{lrrrrcl} \textit{Minimise} & \multicolumn{4}{l}{1500x_1{+}200x_2{+}1200x_3{+}900x_4} & & \\ \textit{subject to} & 30x_1+ & 5x_2+ & 20x_2+ & 10x_4 & \geq & 10 \\ & 10x_1+ & & 10x_3+ & 20x_4 & \geq & 5 \\ & 50x_1+ & 3x_2+ & 50x_3+ & 30x_4 & \geq & 5.5 \\ & \multicolumn{4}{l}{x_1, x_2, x_3, x_4 \geq 0} & & \end{array}$$

In matrix notation, we obtain

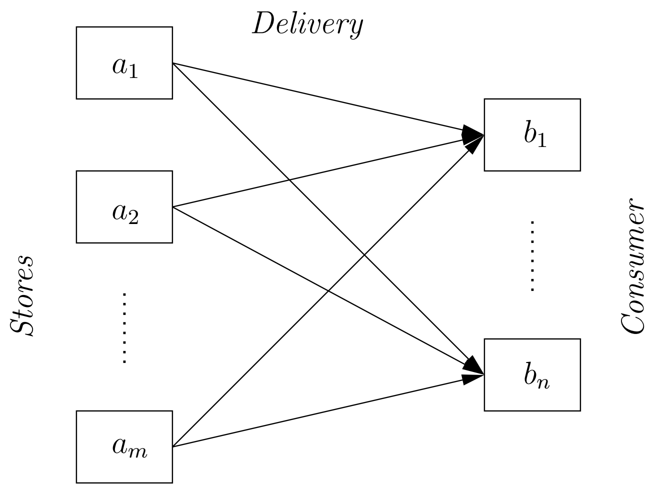
$$\min \quad c^\top x \quad s.t. \quad Ax \geq b,\ x \geq 0,$$

where

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \quad c = \begin{pmatrix} 1500 \\ 200 \\ 1200 \\ 900 \end{pmatrix}, \quad b = \begin{pmatrix} 10 \\ 5 \\ 5.5 \end{pmatrix}, \quad A = \begin{pmatrix} 30 & 5 & 20 & 10 \\ 10 & 0 & 10 & 20 \\ 50 & 3 & 50 & 30 \end{pmatrix}.$$

In the 1940ies nine people spent in total 120 days (!) for the numerical solution of a diet problem with 9 inequalities and 77 variables.

Example 1.1.3 **(Transportation problem)**
A transport company has m stores and wants to deliver a product from these stores to n consumers. The delivery of one item of the product from store i to consumer j costs c_{ij} pound. Store i has stored a_i items of the product. Consumer j has a demand of b_j items of the product. Of course, the company wants to satisfy the demand of all consumers. On the other hand, the company aims at minimising the delivery costs.



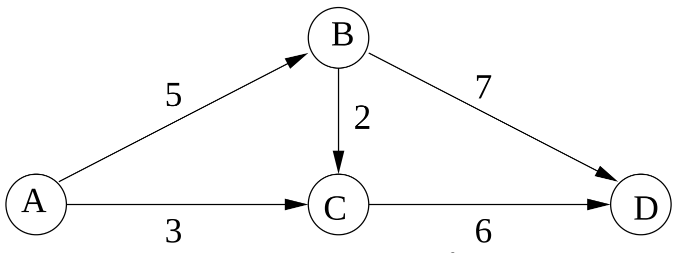
Let x_{ij} denote the amount of products which are delivered from store i to consumer j. In order to find the optimal transport plan, the company has to solve the following linear program:

$$\begin{array}{llll} \text{Minimise} & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} & & \textit{(minimise delivery costs)} \\ \text{s.t.} & \displaystyle\sum_{j=1}^{n} x_{ij} \leq a_i, & i=1,\ldots,m, & \textit{(can't deliver more than it is there)} \\ & \displaystyle\sum_{i=1}^{m} x_{ij} \geq b_j, & j=1,\ldots,n, & \textit{(satisfy demand)} \\ & x_{ij} \geq 0, & i=1,\ldots,m,\ j=1,\ldots,n. & \textit{(can't deliver negative amount)} \end{array}$$

Remark: *In practical problems x_{ij} is often restricted in addition by the constraint $x_{ij} \in \mathbb{N}_0 = \{0,1,2,\ldots\}$, i.e. x_{ij} may only assume integer values. This restriction makes the problem much more complicated and standard methods like the simplex method cannot be applied anymore.*

Example 1.1.4 **(Network problem)**
A company intends to transport as many goods as possible from city A to city D on the

road network below. The figure next to each edge in the network denotes the maximum capacity of the edge.



How can we model this problem as a linear program?
Let $V = \{A, B, C, D\}$ denote the nodes of the network (they correspond to cities). Let $E = \{(A, B), (A, C), (B, C), (B, D), (C, D)\}$ denote the edges of the network (they correspond to roads connecting two cities).
For each edge $(i, j) \in E$ let x_{ij} denote the actual amount of goods transported on edge (i, j) and u_{ij} the maximum capacity. The capacity constraints hold:

$$0 \leq x_{ij} \leq u_{ij} \qquad \forall (i, j) \in E.$$

Moreover, it is reasonable to assume that no goods are lost or no goods are added in cities B and C, i.e. the conservation equations 'outflow - inflow = 0' hold:

$$\begin{aligned} x_{BD} + x_{BC} - x_{AB} &= 0, \\ x_{CD} - x_{AC} - x_{BC} &= 0. \end{aligned}$$

The task is to maximize the amount of goods leaving city A (note that this is the same amount that enters city D):

$$x_{AB} + x_{AC} \to \max .$$

All previous examples are special cases of so-called linear optimisation problems (linear programs, LP), see Definition 2.0.14 below.
Many problems in operations research have a combinatorial character, which means that the feasible set only allows for a finite number of choices, but the number of choices grows very rapidly with increasing problem size. The main goal in this situation is to construct efficient algorithms – if possible at all. However, it will turn out that for certain problems, e.g. traveling salesman problems or knapsack problems, no efficient algorithm is known as yet. Even worse, for these problems an efficient algorithm is unlikely to exist.
What is an efficient algorithm?

Example 1.1.5 (Polynomial and exponential complexity)
Suppose a computer with 10^{10} ops/sec (10 GHz) is given. A problem P (e.g. a linear

program) has to be solved on the computer. Let n denote the size of the problem (e.g. the number of variables). Let five different algorithms be given, which require n, n^2, n^4, 2^n, and $n!$ operations (e.g. steps in the simplex method), respectively, to solve the problem P of size n.

The following table provides an overview on the time which is needed to solve the problem.

ops \ *size n*	*20*	*60*	$\cdots$	*100*	*1000*
n	$0.002\ \mu s$	$0.006\ \mu s$	$\cdots$	$0.01\ \mu s$	$0.1\ \mu s$
n^2	$0.04\ \mu s$	$0.36\ \mu s$	$\cdots$	$1\ \mu s$	$0.1\ ms$
n^4	$16\ \mu s$	$1.296\ ms$	$\cdots$	$10\ ms$	$100\ s$
2^n	$0.1\ ms$	$3\ yrs$	$\cdots$	$10^{12}\ yrs$	$\cdot$
$n!$	$7.7\ yrs$	$\cdot$	$\cdot$	$\cdot$	$\cdot$

Clearly, only the algorithms with polynomial complexity n, n^2, n^4 can be regarded as efficient while those with exponential complexity 2^n and $n!$ are very inefficient.

We don't want to define the complexity or effectiveness of an algorithm in a mathematically sound way. Instead we use the following informal but intuitive definition:

Definition 1.1.6 (Complexity (informal definition))

A method for solving a problem P has polynomial complexity if it requires at most a polynomial (as a function of the size) number of operations for solving any instance of the problem P. A method has exponential complexity, if it has not polynomial complexity.

To indicate the complexity of an algorithm we make use of the $\mathcal{O}$-notation. Let f and g be real-valued functions. We write

$$f(n) = \mathcal{O}(g(n))$$

if there is a constant C such that

$$|f(n)| \leq C|g(n)| \qquad \text{for every } n.$$

In complexity theory the following problem classes play an important role. Again, we don't want to go into details and provide informal definitions only:

- The class $\mathcal{P}$ consists of all problems for which an algorithm with polynomial complexity exists.
- The class $\mathcal{NP}$ (Non-deterministic Polynomial) consists of all problems for which an algorithm with polynomial complexity exists that is able to certify a solution of the problem. In other words: If a solution x of the problem P is given, then the algorithm is able to certify in polynomial time that x is actually a solution of P. On the other hand, if x is not a solution of P then the algorithm may not be able to recognise this in polynomial time.

Clearly, $\mathcal{P} \subseteq \mathcal{NP}$, but it is an open question whether $\mathcal{P} = \mathcal{NP}$ holds or not. If you are the first to answer this question, then the Clay Institute (www.claymath.org) will pay you one million dollars, see http://www.claymath.org/millennium/ for more details. It is conjectured that $\mathcal{P} \neq \mathcal{NP}$. Good luck!

Example 1.1.7 (Assignment Problem)

Given:

- *n employees $E_1, E_2, \ldots, E_n$*
- *n tasks $T_1, T_2, \ldots, T_n$*
- *cost c_{ij} for the assignment of employee E_i to task T_j*

Goal:

Find an assignment of employees to tasks at minimal cost!

Mathematical formulation: Let $x_{ij} \in \{0,1\}$, $i,j = 1,\ldots,n$, be defined as follows:

$$x_{ij} = \begin{cases} 0, & \text{if employee } i \text{ is not assigned to task } j, \\ 1, & \text{if employee } i \text{ is assigned to task } j \end{cases}$$

The assignment problem reads as follows:

$$\begin{aligned} \text{Minimise} \quad & \sum_{i,j=1}^{n} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,\ldots,n, \\ & \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,\ldots,n, \\ & x_{ij} \in \{0,1\}, \qquad i,j = 1,\ldots,n. \end{aligned}$$

As $x_{ij} \in \{0,1\}$, the constraints

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

guarantee that each employee i does exactly one task. Likewise, the constraints

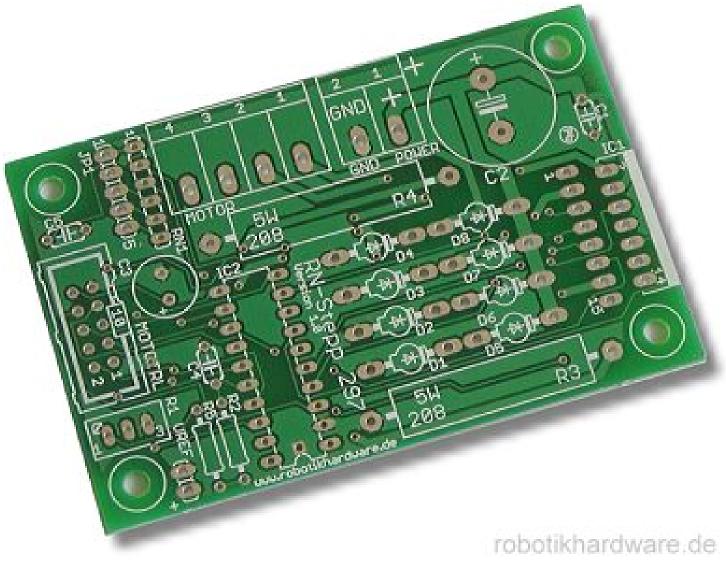
$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,\ldots,n,$$

guarantee that each task j is done by one employee.

From a theoretical point of view, this problem seems to be rather simple as only finitely many possibilities for the choice of x_{ij}, $i, j = 1, \ldots, n$ exist. Hence, in a naive approach one could enumerate all feasible points and choose the best one. How many feasible points are there? Let $x_{ij} \in \{0, 1\}$, $i, j = 1, \ldots, n$, be the entries of an $n \times n$-matrix. The constraints of the assignment problem require that each row and column of this matrix has exactly one non-zero entry. Hence, there are n choices to place the 1 in the first row, $n - 1$ choices to place the 1 in the second row and so on. Thus there are $n! = n(n-1)\cdot(n-2)\cdots 2\cdot 1$ feasible points. The naive enumeration algorithm needs to evaluate the objective function for all feasible points. The following table shows how rapidly the number of evaluations grows:

n	10	20	30	50	70	90
eval	$3.629 \cdot 10^{6}$	$2.433 \cdot 10^{18}$	$2.653 \cdot 10^{32}$	$3.041 \cdot 10^{64}$	$1.198 \cdot 10^{100}$	$1.486 \cdot 10^{138}$

Example 1.1.8 (VLSI design, see Korte and Vygen [KV08])
A company has a machine which drills holes into printed circuit boards.



Since it produces many boards and as time is money, it wants the machine to complete one board as fast as possible. The drilling time itself cannot be influenced, but the time needed to move from one position to another can be minimised by finding an optimal route. The drilling machine can move in horizontal and vertical direction simultaneously. Hence, the time needed to move from a position $p_i = (x_i, y_i)^\top \in \mathbb{R}^2$ to another position $p_j = (x_j, y_j)^\top \in \mathbb{R}^2$ is proportional to the l_∞-distance

$$\|p_i - p_j\|_\infty = \max\{|x_i - x_j|, |y_i - y_j|\}.$$

Given a set of points $p_1, \ldots, p_n \in \mathbb{R}^2$, the task is to find a permutation $\pi : \{1, \ldots, n\} \to$

$\{1, \dots, n\}$ *of the points such that the total distance*

$$\sum_{i=1}^{n-1} \|p_{\pi(i)} - p_{\pi(i+1)}\|$$

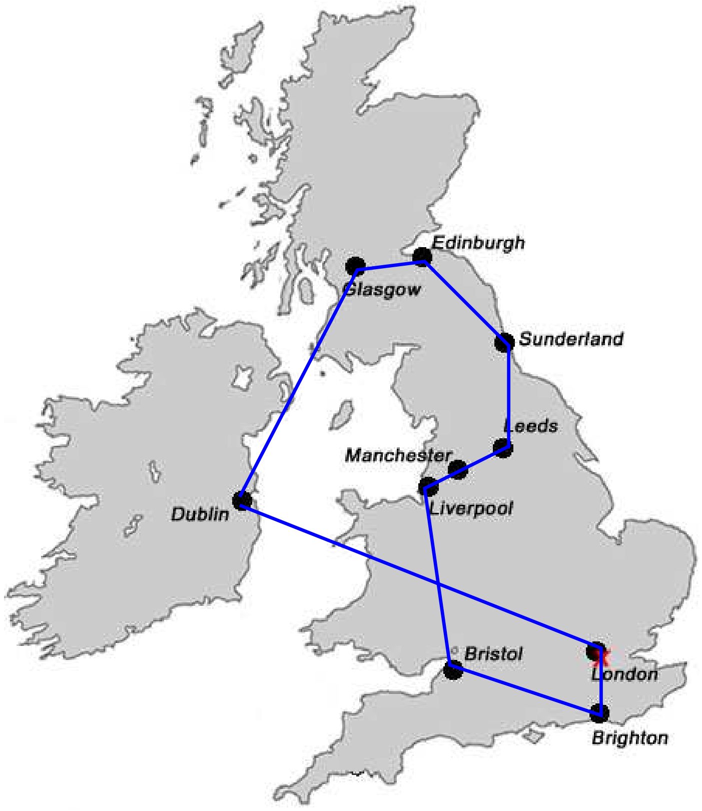
becomes minimal.

Again, the most naive approach to solve this problem would be to enumerate all $n!$ permutations of the set $\{1, \dots, n\}$, to compute the total distance for each permutation and to choose that permutation with the smallest total distance.

Example 1.1.9 **(Traveling salesman problem)**

Let a number of cities $V = \{1, \dots, n\}$ and a number of directed connections between the cities $E \subset V \times V$ be given.

Let c_{ij} denote the length (or time or costs) of the connection $(i,j) \in E$. A tour is a closed directed path which meets each city exactly once. The problem is to find a tour of minimal length.



Let the variables

$$x_{ij} \in \{0,1\}$$

be given with

$$x_{ij} = \begin{cases} 1, & \textit{if } (i,j) \textit{ is part of a tour,} \\ 0 & \textit{otherwise} \end{cases}$$

for every $(i,j) \in E$.
These are finitely many vectors only, but usually very many!
The constraint that every city is met exactly once can be modeled as

$$\sum_{\{i:(i,j)\in E\}} x_{ij} = 1, \qquad j \in V, \tag{1.1}$$
$$\sum_{\{j:(i,j)\in E\}} x_{ij} = 1, \qquad i \in V. \tag{1.2}$$

These constraints don't exclude disconnected sub-tours (actually, these are the constraints of the assignment problem).
Hence, for every disjoint partition of V in non-empty sets

$$\begin{aligned} U &\subset V \\ U^c &\subset V \end{aligned}$$

we postulate: There exists a connection

$$(i,j) \in E \text{ with } i \in U,\ j \in U^c$$

and a connection

$$(k,\ell) \in E \text{ with } k \in U^c,\ \ell \in U\ .$$

The singletons don't have to be considered according to (1.1) and (1.2), respectively. We obtain the additional constraints

$$\sum_{\{(i,j)\in E: i\in U, j\in V\setminus U\}} x_{ij} \geq 1 \tag{1.3}$$

for every $U \subset V$ with $2 \leq |U| \leq |V| - 2$.
The Traveling Salesman Problem reads as:
Minimise

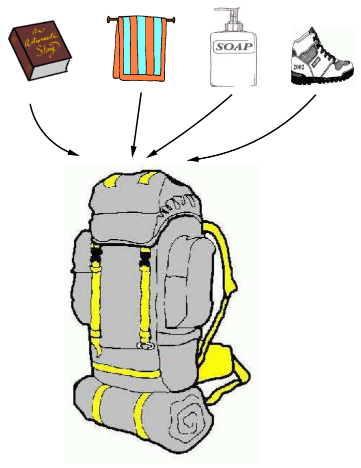
$$\sum_{(i,j)\in E} c_{ij}x_{ij}$$

subject to

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in E,$$

and (1.1), (1.2), (1.3).

Example 1.1.10 (Knapsack Problems)



There is one knapsack and N items. Item j has weight a_j and value c_j for $j = 1, \dots, N$. The task is to create a knapsack with maximal value under the restriction that the maximal weight is less than or equal to A. This leads to the optimization problem

$$\text{Maximise} \sum_{j=1}^{N} c_j x_j \qquad \text{subject to} \qquad \sum_{j=1}^{N} a_j x_j \leq A,\ x_j \in \{0,1\},\ j = 1, \dots, N,$$

where

$$x_j = \begin{cases} 1, & \text{item } j \text{ is put into the knapsack,} \\ 0, & \text{item } j \text{ is not put into the knapsack.} \end{cases}$$

A naive approach is to investigate all 2^N possible combinations.

Example 1.1.11 **(Facility location problem)**

Let a set $I = \{1, \dots, m\}$ of clients and a set $J = \{1, \dots, n\}$ of potential depots be given. Opening a depot $j \in J$ causes fixed maintainance costs c_j.

Each client $i \in I$ has a total demand b_i and every depot $j \in J$ has a capacity u_j. The costs for the transport of one unit from depot $j \in J$ to customer $i \in I$ are denoted by h_{ij}.

The facility location problem aims at minimising the total costs (maintainance and transportation costs) and leads to a mixed-integer linear program:

Minimise

$$\sum_{j \in J} c_j x_j + \sum_{\substack{i \in I \\ j \in J}} h_{ij} y_{ij}$$

subject to the constraints

$$\begin{aligned} x_j &\in \{0,1\}, && j \in J, \\ y_{ij} &\geq 0, && i \in I,\ j \in J, \\ \sum_{j \in J} y_{ij} &= b_i, && i \in I, \\ \sum_{i \in I} y_{ij} &\leq x_j u_j, && j \in J. \end{aligned}$$

Herein, the variable $x_j \in \{0,1\}$ *is used to decide whether depot* $j \in J$ *is opened* ($x_j = 1$) *or not* ($x_j = 0$). *The variable* y_{ij} *denotes the demand of client* i *satisfied from depot* j.

Examples 1.1.7, 1.1.9, and 1.1.10 are integer linear programs (ILP) while Example 1.1.11 is a mixed integer linear program (MILP).

Chapter 2

Linear Programming

The general optimisation problem 1.0.1 is much too general to solve it and one needs to specify f and M. In this chapter we exclusively deal with linear optimisation problems. Linear programming is an important field of optimisation. Many practical problems in operations research can be expressed as linear programming problems. Moreover, a number of algorithms for other types of optimisation problems work with linear programming problems as sub-problems. Historically, ideas from linear programming have inspired many of the central concepts of optimisation theory, such as duality, decomposition, and the importance of convexity and its generalisations. Likewise, linear programming is heavily used in business management, either to maximize the income or to minimize the costs of a production scheme.

What is a linear function?

Definition 2.0.12 (linear function)
A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called linear, if and only if

$$\begin{aligned} f(x+y) &= f(x) + f(y), \\ f(cx) &= cf(x) \end{aligned}$$

hold for every $x, y \in \mathbb{R}^n$ and every $c \in \mathbb{R}$.

Linear vs. nonlinear functions:

Example 2.0.13
The functions

$$\begin{aligned} f_1(x) &= x, \\ f_2(x) &= 3x_1 - 5x_2, \qquad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \\ f_3(x) &= c^\top x, \qquad x, c \in \mathbb{R}^n, \\ f_4(x) &= Ax, \qquad x \in \mathbb{R}^n,\ A \in \mathbb{R}^{m\times n}, \end{aligned}$$

are linear [exercise: prove it!].

The functions

$$\begin{aligned}
f_5(x) &= 1,\\
f_6(x) &= x+1, \qquad x \in \mathbb{R},\\
f_7(x) &= Ax+b, \qquad x \in \mathbb{R}^n,\ A \in \mathbb{R}^{m\times n},\ b \in \mathbb{R}^m,\\
f_8(x) &= x^2,\\
f_9(x) &= \sin(x),\\
f_{10}(x) &= a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0, \quad a_i \in \mathbb{R},\ (a_n,\ldots,a_2,a_0) \neq 0
\end{aligned}$$

are not linear [exercise: prove it!].

Fact:
Every linear function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be expressed in the form $f(x) = Ax$ with some matrix $A \in \mathbb{R}^{m\times n}$.

Notion:
Throughout this course we use the following notion:

- $x \in \mathbb{R}^n$ is the column vector $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ with components $x_1, x_2, \ldots, x_n \in \mathbb{R}$.
- By $x^\top$ we mean the row vector $(x_1, x_2, \ldots, x_n)$. This is called the transposed vector of x.
- $A \in \mathbb{R}^{m\times n}$ is the matrix $\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$ with entries a_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$.
- By $A^\top \in \mathbb{R}^{n\times m}$ we mean the transposed matrix of A, i.e. $\begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$.

This notion allows to characterise linear optimisation problems. A linear optimisation problem is a general optimisation problem in which f is a linear function and the set M is defined by the intersection of finitely many linear equality or inequality constraints. A linear program in its most general form has the following structure:

Definition 2.0.14 **(Linear Program (LP))**

Minimise

$$c^\top x = c_1x_1 + \ldots + c_nx_n = \sum_{i=1}^{n} c_ix_i$$

subject to the constraints

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \quad \left\{ \begin{array}{c} \leq \\ \geq \\ = \end{array} \right\} \quad b_i, \qquad i = 1, \ldots, m.$$

In addition, some (or all) components x_i*,* $i \in I$*, where* $I \subseteq \{1, \ldots, n\}$ *is an index set, may be restricted by sign conditions according to*

$$x_i \left\{ \begin{array}{c} \leq \\ \geq \end{array} \right\} 0, \qquad i \in I.$$

Herein, the quantities

$$c = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{R}^m, \quad A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

are given data.

Remark 2.0.15

- *The relations '*$\leq$*', '*$\geq$*', and '*$=$*' are applied component-wise to vectors, i.e. given two vectors* $x = (x_1, \ldots, x_n)^\top$ *and* $y = (y_1, \ldots, y_n)^\top$ *we have*

$$x \left\{ \begin{array}{c} \leq \\ \geq \\ = \end{array} \right\} y \qquad \Leftrightarrow \qquad x_i \left\{ \begin{array}{c} \leq \\ \geq \\ = \end{array} \right\} y_i, \quad i = 1, \ldots, n.$$

- *Variables* x_i*,* $i \notin I$*, are called free variables, as these variables may assume any real value.*

Example 2.0.16

The LP

$$\begin{array}{lrcrcrcrcr}
\textit{Minimise} & -5x_1 & + & x_2 & + & 2x_3 & + & 2x_4 & & \\
\textit{s.t.} & & & x_2 & - & 4x_3 & & & \leq & 5 \\
 & 2x_1 & + & x_2 & - & x_3 & - & x_4 & \geq & -3 \\
 & x_1 & + & 3x_2 & - & 4x_3 & - & x_4 & = & -1 \\
 & x_1 \leq 0 & , & x_2 \geq 0 & , & & & x_4 \geq 0 & &
\end{array}$$

is a general LP with $I = \{1,2,4\}$ *and*

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix},\ c = \begin{pmatrix} -5 \\ 1 \\ 2 \\ 2 \end{pmatrix},\ b = \begin{pmatrix} 5 \\ -3 \\ -1 \end{pmatrix},\ A = \begin{pmatrix} 0 & 1 & -4 & 0 \\ 2 & 1 & -1 & -1 \\ 1 & 3 & -4 & -1 \end{pmatrix}.$$

The variable x_3 *is free.*

2.1 Transformation to canonical and standard form

LP in its most general form is rather tedious for the construction of algorithms or the derivation of theoretical properties. Therefore, it is desirable to restrict the discussion to certain standard forms of LP into which any arbitrarily structured LP can be transformed. In the sequel we will restrict the discussion to two types of LP: the canonical LP and the standard LP.
We start with the canonical LP.

Definition 2.1.1 (Canonical Linear Program $(\mathrm{LP_C})$**)**
Let $c = (c_1,\dots,c_n)^\top \in \mathbb{R}^n$, $b = (b_1,\dots,b_m)^\top \in \mathbb{R}^m$ *and*

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m\times n}$$

be given. The canonical linear program reads as follows: Find $x = (x_1,\dots,x_n)^\top \in \mathbb{R}^n$ *such that the objective function*

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

becomes minimal subject to the constraints

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \qquad i = 1,\dots,m, \tag{2.1}$$

$$x_j \geq 0, \qquad j = 1,\dots,n. \tag{2.2}$$

In matrix notation:

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax \leq b,\ x \geq 0. \tag{2.3}$$

We need some notation.

Definition 2.1.2 (objective function, feasible set, feasible points, optimality)

(i) The function $f(x) = c^\top x$ is called objective function.

(ii) The set

$$M_C := \{x \in \mathbb{R}^n \mid Ax \leq b,\ x \geq 0\}$$

is called feasible (or admissible) set (of $\mathrm{LP_C}$*).*

(iii) A vector $x \in M_C$ is called feasible (or admissible) (for $\mathrm{LP_C}$*).*

(iv) $\hat{x} \in M_C$ is called optimal (for $\mathrm{LP_C}$*), if*

$$c^\top \hat{x} \leq c^\top x \quad \forall x \in M_C.$$

We now discuss how an arbitrarily structured LP can be transformed into $\mathrm{LP_C}$. The following cases may occur:

(i) Inequality constraints: The constraint

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

can be transformed into a constraint of type (2.1) by multiplication with -1:

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

(ii) Equality constraints: The constraint

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

can be written equivalently as

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

Hence, instead of one equality constraint we obtain two inequality constraints. Using (i) for the inequality on the left we finally obtain two inequality constraints of type (2.1):

$$\begin{aligned} \sum_{j=1}^{n} a_{ij}x_j &\leq b_i, \\ \sum_{j=1}^{n} (-a_{ij})x_j &\leq -b_i. \end{aligned}$$

(iii) Free variables: Any real number x_i can be decomposed into $x_i = x_i^+ - x_i^-$ with nonnegative numbers $x_i^+ \geq 0$ and $x_i^- \geq 0$. Notice, that this decomposition is not unique as for instance $-6 = 0-6 = 1-7 = \ldots$. It will be unique if we postulate that either x_i^+ or x_i^- be zero. But this postulation is not a linear constraint anymore, so we don't postulate it.

In order to transform a given LP into canonical form, every occurrence of the free variable x_i in LP has to be replaced by $x_i^+ - x_i^-$. The constraints $x_i^+ \geq 0$ and $x_i^- \geq 0$ have to be added. Notice, that instead of one variable x_i we now have two variables x_i^+ and x_i^-.

(iv) Maximisation problems: Maximising $c^\top x$ is equivalent to minimising $(-c)^\top x$.

Example 2.1.3

Consider the linear program of maximising

$$-1.2x_1 - 1.8x_2 - x_3$$

subject to the constraints

$$\begin{aligned} x_1 &\geq -\frac{1}{3} \\ x_1 - 2x_3 &\leq 0 \\ x_1 - 2x_2 &\leq 0 \\ x_2 - x_1 &\leq 0 \\ x_3 - 2x_2 &\leq 0 \\ x_1 + x_2 + x_3 &= 1 \\ x_2, x_3 &\geq 0. \end{aligned}$$

This linear program is to be transformed into canonical form (2.3).
The variable x_1 is a free variable. Therefore, according to (iii) we replace it by $x_1 := x_1^+ - x_1^-$, $x_1^+, x_1^- \geq 0$.
According to (ii) the equality constraint $x_1 + x_2 + x_3 = 1$ is replaced by

$$\underbrace{x_1^+ - x_1^-}_{=x_1} + x_2 + x_3 \leq 1 \qquad \text{and} \qquad \underbrace{-x_1^+ + x_1^-}_{=-x_1} - x_2 - x_3 \leq -1.$$

According to (i) the inequality constraint $x_1 = x_1^+ - x_1^- \geq -\frac{1}{3}$ is replaced by

$$-x_1 = -x_1^+ + x_1^- \leq \frac{1}{3}.$$

Finally, we obtain a minimisation problem by multiplying the objective function with -1.

Summarising, we obtain the following canonical linear program:

$$\text{Minimise} \quad (1.2, -1.2, 1.8, 1)\begin{pmatrix} x_1^+ \\ x_1^- \\ x_2 \\ x_3 \end{pmatrix}$$

$$\text{s.t.} \quad \begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & -2 \\ 1 & -1 & -2 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & -2 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_1^+ \\ x_1^- \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 1/3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix}, \qquad \begin{pmatrix} x_1^+ \\ x_1^- \\ x_2 \\ x_3 \end{pmatrix} \geq 0.$$

Theorem 2.1.4

Using the transformation techniques (i)–(iv), we can transform any LP into a canonical linear program (2.3).

The canonical form is particularly useful for visualising the constraints and solving the problem graphically. The feasible set M_C is an intersection of finitely many halfspaces and can be visualised easily for $n = 2$ (and maybe $n = 3$). This will be done in the next section. But for the construction of solution methods, in particular for the simplex method, another notation is preferred: the standard linear program.

Definition 2.1.5 **(Standard Linear Program** $(\mathrm{LP_S})$**)**

Let $c = (c_1, \ldots, c_n)^\top \in \mathbb{R}^n$, $b = (b_1, \ldots, b_m)^\top \in \mathbb{R}^m$ *and*

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

be given. Let $Rank(A) = m$. *The standard linear program reads as follows: Find* $x = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$ *such that the objective function*

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

becomes minimal subject to the constraints

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1,\ldots,m, \tag{2.4}$$

$$x_j \geq 0, \qquad j = 1,\ldots,n. \tag{2.5}$$

In matrix notation:

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b, \; x \geq 0. \tag{2.6}$$

Remark 2.1.6

- *The sole difference between* $\mathrm{LP_C}$ *and* $\mathrm{LP_S}$ *are the constraints* (2.1) *and* (2.4)*, respectively. Don't be confused by the same notation. The quantities* c*,* b *and* A *are* **not** *the same when comparing* $\mathrm{LP_C}$ *and* $\mathrm{LP_S}$*.*
- *The rank assumption Rank*$(A) = m$ *is useful because it avoids linearly dependent constraints. Linearly dependent constraints always can be detected and eliminated in a preprocessing procedure, for instance by using a QR decomposition of* $A^\top$*.*
- $\mathrm{LP_S}$ *is only meaningful if* $m < n$ *holds. Because in the case* $m = n$ *and* A *nonsingular, the equality constraint yields* $x = A^{-1}b$*. Hence,* x *is completely defined and no degrees of freedom remain for optimising* x*. If* $m > n$ *and Rank*$(A) = n$ *the linear equation* $Ax = b$ *possesses at most one solution and again no degrees of freedom remain left for optimisation. Without stating it explicitely, we will assume* $m < n$ *in the sequel whenever we discuss* $\mathrm{LP_S}$*.*

As in Definition 2.1.2 we define the feasible set (of $\mathrm{LP_S}$) to be

$$M_S := \{x \in \mathbb{R}^n \mid Ax = b, \; x \geq 0\}$$

and call $\hat{x} \in M_S$ optimal (for $\mathrm{LP_S}$) if

$$c^\top \hat{x} \leq c^\top x \quad \forall x \in M_S.$$

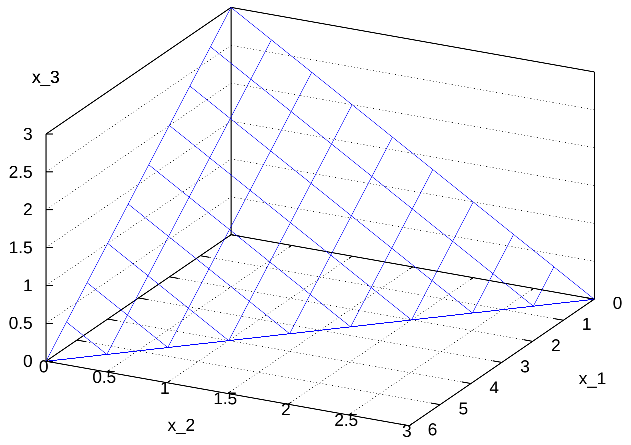
The feasible set M_S is the intersection of the affine subspace $\{x \in \mathbb{R}^n \mid Ax = b\}$ (i.e. intersection of lines in 2D or planes in 3D or hyperplanes in nD) with the nonnegative orthant $\{x \in \mathbb{R}^n \mid x \geq 0\}$. The visualisation is more difficult as for the canonical linear program and a graphical solution method is rather impossible (except for very simple cases).

Example 2.1.7

Consider the linear program

$$\max \quad -90x_1 - 150x_2 \quad s.t. \quad \frac{1}{2}x_1 + x_2 + x_3 = 3, \ x_1, x_2, x_3 \geq 0.$$

The feasible set $M_S = \{(x_1, x_2, x_3)^\top \in \mathbb{R}^3 \mid \frac{1}{2}x_1 + x_2 + x_3 = 3, \ x_1, x_2, x_3 \geq 0\}$ *is the blue triangle below:*



We can transform a canonical linear program into an equivalent standard linear program and vice versa.

(i) Transformation of a canonical problem into a standard problem:
Let a canonical linear program (2.3) be given.

Define slack variables $y = (y_1, \ldots, y_m)^\top \in \mathbb{R}^m$ by

$$y_i := b_i - \sum_{j=1}^{n} a_{ij}x_j, \qquad i = 1, \ldots, m,$$

resp.

$$y = b - Ax.$$

It holds $Ax \leq b$ if and only if $y \geq 0$.

With $\hat{x} := \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^{n+m}$, $\hat{c} := \begin{pmatrix} c \\ 0 \end{pmatrix} \in \mathbb{R}^{n+m}$, $\hat{A} := (A \mid I) \in \mathbb{R}^{m\times(n+m)}$ we obtain a standard linear program of type (2.6):

$$\text{Minimise} \quad \hat{c}^\top \hat{x} \quad \text{s.t.} \quad \hat{A}\hat{x} = b, \ \hat{x} \geq 0. \tag{2.7}$$

Notice, that $\text{Rank}(\hat{A}) = m$ holds automatically.

(ii) Transformation of a standard problem into a canonical problem:
Let a standard linear program (2.6) be given. Rewrite the equality $Ax = b$ as two inequalities $Ax \leq b$ and $-Ax \leq -b$. With

$$\hat{A} := \begin{pmatrix} A \\ -A \end{pmatrix} \in \mathbb{R}^{2m \times n}, \qquad \hat{b} := \begin{pmatrix} b \\ -b \end{pmatrix} \in \mathbb{R}^{2m}$$

a canonical linear program of type (2.3) arises:

$$\text{Minimise} \quad c^\top x \quad \text{s.t.} \quad \hat{A}x \leq \hat{b},\ x \geq 0. \tag{2.8}$$

We obtain

Theorem 2.1.8

Canonical linear programs and standard linear programs are equivalent in the sense that

(i) x solves (2.3) if and only if $\hat{x} = (x, b - Ax)^\top$ solves (2.7).

(ii) x solves (2.6) if and only if x solves (2.8).

Proof: exercise □

2.2 Graphical Solution of LP

We demonstrate the graphical solution of a canonical linear program in the 2-dimensional space.

Example 2.2.1 **(Example 1.1.1 revisited)**

The farmer in Example 1.1.1 aims at solving the following canonical linear program: Minimise

$$f(x_1, x_2) = -100x_1 - 250x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 40 \\ 40x_1 + 120x_2 &\leq 2400 \\ 6x_1 + 12x_2 &\leq 312 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Define the linear functions

$$\begin{aligned} g_1(x_1, x_2) &:= x_1 + x_2, \\ g_2(x_1, x_2) &:= 40x_1 + 120x_2, \\ g_3(x_1, x_2) &:= 6x_1 + 12x_2. \end{aligned}$$

Let us investigate the first constraint $g_1(x_1, x_2) = x_1 + x_2 \leq 40$, cf Figure 2.1. The equation $g_1(x_1, x_2) = x_1 + x_2 = 40$ defines a straight line in $\mathbb{R}^2$ – a so-called hyperplane

$$H := \{(x_1, x_2)^\top \in \mathbb{R}^2 \mid x_1 + x_2 = 40\}.$$

The vector of coefficients $n_{g_1} = (1, 1)^\top$ is the normal vector on the hyperplane H and it points in the direction of increasing values of the function g_1. I.e. moving a point on H into the direction of n_{g_1} leads to a point $(x_1, x_2)^\top$ with $g_1(x_1, x_2) > 40$. Moving a point on H into the opposite direction $-n_{g_1}$ leads to a point $(x_1, x_2)^\top$ with $g_1(x_1, x_2) < 40$. Hence, H separates $\mathbb{R}^2$ into two halfspaces defined by

$$\begin{aligned} H^- &:= \{(x_1, x_2)^\top \in \mathbb{R}^2 \mid x_1 + x_2 \leq 40\}, \\ H^+ &:= \{(x_1, x_2)^\top \in \mathbb{R}^2 \mid x_1 + x_2 > 40\}. \end{aligned}$$

Obviously, points in H^+ are not feasible, while points in H^- are feasible w.r.t. to the constraint $x_1 + x_2 \leq 40$.

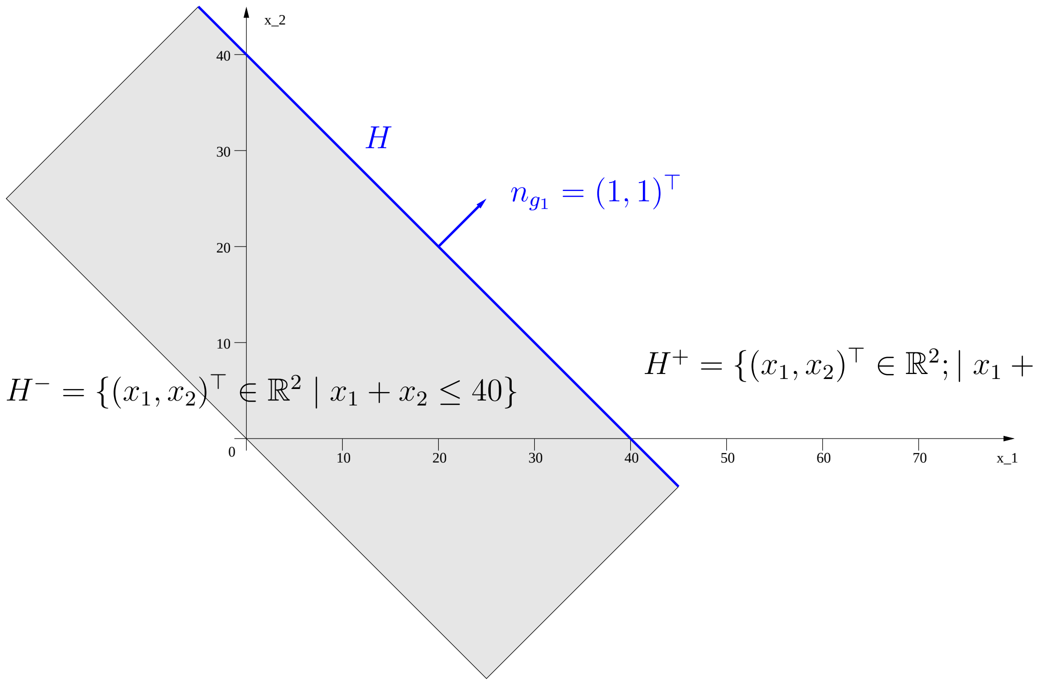

x_2
40
30
20
10
0
10
20
30
40
50
60
70
x_1
H
$n_{g_1} = (1,1)^\top$
$H^+ = \{(x_1,x_2)^\top \in \mathbb{R}^2; \mid x_1 + x_2 > 40\}$
$H^- = \{(x_1,x_2)^\top \in \mathbb{R}^2 \mid x_1 + x_2 \leq 40\}$

Figure 2.1: Geometry of the constraint $g_1(x_1, x_2) = x_1 + x_2 \leq 40$: normal vector $n_{g_1} = (1, 1)^\top$ (the length has been scaled), hyperplane H and halfspaces H^+ and H^-. All points in H^- are feasible w.r.t. to this constraint

The same discussion can be performed for the remaining constraints (don't forget the constraints $x_1 \geq 0$ and $x_2 \geq 0$!). The feasible set is the dark area in Figure 2.2 given by the intersection of the respective halfspaces H^- of the constraints.

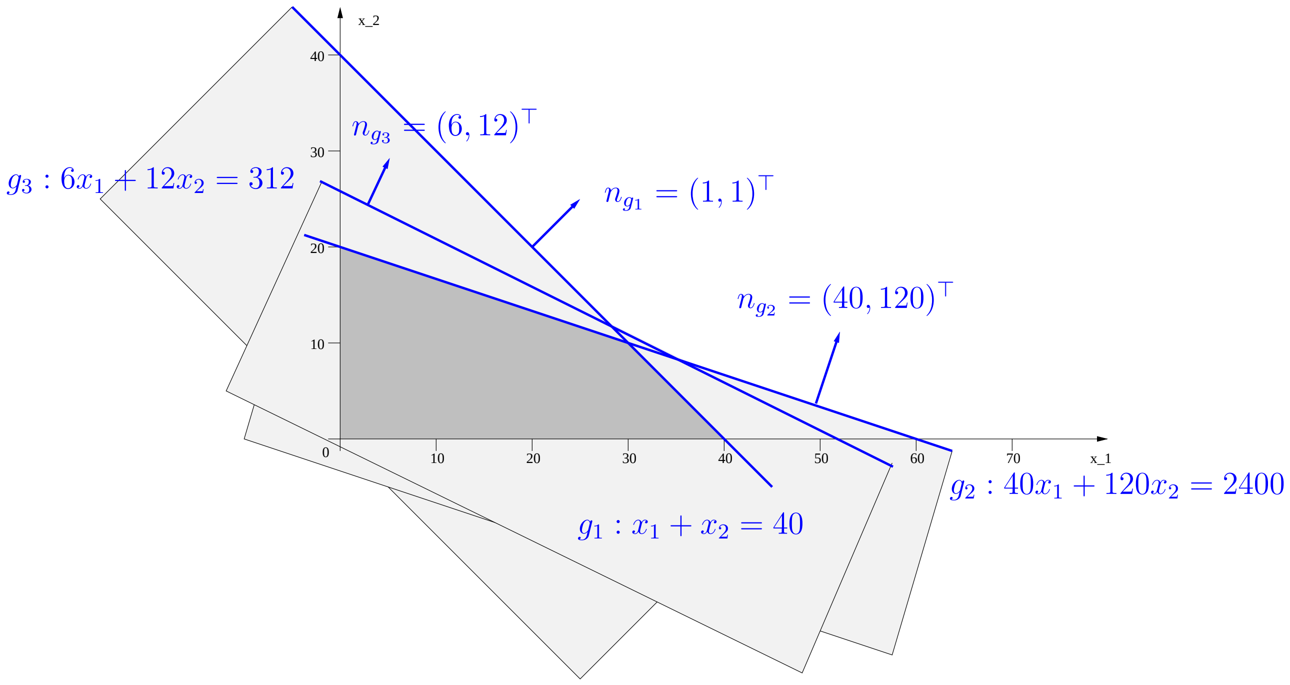


Figure 2.2: Feasible set of the linear program: Intersection of halfspaces for the constraints g_1, g_2, g_3 and $x_1 \geq 0$ and $x_2 \geq 0$ (the length of the normal vectors has been scaled).

We haven't considered the objective function so far. The red line in the Figure 2.3 corresponds to the line

$$f(x_1, x_2) = -100x_1 - 250x_2 = -3500,$$

i.e. for all points on this line the objective function assumes the value -3500. Herein, the value -3500 is just an arbitrary guess of the optimal value. The green line corresponds to the line

$$f(x_1, x_2) = -100x_1 - 250x_2 = -5500,$$

i.e. for all points on this line the objective function assumes the value -5500.
Obviously, the function values of f increase in the direction of the normal vector $n_f(x_1, x_2) = (-100, -250)^\top$, which is nothing else but the gradient of f. As we intend to minimise f, we have to 'move' the 'objective function line' in the opposite direction $-n_f$ (direction of descent!) as long as it does not completely leave the feasible set (the intersection of this line and the feasible set has to be nonempty). The feasible points on this extremal 'objective function line' are then optimal. The green line is the optimal line as there would be no feasible points on it if we moved it any further in the direction $-n_f$.
According to the figure we graphically determine the optimal solution to be $x_1 = 30$, $x_2 = 10$ with objective function $f(x_1, x_2) = -5500$. The solution is attained in a vertex of the feasible set.
In this example, the feasible vertices are given by the points $(0,0)$, $(0,20)$, $(30,10)$, and

$(40, 0)$.

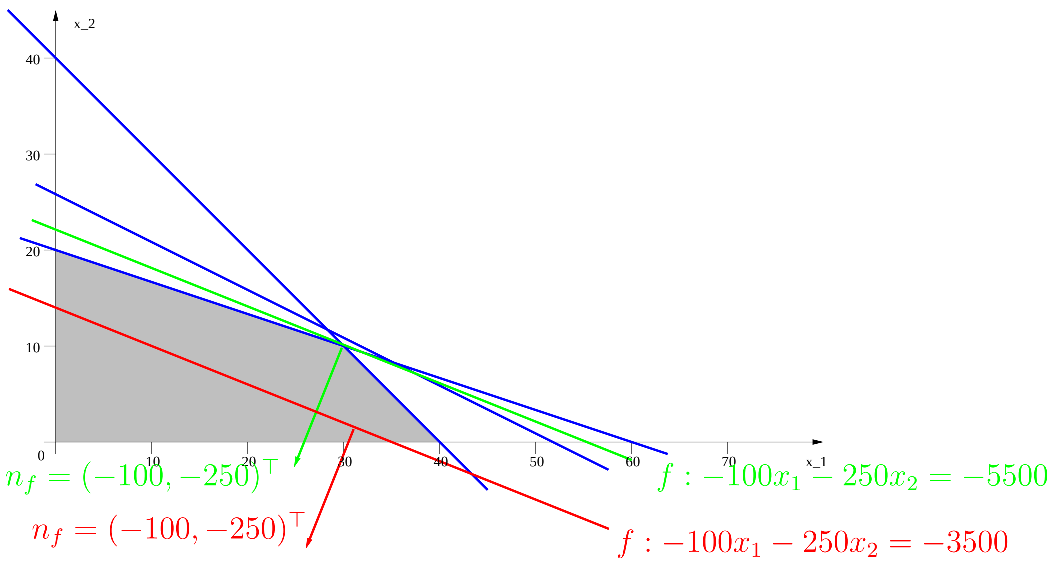


Figure 2.3: Solving the LP graphically: Move the objective function line in the opposite direction of n_f as long as the objective function line intersects with the feasible set (dark area). The green line is optimal. The optimal solution is $(x_1, x_2)^\top = (30, 10)^\top$.

In general, each row

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \le b_i, \qquad i \in \{1, \ldots, m\},$$

of the constraint $Ax \le b$ of the canonical linear program defines a hyperplane

$$H = \{x \in \mathbb{R}^n \mid a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = b_i\},$$

with normal vector $(a_{i1}, \ldots, a_{in})^\top$, a positive halfspace

$$H^+ = \{x \in \mathbb{R}^n \mid a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \ge b_i\},$$

and a negative halfspace

$$H^- = \{x \in \mathbb{R}^n \mid a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \le b_i\}.$$

Recall that the normal vector n is perpendicular to H and points into the positive halfspace H^+, cf. Figure 2.4.

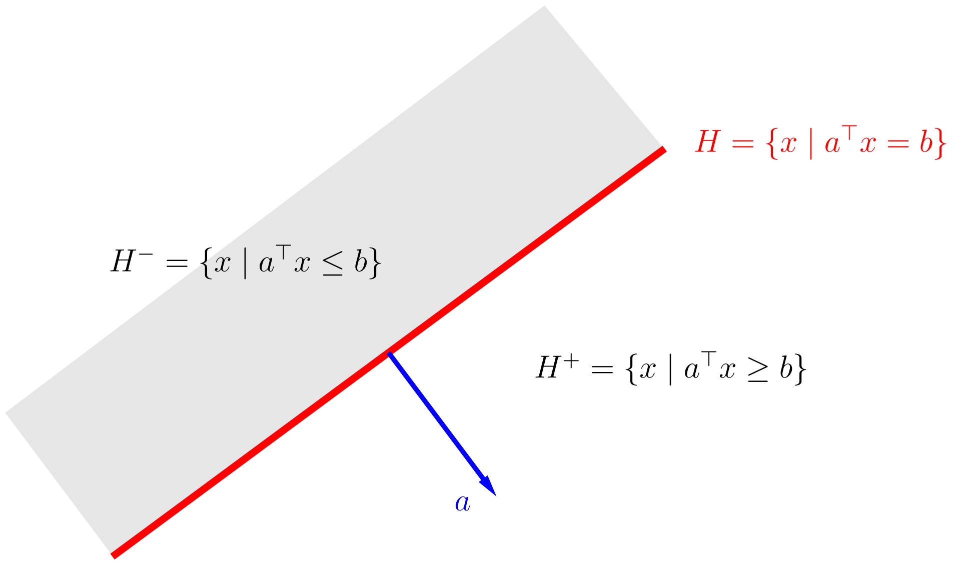

$H = \{x \mid a^\top x = b\}$
$H^- = \{x \mid a^\top x \leq b\}$
$H^+ = \{x \mid a^\top x \geq b\}$
a

Figure 2.4: Geometry of the inequality $a^\top x \leq b$: Normal vector a, hyperplane H, and halfspaces H^+ and H^-.

The feasible set M_C of the canonical linear program is given by the intersection of the respective negative halfspaces for the constraints $Ax \leq b$ and $x \geq 0$. The intersection of a finite number of halfspaces is called polyedric set or polyeder. A bounded polyeder is called polytope. Figure 2.5 depicts the different constellations that may occur in linear programming.

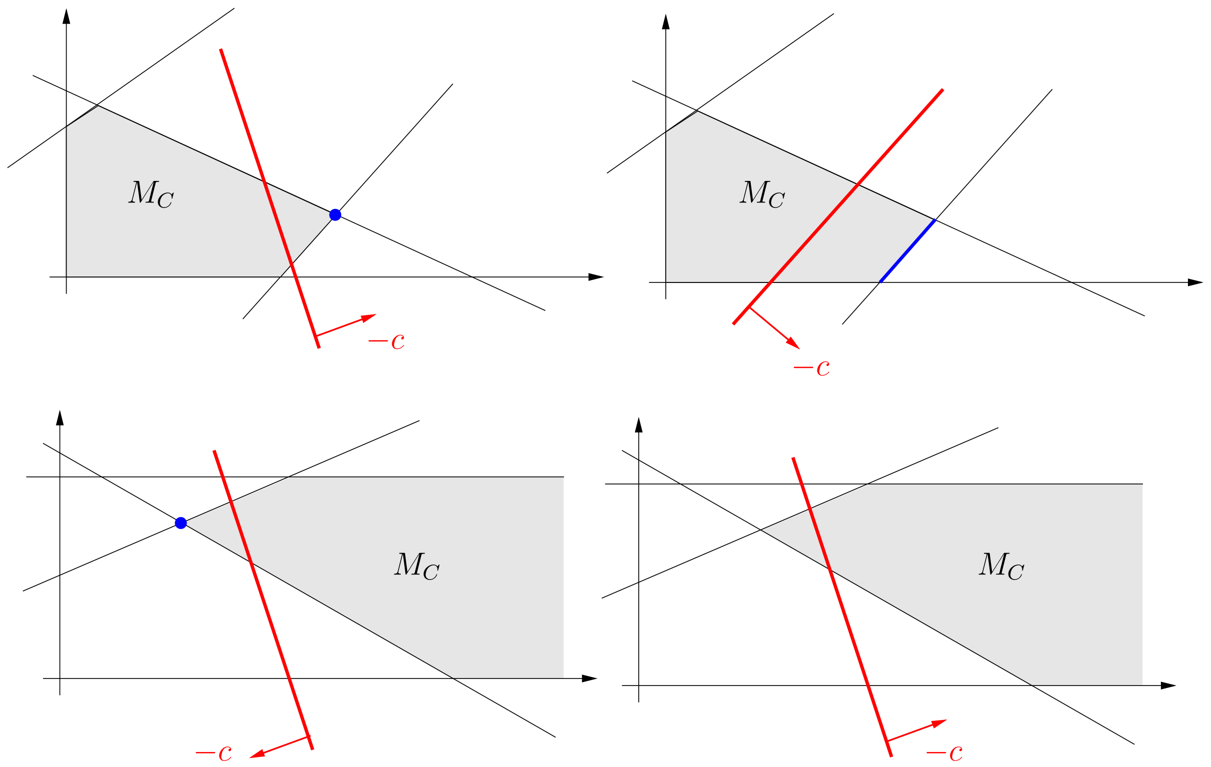


Figure 2.5: Different geometries of the feasible set (grey), objective function line (red), optimal solution (blue).

Summarising, the graphical solution of an LP works as follows:

(1) Sketch the feasible set.

(2) Make a guess $w \in \mathbb{R}$ of the optimal objective function value or alternatively take an arbitrary (feasible) point $\hat{x}$, introduce this point $\hat{x}$ into the objective function and compute $w = c^\top \hat{x}$. Plot the straight line given by $c^\top x = w$.

(3) Move the straight line in (2) in the direction $-c$ as long as the intersection of the line with the feasible set is non-empty.

(4) The most extreme line in (3) is optimal. All feasible points on this line are optimal. Their values can be found in the picture.

Observations:

- The feasible set M_C can be bounded (pictures on top) or unbounded (pictures at bottom).

- If the feasible set is bounded, there always exists an optimal solution by the Theorem of Weierstrass (M_C is compact and the objective function is continuous and thus assumes its minimum and maximum value on M_C).

- The optimal solution can be unique (pictures on left) or non-unique (top-right picture).
- The bottom-left figure shows that an optimal solution may exist even if the feasible set is unbounded.
- The bottom-right figure shows that the LP might not have an optimal solution.

The following observation is the main motivation for the simplex method. We will see that this observation is always true!

Main observation:

If an optimal solution exists, there is always a 'vertex' of the feasible set among the optimal solutions.

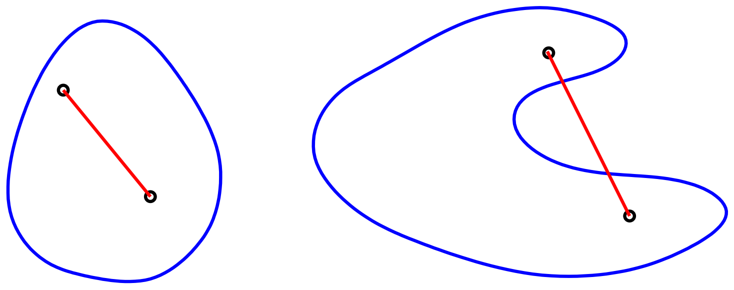
2.3 The Fundamental Theorem of Linear Programming

The main observation suggests that vertices play an outstanding role in linear programming. While it is easy to identify vertices in a picture, it is more difficult to characterise a vertex mathematically. We will define a vertex for convex sets.
We note

Theorem 2.3.1
The feasible set of a linear program is convex.

Herein, a set $M \in \mathbb{R}^n$ is called convex, if the entire line connecting two points in M belongs to M, i.e.

$$x, y \in M \quad \Rightarrow \quad \lambda x + (1-\lambda)y \in M \text{ for all } 0 \leq \lambda \leq 1.$$



convex and non-convex set

Proof: Without loss of generality the discussion is restricted to the standard LP. Let $x, y \in M_S$, $0 \leq \lambda \leq 1$, and $z = \lambda x + (1-\lambda)y$. We have to show that $z \in M_S$ holds. Since

$\lambda \in [0,1]$ and $x, y \geq 0$ it holds $z \geq 0$. Moreover, it holds

$$Az = \lambda Ax + (1-\lambda)Ay = \lambda b + (1-\lambda)b = b.$$

Hence, $z \in M_S$. □

Now we define a vertex of a convex set.

Definition 2.3.2 **(Vertex)**

Let M be a convex set. $x \in M$ is called vertex of M, if the representation $x = \lambda x_1 + (1-\lambda)x_2$ with $0 < \lambda < 1$ and $x_1, x_2 \in M$ implies $x_1 = x_2 = x$.

Remark 2.3.3

The set $\{\lambda x_1 + (1-\lambda)x_2 \mid 0 < \lambda < 1\}$ is the straight line interconnecting the points x_1 and x_2 (x_1 and x_2 are not included). Hence, if x is a vertex it cannot be on this line unless $x = x_1 = x_2$.

According to the following theorem it suffices to visit only the vertices of the feasible set in order to find one (not every!) optimal solution. As any LP can be transformed into standard form, it suffices to restrict the discussion to standard LP's. Of course, the following result holds for any LP.

Theorem 2.3.4 **(Fundamental Theorem of Linear Programming)**

Let a standard LP 2.1.5 be given with $M_S \neq \emptyset$. Then:

(a) Either the objective function is unbounded from below on M_S or Problem 2.1.5 has an optimal solution and at least one vertex of M_S is among the optimal solutions.

(b) If M_S is bounded, then an optimal solution exists and $x \in M_S$ is optimal, if and only if x is a convex combination of optimal vertices.

Proof: The proof exploits the following facts about polyeders, which we don't proof here:

(i) If $M_S \neq \emptyset$, then at least one vertex exists.

(ii) The number of vertices of M_S is finite.

(iii) Let $x^1, \ldots, x^N$ with $N \in \mathbb{N}$ denote the vertices of M_S. For every $x \in M_S$ there exists a vector $d \in \mathbb{R}^n$ and scalars λ_i, $i = 1, \ldots, N$, such that

$$x = \sum_{i=1}^{N} \lambda_i x^i + d$$

and

$$\lambda_i \geq 0, \quad i = 1, \ldots, N, \quad \sum_{i=1}^{N} \lambda_i = 1, \quad d \geq 0, \quad Ad = 0.$$

(iv) If M_S is bounded, then every $x \in M_S$ can be expressed as

$$x = \sum_{i=1}^{N} \lambda_i x^i, \qquad \lambda_i \geq 0, \quad i = 1, \ldots, N, \quad \sum_{i=1}^{N} \lambda_i = 1.$$

With these facts the assertions can be proven as follows.

(a) Case 1: If in (iii) there exists a $d \in \mathbb{R}^n$ with $d \geq 0$, $Ad = 0$, and $c^\top d < 0$, then the objective function is unbounded from below as for an $x \in M_S$ the points $x + td$ with $t \geq 0$ are feasible as well because $A(x+td) = Ax + tAd = Ax = b$ and $x + td \geq x \geq 0$. But $c^\top(x + td) = c^\top x + tc^\top d \to -\infty$ for $t \to \infty$ and thus the objective function is unbounded from below on M_S.

Case 2: Let $c^\top d \geq 0$ hold for every $d \in \mathbb{R}^n$ with $d \geq 0$ and $Ad = 0$. According to (i), (ii), and (iii) every point $x \in M_S$ can be expressed as

$$x = \sum_{i=1}^{N} \lambda_i x^i + d$$

with suitable

$$\lambda_i \geq 0, \quad i = 1, \ldots, N, \quad \sum_{i=1}^{N} \lambda_i = 1, \quad d \geq 0, \quad Ad = 0.$$

Let x^0 be an optimal point. Then, x^0 can be expressed as

$$x^0 = \sum_{i=1}^{N} \lambda_i x^i + d, \qquad \lambda_i \geq 0, \quad \sum_{i=1}^{N} \lambda_i = 1, \quad Ad = 0, \quad d \geq 0.$$

Let x^j, $j \in \{1, \ldots, N\}$, denote the vertex with the minimal objective function value, i.e. it holds $c^\top x^j \leq c^\top x^i$ for all $i = 1, \ldots, N$. Then

$$c^\top x^0 = \sum_{i=1}^{N} \lambda_i c^\top x^i + c^\top d \geq \sum_{i=1}^{N} \lambda_i c^\top x^i \geq c^\top x^j \sum_{i=1}^{N} \lambda_i = c^\top x^j.$$

As x^0 was supposed to be optimal, we found that x^j is optimal as well which shows the assertion.

(b) Let M_S be bounded. Then the objective function is bounded from below on M_S according to the famous Theorem of Weierstrass. Hence, the LP has at least one solution according to (a). Let x^0 be an arbitrary optimal point. According to (iv) x^0 can be expressed as a convex combination of the vertices x^i, $i = 1, \ldots, N$, i.e.

$$x^0 = \sum_{i=1}^{N} \lambda_i x^i, \qquad \lambda_i \geq 0, \quad \sum_{i=1}^{N} \lambda_i = 1.$$

Let x^j, $j \in \{1, \ldots, N\}$, be an optimal vertex (which exists according to (a)), i.e.

$$c^\top x^0 = c^\top x^j = \min_{i=1,\ldots,N} c^\top x^i.$$

Then, using the expression for x^0 it follows

$$c^\top x^0 = \sum_{i=1}^{N} \lambda_i c^\top x^i = c^\top x^j = \min_{i=1,\ldots,N} c^\top x^i.$$

Now let x^k, $k \in \{1, \ldots, N\}$, be a non-optimal vertex with $c^\top x^k = c^\top x^0 + \varepsilon$, $\varepsilon > 0$. Then,

$$c^\top x^0 = \sum_{i=1}^{N} \lambda_i c^\top x^i = \sum_{i=1, i\neq k}^{N} \lambda_i c^\top x^i + \lambda_k c^\top x^k \geq c^\top x^0 \sum_{i=1, i\neq k}^{N} \lambda_i + \lambda_k c^\top x^0 + \lambda_k \varepsilon = c^\top x^0 + \lambda_k \varepsilon.$$

As $\varepsilon > 0$ this implies $\lambda_k = 0$ which shows the first part of the assertion.

If x^0 is a convex combination of optimal vertices, i.e.

$$x^0 = \sum_{i=1}^{N} \lambda_i x^i, \qquad \lambda_i \geq 0, \quad \sum_{i=1}^{N} \lambda_i = 1,$$

then due to the linearity of the objective function and the convexity of the feasible set it is easy to show that x^0 is optimal. This completes the proof.

□

Remark 2.3.5 **(Software)**

- *Matlab (www.mathworks.com) provides the command* `linprog` *for the solution of linear programs.*
- *Scilab (www.scilab.org) is a powerful platform for scientific computing. It provides the command* `linpro` *for the solution of linear programs.*
- *There are many commercial and non-commercial implementations of algorithms for linear programming on the web. For instance, the company* **Lindo Systems Inc.**

develops software for the solution of linear, integer, nonlinear and quadratic programs. On their webpage

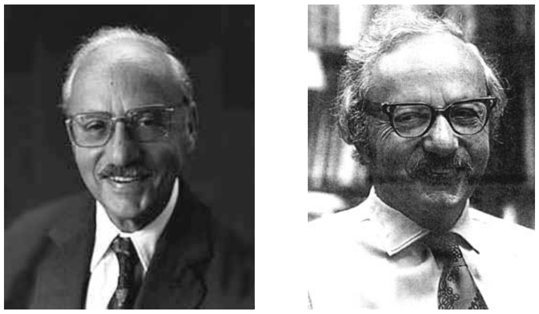
`http://www.lindo.com`

it is possible to download a free trial version of the software `LINDO`*.*

- *The program GeoGebra (www.geogebra.org) is a nice geometry program that can be used to visualise feasible sets.*

2.4 The Simplex Method

A real breakthrough in linear programming was the invention of the simplex method by G. B. Dantzig in 1947.



George Bernard Dantzig
Born: 8.11.1914 in Portland (Oregon)
Died: 13.5.2005 in Palo Alto (California)

The simplex method is one of the most important and popular algorithms for solving linear programs on a computer. The very basic idea of the simplex method is to move from a feasible vertex to a neighbouring feasible vertex and repeat this procedure until an optimal vertex is reached. According to Theorem 2.3.4 it is sufficient to consider only the vertices of the feasible set.
In order to construct the simplex algorithm, we need to answer the following questions:

- How can (feasible) vertices be computed?
- Given a feasible vertex, how can we compute a neighbouring feasible vertex?
- How to check for optimality?

In this chapter we will restrict the discussion to LP's in standard form 2.1.5, i.e.

$$\min \quad c^\top x \quad \text{s.t.} \quad Ax = b,\ x \geq 0.$$

Throughout the rest of the chapter we assume $\text{rank}(A) = m$. This assumption excludes linearly dependent constraints from the problem formulation.

Notation:

- The columns of A are denoted by

$$a^j := (a_{1j}, \dots, a_{mj})^\top \in \mathbb{R}^m, \qquad j = 1, \dots, n,$$

  i.e.

$$A = \left(\begin{array}{|c|c|c|c|} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{array}\right) = \left(\begin{array}{cccc} a^1 & a^2 & \cdots & a^n \end{array} \right).$$

2.4.1 Vertices and Basic Solutions

The geometric definition of a vertex in Definition 2.3.2 is not useful if a vertex has to be computed explicitely within an algorithm. Hence, we are looking for an alternative characterisation of a vertex which allows us to actually compute a vertex. For this purpose the standard LP is particularly well suited.
Consider the feasible set of the standard LP 2.1.5:

$$M_S = \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}.$$

Let $x = (x_1, \dots, x_n)^\top$ be an arbitrary point in M_S. Let

$$B := \{i \in \{1, \dots, n\} \mid x_i > 0\}$$

be the index set of positive components of x, and

$$N := \{1, \dots, n\} \setminus B = \{i \in \{1, \dots, n\} \mid x_i = 0\}$$

the index set of vanishing components of x. Because $x \in M_S$ it holds

$$b = Ax = \sum_{j=1}^{n} a^j x_j = \sum_{j \in B} a^j x_j.$$

This is a linear equation for the components x_j, $j \in B$. This linear equation has a unique solution, if the column vectors a^j, $j \in B$, are linearly independent. In this case, we call x a feasible basic solution.

Definition 2.4.1 (feasible basic solution)
Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $M_S = \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$.

$x \in M_S$ is called feasible basic solution (for the standard LP), if the column vectors $\{a^j \mid j \in B\}$ with $B = \{i \in \{1,\ldots,n\} \mid x_i > 0\}$ are linearly independent.

The following theorem states that a feasible basic solution is a vertex of the feasible set and vice versa.

Theorem 2.4.2

$x \in M_S$ is a vertex of M_S if and only if x is a feasible basic solution.

Proof: Recall that the feasible set M_S is convex, see Theorem 2.3.1.

'$\Rightarrow$' Let x be a vertex. Without loss of generality let $x = (x_1,\ldots,x_r,0,\ldots,0)^\top \in \mathbb{R}^n$ with $x_i > 0$ for $i = 1,\ldots,r$.

We assume that a^j, $j = 1,\ldots,r$, are linearly dependent. Then there exist α_j, $j = 1,\ldots,r$, not all of them are zero with

$$\sum_{j=1}^{r} \alpha_j a^j = 0.$$

Define

$$\begin{aligned} y_+ &= (x_1 + \varepsilon\alpha_1,\ldots,x_r + \varepsilon\alpha_r,0,\ldots,0)^\top \\ y_- &= (x_1 - \varepsilon\alpha_1,\ldots,x_r - \varepsilon\alpha_r,0,\ldots,0)^\top \end{aligned}$$

with

$$0 < \varepsilon < \min\left\{ \frac{|x_j|}{|\alpha_j|} \;\middle|\; \alpha_j \neq 0,\ 1 \leq j \leq r \right\}.$$

According to this choice of ε it follows $y_+, y_- \geq 0$. Moreover, y_+ and y_- are feasible because

$$Ay_\pm = \sum_{j=1}^{r}(x_j \pm \varepsilon\alpha_j)a^j = \sum_{j=1}^{r} x_j a^j \pm \varepsilon \underbrace{\sum_{j=1}^{r}\alpha_j a^j}_{=0,\text{ lin. dep.}} = b.$$

Hence, $y_+, y_- \in M_S$, but $(y_+ + y_-)/2 = x$. This contradicts the assumption that x is a vertex.

'$\Leftarrow$' Without loss of generality let $x = (x_1,\ldots,x_r,0,\ldots,0)^\top \in M_S$ with $x_j > 0$ and a^j linearly independent for $j = 1,\ldots,r$. Let $y, z \in M_S$ and $x = \lambda y + (1-\lambda)z$, $0 < \lambda < 1$. Then,

(i)

$$x_j > 0 \quad (1 \leq j \leq r) \quad \Rightarrow \quad y_j > 0 \text{ or } z_j > 0 \text{ for all } 1 \leq j \leq r.$$

(ii)

$$x_j = 0 \quad (j = r+1,\ldots,n) \quad \Rightarrow \quad y_j = z_j = 0 \text{ for } r+1 \leq j \leq n.$$

$b = Ay = Az$ implies $0 = A(y-z) = \sum_{j=1}^{r}(y_j - z_j)a^j$. Due to the linear independence of a^j it follows $y_j - z_j = 0$ for $j = 1, \ldots, r$. Since $y_j = z_j = 0$ for $j = r+1, \ldots, n$, it holds $y = z$. Hence, x is a vertex.

□

2.4.2 The (primal) Simplex Method

Theorem 2.4.2 states that a vertex can be characterised by linearly independent columns of A. According to Theorem 2.3.4 it is sufficient to compute only the vertices (feasible basic solutions) of the feasible set in order to obtain at least one optimal solution – provided such a solution exists at all. This is the basic idea of the simplex method.

Notation:

Let $B \subseteq \{1, \ldots, n\}$ be an index set.

- Let $x = (x_1, \ldots, x_n)^\top$ be a vector. Then, x_B is defined to be the vector with components x_i, $i \in B$.
- Let A be a $m \times n$-matrix with columns a^j, $j = 1, \ldots, n$. Then, A_B is defined to be the matrix with columns a^j, $j \in B$.

Example:

$$x = \begin{pmatrix} -1 \\ 10 \\ 3 \\ -4 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}, \quad B = \{2, 4\} \quad \Rightarrow \quad x_B = \begin{pmatrix} 10 \\ -4 \end{pmatrix}, \quad A_B = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}.$$

The equivalence theorem 2.4.2 is of fundamental importance for the simplex method, because we can try to find vertices by solving linear equations as follows:

Algorithm 2.4.3 (Computation of a vertex)

(1) Choose m linearly independent columns a^j, $j \in B$, $B \subseteq \{1, \ldots, n\}$, of A and set $N = \{1, \ldots, n\} \setminus B$.

(2) Set $x_N = 0$ and solve the linear equation

$$A_B x_B = b.$$

(3) If $x_B \geq 0$, x is a feasible basic solution and thus a vertex. STOP. If there exists an index $i \in B$ with $x_i < 0$, then x is infeasible. Go to (1) and repeat the procedure with a different choice of linearly independent columns.

Recall $\operatorname{rank}(A) = m$, which guarantees the existence of m linearly independent columns of A in step (1) of the algorithm.
A naive approach to solve a linear program would be to compute all vertices of the feasible set by the above algorithm. As there are at most $\binom{n}{m}$ ways to choose m linearly independent columns from a total of n columns there exist at most

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

vertices (resp. feasible basic solutions). Unfortunately, this is a potentially large number, especially if m and n are large (1 million, say). Hence, it is not efficient to compute every vertex.

Caution: Computing all vertices and choosing that one with the smallest objective function value may not be sufficient to solve the problem, because it may happen that the feasible set and the objective function is unbounded! In this case, no solution exists.

Example 2.4.4

Consider the constraints $Ax = b$, $x \geq 0$, with

$$A = \begin{pmatrix} 2 & 3 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 6 \\ 2 \end{pmatrix}.$$

There are at most $\binom{4}{2} = 6$ possible combinations of two columns of A:

$$\begin{aligned}
B_1 = \{1,2\} &\Rightarrow A_B^{-1}b = \begin{pmatrix} 2 \\ \frac{2}{3} \end{pmatrix}, \quad x^1 = \left(2, \frac{2}{3}, 0, 0\right)^\top, \\
B_2 = \{1,3\} &\Rightarrow A_B^{-1}b = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \quad x^2 = (2,0,2,0)^\top, \\
B_3 = \{1,4\} &\Rightarrow A_B^{-1}b = \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \quad x^3 = (3,0,0,-1)^\top, \\
B_4 = \{2,3\} &\Rightarrow A_B \text{ is singular}, \\
B_5 = \{2,4\} &\Rightarrow A_B^{-1}b = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \quad x^5 = (0,2,0,2)^\top, \\
B_6 = \{3,4\} &\Rightarrow A_B^{-1}b = \begin{pmatrix} 6 \\ 2 \end{pmatrix}, \quad x^6 = (0,0,6,2)^\top.
\end{aligned}$$

x^3 is not feasible. Hence, the vertices are given by x^1, x^2, x^5, x^6.

Remark 2.4.5

It is possible to define basic solutions which are not feasible. Let $B \subseteq \{1, \ldots, n\}$ be an *index set with $|B| = m$, $N = \{1, \dots, n\} \setminus B$, and let the columns a^i, $i \in B$, be linearly independent. Then, x is called a basic solution if*

$$A_B x_B = b, \qquad x_N = 0.$$

Notice that such a $x_B = A_B^{-1} b$ does not necessarily satisfy the sign condition $x_B \geq 0$, i.e. the above x may be infeasible. Those infeasible basic solutions are not of interest for the linear program.

We need some more definitions.

Definition 2.4.6

- *(Basis)*
  Let $rank(A) = m$ and let x be a feasible basic solution of the standard LP. Every system $\{a^j \mid j \in B\}$ of m linearly independent columns of A, which includes those columns a^j with $x_j > 0$, is called basis of x.

- *((Non-)basis index set, (non-)basis matrix, (non-)basic variable)*
  Let $\{a^j \mid j \in B\}$ be a basis of x. The index set B is called basis index set, the index set $N := \{1, \dots, n\} \setminus B$ is called non-basis index set, the matrix $A_B := (a^j)_{j \in B}$ is called basis matrix, the matrix $A_N := (a^j)_{j \in N}$ is called non-basis matrix, the vector $x_B := (x_j)_{j \in B}$ is called basic variable and the vector $x_N := (x_j)_{j \in N}$ is called non-basic variable.

Let's consider an example.

Example 2.4.7

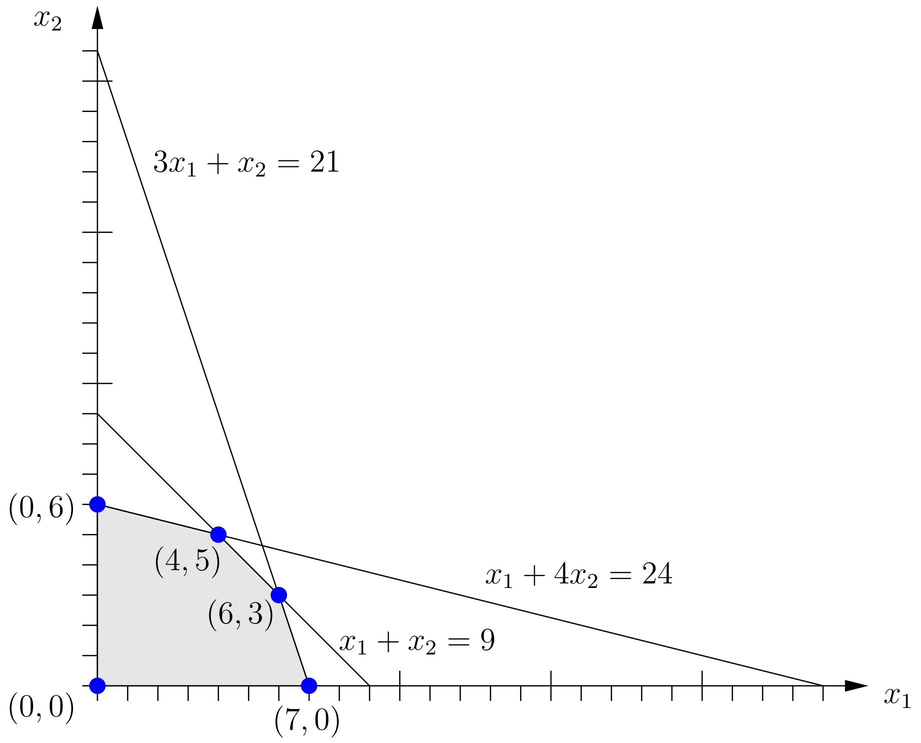
Consider the inequality constraints

$$\begin{aligned} x_1 + 4x_2 &\leq 24, \\ 3x_1 + x_2 &\leq 21, \\ x_1 + x_2 &\leq 9, \qquad x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$

Introduction of slack variables $x_3 \geq 0$, $x_4 \geq 0$, $x_5 \geq 0$ leads to standard constraints $Ax = b$, $x \geq 0$, with

$$A = \begin{pmatrix} 1 & 4 & 1 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 24 \\ 21 \\ 9 \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}.$$

The projection of the feasible set $M_S = \{x \in \mathbb{R}^5 \mid Ax = b,\ x \geq 0\}$ into the (x_1, x_2)-plane looks as follows:



(i) *Consider $x = (6,3,6,0,0)^\top \in M_S$. The first three components are positive and the corresponding columns of A, i.e. $\begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$, $\begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}$ and $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, are linearly independent. x is actually a feasible basic solution and according to Theorem 2.4.2 a vertex. We obtain the basis*

$$\left\{ \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right\}$$

with basis index set $B = \{1,2,3\}$, non-basis index set $N = \{4,5\}$, basic variable $x_B = (6,3,6)^\top$, non-basic variable $x_N = (0,0)^\top$ and basis matrix resp. non-basis matrix

$$A_B := \begin{pmatrix} 1 & 4 & 1 \\ 3 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \qquad A_N := \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

(ii) *Exercise: Consider as in (i) the remaining vertices which are given by $(x_1, x_2) = (0,0)$, $(x_1, x_2) = (0,6)$, $(x_1, x_2) = (7,0)$, and $(x_1, x_2) = (4,5)$.*

With these definition, we are ready to formulate a draft version of the simplex method :

(0) Phase 0:
Transform the linear program into standard form 2.1.5, if necessary at all.

(1) Phase 1:
Determine a feasible basic solution (feasible vertex) x with basis index set B, non-basis index set N, basic variable $x_B \geq 0$ and non-basic variable $x_N = 0$.

(2) Phase 2:
Compute a neighbouring feasible basic solution x^+ with basis index set B^+ and non-basis index set N^+ until either an optimum is computed or it can be decided that no such solution exists.

2.4.3 Computing a neighbouring feasible basic solution

In this section we will discuss how a neighbouring feasible basic solution x^+ can be computed in phase two of the simplex algorithm.
In the sequel, we assume that a feasible basic solution x with basis index set B and non-basis index set N is given. We will construct a new feasible basic solution x^+ with index sets

$$\begin{aligned} B^+ &= (B \setminus \{p\}) \cup \{q\}, \\ N^+ &= (N \setminus \{q\}) \cup \{p\} \end{aligned}$$

by interchanging suitably chosen indices $p \in B$ and $q \in N$. This procedure is called basis change.
The indices p and q are not chosen arbitrarily but in such a way that the following requirements are satisfied:

(i) Feasibility:
x^+ has to remain feasible, i.e.

$$Ax = b, \; x \geq 0 \qquad \Rightarrow \qquad Ax^+ = b, \; x^+ \geq 0.$$

(ii) Descent property:
The objective function value decreases monotonically, i.e.

$$c^\top x^+ \leq c^\top x.$$

Let a basis with basis index set B be given and let x be a feasible basic solution. Then,

- $A_B = (a^i)_{i \in B}$ is non-singular.
- $x_N = 0$.

Hence, the constraint $Ax = b$ can be solved w.r.t. the basic variable x_B according to

$$Ax = A_B x_B + A_N x_N = b \qquad \Rightarrow \qquad x_B = \underbrace{A_B^{-1} b}_{=\beta} - \underbrace{A_B^{-1} A_N}_{=\Gamma} x_N =: \beta - \Gamma x_N. \tag{2.9}$$

Introducing this expression into the objective function yields the expression

$$c^\top x = c_B^\top x_B + c_N^\top x_N = \underbrace{c_B^\top \beta}_{=d} - \underbrace{(c_B^\top \Gamma - c_N^\top)}_{=\zeta^\top} x_N =: d - \zeta^\top x_N. \tag{2.10}$$

A feasible basic solution x satisfies $x_N = 0$ in (2.9) and (2.10) and thus,

$$x_B = \beta \geq 0, \quad x_N = 0, \quad c^\top x = d. \tag{2.11}$$

Herein, we used the notation: $\beta = (\beta_i)_{i \in B}$, $\zeta = (\zeta_j)_{j \in N}$, $\Gamma = (\gamma_{ij})_{i \in B, j \in N}$.
Now we intend to change the basis in order to get to a neighbouring basis. Therefore, we consider the ray

$$z(t) = \begin{pmatrix} z_1(t) \\ \vdots \\ z_n(t) \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + t \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} = x + ts, \qquad t \geq 0, \tag{2.12}$$

emanating from the current basic solution x in direction s with step length $t \geq 0$, cf. Figure 2.6.

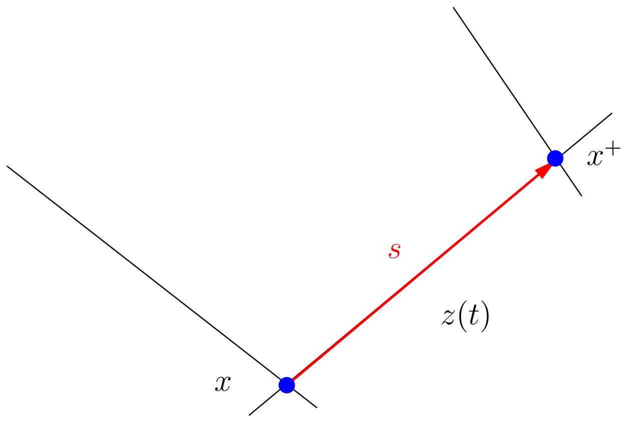


Figure 2.6: Idea of the basis change in the simplex method. Find a search direction s such that the objective function decreases monotonically along this direction.

We choose the search direction s in such a way that only the non-basic variable x_q with suitable index $q \in N$ will be changed while the remaining components $x_j = 0$, $j \in N$, $j \neq q$, remain unchanged. For a suitable index $q \in N$ we define

$$s_q := 1, \qquad s_j := 0 \text{ for } j \in N, \ j \neq q, \tag{2.13}$$

and thus,

$$z_q(t) = t \geq 0, \qquad z_j(t) = 0 \text{ for } j \in N,\ j \neq q. \tag{2.14}$$

Of course, the point $z(t)$ ought to be feasible, i.e. as in (2.9) the following condition has to hold:

$$b = Az(t) = \underbrace{Ax}_{=b} + tAs \quad \Rightarrow \quad 0 = As = A_B s_B + A_N s_N.$$

Solving this equation leads to

$$s_B = -A_B^{-1} A_N s_N = -\Gamma s_N. \tag{2.15}$$

Thus, the search direction s is completely defined by (2.13) and (2.15). Along the ray $z(t)$ the objective function value computes to

$$\begin{aligned} c^\top z(t) &\overset{(2.12)}{=} c^\top x + tc^\top s \\ &\overset{(2.11)}{=} d + tc_B^\top s_B + tc_N^\top s_N \\ &\overset{(2.15)}{=} d - t\left(c_B^\top \Gamma - c_N^\top\right) s_N \\ &= d - t\zeta^\top s_N \\ &\overset{(2.13)}{=} d - t\zeta_q. \end{aligned} \tag{2.16}$$

This representation immediately reveals that the objective function value decreases along s for $t \geq 0$, if $\zeta_q > 0$ holds.

If, on the other hand, $\zeta_j \leq 0$ holds for all $j \in N$, then a descent along s in the objective function is impossible. It remains to be investigated whether there are other points, which are not on the ray, but possibly lead to a smaller objective function value. This is not the case as for an arbitrary $\hat{x} \in M_S$ it holds $\hat{x} \geq 0$ and $A_B\hat{x}_B + A_N\hat{x}_N = b$ resp. $\hat{x}_B = \beta - \Gamma\hat{x}_N$. With $\zeta_j \leq 0$ and $\hat{x}_N \geq 0$ it follows

$$c^\top \hat{x} = c_B^\top \hat{x}_B + c_N^\top \hat{x}_N = c_B^\top \beta - (c_B^\top \Gamma - c_N^\top)\hat{x}_N = d - \zeta^\top \hat{x}_N \geq d = c^\top x.$$

Hence, if $\zeta_j \leq 0$ holds for all $j \in N$, then the current basic solution x is optimal! We summarise:

Choice of the pivot column q:
In order to fulfil the descent property in (ii), the index $q \in N$ has to be chosen such that $\zeta_q > 0$. If $\zeta_j \leq 0$ for all $j \in N$, then the current basic solution x is optimal.

Now we aim at satisfying the feasibility constraint in (i). The following has to hold:

$$z_B(t) = x_B + ts_B \overset{(2.15)}{=} \beta - t\Gamma s_N \geq 0,$$

respectively

$$\begin{aligned} z_i(t) &= \beta_i - t\sum_{j\in N}\gamma_{ij}s_j \\ &= \beta_i - t\gamma_{iq}\underbrace{s_q}_{=1} - t\sum_{j\in N, j\neq q}\gamma_{ij}\underbrace{s_j}_{=0} \\ &\overset{(2.13)}{=} \beta_i - \gamma_{iq}t \geq 0, \qquad i \in B. \end{aligned}$$

The conditions $\beta_i - \gamma_{iq}t \geq 0$, $i \in B$, restrict the step size $t \geq 0$. Two cases may occur:

(a) Case 1: It holds $\gamma_{iq} \leq 0$ for all $i \in B$.
Due to $\beta_i \geq 0$ it holds $z_i(t) = \beta_i - \gamma_{iq}t \geq 0$ for every $t \geq 0$ and every $i \in B$. Hence, $z(t)$ is feasible for every $t \geq 0$.

If in addition $\zeta_q > 0$ holds, then the objective function is not bounded from below for $t \to \infty$ according to (2.16). Thus, the linear program does not have a solution!

Unsolvability of LP
If for some $q \in N$ it holds $\zeta_q > 0$ and $\gamma_{iq} \leq 0$ for every $i \in B$, then the LP does not have a solution. The objective function is unbounded from below.

(b) Case 2: It holds $\gamma_{iq} > 0$ for at least one $i \in B$.
$\beta_i - \gamma_{iq}t \geq 0$ implies $t \leq \frac{\beta_i}{\gamma_{iq}}$. This postulation restricts the step length t.

Choice of pivot row p:
The feasibility in (i) will be satisfied by choosing an index $p \in B$ with

$$t_{min} := \frac{\beta_p}{\gamma_{pq}} := \min\left\{ \frac{\beta_i}{\gamma_{iq}} \;\middle|\; \gamma_{iq} > 0,\ i \in B \right\}.$$

It holds

$$\begin{aligned} z_p(t_{min}) &= \beta_p - \gamma_{pq}t_{min} = \beta_p - \gamma_{pq}\frac{\beta_p}{\gamma_{pq}} = 0, \\ z_i(t_{min}) &= \underbrace{\beta_i}_{\geq 0} - \underbrace{\gamma_{iq}}_{\leq 0}\underbrace{t_{min}}_{\geq 0} \geq 0, \qquad \text{for } i \text{ with } \gamma_{iq} \leq 0, \\ z_i(t_{min}) &= \beta_i - \gamma_{iq}\underbrace{t_{min}}_{\leq \frac{\beta_i}{\gamma_{iq}}} \geq \beta_i - \gamma_{iq}\frac{\beta_i}{\gamma_{iq}} = 0 \qquad \text{for } i \text{ with } \gamma_{iq} > 0. \end{aligned}$$

Hence, the point $x^+ := z(t_{min})$ is feasible and satisfies $x_p = 0$. Hence, x_p leaves the basic variables and enters the non-basic variables.

The following theorem states that x^+ is actually a feasible basic solution.

Theorem 2.4.8

Let x be a feasible basic solution with basis index set B. Let a pivot column $q \in N$ with $\zeta_q > 0$ and a pivot row $p \in B$ with $\gamma_{pq} > 0$ exist. Then, $x^+ = z(t_{min})$ is a feasible basic solution and $c^\top x^+ \leq c^\top x$. In particular, $\{a^j \mid j \in B^+\}$ with $B^+ = (B \setminus \{p\}) \cup \{q\}$ is a basis and A_{B^+} is non-singular.

Proof: By construction x^+ is feasible. It remains to show that $\{a^j \mid j \in B^+\}$ with $B^+ = (B \setminus \{p\}) \cup \{q\}$ is a basis. We note that $x_j = 0$ for all $j \notin B^+$.
The definition of $\Gamma = A_B^{-1} A_N$ implies $A_N = A_B \Gamma$. The column q of this matrix equation reads as

$$a^q = \sum_{i \in B} a^i \gamma_{iq} = a^p \gamma_{pq} + \sum_{i \in B, i \neq p} a^i \gamma_{iq}, \tag{2.17}$$

where γ_{iq} are the components of column q of Γ and $\gamma_{pq} \neq 0$ according to the assumption of this theorem.
We will now show that the vectors a^i, $i \in B$, $i \neq p$, and a^q are linearly independent. Therefore, we consider the equation

$$\alpha a^q + \sum_{i \in B, i \neq p} \alpha_i a^i = 0 \tag{2.18}$$

with coefficients $\alpha \in \mathbb{R}$ and $\alpha_i \in \mathbb{R}$, $i \in B$, $i \neq p$. We introduce a^q from (2.17) into (2.18) and obtain

$$\alpha \gamma_{pq} a^p + \sum_{i \in B, i \neq p} (\alpha_i + \alpha \gamma_{iq}) a^i = 0.$$

As $\{a^i \mid i \in B\}$ is a basis, we immediately obtain that all coefficients vanish, i.e.

$$\alpha \gamma_{pq} = 0, \qquad \alpha_i + \alpha \gamma_{iq} = 0, \qquad i \in B, i \neq p.$$

Since $\gamma_{pq} \neq 0$ it follows $\alpha = 0$ and this in turn implies $\alpha_i = 0$ for all $i \in B$, $i \neq p$. Hence, all coefficients vanish. This shows the linear independence of the set $\{a^i \mid i \in B^+\}$. □

2.4.4 The Algorithm

We summarise our findings in an algorithm:

Algorithm 2.4.9 ((Primal) Simplex Method)

(0) Phase 0:

Transform the linear program into standard form 2.1.5, if necessary at all.

(1) Phase 1:

Determine a feasible basic solution (feasible vertex) x for the standard LP 2.1.5 with *basis index set B, non-basis index set N, basis matrix A_B, non-basis matrix A_N, basic variable $x_B \geq 0$ and non-basic variable $x_N = 0$.*

If no feasible solution exists, STOP. The problem is infeasible.

(2) *Phase 2:*

(i) *Compute $\Gamma = (\gamma_{ij})_{i\in B, j\in N}$, $\beta = (\beta_i)_{i\in B}$, and $\zeta = (\zeta_j)_{j\in N}$ according to*

$$\Gamma = A_B^{-1}A_N, \qquad \beta = A_B^{-1}b, \qquad \zeta^\top = c_B^\top \Gamma - c_N^\top.$$

(ii) *Check for optimality:*
If $\zeta_j \leq 0$ for every $j \in N$, then STOP. The current feasible basic solution $x_B = \beta$, $x_N = 0$ is optimal. The objective function value is $d = c_B^\top \beta$.

(iii) *Check for unboundedness:*
If there exists an index q with $\zeta_q > 0$ and $\gamma_{iq} \leq 0$ for every $i \in B$, then the linear program does not have a solution and the objective function is unbounded from below. STOP.

(iv) *Determine pivot element:*
Choose an index q with $\zeta_q > 0$. q defines the pivot column. Choose an index p with

$$\frac{\beta_p}{\gamma_{pq}} = \min\left\{ \frac{\beta_i}{\gamma_{iq}} \;\middle|\; \gamma_{iq} > 0,\ i \in B \right\}.$$

p defines the pivot row.

(v) *Perform basis change:*
Set $B := (B \setminus \{p\}) \cup \{q\}$ and $N := (N \setminus \{q\}) \cup \{p\}$.

(vi) *Go to (i).*

Remark 2.4.10

It is very important to recognise, that the indexing of the elements of the matrix Γ and the vectors β and ζ in (i) depends on the current entries and the orders in the index sets B and N. Notice that B and N are altered in each step (v) of the algorithm. So, Γ, β, and ζ are altered as well in each iteration. For instance, if $B = \{2,4,5\}$ and $N = \{1,3\}$, the entries of the matrix Γ and the vectors β and ζ are indexed as follows:

$$\Gamma = \begin{pmatrix} \gamma_{21} & \gamma_{23} \\ \gamma_{41} & \gamma_{43} \\ \gamma_{51} & \gamma_{53} \end{pmatrix}, \qquad \beta = \begin{pmatrix} \beta_2 \\ \beta_4 \\ \beta_5 \end{pmatrix}, \qquad \zeta = \begin{pmatrix} \zeta_1 \\ \zeta_3 \end{pmatrix}.$$

More generally, if $B = \{i_1, \ldots, i_m\}$*,* $i_k \in \{1, \ldots, n\}$*, and* $N = \{j_1, \ldots, j_{n-m}\}$*,* $j_k \in \{1, \ldots, n\} \setminus B$*, then*

$$\Gamma = \begin{pmatrix} \gamma_{i_1 j_1} & \gamma_{i_1 j_2} & \cdots & \gamma_{i_1 j_{n-m}} \\ \gamma_{i_2 j_1} & \gamma_{i_2 j_2} & \cdots & \gamma_{i_2 j_{n-m}} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{i_m j_1} & \gamma_{i_m j_2} & \cdots & \gamma_{i_m j_{n-m}} \end{pmatrix}, \qquad \beta = \begin{pmatrix} \beta_{i_1} \\ \beta_{i_2} \\ \vdots \\ \beta_{i_m} \end{pmatrix}, \qquad \zeta = \begin{pmatrix} \zeta_{j_1} \\ \zeta_{j_2} \\ \vdots \\ \zeta_{j_{n-m}} \end{pmatrix}.$$

The simplex method often is given in a more compact notation – the simplex table. The relations

$$\begin{aligned} x_B &= \beta - \Gamma x_N, \\ c^\top x &= d - \zeta^\top x_N, \end{aligned}$$

compare (2.9) and (2.10), are combined in the following table:

	x_N	
x_B	$\Gamma = (\gamma_{ij}) := A_B^{-1} A_N$	$\beta := A_B^{-1} b$
	$\zeta^\top := c_B^\top A_B^{-1} A_N - c_N^\top$	$d := c_B^\top A_B^{-1} b$

As the non-basic variable is zero, the current value of the variable x can be immediately obtained from the table: $x_B = \beta$, $x_N = 0$. The corresponding objective function is $c^\top x = d$.

Example 2.4.11 (compare Example 2.4.7)

Consider the standard LP

$$\textit{Minimise} \quad c^\top x \quad \textit{subject to} \quad Ax = b,\ x \geq 0$$

with the data (x_3, x_4, x_5 are slack variables):

$$c = \begin{pmatrix} -2 \\ -5 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 4 & 1 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 24 \\ 21 \\ 9 \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}.$$

Phase 1:

A feasible basic solution is given by the basis index set $B = \{3,4,5\}$*. Then,* $N = \{1,2\}$*,*

$$A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; A_N = \begin{pmatrix} 1 & 4 \\ 3 & 1 \\ 1 & 1 \end{pmatrix}, \; c_B = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \; c_N = \begin{pmatrix} -2 \\ -5 \end{pmatrix}.$$

and

$$\begin{pmatrix} \gamma_{31} & \gamma_{32} \\ \gamma_{41} & \gamma_{42} \\ \gamma_{51} & \gamma_{52} \end{pmatrix} = \begin{pmatrix} 1 & 4 \\ 3 & 1 \\ 1 & 1 \end{pmatrix}, \; \begin{pmatrix} \beta_3 \\ \beta_4 \\ \beta_5 \end{pmatrix} = \begin{pmatrix} 24 \\ 21 \\ 9 \end{pmatrix}, \; \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \end{pmatrix}.$$

Obviously, columns 3,4,5 of A *are linearly independent and*

$$x_B = \begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 24 \\ 21 \\ 9 \end{pmatrix} = b = A_B^{-1} b > 0 \qquad \textit{and} \qquad x_N = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

is feasible, i.e. $x \geq 0$ *and* $Ax = b$*.*

The initial feasible basic solution is represented by the following initial table:

	x_1	x_2	
x_3	1	$\boxed{4}$	24
x_4	3	1	21
x_5	1	1	9
	2	5	0

We now enter Phase 2:
The checks for optimality and unboundedness fail. Hence, we choose the index $q = 2 \in N$ *with* $\zeta_q = 5 > 0$ *as the pivot column (we could choose* $q = 1 \in N$ *as well) and compute*

$$\frac{\beta_3}{\gamma_{32}} = \frac{24}{4} = 6, \quad \frac{\beta_4}{\gamma_{42}} = \frac{21}{1} = 21, \quad \frac{\beta_5}{\gamma_{52}} = \frac{9}{1} = 9.$$

The first fraction achieves the minimal value and thus defines the pivot row to be $p = 3 \in B$*.*

Table 1:

	x_1	x_3	
x_2	$\frac{1}{4}$	$\frac{1}{4}$	6
x_4	$\frac{11}{4}$	$-\frac{1}{4}$	15
x_5	$\boxed{\frac{3}{4}}$	$-\frac{1}{4}$	3
	$\frac{3}{4}$	$-\frac{5}{4}$	-30

The checks for optimality and unboundedness fail again. Pivot row and column: $p = 5$, $q = 1$, *pivot element:* $\gamma_{pq} = \frac{3}{4}$.
Table 2:

	x_5	x_3	
x_2	$-\frac{1}{3}$	$\frac{1}{3}$	5
x_4	$-\frac{11}{3}$	$\frac{2}{3}$	4
x_1	$\frac{4}{3}$	$-\frac{1}{3}$	4
	-1	-1	-33

Table 2 is optimal. The optimal solution is $x_2 = 5$, $x_4 = 4$, $x_1 = 4$, $x_5 = 0$, *and* $x_3 = 0$. *The optimal objective function value is* -33.

The simplex method requires to solve many linear equations in order to compute Γ and β after each basis change. This effort can be reduced by so-called update formulae for Γ, β, ζ, and d. It can be shown that the updated simplex table after a basis change is given by the following table, where γ_{ij}, ζ_j, β_i denote the entries of the current table:

	$x_j, j \in N \setminus \{q\}$	x_p	
$x_i, i \in B \setminus \{p\}$	$\gamma_{ij} - \frac{\gamma_{iq}\gamma_{pj}}{\gamma_{pq}}$	$-\frac{\gamma_{iq}}{\gamma_{pq}}$	$\beta_i - \frac{\gamma_{iq}\beta_p}{\gamma_{pq}}$
x_q	$\frac{\gamma_{pj}}{\gamma_{pq}}$	$\frac{1}{\gamma_{pq}}$	$\frac{\beta_p}{\gamma_{pq}}$
	$\zeta_j - \frac{\zeta_q\gamma_{pj}}{\gamma_{pq}}$	$-\frac{\zeta_q}{\gamma_{pq}}$	$d - \frac{\zeta_q\beta_p}{\gamma_{pq}}$

Remark 2.4.12
The vector ζ *in an optimal table indicates whether the solution is unique. If* $\zeta < 0$ *holds in an optimal simplex table, then the solution of the LP is unique. If there exists a component* $\zeta_j = 0$, $j \in N$, *in an optimal simplex table, then the optimal solution may not be unique. Further optimal solutions can be computed by performing additional basis changes by choosing those pivot columns with* $\zeta_j = 0$, $j \in N$. *By investigation of all possible basis changes it is possible to compute all optimal vertices. According to part (b) of the Fundamental Theorem of Linear Programming every optimal solution can be expressed as a convex combination of these optimal vertices.*

2.5 Phase 1 of the Simplex Method

A feasible basic solution is required to start the simplex method. In many cases such a solution can be obtained as follows.

Theorem 2.5.1 (Canonical Problems)
Consider an LP in canonical form 2.1.1

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax \le b, \; x \ge 0.$$

If $b \ge 0$ holds, then a feasible basic solution is given by

$$y = b \ge 0, \quad x = 0,$$

where y denotes the vector of slack variables. An initial simplex table is given by

$$\begin{array}{c|c|c} & x & \\ \hline y & A & b \\ \hline & -c & 0 \end{array}$$

Proof: Introducing slack variables leads to a problem in standard form

$$Ax + Iy = b, \quad x \ge 0, \; y \ge 0.$$

As the unity matrix I is non-singular, a feasible basic solution is given by

$$y = b \ge 0, \quad x = 0.$$

$\square$

Caution: If there exists a component $b_i < 0$, then it is much more complicated to find a feasible basic solution. The same holds true if the problem is not given in canonical form. In all these cases, the following method has to be applied.

Consider an LP in standard form 2.1.5

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b, \; x \ge 0.$$

Without loss of generality we may assume $b \ge 0$. This is not a restriction as this property can always be achieved by multiplying by -1 those equations with $b_i < 0$.
Define

Auxiliary LP:

$$\text{Minimise} \quad e^\top y = \sum_{i=1}^{m} y_i \quad \text{subject to} \quad Ax + Iy = b,\ x \geq 0,\ y \geq 0, \tag{2.19}$$

where $e = (1, \ldots, 1)^\top \in \mathbb{R}^m$.

Theorem 2.5.2 **(Feasible solution for auxiliary LP)**
Consider the auxiliary LP 2.19 with $b \geq 0$. Then, a feasible basic solution is given by

$$y = b \geq 0, \quad x = 0.$$

An initial simplex table for the auxiliary LP is given by

	x	
y	A	b
	$e^\top A$	$e^\top b$

Proof: The auxiliary LP is a standard LP for $z = (x, y)^\top$. Obviously, it holds $c = (0, e^\top)^\top$ and $y = b - Ax$ and thus $\Gamma = A$ and $\beta = b$. Moreover, $c_B = e$ and $c_N = 0$. Hence, $\zeta^\top = c_B^\top \Gamma - c_N^\top = e^\top A$ and $d = c_B^\top \beta = e^\top b$. $\square$

Theorem 2.5.3

(i) If the auxiliary LP has the optimal solution $y = 0$, then the corresponding x obtained in the simplex method is a feasible basic solution for the standard LP 2.1.5.

(ii) If the auxiliary LP has the optimal solution $y \neq 0$ and thus $e^\top y > 0$, then the standard LP 2.1.5 is infeasible.

Proof:

(i) If $y = 0$ is a solution, the corresponding x in the final simplex tableau satisfies $Ax = b$, $x \geq 0$, and thus is a feasible basic solution for the standard LP.

(ii) The objective function of the auxiliary LP is bounded from below by 0 because of $y \geq 0$ implies $e^\top y \geq 0$. Assume that the standard LP has a feasible point x. Then, $y = 0$ is feasible for the auxiliary LP because $Ax + y = b$, $x \geq 0$ and $y = 0$. Hence, $y = 0$ solves the auxiliary problem which contradicts the assumption in (ii).

□

Example 2.5.4 **(Finding a feasible basic solution)**
Consider the feasible set $Ax = b$, $x \geq 0$ with the data

$$A = \begin{pmatrix} 1 & 1 & 2 \\ -1 & 1 & 0 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Notice that it is difficult to find a feasible basic solution at a first glance. Hence, we solve the following auxiliary problem by the simplex method.

$$\text{Minimise} \quad y_1 + y_2 \quad \text{subject to} \quad Ax + Iy = b, \quad x \geq 0, \quad y = (y_1, y_2)^\top \geq 0.$$

We obtain the following solution (notice that $x_4 := y_1$ and $x_5 := y_2$ in the computation):
Initial table:

	x_1	x_2	x_3	
x_4	1	1	2	1
x_5	-1	$\boxed{1}$	0	1
	0	2	2	2

Pivot row and column: $p = 5$, $q = 2$, pivot element: $\gamma_{pq} = 1$.
Table 1:

	x_1	x_5	x_3	
x_4	$\boxed{2}$	-1	2	0
x_2	-1	1	0	1
	2	-2	2	0

Pivot row and column: $p = 4$, $q = 1$, pivot element: $\gamma_{pq} = 2$.
Table 2:

	x_4	x_5	x_3	
x_1	$\frac{1}{2}$	$-\frac{1}{2}$	1	0
x_2	$\frac{1}{2}$	$\frac{1}{2}$	1	1
	-1	-1	0	0

This table is optimal with objective function value 0 and $y_1 = y_2 = 0 (= x_4 = x_5)$. Hence, $x = (x_1, x_2, x_3)^\top = (0, 1, 0)^\top$ satisfies the constraints $Ax = b$ and $x \geq 0$. x is a feasible basic solution with basis index set $B = \{1, 2\}$ and non-basis index set $N = \{3\}$.
Suppose now that we want to minimise the function

$$x_1 + 2x_2 + x_3 \qquad (\textit{i.e. } c = (1, 2, 1)^\top)$$

on the feasible set above. The initial simplex table is given by table 2 if the columns corresponding to the auxiliary variables y_1 and y_2 (resp. x_4 and x_5) are deleted. As the *objective function has changed, we have to recompute the bottom row of the table, that is*

$$\begin{aligned} d = c_B^\top \beta &= (1,2)\begin{pmatrix} 0 \\ 1 \end{pmatrix} = 2, \\ \zeta^\top &= c_B^\top \Gamma - c_N^\top = (1,2)\begin{pmatrix} 1 \\ 1 \end{pmatrix} - (1) = 2. \end{aligned}$$

Initial table:

	x_3	
x_1	$\boxed{1}$	0
x_2	1	1
	2	2

Pivot row and column: $p = 1$, $q = 3$, *pivot element:* $\gamma_{pq} = 1$.

Table 1:

	x_1	
x_3	1	0
x_2	-1	1
	-2	2

This table is optimal.

Remark 2.5.5

In might happen that some components of y are among the basic variables in the final simplex table. In this case the columns corresponding to the components x_i, $i \in B$, do not define a full basis matrix (always m linearly independent columns are necessary). Hence, additional linearly independent columns a^j of A with $j \in N$ have to be determined in order to obtain a complete basis matrix. This can be done by inspection or by performing additional simplex steps with the aim to move the auxiliary variable y into the non-basis. This can be achieved by performing steps of the so-called dual simplex method.

Remark 2.5.6 (Finite Termination)

The simplex method does not specify which pivot column and pivot row has to be chosen if p and q are not uniquely determined. Dantzig suggested to choose the pivot column according to the formula

$$\zeta_q = \max\{\zeta_j \mid \zeta_j > 0,\ j \in N\}, \tag{2.20}$$

Marshall and Suurballe [MS69] however showed that this choice may lead to cycles in the simplex method which will not terminate in this case. This may happen, if a so-called degenerated basic solution occurs. A feasible basic solution is called degenerated, if there is an index $i \in B$ with $x_i = 0$.

Fortunately, cycles can be avoided if the following rule for the choice of the pivot element is obeyed.

Bland's rule [Bla77]
Among all possible choices for the pivot element, always choose the pivot column $q \in N$ and the pivot column $p \in B$ in step (4) of Algorithm 2.4.9 with the smallest indices q and p.

Remark 2.5.7 **(Complexity)**
Klee and Minty [KM72] showed with an example (Klee and Minty cube) in 1972 that the simplex method in its worst case has exponential complexity! Hence, the simplex method does not belong to the class of algorithms with polynomial complexity. This was a very disappointing result from a theoretical viewpoint. Nevertheless, there is also a practical point of view and the simplex method is still one of the most frequently used algorithms in linear programming, even for large-scale problems. Further investigations by Borgwardt [Bor87] show: If the input data A, b and c is reasonably distributed, then the expected effort of the simplex method is polynomial.

Chapter 3

Duality and Sensitivity

This chapter provides an introduction to duality theory for linear programs and points out the connection between dual programs and sensitivity analysis for linear programs. Sensitivity analysis addresses the question of how optimal solutions depend on perturbations in the problem data (A, b, and c). Finally, the results from sensitivity analysis will be used to derive so-called shadow prices which play an important role in economy.
Dual linear programs are important because:

- Dual problems allow to determine lower bounds for the optimal objective function value of a primal minimisation problem.
- Dual problems allow to compute sensitivities of linear programs.
- Occasionally, the dual problem is easier to solve than the primal problem.

3.1 The Dual Linear Program

Before we state the dual problem of a standard LP in a formal way, we consider two motivations for the so-called dual problem.

Example 3.1.1

A company produces three products P_1, P_2, and P_3. The profit of the products per unit amounts to 10, 5, and 5.5 pounds, respectively. The production requires four raw materials B_1, B_2, B_3, and B_4. There are 1500, 200, 1200, and 900 units of these raw materials in stock. The amount of raw materials needed for the production of one unit of the product is given by the following table.

	P_1	P_2	P_3
B_1	30	10	50
B_2	5	0	3
B_3	20	10	50
B_4	10	20	30

Let x_i, $i = 1, 2, 3$, denote the amount of product P_i, $i = 1, 2, 3$, to be produced. The company aims at maximising the profit and hence solves the following so-called primal problem:

Maximise

$$10x_1 + 5x_2 + 5.5x_3$$

subject to

$$\begin{aligned} 30x_1 + 10x_2 + 50x_3 &\leq 1500, \\ 5x_1 + 3x_3 &\leq 200, \\ 20x_1 + 10x_2 + 50x_3 &\leq 1200, \\ 10x_1 + 20x_2 + 30x_3 &\leq 900, \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

Assume now, that a second company offers to buy all raw materials from the first company. The second company offers a price of $\lambda_i \geq 0$ *pounds for one unit of the raw material* B_i, $i = 1, \ldots, 4$. *Of course, the second company intends to minimise its total costs*

$$1500\lambda_1 + 200\lambda_2 + 1200\lambda_3 + 900\lambda_4.$$

Moreover, the first company will only accept the offer of the second company if the resulting price for one unit of the product P_j *is greater than or equal to the (not realised) profit* c_j, $j = 1, 2, 3$, *i.e. if*

$$\begin{aligned} 30\lambda_1 + 5\lambda_2 + 20\lambda_3 + 10\lambda_4 &\geq 10, \\ 10\lambda_1 + 10\lambda_3 + 20\lambda_4 &\geq 5, \\ 50\lambda_1 + 3\lambda_2 + 50\lambda_3 + 30\lambda_4 &\geq 5.5, \end{aligned}$$

hold. Summarising, the second company has to solve the following linear program:
Minimise

$$1500\lambda_1 + 200\lambda_2 + 1200\lambda_3 + 900\lambda_4.$$

subject to

$$\begin{aligned} 30\lambda_1 + 5\lambda_2 + 20\lambda_3 + 10\lambda_4 &\geq 10, \\ 10\lambda_1 + 10\lambda_3 + 20\lambda_4 &\geq 5, \\ 50\lambda_1 + 3\lambda_2 + 50\lambda_3 + 30\lambda_4 &\geq 5.5, \\ \lambda_1, \lambda_2, \lambda_3, \lambda_4 &\geq 0. \end{aligned}$$

This problem is called the dual problem of the primal problem.
More generally, we may associate to the primal problem

$$\text{Maximise} \quad c^\top x \quad \text{subject to} \quad Ax \leq b, \ x \geq 0$$

the dual problem

$$\text{Minimise} \quad b^\top \lambda \quad \text{subject to} \quad A^\top \lambda \geq c,\ \lambda \geq 0.$$

A second motivation is based on the investigation of perturbed linear programs. We will use this approach to derive the dual problem for the standard problem, which is considered to be the primal problem.

Definition 3.1.2 **(Primal problem (P))**

The standard LP

$$(P) \qquad \text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b,\ x \geq 0$$

with $A \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ is called primal problem.

In practical applications, the vector b often is used to model capacities (budget, resources, size, etc.) while the objective function often denotes the costs. An important economical question is the following: How does a variation in b (i.e. a variation in budget, resources, size) influence the optimal objective function value (i.e. the costs)?
To answer this question, the primal problem (P) is embedded into a family of perturbed linear programs:

Definition 3.1.3 **(Perturbed problem (P_δ))**

For an arbitrary $\delta \in \mathbb{R}^m$ the LP

$$(P_\delta) \qquad \text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b + \delta,\ x \geq 0$$

with $A \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ is called perturbed primal problem. The set

$$M_\delta := \{x \in \mathbb{R}^n \mid Ax = b + \delta,\ x \geq 0\}$$

is the feasible set of the perturbed primal problem.

Obviously, (P_δ) with $\delta = 0$ is identical with the primal problem (P) which is referred to as the unperturbed problem.
We may assign to each $\delta \in \mathbb{R}^m$ the optimal objective function value of (P_δ), provided that (P_δ) has an optimal solution.

Definition 3.1.4 **(Optimal value function)**

The optimal value function is defined as

$$w(\delta) := \begin{cases} \inf\{c^\top x \mid Ax = b + \delta,\ x \geq 0\}, & \text{if } M_\delta \neq \emptyset, \\ \infty, & \text{if } M_\delta = \emptyset. \end{cases}$$

The optimal value function has the following properties:

- The optimal value function is convex.
- The optimal value function is piecewise linear.

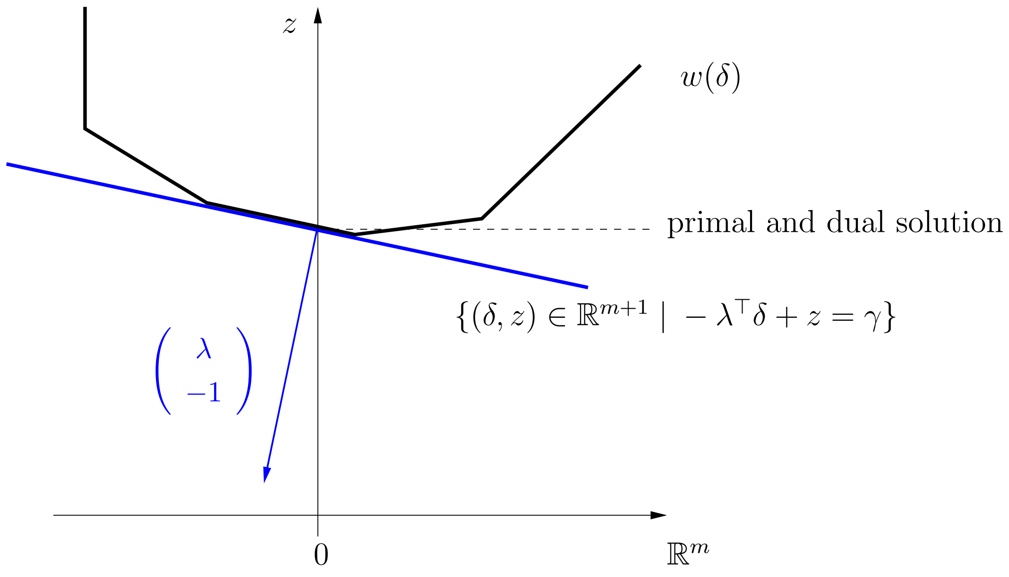

z
$w(\delta)$
primal and dual solution
$\{(\delta, z) \in \mathbb{R}^{m+1} \mid -\lambda^\top \delta + z = \gamma\}$
$\begin{pmatrix} \lambda \\ -1 \end{pmatrix}$
0
$\mathbb{R}^m$

Figure 3.1: Graphical interpretation of the dual problem: Approximation of the optimal value function from below by a hyperplane with normal vector $(-\lambda, 1)^\top$ resp. $(\lambda, -1)^\top$.

The dual problem is informally defined by the following task: Approximate the optimal value function at $\delta = 0$ (unperturbed problem=primal problem) from below as good as possible by a hyperplane

$$H = \{(\delta, z) \in \mathbb{R}^{m+1} \mid -\lambda^\top \delta + z = \gamma\}.$$

Herein, the normal vector $\begin{pmatrix} -\lambda \\ 1 \end{pmatrix} \in \mathbb{R}^{m+1}$ and γ are not fixed but can be chosen appropriately.

We are particularly interested in the quality of the approximation at the point $\delta = 0$ respectively $(0, \gamma)^\top \in H$ because the hyperplane can be viewed as a 'linearisation' of the optimal value function at $\delta = 0$. Hence, the slope of this 'linearisation' provides information on how the optimal value function behaves locally around $\delta = 0$, i.e. the slope essentially indicates how sensitive the problem is with respect to perturbations δ in b. We will come back to this issue later.

We have to specify what 'as good as possible' in the informal definition means. This is done in the following formal definition of the dual problem:

Dual problem (D):

Find $\lambda \in \mathbb{R}^m$ such that γ becomes maximal subject to the constraint

$$z \leq w(\delta) \quad \forall \delta \in \mathbb{R}^m,$$

where $z = \gamma + \lambda^\top \delta$.

Transformations:

$$\begin{aligned}
(D) \quad &\Leftrightarrow \quad \max_{\lambda \in \mathbb{R}^m} \gamma \text{ subject to } \gamma + \lambda^\top \delta \leq w(\delta) \qquad \forall \delta \in \mathbb{R}^m \\
&\Leftrightarrow \quad \max_{\lambda \in \mathbb{R}^m} \gamma \text{ subject to } \gamma \leq c^\top x - \lambda^\top (Ax - b) \qquad \forall x \geq 0 \\
&\Leftrightarrow \quad \max_{\lambda \in \mathbb{R}^m} \gamma \text{ subject to } \gamma \leq (c^\top - \lambda^\top A)x + \lambda^\top b \qquad \forall x \geq 0
\end{aligned}$$

It holds

$$\inf_{x \geq 0} \left(c^\top - \lambda^\top A\right) x + \lambda^\top b \quad = \quad \begin{cases} \lambda^\top b, & \text{if } c^\top - \lambda^\top A \geq 0, \\ -\infty, & \text{otherwise.} \end{cases}$$

Hence, the dual problem is equivalent to the following linear program:

Definition 3.1.5 (Dual problem (D))

The linear program

$$(D) \qquad \textit{Maximise} \quad b^\top \lambda \quad \textit{subject to} \quad A^\top \lambda \leq c$$

is called the dual problem of (P).

Summarising: The primal problem

$$(P) \qquad \text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b, \ x \geq 0$$

is associated the dual problem

$$(D) \qquad \text{Maximise} \quad b^\top \lambda \quad \text{subject to} \quad A^\top \lambda \leq c.$$

Example 3.1.6

Let the primal LP be given by the following linear program:
Minimise $-3x_1 - 4x_2$ *subject to*

$$\begin{aligned}
2x_1 + x_2 + x_3 &= 8, \\
4x_1 + x_2 + x_4 &= 10, \\
x_1, x_2, x_3, x_4 &\geq 0.
\end{aligned}$$

The dual problem reads as:

Maximise $8\lambda_1 + 10\lambda_2$ *subject to*

$$\begin{aligned} 2\lambda_1 + 4\lambda_2 &\leq -3, \\ \lambda_1 + \lambda_2 &\leq -4, \\ \lambda_1 &\leq 0, \\ \lambda_2 &\leq 0. \end{aligned}$$

By application of the well-known transformation techniques, which allow to transform a general LP into a standard LP, and writing down the dual problem of the resulting standard problem, it is possible to formulate the dual problem of a general LP. We demonstrate this procedure for the following problem (compare Example 3.1.1). Please note the nice symmetry of the primal and dual problem!

Theorem 3.1.7

The dual problem of the LP

$$\textit{Maximise} \quad c^\top x \quad \textit{subject to} \quad Ax \leq b,\ x \geq 0$$

is given by

$$\textit{Minimise} \quad b^\top \lambda \quad \textit{subject to} \quad A^\top \lambda \geq c,\ \lambda \geq 0.$$

Proof: Transformation into standard form yields:

$$\text{Minimise} \quad \begin{pmatrix} -c^\top & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{subject to} \quad \begin{pmatrix} A & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = b, \begin{pmatrix} x \\ y \end{pmatrix} \geq 0.$$

The dual problem of this standard problem reads as

$$\text{Maximise} \quad b^\top \tilde{\lambda} \quad \text{subject to} \quad \begin{pmatrix} A^\top \\ I \end{pmatrix} \tilde{\lambda} \leq \begin{pmatrix} -c \\ 0 \end{pmatrix}.$$

Defining $\lambda := -\tilde{\lambda}$ yields the problem

$$\text{Maximise} \quad -b^\top \lambda \quad \text{subject to} \quad -A^\top \lambda \leq -c,\ -\lambda \leq 0,$$

which proves the assertion. □

In a similar way, the following scheme applies. Notice in the scheme below, that the primal problem is supposed to be a minimisation problem. As we know already, this can always be achieved by multiplying the objective function of a maximisation problem by -1.

primal constraints (minimise $c^\top x$)	dual constraints (maximise $b^\top \lambda$)
$x \geq 0$	$A^\top \lambda \leq c$
$x \leq 0$	$A^\top \lambda \geq c$
x free	$A^\top \lambda = c$
$Ax = b$	λ free
$Ax \leq b$	$\lambda \leq 0$
$Ax \geq b$	$\lambda \geq 0$

The same scheme applies component-wise if primal constraints of the different types

$$x_i \left\{ \begin{array}{c} \geq 0 \\ \leq 0 \\ \text{free} \end{array} \right\}, \; i = 1, \dots, n, \qquad \sum_{j=1}^{n} a_{ij} x_j \left\{ \begin{array}{c} = \\ \leq \\ \geq \end{array} \right\} b_i, \; i = 1, \dots, m,$$

occur simultaneously in a general LP. More precisely, the dual variable λ_i is associated with the i-th primal constraint according to the scheme

$$\sum_{j=1}^{n} a_{ij} x_j \left\{ \begin{array}{c} = \\ \leq \\ \geq \end{array} \right\} b_i \qquad \leftrightarrow \qquad \lambda_i \left\{ \begin{array}{c} \text{free} \\ \leq 0 \\ \geq 0 \end{array} \right\}.$$

Moreover, the i-th primal variable x_i is associated with the i-th component of the dual constraint according to the scheme

$$x_i \left\{ \begin{array}{c} \geq 0 \\ \leq 0 \\ \text{free} \end{array} \right\} \qquad \leftrightarrow \qquad (A^\top \lambda)_i \left\{ \begin{array}{c} \leq \\ \geq \\ = \end{array} \right\} c_i.$$

In the latter, $(A^\top \lambda)_i$ denotes the i-th component of the vector $A^\top \lambda$, where $A = (a_{ij})$ denotes the matrix of coefficients of the primal constraints (excluding sign conditions of course). Again, the above relations assume that the primal problem is a minimisation problem.

Remark 3.1.8

Dualisation of the dual problem yields the primal problem again.

3.2 Weak and Strong Duality

We summarise important relations between the primal problem and its dual problem.

Theorem 3.2.1 (Weak Duality Theorem)

Let x be feasible for the primal problem (P) (i.e. $Ax = b$, $x \geq 0$) and let λ be feasible for the dual problem (D) (i.e. $A^\top \lambda \leq c$). Then it holds

$$b^\top \lambda \leq c^\top x.$$

Proof: Owing to $b = Ax$ and $x \geq 0$ it holds

$$b^\top \lambda = (Ax)^\top \lambda = x^\top A^\top \lambda \leq x^\top c = c^\top x.$$

□

The weak duality theorem provides a motivation for the dual problem: dual feasible points provide lower bounds for the optimal objective function value of the primal problem. Vice versa, primal feasible points provide upper bounds for the optimal objective function value of the dual problem. This property is very important in the context of Branch & Bound methods for integer programs.
Moreover, it holds

Theorem 3.2.2 **(Sufficient Optimality Criterion)**
Let x be feasible for the primal problem (P) and let λ be feasible for the dual problem (D).

(i) If $b^\top \lambda = c^\top x$, then x is optimal for the primal problem (P) and λ is optimal for the dual problem (D).

(ii) $b^\top \lambda = c^\top x$ holds if and only if the complementarity slackness condition holds:

$$x_j \left(\sum_{i=1}^{m} a_{ij}\lambda_i - c_j \right) = 0, \qquad j = 1, \ldots, n.$$

Remark 3.2.3
The complementary slackness condition is equivalent with the following: For $j = 1, \ldots, n$ it holds

$$x_j > 0 \qquad \Rightarrow \qquad \sum_{i=1}^{m} a_{ij}\lambda_i - c_j = 0$$

and

$$\sum_{i=1}^{m} a_{ij}\lambda_i - c_j < 0 \qquad \Rightarrow \qquad x_j = 0.$$

This means: Either the primal constraint $x_j \geq 0$ is active (i.e. $x_j = 0$) or the dual constraint $\sum_{i=1}^{m} a_{ij}\lambda_i \leq c_i$ is active (i.e. $\sum_{i=1}^{m} a_{ij}\lambda_i = c_i$). It cannot happen that both constraints are inactive at the same time (i.e. $x_j > 0$ and $\sum_{i=1}^{m} a_{ij}\lambda_i < c_i$).

Proof: The assertion in (i) is an immediate consequence of the weak duality theorem. The assertion in (ii) follows from

$$c^\top x - b^\top \lambda = c^\top x - \lambda^\top b = c^\top x - \lambda^\top Ax = \left(c - A^\top \lambda\right)^\top x = \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m} a_{ij}\lambda_i \right) x_j.$$

If $c^\top x = b^\top \lambda$, then owing to $x \geq 0$ and $c - A^\top \lambda \geq 0$ every term in the sum has to vanish. If, on the other hand, the complementarity conditions hold, then every term of the sum vanishes and thus $c^\top x = b^\top \lambda$. □

The sufficient optimality condition rises the question, whether $c^\top x = b^\top \lambda$ actually can be obtained. We will answer this question by exploitation of the simplex method. We have seen before that the simplex method with Bland's rule (and assuming $\text{rank}(A) = m$) always terminates, either with an optimal feasible basic solution or with the message that the problem does not have a solution. Let us investigate the first case. Let x be an optimal feasible basic solution of the primal problem (P) with basis index set B, which has been computed by the simplex method using Bland's rule. Hence, it holds $x_B \geq 0$ and $x_N = 0$. Moreover, the optimality criterion

$$\zeta^\top = c_B^\top A_B^{-1} A_N - c_N^\top \leq 0$$

holds. Let λ be defined by

$$\lambda^\top := c_B^\top A_B^{-1}.$$

We will show that this λ solves the dual problem! Firstly, it holds

$$c^\top x = c_B^\top x_B = c_B^\top A_B^{-1} b = \lambda^\top b = b^\top \lambda.$$

Secondly, λ is feasible for the dual problem (D) because

$$\begin{aligned} A_B^\top \lambda &= c_B, \\ A_N^\top \lambda - c_N &= A_N^\top \left(A_B^\top\right)^{-1} c_B - c_N = \zeta \leq 0. \end{aligned}$$

The latter is just the primal optimality criterion. Combining both relations yields $A^\top \lambda \leq c$. Hence, we have shown:

If the primal problem (P) has an optimal solution, then the dual problem (D) has an optimal solution, too, and an optimal dual solution is given by $\lambda^\top = c_B^\top A_B^{-1}$ (there may be other solutions).

The converse can be shown as well, if we exploit the fact that the dual of the dual problem is equivalent to the primal problem.

The following theorem is the main result of this section.

Theorem 3.2.4 **(Strong Duality Theorem)**

The primal problem (P) has an optimal solution x if and only if the dual problem (D) has an optimal solution λ. Moreover, the primal and dual objective function values coincide if an optimal solution exists, i.e. $c^\top x = b^\top \lambda$.

Remark 3.2.5

The primal simplex method, compare Algorithm 2.4.9, computes vertices x, which are primally feasible, and dual variables λ, which satisfy $c^\top x = b^\top \lambda$ in each step. The primal simplex method stops as soon as λ becomes feasible for the dual problem, i.e. dual feasibility is the optimality criterion for the primal simplex method.

3.3 Sensitivities and Shadow Prices

The solutions of the dual problem possess an important interpretation for economical applications. Under suitable assumptions they provide the sensitivity of the primal objective function value $c^\top x$ with respect to perturbations in the vector b. In economy, the dual solutions are known as shadow prices. It was mentioned before, that the vector b often refers to capacities (budget, resources, size, etc.) while the objective function often denotes the costs. Thus, a manager is particularly interested in the question of how a variation of the capacity (e.g. a reduction of budget, resources, size) influences the costs.

Let us approach the problem graphically first.

Example 3.3.1 (compare Examples 1.1.1, 2.2.1)
A farmer intends to plant 40 acres with sugar beets and wheat. He can use up to 312 working days to achieve this. For each acre his cultivation costs amount to 40 pounds for sugar beets and to 120 pounds for wheat. For sugar beets he needs 6 working days per acre and for wheat 12 working days per acre. The profit amounts to 100 pounds per acre for sugar beets and to 250 pounds per acre for wheat. As the farmer wants to consider unforeseen expenses in his calculation, he assumes that he has a maximal budget of $2400 + \delta$ pounds, where $\delta \in \mathbb{R}$ is a perturbation potentially caused by unforeseen expenses. Of course, the farmer wants to maximise his profit (resp. minimise the negative profit).

The resulting canonical linear program reads as follows (compare Example 2.2.1):

Minimise $f(x_1, x_2) = -100x_1 - 250x_2$ subject to the constraints

$$x_1 + x_2 \le 40, \quad 40x_1 + 120x_2 \le 2400 + \delta, \quad 6x_1 + 12x_2 \le 312, \quad x_1, x_2 \ge 0.$$

In Example 2.2.1 we solved the unperturbed problem for $\delta = 0$ graphically:

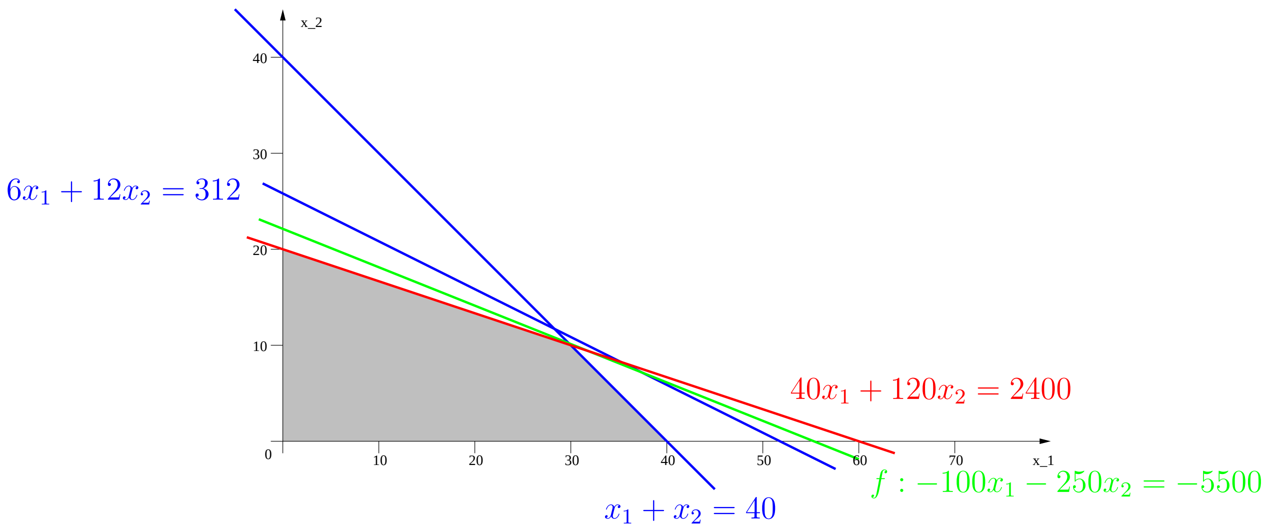


Figure 3.2: Solution of the unperturbed problem for $\delta = 0$. The optimal solution is $(x_1, x_2)^\top = (30, 10)^\top$ with objective function value -5500.

The optimal solution is obtained at the point where the first and second constraint intersect (this corresponds to x_1, x_2 being basic variables in the simplex method). What happens if perturbations $\delta \neq 0$ are considered? Well, δ influences only the second constraint. Moreover, the slope of this constraint is not affected by changing δ. Hence, changing δ results in lines which are in parallel to the red line in Figure 3.2. Figure 3.3 depicts the situation for $\delta = -600$.

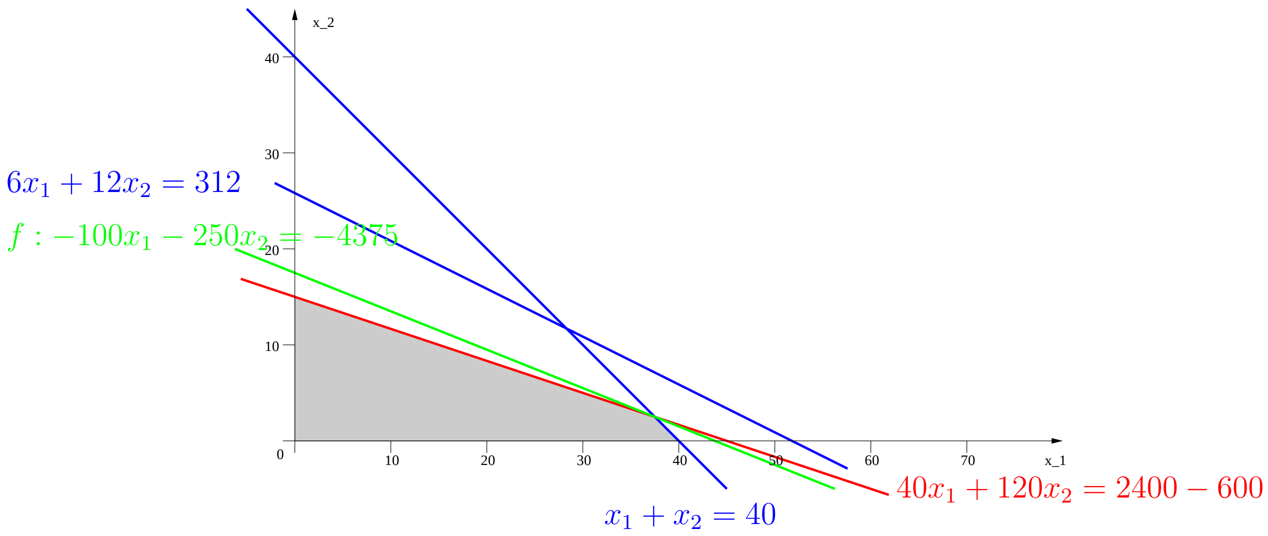


Figure 3.3: Solution of the problem for $\delta = -600$. The optimal solution is $(x_1, x_2)^\top = (37.5, 2.5)^\top$ with objective function value -4375.

Clearly, the feasible set and the optimal solution have changed. But still, the optimal solution is obtained at the point where the first and second constraint intersect (again, x_1, x_2 are among the basic variables in the simplex method).

What happens if we further reduce δ, say $\delta = -1200$? Figure 3.4 depicts the situation for $\delta = -1200$.

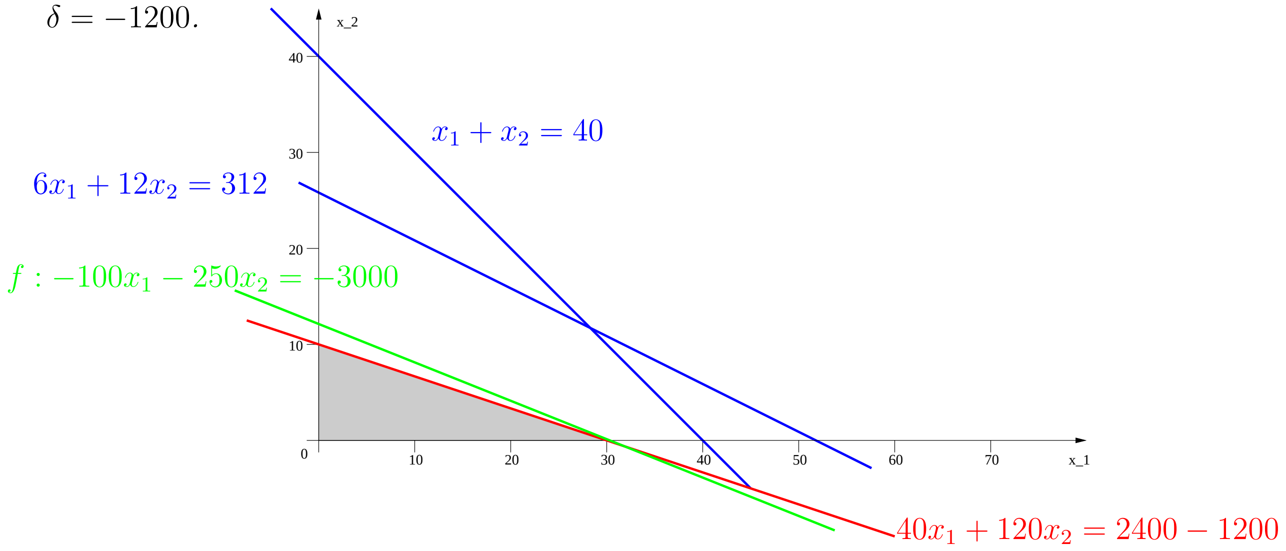


Figure 3.4: Solution of the problem for $\delta = -1200$. The optimal solution is $(x_1, x_2)^\top = (30, 0)^\top$ with objective function value -3000.

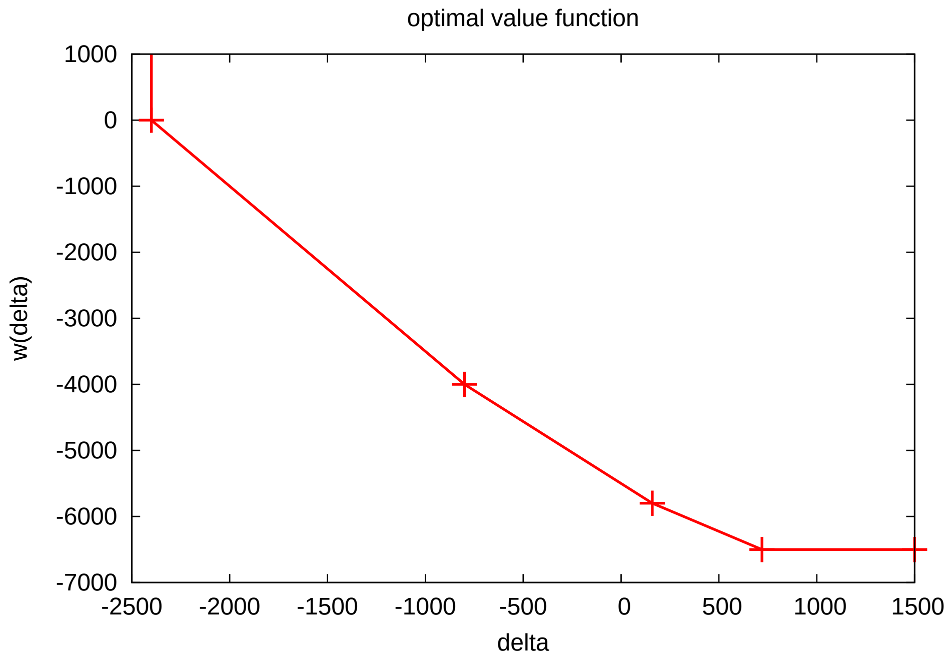
Clearly, the feasible set and the optimal solution have changed again. But now, the optimal solution is obtained at the point where the second constraint intersects with the constraint $x_2 = 0$. So, the structure of the solution has changed (now, x_2 became a non-basic variable and thus a switch of the basis index set occurred).
Finally, we could ask ourselves, how the optimal objective function value depends on δ? A graphical investigation of the problem for all values of δ yields the optimal solutions

$$\begin{pmatrix} x_1(\delta) \\ x_2(\delta) \end{pmatrix} = \begin{cases} \begin{pmatrix} 0 \\ 26 \end{pmatrix}, & \text{if } 720 \leq \delta, \\ \begin{pmatrix} 36 - \frac{1}{20}\delta \\ 8 + \frac{1}{40}\delta \end{pmatrix}, & \text{if } 160 \leq \delta < 720, \\ \begin{pmatrix} 30 - \frac{1}{80}\delta \\ 10 + \frac{1}{80}\delta \end{pmatrix}, & \text{if } -800 \leq \delta < 160, \\ \begin{pmatrix} 60 + \frac{1}{40}\delta \\ 0 \end{pmatrix}, & \text{if } -2400 \leq \delta < -800, \\ \text{no solution}, & \text{if } \delta < -2400, \end{cases}$$

and the optimal value function, cf. Definition 3.1.4,

$$w(\delta) = c^\top x(\delta) = \begin{cases} -6500, & \text{if } 720 \leq \delta, \\ -5600 - 1.25\delta, & \text{if } 160 \leq \delta < 720, \\ -5500 - 1.875\delta, & \text{if } -800 \leq \delta < 160, \\ -6000 - 2.5\delta, & \text{if } -2400 \leq \delta < -800, \\ \infty, & \text{if } \delta < -2400, \end{cases}$$

which is piecewise linear and convex:



In the sequel we restrict the discussion to perturbations of the vector b only. Perturbations in A and c can be investigated as well, but the situation is more complicated. We consider the perturbed primal problem in Definition 3.1.3, i.e.

$$\text{Minimise} \qquad c^\top x \qquad \text{subject to} \qquad Ax = b + \delta,\ x \geq 0.$$

Obviously, for $\delta = 0$ the primal problem (cf. Definition 3.1.2) arises. This problem is referred to as the unperturbed problem or nominal problem. Each perturbation δ may be assigned the corresponding optimal solution (if it exists), i.e.

$$\delta \mapsto x(\delta).$$

Introducing this function $x(\delta)$ into the objective function $c^\top x$ leads to the optimal value function as defined in Definition 3.1.4.
We now intend to investigate the optimal solution $x(\delta)$ of the perturbed problem 3.1.3 in a small neighbourhood of the unperturbed solution at $\delta = 0$, i.e. we consider perturbations $\delta \in \mathbb{R}^m$ sufficiently close to zero.
Moreover, we will assume the following:

- the unperturbed problem with $\delta = 0$ possesses the feasible basic solution $x_B(0) = A_B^{-1}b$, $x_N(0) = 0$ with basis index set B and non-basis index set N.
- the solution is not degenerate, i.e. $x_B(0) > 0$.

An extension to degenerate solutions is possible, but more involved.
Now we introduce perturbations $\delta \neq 0$ sufficiently close to zero and define

$$x_B(\delta) := A_B^{-1}(b+\delta), \qquad x_N(\delta) := 0. \tag{3.1}$$

The point given by $x_B(\delta)$ and $x_N(\delta)$ is feasible as long as

$$x_B(\delta) = A_B^{-1}(b+\delta) = x_B(0) + A_B^{-1}\delta \geq 0$$

holds. As x is supposed to be non-degenerate, i.e. $x_B(0) = A_B^{-1}b > 0$, feasibility can be guaranteed for perturbations δ sufficiently close to zero owing to the continuity of the function $\delta \mapsto x_B(0) + A_B^{-1}\delta$.
What about the optimality criterion for $x_B(\delta)$ and $x_N(\delta)$? Interestingly, perturbations δ in b do not influence the optimality criterion of the simplex method at all because the vector ζ does not depend on b:

$$\zeta^\top = c_B^\top A_B^{-1} A_N - c_N^\top.$$

As $x_B(0)$ was supposed to be optimal it holds $\zeta \leq 0$. This optimality condition is still satisfied for $x_B(\delta)$ and $x_N(\delta)$ in (3.1) and hence, (3.1) is optimal for the perturbed problem!
The corresponding optimal objective function value computes to

$$w(\delta) = c_B^\top x_B(\delta) = c_B^\top x_B(0) + c_B^\top A_B^{-1}\delta = w(0) + \lambda^\top \delta,$$

where $\lambda = (A_B^{-1})^\top c_B$ is a solution of the dual problem. Hence:

Theorem 3.3.2 (Sensitivity Theorem)
Let rank$(A) = m$ and let the unperturbed problem possess an optimal feasible basic solution with basis index set B and non-basis index set N which is not degenerate. Then, for all perturbations $\delta = (\delta_1, \ldots, \delta_m)^\top$ sufficiently close to zero,

$$x_B(\delta) = A_B^{-1}(b+\delta), \qquad x_N(\delta) = 0,$$

is an optimal feasible basic solution for the perturbed problem and it holds

$$\Delta w := w(\delta) - w(0) = \lambda^\top \delta = \sum_{i=1}^{m} \lambda_i \delta_i.$$

The dual solution $\lambda^\top = c_B^\top A_B^{-1}$ indicates the sensitivity of the optimal objective function value w.r.t. perturbations in b.

The sensitivity theorem shows:

- If $|\lambda_i|$ is large, then perturbations δ_i with $|\delta_i|$ small have a comparatively large influence on the optimal objective function value. The problem is sensitive w.r.t. to perturbations.

- If $|\lambda_i|$ is small, then perturbations δ_i with $|\delta_i|$ small have a comparatively small influence on the optimal objective function value. The problem is not sensitive w.r.t. to perturbations.

- If $\lambda_i = 0$, then perturbations δ_i close to zero have no influence on the optimal objective function value.

The sensitivity theorem states in particular, that under the assumptions of the theorem, $x_B(\delta)$ and $w(\delta)$ are continuously differentiable at $\delta = 0$. Differentiation of the optimal value function w.r.t. δ yields

Shadow price formula:

$$w'(\delta) = \lambda.$$

Example 3.3.3 (compare Example 3.3.1)

We consider again Example 3.3.1:

Minimise $f(x_1, x_2) = -100x_1 - 250x_2$ subject to the constraints

$$x_1 + x_2 \le 40, \quad 40x_1 + 120x_2 \le 2400 + \delta, \quad 6x_1 + 12x_2 \le 312, \quad x_1, x_2 \ge 0.$$

The simplex method for the unperturbed problem with $\delta = 0$ yields:

Initial table:

	x_1	x_2	
x_3	$\boxed{1}$	1	40
x_4	40	120	2400
x_5	6	12	312
	100	250	0

Table 1:

	x_3	x_2	
x_1	1	1	40
x_4	-40	$\boxed{80}$	800
x_5	-6	6	72
	-100	150	-4000

Table 2:

	x_3	x_4	
x_1	1.5	-0.0125	30
x_2	-0.5	0.0125	10
x_5	-3	-0.075	12
	-25	-1.875	-5500

Table 2 is optimal. The optimal solution $x = (30, 10, 0, 0, 12)^\top$ is not degenerate. The corresponding dual solution is

$$\lambda^\top = c_B^\top A_B^{-1} = (-25, -1.875, 0),$$

where $B = \{1, 2, 5\}$ and

$$c_B = \begin{pmatrix} -100 \\ -250 \\ 0 \end{pmatrix}, \qquad A_B = \begin{pmatrix} 1 & 1 & 0 \\ 40 & 120 & 0 \\ 6 & 12 & 1 \end{pmatrix}, \qquad A_B^{-1} = \begin{pmatrix} \frac{3}{2} & -\frac{1}{80} & 0 \\ -\frac{1}{2} & \frac{1}{80} & 0 \\ -3 & -\frac{3}{40} & 1 \end{pmatrix}.$$

The sensitivity theorem states:

(i)

$$\underbrace{\begin{pmatrix} x_1(\delta) \\ x_2(\delta) \\ x_5(\delta) \end{pmatrix}}_{=x_B(\delta)} = \underbrace{\begin{pmatrix} x_1(0) \\ x_2(0) \\ x_5(0) \end{pmatrix}}_{=x_B(0)} + A_B^{-1} \begin{pmatrix} 0 \\ \delta \\ 0 \end{pmatrix} = \begin{pmatrix} 30 \\ 10 \\ 12 \end{pmatrix} + \delta \begin{pmatrix} -\frac{1}{80} \\ \frac{1}{80} \\ -\frac{3}{40} \end{pmatrix}$$

is optimal for δ sufficiently close to zero. How large may the perturbation δ be? Well, $x_B(\delta)$ has to be feasible, i.e.

$$30 - \frac{1}{80}\delta \geq 0, \quad 10 + \frac{1}{80}\delta \geq 0, \quad 12 - \frac{3}{40}\delta \geq 0.$$

These three inequalities are satisfied if $\delta \in [-800, 160]$. Hence, for all $\delta \in [-800, 160]$, the above $x_B(\delta)$ is optimal.

(ii)

$$\Delta w = w(\delta) - w(0) = \lambda^\top \begin{pmatrix} 0 \\ \delta \\ 0 \end{pmatrix} = -1.875\delta.$$

The change of the negative profit w.r.t. δ amounts to -1.875δ.

Compare this with our graphical investigation in Example 3.3.1!

Remark 3.3.4 (Caution!)

The assumption that the unperturbed solution is not degenerate is essential and cannot be *dropped! The above analysis only works, if the index sets B and N don't change! This can only be guaranteed for perturbations δ sufficiently close to zero. For large perturbations or in the degenerate case, the index sets B and N usually do change and the optimal value function is not differentiable at those points, where the index sets change. Example 3.3.1 clearly shows, that this does happen.*

3.4 A Primal-Dual Algorithm

This section discusses an algorithm which aims to satisfy the complementary slackness conditions. This algorithm will be particularly useful for shortest path problems later on. We consider the primal linear program

$$(P) \qquad \text{Minimise} \qquad c^\top x \qquad \text{subject to} \qquad Ax = b,\ x \geq 0$$

and the corresponding dual problem

$$(D) \qquad \text{Maximise} \qquad b^\top \lambda \qquad \text{subject to} \qquad A^\top \lambda \leq c$$

with $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$, and $\lambda \in \mathbb{R}^m$. Without loss of generality we assume $b \geq 0$. Moreover, let a^j, $j = 1,\dots,n$, denote the columns of A.
Our aim is to construct an algorithm that computes a feasible x for (P) and a feasible λ for (D) such that the complementary slackness conditions

$$x_j\left((A^\top\lambda)_j - c_j\right) = 0, \qquad j = 1,\dots,n,$$

hold, where $(A^\top\lambda)_j = (a^j)^\top\lambda$ denotes the j-th component of the vector $A^\top\lambda$. Theorem 3.2.2 guarantees then the optimality of x and λ.

Construction of the algorithm:

- Let a feasible $\lambda \in \mathbb{R}^m$ for (D) be given (if $c \geq 0$, we can choose $\lambda = 0$; otherwise we have to solve an auxiliary problem first).

  Define the index set

  $$B := \{j \in \{1,\dots,n\} \mid (A^\top\lambda)_j = c_j\}$$

  of active dual constraints.

- The complementary slackness conditions will be satisfied if we can find a feasible x for (P) with

  $$x_j \underbrace{\left((A^\top\lambda)_j - c_j\right)}_{<0 \text{ for } j\notin B} = 0, \qquad \forall j \notin B,$$

which is the case if and only if $x_j = 0$ holds for all $j \notin B$. Summarising, we have to find $x \in \mathbb{R}^n$ with

$$\begin{aligned} \sum_{j\in B} a^j x_j = A_B x_B &= b, & \\ x_j &\geq 0, & \forall j \in B, \\ x_j &= 0, & \forall j \notin B. \end{aligned}$$

This can be achieved by solving an auxiliary linear program which was already useful to find an initial feasible basic solution for the simplex method (Phase 1). This problem is obtained by introducing an artificial slack variable $y \in \mathbb{R}^m$ with non-negative components $y_i \geq 0$, $i = 1, \dots, m$, and is referred to as Restricted Primal:

$$(RP) \qquad \begin{array}{ll} \text{Minimise} & \displaystyle\sum_{i=1}^m y_i \\ \text{subject to} & A_B x_B + y = b, \\ & x_j \geq 0, \qquad j \in B, \\ & x_j = 0, \qquad j \notin B, \\ & y_i \geq 0, \qquad i = 1, \dots, m. \end{array}$$

This linear program can be solved by, e.g., the simplex method. Notice that (RP) always has an optimal solution as the objective function is bounded below by 0 on the feasible set and $x = 0$ and $y = b \geq 0$ is feasible.

Let f_{opt} denote the optimal objective function value of (RP) and let (x_{opt}, y_{opt}) be an optimal solution of (RP).

- Two cases may occur:
  - (i) If $f_{opt} = 0$, then the corresponding solution x_{opt} is feasible for (P) and satisfies the complementary slackness conditions. Thus, x_{opt} is optimal for (P) and λ is optimal for (D).
  - (ii) If $f_{opt} > 0$, then the corresponding solution x_{opt} is not feasible for (P) as it only satisfies $Ax_{opt} + y_{opt} = b$ with a non-zero vector y_{opt}. How can we proceed? To answer this question, we will exploit the dual problem of (RP) which is given by

$$(DRP) \qquad \begin{array}{ll} \text{Maximise} & b^\top \lambda \\ \text{subject to} & A_B^\top \lambda \leq 0, \\ & \lambda_j \leq 1, \qquad j = 1, \dots, m. \end{array}$$

Let $\bar{\lambda}$ denote an optimal solution of this problem. Owing to the strong duality theorem we have $b^\top \bar{\lambda} = f_{opt} > 0$.

We intend to improve the dual objective function by computing a better dual vector

$$\lambda^* = \lambda + t\bar{\lambda}$$

for some $t \in \mathbb{R}$. The dual objective function value then computes to

$$b^\top \lambda^* = b^\top \lambda + t \underbrace{b^\top \bar{\lambda}}_{>0}$$

and it will increase for $t > 0$.
Moreover, we want to maintain dual feasibility of λ^*, i.e.

$$(A^\top \lambda^*)_j = \underbrace{(A^\top \lambda)_j}_{\leq c_j} + t(A^\top \bar{\lambda})_j \leq c_j, \qquad \forall j = 1, \ldots, n.$$

As $\bar{\lambda}$ satisfies $A_B^\top \bar{\lambda} \leq 0$ resp. $(a^j)^\top \bar{\lambda} = (A^\top \bar{\lambda})_j \leq 0$ for $j \in B$, the above inequality is satisfied for every $t \geq 0$ for $j \in B$.
It remains to check dual feasibility for $j \notin B$. Two cases:

(iia) If $(A^\top \bar{\lambda})_j \leq 0$ for all $j \notin B$, then dual feasibility is maintained for every $t \geq 0$. For $t \to \infty$ the dual objective function is unbounded because $b^\top \lambda^* \to \infty$ for $t \to \infty$. Hence, in this case the dual objective function is unbounded above and the dual does not have a solution. According to the strong duality theorem the primal problem does not have a solution as well.

(iib) If $(A^\top \bar{\lambda})_j > 0$ for some $j \notin B$, then dual feasibility is maintained for

$$t_{max} := \min \left\{ \frac{c_j - (A^\top \lambda)_j}{(A^\top \bar{\lambda})_j} \;\middle|\; j \notin B,\ (A^\top \bar{\lambda})_j > 0 \right\}.$$

Hence, $\lambda^* = \lambda + t_{max}\bar{\lambda}$ is feasible for (D) and improves the dual objective function value.

These considerations lead to the following algorithm.

Algorithm 3.4.1 (Primal-Dual Algorithm)

(0) Let λ be feasible for (D).

(1) Set $B = \{j \in \{1, \ldots, n\} \mid (A^\top \lambda)_j = c_j\}$.

(2) Solve (RP) (for the time being by the simplex method). Let f_{opt} denote the optimal objective function value and let $\bar{\lambda}$ be an optimal solution of (DRP).

(3) If $f_{opt} = 0$, then STOP: optimal solution found; x is optimal for (P) and λ is optimal for (D).

(4) If $f_{opt} > 0$ and $(A^\top \bar{\lambda})_j \leq 0$ for every $j \notin B$, then STOP: (D) is unbounded and (P) is infeasible.

(5) Compute

$$t_{max} := \min \left\{ \frac{c_j - (A^\top \lambda)_j}{(A^\top \bar{\lambda})_j} \;\middle|\; j \notin B,\ (A^\top \bar{\lambda})_j > 0 \right\}$$

set $\lambda := \lambda + t_{max}\bar{\lambda}$, and go to (1).

Example 3.4.2

Consider the standard LP

$$(P) \qquad \text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b,\ x \geq 0,$$

with

$$c = \begin{pmatrix} -2 \\ -3 \\ 0 \\ 0 \end{pmatrix}, \quad b = \begin{pmatrix} 8 \\ 9 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}.$$

The dual is given by

$$(D) \qquad \text{Maximise} \quad b^\top \lambda \quad \text{subject to} \quad A^\top \lambda \leq c,$$

i.e. by

$$\begin{array}{lrcl} \text{Maximise} & 8\lambda_1 + 9\lambda_2 & & \\ \text{subject to} & \lambda_1 + 2\lambda_2 & \leq & -2, \\ & 2\lambda_1 + \lambda_1 & \leq & -3, \\ & \lambda_1 & \leq & 0, \\ & \lambda_2 & \leq & 0. \end{array}$$

Iteration 1:

$\lambda = (-1,-1)^\top$ is feasible for (D) because $A^\top \lambda = (-3,-3,-1,-1)^\top \leq c$ and only the second constraint is active, i.e. $B = \{2\}$. The restricted primal reads as follows:

$$\begin{array}{lrcll} \text{Minimise} & y_1 + y_2 & & & \\ \text{subject to} & 2x_2 + y_1 & = & 8, & \\ & x_2 + y_2 & = & 9, & \quad x_2, y_1, y_2 \geq 0. \end{array}$$

This LP has the solution $x_2 = 4, y_1 = 0, y_2 = 5$ and $f_{opt} = 5 > 0$. A solution of the dual of the restricted primal is $\bar{\lambda} = (-1/2, 1)^\top$. We find $A^\top \bar{\lambda} = (3/2, 0, -1/2, 1)^\top$ and thus the unboundedness criterion is not satisfied. We proceed with computing

$$t_{max} = \min \left\{ \frac{-2-(-3)}{3/2}, \frac{0-(-1)}{1} \right\} = \frac{2}{3}$$

and update $\lambda = \lambda + t_{max}\bar{\lambda} = (-4/3, -1/3)^\top$.

Iteration 2:
$A^\top \lambda = (-2, -3, -4/3, -1/3)^\top$ *and thus* $B = \{1, 2\}$. *The restricted primal is*

$$\begin{array}{lrcll} \textit{Minimise} & y_1 + y_2 & & & \\ \textit{subject to} & x_1 + 2x_2 + y_1 & = & 8, & \\ & 2x_1 + x_2 + y_2 & = & 9, & \quad x_1, x_2, y_1, y_2 \geq 0. \end{array}$$

This LP has the solution $x_1 = 10/3, x_2 = 7/3, y_1 = y_2 = 0$ *and* $f_{opt} = 0$. *Hence,* $x = (10/3, 7/3, 0, 0)^\top$ *is optimal for (P) and* $\lambda = (-4/3, -1/3)^\top$ *is optimal for (D).*

Chapter 4

Integer Linear Programming

In this chapter we consider integer programs of the following type.

Definition 4.0.1 **(Integer linear program (ILP))**
Let $c = (c_1, \ldots, c_n)^\top \in \mathbb{R}^n$, $b = (b_1, \ldots, b_m)^\top \in \mathbb{R}^m$, *and*

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

be given. The integer linear program reads as follows: Find $x = (x_1, \ldots, x_n)^\top$ *such that the objective function*

$$c^\top x = \sum_{i=1}^{n} c_i x_i$$

becomes minimal subject to the constraints

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j &= b_i, & i &= 1, \ldots, m, \\ x_j &\geq 0, & j &= 1, \ldots, n, \\ x_j &\in \mathbb{Z}, & j &= 1, \ldots, n. \end{aligned}$$

In matrix notation:

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b, \quad x \geq 0, \quad x \in \mathbb{Z}^n. \tag{4.1}$$

Remark 4.0.2
In the case $x_i \in \{0, 1\}$, $i = 1, \ldots, n$, *the problem is called a binary optimisation problem. A more general class is given by so-called mixed-integer optimisation problems. Herein, some of the variables may assume real values while the others may assume integer values only. We don't want to discuss mixed-integer optimisation problems here although most of the following ideas for integer linear programs can be extended to mixed-integer problems as well.*

We have seen already some integer linear programs in the introduction, for instance the

assignment problem 1.1.7 and the knapsack problem 1.1.10 are integer linear programs (actually they are binary optimisation problems). Also the traveling salesman problem 1.1.9 is a integer linear program and a binary optimisation problem, respectively. The facility location problem 1.1.11 is an example for a mixed-integer optimisation problem.
Very frequently, we will make use of the following integer linear program relaxation, which is obtained by omitting the integer constraints $x_j \in \mathbb{Z}$, $j = 1, \ldots, n$.

Definition 4.0.3 **(Integer linear program relaxation (ILP relaxation))**
The linear program

$$\textit{Minimise} \quad c^\top x \quad \textit{subject to} \quad Ax = b, \quad x \geq 0$$

is called integer linear program relaxation (ILP relaxation).

In general, a relaxation of a given optimisation problem is obtained by omitting or simplifying constraints such that the feasible set of the relaxed problem becomes larger than the feasible set of the original problem but otherwise contains all feasible points of the original problem. In the ILP relaxation we omitted integer constraints admitting feasible points with real valued components. Nevertheless, all integer feasible points are still feasible for the ILP relaxation. Usually the relaxation of a problem is easier to solve than the original problem. For instance, the ILP relaxation is a standard linear program and can be solved by the simplex method while the ILP itself is hard to solve.
The ILP relaxation has an important property:

Theorem 4.0.4 **(Lower Bound Property)**
Let $\hat{x}$ be an optimal solution of the integer linear program and let $\hat{x}^{rel}$ be an optimal solution of the ILP relaxation. Then

$$c^\top \hat{x}^{rel} \leq c^\top \hat{x}.$$

Proof: The feasible set of the ILP relaxation contains the feasible set of ILP. As the feasible set of the ILP relaxation is larger than that of ILP, the optimal objective function value of the ILP relaxation has to be less than or equal to the optimal objective function of ILP. □

How can ILP be solved?

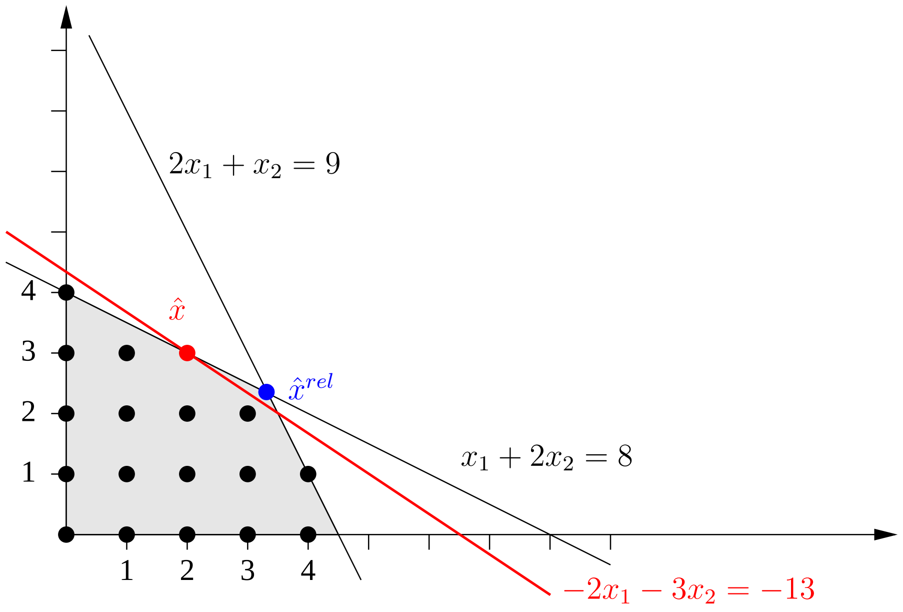
A very naive (and false) approach would be to solve the ILP relaxation and round the obtained solution $\hat{x}^{rel}$ componentwise to the next feasible integer solution. The following example shows that this rounding procedure fails in general.

Example 4.0.5 **(compare problem sheet 1, question 2)**
Consider the ILP:

Minimise $-2x_1 - 3x_2$ *subject to*

$$\begin{aligned} x_1 + 2x_2 &\leq 8, \\ 2x_1 + x_2 &\leq 9, \quad x_1, x_2 \geq 0,\ x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

The feasible set of ILP is given by the black points. The optimal ILP solution is given by the red point $\hat{x} = (2,3)^\top$ *with objective function value* -13. *The blue point* $\hat{x}^{rel} = (10/3, 7/3)^\top$ *with objective function value* $-13\frac{2}{3}$ *is the solution of the ILP relaxation, whose feasible set is the shaded area. Rounding the point* $\hat{x}^{rel}$ *to the next feasible integer solution yields the point* $x = (3,2)^\top$ *with objective function value* -12. *This point is not optimal for ILP.*



4.1 Gomory's Cutting Plane Algorithm

The cutting plane method of Gomory is based on the construction of 'cuts', which are additional linear constraints that 'cut-off' parts of the ILP relaxation until eventually an integer solution for ILP is being found. The algorithm works as follows:

Algorithm 4.1.1 **(Cutting plane algorithm of Gomory)**

(0 Let (R) denote the ILP relaxation 4.0.1.

(1) Solve (e.g. using simplex) the problem (R) and let $\hat{x}^{rel}$ *denote the optimal solution.*

If $\hat{x}^{rel}$ is integer, then $\hat{x}^{rel}$ is also optimal for ILP. STOP: optimal solution found!

If problem (R) does not have a solution, then ILP has no solution as well. STOP: no solution exists!

(2) *Construct an additional linear constraint (a 'cut') with the following properties:*

(i) *$\hat{x}^{rel}$ does* **not** *satisfy this additional constraint, that is, $\hat{x}^{rel}$ is being cut-off by the constraint.*

(ii) *Every feasible point of ILP satisfies the additional constraint.*

(3) *Add the constraint constructed in (2) to the problem (R) and go to (1).*

Of course, it has to be specified how the additional linear constraint in step (2) of the algorithm has to be constructed.

Construction of the cut in step (2):

Problem (R) in step (1) of the algorithm can be solved by the simplex method. If an optimal solution exists then the simplex method terminates in a feasible basic solution

$$\hat{x}_B^{rel} = A_B^{-1}b - A_B^{-1}A_N\hat{x}_N^{rel} = \beta - \Gamma\hat{x}_N^{rel},$$

with basis index set B and non-basis index set N. If $\hat{x}_B^{rel}$ is integer, then $\hat{x}_B^{rel} = \beta$, $\hat{x}_N^{rel} = 0$ is optimal for ILP and the algorithm terminates.

If $\hat{x}_B^{rel} = \beta \notin \mathbb{Z}^m$, then we have to construct a linear constraint which cuts-off $\hat{x}^{rel}$ but does not cut-off feasible points from ILP. We need the following ingredients:

- Let $i \in B$ be an arbitrary index with $\beta_i \notin \mathbb{Z}$.
- Let $\lfloor a \rfloor$ denote the largest integer which is less than or equal to a. Then $\lfloor a \rfloor \leq a$ holds.
- Let x be an arbitrary feasible point of ILP.

Using the same basis index set B which defines the feasible basic solution $\hat{x}^{rel}$, we can express x by

$$x_i = \beta_i - \sum_{j \in N} \gamma_{ij}x_j,$$

where β_i, $i \in B$, are the components of $\beta = A_B^{-1}b$ and γ_{ij}, $i \in B$, $j \in N$, are the components of $\Gamma = A_B^{-1}A_N$. Then:

$$\beta_i = x_i + \sum_{j \in N} \gamma_{ij}x_j. \tag{4.2}$$

Since $x_j \geq 0$ and $\lfloor \gamma_{ij} \rfloor \leq \gamma_{ij}$ we find

$$\beta_i \geq x_i + \sum_{j \in N} \lfloor \gamma_{ij} \rfloor x_j. \tag{4.3}$$

Since the components x_i, $i \in B$, and x_j, $j \in N$, of x are integer (recall that x is a feasible point of ILP), we obtain

$$x_i + \sum_{j \in N} \lfloor \gamma_{ij} \rfloor x_j \in \mathbb{Z}.$$

Hence, the feasible point x of ILP satisfies even the stronger inequality (compared to (4.3))

$$\lfloor \beta_i \rfloor \geq x_i + \sum_{j \in N} \lfloor \gamma_{ij} \rfloor x_j. \tag{4.4}$$

Owing to (4.2) and (4.4) the feasible point x of ILP satisfies the inequality

$$\lfloor \beta_i \rfloor - \beta_i \geq \sum_{j \in N} (\lfloor \gamma_{ij} \rfloor - \gamma_{ij}) x_j. \tag{4.5}$$

However, the current non-integer basic solution $\hat{x}_B^{rel} = \beta$ does not satisfy this inequality because $\beta_i \notin \mathbb{Z}$ and $\lfloor \beta_i \rfloor - \beta_i < 0$ and $\hat{x}_j^{rel} = 0$, $j \in N$. Hence, the additional constraint (4.5) cuts-off the point $\hat{x}^{rel}$ as desired.

Summarising, we can specify step (2) in Algorithm 4.1.1:

Step (2): Construction of a cut

Let $\hat{x}_B^{rel} = \beta$, $\hat{x}_N^{rel} = 0$ denote the optimal solution of problem (R) with basis index set B and non-basis index set N. Choose an index $i \in B$ with $\beta_i \notin \mathbb{Z}$. Add the constraint

$$\begin{aligned} x_{n+1} + \sum_{j \in N} (\lfloor \gamma_{ij} \rfloor - \gamma_{ij}) x_j &= \lfloor \beta_i \rfloor - \beta_i, \\ x_{n+1} &\geq 0 \end{aligned}$$

where x_{n+1} is a slack variable for (4.5).

Example 4.1.2 (compare Example 4.0.5)

Consider the following integer linear program (x_3, x_4 are slack variables):

Minimise $-2x_1 - 3x_2$ subject to

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 8, \\ 2x_1 + x_2 + x_4 &= 9, \quad x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 1,2,3,4. \end{aligned}$$

The solution of the ILP relaxation using the simplex method yields the following feasible basic solution with basis index set $B = \{1, 2\}$:

$$A_B = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \ b = \begin{pmatrix} 8 \\ 9 \end{pmatrix}, \ A_N = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \ \Gamma = A_B^{-1} A_N = \begin{pmatrix} -\frac{1}{3} & \frac{2}{3} \\ \frac{2}{3} & -\frac{1}{3} \end{pmatrix},$$

$$\begin{pmatrix} \hat{x}_1^{rel} \\ \hat{x}_2^{rel} \end{pmatrix} = \beta = A_B^{-1} b = \begin{pmatrix} \frac{10}{3} \\ \frac{7}{3} \end{pmatrix}, \qquad \left(\hat{x}_3^{rel}, \hat{x}_4^{rel}\right)^\top = (0, 0)^\top.$$

The component $\hat{x}_1^{rel} = \frac{10}{3}$ is not an integer, so we may use the index $1 \in B = \{1, 2\}$ for constructing the cut. The inequality (4.5) with $\beta_1 = 10/3$, $\gamma_{14} = 2/3$, $\gamma_{13} = -1/3$ reads as:

$$-\frac{1}{3} = \left\lfloor \frac{10}{3} \right\rfloor - \frac{10}{3} \geq \left(\left\lfloor -\frac{1}{3} \right\rfloor + \frac{1}{3} \right) x_3 + \left(\left\lfloor \frac{2}{3} \right\rfloor - \frac{2}{3} \right) x_4 = -\frac{2}{3} x_3 - \frac{2}{3} x_4.$$

Hence, the cut is given by the inequality

$$-\frac{1}{3} \geq -\frac{2}{3} x_3 - \frac{2}{3} x_4.$$

We now intend to visualise the line

$$-\frac{2}{3} x_3 - \frac{2}{3} x_4 = -\frac{1}{3} \tag{4.6}$$

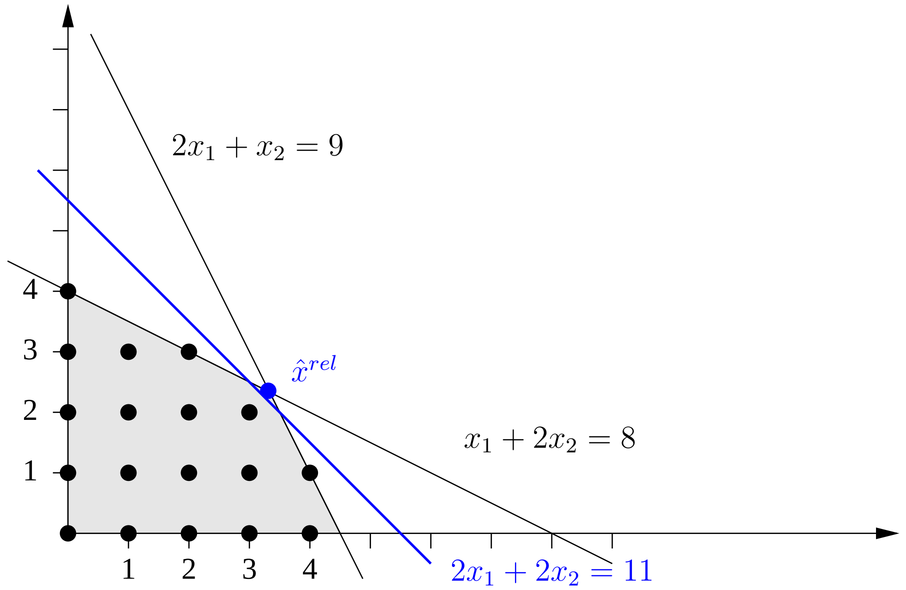
which defines the cut in the (x_1, x_2)-plane. We exploit the original constraints of ILP in order to obtain

$$\begin{aligned} x_3 &= 8 - x_1 - 2x_2, \\ x_4 &= 9 - 2x_1 - x_2. \end{aligned}$$

Introducing these relations into (4.6) yields the line

$$2x_1 + 2x_2 = 11.$$

This additional constraint (blue) actually cuts-off $\hat{x}^{rel}$:



In the next iteration of the cutting plane algorithm we have to solve the following extended linear program:

Minimise $-2x_1 - 3x_2$ subject to

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 8, \\ 2x_1 + x_2 + x_4 &= 9, \\ -\frac{2}{3}x_3 - \frac{2}{3}x_4 + x_5 &= -\frac{1}{3}, \quad x_i \geq 0,\ i = 1, \ldots, 5. \end{aligned}$$

The solution of this LP using the simplex method yields the following feasible basic solution with basis index set $B = \{1, 2, 4\}$:

$$A_B = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & -\frac{2}{3} \end{pmatrix}, \ b = \begin{pmatrix} 8 \\ 9 \\ -\frac{1}{3} \end{pmatrix}, \ A_N = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ -\frac{2}{3} & 1 \end{pmatrix}, \ \Gamma = \begin{pmatrix} -1 & 1 \\ 1 & -\frac{1}{2} \\ 1 & -\frac{3}{2} \end{pmatrix},$$

$$\begin{pmatrix} \hat{x}_1^{rel} \\ \hat{x}_2^{rel} \\ \hat{x}_4^{rel} \end{pmatrix} = \beta = A_B^{-1} b = \begin{pmatrix} 3 \\ \frac{5}{2} \\ \frac{1}{2} \end{pmatrix}, \qquad \left(\hat{x}_3^{rel}, \hat{x}_5^{rel}\right)^\top = (0,0)^\top.$$

The component $\hat{x}_2^{rel} = \frac{5}{2}$ is not an integer, so we may use the index $2 \in B = \{1, 2, 4\}$ for constructing a further cut. The inequality (4.5) with $\beta_2 = 5/2$, $\gamma_{25} = -1/2$, $\gamma_{23} = 1$ reads as

$$-\frac{1}{2} = \left\lfloor \frac{5}{2} \right\rfloor - \frac{5}{2} \geq \left(\left\lfloor -\frac{1}{2} \right\rfloor + \frac{1}{2} \right) x_5 + \left(\lfloor 1 \rfloor - 1 \right) x_3 = -\frac{1}{2} x_5$$

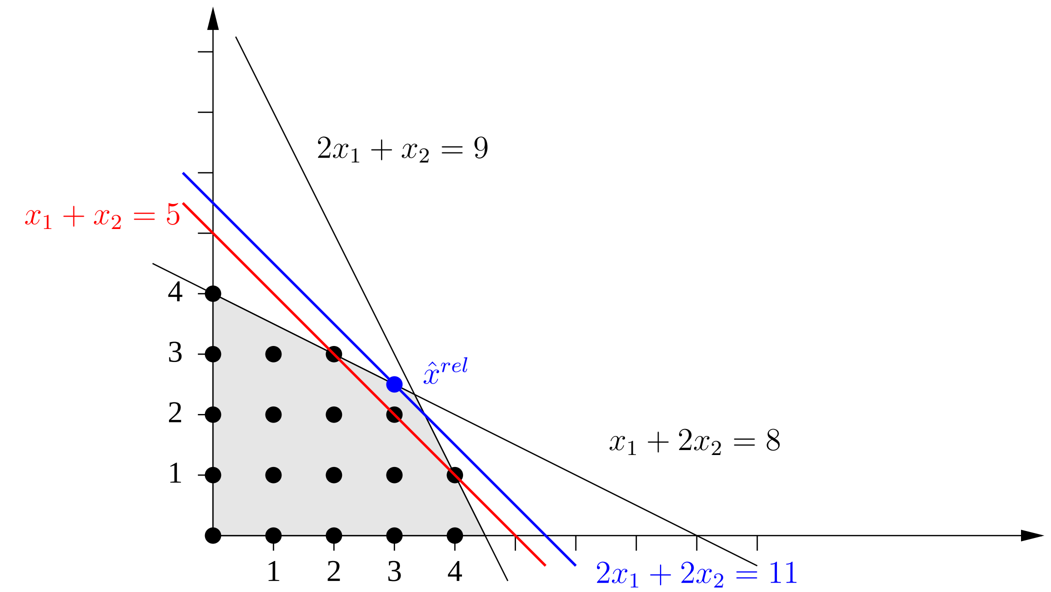
Hence, the cut is given by the inequality

$$-\frac{1}{2} \geq -\frac{1}{2} x_5.$$

Again we intend to visualise the line $-1/2 = -1/2 x_5$ in the (x_1, x_2)-plane. Exploiting the original constraints yields

$$x_1 + x_2 = 5.$$

This additional constraint (red) cuts-off the current point $\hat{x}^{rel}$ (blue point):



In the next iteration of the cutting plane algorithm we have to solve the following extended linear program:
Minimise $-2x_1 - 3x_2$ *subject to*

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 8, \\ 2x_1 + x_2 + x_4 &= 9, \\ -\frac{2}{3}x_3 - \frac{2}{3}x_4 + x_5 &= -\frac{1}{3}, \\ -\frac{1}{2}x_5 + x_6 &= -\frac{1}{2}, \quad x_i \geq 0,\ i = 1, \ldots, 6. \end{aligned}$$

It turns out that this problem has the following solution, which is feasible for ILP and thus optimal:

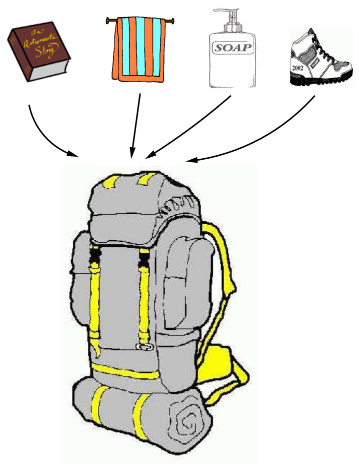
$$\hat{x}^{rel} = (2, 3, 0, 2, 1, 0)^\top \qquad (B = \{1, 2, 4, 5\}).$$

Remark 4.1.3

- *A disadvantage of Gomory's cutting plane method is the monotonically increasing dimension of the problem (R) in step (1) of the algorithm since in each iteration an additional constraint is being added.*
- *An efficient implementation of the method is based on the dual simplex method, which allows to re-use the optimal simplex tables constructed in step (1).*
- *Under some conditions, the finiteness of the cutting plane algorithm can be shown, see Gerdts and Lempio [GL06, Sec. 8.2, Th. 8.2.10].*

4.2 Branch&Bound

The Branch&Bound method is an enumeration algorithm which aims at breaking down a given difficult problem into possibly many problems that are easier to solve.
We explain the working principle of Branch&Bound for the knapsack problem with $N = 3$ items, compare Example 1.1.10.



There is one knapsack and 3 items. The items have the values $3, 5, 2$ and the weights $30, 50, 20$. The task is to create a knapsack with maximal value under the restriction that the weight is less than or equal to $A = 70$. This leads to the following optimisation problem.
Maximise $3x_1 + 5x_2 + 2x_3$ subject to the constraints

$$30x_1 + 50x_2 + 20x_3 \leq 70, \quad x_i \in \{0, 1\},\ i = 1, 2, 3.$$

Let

$$S = \{(x_1, x_2, x_3)^\top \mid 30x_1 + 50x_2 + 20x_3 \leq 70,\ x_i \in \{0, 1\},\ i = 1, 2, 3\}$$

denote the feasible set. We know that the optimal value for x_1 is either 0 or 1. Hence, we can split the knapsack problem into two smaller problems by fixing x_1 to 0 or 1, respectively. Hence we can partition the feasible set S as

$$S = S_0 \cup S_1, \qquad S_0 \cap S_1 = \emptyset,$$

where

$$\begin{aligned} S_0 &= \{(x_1, x_2, x_3)^\top \mid 30x_1 + 50x_2 + 20x_3 \leq 70,\ x_1 = 0,\ x_i \in \{0, 1\},\ i = 2, 3\} \\ S_1 &= \{(x_1, x_2, x_3)^\top \mid 30x_1 + 50x_2 + 20x_3 \leq 70,\ x_1 = 1,\ x_i \in \{0, 1\},\ i = 2, 3\}. \end{aligned}$$

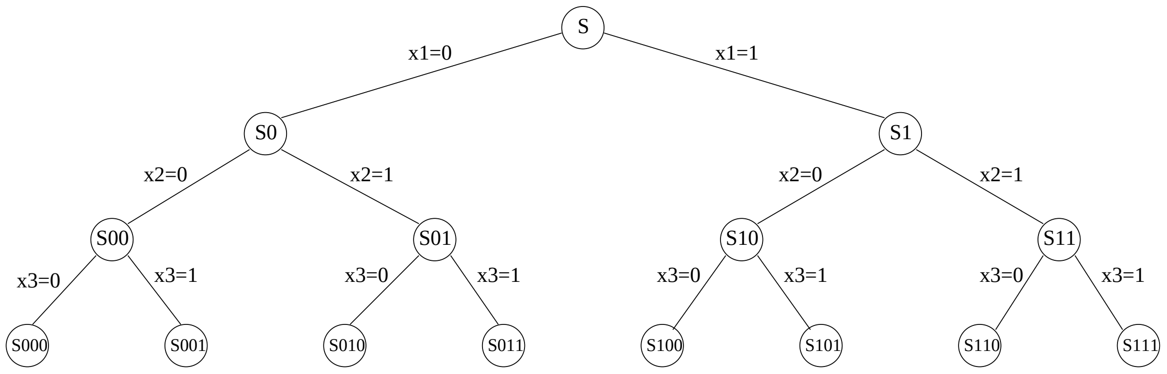
For each of the subsets S_0 and S_1 we can do the same with the component x_2:

$$S_0 = S_{00} \cup S_{01} \qquad \text{and} \qquad S_1 = S_{10} \cup S_{11},$$

where $S_{00} \cap S_{01} = \emptyset$ and $S_{10} \cap S_{11} = \emptyset$ and

$$\begin{aligned}
S_{00} &= \{(x_1, x_2, x_3)^\top \mid 30x_1 + 50x_2 + 20x_3 \leq 70,\ x_1 = x_2 = 0,\ x_3 \in \{0,1\}\} \\
S_{01} &= \{(x_1, x_2, x_3)^\top \mid 30x_1 + 50x_2 + 20x_3 \leq 70,\ x_1 = 0,\ x_2 = 1,\ x_3 \in \{0,1\}\} \\
S_{10} &= \{(x_1, x_2, x_3)^\top \mid 30x_1 + 50x_2 + 20x_3 \leq 70,\ x_1 = 1,\ x_2 = 0,\ x_3 \in \{0,1\}\} \\
S_{11} &= \{(x_1, x_2, x_3)^\top \mid 30x_1 + 50x_2 + 20x_3 \leq 70,\ x_1 = x_2 = 1,\ x_3 \in \{0,1\}\}.
\end{aligned}$$

Finally, we can repeat the procedure for the component x_3. This leads to the following tree structure:



Essentially we constructed a partition of the feasible set S according to

$$S = S_{000} \cup S_{001} \cup S_{010} \cup S_{011} \cup S_{100} \cup S_{101} \cup S_{110} \cup S_{111}.$$

Now, we can associate with each node the following optimisation problem:

$$\max \quad 3x_1 + 5x_2 + 2x_3 \quad \text{subject to} \quad (x_1, x_2, x_3)^\top \in S_*, \tag{4.7}$$

where $* \in \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111\}$. Notice that the feasible set S_* gets smaller as we descent in the tree and at the same time the corresponding optimal objective function values are non-decreasing.

Eventually, the optimisation problems corresponding to the nodes in the bottom row are very easy to solve, because for each of these 'leaf nodes' all components of the vector $(x_1, x_2, x_3)^\top$ are fixed. Hence, either the corresponding feasible set is empty or contains exactly one element. For instance $S_{101} = \{(1, 0, 1)^\top\}$ with objective function value 5, while $S_{111} = \emptyset$. By investigating all eight leafs of the tree we can easily find the optimal solution of the problem (find it!).

Unfortunately, the tree becomes extremely large for large values of N and this complete enumeration strategy (branching) will be computationally intractable. At this point, the bounding procedure comes into play. This bounding procedure aims at pruning those subtrees which don't contain a solution of the problem. What cases might occur in which it makes no sense to expand a subtree any further?

Case 1: Pruning by infeasibility

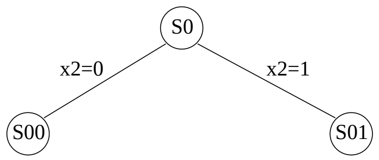
Consider the node S_{11} in the above knapsack problem. It holds $S_{11} = \emptyset$ because $x_1 = x_2 = 1$ and $30 + 50 = 80 \not\leq 70$. Hence, it is unnecessary to investigate the descendants S_{110} and S_{111} because these are infeasible as well. Hence, the whole subtree below the infeasible node S_{11} can be pruned.

Case 2: Pruning by optimality

If an optimisation problem associated with a node can be solved exactly, then the subtree emanating from this node needs not to be investigated and can be pruned.

Case 3: Pruning by bound

Consider the following subtree of the knapsack tree.



For the node S_{00} it is easy to see that the optimal objective function value is between 0 (if $x_3 = 0$) and 2 (if $x_3 = 1$). Likewise, the optimal objective function value of the node S_{01} is between 5 (if $x_3 = 0$) and 7 (if $x_3 = 1$). As we aim at maximising the objective function, it is clear that the optimum of the problem S is not going to be in the subtree emanating from S_{00} because the bounds for the node S_{01} guarantee a higher objective function value. Hence it is not necessary to expand node S_{00} any further.

The above pruning strategies help to reduce the number of nodes in the tree considerably. It remains to answer the question of how to find lower and upper bounds on the optimal objective function value at each node of the tree. The following approaches are useful:

- Any feasible point provides a lower bound on the optimal objective function in a maximisation problem.
- The ILP relaxation provides an upper bound on the optimal objective function in a maximisation problem. The ILP relaxation of the knapsack problem is given by the following LP:

  Maximise $3x_1 + 5x_2 + 2x_3$ subject to the constraints

$$30x_1 + 50x_2 + 20x_3 \leq 70, \quad 0 \leq x_i \leq 1, \ i = 1, 2, 3.$$

It is straight forward to define an S_* relaxation for each node S_*. Each of these relaxed problems can be solved by the simplex method.

- The dual of the ILP relaxation provides an upper bound on the optimal objective function value in a maximisation problem (weak duality!).

Summarising, a general Branch&Bound method has the following components:

- branching rule: rules for generating new nodes in the tree (how should the feasible set be partitioned?)
- bounding rule: procedure for finding upper and lower bounds of the optimal objective function value (relaxation, duality)
- traversing rule: defines the order in which the tree is traversed (e.g. depth-first search)
- fathoming rules: defines when a node does not need to be explored any further

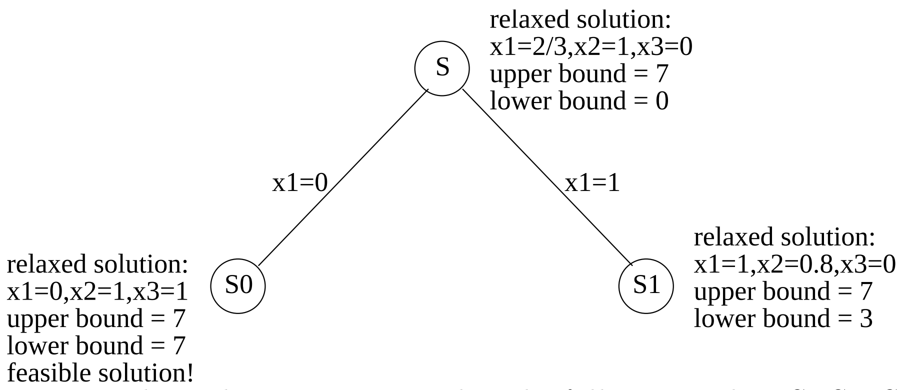
Example 4.2.1 **(Knapsack Branch&Bound)**

In the following example we apply the Branch&Bound method again to the knapsack problem:

Maximise $3x_1 + 5x_2 + 2x_3$ *subject to the constraints*

$$30x_1 + 50x_2 + 20x_3 \le 70, \quad x_i \in \{0,1\}, \; i = 1,2,3.$$

For each node an upper bound is computed by solving the ILP relaxation. Lower bounds are provided by feasible points. Branching is done on the component x_i^{rel} *with smallest index* i *for which* $x_i^{rel} \notin \{0,1\}$. *Herein,* x^{rel} *denotes the optimal solution of the ILP relaxation.*



In the above picture the nodes were generated in the following order: S, S_1, S_0. *As the* S_0 *relaxation produces a feasible solution with objective function value* 7, *it is not necessary to explore the subtree of* S_1, *because the value of* S_1 *is bounded between* 3 *and* 7, *so the subtree does not contain a solution which is better than that of* S_0.

4.2.1 Branch&Bound for ILP

We briefly discuss how Branch&Bound can be used to solve an integer linear program, see Definition 4.0.1.

Branching for ILP:

Consider a node of the Branch&Bound tree (for instance the root ILP). Let

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b,\ x \geq 0,\ x \in \mathbb{Z}^n.$$

be the corresponding integer linear program. Let x^{rel} be the solution of the ILP relaxation

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b,\ x \geq 0.$$

If x^{rel} is feasible for the original ILP then no further branching is necessary for the current node. If $x^{rel} \not\in \mathbb{Z}^n$ then branching consists of two steps:

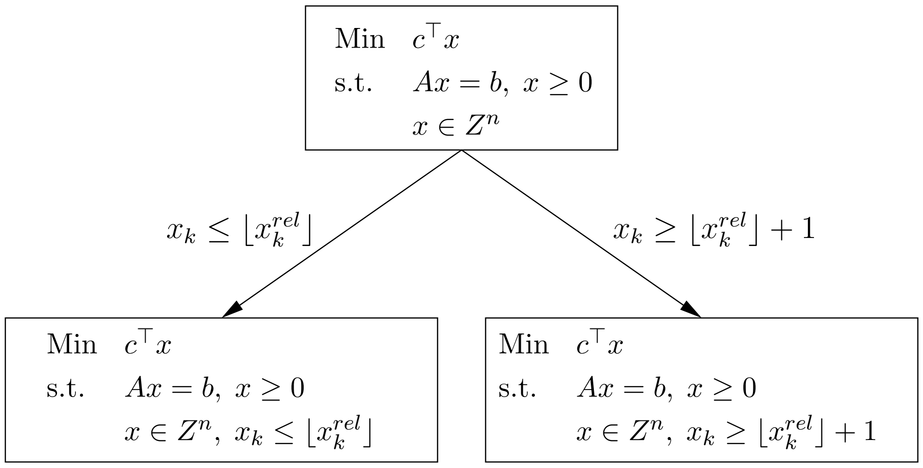
(i) Choose and index k with $x_k^{rel} \not\in \mathbb{Z}$. .

(ii) Create two successor nodes S_L (left successor) and S_R (right successor) according to

$$(S_L): \qquad \text{Minimise} \quad c^\top x \quad \text{s.t.} \quad Ax = b,\ x \geq 0,\ x \in \mathbb{Z}^n,\ x_k \leq \lfloor x_k^{rel} \rfloor.$$

and

$$(S_R): \qquad \text{Minimise} \quad c^\top x \quad \text{s.t.} \quad Ax = b,\ x \geq 0,\ x \in \mathbb{Z}^n,\ x_k \geq \lfloor x_k^{rel} \rfloor + 1.$$


Min $c^\top x$
s.t. $Ax = b,\ x \geq 0$
$x \in Z^n$
$x_k \leq \lfloor x_k^{rel} \rfloor$
$x_k \geq \lfloor x_k^{rel} \rfloor + 1$
Min $c^\top x$
s.t. $Ax = b,\ x \geq 0$
$x \in Z^n,\ x_k \leq \lfloor x_k^{rel} \rfloor$
Min $c^\top x$
s.t. $Ax = b,\ x \geq 0$
$x \in Z^n,\ x_k \geq \lfloor x_k^{rel} \rfloor + 1$

A recursive application of the branching rule starting with ILP creates a tree structure. Every edge corresponds to an additional linear constraint. Notice that the union of the feasible sets of (S_L) and (S_R) is again the feasible set of the predecessor node.

Bounding for ILP

Consider a node and the corresponding ILP

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b,\ x \geq 0,\ x \in \mathbb{Z}^n.$$

Let $\hat{x}$ denote the optimal solution with objective function value $\hat{f} = c^\top \hat{x}$. Of course, $\hat{f}$ is not known. A lower bound and an upper bound of $\hat{f}$ can be found as follows:

- **Upper bound:**
  For every x with $Ax = b$, $x \geq 0$, $x \in \mathbb{Z}^n$ it holds
  $$\hat{f} = c^\top \hat{x} \leq c^\top x.$$
  Hence, every feasible point x provides an upper bound $U = c^\top x$ of $\hat{f}$.

- **Lower bound:**
  Let x^{rel} denote the solution of the ILP relaxation
  $$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b,\ x \geq 0.$$
  Then
  $$f^{rel} := c^\top x^{rel} \leq c^\top \hat{x} = \hat{f}$$
  and f^{rel} is a lower bound for $\hat{f}$.

Fathoming rules for ILP

A node is called explored and no further branching is necessary for the node if one of the following holds:

- The ILP relaxation is infeasible.
- The ILP relaxation has an integer solution. This solution is feasible for the original ILP and the corresponding objective function value is an upper bound of the optimal objective function value.
- The lower bound of the node is larger than or equal to the current upper bound U.

Summarising we obtain the following algorithm.

Algorithm 4.2.2 (Branch&Bound algorithm)

```
Init:
    – Let $U$ be an upper bound of the optimal objective function value (if none is known,
      take $U = \infty$).
    – Let ILP be active.
while (there are active nodes) do
    Choose an active node $v$ according to a traversing rule.
    Compute (if possible) an optimal solution $\hat{x}$ of the $v$ relaxation
    and set $\hat{f} := c^\top \hat{x}$.
    if ($v$ infeasible) then label node $v$ as explored. endif
    if ($\hat{x} \in \mathbb{Z}^n$) then
        if ($\hat{f} < U$) then Save $\hat{x}$ as the current best solution and set $U = \hat{f}$. endif
        Label node $v$ as explored.
    endif
    if ($\hat{f} \geq U$) then Label node $v$ as explored. endif
    if ($\hat{f} < U$) then
        Apply branching rule, label all successors active and label
        current node $v$ inactive.
    endif
end
```

The size of the tree constructed by the algorithm depends on the data of the problem and can be very large. Often the algorithm has to be terminated without finding an optimal solution. However, an advantage of the method is it provides bounds of the optimal value even if it is terminated. The upper bound is given by U. A lower bound L is given by the minimum of all lower bounds of active nodes. U and L can be also used to define a stopping criterion, that is the algorithm stops if $U - L < \text{tol}$ holds for a given tolerance $\text{tol} > 0$.

The Branch&Bound method can be also used to solve traveling salesman problems, assignment problems, global optimisation problems, ... It's a very flexible algorithm and extremely important for practical applications in economy and industry. There exist many variations of the main algorithm, for instance:

- Lower bounds can be obtained by solving dual problems instead of ILP relaxations.
- Lower bounds obtained by ILP relaxations can be improved by performing a few cuts of the cutting plane method. For the performance of the Branch & Bound method it is important to compute good bounds which often lead to smaller search trees.

If the feasible set of ILP is unbounded, then the Branch & Bound method may not terminate as it can be observed for the following example

$$\max \quad x+y+z \quad \text{subject to} \quad 2x+2y \leq 1,\ z=0,\ x,y \in \mathbb{Z}.$$

Obviously, the example has infinitely many solutions satisfying $x_1 + x_2 = 0$.

4.3 Totally Unimodular Matrices

Let us consider again the Integer Linear Program 4.0.1, i.e.

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b, \quad x \geq 0, \quad x \in \mathbb{Z}^n,$$

and its relaxation

$$\text{Minimise} \quad c^\top x \quad \text{subject to} \quad Ax = b, \quad x \geq 0.$$

In general, the optimal solution of the ILP relaxation will not be feasible for ILP. However, there is a large class of problems for which it turns out that optimal basic solutions of the ILP relaxation are automatically feasible for ILP and hence solve ILP. Thus, for this problem class it is sufficient to solve just the ILP relaxation using the simplex method.

Definition 4.3.1 (totally unimodular matrix)

An integer matrix $A \in \mathbb{Z}^{m\times n}$ is called totally unimodular, if every square submatrix $\tilde{A}$ of A satisfies $\det(\tilde{A}) \in \{0, 1, -1\}$.

Note, that a totally unimodular matrix can only have entries $0, 1, -1$ as every element of A can be viewed as a square submatrix of dimension one.
Total unimodularity has an important connection to linear programming and especially to the feasible set of a linear program as the following theorem shows.

Theorem 4.3.2

Let $A \in \mathbb{Z}^{m\times n}$ be totally unimodular and let $b \in \mathbb{Z}^m$ be integral. Then all (feasible) basic solutions of the set

$$M_S := \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$$

are in $\mathbb{Z}^n$ (i.e. integral).

Proof: Let x be a basic solution of M_S with $A_B x_B = b$, $x_N = 0$ for a basis index set B and a non-basis index set N. As A_B is a square submatrix of A, it is totally unimodular. We show that all entries of $X := A_B^{-1}$ are integer. X satisfies $A_B X = I$ that is $A_B x^j = e^j$, $j = 1, \ldots, m$, where x^j denotes the j-th column of X and e^j the j-th unit vector. According to Cramer's rule it holds

$$X_{ij} = x_i^j = \frac{\det(A_B^i)}{\det(A_B)},$$

where A_B^i denotes the matrix A_B with the i-th column replaces by e^j. Now we can express the determinant of A_B^i by developing it with respect to the i-th column:

$$\det(A_B^i) = (-1)^{i+j}\det(A_B^{i,j}),$$

where $A_B^{i,j}$ denotes the submatrix of A_B with the i-th column and the j-th row deleted. As A has only integer entries, $\det(A_B^{i,j})$ is in $\mathbb{Z}$. As A_B is totally unimodular and non-singular, it holds $\det(A_B) = \pm 1$ and thus, $X_{ij} \in \mathbb{Z}$ for all i, j. Hence, $A_B^{-1} = X \in \mathbb{Z}^{m\times m}$ and thus $x_B = A_B^{-1} b \in \mathbb{Z}^m$ as b was assumed to be in $\mathbb{Z}^m$. □

This theorem has a very important consequence. Recall that the simplex method always terminates in an optimal vertex of the feasible set (assuming that an optimal solution exists). Hence, the simplex method, when applied to the ILP relaxation automatically terminates in an integer solution with $x \in \mathbb{Z}^n$ and consequently, this x is optimal for ILP as well.

Remark 4.3.3

Caution: The above does not imply that every optimal solution is integral! It just states that optimal vertex solutions are integral, but there may be non-vertex solutions as well which are not integral.

There are many network optimisation problems like transportation problems, shortest path problems or maximum flow problems that can be formulated with totally unimodular matrices. We investigate transportation problems, see Example 1.1.3, in more detail. For the transportation problem in Example 1.1.3 we assume that the supply $a_i \geq 0$, $i = 1, \ldots, m$, and demand $b_j \geq 0$, $j = 1, \ldots, n$, are balanced, that is

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j.$$

This is not a restriction as we could introduce an artificial consumer b_{n+1} whenever the total supply exceeds the total demand. On the other hand, if the total demand exceeded the total supply, then the problem would be infeasible.

Under this assumption, the transportation problem reads as follows and it can be shown that it always possesses a solution:

$$\begin{array}{lrcll}
\text{Minimise} & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} & & & \text{(minimise delivery costs)} \\
\text{s.t.} & \displaystyle\sum_{j=1}^{n} x_{ij} & = & a_i, & i=1,\ldots,m, \\
 & \displaystyle\sum_{i=1}^{m} x_{ij} & = & b_j, & j=1,\ldots,n, \\
 & x_{ij} & \geq & 0, & i=1,\ldots,m,\ j=1,\ldots,n.
\end{array}$$

In matrix notation we obtain:

$$\min \quad c^\top x \quad \text{s.t.} \quad Ax=d,\ x\geq 0,$$

where

$$\begin{array}{rcl}
x & = & (x_{11},\ldots,x_{1n},x_{21},\ldots,x_{2n},\ldots,x_{m1},\ldots,x_{mn})^\top, \\
c & = & (c_{11},\ldots,c_{1n},c_{21},\ldots,c_{2n},\ldots,c_{m1},\ldots,c_{mn})^\top, \\
d & = & (a_1,\ldots,a_m,b_1,\ldots,b_n)^\top, \\
A & = & \left(\begin{array}{cccc|cccc|c|cccc}
1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 1 & 1 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 1 & 1 & \cdots & 1 \\
\hline
1 & & & & 1 & & & & \cdots & 1 & & & \\
 & 1 & & & & 1 & & & \cdots & & 1 & & \\
 & & \ddots & & & & \ddots & & & & & \ddots & \\
 & & & 1 & & & & 1 & \cdots & & & & 1
\end{array}\right) \in \mathbb{R}^{(m+n)\times mn}.
\end{array}$$

In practical problems x_{ij} is often restricted in addition by the constraint $x_{ij} \in \mathbb{N}_0 = \{0,1,2,\ldots\}$, i.e. x_{ij} may only assume integer values. However, it turns out, that the transport matrix A is totally unimodular.

Corollary 4.3.4

The transport matrix A is totally unimodular.

Proof: Proof by induction on the size t of the $t\times t$ sub-matrices.
$t=1$: Let B a 1×1 sub-matrix of A. As A has only entries 0 and 1, the determinant of each 1×1 submatrix has value 0 or 1.
Induction step: Let the assumption hold for $t-1$ and let B be a $t\times t$ sub-matrix of A. Three cases may occur:

1. There is a zero column in B. Then $\det(B) = 0$.

2. Every column of B has exactly two entries 1. One entry refers to the first block row in A and one to the second. Summing up the rows corresponding to the first block row yields a vector of all ones. Summing up the rows corresponding to the second block row yields a vector of all ones again. Hence, the rows of the matrix B are linearly independent and thus $\det(B) = 0$.

3. B has a column with exactly one entry 1. Then, $\det(B) = \pm 1 \cdot \det(\tilde{B})$, where $\tilde{B}$ is the $(t-1) \times (t-1)$ sub-matrix of B that is obtained by deleting in B the respective column and the row with the 1 entry. According to the induction assumption, $\det(\tilde{B}) \in \{0, 1, -1\}$. Hence, $\det(B) \in \{0, 1, -1\}$.

$\square$

The total unimodularity of A and Theorem 4.3.2 guarantee that the simplex method, when applied to a transportation problem with $d \in \mathbb{Z}^{m+n}$, always computes a solution $x \in \mathbb{Z}^{mn}$.

Chapter 5

Minimum Spanning Trees

Many combinatorial optimisation problems like routing problems, network flow problems, traveling salesman problems, and transportation problems use graphs and networks. For instance, a roadmap is a graph which defines connections between cities.
This chapter will be used to

- introduce basic notations and terminologies from graph theory
- discuss minimum spanning tree problems
- introduce a greedy strategy for minimum spanning tree problems

As a motivation for this chapter, consider the following practically relevant problem.
A telephone company wants to rent a subset of an existing telephone network. The subset of the network should be large enough to connect all cities, but at the same time renting it should be as cheap as possible. Figure 5.1 illustrates the situation with 6 cities.

How to model this problem? The telephone network is naturally modeled by a graph or, more precisely, by a network. Intuitively, we think of a graph as a set of nodes and edges, which describe connections between nodes in terms of undirected lines or directed lines (arrows), compare Figure 5.1.

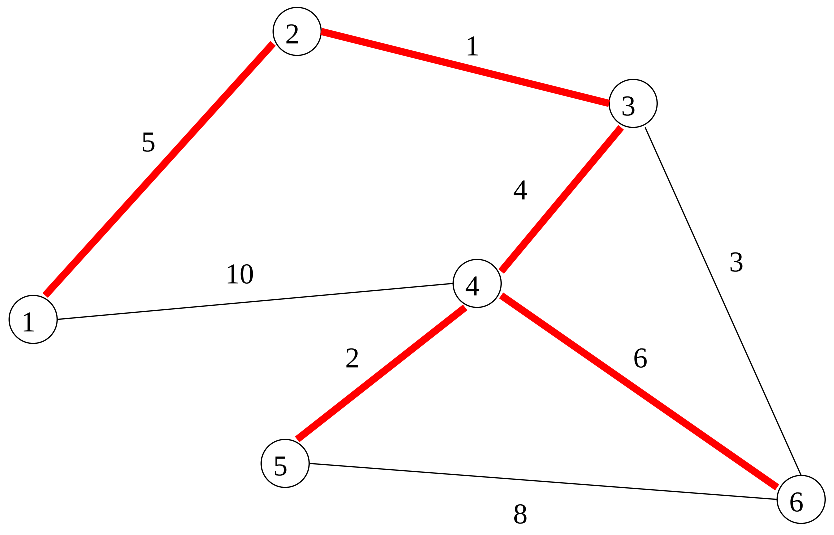


Figure 5.1: Spanning tree in a (undirected) graph.

In our example the nodes of the graph correspond to the cities and the edges correspond to the telephone connections between cities. The weight of each edge corresponds to the cost of renting the connection. We are looking for a so-called spanning tree of minimum weight which will be defined below.
Finding a spanning tree of minimum weight in a network belongs to the oldest problems in combinatorial optimisation and a first algorithm was given by Boruvka in 1926.

5.1 Elements from Graph Theory

Many combinatorial optimisation problems like the spanning tree problem, routing problems, network flow problems, traveling salesman problems, and transportation problems use graphs and networks. For instance, a roadmap or an electricity network is a graph which defines connections between cities.
Sometimes, the edges of a graph are just used to indicate that some kind of relation between two nodes exists while the orientation of the edge is not important. But often, the edges of a graph describe flow directions or one-way street connections. In this case, the orientation of an edge is actually important as it is only possible to reach some node from another node following the connecting edge in the right direction. Hence, it is useful and necessary to distinguish directed and undirected graphs.

Definition 5.1.1 (graph, directed graph, undirected graph, underlying undirected graph)

- *A graph G is a tupel $G = (V, E)$, where $V \neq \emptyset$ is a finite set of nodes (or vertices) and $E \subseteq V \times V$ are the edges of G.*

- *A graph (V, E) is called undirected, if all edges $e \in E$ are undirected, i.e. all tupels $(v, w) \in E$ are not ordered.*

- *A graph (V, E) is called digraph, if all edges $e \in E$ are directed, i.e. all tupels $(v, w) \in E$ are ordered.*

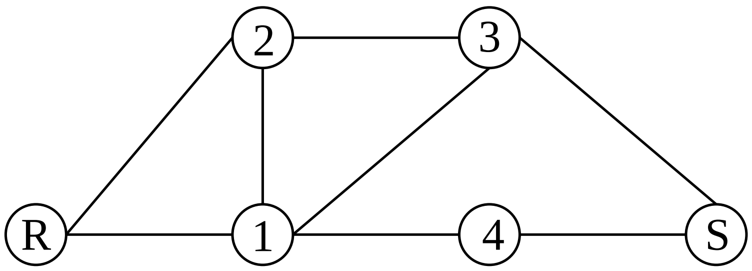
- *The underlying undirected graph of a digraph (V, E) is the undirected graph $(V, \tilde{E})$ with $(v, w) \in E \Leftrightarrow (v, w) \in \tilde{E}$, i.e. the underlying undirected graph has the same nodes and edges as the directed graph but the orientation of the edges are omitted.*

In the sequel we always assume that the number of nodes in a graph is finite. Moreover, we exclude parallel edges, i.e. edges having the same endpoints (undirected graphs) or the same tail and head (directed graphs), respectively. Finally, we exclude sweeps, i.e. edges of type (i, i) with $i \in V$.

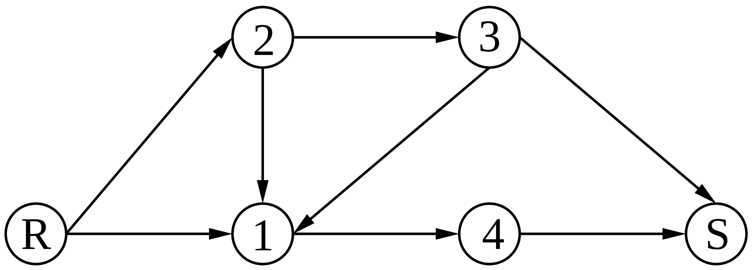
Usually, nodes are visualised by circles. Edges in an undirected graph are visualised by lines connecting two nodes. Edges in a digraph are visualised by arrows and indicate, e.g., flow directions or one-way street connections.

Example 5.1.2

Undirected graph $G = (V,E)$ *with nodes* $V = \{R,1,2,3,4,S\}$ *and edges* $E = \{(R,1),(R,2),(1,2),(1,3),(1,4),(2,3),(3,S),(4,S)\}$*:*



Directed graph $G = (V,E)$ *with nodes* $V = \{R,1,2,3,4,S\}$ *and edges* $E = \{(R,1),(R,2),(2,1),(1,4),(2,3),(3,1),(3,S),(4,S)\}$*:*



Caution: Let i and j be nodes and let $e = (i,j)$ be an edge. In an undirected graph the edges (i,j) and (j,i) are considered to be the same. Whereas in a digraph the ordering of the nodes i and j in the edge (i,j) is highly relevant, i.e. (i,j) and (j,i) denote different edges and only one of which may be present in the directed graph.

Often it is important to find a way through a given graph starting at some node and ending at another node.

Definition 5.1.3 (Walks and Paths)

Let $G = (V,E)$ *be a graph.*

- *Each sequence*

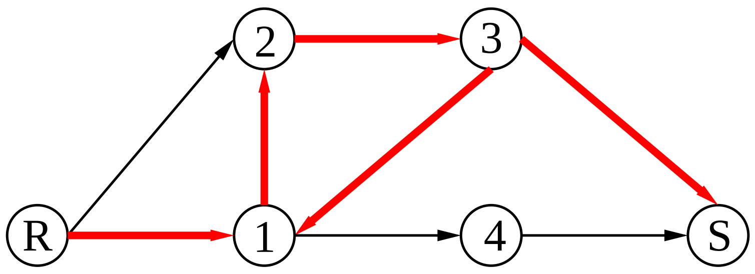
$$W = v_1, e_1, v_2, e_2, \dots, v_k, e_k, v_{k+1}, \qquad k \geq 0, \tag{5.1}$$

of nodes $v_1,\dots,v_{k+1} \in V$ *and edges* $e_1,\dots,e_k \in E$ *with* $e_i = (v_i, v_{i+1})$ *for* $i = 1,\dots,k$ *is called edge progression (in G) with initial node* v_1 *and final node* v_{k+1}.

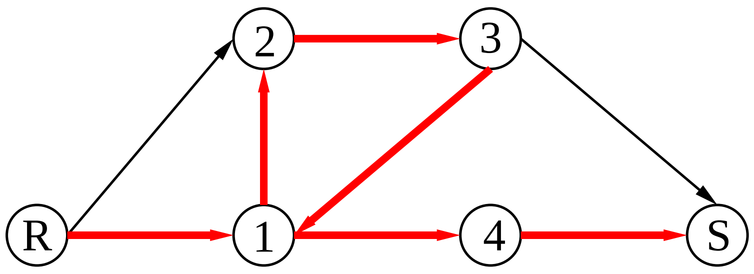
We use the shorthand notation $W = [v_1, v_2, \dots, v_k, v_{k+1}]$.

- *An edge progression (5.1) is called walk (in G), if $e_i \neq e_j$ for $1 \leq i < j \leq k$. A walk is closed if $v_1 = v_{k+1}$.*
- *A walk (5.1) is called $v_1 - v_{k+1}-$path (in G), if $v_i \neq v_j$ for $1 \leq i < j \leq k+1$.*
- *A circuit is a closed walk (5.1) with $v_i \neq v_j$ for $1 \leq i < j \leq k$.*
- *A node w is said to be reachable from a node v, if a $v - w-$path exists.*

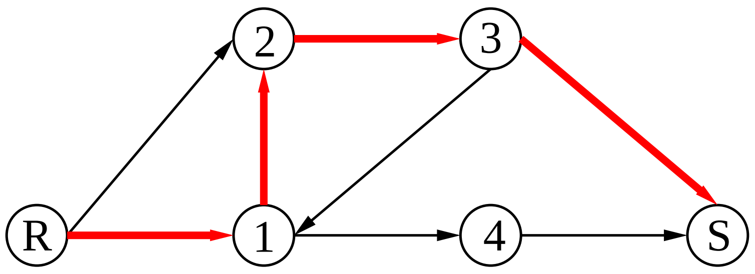
Example 5.1.4



Edge progression W=R,(R,1),1,(1,2),2,(2,3),3,(3,1),1,(1,2),2,(2,3),3,(3,S),S
not a walk as the edges (1,2) and (2,3) are visited twice!



Walk W=R,(R,1),1,(1,2),2,(2,3),3,(3,1),1,(1,4),4,(4,S),S
not a path as vertex 1 is visited twice!
The walk contains a circuit: 1,(1,2),2,(2,3),3,(3,1),1



Path W=R,(R,1),1,(1,2),2,(2,3),3,(3,S),S
S is reachable from every other vertex,
vertices R,1,2,3 are not reachable from vertex 4

Often, we are interested in connected graphs.

Definition 5.1.5 **(connected graphs)**

An undirected graph G is called connected, if there is a $v-w-$path for every $v, w \in V$. A digraph G is called connected, if its underlying undirected graph is connected.

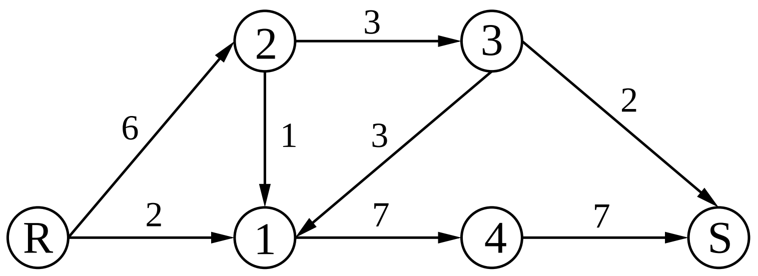
Often, edges in graphs are weighted, for instance moving along an edge causes costs or an edge corresponds to the length of a street or an edge has a certain capacity.

Definition 5.1.6 **(Weighted Graph, Network)**

A weighted graph or network N is a tripel $N = (V, E, c)$, where (V, E) is a graph and $c : E \to \mathbb{R}$ is a weight function with $e = (i, j) \in E \mapsto c(e) = c_{ij}$. The network is called directed (undirected), if the underlying graph (V, E) is directed (undirected).

Example 5.1.7

Network $N = (V, E, c)$ with nodes $V = \{R, 1, 2, 3, 4, S\}$, edges $E = \{(R, 1), (R, 2), (2, 1), (1, 4), (2, 3), (3, 1), (3, S), (4, S)\}$, and mapping $c : E \to \mathbb{R}$ with $c(R, 1) = c_{R1} = 2$, $c(R, 2) = c_{R2} = 6$, $c(2, 1) = c_{21} = 1$, $c(1, 4) = c_{14} = 7$, $c(2, 3) = c_{23} = 3$, $c(3, 1) = c_{31} = 3$, $c(3, S) = c_{3S} = 2$, $c(4, S) = c_{4S} = 7$.



5.2 Minimum Spanning Tree Problem

Before we formulate the problem in a formal way we need to clarify what a tree and a spanning tree are.

Definition 5.2.1 **(Tree, spanning tree)**

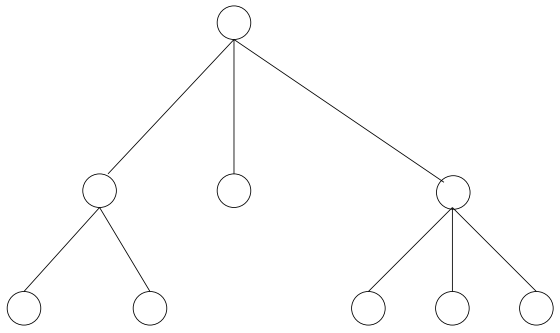
- *$G' = (V', E')$ is a subgraph of the graph $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$. A subgraph $G' = (V', E')$ of G is called spanning, if $V' = V$.*

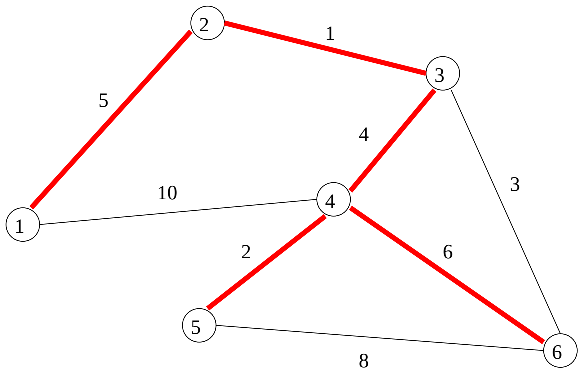
- *An undirected graph which is connected and does not have a circuit (as a subgraph) is called a tree.*

- *A spanning tree of an undirected graph is a spanning subgraph which is a tree.*

Example 5.2.2

A tree and a spanning tree (red) in a graph:





Theorem 5.2.3

Let G be an undirected graph with n nodes. Then the following statements are equivalent:

(a) G is a tree.

(b) G has $n-1$ edges and no circuits.

(c) G has $n-1$ edges and is connected.

(d) G is a maximal circuit-free graph, i.e. any additional edge creates a circuit.

(e) For any two distinct nodes v, w there exists a unique $v-w-$path.

With these definitions we can state the

Minimum Spanning Tree Problem:
Let an undirected graph $G = (V, E)$ with nodes $V = \{v_1, \dots, v_n\}$ and edges $E = \{e_1, \dots, e_m\}$, and a weight function $c : E \to \mathbb{R}$ be given. The task is either to find a spanning tree $T = (V, E')$ in G with $E' \subseteq E$ such that $\sum_{e \in E'} c(e)$ becomes minimal or to decide that G is not connected.

Note that a graph has at least one spanning tree, if and only if it is connected.

5.3 Kruskal's Algorithm: A Greedy Algorithm

How can we solve the problem? We know already that the result will be a spanning tree (provided such a tree exists) and thus by definition the nodes of the spanning tree are the nodes of G. So, all we have to do is to find the subset of edges $E' \subset E$ which defines the spanning tree and minimises the costs

$$\sum_{e \in E'} c(e).$$

How can the set of edges E' be constructed? A straightforward idea is to construct E' step by step by adding successively edges from E to E'. Let's do this in a greedy fashion: as a spanning tree with minimum weight is sought, in each step we prefer edges $(v_i, v_j) \in E$ with small costs $c(v_i, v_j) = c_{ij}$. Of course, we have to check whether adding an edge creates a circuit. This outline of an algorithm is essentially Kruskal's Algorithm.

Algorithm 5.3.1 **(Kruskal's Algorithm)**

(0) Let an undirected Graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $|E| = m$ and a weight function $c : E \longrightarrow \mathbb{R}$ be given.

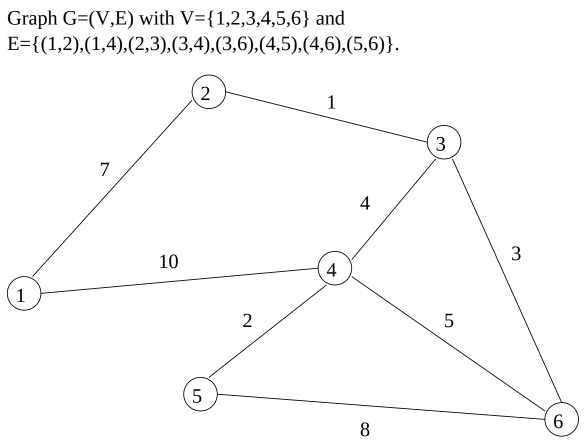
(1) Sort the edges in E such that $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$, where $e_i \in E$, $i = 1, \ldots, m$, denote the edges after sorting.

(2) Set $E' = \emptyset$ and $T = (V, E')$.

```
(3) For $i = 1$ to $m$ do
        If $(V, E' \cup \{e_i\})$ contains no circuit then set $E' := E' \cup \{e_i\}$ and $T := (V, E')$.
```

Example 5.3.2

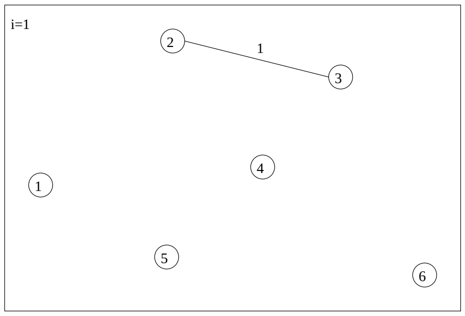
Consider the following example:


Graph G=(V,E) with V={1,2,3,4,5,6} and
E={(1,2),(1,4),(2,3),(3,4),(3,6),(4,5),(4,6),(5,6)}.
1
2
3
4
5
6
1
7
4
3
10
2
5
8

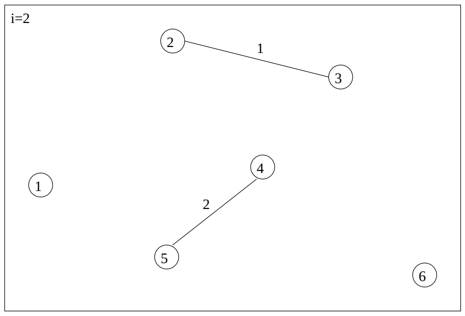
Sorting the edges in step (1) leads to the ordering

$$e_1 = (2,3), e_2 = (4,5), e_3 = (3,6), e_4 = (3,4), e_5 = (4,6), e_6 = (1,2), e_7 = (5,6), e_8 = (1,4).$$

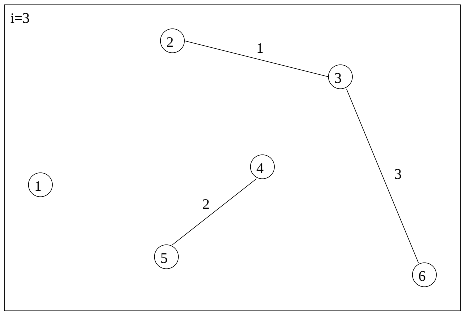
Kruskal's algorithm produces the following subgraphs:



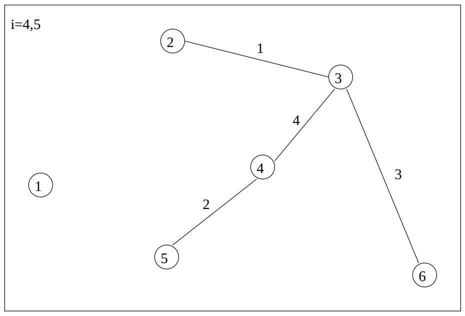
Add edge $e_1 = (2,3)$ *with weight* $c(e_1) = 1$.



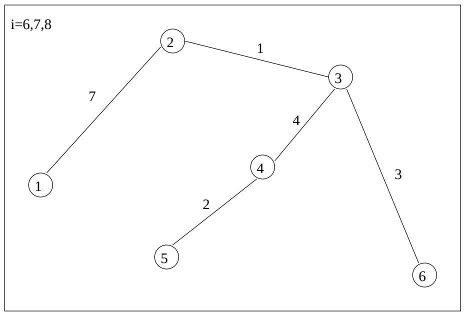
Add edge $e_2 = (4,5)$ *with weight* $c(e_2) = 2$.



Add edge $e_3 = (3,6)$ *with weight* $c(e_3) = 3$.



Add edge $e_4 = (3,4)$ *with weight* $c(e_4) = 4$. *Adding edge* $e_5 = (4,6)$ *with weight* $c(e_5) = 5$ *would lead to a cycle, so* e_5 *is not added.*



Add edge $e_6 = (1,2)$ *with weight* $c(e_6) = 7$. *Adding edges* $e_7 = (5,6)$ *with weight* $c(e_7) = 8$ *and* $e_8 = (1,4)$ *with weight* $c(e_8) = 10$ *would lead to cycles, so* e_7 *and* e_8 *are not added.*

Kruskal's algorithm is a so-called greedy algorithm. Greedy algorithms exist for many other applications. All greedy algorithms have in common that they use locally optimal decisions in each step of the algorithm in order to construct a hopefully globally optimal solution. With other words: Greedy algorithms choose the best possible current decision in each step of an algorithm. In Kruskal's algorithm in each step the edge with the smallest cost was used. In general, it is not true that this strategy leads to an optimal solution. But for the minimum spanning tree problem it turns out to work successfully. The correctness of Kruskal's algorithm is based on the following result.

Corollary 5.3.3

Let $G = (V, E)$ be an undirected graph with set of nodes $V = \{1, \ldots, n\}$ and set of edges E. Let $c : E \longrightarrow \mathbb{R}$, $(i, j) \in E \mapsto c(i, j) = c_{ij}$, be a weight function. Let $(V_1, E_1), (V_2, E_2), \ldots, (V_k, E_k)$ be sub-trees in G with pairwise disjoint node sets V_j, $j = 1, \ldots, k$, and with $V = \bigcup_{j=1}^{k} V_j$. Moreover, let $(r, s) \in E$ be an edge with

$$c_{rs} = \min\{c_{ij} \mid i \in V_1,\ j \notin V_1\}.$$

Then, among all spanning trees in G which contain all edges in $\bigcup_{i=1}^{k} E_i$ there exists a minimum spanning tree which contains the edge (r, s).

Proof: Assume the contrary: Every minimum spanning tree T in G which contains all edges in $\bigcup_{i=1}^{k} E_i$ does not contain the edge $(r, s) \in E$.
Let $T = (V, E')$ be such a minimum spanning tree. Without loss of generality we may assume that $r \in V_1$ and $s \in V_2$. T is connected because it is a tree. Moreover, as the V_j's are pairwise disjoint, there exists an edge $(u, v) \in E'$ with $u \in V_1$ and $v \in V_2$. According to the definition of c_{rs} it holds $c_{rs} \leq c_{uv}$. Now, if we replace in the tree T the edge (u, v) by (r, s) we get again a tree whose sum of weights is less than or equal to that of T. If $c_{rs} < c_{uv}$ then the sum of weights of the new tree is less than that of T which contradicts the minimality of the spanning tree T. Hence, $c_{rs} = c_{uv}$ and the new tree contains (r, s) and is minimal as well which contradicts the assumption. □

Theorem 5.3.4

Kruskal's algorithm constructs a spanning tree with minimal costs for the graph G, provided G has a spanning tree.

Proof: Let G be connected, which implies that $m = |E| \geq |V| - 1 = n - 1$. We need to show that T is a tree. By construction, $T = (V, E')$ is spanning and it contains no circuits. We will show that T is connected. Suppose, T is not connected. Then T can be decomposed into disconnected subtrees $T_i = (V_i, E_i)$, $i = 1, \ldots, k$, $k \geq 2$, of T with $E' = \cup_{i=1,\ldots,k} E_i$, $V = \cup_{i=1,\ldots,k} V_i$ and $|E'| = \sum_{i=1}^{k} |E_i| = \sum_{i=1}^{k} (|V_i| - 1) = n - k \leq n - 2$. Let $e \in E \setminus E'$ denote the edge with minimal cost $c(e)$ among all edges not in E'. Adding e to E' will not create a circuit. Hence, e would have been added by the algorithm, which contradicts the assumption.

It remains to show that T is a minimum spanning tree. In each iteration i of Kruskal's algorithm, T can be decomposed into disconnected subtrees $T_j^i = (V_j^i, E_j^i)$, $j = 1, \ldots, k_i$. According to Corollary 5.3.3 each of these graphs can be extended to a minimum spanning tree (not necessarily the same as the one that is constructed by the algorithm). Eventually, the algorithm will create the spanning tree T and application of Corollary 5.3.3 (with a trivial extension of zero edges) proves the assertion. □

The complexity of Kruskal's algorithm depends on the complexity of the sorting algorithm used in step (1) and the complexity of the circuit checking algorithm in step (3) of Kruskal's algorithm.
A well-known sorting algorithm is the Merge-Sort algorithm. It divides the list $a_1, a_2, \ldots, a_n$ of n real numbers to be sorted into two sublists of approximately the same size. These sublists are sorted recursively by the same algorithm. Finally, the sorted sublists are merged together. This strategy is known as a divide-and-conquer strategy.
The MERGE-SORT algorithm below computes a permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that

$$a_{\pi(1)} \le a_{\pi(2)} \le \ldots \le a_{\pi(n)}.$$

It can be shown that the Merge-Sort algorithm needs $\mathcal{O}(n \log n)$ operations, compare Korte and Vygen [KV08], Theorem 1.5, page 10.

Algorithm 5.3.5 (MERGE-SORT)

```
(0) Let a list $a_1, \ldots, a_n$ of real numbers be given.

(1) If $n = 1$ then set $\pi(1) = 1$ and STOP.

(2) Set $m = \lfloor \frac{n}{2} \rfloor$. Let $\rho = $ MERGE-SORT$(a_1, \ldots, a_m)$ and $\sigma = $ MERGE-SORT$(a_{m+1}, \ldots, a_n)$.

(3) Set $k = 1$ and $\ell = 1$.

    While $k \le m$ and $\ell \le n - m$ do
        If $a_{\rho(k)} \le a_{m+\sigma(\ell)}$ then
            set $\pi(k + \ell - 1) = \rho(k)$ and $k = k + 1$.
        else
            set $\pi(k + \ell - 1) = m + \sigma(\ell)$ and $\ell = \ell + 1$.
        endif
    end do
    While $k \le m$ do
        set $\pi(k + \ell - 1) = \rho(k)$ and $k = k + 1$.
    end do
    While $\ell \le n - m$ do
```

```
        set $\pi(k+\ell-1) = m+\sigma(\ell)$ and $\ell = \ell+1$.
    end do
```

Detecting a cycle in a graph can be done by the following labeling algorithm. Herein, every node initially gets assigned the label 'not explored'. Using a recursive depth first search, those nodes which are adjacent to the current node and are labeled 'not explored', are labeled 'active'. This is done until no adjacent node exists anymore or in the function CircDetectRec a node labeled 'active' is adjacent to the current 'active' node. In the latter case a circuit has been detected. The second argument u in CircDetectRec denotes the predecessor of v w.r.t. the calling sequence of CircDetectRec.

Algorithm 5.3.6 (CircDetect)

```
(0) Let a graph $G = (V,E)$ with $n$ nodes be given.
(1) Label every node $v \in V$ to be 'not explored'.
(2) For each $v \in V$ do
        If $v$ is 'not explored' then
            If CircDetectRec(v,v)=false then STOP: Graph contains cycle.
    end do
```

Function: result=CircDetectRec(v,u):

```
(0) Label node $v$ as 'active'.
(1) For each $w \in V$ adjacent to $v$ and $w \neq u$ do
        If $w$ is 'active' then
            result=false
            return
        else if $w$ is 'not explored' then
            If CircDetectRec(w,v)=false then
                result=false
                return
            end if
        end if
    end do
    Label node $v$ to be 'explored', set result=true, and return.
```

The complexity of the CircDetect algorithm for a graph with n nodes and m edges is $\mathcal{O}(n+m)$. As every graph $(V, E' \cup \{e_i\})$ in step (3) of Kruskal's algorithms has at most

n edges (finally it will be a spanning tree which has $n-1$ edges), the complexity of the circuit detection algorithm is given by $\mathcal{O}(n)$. Step (3) is executed m times and together with the complexity $\mathcal{O}(m \log m)$ of the MERGE-SORT algorithm we obtain

Corollary 5.3.7 (Complexity of Kruskal's algorithm)
Kruskal's algorithm can be implemented in $\mathcal{O}(m\log(m)+mn)$ operations and Kruskal's algorithm has polynomial complexity. The minimum spanning tree problem belongs to the class $\mathcal{P}$ of polynomially time solvable problems.

Remark 5.3.8
There are even more efficient implementations of Kruskal's algorithm which use better data structures and need $\mathcal{O}(m \log n)$ operations only. Details can be found in Korte and Vygen [KV08], page 130, and in Papadimitriou and Steiglitz [PS98], section 12.2, page 274.

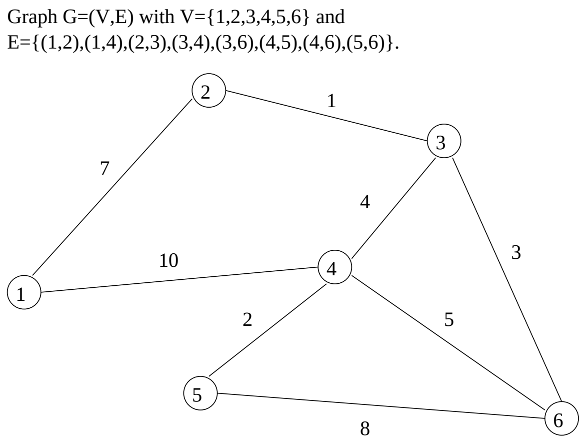
5.4 Prim's Algorithm: Another Greedy Algorithm

Kruskal's algorithm added successively edges to a graph with node set V. Prim's algorithm starts with a single node and constructs successively edges and adjacent nodes. This leads to a growing tree. Again, the greedy idea is employed as the edge with smallest cost is chosen in each step.

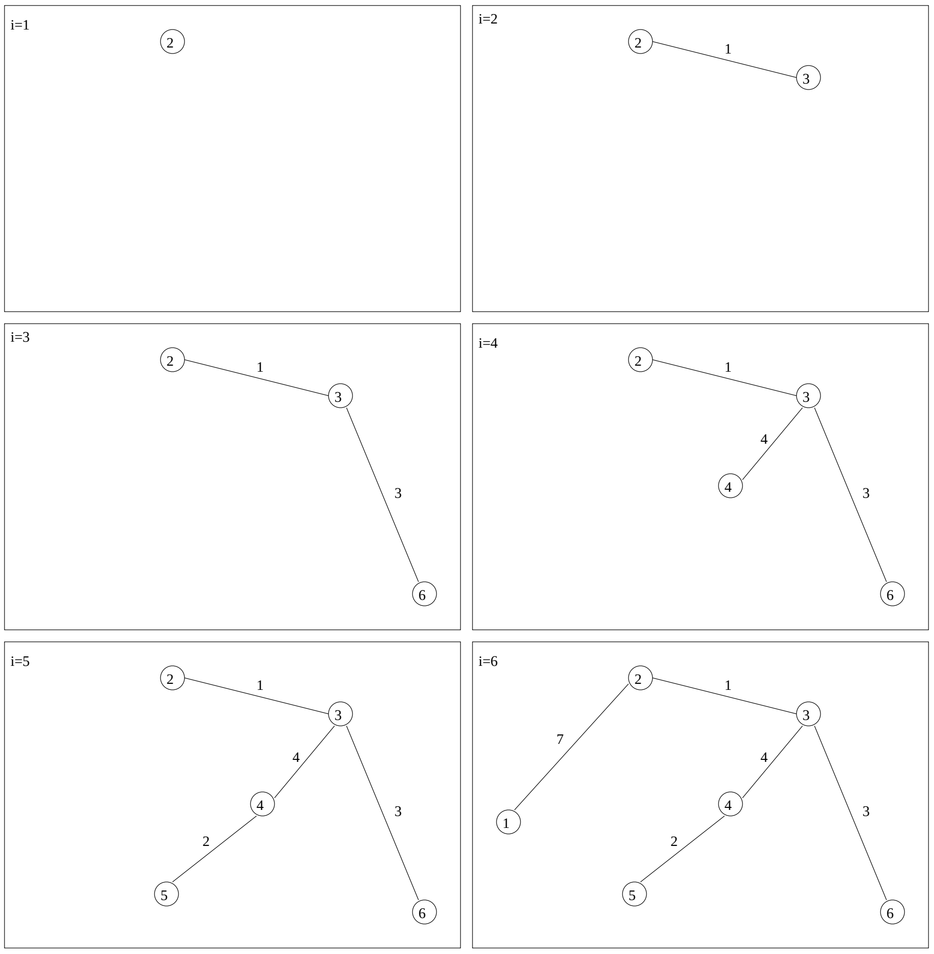
Algorithm 5.4.1 (Prim's Algorithm)

```
(0) Let an undirected Graph $G=(V,E)$ with $V=\{v_1,\ldots,v_n\}$ and $E=\{e_1,\ldots,e_m\}$
    and a weight function $c: E \to \mathbb{R}$ be given.

(1) Choose $v \in V$. Set $V' := \{v\}$, $E' := \emptyset$, and $T := (V', E')$.

(2) While $V' \neq V$ do
        Choose an edge $e=(v,w)$ with $v \in V'$ and $w \in V \setminus V'$ of minimum weight $c(e)$.
        Set $V' := V' \cup \{w\}$, $E' := E' \cup \{e\}$, and $T := (V', E')$.
    end do
```

Example 5.4.2
Consider the following example:


Graph G=(V,E) with V={1,2,3,4,5,6} and
E={(1,2),(1,4),(2,3),(3,4),(3,6),(4,5),(4,6),(5,6)}.
1
2
3
4
5
6
7
10
1
4
3
2
5
8

Prim's algorithm produces the following subgraphs, starting with node 2:



The correctness of the algorithm is established in

Theorem 5.4.3

Prim's algorithm constructs a spanning tree with minimal costs for the graph G, provided G has a spanning tree. Prim's algorithm can be implemented in $\mathcal{O}(n^2)$ operations *(polynomial complexity).*

Proof: Can be shown in a similar way as the result for Kruskal's algorithm. □

Remark 5.4.4

There are even more efficient implementations of Prim's algorithm which use better data structures (Fibonacci heap) and need $\mathcal{O}(m+n\log n)$ operations only. Details can be found in Korte and Vygen [KV08], pages 131-133.

Chapter 6

Shortest Path Problems

Many combinatorial optimisation problems can be formulated as linear optimisation problems on graphs or networks. Among them are

- transportation problems
- maximum flow problems
- assignment problems
- shortest path problems

In this chapter we will discuss shortest path problems and the algorithms of Dijkstra and Floyd-Warshall. We will see that these algorithms have polynomial complexity.
Typical applications for shortest path problems arise for instance in satellite navigation systems for cars and routing tasks in computer networks.
Let a directed network $N = (V, E, c)$ with set of nodes $V = \{Q = 1, 2, \ldots, n = S\}$ and set of edges E, $|E| = m$, be given with:

- Let Q be a source (i.e. Q has no predecessor and no edge enters Q) and let S be a sink (i.e. S has no successor and no edge leaves S).
- Let $c : E \longrightarrow \mathbb{R}$ with $(i, j) \in E \mapsto c(i, j) = c_{ij}$ denote the length of the edge (i, j).
- N is connected.

The task is to find a shortest path from Q to S that is a path of minimal length. Herein, the length of a path $P = [v_1, \ldots, v_k, v_{k+1}]$ in the network N with $v_i \in V$, $i = 1, \ldots, k + 1$, is defined by

$$\sum_{i=1}^{k} c(v_i, v_{i+1}) = \sum_{i=1}^{k} c_{v_i v_{i+1}}.$$

Shortest path problem:
Find a shortest $Q - S-$path in the network N or decide that S is not reachable from Q.

6.1 Linear Programming Formulation

The shortest path problem can be written as a linear program. The idea is to transport one unit from Q to S through the network. You may think of a car that is driving on the road network. The task is then to find a shortest $Q-S-$path for this car.
Let's formalise this idea and let $x : E \to \mathbb{R}$, $(i,j) \in E \mapsto x(i,j) = x_{ij}$, denote the amount which moves along the edge (i,j). Of course we want x_{ij} to be either one or zero indicating whether an edge (i,j) is used or not. But for the moment let's neglect this additional restriction. We will see later that our linear program will always have a zero-one solution.
We supply one unit at Q and demand one unit at S. Moreover, nothing shall be lost at intermediate nodes $i = 2,3,\ldots,n-1$, i.e. the so-called conservation equations

$$\underbrace{\sum_{k:(j,k)\in E} x_{jk}}_{\text{outflow out of node } j} - \underbrace{\sum_{i:(i,j)\in E} x_{ij}}_{\text{inflow into node } j} = 0, \quad j \in V\backslash\{Q,S\}.$$

have to hold at every interior node $j = 2,\ldots,n-1$. As we supply one unit at source Q and demand one unit at sink S, the following conservation equations have to hold:

$$\underbrace{\sum_{k:(Q,k)\in E} x_{Qk}}_{\text{outflow out of node } Q} = 1,$$

$$-\underbrace{\sum_{i:(i,S)\in E} x_{iS}}_{\text{inflow into node } S} = -1.$$

These linear equations conservation equations can be written in compact form as

$$Hx = d,$$

where $d = (1,0,\ldots,0,-1)^\top$ and H is the so-called node-edge incidence matrix.

Definition 6.1.1 **(Node-edge incidence matrix)**
Let $G = (V,E)$ be a digraph with nodes $V = \{1,\ldots,n\}$ and edges $E = \{e_1,\ldots,e_m\}$. The matrix $H \in \mathbb{R}^{n\times m}$ with entries

$$h_{ij} = \begin{cases} 1, & \text{if } i \text{ is the tail of edge } e_j, \\ -1, & \text{if } i \text{ is the head of edge } e_j, \\ 0, & \text{otherwise} \end{cases}, \qquad i = 1,\ldots,n, \ j = 1,\ldots,m,$$

is called (node-edge) incidence matrix of G.

Remark 6.1.2

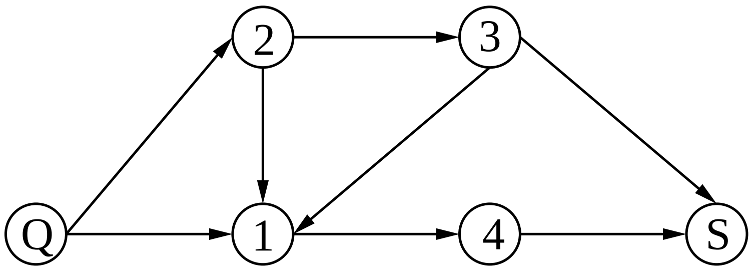
- *Every column of H refers to an edge $(i,j) \in E$ and is defined as follows:*

$$\begin{array}{cl} (i,j) & \\ \left(\begin{array}{c} \\ 1 \\ \\ \\ -1 \\ \\ \end{array}\right) & \begin{array}{l} \\ \leftarrow \text{ row } i \\ \\ \\ \leftarrow \text{ row } j \\ \\ \end{array} \end{array} \qquad \textit{consequence:} \quad \begin{array}{l} \textit{outflow is positive} \\ \textit{inflow is negative} \end{array}$$

- *Some authors define the incidence matrix to be the negative of H.*

Example 6.1.3

Directed graph $G = (V,E)$ with nodes $V = \{Q,1,2,3,4,S\}$ and edges $E = \{(Q,1),(Q,2),(2,1),(1,4),(2,3),(3,1),(3,S),(4,S)\}$:



Incidence matrix H:

$$H = \begin{array}{c} \\ \left(\begin{array}{cccccccc} (Q,1) & (Q,2) & (2,1) & (1,4) & (2,3) & (3,1) & (3,S) & (4,S) \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{array}\right) \end{array} \begin{array}{l} \\ \textit{node } Q \\ \textit{node } 1 \\ \textit{node } 2 \\ \textit{node } 3 \\ \textit{node } 4 \\ \textit{node } S \end{array}$$

Each row of H corresponds to a node in V and each column of H corresponds to an edge in E according to the ordering in V and E.
We can immediately see that the conservation equation is equivalent to $Hx = d$ where each component d_i of d denotes a demand ($d_i < 0$) or a supply ($d_i \geq 0$). For instance, with

$$x = (x_{Q1}, x_{Q2}, x_{21}, x_{14}, x_{23}, x_{31}, x_{3S}, x_{4S})^\top$$

conservation at the intermediate node 2 with $d_2 = 0$ reads as

$$-x_{Q2} + x_{21} + x_{23} = 0 (= d_2),$$

which on the left is just the row corresonding to node 2 in H multiplied by x.

There remains one question: Does the problem always have a solution with $x_{ij} \in \{0, 1\}$? Fortunately yes, because the matrix H is a totally unimodular matrix and d is integral.

Theorem 6.1.4

The node-edge incidence matrix H of a directed graph $G = (V, E)$ is totally unimodular.

Proof: Proof by induction on the size t of the $t \times t$ sub-matrices.
$t = 1$: Let B a 1×1 sub-matrix of H. As H is the incidence matrix, each column has at most one entry 1 and at most one entry -1. All remaining entries are 0. Hence, the determinant of B is in $\{0, 1, -1\}$.
Induction step: Let the assumption hold for $t - 1$ and let B be a $t \times t$ sub-matrix of H. Three cases may occur:

1. There is a zero column in B. Then $\det(B) = 0$.
2. Every column of B has exactly one entry 1 and one entry -1. Adding up the rows of B leads to a zero row. Hence, $\det(B) = 0$.
3. B has a column with exactly one entry with either 1 or -1. Then, $\det(B) = \pm 1 \cdot \det(\tilde{B})$, where $\tilde{B}$ is the $(t-1) \times (t-1)$ sub-matrix of B that is obtained by deleting in B the respective column and the row with the ± 1 entry. According to the induction assumption, $\det(\tilde{B}) \in \{0, 1, -1\}$. Hence, $\det(B) \in \{0, 1, -1\}$.

$\square$

Hence, the simplex method, when applied to the above LP formulation (6.1) of the shortest path problem, automatically terminates in an integer solution with $x_{ij} \in \{0, 1\}$. Herein, the upper bound 1 results from the conservation equations as only one unit is supplied in Q. Hence, constraints of type $x_{ij} \in \{0, 1\}$ or $x_{ij} \in \mathbb{Z}$ can be omitted.
Summarising, the shortest path problem can written as the following linear program.

$$\text{Minimise} \quad \sum_{(i,j) \in E} c_{ij} x_{ij} \quad \text{subject to} \quad Hx = d,\ x_{ij} \geq 0,\ (i,j) \in E. \tag{6.1}$$

Herein, the vector x has the components x_{ij}, $(i, j) \in E$, in the same order as in E.
What can we say about solvability of the problem? The problem is feasible, if there exists a $Q - S-$path. As $0 \leq x_{ij} \leq 1$ holds for all $(i, j) \in E$, the feasible set is compact and thus an optimal solution exists, provided a $Q - S-$path exists.

6.2 Dijkstra's Algorithm

In this section we restrict the discussion to shortest paths problems with non-negative weights:

Assumption 6.2.1 (Non-negative weights)
It holds $c_{ij} \geq 0$ for every $(i,j) \in E$ in the network N.

It can be shown that the rank of the $n \times m$ incidence matrix H is $n-1$ if the graph (V,E) is connected. In particular, the last constraint in $Hx = d$ which corresponds to the sink S is linearly dependent (redundant) and can be omitted. This can be seen as follows: As each column of H has one entry 1 and one entry -1, summing up all rows of H yields a zero vector. Hence, the rank of H is at most $n-1$.
As (V,E) is supposed to be connected, the underlying undirected graph has a spanning tree T with $n-1$ edges. The corresponding incidence matrix H_T of T (with directed edges) is a $n \times (n-1)$ submatrix of H.
As $n \geq 2$ holds there is at least one endpoint v_k in T that is incident with exactly one edge of T. Hence, in the k-th row of H_T there is exactly one entry ± 1.
By row and column perturbations we can achieve that this entry moves into the first row and column of H_T resp. H.
Removing node v_k and the corresponding edge yields a subtree T' of T and we can apply the same procedure recursively. After $n-1$ steps H has the structure

$$\left(\begin{array}{cccc|ccc}
\pm 1 & 0 & \ldots & 0 & h'_{1n} & \ldots & h'_{1m} \\
\star & \ddots & \ddots & \vdots & \vdots & & \\
\vdots & \ddots & \ddots & 0 & \vdots & & \\
\star & \ldots & \star & \pm 1 & h'_{n-1,n} & \ldots & h'_{n-1,m} \\
\hline
h'_{n1} & \ldots & \ldots & h'_{n,n-1} & h'_{nn} & \ldots & h'_{nm}
\end{array}\right),$$

where the last row depends linearly on the remaining rows. This shows that the rank of H is actually $n-1$.
Using this reasoning, (6.1) is equivalent to the problem

$$\text{Minimise} \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \quad \text{subject to} \quad \tilde{H}x = \tilde{d},\ x_{ij} \geq 0,\ (i,j) \in E, \tag{6.2}$$

where $\tilde{d} = (1,0,\ldots,0)^\top \in \mathbb{R}^{n-1}$ and $\tilde{H}$ is the incidence matrix H with the last row omitted.
The dual of (6.2) reads as

$$\text{Maximise} \quad \lambda_1 \quad \text{subject to} \quad \lambda_i - \lambda_j \leq c_{ij}, \qquad (i,j) \in E. \tag{6.3}$$

As $c_{ij} \geq 0$ we can easily provide a feasible dual point: $\lambda_i = 0$ for every $i = 1, \ldots, n-1$. For notational convenience, we also define $\lambda_n = 0$, where λ_n refers to the sink node S and the omitted last row of H.
We intend to apply the Primal-Dual Algorithm 3.4.1 to (6.2) and define the index set

$$B = \{(i,j) \in E \mid \lambda_i - \lambda_j = c_{ij}\}.$$

The restricted primal problem with artificial slack variables $y = (y_1, \ldots, y_{n-1})^\top$ reads as

$$\text{Minimise} \quad \sum_{i=1}^{n-1} y_i \quad \text{subject to} \quad \tilde{H}_B x_B + y = \tilde{d},\ x_{ij} \geq 0,\ (i,j) \in B,\ y \geq 0. \tag{6.4}$$

The dual of the restricted primal is given by

$$\begin{array}{ll} \text{Maximise} & \lambda_1 \\ \text{subject to} & \lambda_i - \lambda_j \leq 0, \qquad (i,j) \in B, \\ & \lambda_i \leq 1, \qquad i = 1, \ldots, n-1, \\ & \lambda_n = 0. \end{array} \tag{6.5}$$

The constraint $\lambda_n = 0$ is due to the fact, that we deleted the last row of H. The problem (6.5) can be easily solved. As $\lambda_1 = \lambda_Q \leq 1$ has to be maximised, we can set $\lambda_1 = 1$ and propagate the 1 to all nodes reachable from node 1 in order to satisfy the constraints $\lambda_i - \lambda_j \leq 0$. Hence, an optimal solution of (6.5) is given by

$$\bar{\lambda}_i = \begin{cases} 0, & \text{if node } S = n \text{ is reachable from node } i \text{ using edges in } B, \\ 1, & \text{otherwise,} \end{cases} \qquad i = 1, \ldots, n-1.$$

Notice that there may be other solutions to (6.5). We then proceed with the primal-dual algorithm by computing the step-size

$$t_{max} = \min_{(i,j) \notin B : \bar{\lambda}_i - \bar{\lambda}_j > 0} \{c_{ij} - (\lambda_i - \lambda_j)\},$$

updating $\lambda := \lambda + t_{max}\bar{\lambda}$, and performing the next iteration of the primal-dual algorithm.

Optimality:

If it turns out that there is a path from the source Q to the sink S using edges in the index set B, then $\bar{\lambda}_1 = 0$ and the optimal objective function values of the dual of the restricted primal and the restricted primal are both zero and the primal-dual algorithm stops with an optimal solution.

Observations and interpretations:

The following theorem gives an important explanation of what is going on in the primal-dual algorithm for the shortest path problem. We use the set

$$W = \{i \in V \mid S \text{ is reachable from } i \text{ by edges in } B\} = \{i \in V \mid \bar{\lambda}_i = 0\}.$$

Theorem 6.2.2

Let $N = (V, E, c)$ with $|V| = n$ be a network with positive weight $c_{ij} > 0$ for every $(i, j) \in E$.

(a) If the primal-dual algorithm starts with the feasible dual point $\lambda_i = 0$, $i = 1, \ldots, n$, then the shortest path from Q to S is found after at most $n - 1$ iterations.

(b) In every iteration an edge $(i, j) \in E$ with $i \notin W$ and $j \in W$ is added to the index set B. This edge will stay in B for all following iterations.

(c) The value of the variable λ_i with $i \in W$ is the length of a shortest path from node i to node $S = n$.

Proof:

- Initialisation:

  Consider the first iteration of the algorithm. Let $\ell = 0$ denote the iteration index. As $c_{ij} > 0$ for all $(i, j) \in E$, $\lambda_i^{(0)} = 0$, $i = 1, \ldots, n$, is feasible for the dual problem. Moreover, it holds $B^{(0)} = \emptyset$ because $0 = \lambda_i^{(0)} - \lambda_j^{(0)} < c_{ij}$ for all $(i, j) \in E$.

  Then, in the first iteration $\ell = 0$ of the algorithm it holds $\bar{\lambda}_n = 0$, $\bar{\lambda}_i = 1$, $i = 1, \ldots, n - 1$, and $W^{(0)} = \{n\}$.

  Let $c_{kn} := \min_{(i,n) \in E} c_{in}$. Then, $t_{max} = c_{kn}$ and after the first iteration we have

$$\lambda_i^{(1)} = \begin{cases} \lambda_i^{(0)}, & i \in W^{(0)}, \\ \lambda_i^{(0)} + t_{max}, & i \notin W^{(0)} \end{cases} = \begin{cases} 0, & i = n, \\ c_{kn}, & i = 1, \ldots, n-1. \end{cases}$$

  In particular, $\lambda_k^{(1)} = c_{kn}$ and thus $\lambda_k^{(1)} - \lambda_n^{(1)} = c_{kn}$, i.e. the dual constraint $\lambda_k - \lambda_n \leq c_{kn}$ becomes active for $\lambda^{(1)}$. Hence, $(k, n) \in E$ will be added to $B^{(0)}$ and $\lambda_k^{(1)} = c_{kn} = \min_{(i,n) \in E} c_{in}$ denotes the length of a shortest $k - n-$path.

- Induction step:

  The assertion is now shown by induction for the iteration index ℓ. Let $\lambda_j^{(\ell)}$, $j \in W^{(\ell)}$, denote the length of a shortest $j - n-$path at iteration ℓ. Let $\bar{\lambda}$ denote a solution of (DRP). It holds $\bar{\lambda}_i = 1$ for every $i \notin W^{(\ell)}$ and $\bar{\lambda}_j = 0$ for every $j \in W^{(\ell)}$.

The step size then computes to

$$\begin{aligned} t_{max} &= c_{pq} - (\lambda_p^{(\ell)} - \lambda_q^{(\ell)}) \\ &:= \min_{(i,j)\notin B^{(\ell)}, \bar{\lambda}_i - \bar{\lambda}_j > 0} \left(c_{ij} - (\lambda_i^{(\ell)} - \lambda_j^{(\ell)}) \right) \\ &= \min_{(i,j)\notin B^{(\ell)}, i\notin W^{(\ell)}, j\in W^{(\ell)}} \left(c_{ij} - (\lambda_i^{(\ell)} - \lambda_j^{(\ell)}) \right). \end{aligned}$$

For every $(i,j) \in E$ with $i \notin W^{(\ell)}$ and $j \in W^{(\ell)}$ it holds

$$\lambda_i^{(\ell+1)} = \begin{cases} \lambda_i^{(\ell)}, & i \in W^{(\ell)}, \\ \lambda_i^{(\ell)} + t_{max}, & i \notin W^{(\ell)}. \end{cases}$$

In particular,

$$\lambda_p^{(\ell+1)} = \lambda_p^{(\ell)} + t_{max} = \lambda_p^{(\ell)} + c_{pq} - \lambda_p^{(\ell)} + \lambda_q^{(\ell)} = \lambda_q^{(\ell)} + c_{pq},$$

i.e. (p,q) with $p \notin W^{(\ell)}$ and $q \in W^{(\ell)}$ will be added to $B^{(\ell)}$. As $\lambda_q^{(\ell)}$ by induction assumption denotes the length of a shortest $q-n-$path and $\lambda_p^{(\ell+1)}$ satisfies

$$\lambda_p^{(\ell+1)} = \lambda_q^{(\ell)} + c_{pq} = \min_{(i,j)\notin B^{(\ell)}, i\notin W^{(\ell)}, j\in W^{(\ell)}} \left(\lambda_j^{(\ell)} + c_{ij} \right),$$

$\lambda_k^{(\ell+1)}$ contains the length of a shortest $p-n-$path.

- Finiteness:

  In each iteration a node p is added to $W^{(\ell)}$, hence, the algorithm will terminate after at most $n-1$ iterations.

- Active dual constraints:

  It remains to show that whenever an edge $(i,j) \in E$ with $i \notin W^{(\ell-1)}$ and $j \in W^{(\ell-1)}$ was added to $B^{(\ell-1)}$, then $(i,j) \in B^{(k)}$ for $k \geq \ell$. So, let $(i,j) \in B^{(\ell)}$ with $i,j \in W^{(\ell)}$. Then, in iteration ℓ it holds $\bar{\lambda}_i = \bar{\lambda}_j = 0$ and thus $\lambda_i^{(\ell+1)} = \lambda_i^{(\ell)}$ and $\lambda_j^{(\ell+1)} = \lambda_j^{(\ell)}$ and thus $\lambda_i^{(\ell+1)} - \lambda_j^{(\ell+1)} = \lambda_i^{(\ell)} - \lambda_j^{(\ell)} = c_{ij}$, i.e. $(i,j) \in B^{(\ell+1)}$ and $i,j \in W^{(\ell+1)}$. The assertion follows inductively. □

Remark 6.2.3

- *The assumption of positive weights c_{ij} in the theorem is essential for the interpretation of the dual variables as lengths of shortest paths.*

- *The algorithm not only finds the lengths of shortest paths but also for every node in W the shortest path to S.*

- *If t_{max} is not uniquely determined by some $(i,j) \notin B$, then every such (i,j) can be added to B. Notice that the above theorem only provides an interpretation for those dual variables λ_i for which i belongs to W (there may be other variables in B).*

This essentially is the working principle of Dijkstra's algorithm. An efficient implementation looks as follows (in contrast to the primal-dual algorithm we start with node Q instead of S and work through the network in forward direction instead of backwards):

Algorithm 6.2.4 **(Dijkstra's Algorithm)**

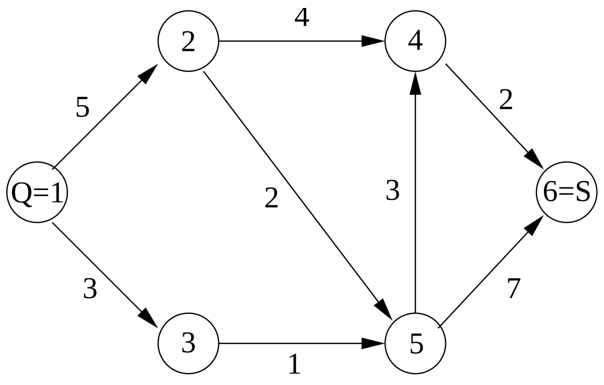
```
(0) Initialisation: Set $W = \{1\}$, $d(1) = 0$, and $d(i) = \infty$ for every $i \in V \setminus \{1\}$.

(1) For every $i \in V \setminus \{1\}$ with $(1,i) \in E$ set $d(i) = c_{1i}$ and $p(i) = 1$.

(2) While $W \neq V$ do
        Find $k \in V \setminus W$ with $d(k) = \min\{d(i) \mid i \in V \setminus W\}$.
        Set $W = W \cup \{k\}$.
        For every $i \in V \setminus W$ with $(k,i) \in E$ do
            If $d(i) > d(k) + c_{ki}$ then
                Set $d(i) = d(k) + c_{ki}$ and $p(i) = k$.
            end if
        end do
    end do
```

Output:

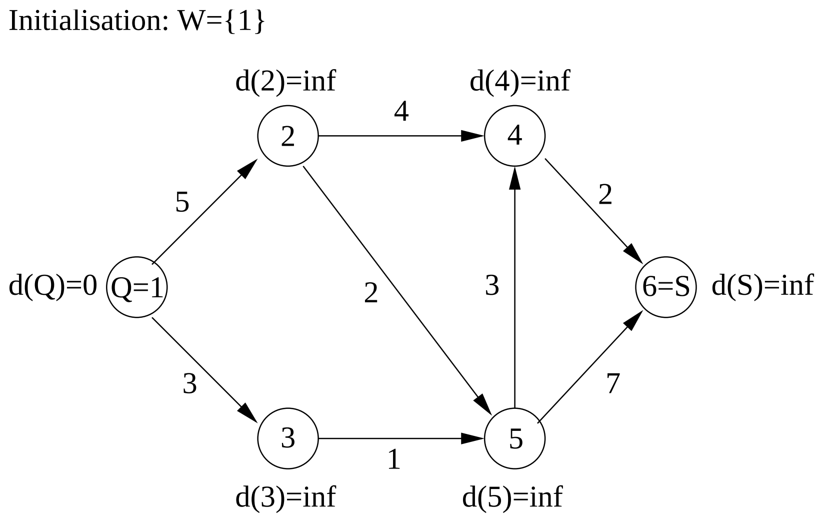
- shortest paths from node $Q = 1$ to all $i \in V$ and their lengths.
- for every $i \in V$, $d(i)$ is the length of the shortest $Q-i$–path. $p(i)$ denotes the predecessor of node i on the shortest path.
- if i is not reachable from Q, then $d(i) = \infty$ and $p(i)$ is undefined.

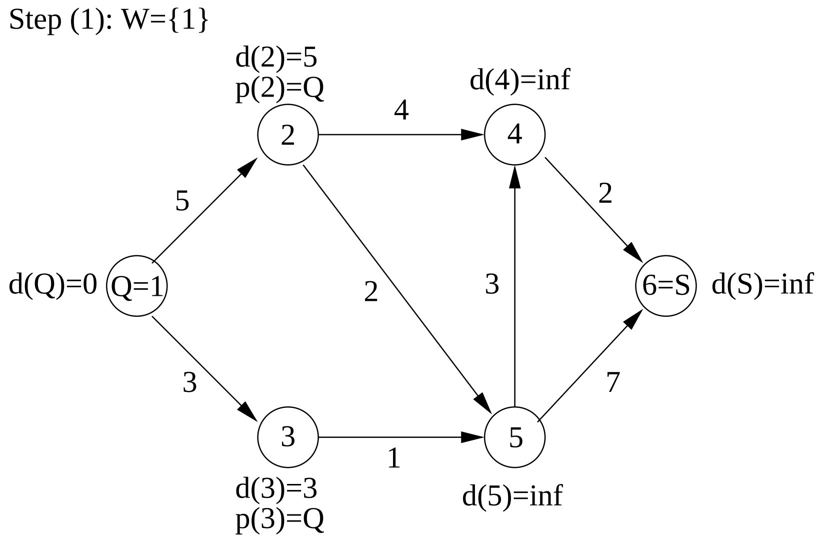
Complexity:
The While-loop has $n-1$ iterations. Each iteration requires at most n steps in the For-Loop within the While-loop to update d. Likewise at most n steps are needed to find the minimum $d(k)$ within the While-loop. Hence, the overall complexity of Dijkstra's algorithm is $\mathcal{O}(n^2)$ where n denotes the number of nodes in the network.

Example 6.2.5
Find the shortest path from $Q = 1$ to $S = 6$ using Dijkstra's algorithm for the following network.

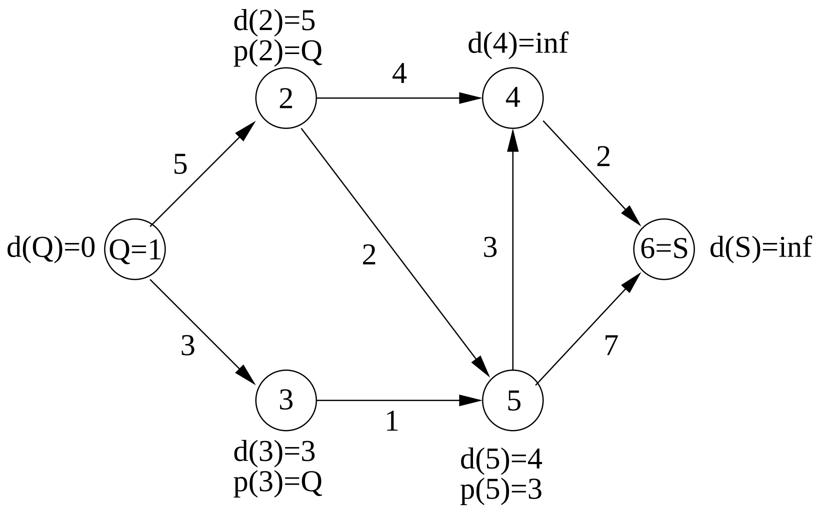


Dijkstra's algorithm produces the following intermediate steps:

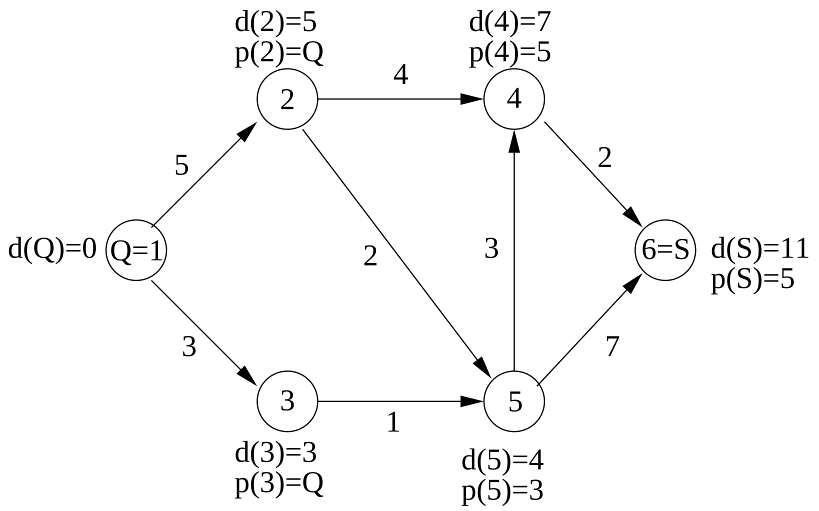




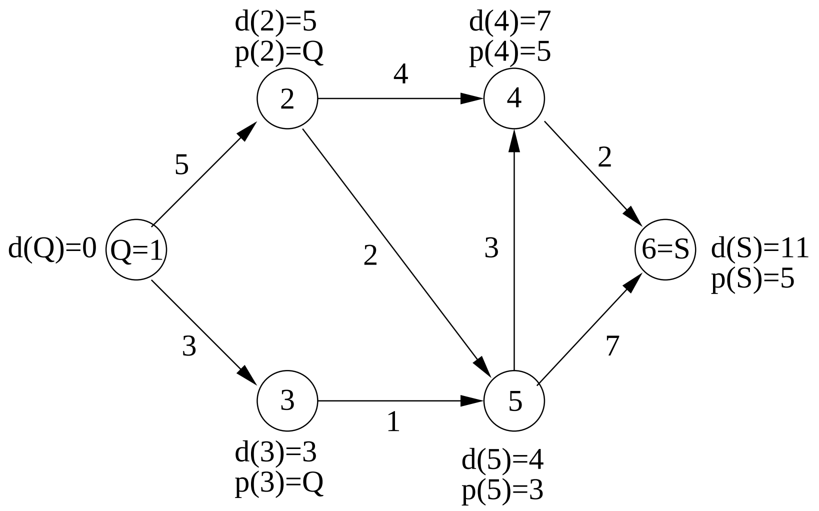
Step (2), Iteration 1: W={1,3}, k=3

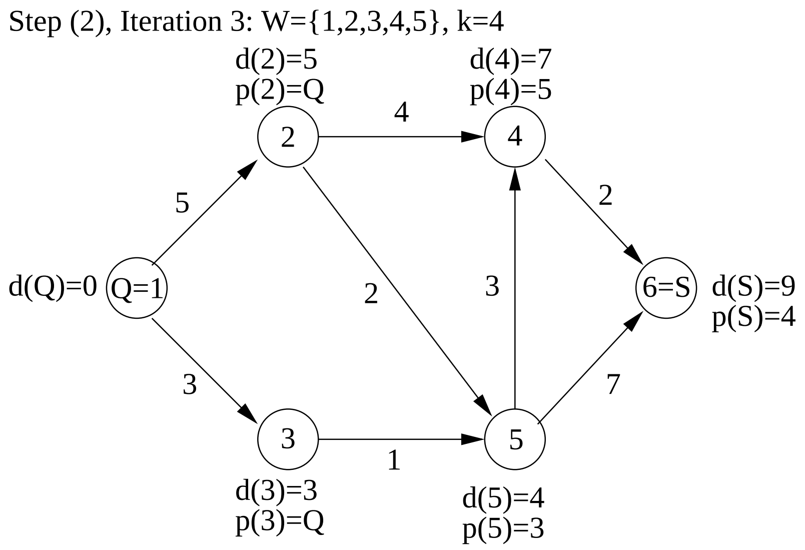


Step (2), Iteration 2: W={1,3,5}, k=5

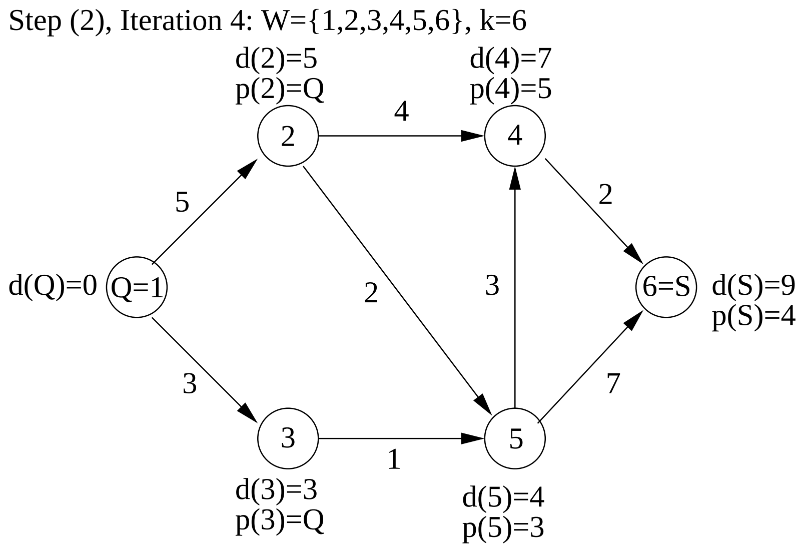


Step (2), Iteration 3: W={1,2,3,5}, k=2





Final result: The shortest path from Q to S is the path $[Q, 3, 5, 4, S]$ with length 9.



Remark 6.2.6

- *Dijkstra's algorithm only works for non-negative weights $c_{ij} \geq 0$. In the presence of negative weights it may fail.*

- *The assumptions that Q be a source and S be a sink are not essential and can be dropped.*

6.3 Algorithm of Floyd-Warshall

In contrast to the previous section, we now allow negative weights $c_{ij} < 0$, but with the restriction that the network does not contain a circuit of negative length.

Assumption 6.3.1

The network N does not contain a circuit of negative length.

The algorithm of Floyd-Warshall computes not only the shortest paths from Q to any other node $j \in V$ (as Dijkstra's algorithm) but also the shortest paths between any two nodes $i \in V$ and $j \in V$. Moreover, it is very simple to implement. Finally, the algorithm works for negative weights and it detects circuits of negative length. It is not a primal-dual algorithm.

Algorithm 6.3.2 **(Floyd-Warshall)**

```
(0) Initialisation:
    Set $d_{ij} = \infty$ for every $(i,j)$, $i,j = 1,\dots,n$.
    Set $d_{ij} = c_{ij}$ for every $(i,j) \in E$.
    Set $d_{ii} = 0$ for every $i = 1,\dots,n$.
    Set $p_{ij} = i$ for every $i,j \in V$.

(1) For $j = 1,\dots,n$ do
        For $i = 1,\dots,n$, $i \neq j$ do
            For $k = 1,\dots,n$, $k \neq j$ do
                If $d_{ik} > d_{ij} + d_{jk}$ then
                    Set $d_{ik} = d_{ij} + d_{jk}$ and $p_{ik} = p_{jk}$.
                end if
            end do
        end do
    end do
```

Output:

- Distance matrix
$$D = \begin{pmatrix} d_{11} & \cdots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{n1} & \cdots & d_{nn} \end{pmatrix},$$
where d_{ij} is the length of a shortest path from i to j.

- Predecessor matrix
$$P = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{pmatrix},$$

where (p_{ij}, j) is the final edge of a shortest path from i to j, that is p_{ij} is the predecessor of the node j on a shortest $i-j-$path (if such a path exists).

Complexity:

Each For-loop has at most n iterations. Hence, the overall complexity of the algorithm of Floyd-Warshall is $\mathcal{O}(n^3)$ where n denotes the number of nodes in the network.

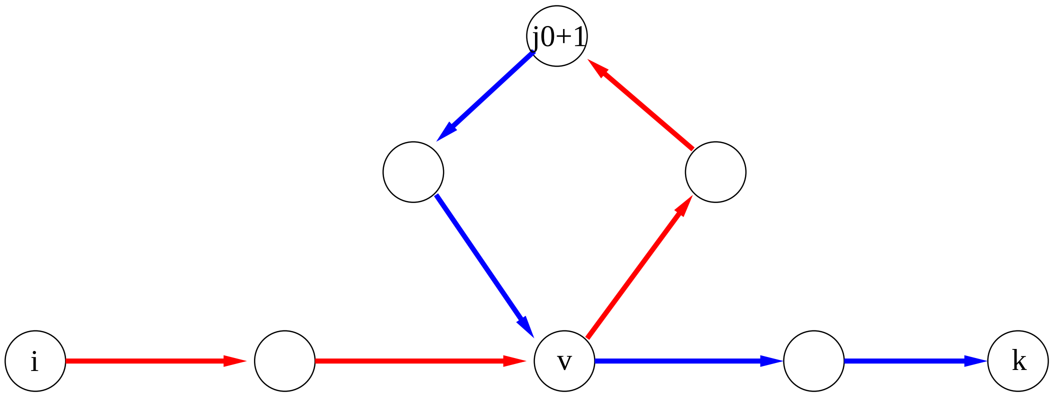
The correctness of the algorithm is established in the following theorem.

Theorem 6.3.3

For every $i, j \in V$ the algorithm of Floyd-Warshall correctly computes the length d_{ij} of a shortest path from i to j, if no circuit of negative length exists in the network N.

Proof: We will show the following: After the outer loop for $j = 1, \ldots, j_0$ is completed, the variable d_{ik} (for all i and k) contains the length of a shortest $i-k-$path with intermediate nodes $v \in \{1, \ldots, j_0\}$ only.

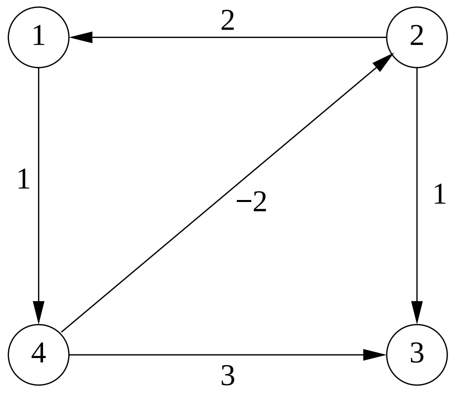
Induction for j_0: For $j_0 = 0$ the assertion is true owing to the initialisation in step (0).
Let the assertion hold for some $j_0 \in \{0, \ldots, n-1\}$. We have to show the assertion to be true for $j_0 + 1$ as well.
Let i and k be arbitrary nodes. According to the induction assumption, d_{ik} contains the length of an $i-k-$path with intermediate nodes $v \in \{1, \ldots, j_0\}$ only. In the loop for $j = j_0 + 1$, d_{ik} is replaced by $d_{i,j_0+1} + d_{j_0+1,k}$, if this value is smaller.
It remains to show that the corresponding $i-(j_0+1)-$path P and the $(j_0+1)-k-$path $\tilde{P}$ have no inner nodes in common (otherwise the combined way $P \cup \tilde{P}$ would not be a path by definition).
Suppose that P and $\tilde{P}$ have an inner node v in common, see figure below (P is colored red and $\tilde{P}$ is blue).



Then the combined walk $P \cup \tilde{P}$ contains at least one circuit from v to $j_0 + 1$ (in P) and back to v (in $\tilde{P}$). By assumption any such circuit does not have negative length and thus they can be shortcut. Let R denote the shortcut $i-k-$path. This path uses nodes in $\{1, \dots, j_0\}$ only and the length is no longer than $d_{i,j_0+1} + d_{j_0+1,k} < d_{ik}$. But this contradicts the induction assumption as d_{ik} was supposed to be the length of a shortest $i-k-$path using nodes in $\{1, \dots, j_0\}$ only. □

Example 6.3.4

Consider the following network which contains a circuit. Notice that the circuit does not have negative length.



The algorithm of Floyd-Warshall produces the following result:

Initialisation:

$D:$

0	∞	∞	1
2	0	1	∞
∞	∞	0	∞
∞	-2	3	0

$P:$

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

$j = 1$:

$D:$

0	∞	∞	1
2	0	1	3
∞	∞	0	∞
∞	-2	3	0

$P:$

1	1	1	1
2	2	2	1
3	3	3	3
4	4	4	4

$j = 2$:

$D:$

0	∞	∞	1
2	0	1	3
∞	∞	0	∞
0	-2	-1	0

$P:$

1	1	1	1
2	2	2	1
3	3	3	3
2	4	2	4

$j = 3$:

$D:$

0	∞	∞	1
2	0	1	3
∞	∞	0	∞
0	-2	-1	0

$P:$

1	1	1	1
2	2	2	1
3	3	3	3
2	4	2	4

$j = 4$:

D :

0	−1	0	1
2	0	1	3
∞	∞	0	∞
0	−2	−1	0

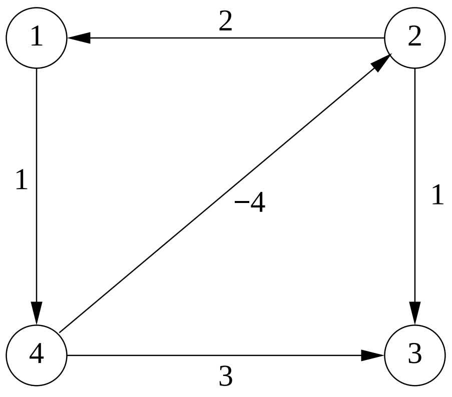
P :

1	4	2	1
2	2	2	1
3	3	3	3
2	4	2	4

From the predecessor matrix P we can easily reconstruct an optimal path. For instance: The shortest path from node 1 to node 3 has length $d_{13} = 0$. The entry $p_{13} = 2$ in P tells us the predecessor of node 3 on that path, that is node 2. In order to get the predecessor of node 2 on that path we use the entry $p_{12} = 4$. Now, the predecessor of node 4 on that path is $p_{14} = 1$. Hence, we have reconstructed an optimal path from node 1 to node 3 by backtracking. The path is $[1, 4, 2, 3]$.
The reasoning behind this backtracking method is as follows: Suppose the $v_1 - v_{k+1}$−path $[v_1, v_2, \ldots, v_k, v_{k+1}]$ has minimal length. Then for every $1 \leq j < \ell \leq k+1$ the $v_j - v_\ell$−path $[v_j, \ldots, v_\ell]$ connecting v_j and v_ℓ has minimal length as well. In other words: Subpaths of optimal paths remain optimal.

Remark 6.3.5 (Circuit Detection)
It can be shown that the diagonal elements d_{ii}, $i = 1, \ldots, n$, remain non-negative if and only if the network does not contain a circuit of negative length.
Hence, the algorithm of Floyd-Warshall is able to detect circuits of negative length. It can be halted if a diagonal element in D becomes negative.

Example 6.3.6 (Exercise)
Apply the algorithm of Floyd-Warshall to the following network which contains a circuit of negative length.

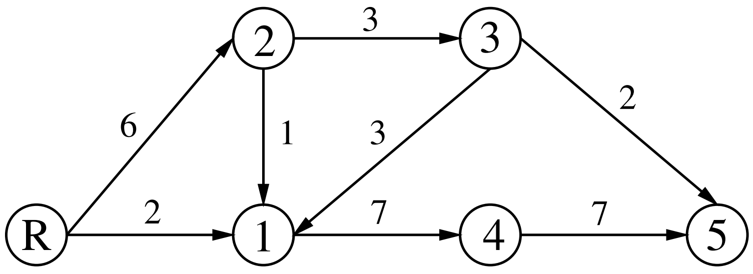


Chapter 7

Maximum Flow Problems

In this chapter we will discuss a special class of network optimisation problems – maximum flow problems – and the algorithm of Ford and Fulkerson. We will see that this algorithm has polynomial complexity. We have seen a specific example of a maximum flow problem already in Example 1.1.4.

Example 7.0.7 **(Maximum flow problem)**
An oil company intends to transport as much oil as possible through a given system of pipelines from vertex R to vertex 5, see the graph below. Depending on the diameter of each pipeline the capacity is limited by the numbers (in million barrel per hour) next to the edges of the graph.



The set of vertices of the directed graph is given by $V = \{R, 1, 2, 3, 4, 5\}$ *and the edges are* $E = \{R1, R2, 14, 21, 23, 35, 31, 45\}$. *For each edge* $ij \in E$ *of the graph let* x_{ij} *denote the amount of oil transported on edge* ij.

The following conservation equation has to hold at each inner vertex $k \in V$, $k \notin \{R, 5\}$ *('outflow-inflow=0'):*

$$\sum_{j:(k,j)\in E} x_{kj} - \sum_{i:(i,k)\in E} x_{ik} = 0$$

Moreover, capacity constraints apply at every edge $ij \in E$ *(*u_{ij} *denotes the maximum capacity of edge* ij*):*

$$0 \leq x_{ij} \leq u_{ij}$$

Let v be the amount of oil being pumped into vertex R. The following linear program solves the problem:

Maximise v

s.t.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_{R1} \\ x_{R2} \\ x_{14} \\ x_{21} \\ x_{23} \\ x_{31} \\ x_{35} \\ x_{45} \end{pmatrix} + \begin{pmatrix} -v \\ 0 \\ 0 \\ 0 \\ 0 \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{aligned} & v \geq 0, \\ & 0 \leq x_{R1} \leq 2, \\ & 0 \leq x_{R2} \leq 6, \\ & 0 \leq x_{14} \leq 7, \\ & 0 \leq x_{21} \leq 1, \\ & 0 \leq x_{23} \leq 3, \\ & 0 \leq x_{31} \leq 3, \\ & 0 \leq x_{35} \leq 2, \\ & 0 \leq x_{45} \leq 7. \end{aligned}$$

7.1 Problem Statement

Let a directed network $N = (V, E, u)$ with set of nodes $V = \{Q = 1, 2, \ldots, n = S\}$ and set of edges E, $|E| = m$, be given. Let $u : E \to \mathbb{R}$ with $(i, j) \in E \mapsto u(i, j) = u_{ij}$ denote maxmimal capacities. Let $x : E \to \mathbb{R}$ with $(i, j) \in E \mapsto x(i, j) = x_{ij}$ denote the amount of goods (e.g.oil) transported on edge $(i, j) \in E$.
Let Q and S be two distinct nodes (source and sink, respectively). The remaining nodes $2, 3, \ldots, n-1$ are called interior nodes.

Assumption 7.1.1

(i) Let Q be a source (i.e. Q has no predecessor and no edge enters Q) and let S be a sink (i.e. S has no successor and no edge leaves S).

(ii) The amount x_{ij} transported on edge $(i, j) \in E$ is subject to capacity constraints

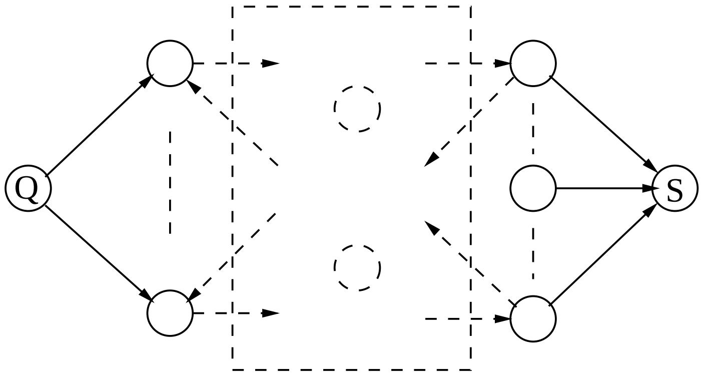
$$0 \leq x_{ij} \leq u_{ij}, \qquad u_{ij} \geq 0, \ \forall (i, j) \in E. \tag{7.1}$$

Herein, $u_{ij} = \infty$ is permitted.

(iii) The conservation equations

$$\underbrace{\sum_{k:(j,k)\in E} x_{jk}}_{\textit{outflow out of node } j} - \underbrace{\sum_{i:(i,j)\in E} x_{ij}}_{\textit{inflow into node } j} = 0, \quad j \in V\backslash\{Q,S\}. \tag{7.2}$$

hold at every interior node $j = 2, \ldots, n-1$.



Remark 7.1.2

Problems with multiple sources or sinks are not excluded by the above assumptions. In fact, such problems can be transformed into problems with only one source and one sink. In order to transform a problem with multiple sources, an artificial 'super source' and edges without capacity restrictions leaving the super source and entering the original sources can be introduced. Likewise, a 'super sink' and edges without capacity restrictions leaving the original sinks and entering the super sink can be introcuded for problems with multiple sinks. For the transformed problem, assumption (i) is satisfied.

Definition 7.1.3 (Flow, feasible flow, volume, maximum flow)

- *A flow $x : E \to \mathbb{R}$ through the network is an assignment of values $x(i,j) = x_{ij}$, such that the conservation equations (7.2) hold.*
- *A flow x is feasible, if the capacity constraints (7.1) are satisfied.*
- *The value*

$$v = \sum_{(Q,j)\in E} x_{Qj} \left(= \sum_{(i,S)\in E} x_{iS} \right) \tag{7.3}$$

  is called volume of the flow x.
- *A feasible flow with maximum volume is called maximum flow through the network.*

Using these definitions, the maximum flow problem reads as follows.

Maximum flow problem:

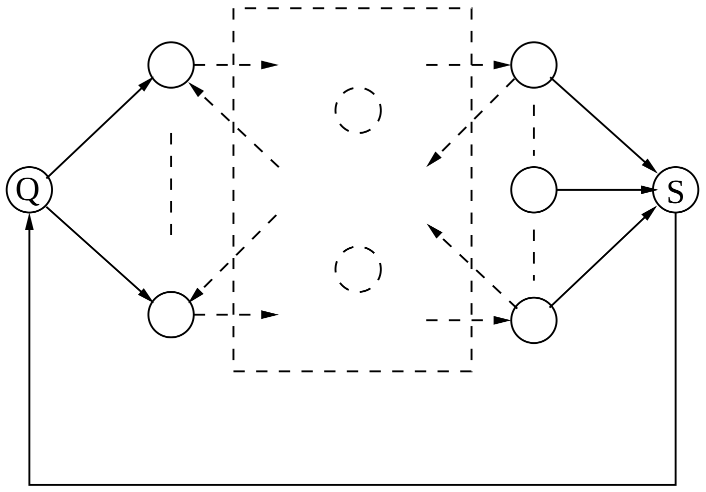
Find a maximum flow x through the network.

Solvability:

- The maximum flow problem is feasible as $x_{ij} = 0$ for every $(i,j) \in E$ satisfies all constraints.
- If all capacity bounds u_{ij} satisfy $0 \le u_{ij} < \infty$, then the maximum flow problem has an optimal solution as in this case the feasible set is compact and non-empty.

7.2 Linear Programming Formulation

We intend to write the maximum flow problem as a linear program and note that digraphs can be uniquely described by matrices.
The maximum flow problem can be formulated as a linear program, if an additional edge (S,Q) from the sink to the source with no capacity restriction is introduced, cf. figure below.



The volume v is transported along edge (S,Q) and the conservation equations are extended to the source Q and the sink S. The conservation equations can be written in a compact way using the incidence matrix H (compare Definition 6.1.1) of the original network as

$$Hx + v \cdot d = 0,$$

where $d = (-1, 0, \ldots, 0, 1)^\top \in \mathbb{R}^n$ is the incidence column for the additional edge (S,Q).

The maximum flow problem is equivalent to the following linear program:

$$\begin{array}{ll} \text{Maximise} & v \\ \text{subject to} & \begin{array}{rcll} Hx + v \cdot d & = & 0, & \\ 0 \leq \quad x_{ij} & \leq & u_{ij}, & (i,j) \in E, \\ 0 \leq \quad v. & & & \end{array} \end{array}$$

The dual problem reads as

$$\begin{array}{ll} \text{Minimise} & \displaystyle\sum_{(i,j)\in E} z_{ij} u_{ij} \\ \text{subject to} & \begin{array}{rcll} y_j - y_i - z_{ij} & \leq & 0, & (i,j) \in E, \\ y_S - y_Q & \geq & 1, & \\ z_{ij} & \geq & 0, & (i,j) \in E. \end{array} \end{array}$$

where z_{ij}, $(i,j) \in E$, and y_i, $i \in V$, denote the dual variables. The dual variables y_i, $i \in V$, are known as node numbers.

For the maximum flow problem the complementary slackness conditions read as

$$\begin{array}{rcll} 0 & = & x_{ij}(y_j - y_i - z_{ij}) & \text{for } (i,j) \in E, \\ 0 & = & z_{ij}(u_{ij} - x_{ij}) & \text{for } (i,j) \in E, \\ 0 & = & v(-1 + y_S - y_Q). & \end{array} \tag{7.4}$$

7.3 Minimal cuts and maximum flows

Definition 7.3.1 (Cut, capacity of a cut)

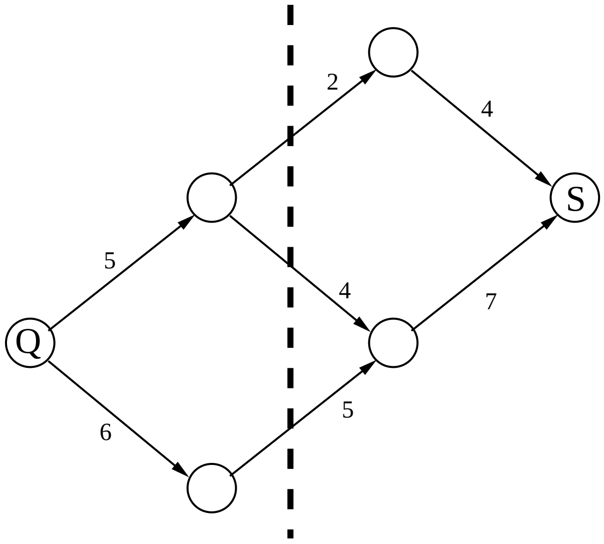
- *A subset $C \subset V$ of nodes is called cut in a network, if $Q \in C$ and $S \notin C$.*

- *The capacity of a cut C is defined to be the value*

$$\sum_{j\in C, k\notin C} u_{jk}.$$

Example 7.3.2

The capacity of the cut is $2 + 4 + 5 = 11$.



Corollary 7.3.3

Let C be a cut and x a flow with volume v. Then

$$v = \sum_{j\in C, k\notin C} x_{jk} - \sum_{i\notin C, j\in C} x_{ij}. \tag{7.5}$$

Proof: Summation of the conservation equations w.r.t. $j \neq Q, j \in C$ and adding the identy $v = \sum_{(Q,k)\in E} x_{Qk}$ yields

$$v + \sum_{j\in C, j\neq Q} \sum_{(i,j)\in E} x_{ij} = \sum_{(Q,k)\in E} x_{Qk} + \sum_{j\in C, j\neq Q} \sum_{(j,k)\in E} x_{jk}.$$

Changing the order of summation leads to

$$\begin{aligned} v + \sum_{j\in C, i\in C, (i,j)\in E} x_{ij} + \sum_{j\in C, i\notin C, (i,j)\in E} x_{ij} &= \sum_{(Q,k)\in E} x_{Qk} + \sum_{j\in C, j\neq Q, k\in C} x_{jk} + \sum_{j\in C, j\neq Q, k\notin C} x_{jk} \\ &= \sum_{j\in C, k\in C} x_{jk} + \sum_{j\in C, k\notin C} x_{jk} \end{aligned}$$

The assertion follows immediately:

$$v = \sum_{j\in C, k\notin C} x_{jk} - \sum_{j\in C, i\notin C, (i,j)\in E} x_{ij}.$$

□

Remark 7.3.4

If $C = \{Q\}$ and $C = V\backslash\{S\}$, respectively, are chosen in the preceding corollary, then (7.3) is obtained immediately.

In particular, it follows that the volume of a flow is bounded by the capacity of a cut:

$$v = \sum_{j\in C, k\notin C} x_{jk} - \sum_{i\notin C, j\in C} x_{ij} \overset{x_{jk}\leq u_{jk}, x_{ij}\geq 0}{\leq} \sum_{j\in C, k\notin C} u_{jk} \tag{7.6}$$

This fact can be described by the dual problem:

Theorem 7.3.5

Every cut C defines a feasible point of the dual problem according to

$$z_{ij} = \begin{cases} 1, & \text{if } i \in C,\ j \notin C, \\ 0, & \text{otherwise} \end{cases}$$
$$y_i = \begin{cases} 0, & \text{if } i \in C, \\ 1, & \text{otherwise} \end{cases}$$

The value of the dual problem at this point equals the capacity of the cut.

Proof: Feasibility immediately follows by computing the inequality constraints for the cases $i, j \in C$, and $i, j \notin C$, and $i \in C, j \notin C$, and $i \notin C, j \in C$. The values of the inequality constraints for these cases turn out to be $0, 0, 0, -1$, respectively. As $z_{ij} = 1$ holds exactly for those edges that define the capacity of the cut, the assertion follows. □

Remark 7.3.6

The theorem states that the capacity of a cut is greater than the volume v of a flow x, because owing to the weak duality theorem, the objective function value of the dual problem is always greater than or equal to the objective function value v of the primal problem.

Theorem 7.3.7 **(Max-Flow Min-Cut)**

Every maximum flow problem has exactly one of the following properties:

(a) There exist feasible flows with arbitrarily large volume and every cut has an unbounded capacity.

(b) There exists a maximum flow whose volume is equal to the minimal capacity of a cut.

Proof: We have seen that every maximum flow problem can be transformed into a linear program. According to the fundamental theorem of linear programming, every linear program satisfies exactly one of the following alternatives:

(i) It is unbounded.

(ii) It has an optimal solution.

(iii) It is infeasible.

The third alternative does not apply to maximum flow problems as the zero flow $x = 0$ is feasible. Consequently, the problem is either unbounded or it has an optimal solution. If the problem is unbounded, then there exists a flow with unbounded volume. According to (7.6) for every cut C and every flow x with volume v it holds

$$v \leq \sum_{j\in C, k\notin C} u_{jk}. \tag{7.7}$$

As the volume is unbounded, the capacity of every cut has to be infinity.
If the problem has an optimal solution, the simplex method finds an optimal solution x with node numbers y_k such that

$$0 > y_j - y_i \quad\Rightarrow\quad x_{ij} = 0, \tag{7.8}$$

$$0 < y_j - y_i \quad\Rightarrow\quad x_{ij} = u_{ij} \tag{7.9}$$

hold owing to the complementary slackness conditions (7.4) and the first constraint of the dual problem. The second constraint of the dual problem implies $y_S - 1 \geq y_Q$, hence $y_S > y_Q$. Hence, the set C of nodes k with $y_k \leq y_Q$ defines a cut. Notice that y_S does not belong to C as $y_S \not\leq y_Q$ holds. (7.9) implies $x_{ij} = u_{ij}$ for every original edge (i,j) with $i \in C$, $j \notin C$. (7.8) implies $x_{ij} = 0$ for every original edge (i,j) with $i \notin C$, $j \in C$. For the cut C we obtain with (7.5) the relation

$$v = \sum_{j\in C, k\notin C} u_{jk}.$$

On the other hand, (7.6) holds for an arbitrary cut. Thus, C is a cut with minimal capacity. As x is a maximum flow the assertion follows. □

Theorem 7.3.8

If every finite capacity u_{ij} is a positive integer and there exists a maximum flow, then there exists an integral maximum flow.

7.4 Algorithm of Ford and Fulkerson (Augmenting Path Method)

The algorithm of Ford and Fulkerson is motivated by the primal-dual algorithm. It turns out that the dual of the restricted primal problem corresponds to finding a so-called augmenting path from Q to S. The primal-dual algorithm then aims at increasing the volume of the flow by adapting the current flow such that it meets either a lower or an upper capacity constraint on this path.

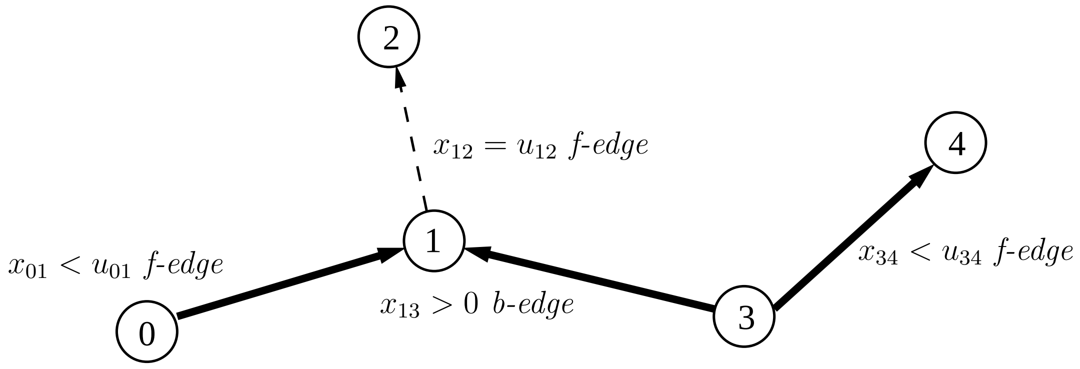
Definition 7.4.1 **(useable path, augmenting path)**

Let x be a feasible flow with volume v.

- *Let a $v_0 - v_r-$path $[v_0, v_1, \ldots, v_r]$ in the underlying undirected network with nodes $v_0, v_1, \ldots, v_r$ be given. The path is called useable, if $x_{ij} < u_{ij}$ holds for every forward* *edge and $x_{ij} > 0$ for very backward edge. Herein, an edge (i,j) is called forward edge (f-edge), if $i = v_k$ and $j = v_{k+1}$ hold for some k. Similarly, an edge (i,j) is called backward arc (b-edge), if $i = v_{k+1}$ and $j = v_k$ hold for some k.*

- *An useable $Q-S-$path is called augmenting path.*

Example 7.4.2 **(Useable path)**



The idea of the algorithm of Ford and Fulkerson is to create augmenting paths. It is exploited that a flow is maximal if and only if no augmenting path with useable edges exists.

Algorithm 7.4.3 **(Ford and Fulkerson (Augmenting Path Method))**

(0) Let a feasible flow x be given (e.g. $x_{ij} = 0$ for every $(i,j) \in E$).

(1) Find an augmenting path $W = [Q = v_0, v_1, \ldots, v_r = S]$. If no augmenting path exists, STOP: x is a maximum flow.

(2) Compute the flow $\bar{x}$ according to

$$\bar{x}_{ij} = \begin{cases} x_{ij} + d, & \text{if } (i,j) \text{ is an f-edge in the path } W, \\ x_{ij} - d, & \text{if } (i,j) \text{ is a b-edge in the path } W, \\ x_{ij}, & \text{if } (i,j) \text{ does not appear in the path } W. \end{cases}$$

Herein, d is defined as

$$d = \min\{d_v, d_r\}$$

with

$$\begin{aligned} d_v &= \min\{u_{ij} - x_{ij} \mid (i,j) \text{ is an f-edge in the path } W\}, \\ d_r &= \min\{x_{ij} \mid (i,j) \text{ is a b-edge in the path } W\}. \end{aligned}$$

(3) If $d = \infty$, STOP: the problem is unbounded.

(4) Goto (1) with x replaced by $\bar{x}$.

It holds:

- As the path W is augmenting it holds $d > 0$.
- $\bar{x}$ is a feasible flow because the conservation equations still hold at inner nodes and d is chosen such that $0 \leq x_{ij} \leq x_{ij} + d \leq u_{ij}$ (f-edges) and $u_{ij} \geq x_{ij} \geq x_{ij} - d \geq 0$ (b-edges) hold.
- Let $\bar{v}$ be the volume of $\bar{x}$ and v the volume of x. Then $\bar{v} = v + d > v$, because

$$\bar{v} = \sum_{(Q,j)\in E} \bar{x}_{Qj} = (x_{Qv_1} + d) + \sum_{(Q,j)\in E, j\neq v_1} x_{Qj} = v + d \overset{d>0}{>} v.$$

It remains to clarify how an augmenting path can be constructed. Ford and Fulkerson developed a labeling algorithm which divides the nodes in 'labeled' and 'unlabeled'. Let the set C denote the set of labeled nodes. The labeled nodes are once more divided into 'explored' and 'not explored'.
Exploring a labeled node $i \in C$ means the following:

- Consider every edge (i, j). If the conditions $x_{ij} < u_{ij}$ and $j \notin C$ are satisfied, add node j to C.
- Consider every edge (j, i). If the conditions $x_{ji} > 0$ and $j \notin C$ are satisfied, add node j to C.

The following algorithm constructs an augmenting path needed in step (1) of the Algorithm of Ford and Fulkerson.

Algorithm 7.4.4 (Labeling Algorithm of Ford and Fulkerson)

(0) Label the source Q and set $C = \{Q\}$. The remaining nodes are unlabeled.

(1) If all labeled nodes have been explored, STOP. Otherwise choose a labeled node $i \in C$ which is not explored.

(2) Explore i. If the sink S has been labeled, STOP. Otherwise go to step (1).

The labeling algorithm stops if either an augmenting path is found (the sink is labeled) or no useable path from the source to the sink exists. In the latter case the set C of all labeled nodes defines a cut, because $Q \in C$ and $S \notin C$.

The cut C by construction has the following properties:

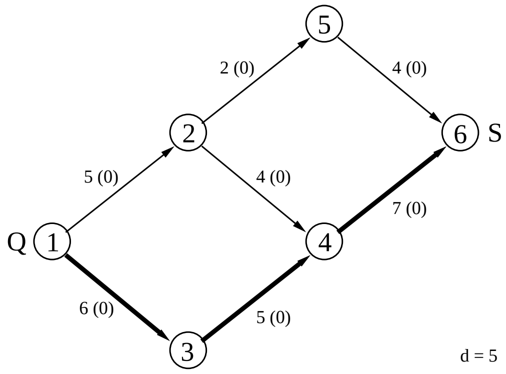
$$\begin{aligned} x_{jk} &= u_{jk}, && \text{for every edge } (j,k) \text{ with } j \in C,\ k \notin C, \\ x_{ij} &= 0, && \text{for every edge } (i,j) \text{ with } i \notin C,\ j \in C. \end{aligned}$$

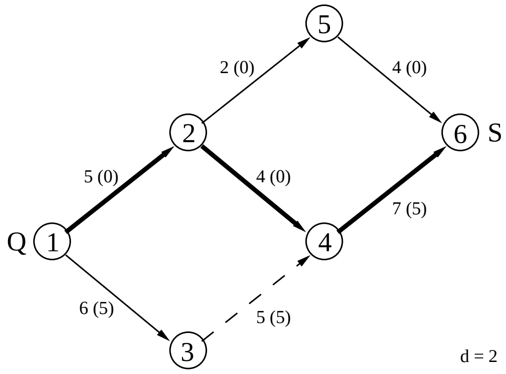
Particularly, C is a cut of minimal capacity, because with (7.5) we find

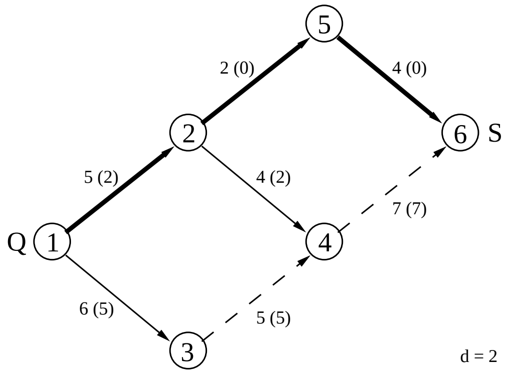
$$v = \sum_{j\in C, k\notin C} \underbrace{x_{jk}}_{=u_{jk}} - \sum_{i\notin C, j\in C} \underbrace{x_{ij}}_{=0} = \sum_{j\in C, k\notin C} u_{jk}.$$

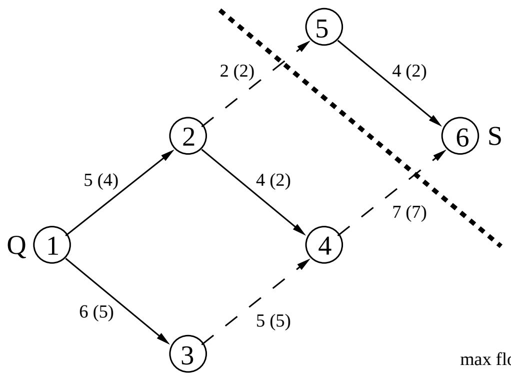
It follows that x is a flow with maximum volume v, because v is always bounded by the capacity of a cut.

Example 7.4.5 **(Passenger Scheduling)**
As many passengers as possible shall be transported from Q to S. As direct flights are not available anymore, flights via alternative cities have to be considered. For the respective connections only a limited number of free seats are available.









max flow = 4+5=9

min cut = 2+7=9

C={1,2,3,4} defines a cut of minimal capacity. It is the result of the labeling procedure.

7.4.1 Finiteness and Complexity

So far, we haven't specified in which order the labeled nodes should be investigated. This flaw may lead to cycles in the algorithm of Ford and Fulkerson and in this case the algorithm does not terminate. Fortunately, there are certain conditions that guarantee finite termination.

Theorem 7.4.6

Under the assumptions

- *Every finite u_{ij} is a positive integer.*
- *At least one cut C possesses a finite capacity M.*
- *The flow, which is used to initialise the algorithm, is integral.*

the algorithm of Ford and Fulkerson terminates after at most $M + 1$ steps.

Proof: Let the initial flow be integral. Then in each step an integral flow is generated because $d > 0$ is integral. In particular, the volume of the flow is increased by at least 1 in each iteration. The flow at the beginning of iteration k has at least volume $k - 1$. On the other hand, the volume of every feasible flow is bounded by the finite capacity M of the cut. Hence, the k-th iteration takes place only if $k - 1 \leq M$, i.e. the algorithm terminates after at most $M + 1$ iterations. $\square$

Remark 7.4.7

- *The algorithm may fail if at least one of the above conditions fails to hold.*
- *A variant of the algorithm of Ford and Fulkerson exists which terminates even for infinite and/or non-integral capacity bounds. This version considers the labeled nodes according to the first-labeled, first scanned principle, compare Edmonds and Karp (1972). This version determines a maximum flow in at most $nm/2$ iterations and has the total complexity $\mathcal{O}(m^2 n)$, see Korte and Vygen [KV08], Th. 8.14, Cor. 8.15, page 173.*

Chapter A

Software

Available software in the world wide web:

- Scilab: A Free Scientific Software Package; http://www.scilab.org
- GNU Octave: A high-level language, primarily intended for numerical computations; http://www.octave.org/octave.html
- GAMS: Guide to Available Mathematical Software; http://gams.nist.gov/
- Netlib: collection of mathematical software, papers, and databases; http://www.netlib.org/
- Decision Tree for Optimization Software; http://plato.la.asu.edu/guide.html
- NEOS Guide: www-fp.mcs.anl.gov/otc/Guide
- GeoGebra: www.geogebra.org

Recommended Literature for Operations Research:

- Bomze, I. M. and Grossmann, W. *Optimierung - Theorie und Algorithmen*. BI-Wissenschaftsverlag, Mannheim, 1993.
- Dempe, S. and Schreier, H. *Operations research. Deterministische Modelle und Methoden.*. Teubner Studienbücher Wirtschaftsmathematik. Wiesbaden: Teubner., 2006.
- Neumann, K. and Morlock, M. *Operations Research*. Carl Hanser Verlag, München Wien, 2002.
- Many examples can be found in the following book:

  Winston, W. L. *Operations Research: Applications and Algorithms*. Brooks/Cole–Thomson Learning, Belmont, 4th edition, 2004.
- A script on Mathematical Methods of Operations Research by M. Gerdts and F. Lempio can be downloaded from

  http://www.math.uni-bayreuth.de/∼flempio/lectures/OR/Mat.pdf

Literature with focus on Combinatorial Optimisation:

- Korte, B. and Vygen, J. *Combinatorial Optimization – Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer Berlin Heidelberg, fourth edition edition, 2008.
- Papadimitriou, C. H. and Steiglitz, K. *Combinatorial Optimization–Algorithms and Complexity*. Dover Publications, 1998.

Literature with focus on Integer Programming:

- Wolsey, L. A. *Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1998.

Bibliography

[Bla77] Bland, R. G. *New finite pivoting rules for the simplex method*. Mathematics of Operations Research, 2 (2); 103–107, 1977.

[Bor87] Borgwardt, K. H. *The simplex method – A probabilistic analysis*. Springer, Berlin-Heidelberg-New York, 1987.

[Chv83] Chvatal, V. *Linear Programming*. W. H. Freeman and Comp., New York, 1983.

[Dan98] Dantzig, G. B. *Linear Programming and Extensions*. Princeton University Press, new ed edition, 1998.

[DS06] Dempe, S. and Schreier, H. *Operations research. Deterministische Modelle und Methoden.*. Teubner Studienbücher Wirtschaftsmathematik. Wiesbaden: Teubner., 2006.

[Gas03] Gass, S. *Linear Programming – Methods and Applications*. Dover Publications, 5th edition, 2003.

[GL06] Gerdts, M. and Lempio, F. *Mathematische Methoden des Operations Research*. Lecture Note, http://www.uni-bayreuth.de/departments/math/∼flempio/lectures/OR/Mat.pdf, Universität Hamburg, Universität Bayreuth, 2006.

[KM72] Klee, V. and Minty, G. J. *How good is the simplex algorithm?*. In *Inequalities III* (O. Shisha, editor), pp. 159–175. Academic Press, New York, 1972.

[KV08] Korte, B. and Vygen, J. *Combinatorial Optimization – Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer Berlin Heidelberg, fourth edition edition, 2008.

[MS69] Marshall, K. T. and Suurballe, J. W. *A note on cycling in the simplex method*. Naval Res. Logist. Quart., 16; 121–137, 1969.

[NW99] Nocedal, J. and Wright, S. J. *Numerical optimization*. Springer Series in Operations Research, New York, 1999.

[PS98] Papadimitriou, C. H. and Steiglitz, K. *Combinatorial Optimization–Algorithms and Complexity*. Dover Publications, 1998.

[Win04] Winston, W. L. *Operations Research: Applications and Algorithms*. Brooks/Cole–Thomson Learning, Belmont, 4th edition, 2004.