Department of Computer Science
and Software Engineering

COSC 364
Network Flow and Capacity Planning

Andreas Willig
andreas.willig@canterbury.ac.nz

Version: 1.2 (24 May 2017)

Contents

1 Administrative Matters **4**
- 1.1 How to use the Booklets 4
- 1.2 General Questions and Answers 5
- 1.3 Typographic Conventions 6
- 1.4 Overview and Timetable 7
- 1.5 Timetable 7
- 1.6 Resources and References 7

PartI Theory

2 Introduction to Network Flow Planning **9**
- 2.1 A Single-Commodity Scenario 11
  - 2.1.1 Minimum Cost Routing 12
  - 2.1.2 Load Balancing 14
  - 2.1.3 Average Delay 15
  - 2.1.4 A First Comparison 16
- 2.2 A Multi-Commodity Scenario 16
  - 2.2.1 Minimum Cost Routing 17
  - 2.2.2 Load Balancing 17
  - 2.2.3 Average Delay 19
- 2.3 Summary 21

3 Optimization – The Big Picture **22**
- 3.1 Optimization Problems 22
- 3.2 Classes of Optimization Problems 23
- 3.3 Nonlinear Programming 25
- 3.4 Convex Programming 25
- 3.5 Linear Programming 27
- 3.6 Integer Linear Programming 28

4 Linear Optimization **30**
- 4.1 Examples 31
  - 4.1.1 The Diet Problem 32
  - 4.1.2 Wireless Communications 32
  - 4.1.3 A Transportation Problem 33
- 4.2 Reduction to Standard Form 34
  - 4.2.1 Case 1: “Greater-Than” Constraints 34
  - 4.2.2 Case 2: “Smaller-Than” Constraints 35
  - 4.2.3 Case 3: Free Decision Variables 36
  - 4.2.4 Reduction Exercises 37
- 4.3 Excursion: Geometric View 38
- 4.4 Basic Solutions and Basic Feasible Solutions 41
  - 4.4.1 Fundamental Theorem of Linear Programming 42
  - 4.4.2 Excursion: Convex Polytopes and Basic Feasible Solutions 44
- 4.5 The Simplex Algorithm 46
  - 4.5.1 Tableaus and Pivoting 46
  - 4.5.2 Which Basis Element to Remove from Basis? 50

4.5.3 Which New Non-Basis Element to Bring into Basis? 51
4.5.4 Finding the First Basic Feasible Solution 53
4.5.5 Summary of Simplex Algorithm 54
4.5.6 Example 56

5 Network Planning Problems 59
5.1 Setup and Notations 59
5.1.1 Demand Volumes 59
5.1.2 Paths and Links 60
5.1.3 Revisiting Network Flow Problems 61
5.2 Further Network Flow Problems 64
5.2.1 Capacity Design 65
5.2.2 Unsplittable Flows 65
5.2.3 Influencing Routing 66
5.3 Topology Design and Related Problems 68
5.3.1 Placement of Transit Routers 69
5.3.2 Placement and Interconnection of Transit Routers 70
5.3.3 Adding Traffic Demands 70
5.4 Comments 73

6 Excursion: Link-Weight Determination for Shortest-Path Routing 74
6.1 Duality 74
6.1.1 Finding the Dual Problem 75
6.1.2 Interpretation as Prices 76
6.1.3 Duality Theorems 77
6.2 Link-Weight Determination 82
6.2.1 Impact of Shortest-Path Routing 83
6.2.2 Problem Formulation 84
6.2.3 Duality-Based Link-Weight Determination 85

PartII Labs

7 Labs 90
7.1 The CPLEX Solver 90
7.1.1 Installation 90
7.1.2 Interactive Usage 92
7.1.3 Batch Usage 93
7.1.4 LP File Format 94
7.2 First Steps with CPLEX 95
7.3 Manually Solving LPs 96
7.4 Further Problems 104

PartIII Appendices

A Vectors and Matrices 107
A.1 Vectors and the $\mathbb{R}^n$ 107
A.2 Linear Independence, Dimension and Basis 110
A.3 Matrices 111

Chapter 1

Administrative Matters

1.1 How to use the Booklets

The course COSC 364 will be sub-divided into a number of modules, these are:

- Module 0: Course organization [0.5 weeks]
- Module 1 - **IPv4 and Routers**: IPv4 refresher, router architectures [2.5 weeks]
- Module 2 - **Routing**: Internet routing (AS, OSPF, BGP) [4 weeks]
- Module 3 - **Planning**: Optimization, flow and network planning [5 weeks]

For the last two modules (Routing and Planning) there are separate **booklets**. The booklets are designed to contain **all** material relevant for the respective module. This includes as a minimum the relevant theory, lab materials, and a set of problems and review questions.

The booklets play a key role in the course:

- It is essential that you **read the booklet before the module starts**. In particular, you need to read (and understand!) the theory part of it and make a first attempt to work through the problems and review questions. We will **not** go through the entire theory in class. You also need to read the lab materials before you start working in the lab. Note that some preparatory lab problems should be done **before** the lab.
- We will not go through the theory during the lectures in detail. Instead we will use the lectures to do one or more of the following:
  - Review the theoretical part at a high level.
  - Discuss selected issues in some more depth.
  - Work through examples and selected exercises / review questions.
  - Discuss your questions and provide clarifications.

This is the second year the booklets are available, and therefore they are probably still a bit rough around the edges. All errors are mine, but I really would appreciate if you can point out any mistakes, typos, omissions, unclear points and the like. The booklets will be versioned, the version is shown on the cover page. From time to time I will publish new versions in which a bulk of errors have been corrected and some parts may have been revised/added. When writing me about an error, please let me know the version you are referring to and the relevant page numbers (if any).

1.2 General Questions and Answers

Question: Why are there three different types of problems?

Answer:

In the text you will find three different types of problems:

- Normal problems (or just problems) are part of the main text and need to be worked on by you. Some of these will be discussed in the lectures, others will not. All of these problems (or variations thereof) can occur in a test or exam.
- Bonus problems are also part of the main text, but you do not need to do them and they will not occur in a test or exam.
- Review problems occur typically in the labs and typically ask you to read some bits of documentation or a `man` page. You need to do these problems but they will not occur in a test or exam.

These types of problems are marked differently in the text (see also Section 1.3).

Question: What parts of the theory can come up in the exam/test?

Answer:

Everything from the theory part that is not marked as optional / excursion, including the problems (but not the bonus problems or review problems) and examples. The contents of the appendices will normally not be covered directly in the exam, but they provide pre-requisite material that is needed to understand the contents and which possibly not all people have on top of their head.

Question: What parts of the labs can come up in the exam/test?

Answer:

Again, more or less everything that is not marked as optional / excursion, including examples and most of the problems, but not the review problems.

Question: Will the exam/test be completely based on the booklets?

Answer:

You should not expect that **all** the problems in the exam/test will have shown up in the booklets directly or in a similar way. Many of the questions actually will, but some percentage of exam/test questions will also require some transfer of concepts or even creative thinking.

1.3 Typographic Conventions

Problems are an integral part of the text and are relevant for tests / exams. They appear like this:

Problem 1.3.1 (A not-so-hard problem)**.**
Disprove that all graphs are 3-colorable.

A particular type of problem are review problems. You need to do these as well (because they usually ask you to familiarize yourself with some important material), but they will not be asked in the exam. A classical example problem of this type is to read a certain `man` page. Review problems look like this:

Review problem 1.3.1 (Read tons of `man` pages)**.**
Read the `man` pages for the following commands: `find`, `ls`, `grep`, `awk`, `sed`, ...

There are also bonus problems, which, while interesting, are not important for the main part and will not feature in tests / exams. They look like this:

Bonus problem 1.3.1 (A harder problem)**.**
Prove that $P \neq NP$.

Sometimes I will add information that is not central to the ongoing discussion. Such an excursion (or digression) looks as follows:

Excursion (Dramatic food crisis)
A bag of rice has fallen over in China. Film at 11.

and can be completely ignored.

1.4 Overview and Timetable

1.5 Timetable

Week	Lecture topic	Lab
1	Introduction to network planning problems Introduction to optimization	Introducing CPLEX
2	Linear programming: example problems LP: simplex algorithm	Formulating and solving LPs
3	Simplex algorithm Network flow problems	Network flow problems
4	Routing and capacity planning Routing and capacity planning	Network flow problems

1.6 Resources and References

This booklet contains all that you need to know for this course. I recommend the following books and resources:

- General optimization: [6]
- Some texts on combinatorial / discrete optimization are [11], [15]
- Linear programming: [17], [18], [13]
- Convex analysis and optimization: [3], [5]
- Network and flow optimization: [1]

Part I

Theory

Chapter 2

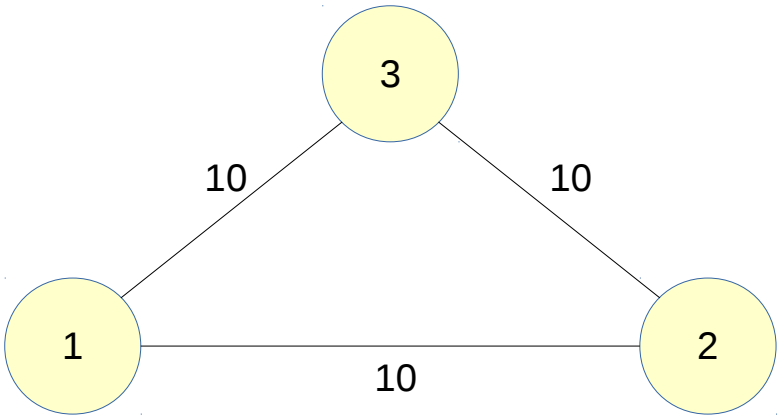
Introduction to Network Flow Planning

In this chapter we give an introduction to network flow planning problems. It is largely based on [14].

When planning, deploying and operating a network, an ISP or network operator has to balance a number of conflicting requirements, including the desire of users to enjoy high data rates and low delay, and its own desire to keep capital or operational cost low and to utilize the own equipment well. An operator has to make decisions on different time horizons, for example:

- In the long term (several months, years) a network operator needs to make investment decisions. For example, it needs to decide how much new capacity (new cables, routers, etc) should be added to its network to satisfy growing customer needs, and where exactly this new capacity is needed. As an analogy, when an airline observes that a lot of people want to go from Christchurch to Alice Springs, it might contemplate establishing a direct connection between these two cities, or it might decide that existing routes (e.g. via Sydney) are sufficient.
- In the medium term (weeks, months) and as long as the physical topology of a network does not change, an operator can make decisions about how to route traffic flow between a given source and destination. In an OSPF context, the routing of traffic is affected by the cost values assigned to individual links. To again use an airline example, suppose Air New Zealand has two different ways to reach Perth: a direct flight (operated by Air NZ) and an indirect flight via Alice Springs which is not well utilized. By changing the pricing of these two options it is possible to shift more customers to the route via Alice Springs.

In this unit we develop simple models for this sort of questions. For simplicity, most of the time we consider networks that can be modeled as undirected graphs, an example is shown in the following figure:



The nodes in the network correspond to routers, and the edges of the network correspond to transmission links. In this particular example the links are labeled with numbers, and these numbers indicate their available transmission capacity in some given unit (e.g. Mb/s, Gb/s or Tb/s). We call such a network a **capacitated network** and the link capacities actually provide constraints on the total amount of traffic that can be put on a link. Working with capacitated networks corresponds to a situation where the link capacities have already been decided. In the case of **uncapacitated networks** the capacity of a link has not yet been decided and one is often interested in finding a good value for this capacity (e.g. how many seats should the airplane between Christchurch and Alice Springs really have?). In this chapter, however, we focus on capacitated networks.

The routers generate traffic destined to other routers.[1] For a particular source and destination router the generated traffic is also known as a **demand volume**. The demand volume is usually regarded as the long-term average amount of traffic generated by the source per unit time, i.e. it has the dimensions of a rate. For the kind of questions we want to address in this booklet we treat the actual traffic generated by the source router as constant bit-rate traffic, i.e. it does not change essentially over a reasonably large time span and we do not account for short-term transients. It is helpful to imagine that an individual source is something like an AS border router rather than a router very close to end hosts. An AS border router will forward the aggregate traffic of many hosts inside an AS to another AS, and the resulting aggregate data rate is often much more stable over time than the data rate generated by an individual host.

By making the right routing decisions (in the OSPF context: by assigning the right cost values to links), an operator seeks to distribute the existing demand volumes over its network to avoid overloading individual links beyond their capacity, to achieve a balanced utilization of resources (links, routers), or to satisfy customer demands for rate or delay. Note that we only treat links with their limited capacity as the bottlenecks and assume that the processing speed of routers is sufficient for all our purposes.

For a given demand volume between two nodes in the network it may be necessary to split this volume up to use several paths in parallel. Suppose the set of available paths between source and destination is $\mathcal{P} = \{P_1, \ldots, P_k\}$. A **path flow** for a path $P_i \in \mathcal{P}$ refers to the part of the demand volume that is carried on P_i, and a **link flow** is the part of a demand volumes traffic that is carried over an individual link. The pair of nodes, i and j, involved in a demand volume is also called a **node pair**, and we write this as $i : j$. A direct link between two nodes will be denoted as $i - j$.

Definition 2.1 (Traffic engineering)**.** *(Adapted from: [14])* ***Traffic engineering*** *refers to the best way (w.r.t. some objective function) to distribute the demand volume in a capacitated network.*

[1] Clearly, the "destination router" will then forward the packets to their ultimate destination hosts, but we completely ignore end hosts in this booklet.

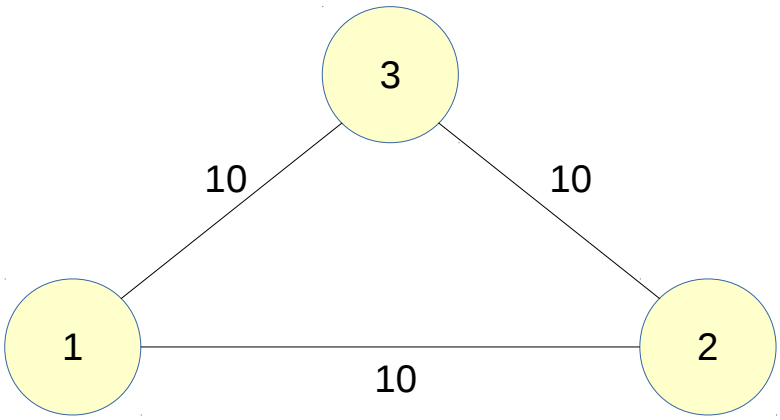
Problem 2.0.1 (Splitting up traffic)**.**
Splitting up demand volumes across several paths might be necessary to actually satisfy this volume, but some (versions of some) protocols might not operate satisfactorily when different paths are used in parallel.
For which transport layer protocol might this be a difficult thing to do and why?

In the remainder of this chapter I will give you a first taste of the problems we are interested in, and in particular I will demonstrate that all these problems can be formulated as optimization problems (which explains why optimization plays such a role in this module). We will become much more systematic later in this booklet.

2.1 A Single-Commodity Scenario

In single-commodity problems (commodity is another word for demand volume in our context) there is only a single demand volume in the network, which flows from some source node to some destination node.

Consider again the three node network in this figure:



There is a single demand volume from node 1 to node 2 of rate h (expressed in the same units as the link capacities). In general, the links can have different capacities, and the capacity of a path is given by the minimum capacity of all the links involved in that path. In this example there are two paths between nodes 1 and 2, the first one being the single link path $1-2$, and the second one being the two-link path $1-3-2$. So, the capacity available on the two different paths can be different as well, and we denote these capacities by c_{12} and c_{132}, respectively. We regard the values of c_{12}, c_{132} and h as parameters of the problem, they are in general given to us.

Having two different paths at our disposal, we now have to decide which fraction of the overall demand volume h to put on path $1-2$, and which fraction to put on the other path $1-3-2$. These fractions are denoted as x_{12} and x_{132}, respectively, and represent the path flows for these two paths. These two variables are our **decision variables**, and we can already state some constraints on them:

- **Demand constraint**: the path flows must add up to give the total demand volume, i.e. we must have:

$$x_{12} + x_{132} = h$$

- **Non-negativity constraints**: clearly the path flows x_{12} and x_{132} must be non-negative, as negative data rates make no sense. In other words, the following constraint inequalities hold:

$$\begin{aligned} x_{12} &\geq 0 \\ x_{132} &\geq 0 \end{aligned}$$

- **Capacity constraints**: each path in our network has a maximum capacity, or a maximum data rate it can carry, and clearly a path flow (or its data rate) should not exceed that capacity. In general, for a path $a - b - c - \ldots - z$ the path capacity is the minimum of all the link capacities for links $a - b$, $b - c$, and so on. Hence we arrive at the following two constraint inequalities:

$$\begin{aligned} x_{12} &\leq c_{12} \\ x_{132} &\leq c_{132} \end{aligned}$$

Note that our constraints include equations and inequalities. Note furthermore that so far our constraint equations/inequalities are all **linear**, i.e. they only involve sums of terms made up of the decision variables themselves (perhaps multiplied by some constant value) and given parameters (which we regard as constants).

Problem 2.1.1 (How many solutions?)**.**
Suppose that $c_{12} = c_{132} = c = 10$ and $h = 7$. How many different solutions to the decision problem that satisfy all constraints simultaneously (such a solution is called a **feasible solution**) are there? In other words: how many different assignments to the decision variables x_{12}, x_{132} exist which satisfy the constraints?

2.1.1 Minimum Cost Routing

Now imagine that we do not only have capacities associated to links, but that we additionally have some notion of **cost** associated to either links or entire paths. For simplicity here we assume that a cost is associated to an entire path: path $1 - 2$ has cost ϕ_{12}, and path $1 - 3 - 2$ has cost ϕ_{132}, where the cost values are non-negative numbers and express the cost of transporting one unit of traffic over the given path. Then we want to find values for our decision variables x_{12} and x_{132} which **minimize** the total cost for serving the demand volume. We express this total cost through the following **objective function**:

$$F = f(x_{12}, x_{132}) = \phi_{12}x_{12} + \phi_{132}x_{132} \tag{2.1}$$

which is a linear function in both arguments.[2] Now we have all the ingredients to formulate the single-commodity minimum-cost flow planning problem for our network:

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \phi_{12}x_{12} + \phi_{132}x_{132} \\ \text{subject to} \quad & x_{12} + x_{132} = h \\ & x_{12} \geq 0 \\ & x_{132} \geq 0 \\ & x_{12} \leq c_{12} \\ & x_{132} \leq c_{132} \end{aligned} \tag{2.2}$$

[2]Strictly speaking, a function $f(x)$ is called **linear** if for any x and y from the domain of the function and any real numbers λ and μ it holds that $f(\lambda x + \mu y) = \lambda f(x) + \mu f(y)$ holds. It immediately follows that $f(0) = 0$ holds. In this sense the objective function 2.1 is **not** linear in, say, the first argument x_{12}, but rather is what is known as **affine-linear**. A function $f(x)$ is called affine-linear when it can be written as the sum of a strictly linear function $g(x)$ and a constant c, i.e. $f(x) = g(x) + c$. In this booklet I will freely use the term "linear" to mean both strictly linear and affine-linear functions.

Note that here the vector $\mathbf{x} = (x_{12}, x_{132})$ collects all the decision variables. Obviously, the problem consists of an objective function, which depends on the decision variables and which we want to minimize or maximize by finding the right values for the decision variables. The second main part of the problems are the constraints, which describe which conditions the decision variables must fulfill to be feasible.

This is the general form of an **optimization problem**, in which we aim to minimize the value of the objective function for all feasible solutions (x_{12}, x_{132}), where a feasible solution is a solution which satisfies all the constraints simultaneously. This particular problem is actually a rather special type of optimization problem: it is a **linear optimization problem** or a **linear program** (LP). For linear programs it is very well established how they can be solved, and highly tuned algorithms and software packages exist for solving them on a computer. We will work with one of these, the CPLEX package by IBM (see Section 7.1). We will learn more about optimization in Chapter 3, and we will see how to solve linear programs in Chapter 4.

Problem 2.1.2 (Optimal solution)**.**
Solve problem 2.2 for $c_{12} = c_{132} = c = 10$, $\phi_{12} > \phi_{132} > 0$ and assuming three different values for h:

- $h = 5$
- $h = 15$
- $h = 25$

We will later on establish and solve a range of linear models using the CPLEX solver by IBM. An input file (called `tm.lp`) with the model specification of one particular instance of the problem (in which we assume $\phi_{12} = 5$, $\phi_{132} = 12$, $h = 17$, and the capacity of path $1-2$ is $c_{12} = 10$, whereas the capacity of path $1-3-2$ is $c_{132} = 12$) is shown here:

```
Minimize
  5 x12 + 12 x132
Subject to
  demandflow:  x12 + x132 = 17
  capp1:       x12  <= 10
  capp2:       x132 <= 12
Bounds
  0 <= x12
  0 <= x132
End
```

Note that this model description (written in a particular syntax called the LP format, see Section 7.1.4) is relatively close to the way we have written the mathematical model above. When you start the command-line version of the CPLEX solver (see Section 7.1 on how to install and run CPLEX) then you could get a solution by using the following dialogue:

```
CPLEX> read tm.lp
Problem 'tm.lp' read.
Read time = 0.00 sec. (0.00 ticks)
CPLEX> optimize
Tried aggregator 1 time.
LP Presolve eliminated 3 rows and 2 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec. (0.00 ticks)
```

```
Dual simplex - Optimal:  Objective =  1.3400000000e+02
Solution time =    0.00 sec.  Iterations = 0 (0)
Deterministic time = 0.00 ticks  (3.08 ticks/sec)

CPLEX> display solution variables -
Variable Name           Solution Value
x12                          10.000000
x132                          7.000000
CPLEX>
```

where the user has entered the commands after the `CPLEX>` prompt, followed by CPLEX' output. Does this solution correspond to what you would get in your solution to Problem 2.1.2?

2.1.2 Load Balancing

We now establish another problem formulation by exchanging the objective function. We again consider our three-node network, but now our aim is to balance load properly, or more precisely, to make sure that the path / link utilization is the same on both available paths.

Denoting the decision variables again by x_{12} and x_{132}, the path utilization along both available paths can be expressed as

$$\frac{x_{12}}{c_{12}} \quad , \quad \frac{x_{132}}{c_{132}}$$

and a suitable objective function is given by

$$\max\left\{\frac{x_{12}}{c_{12}}, \frac{x_{132}}{c_{132}}\right\}$$

so the resulting problem becomes:

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \max\left\{\frac{x_{12}}{c_{12}}, \frac{x_{132}}{c_{132}}\right\} \\ \text{subject to} \quad & x_{12} + x_{132} = h \\ & x_{12} \geq 0 \\ & x_{132} \geq 0 \\ & x_{12} \leq c_{12} \\ & x_{132} \leq c_{132} \end{aligned} \tag{2.3}$$

Problem 2.1.3 (Understanding the formulation)**.**
Make sure you understand why this problem formulation (in particular the objective) makes sense for load balancing.

Problem 2.1.4 (Solving the problem)**.**
For a given $h > 0$ and assuming that $c_{12} = c_{132} = c > 0$, solve the load balancing problem and formulate a condition on h which guarantees feasibility / solvability of the problem.

This problem formulation has a disadvantage, though: the objective function appears not to be a linear function, and therefore we cannot immediately apply LP solvers like CPLEX. We will see in Section 2.2.2, however, that we actually can give an equivalent problem with a linear objective function.

2.1.3 Average Delay

If we want to minimize the average delay, then we need some idea how we can calculate the average delay on a path. For starters, each link along a path incurs its own delay and the delay of a path is the sum of the delays along all the links of this path.

There are several factors contributing to the delay on a link, including the propagation delay, the bitrate of the link, router processing times, and the delay introduced by queueing packets in the router output buffers for the link in question.[3] Here we are only concerned with the queueing delay component.

There is a mathematical theory called queueing theory [9],[10], [4] which (under very specific assumptions) gives some insight into the queueing delay, measuring the amount of time a packet spends waiting in a queue. The results from queueing theory suggest that the average delay on a link, say link $1-2$, depends on the flow volume x_{12} put on the path of which link $1-2$ is a component, and on this link's capacity in the following way:

$$d_{12} = \frac{x_{12}}{c_{12} - x_{12}} \tag{2.4}$$

which, amongst others, says that as the flow volume approaches the capacity (from below), the average delay approaches infinity.

One way to formulate the resulting problem is then given by:

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \frac{x_{12}}{c_{12} - x_{12}} + \frac{2x_{132}}{c_{132} - x_{132}} \\ \text{subject to} \quad & x_{12} + x_{132} = h \\ & x_{12} \geq 0 \\ & x_{132} \geq 0 \\ & x_{12} \leq c_{12} - \epsilon \\ & x_{132} \leq c_{132} - \epsilon \end{aligned} \tag{2.5}$$

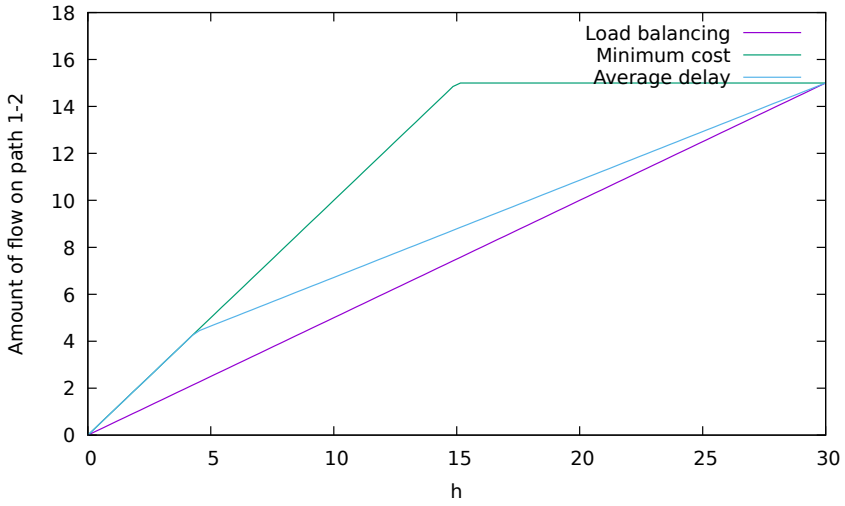
where we have assumed that both links of the path $1-3-2$ have the same transmission capacity, and where $\epsilon > 0$ is a small positive number which we use to ensure that we never evaluate the objective function for $x_{12} = c_{12}$ or $x_{132} = c_{132}$. The objective function of this problem is decidedly non-linear, but we will consider later on how we can modify the problem to fit it into the LP framework.

Bonus problem 2.1.1 (Solving the problem)**.**
Solve the average delay problem assuming $c_{12} = c_{132} = c > 0$ and $h < 2c$ by using calculus (no need to introduce some explicit ϵ). You need to calculate derivatives and the solution of quadratic equations for this problem.

[3] You can imagine that each router has a separate queue on each of its outputs, which collects all the packets destined to that output. Sometimes many packets arrive in a short interval and one particular packet might get stuck behind them in a queue.

2.1.4 A First Comparison

In the following figure we fix the capacities along both paths to $c_{12} = c_{132} = 15$, fix the path-cost for minimum-cost routing to $\phi_{12} = 1$ and $\phi_{132} = 2$, and vary the demand volume h on the x-axis. The y-axis shows the optimum amount of flow along path $1 - 2$ (i.e. the optimal decision value x_{12}) for the three objective functions introduced so far:



It clearly can be seen that the choice of objective function has a substantial impact on the routing of flows!

2.2 A Multi-Commodity Scenario

In the setting used in Section 2.1 we have used only a single demand flow between nodes 1 and 2. Now we consider again the three-node network, but there are now demand volumes between each pair of nodes:

- Demand volume h_{12} between nodes 1 and 2, which can use two different paths: $1 - 2$ and $1 - 3 - 2$.
- Demand volume h_{13} between nodes 1 and 3, which can use two different paths: $1 - 3$ and $1 - 2 - 3$.
- Demand volume h_{23} between nodes 2 and 3, which can use two different paths: $2 - 3$ and $2 - 1 - 3$.

For each of these demand volumes there are two decision variables: one for the respective single-hop path, and another one for the two-hop path, so that in total we have the following decision variables: x_{12} and x_{132} for flow h_{12}, x_{13} and x_{123} for flow h_{13}, x_{23} and x_{213} for flow h_{23}. We assume furthermore that each individual link has a given transmission capacity c_{ij} for the link between nodes i and j.

Problem 2.2.1 (Establishing constraints)**.**
Write down equations for the demand constraints and the capacity constraints.

2.2.1 Minimum Cost Routing

Problem 2.2.2 (Minimum cost routing)**.**
Assume that routing costs are specified per path (not per link), e.g. cost ϕ_{12} for path $1-2$. Write down the full minimum cost-routing problem (objective function and constraints).

2.2.2 Load Balancing

While finding the objective function for minimum cost routing should have led you to a linear objective function, we next consider the objective of load-balancing, where we want to level out the utilization on all three involved links. We introduce new non-negative auxiliary variables y_{12}, y_{13} and y_{23} to represent the amount of flow on a given link. These auxiliary variables are given by:

$$\begin{aligned} y_{12} &= x_{12} + x_{123} + x_{213} \\ y_{13} &= x_{13} + x_{132} + x_{213} \\ y_{23} &= x_{23} + x_{132} + x_{123} \end{aligned}$$

and the resulting utilization on each of the three links is given by:

$$\frac{y_{12}}{c_{12}}, \frac{y_{13}}{c_{13}}, \frac{y_{23}}{c_{23}}$$

where each of these three numbers should be between zero and one (why?). The objective function can be expressed as

$$f(y_{12}, y_{13}, y_{23}) = \max\left\{\frac{y_{12}}{c_{12}}, \frac{y_{13}}{c_{13}}, \frac{y_{23}}{c_{23}}\right\} \tag{2.6}$$

which is not a linear function (i.e. it does not depend linearly on its arguments), but which is piecewise linear in each argument, and where each argument expression in itself is a linear function in y_{ij}. We can actually "linearise" this particular objective function, so that we end up with a linear program. We can do so by introducing another auxiliary variable r, representing the value of the objective function 2.6, and by introducing new constraints, one for each of the "component functions" $\frac{y_{ij}}{c_{ij}}$ (which are actually linear functions in the variables y_*, which we introduce as additional decision variables) of the objective function:

$$\begin{aligned} r &\geq \frac{y_{12}}{c_{12}} \\ r &\geq \frac{y_{13}}{c_{13}} \\ r &\geq \frac{y_{23}}{c_{23}} \end{aligned}$$

Clearly, if these three relations are true simultaneously, then r will be just as large as the original objective function 2.6. We can then state the optimization problem as follows:

$$\begin{aligned} \text{minimize}_{[\mathbf{x},\mathbf{y},r]} \quad & r \\ \text{subject to} \quad & x_{12} + x_{132} = h_{12} \\ & x_{13} + x_{123} = h_{13} \\ & x_{23} + x_{213} = h_{23} \\ & x_{12} + x_{123} + x_{213} = y_{12} \end{aligned} \tag{2.7}$$

$$
\begin{aligned}
& x_{13} + x_{132} + x_{213} = y_{13} \\
& x_{23} + x_{132} + x_{123} = y_{23} \\
& y_{12} \leq c_{12} \\
& y_{13} \leq c_{13} \\
& y_{23} \leq c_{23} \\
& y_{12} \leq c_{12} r \\
& y_{13} \leq c_{13} r \\
& y_{23} \leq c_{23} r \\
& x_{12} \geq 0, x_{132} \geq 0 \\
& x_{13} \geq 0, x_{123} \geq 0 \\
& x_{23} \geq 0, x_{213} \geq 0 \\
& y_{12} \geq 0, y_{13} \geq 0, y_{23} \geq 0 \\
& r \geq 0
\end{aligned}
$$

where the minimization is done over decision variables $\mathbf{x}$, $\mathbf{y}$ and r. This is a linear program (you can verify this claim yourself when you have worked through Chapters 3 and 4). We will often lump together the decision variables x_* into a vector $\mathbf{x}$ and sometimes indicate the variables to optimize (decision variables) in the "minimize" statement, e.g.:

$$
\begin{aligned}
\text{minimize}_{[\mathbf{x},\mathbf{y},r]} \quad & r \\
\text{subject to} \quad & \ldots
\end{aligned}
$$

Note that linearization worked in this example because the value of the objective function $\max(\cdot,\cdot,\cdot)$ is always larger than or equal to the value assumed by each of its arguments (which are in fact linear functions). Furthermore, the solution to the modified problem with the additional auxiliary variables is in fact a solution to the original problem, which again is due to the special character of the objective function, which is piecewise-linear in its arguments.

As a last comment, note that we "pay" for the linearization by:

- Introducing new decision variables $\mathbf{y}$ and r in addition to the variables $\mathbf{x}$.
- Introducing new constraints for these new variables.

These new variables and constraints will make the problem "bigger" and in general increase the time required to numerically solve an instance of the problem.

Generic problem vs. Instance
We need to carefully distinguish between a "generic" problem formulation in which given parameters are denoted by abstract symbols (e.g. using the symbol h_{12} in Problem 2.7 to denote the demand volume between nodes 1 and 2), and a particular "instance" of the problem which we get when plugging in specific values for parameters. Optimization tools like CPLEX can only solve instances, not generic problems.

2.2.3 Average Delay

We use the minimization of the average delay as another example of how problems can sometimes be "linearised". In terms of the newly introduced link-flow variables y_{ij} (Section 2.2.2) the average delay on one individual link $i-j$ is given by:

$$\frac{y_{ij}}{c_{ij}-y_{ij}}$$

(compare Equation 2.4 from Section 2.1.3). From this our objective function, the overall average delay for the involved demand volumes, can be seen to be given by

$$f(y_{12},y_{13},y_{23}) = \frac{1}{h_{12}+h_{13}+h_{23}}\left(\frac{y_{12}}{c_{12}-y_{12}}+\frac{y_{13}}{c_{13}-y_{13}}+\frac{y_{23}}{c_{23}-y_{23}}\right)$$

It is not relevant here to fully understand the derivation of this expression, but it is rather the functional form which matters to us. This function is not piecewise linear, but is a truly non-linear function. The resulting optimization problem then becomes:

$$\begin{aligned}
\text{minimize}_{[\mathbf{x},\mathbf{y}]} \quad & \left(\frac{y_{12}}{c_{12}-y_{12}}+\frac{y_{13}}{c_{13}-y_{13}}+\frac{y_{23}}{c_{23}-y_{23}}\right) \\
\text{subject to} \quad & x_{12}+x_{132}=h_{12} \\
& x_{13}+x_{123}=h_{13} \\
& x_{23}+x_{213}=h_{23} \\
& x_{12}+x_{123}+x_{213}=y_{12} \\
& x_{13}+x_{132}+x_{213}=y_{13} \\
& x_{23}+x_{132}+x_{123}=y_{23} \\
& y_{12}\leq c_{12}-\epsilon \\
& y_{13}\leq c_{13}-\epsilon \\
& y_{23}\leq c_{23}-\epsilon \\
& x_{12}\geq 0, x_{132}\geq 0 \\
& x_{13}\geq 0, x_{123}\geq 0 \\
& x_{23}\geq 0, x_{213}\geq 0 \\
& y_{12}\geq 0, y_{13}\geq 0, y_{23}\geq 0
\end{aligned}$$

where we have left out the initial factor in the objective function (it is constant and does not matter in the minimization) and where, as before, the safety margin $\epsilon > 0$ shall prevent us from dividing by zero.

We focus on one of the terms in the objective function and simplify notation somewhat:

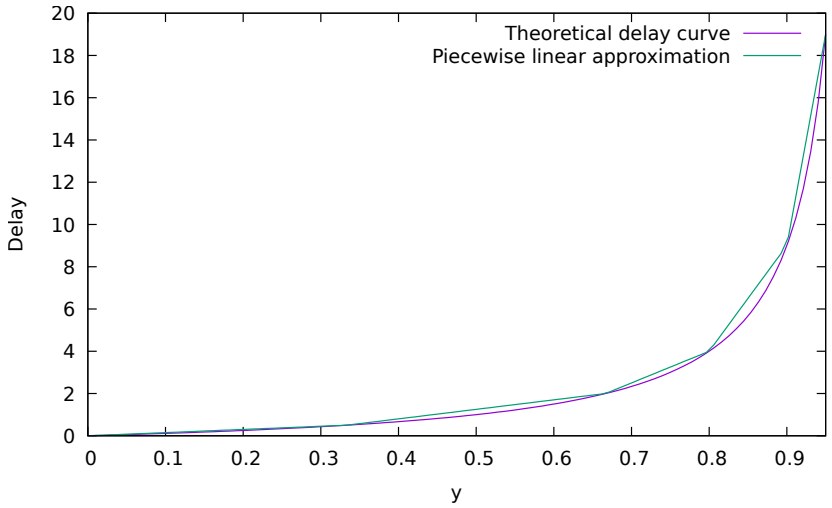
$$f(y) = \frac{y}{c-y}$$

where c is the capacity of a link and y is the total amount of flow through this link (coming from all parts of demand volumes routed through this link), $0 \leq y < c$. We want to approximate this function using a piecewise linear function. For $0 \leq y < c$ the objective function $f(\cdot)$ has a particular property called "convexity" (see Chapter 3), which guarantees that $f(\cdot)$ will remain below the piecewise linear approximation (with the possible exception of the interval closest to $\frac{y}{c}=1$). We carry out the approximation by evaluating the function for some chosen values of the argument (e.g. $y=\frac{2}{3}c$) and joining the resulting points through lines. If we carry out this

procedure then we might for example get the following approximation

$$c \cdot f(y) \approx \begin{cases} \frac{3}{2}y & : \text{ for } 0 \leq \frac{y}{c} \leq \frac{1}{3} \\ \frac{9}{2}y - c & : \text{ for } \frac{1}{3} \leq \frac{y}{c} \leq \frac{2}{3} \\ 15y - 8c & : \text{ for } \frac{2}{3} \leq \frac{y}{c} \leq \frac{4}{5} \\ 50y - 36c & : \text{ for } \frac{4}{5} \leq \frac{y}{c} \leq \frac{9}{10} \\ 200y - 171c & : \text{ for } \frac{9}{10} \leq \frac{y}{c} \leq \frac{19}{20} \\ 4000y - 3781c & : \text{ for } \frac{y}{c} \geq \frac{9}{10} \end{cases} \tag{2.8}$$

(where for convenience we have multiplied the function $f(\cdot)$ by c so that we do not have to write lots of fractions on the right-hand side). We compare the original function and its approximation for the special case of $c = 1$ in the following figure:



The key observation is now that we can write the approximating function $\tilde{f}(y) \approx c \cdot f(y)$ in another way:

$$\tilde{f}(y) = \max\left\{\frac{3}{2}y, \frac{9}{2}y - c, 15y - 8c, 50y - 36c, 200y - 171c, 4000y - 3781c\right\}$$

i.e. as the $\max(\cdot)$ of a number of functions representing straight lines. Each of these straight lines dominates the other lines (i.e. assumes values larger than the other lines) in a certain interval, which is a result of the convexity of $f(\cdot)$ and the particular approach to approximation, in which $\tilde{f}(\cdot)$ and $c \cdot f(\cdot)$ agree in chosen points.

We can now apply the same strategy for linearisation as we have done before for the load-balancing objective (which also involved a $\max(\cdot)$ operator over linear arguments): we introduce auxiliary variables r_{ij} (one for each link) and appropriate constraints, and solve the problem:

$$\begin{aligned} \text{minimize}_{[\mathbf{x},\mathbf{y},\mathbf{r}]} \quad & \frac{r_{12}}{c_{12}} + \frac{r_{13}}{c_{13}} + \frac{r_{23}}{c_{23}} \\ \text{subject to} \quad & x_{12} + x_{132} = h_{12} \\ & x_{13} + x_{123} = h_{13} \\ & x_{23} + x_{213} = h_{23} \\ & - - - - - - \\ & x_{12} + x_{123} + x_{213} = y_{12} \\ & x_{13} + x_{132} + x_{213} = y_{13} \end{aligned}$$

$$
\begin{aligned}
& x_{23} + x_{132} + x_{123} = y_{23} \\
& - - - - - - \\
& r_{12} \geq \frac{3}{2} y_{12} \, , \, r_{13} \geq \frac{3}{2} y_{13} \, , \, r_{23} \geq \frac{3}{2} y_{23} \\
& r_{12} \geq \frac{9}{2} y_{12} - c_{12} \, , \, r_{13} \geq \frac{9}{2} y_{13} - c_{13} \, , \, r_{23} \geq \frac{9}{2} y_{23} - c_{23} \\
\ldots & \\
& r_{12} \geq 4000 y_{12} - 3781 c_{12} \, , \, r_{13} \geq 4000 y_{13} - 3781 c_{13} \, , \, r_{23} \geq 4000 y_{23} - 3781 c_{23} \\
& - - - - - - \\
& x_{12} \geq 0, x_{132} \geq 0 \\
& x_{13} \geq 0, x_{123} \geq 0 \\
& x_{23} \geq 0, x_{213} \geq 0 \\
& y_{12} \geq 0, y_{13} \geq 0, y_{23} \geq 0 \\
& r_{12} \geq 0, r_{13} \geq 0, r_{23} \geq 0
\end{aligned}
$$

where we have used lines with $- - - - - --$ to group the constraints and where some of the "linearisation" constraints have been left out. The scaling of the r_{ij} in the objective function has been introduced to revert the scaling of $f(\cdot)$ by c when writing down the piecewise-linear approximation in Equation 2.8.

In this particular example we have used six different line segments in the piecewise-linear approximation (corresponding to six linear terms in the approximating function $\tilde{f}(\cdot)$), each of which creates three constraints (one for each link available). Clearly, in networks with more links or when a finer approximation is desired, the number of resulting constraints can become quite substantial and reach the hundreds, thousands or even higher numbers. LP solvers like CPLEX are capable of addressing problem sizes with this number of constraints or decision variables.

2.3 Summary

In this chapter we have introduced simple problems on partitioning demand volumes over a number of available paths under different objective functions. This sort of problems is also called **network flow problems**, and the resulting allocation of flows can then be used as input to a procedure in which, for example, we find link weights for OSPF which actually achieve these flows. We will see further problems of this and of other types in Chapters 5 and 6.

The key insight of this chapter is that the considered problems can be formulated as optimization problems, which we then try to solve. Incidentally, all the problems in this chapter have either been linear programming problems from the beginning, or could be turned into linear problems after modifying the problem (possibly involving some approximation). In the remainder of this module we will stick to linear problems, and these turn up surprisingly often in network and flow planning problems. But you should be aware that many other problems of practical relevance are decidedly non-linear and cannot reasonably be turned into linear ones at all.

Chapter 3

Optimization – The Big Picture

In this chapter we consider some global aspects of mathematical optimization (or mathematical programming, as it is often called). In the next Chapter 4 we will consider the particular case of linear programming.

3.1 Optimization Problems

A relatively generic setup for optimization can be stated as follows: We are given some subset Ω of the real n-dimensional vector space $\mathbb{R}^n$ (a review of vectors and matrices is given in Appendix A) and we are given a function $f : \Omega \subset \mathbb{R}^n \mapsto \mathbb{R}, \mathbf{x} \mapsto f(\mathbf{x})$, which is called the **target function** or **objective function** or **cost function**. Then we can pose the following problem:

$$\begin{array}{rl} \text{minimize}_{[\mathbf{x}]} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \Omega \end{array}$$

where we want to find a vector $\mathbf{x}$ which satisfies the constraint $\mathbf{x} \in \Omega$ and which among all other vectors satisfying the constraints assumes the smallest possible value. When $\Omega = \mathbb{R}^n$ then we have an **unconstrained problem**, otherwise when Ω is a strict subset of $\mathbb{R}^n$ then we have a **constrained problem**. When actually $\Omega = \emptyset$ then the problem as a whole is called **infeasible**. The components of the vector $\mathbf{x} = (x_1, \ldots, x_n)^T$ are called the **decision variables**, we need to determine values for these components which minimize the target function.

Here we have posed the optimization problem as a minimization problem. If we want to maximize the objective function $f(\cdot)$, then we can do this with the above problem formulation by minimizing the function $-f(\cdot)$. We will therefore restrict ourselves to minimization problems most of the time.

Definition 3.1 (Global minimizer)**.** *A point $\mathbf{x}^* \in \Omega$ is called a **global minimizer**, if*

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \tag{3.1}$$

*holds for all $\mathbf{x} \in \Omega$, i.e. if there exists no point in Ω where $f(\cdot)$ assumes a truly smaller value. The argument point $\mathbf{x}^*$ is the **minimizer**, and the value of the target function $f(\cdot)$ at this point, $f(\mathbf{x}^*)$, is called the **global minimum**. If the stronger condition*

$$f(\mathbf{x}^*) < f(\mathbf{x}) \tag{3.2}$$

*holds for all $\mathbf{x} \neq \mathbf{x}^*$, $\mathbf{x} \in \Omega$, then the minimizer and the global minimum are called **strict**.*

Definition 3.2 (Open ball)**.** *An **open ball** around $\mathbf{x}_0 \in \mathbb{R}^n$ of radius $R > 0$ is given by the set*

$$B_R(\mathbf{x}_0) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{x}_0\|_2 < R\}$$

where $\|x\|_2$ is the Euclidean norm of vector / point $\mathbf{x}$ (see Appendix A).

Definition 3.3 (Local minimizer)**.** *A particular point $\mathbf{x}^* \in \Omega$ is called a **local minimizer**, if there exists a radius $R > 0$ such that $B_R(\mathbf{x}^*) \subset \Omega$ and*

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \tag{3.3}$$

*holds for all $\mathbf{x} \in B_R(\mathbf{x}^*)$, i.e. if there exists an open ball (however small) around $\mathbf{x}^*$ in which $f(\mathbf{x}^*)$ assumes the smallest value. The argument point $\mathbf{x}^*$ is the local minimizer, and the value of the target function $f(\cdot)$ at this point is called the **local minimum**. Similarly, for a **strict local minimizer** $\mathbf{x}^*$ we can find a radius $R > 0$ such that $B_R(\mathbf{x}^*) \subset \Omega$ and*

$$f(\mathbf{x}^*) < f(\mathbf{x}) \tag{3.4}$$

holds for all $\mathbf{x} \in B_R(\mathbf{x}^*) \setminus \{\mathbf{x}^*\}$.

Problem 3.1.1 (Minimizers)**.**
We only work over the real numbers, i.e. in all the following problems we have $\Omega \subset \mathbb{R}$.

- Give an example minimization problem which has both a global and a local minimizer.
- Give an example minimization problem which has no minimizer.
- Give an example minimization problem which has several minimizers.
- Give an example minimization problem which has infinitely many strict local minimizers in a finite interval.

To give a problem you need to specify both the set $\Omega \subset \mathbb{R}$ and the cost function $f(\cdot)$.

3.2 Classes of Optimization Problems

We have so far assumed very little about the set Ω, which is also called the **set of feasible points**. In many optimization problems it is possible to describe the set Ω more explicitly through a set of equations and inequalities, summarily called **constraint equations**, like for example in the following problem:

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0 \quad , i \in \{1, \ldots, m\} \\ & h_j(\mathbf{x}) = 0 \quad , j \in \{1, \ldots, p\} \end{aligned}$$

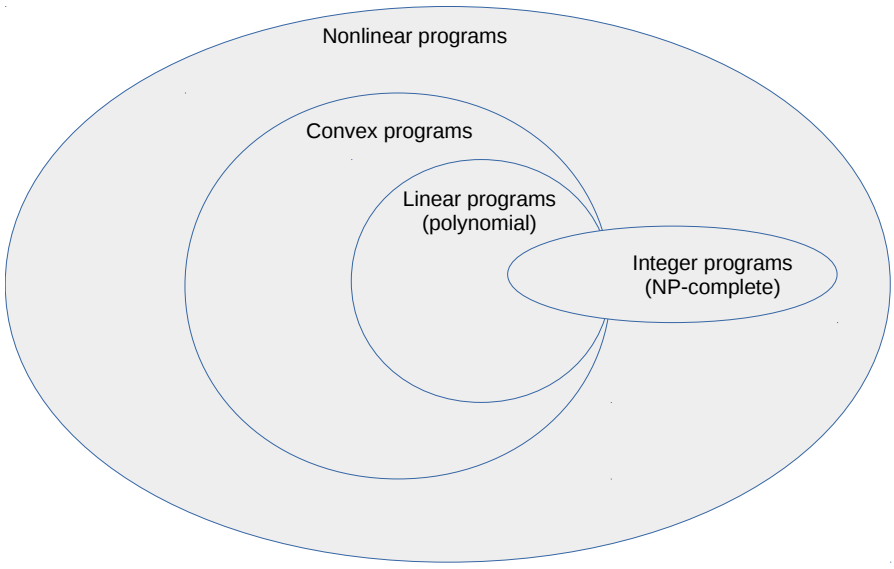
Here, the functions $g_i(\cdot)$ are the **inequality constraints** and the functions $h_j(\cdot)$ are the **equality constraints**. As an example, consider the following problem:

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & (x_1 - 1)^2 + x_2 - 2 \\ \text{subject to} \quad & x_2 - x_1 = 1 \\ & x_1 + x_2 \leq 5 \end{aligned}$$

Depending on the particular shape of the equality and inequality constraints it is possible to coarsely classify minimization problems (these are also historically called **mathematical programming** problems):

- In **nonlinear programming** there are no restrictions to the form of the equality and inequality constraints or the form of the cost function.
- In **convex programming** the set Ω described by the functions $g_i(\cdot)$ and $h_i(\cdot)$ (i.e. the set of all points $\mathbf{x} \in \mathbb{R}^n$ which satisfy all the equalities and inequalities given by $f_i(\cdot)$ and $g_i(\cdot)$ simultaneously) is a particular kind of set, a so-called convex set, and the cost function $f(\cdot)$ is a particular kind of function, a convex function. We discuss this class of problems further in Section 3.4. Clearly, convex programs are a subset of the set of nonlinear programs.
- In **linear programming** the set Ω is an even more specialized set, the constraint functions $g_i(\cdot)$ and $h_j(\cdot)$ have to be linear (or affine-linear) functions, and the cost function is a linear function as well. Through this prescription the set Ω assumes a particular geometric shape, it is a **polytope**. We discuss linear programming in Section 3.5 and in Chapter 4.
- In **integer programming** the decision variables are not arbitrary real numbers but are restricted to take only integer values, so we have $\Omega \subset \mathbb{Z}^n$. In **mixed-integer programming** some of the decision variables can be real numbers, but others are restricted to integer values. In **integer linear programs** the problem formulation looks like an ordinary linear program but has additional integer constraints on the decision variables.

The relationships between these classes are shown in the following figure (adapted from [15, Fig. 1-1])



There is not really a general approach which can solve any given optimization program efficiently. A lot depends on the structure of the set Ω: Ω can be discrete (i.e. consist of finitely many or at most a countably infinite number of points) or continuous (i.e. have an un-countable number of points), and the more "regular" the set Ω is, the better we are able to find more efficient and taylored optimization algorithms. In the discrete case we enter the realm of combinatorial optimization [15], [11] which includes a wide range of graph problems, scheduling problems and many others, and many of these problems are computationally hard (NP-complete). For nonlinear programs, it is often true that a global optimum is very hard, if not impossible to find and that we have to content ourselves with a local optimum.

3.3 Nonlinear Programming

Nonlinear programming [2] is the most general class of optimization problems. In unconstrained problems (where $\Omega = \mathbb{R}^n$) the cost function then can be a highly non-linear function with several local minima. It is relatively easy (at least conceptually) to find a local minimum, but it is very hard to find a global optimum. Some of the methods used for nonlinear programming include genetic algorithms, simulated annealing or particle swarm optimization. These methods are basically search methods: pick some starting point, find a local optimum from there, then pick another starting point, find another local optimum and keep the better of the optima identified so far, and so on, until you run out of computation time. With such an approach we are not guaranteed to find a global optimum. Instead, they will usually identify a number of local minima (depending on how much computation time you have allocated to the problem) and will give you the best amongst these (i.e. the local minimizer $\mathbf{x}^*$ that has the smallest cost among all the local minimizers found during the search).

Bonus problem 3.3.1 (How to find a local minimum)**.**
Suppose that $\Omega = (-1, 1)$ and you are given a target function $f(\cdot)$ that is differentiable a sufficent number of times (e.g. $f(x) = x^2$) but so unwieldy that you cannot apply the typical calculus approach (find the derivative, set it to zero and solve for x) using pen and paper, but at least you can compute derivatives of $f(\cdot)$ at any point in $(-1, 1)$. You may remember from high-school calculus that a necessary condition for a local minimum in some point x is that $f'(x) = 0$ holds. Assume that our function has no saddle points, only local minima or maxima. You are given a starting point $x_0 \in (-1, 1)$. Sketch a search strategy to find a local minimum.

Nonlinear programming is in general hard, and no algorithm is known which solves a general nonlinear problem in polynomial time such that a global minimum is found.

3.4 Convex Programming

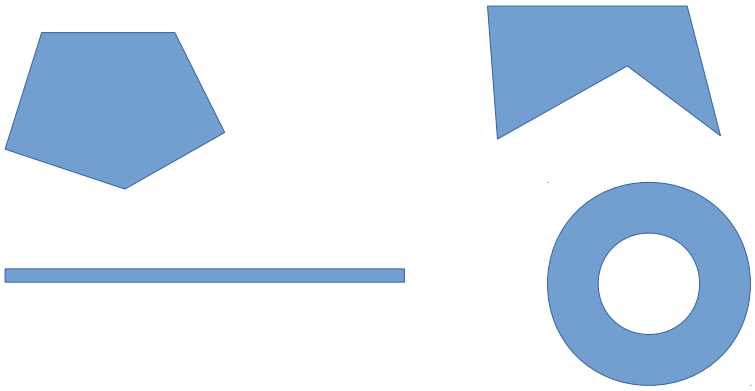
We start with the definition of a convex set and a convex function.

Definition 3.4 (Convex set)**.** *A non-empty set $\Omega \subset \mathbb{R}^n$ is called **convex** (or a **convex set**) when for all $\mathbf{u} \in \Omega$ and $\mathbf{v} \in \Omega$ the set Ω also includes the "connecting line" between $\mathbf{u}$ and $\mathbf{v}$, i.e. if*

$$\{\lambda\mathbf{u} + (1 - \lambda)\mathbf{v} : 0 \leq \lambda \leq 1\} \subset \Omega \tag{3.5}$$

*For given $\mathbf{u}$ and $\mathbf{v}$ and $0 \leq \lambda \leq 1$ the point $\lambda\mathbf{u} + (1 - \lambda)\mathbf{v}$ is called a **convex combination** of $\mathbf{u}$ and $\mathbf{v}$. With this notion, a convex set is a set which contains all convex combinations of all pairs of its points.*

Note that in this definition the restriction of λ to the set $[0, 1]$ is essential, as we really only care about the line segment *between* the two points $\mathbf{u}$ and $\mathbf{v}$. Examples of convex and non-convex sets are shown in the following figure:



The real n-dimensional vector space $\mathbb{R}^n$ is a convex set, singletons (i.e. sets consisting of exactly one point) are convex sets, and an open ball is a convex set too.

Problem 3.4.1 (Intersections of convex sets)**.**
Show that if $C_1, C_2, C_3, \ldots$ is a sequence of convex sets then their intersection

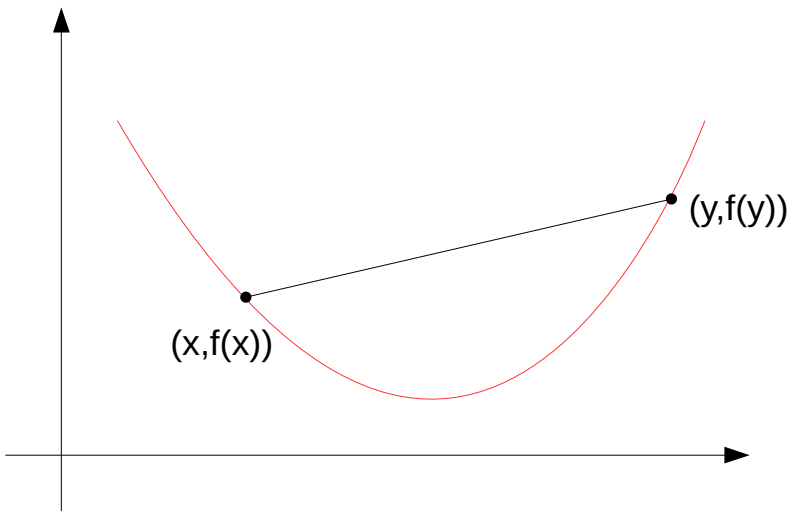
$$C := \bigcap_{i=1}^{\infty} C_i$$

is a convex set (if not empty).

Definition 3.5 (Convex function)**.** *Be $\Omega \subset \mathbb{R}^n$ a convex set and $f : \Omega \mapsto \mathbb{R}$ a function. Then $f(\cdot)$ is called a **convex function** in Ω if for any two points $\mathbf{u} \in \Omega$ and $\mathbf{v} \in \Omega$ and for each $\lambda \in \mathbb{R}$ with $0 \leq \lambda \leq 1$ the following holds:*

$$f\left(\lambda \mathbf{u} + (1-\lambda)\mathbf{v}\right) \leq \lambda f(\mathbf{u}) + (1-\lambda) f(\mathbf{v}) \tag{3.6}$$

In other words, if we pick two points $(x, f(x))$ and $(y, f(y))$ on the graph of $f(\cdot)$, then between x and y the function remains below the connecting line between these two points, compare the following figure:



A key property of convex functions is the following:

Theorem 3.1 (Local minima are global minima)**.** *Be $f(\cdot)$ a convex function over the convex set Ω, and be $\mathbf{x}_0 \in \Omega$ a local minimizer. Then $\mathbf{x}_0$ is also a global minimizer.*

Excursion (Proof of this statement [5, Chap. 4])

Since $\mathbf{x}_0$ is a local minimizer, there exists an $R > 0$ such that

$$f(\mathbf{x}_0) = \inf \left\{ f(\mathbf{x}) : \mathbf{x} \in \Omega, \|\mathbf{x} - \mathbf{x}_0\|_2 < R \right\}$$

Suppose now that $\mathbf{x}_0$ is not a global minimizer, i.e. there is another point $\mathbf{x}_1 \in \Omega$ with $\mathbf{x}_1 \neq \mathbf{x}_0$ such that $f(\mathbf{x}_1) < f(\mathbf{x}_0)$. We must have $\|\mathbf{x}_1 - \mathbf{x}_0\|_2 \geq R$, i.e. $\mathbf{x}_1$ must lie outside the ball $B_R(\mathbf{x}_0)$ in which $\mathbf{x}_0$ is a local minimizer.

Since Ω is a convex set, any point on the line segment between $\mathbf{x}_0$ and $\mathbf{x}_1$ must also lie in Ω, which holds in particular for the point

$$\mathbf{y} = \left(1 - \frac{R}{2\|\mathbf{x}_1 - \mathbf{x}_0\|_2}\right) \cdot \mathbf{x}_0 + \frac{R}{2\|\mathbf{x}_1 - \mathbf{x}_0\|_2} \cdot \mathbf{x}_1$$

For this point we have $\|\mathbf{x}_0 - \mathbf{y}\|_2 = \frac{R}{2} < R$ and by the convexity of $f(\cdot)$ we must have

$$f(\mathbf{y}) \leq \left(1 - \frac{R}{2\|\mathbf{x}_1 - \mathbf{x}_0\|_2}\right) \cdot f(\mathbf{x}_0) + \frac{R}{2\|\mathbf{x}_1 - \mathbf{x}_0\|_2} \cdot f(\mathbf{x}_1) < f(\mathbf{x}_0)$$

which is a contradiction. Therefore there can be no global minimizer $\mathbf{x}_1$ with $f(\mathbf{x}_1) < f(\mathbf{x}_0)$.

It is exactly this previous property (local minimum equals global minimum) which makes convex programming problems so interesting [5], [3]. In principle, this allows to carry out minimization through iterative methods, a caricature of which could look as follows: when we are at some point $\mathbf{x}_0$, inspect the value of the cost function at neighboured points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ at a fixed distance to $\mathbf{x}_0$ and covering many different directions in space. If there is any other point $\mathbf{x}_i$ for which $f(\mathbf{x}_i) < f(\mathbf{x}_0)$ then repeat the procedure with the best such $\mathbf{x}_i$, otherwise stop, as we have found a local (and thus global) minimum. With iterative methods like this (which come under more dignified names like for example gradient-descent or interior-point methods) it is in fact possible to find the global optimum very efficiently, with a modest number of iterations.

3.5 Linear Programming

Linear programming is a special case of convex programming and of great practical importance. There are several open-source and commercial solvers available which can tackle linear programming problems with thousands of decision variables and constraint equations. We will see later on in more detail that the set of feasible points Ω described by affine-linear constraint functions $g_i(\cdot)$ and $h_i(\cdot)$ is not only a convex set, but is a convex polytope, i.e. is the n-dimensional analogon of a two-dimensional convex polygon.

In Chapter 4 we will discuss linear programming in more detail, including one particular solution algorithm, the iterative simplex algorithm. This particular algorithm is of a combinatorial nature, i.e. it does not use any methods from calculus and only considers a finite (but potentially very large) number of candidate points. It works very well on a wide range of practical problems, but there are pathological linear programming problems in which an exponentially growing (in the number of decision variables or constraints) number of candidate solutions is inspected. More precisely, in [15, Chap. 8] the following is proven (paraphrased here):

Theorem 3.2 (Simplex algorithm)**.** *For every $d > 1$ there exists a linear programming problem with $2d$ constraint equations, $3d$ decision variables and integer coefficients with absolute value bounded by 4, such that the simplex algorithm may take $2^d - 1$ iterations to find the optimum.*

There are, however, other algorithms for linear programming (for example the ellipsoid method by Khachyan [15], [11] or the interior-point method by Karmarkar [8]) which are not combinatorial but rather make use of analytical properties of the constraint and cost functions. These algorithms can be proven to require only a polynomial number of iterations, but in practice they are often slower than the simplex algorithm. We will not dwell on this any further.

3.6 Integer Linear Programming

The available solution methods for linear programming (simplex algorithm, Karmarkars algorithm) all work under the assumption that the decision variables x_i can take on any real number, or perhaps any non-negative real number.

At a first glance, integer linear programming problems look very similar to linear programming problems, but an additional restriction is introduced that **the decision variables x_i have to assume only (non-negative) integer values**. Typically, an integer decision variable x_i only can take on a finite number of integer values, and for **binary** variables only the values 0 and 1 are allowed.

The integer constraint is a game changer, and turns linear programming from an efficiently solvable (in polynomial time) problem into an NP-hard problem. In fact, many other NP-hard problems like the traveling salesman problem or the satisfiability problem for Boolean expressions can be formulated as integer linear programs [15, Chap. 13]. However, not all integer linear programming problems are hard, there are some problems like network flow problems which can be formulated this way and for which very efficient algorithms exist. Saying that integer LP problems are NP-hard means that there is in general no substantially better approach than the brute-force approach, in which we do the following **for all allocations of possible values to the decision variables**:

- Check whether the given allocation $\mathbf{a}$ is a feasible allocation, i.e. whether it satisfies the constraint equations $g_i(\cdot)$ and $h_i(\cdot)$.
- Check whether the cost value for this allocation, $f(\mathbf{a})$ is smaller than the smallest cost value found so far. If so, keep it as the new least-cost value.

Problem 3.6.1 (Integer Linear Programming)**.**
Suppose you have $d > 1$ decision variables, each of which can assume $n > 1$ different integer values. How many different allocations do you have to try with the brute-force approach? **Bonus**: Calculate this number for $d = 128$ and $n = 3$. Give it as a number. If you can, give it in words :-)

When faced with an integer linear programming problem and knowing that these may be very hard, one might be tempted to try an approached called **relaxation**:

- Drop the integer constraints on the decision variables x_i and pretend that these can assume arbitrary (non-negative) real numbers.
- Solve the linear program under this assumption, giving the values x_i^* for the decision variables where the cost function takes the minimum value.
- Round the x_i^* to the nearest integer.

In general this approach does not give the minimizing integer allocation (it might not even give a feasible integer allocation), and depending on the problem the cost for the rounded allocation might be very far away from the true best cost. Occasionally a relaxation approach works well and gives at least a reasonable approximation to the true best cost, but you have to double-check that the allocation you get after rounding indeed satisfies the constraints.

Chapter 4

Linear Optimization

In this chapter we discuss linear optimization (or linear programming) in general and have a closer look at the simplex algorithm, which is available in a number of commercial and open-source packages like `CPLEX` or `glpk`. The material is entirely standard and based on a number of sources, in particular [13] and [6].

A linear optimization (or linear programming) has the following form:

$$\begin{array}{rl} \text{minimize}_{[\mathbf{x}]} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{subject to} & a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = b_1 \\ & a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = b_2 \\ & \ldots \\ & a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n}x_n = b_m \\ & x_1 \geq 0 \\ & x_2 \geq 0 \\ & \ldots \\ & x_n \geq 0 \end{array} \tag{4.1}$$

where $x_1, \ldots, x_n$ (with $x_i \in \mathbb{R}^{\geq 0}$) are the decision variables, the values c_i are the given cost coefficients, and the coefficients $a_{i,j}$ and b_i are given real values. Such a problem is called linear since the decision variables x_i occur only as a linear term in both the cost function and the constraint functions (which all are only simple sums of linear terms), and not in a non-linear fashion like in x_i^2 or e^{x_i}. Note that there are n decision variables and m constraint equations, and one usually has that $m \leq n$.[1] In more compact matrix and vector notation the problem becomes

$$\begin{array}{rl} \text{minimize}_{[\mathbf{x}]} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \tag{4.2}$$

where $\mathbf{x} = (x_1, \ldots, x_n)^T$ is the vector of decision variables, $\mathbf{A} = ((a_{i,j}))_{i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\}}$ is the coefficient matrix for the linear constraints, the vector $\mathbf{b} = (b_1, \ldots, b_m)$ contains the right-hand sides of the constraints and the vector $\mathbf{c} = (c_1, \ldots, c_n)^T$ contains the costs. This form is also known as the **standard form**. Refer to

[1] Otherwise, for $m > n$ the problem would be **overconstrained**, and is either infeasible (i.e. the constraints cannot be satisfied simultaneously) or it is possible to eliminate constraints and reduce their number from m down to n.

Appendix A for a summary of matrix and vector algebra and notation. There are important variations of linear programs, for example by replacing equality- with inequality constraints:

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

or

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

or a mixture of these two:

$$\begin{aligned} \text{minimize}_{[\mathbf{x}_1,\mathbf{x}_2]} \quad & \mathbf{c}^T \cdot \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \\ \text{subject to} \quad & \mathbf{A}_1\mathbf{x}_1 \leq \mathbf{b}_1 \\ & \mathbf{A}_2\mathbf{x}_2 \geq \mathbf{b}_2 \\ & \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \geq \mathbf{0} \end{aligned}$$

All these variations can be transformed into the canonical (or standard) form given in Equation 4.2. We will show in Section 4.2 how this can be done. The importance of the standard form for us is that the simplex algorithm can be conveniently explained in terms of this representation, in practice (and when using CPLEX to solve LP instances) you can freely use any of the variations given below.

The set of all points $\mathbf{x} \in \mathbb{R}^n$ which satisfy the constraints

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

is called the **feasible set** and its points are called the **feasible points**. We normally assume that the feasible set is non-empty, as otherwise the problem is ill-posed. It is also advantageous if the feasible set is bounded:

Definition 4.1 (Bounded subset of $\mathbb{R}^n$)**.** *A subset $\Omega \subset \mathbb{R}^n$ is bounded if we can find a $0 < R < \infty$ such that Ω is completely contained in the open ball of radius R around the origin, i,e, $\Omega \subset B_R(\mathbf{0})$.*

For a non-empty and bounded feasible set an optimum/minimum always exists and will be found by the simplex algorithm. In the case of unbounded feasible sets the simplex algorithm will abort operation, as it is not clear that a minimum value actually exists.

4.1 Examples

We will discuss a number of examples for the formulation of linear programs.

4.1.1 The Diet Problem

Suppose that a human needs m different types of nutrients per day, b_1 units of nutrient one, b_2 units of nutrient two, ..., and b_m units of nutrient m. Furthermore there are n different types of food, and the j-th type of food has cost c_j ($j \in \{1, \ldots, n\}$). A unit of food j contains $a_{i,j}$ units of the i-th nutrient. The decision to make is how many units of food j we want to buy, we denote this number by x_j, and the goal is to minimize the monetary cost of our diet, i.e. to minimize

$$c_1 x_1 + \ldots + c_n x_n$$

subject to the nutritional constraints

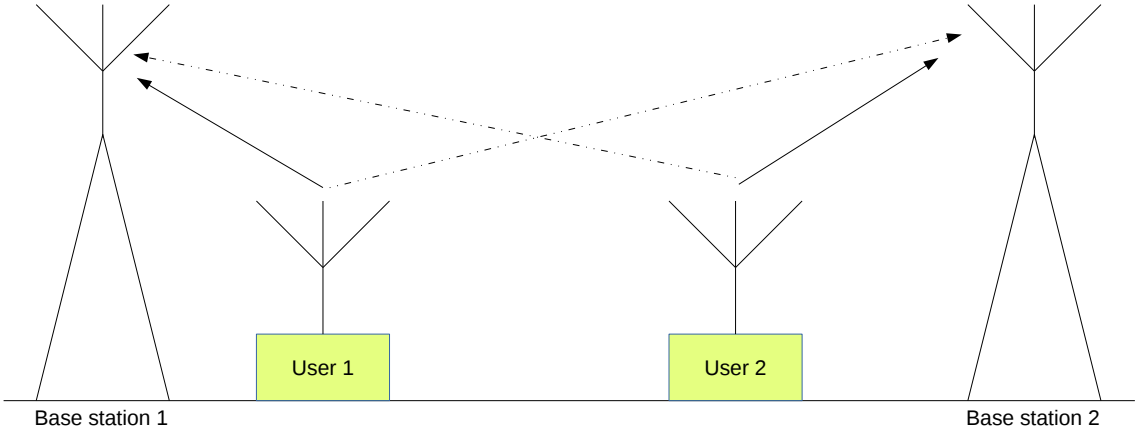
$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n &\geq b_1 \\ \ldots & \quad \ldots \\ a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n}x_n &\geq b_m \end{aligned}$$

Here we assume that the x_i can be arbitrary non-negative real numbers, i.e. $x_i \geq 0$.

Problem 4.1.1 (Diet problem in vector notation)**.**
Re-write the problem in vector and matrix notation.

4.1.2 Wireless Communications

Consider a wireless communication system as shown in the following figure:



We are given a number n of mobile users and the same number n of base stations. Each mobile user i wants to transmit data wirelessly to base station i. The i-th user uses a transmit power of p_i Watt and has an attenuation factor $h_{i,j}$ to the j-th base station – as wireless waves propagate into entire space, any wireless transmission will be received however weak at each of the available base stations. As a result, base station j receives power $h_{i,j}p_i$ from user i. Typically, user i is closest to base station i and hence $h_{i,i} \gg h_{i,j}$ for $j \neq i$. If we focus on one user i transmitting data to base station i, then BS i will treat i's signal as a useful signal, whereas the signals of all other users j are treated as interference. Furthermore, base station i (and any other base station) will observe

a certain level of noise $\sigma^2 > 0$ (e.g. thermal noise in the receiver circuitry). For successful reception at base station i it is required that user i's signal exceeds a given so-called signal-to-interference-plus-noise ratio, i.e. for user i the following relation must be satisfied:

$$\frac{h_{i,i}p_i}{\sum_{j\neq i} p_j h_{j,i} + \sigma^2} \geq \gamma_i$$

where γ_i is a given threshold value. When the sum of interference and noise for user i is too strong, base station i cannot decode user i's data.

The goal is to find an allocation of transmit powers $p_i \geq 0$ to the different users such that the total transmit power $p_1 + \ldots + p_n$ becomes minimal and the data from all stations can be reliably received, i.e. we want to solve the problem

$$\begin{array}{rll} \text{minimize}_{[\mathbf{p}]} & p_1 + \ldots + p_n & \\ \text{subject to} & \frac{p_i h_{i,i}}{\sum_{j\neq i} p_j h_{j,i} + \sigma^2} \geq \gamma_i & \text{for } i = 1, 2, \ldots, n \\ & p_i \geq 0 & \text{for } i = 1, 2, \ldots, n \end{array}$$

which after a small re-arrangement can be turned into the following linear program:

$$\begin{array}{rll} \text{minimize}_{[\mathbf{p}]} & p_1 + \ldots + p_n & \\ \text{subject to} & p_i h_{i,i} - \gamma_i \left(\sum_{j\neq i} p_j h_{j,i} \right) \geq \gamma_i \sigma^2 & \text{for } i = 1, 2, \ldots, n \\ & p_i \geq 0 & \text{for } i = 1, 2, \ldots, n \end{array}$$

Problem 4.1.2 (Matrix form)**.**
Give the matrix form of this problem. Give explicit expressions for the involved vectors / matrices. Give also a variation of this problem where additionally each transmit power p_i is upper-bounded by some maximum transmit power $P > 0$ (e.g. for legal reasons).

4.1.3 A Transportation Problem

A company produces a certain good in m different production locations such that in location i an amount of a_i units of the good is produced. This company has n different customers at different locations. Customer j requires b_j units of the goods and the company will balance its overall production such that

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$$

holds. The cost of shipping a unit of the goods from production location i to destination location j is given by $c_{i,j}$. The company wants to determine how many units of goods to ship from production location i to destination location j, denoted as $x_{i,j}$ so as to minimize the total cost.

Problem 4.1.3 (Formulating the transportation problem)**.**
Formulate the transportation problem as a linear programming problem, i.e. give the cost function and the constraint equalities/inequalities explicitly, and then represent it in vector / matrix notation (You do not need to explicitly write down the matrix, a verbal description will suffice).

Problem 4.1.4 (A variation of the transportation problem)**.**
In the previous transportation problem 4.1.3 we have assumed that the quantities a_i of goods produced in production location i were given to us. We can modify the problem to also make the quantities a_i into decision variables, still with the overall objective of minimizing the cost.
Formulate the problem.

4.2 Reduction to Standard Form

We say that a linear program is given in standard or canonical form when it is written in the following way:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{4.3}$$

where $\mathbf{x}$ is an n-dimensional vector of decision variables, $\mathbf{c}$ is an n-dimensional vector of cost values, $\mathbf{b}$ is an m-dimensional vector (the "right-hand side") and matrix $\mathbf{A}$ is an $m \times n$ matrix (m rows and n columns). All involved numbers are real numbers. One can assume that $\mathbf{b} \geq \mathbf{0}$ holds, since any equation (or matrix row) corresponding to a negative b_i can be multiplied by -1.

Problem 4.2.1 (Reduction to standard form)**.**
Reduce the following problem to standard form:

$$\begin{aligned} \text{minimize} \quad & x_1 + 2x_2 + 3x_3 \\ \text{subject to} \quad & x_1 + 6x_2 - 4x_3 = 5 \\ & 3x_1 + x_2 - 2x_3 = -8 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{aligned}$$

Write the result in full vector/matrix form.

4.2.1 Case 1: "Greater-Than" Constraints

We first consider the case of "greater-than" constraints. Suppose we are given a linear program in the form

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Then we can reduce this problem to the canonical form 4.3 by introducing *surplus variables* y_i and re-writing the problem as

$$\begin{aligned} \text{minimize}_{[\mathbf{x},\mathbf{y}]} \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{subject to} \quad & a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n - y_1 = b_1 \\ & a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n - y_2 = b_2 \end{aligned}$$

$$
\begin{aligned}
&\ldots \\
&a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n}x_n - y_m = b_m \\
&x_1 \geq 0, \ldots, x_n \geq 0 \\
&y_1 \geq 0, \ldots, y_m \geq 0
\end{aligned}
$$

So, we introduce new decision variables $y_1, \ldots, y_m$ which appear only in the constraints but not in the cost function. In matrix/vector notation the re-written problem looks like

$$
\begin{aligned}
\text{minimize}_{[\mathbf{x},\mathbf{y}]} \quad & \mathbf{c}^T \cdot \mathbf{x} \\
\text{subject to} \quad & \mathbf{A} \cdot \mathbf{x} - \mathbf{I}_m \cdot \mathbf{y} = [\mathbf{A}, -\mathbf{I}_m] \cdot \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \mathbf{b} \\
& \mathbf{x} \geq \mathbf{0},\ \mathbf{y} \geq \mathbf{0}
\end{aligned}
$$

where we have created the larger coefficient matrix $[\mathbf{A}, -\mathbf{I}_m]$ by horizontally stacking the negative of the identity matrix $\mathbf{I}_m$ on the right to matrix $\mathbf{A}$, and the new decision vector

$$
\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}
$$

by stacking the "old" decision vector $\mathbf{x}$ on top of the additional vector $\mathbf{y}$.

Note that we only introduce surplus variables for actual "greater-than" constraints – when the constraints also contain equations then these remain untouched and do not involve the surplus variables.

Problem 4.2.2 (Reduction to standard form)**.**
Reduce the following problem to standard form:

$$
\begin{aligned}
\text{minimize}_{[\mathbf{x}]} \quad & x_1 + 2x_2 + 3x_3 + 4x_4 \\
\text{subject to} \quad & x_1 + 6x_2 - 4x_3 = 5 \\
& 3x_2 + x_3 - 2x_4 \geq 8 \\
& 3x_1 - 4x_3 + 13x_4 \geq 10 \\
& x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0
\end{aligned}
$$

and write the result both as equations and in full matrix/vector notation (i.e. with one vector component for each decision variable).

4.2.2 Case 2: "Smaller-Than" Constraints

If the problem is of the form

$$
\begin{aligned}
\text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\
& \mathbf{x} \geq \mathbf{0}
\end{aligned}
$$

then we follow a very similar approach by introducing *slack variables* y_i and converting the problem to

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} + \mathbf{I}_m\mathbf{y} = [\mathbf{A}, \mathbf{I}_m] \cdot \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0},\ \mathbf{y} \geq \mathbf{0} \end{array}$$

One may (and should!) ask at this point whether it actually makes sense to change the problem, i.e. if the modified problem is in some sense equivalent to the initial problem. Without going into further detail we state that no harm is done by this: an optimal solution to the modified problem is an optimal solution to the original problem.

Problem 4.2.3 (Reduction to standard form)**.**
Reduce the following problem to standard form:

$$\begin{array}{ll} \text{minimize}_{[\mathbf{x}]} & x_1 + 2x_2 + 4x_3 + 8x_4 \\ \text{subject to} & 2x_1 + 8x_2 - x_3 \leq 7 \\ & 3x_2 + x_3 - 2x_4 \geq 8 \\ & 5x_1 - 2x_2 + 11x_4 = 6 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0 \end{array}$$

and write the result both as equations and in full matrix/vector notation (i.e. with one vector component for each decision variable).

4.2.3 Case 3: Free Decision Variables

Now suppose we have a problem in which (some of) the decision variables are not constrained to be non-negative, e.g. the problem is just

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b} \end{array}$$

Any decision variable x_i that is not constrained to be non-negative is also called a **free variable**. We can then write x_i as the difference of two new non-negative decision variables:

$$x_i = u_i - v_i$$

and substitute x_i by $u_i - v_i$ in our problem (in both the objective function and the constraints). With respect to our coefficient matrix $\mathbf{A}$ this means that the column $\mathbf{a}_i$ corresponding to decision variable x_i is replaced by two columns, the first of which is given by $\mathbf{a}_i$ for the new variable u_i and the column is given by $-\mathbf{a}_i$ for the new variable v_i. Through this approach we increase the number of decision variables by one per free variable.

Problem 4.2.4 (Reduction to standard form)**.**
Reduce the following problem to standard form:

$$\begin{array}{rl} \text{minimize}_{[\mathbf{x}]} & x_1 + 2x_2 + 4x_3 + 8x_4 \\ \text{subject to} & 2x_1 + 8x_2 - x_3 = 7 \\ & 3x_2 + x_3 - 2x_4 = 8 \\ & 5x_1 - 2x_2 + 11x_4 = 6 \\ & x_1 \geq 0, x_2 \geq 0 \end{array}$$

and write the result both as equations and in full matrix/vector notation (i.e. with one vector component for each decision variable).

4.2.4 Reduction Exercises

Problem 4.2.5 (Reduction to standard form)**.**
Reduce the following problem to standard form:

$$\begin{array}{rl} \text{minimize}_{[x_1,x_2]} & -x_1 + x_2 \\ \text{subject to} & x_1 - x_2 \leq 2 \\ & x_1 + x_2 \leq 6 \\ & x_1 \geq 0 \end{array}$$

Just give equations.

Problem 4.2.6 (Reduction to standard form)**.**
Reduce the following problem to standard form:

$$\begin{array}{rl} \text{minimize}_{[x_1,x_2]} & 6x_1 + 7x_2 \\ \text{subject to} & 4x_1 - 3x_2 \leq 9 \\ & x_1 + x_2 \geq 13 \end{array}$$

Just give equations.

Problem 4.2.7 (Reduction to standard form)**.**
Reduce the following problem to standard form:

$$\begin{array}{rl} \text{minimize}_{[x_1,x_2,x_3,x_4]} & x_1 + 2x_2 + 3x_3 + 4x_4 \\ \text{subject to} & x_1 + 6x_2 - 4x_3 = 5 \\ & 3x_2 + x_3 - 2x_4 \geq 8 \\ & 3x_1 - 4x_3 + 13x_4 \geq 10 \end{array}$$

Just give equations.

4.3 Excursion: Geometric View

In this section we consider a two-dimensional linear program (see [6, Sec. 5.3]) to gain some more insight into the nature of linear programming problems. For purposes of illustration the problem here is not given in standard form. It reads:

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

where

$$\begin{aligned} \mathbf{c} &= (1,5)^T \\ \mathbf{x} &= (x_1,x_2)^T \\ \mathbf{A} &= \begin{pmatrix} 5 & 6 \\ 3 & 2 \end{pmatrix} \\ \mathbf{b} &= (30,12)^T \end{aligned}$$

If we write this out and introduce explicit names for our objective and constraint functions our problem becomes

$$\begin{array}{ll} \text{maximize} & f(x_1,x_2) = x_1 + 5x_2 \qquad (4.4) \\ \text{subject to} & g_1(x_1,x_2) = 5x_1 + 6x_2 \leq 30 \qquad (4.5) \\ & g_2(x_1,x_2) = 3x_1 + 2x_2 \leq 12 \qquad (4.6) \\ & x_1 \geq 0, x_2 \geq 0 \qquad (4.7) \end{array}$$

Suppose we are given three numbers $a \neq 0$, $b \neq 0$ and c, then the set:

$$\{(x_1,x_2) \in \mathbb{R}^2 : ax_1 + bx_2 = c\}$$

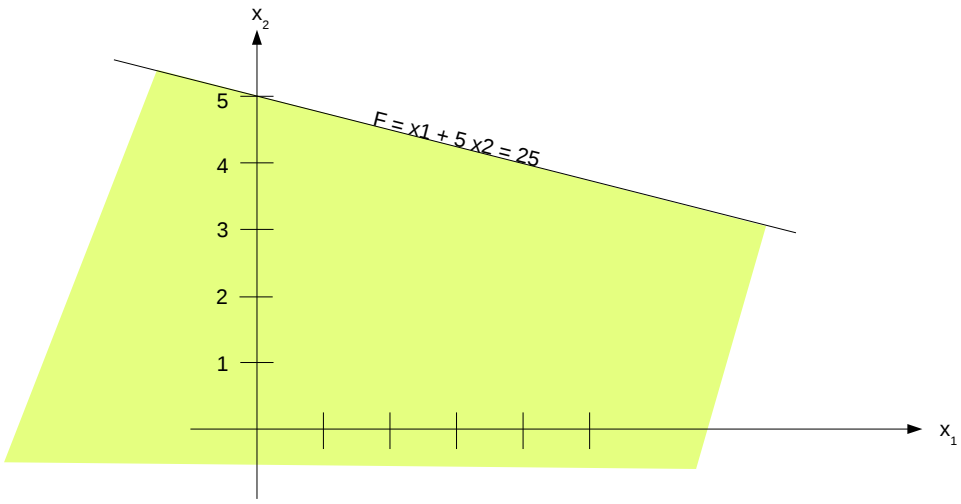
actually describes a line in $\mathbb{R}^2$. To see this, one can pick an arbitrary value for x_1 and then use the equation $ax_1 + bx_2 = c$ and the non-zeroness of a and b to determine the value x_2 that makes the equation true. This value x_2 is uniquely determined. If we replace the equality sign in $ax_1 + bx_2 = c$ by an inequality sign, say $ax_1 + bx_2 \leq c$, then the set

$$\{(x_1,x_2) \in \mathbb{R}^2 : ax_1 + bx_2 \leq c\}$$

actually describes the half-plane below (and including) the line

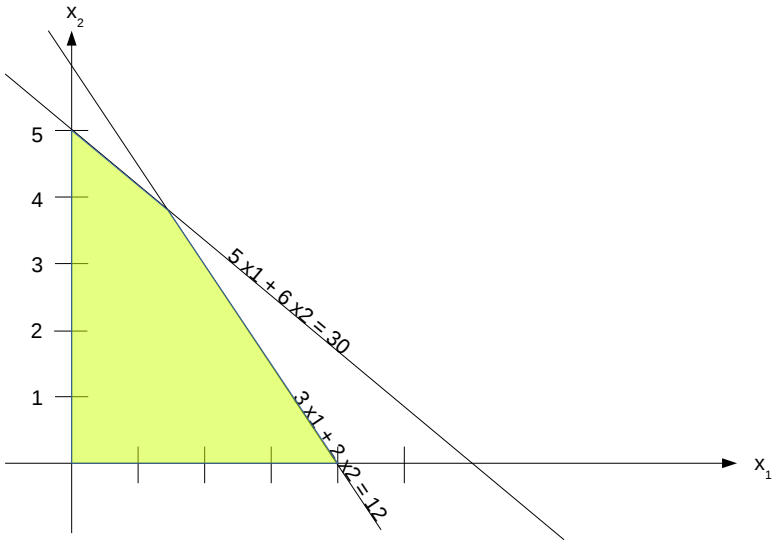
$$\{(x_1,x_2) \in \mathbb{R}^2 : ax_1 + bx_2 = c\}$$

This is illustrated in the following figure:



for the example $x_1 + 5x_2 \leq 25$.

Going back to our problem, the constraint functions $g_1(\cdot)$ and $g_2(\cdot)$ and the non-negativity constraints for x_1 and x_2 give four different half-spaces (what are these?), and a point (x_1, x_2) which satisfies all four constraints simultaneously (and thus is a feasible point) hence must lie in the *intersection* of these four half-spaces. This intersection is shown in the following figure:



and displays the entire set of feasible points.

Problem 4.3.1 (Intersection of half-spaces)**.**
Show that a half-space

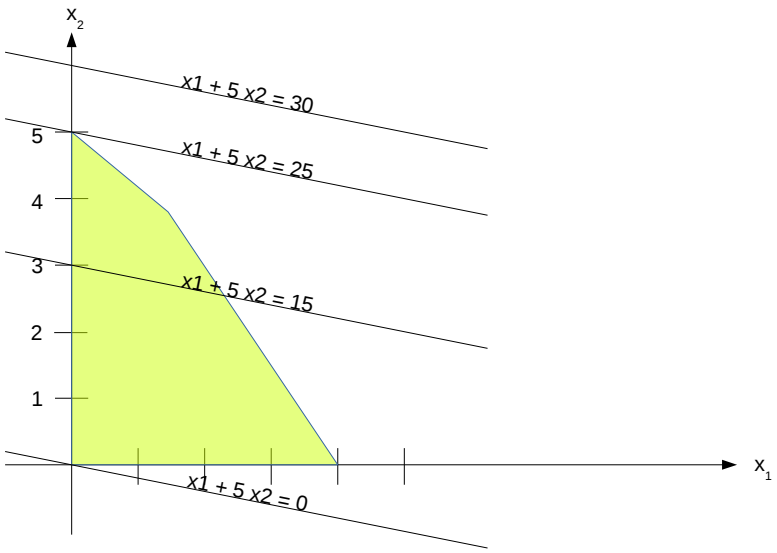
$$\mathcal{H} = \{(x, y) \in \mathbb{R}^2 : ax + by \leq c\}$$

is a convex set (see Section 3.4). Then the solution to Problem 3.4.1 guarantees that any intersection of half-spaces is a convex set.

Now consider the cost function $f = x_1 + 5x_2$ with different values for the right-hand side, i.e. we consider different lines

$$\{(x_1, x_2) \in \mathbb{R}^2 : x_1 + 5x_2 = c\}$$

for different values of c as shown in the following figure:



Note that the objective function $f(\cdot)$ has the same value, c, on all points (x_1, x_2) of the line $x_1 + 5x_2 = c$. To find the desired maximum we need to find values of c such that the intersection of the line $x_1 + 5x_2 = c$ with the set of feasible points is not empty, and among these we need to find the one with the largest value of c. In the figure this is the point $(0, 5)$, and in this point the objective function assumes the value 25. No feasible point (x_1, x_2) with a larger value of the objective function can be found.

In this particular example the maximum is achieved in a "corner point" of the set of feasible points. **This is not a coincidence!** In fact, the constraints in linear programs always lead to a situation in which the set of feasible points can be described as an intersection of half-spaces. Such an intersection of half-spaces is called a **(convex) polytope**. Furthermore, in arbitrary dimensions we can consider the level sets

$$\{(x_1, \ldots, x_n) : c_1x_1 + \ldots + c_nx_n = c\}$$

in which the objective function $f = c_1x_1 + \ldots + c_nx_n$ just assumes the value c. Such a level-set describes an object called a $n-1$-dimensional **hyperplane**,[2] and we need to find among all the possible hyperplanes (each described by one particular value of c) the ones that intersect with the feasible set, and among these the one with the smallest value of c (in minimization problems – in maximization problems we need to find the largest such c).

The crucial point now is that it can be shown that the optimum hyperplane (with the largest value of c) always intersects with the feasible set in a "corner point" (we will make this notion more rigorous later on in Section 4.4.2). In other words: **for finding the optimum point it suffices to inspect the corner points of a convex polytope (the feasible set), of which there are only finitely many!** This is the observation on which the simplex algorithm is built: it identifies corner points of the polytope and from one corner point tries to find the next corner point at which the objective function assumes a truly smaller cost.

[2] You might wonder why such a hyperplane is called $n-1$-dimensional when we are working in the $\mathbb{R}^n$. In our two-dimensional example the hyperplane is actually a line. In three dimensions it would be a plane. A line is considered a one-dimensional object, as you have only one degree of freedom. A plane is considered a two-dimensional object, as you only have two degrees of freedom and so forth.

In the next sections we will make these things more rigorous and discuss the operation of the simplex algorithm in some detail. Much of the discussion will be of a more algebraic nature, but we will make an effort to relate the discussion to this geometric view.

4.4 Basic Solutions and Basic Feasible Solutions

Let us go back to an abstract linear program in standard form:

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \tag{4.8}$$

where $\mathbf{x} \in \mathbb{R}^n$ is an n-dimensional vector of decision variables, $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the $m \times n$ coefficient matrix and the right-hand side $\mathbf{b} \in \mathbb{R}^m$ is an m-dimensional vector. We make two main assumptions for the remainder of this chapter:

A1 We have $n \geq m$, i.e. more decision variables than constraints.

A2 The rank of matrix $\mathbf{A}$ is m. This is called the **full-rank assumption** – if the rank of $\mathbf{A}$ were smaller than m, then either some of the rows of $\mathbf{A}$ contradict each other (and the whole problem has no solutions), or some of the rows of $\mathbf{A}$ and $\mathbf{b}$ would be redundant (i.e. linear combinations of other rows) and can be eliminated.

This is a good time to again consult Appendix A and make sure you have reviewed its contents. We will use several of the facts collected there.

Definition 4.2 (Feasible solution)**.** *A vector $\mathbf{x} \in \mathbb{R}^n$ which satisfies*

$$\begin{array}{rcl} \mathbf{A}\mathbf{x} & = & \mathbf{b} \\ \mathbf{x} & \geq & \mathbf{0} \end{array}$$

*is called a **feasible solution** or a **feasible vector**.*

Definition 4.3 (Optimal feasible solution)**.** *A vector $\mathbf{x} \in \mathbb{R}^n$ which is a feasible solution of*

$$\begin{array}{rcl} \mathbf{A}\mathbf{x} & = & \mathbf{b} \\ \mathbf{x} & \geq & \mathbf{0} \end{array}$$

and for which

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{c}^T\mathbf{y}$$

*holds for all feasible solutions $\mathbf{y}$ is called an **optimal feasible solution**.*

Note that optimal feasible solutions need not be unique, there can be many feasible solutions $\mathbf{x}$ which take the same minimal value. It can also happen that no optimal feasible solution exists despite there being many feasible solutions: this can happen when the feasible set is un-bounded (compare Definition 4.1).

Our next object of investigation is the (vectorial) equality constraint

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.9}$$

Because of our assumptions for the standard problem 4.8, matrix $\mathbf{A}$ has m linearly independent columns and to simplify presentation we assume that it is just the first m columns of $\mathbf{A}$ which are linear independent. If we denote by $\mathbf{B}$ the sub-matrix made up by the first m columns of $\mathbf{A}$, then we can write $\mathbf{A}$ in the form

$$\mathbf{A} = [\mathbf{B}, \mathbf{C}]$$

where the matrix $\mathbf{C} \in \mathbb{R}^{m \times (n-m)}$ collects the remaining columns of $\mathbf{A}$. By construction, $\mathbf{B}$ is an $m \times m$ rectangular matrix which, because of the rank assumption, is also non-singular / invertible (recall that a quadratic $m \times m$ matrix with full rank m is always invertible). Hence, it is possible to find a unique vector $\mathbf{x}_B \in \mathbb{R}^m$ which solves

$$\mathbf{B}\mathbf{x}_B = \mathbf{b}$$

Then clearly for the vector $\mathbf{x} = [\mathbf{x}_B^T, \mathbf{0}]^T \in \mathbb{R}^n$ (which is obtained by appending $n-m$ zeros below the column vector $\mathbf{x}_B$) we have

$$\mathbf{A} \cdot \mathbf{x} = [\mathbf{B}, \mathbf{C}] \cdot [\mathbf{x}_B^T, \mathbf{0}]^T = \mathbf{b}$$

Generalizing this argument, we have found a recipe to find further solutions to $\mathbf{A}\mathbf{x} = \mathbf{b}$:

- Pick any m linearly independent columns of $\mathbf{A}$, collect them into a matrix $\mathbf{B}$.
- Find a vector $\mathbf{x}_B$ solving $\mathbf{B}\mathbf{x}_B = \mathbf{b}$
- Then the vector $\mathbf{x} \in \mathbb{R}^n$ which contains the entries of $\mathbf{x}_B$ corresponding to the columns chosen in the first step and zero elsewhere, is a solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Definition 4.4 (Basic solution)**.** *The matrix $\mathbf{A}$ of our standard problem 4.8 is of rank m and has an $m \times m$ sub-matrix $\mathbf{B}$ with linearly independent columns (and thus of rank m too). We form a vector $\mathbf{x}$ by filling the entries of the (unique) m-dimensional solution vector $\mathbf{x}_B$ to the problem $\mathbf{B}\mathbf{x}_B = \mathbf{b}$ into an all-zero vector at the places corresponding to the columns chosen for $\mathbf{B}$. Then $\mathbf{x}$ contains at least $n - m$ zeros. Then this vector $\mathbf{x}$ is a solution to $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ and is called a **basic solution**. The components of $\mathbf{x}$ that are associated with $\mathbf{x}_B$ are called **basic variables**. If one or more of the basic variables is zero, then the solution is called a **degenerate basic solution**.*

Definition 4.5 (Basic feasible solution)**.** *A vector $\mathbf{x}$ that is both a feasible solution (compare Definition 4.2) and a basic solution (Definition 4.4) is called a **basic feasible solution** (bfs). If some of the basic variables of a basic feasible solution are zero, then it is called a **degenerate basic feasible solution**.*

4.4.1 Fundamental Theorem of Linear Programming

The following theorem underlies the operation of the simplex algorithm, which only considers basic feasible solutions to find an optimal solution to a linear program.

Theorem 4.1 (Fundamental theorem of linear programming)**.** *For the linear program 4.8 with the above assumptions **A1** and **A2** the following holds:*

1. *If the problem has a feasible solution, then it also has a basic feasible solution.*
2. *If the problem has an optimal feasible solution, then it also has an optimal basic feasible solution.*

Proof. (from: [13])

1. Be $\mathbf{x} = (x_1, \ldots, x_n)^T$ a feasible solution, and be $\mathbf{a}_1, \ldots, \mathbf{a}_n$ the columns of $\mathbf{A}$, i.e. $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$. Then

$$x_1\mathbf{a}_1 + \ldots + x_n\mathbf{a}_n = \mathbf{b}$$

holds. Suppose that exactly p out of the n components of $\mathbf{x}$ are non-zero (i.e. positive) and the other components are zero. To simplify notation, let us assume that just the first p components of $\mathbf{x}$ are non-zero. Then in fact we have:

$$x_1\mathbf{a}_1 + \ldots + x_p\mathbf{a}_p = \mathbf{b} \tag{4.10}$$

and we have to distinguish two different cases:

- First case: the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_p$ are linearly independent. Because of the rank assumption we must then have $p \leq m$. If $p = m$, then $\mathbf{x}$ is already a basic feasible solution and we are finished. If $p < m$ then, again because of the rank assumption, we can find at least $m - p$ further columns of $\mathbf{A}$ which, together with columns $\mathbf{a}_1$,...,$\mathbf{a}_p$, form an independent set of vectors (suppose these are columns $p+1$ to m). We can choose the components $x_{p+1}, \ldots, x_m, x_{m+1}, \ldots, x_n$ as zero and hence get a degenerate basic feasible solution.
- Second case: the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_p$ are linearly dependent. Then there is some non-trivial linear combination of $\mathbf{a}_1, \ldots, \mathbf{a}_p$ that is zero, i.e. there exist scalars $y_1, \ldots, y_p$ with at least one y_i being positive (which we can always achieve by multiplying by -1 if necessary) such that

  $$y_1\mathbf{a}_1 + \ldots + y_p\mathbf{a}_p = 0$$

  When we multiply this equation by an arbitrary scalar ϵ and subtract it from Equation 4.10 we get

  $$(x_1 - \epsilon y_1)\mathbf{a}_1 + \ldots + (x_p - \epsilon y_p)\mathbf{a}_p = \mathbf{b}$$

  If we denote $\mathbf{y} = (y_1, \ldots, y_p, 0, \ldots, 0)^T \in \mathbb{R}^n$ then the previous equation says that

  $$\mathbf{x} - \epsilon\mathbf{y}$$

  is a solution of the equality constraint for every ϵ (but not necessarily fulfilling the non-negativity constraint yet). We have assumed that at least one y_i is positive, so at least one component of the vector $\mathbf{x} - \epsilon\mathbf{y}$ decreases when ϵ is increased. By specifically setting

  $$\epsilon^* = \min\left\{\frac{x_i}{y_i} : y_i > 0\right\}$$

  we get a solution $\mathbf{x} - \epsilon^*\mathbf{y}$ of the equality constraint which:
  - is a feasible solution,[3] and
  - which has a zero in one of the components with a positive y_i

  So we have found a new feasible solution $\mathbf{x} - \epsilon^*\mathbf{y}$ which has at most $p - 1$ positive variables. By repeating this elimination procedure we can can find a solution vector which only involves linearly independent columns of $\mathbf{A}$ (of which there are at most m). Then the first case applies.

2. Be $\mathbf{x} = (x_1, \ldots, x_n)^T$ an optimal feasible solution with p positive variables, which again for convenience are just the first p ones, the others are zero. Again we have two cases to consider:

   - When $\mathbf{a}_1, \ldots, \mathbf{a}_p$ are linearly independent, then similarly to the first case in the previous proof we can turn $\mathbf{x}$ into a (possibly degenerate) basic feasible solution by adjoining additional linearly independent column vectors from $\mathbf{A}$ and setting their variables to zero, not changing the value of the objective function.

[3] All the x_i are non-negative, ϵ^* is non-negative, for those j with $y_j < 0$ clearly $x_j - \epsilon^* y_j \geq 0$, and for those j with $y_j > 0$ the specific choice of ϵ^* ensures non-negativity.

- When $\mathbf{a}_1, \ldots, \mathbf{a}_p$ are linearly dependent, we also proceed in the same way as in the second case of the previous proof, i.e. we construct new solutions of the equality constraint in the form

  $$\mathbf{x} - \epsilon \mathbf{y}$$

  For any choice of ϵ the value of the objective function becomes

  $$\mathbf{c}^T(\mathbf{x} - \epsilon \mathbf{y}) = \mathbf{c}^T \mathbf{x} - \ \epsilon \, \mathbf{c}^T \mathbf{y}$$

  We claim that $\mathbf{c}^T \mathbf{y} = 0$. Indeed, suppose otherwise that $\mathbf{c}^T \cdot \mathbf{y} \neq 0$. Then we can fix the sign of ϵ so that $\epsilon \ \mathbf{c}^T \mathbf{y}$ becomes negative. And by choosing the magnitude of ϵ to be small, we can ensure that $\mathbf{x} - \epsilon \mathbf{y}$ is a feasible vector. Putting these two things together, we have found a feasible vector for which the value of the objective function is actually smaller than $\mathbf{c}^T \mathbf{x}$, which violates our assumption of optimality of $\mathbf{x}$. Hence $\mathbf{c}^T \mathbf{y} = 0$.

□

This theorem tells us that we only have to inspect all basic feasible solutions, of which there is a finite number.

Problem 4.4.1 (Maximum number of basic feasible solutions)**.**
What is an upper bound on the maximum number of basic feasible solutions?

4.4.2 Excursion: Convex Polytopes and Basic Feasible Solutions

While not strictly necessary for further development, it is also interesting to connect the purely algebraic developments so far (which tell us that we only need to consider linearly independent selections of columns of $\mathbf{A}$) back to the geometric considerations introduced in Section 4.3. In that section we have argued heuristically that the set of feasible solutions of an LP is in fact a convex polytope, and that an optimal feasible solution can be found in a "corner" of this polytope. Here introduce a suitable notion for such a "corner" and relate it to basic feasible solutions.

Definition 4.6 (Extreme point)**.** *Be C a convex set and $\mathbf{x} \in C$ a point in C. Point $\mathbf{x}$ is an **extreme point** of C when there are no two distinct points $\mathbf{y}$ and $\mathbf{z}$ (both different from $\mathbf{x}$) and no $0 < \lambda < 1$ such that $\mathbf{x} = \lambda \mathbf{y} + (1 - \lambda)\mathbf{z}$, i.e. $\mathbf{x}$ cannot be written as a convex combination of other points.*

As an example, a triangle together with its interior is a convex set and its corners are the extreme points.

Theorem 4.2 (Equivalence of extreme points and basic feasible solutions [13])**.** *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ with rank m and $\mathbf{b} \in \mathbb{R}^m$. Let K be the convex polytope of all points $\mathbf{x}$ fulfilling*

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

A vector $\mathbf{x}$ is an extreme point of K if and only if $\mathbf{x}$ is a basic feasible solution.

Proof. As usual, we have to show two directions.

‘ $\Longleftarrow$ ’ : suppose that $\mathbf{x} = (x_1 \ldots, x_m, 0, \ldots, 0)$ is a basic feasible solution (where again for convenience we assume that we are working with just the first m columns of $\mathbf{A}$ and that these are linearly independent, which can always be achieved through re-ordering the columns), i.e.

$$x_1\mathbf{a}_1 + \ldots + x_m\mathbf{a}_m = \mathbf{b}$$

with the remaining components of $\mathbf{x}$ being zero. Assume furthermore that $\mathbf{x}$ can be expressed as a convex combination of two other points $\mathbf{y}$ and $\mathbf{z}$, $\mathbf{y} \neq \mathbf{z}$, from K, i.e. that

$$\mathbf{x} = \alpha\mathbf{y} + (1-\alpha)\mathbf{z}$$

for some $0 < \alpha < 1$. Since $\mathbf{y}$ and $\mathbf{z}$ are from K, they are also feasible solutions and have non-negative components. and because α is positive, the last $n - m$ components of both $\mathbf{y}$ and $\mathbf{z}$ must be zero. Therefore:

$$\begin{aligned} y_1\mathbf{a}_1 + \ldots + y_m\mathbf{a}_m &= \mathbf{b} \\ z_1\mathbf{a}_1 + \ldots + z_m\mathbf{a}_m &= \mathbf{b} \end{aligned}$$

hold simultaneously. From the definition of linear independence this readily implies that $\mathbf{x} = \mathbf{y} = \mathbf{z}$ holds (argue this!).

‘ $\Longrightarrow$ ’ : Be $\mathbf{x} = (x_1, \ldots, x_n)$ an extreme point of K, and assume that $\mathbf{x}$ has k non-zero (i.e. strictly positive) components, which conveniently are the first ones, so that

$$x_1\mathbf{a}_1 + \ldots + x_k\mathbf{a}_k = \mathbf{b}$$

holds. For this solution to be a basic feasible solution we need that $\mathbf{a}_1$,…,$\mathbf{a}_k$ are linearly independent. Suppose they are not, then there exist $y_1, \ldots, y_k$ not simultaneously zero such that

$$y_1\mathbf{a}_1 + \ldots + y_k\mathbf{a}_k = \mathbf{0}$$

If we denote $\mathbf{y} = (y_1, \ldots, y_k, 0, \ldots, 0)$ then, since $x_i > 0$ for $i \in \{1, \ldots, k\}$, one can find an $\epsilon > 0$ such that both

$$\begin{aligned} \mathbf{x} + \epsilon\mathbf{y} &\geq \mathbf{0} \\ \mathbf{x} - \epsilon\mathbf{y} &\geq \mathbf{0} \end{aligned}$$

hold simultaneously and thus both these vectors would be feasible and are contained in K. Now note that we could then write

$$\mathbf{x} = \frac{1}{2}(\mathbf{x} + \epsilon\mathbf{y}) + \frac{1}{2}(\mathbf{x} - \epsilon\mathbf{y})$$

and can thus be represented as a convex combination of two distinct points in K, which violates our assumption that $\mathbf{x}$ is an extreme point.

□

Hence, to find an optimum it suffices to inspect the extreme points (or “corner points”) of the convex polytope containing the feasible solutions. There is only a finite number of these.

4.5 The Simplex Algorithm

In this section we explain the basic ideas of the simplex algorithm. The simplex algorithm takes an LP in standard form (Equation 4.8) and either outputs an optimal solution or determines that the feasible set is unbounded or the problem is infeasible and stops. With an unbounded feasible set there is the risk that the objective function might take on arbitrarily low values.[4]

We will discuss the algorithm under simplifying assumptions and leave out a number of details. Furthermore, we will only focus on a generic version of the simplex algorithm for general LPs. When an LP has more structure (in particular, if matrix $\mathbf{A}$ has more structure), then more refined and efficient versions of the simplex algorithm can be found.

As before, we consider the problem given in Equation 4.8 with the two assumptions given there. We furthermore make the simplifying assumption that all the basic feasible solutions $\mathbf{x}$ which occur in our algorithm description are non-degenerate, i.e. they have exactly m positive components x_i, and all other components are zero. This assumption is called the **non-degeneracy assumption**. It is possible to remove this assumption, but this would lead to a more technical treatment of the algorithm.

The m positive components of a basic feasible solution correspond to m linearly independent column vectors of the problem matrix $\mathbf{A}$ (or a matrix equivalent to it, see below). We refer to these m vectors as a **basis**, since they span $\mathbb{R}^m$ (which means that every vector $\mathbf{v} \in \mathbb{R}^m$ can be written in a unique way as a linear combination of the basis vectors). The simplex algorithm modifies the basis during its operation. More precisely:

- We find an initial basic feasible solution, giving us an initial basis $\mathbf{a}_1, \ldots, \mathbf{a}_m$.
- For a given basis $\mathbf{a}_1, \ldots, \mathbf{a}_m$, accompanying feasible solution $\mathbf{x}$ and objective value $z_0 = \mathbf{c}^T\mathbf{x}$ we check the following:
  - Can we find a new column vector $\mathbf{a}_p$, $p \in \{m+1, \ldots, n\}$, from the remaining columns of $\mathbf{A}$ to replace one of the basis vectors $\mathbf{a}_1$,…,$\mathbf{a}_m$ so that the resulting new basic feasible solution has smaller cost $z < z_0$? If not, then we conclude that the vector $\mathbf{x}$ (for basis $\mathbf{a}_1, \ldots, \mathbf{a}_m$) is already an optimal feasible solution and stop.
  - Otherwise, we have to bring $\mathbf{a}_p$ into the basis and identify one of the basis members $\mathbf{a}_1, \ldots, \mathbf{a}_m$ to remove from the basis. During this step it may turn out that the set of feasible solutions is unbounded. In this case the algorithm stops.

We next discuss a number of individual bits and pieces of the algorithm, then tie everything together and give an example of how the algorithm operates.

4.5.1 Tableaus and Pivoting

In the following I assume that you are familiar with solving linear systems of equations and the manipulations done when carrying out the Gaussian elimination algorithm (or getting the matrix into row echolon form). These manipulations rest on the basic principle that one may replace one equation by a sum of a non-zero multiple of itself and multiples of other equations without changing the solution set.

[4] It is a hard problem to figure out whether this actually occurs or not – the simplex algorithm makes no attempt to solve this problem and therefore stops.

We consider the equality constraint $\mathbf{Ax} = \mathbf{b}$, or in more explicit form:

$$\begin{array}{ccccccccc} a_{1,1}x_1 & + & a_{1,2}x_2 & + & \ldots & + & a_{1,n}x_n & = & b_1 \\ a_{2,1}x_1 & + & a_{2,2}x_2 & + & \ldots & + & a_{2,n}x_n & = & b_2 \\ \ldots & & & & & & & & \\ a_{m,1}x_1 & + & a_{m,2}x_2 & + & \ldots & + & a_{m,n}x_n & = & b_m \end{array}$$

When running the Gaussian algorithm / reduction to row echolon form, it is common to represent this as a **tableau** in the following form:

$$\left|\begin{array}{ccccccccc} a_{1,1} & a_{1,2} & a_{1,3} & \ldots & a_{1,m} & a_{1,m+1} & \ldots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \ldots & a_{2,m} & a_{2,m+1} & \ldots & a_{2,n} & b_2 \\ \ldots & & & & & & & & \\ a_{m,1} & a_{m,2} & a_{m,3} & \ldots & a_{m,m} & a_{m,m+1} & \ldots & a_{m,n} & b_m \end{array}\right|$$

When the first m columns of this equation system are linearly independent, one normally applies the steps of Gaussian elimination to bring this tableau into the following form:

$$\left|\begin{array}{ccccccccc} y_{1,1} & * & * & \ldots & * & * & \ldots & * & * \\ 0 & y_{2,2} & * & \ldots & * & * & \ldots & * & * \\ \ldots & & & & & & & & \\ 0 & 0 & 0 & \ldots & y_{m,m} & * & \ldots & * & * \end{array}\right|$$

where the "diagonal elements" $y_{i,i}$ ($1 \leq i \leq m$) are non-zero. By continuing to apply the same kind of operations and multiplying each row with a suitable constant, it is also possible to reach a tableau with the following form:

$$\left|\begin{array}{ccccccccc} 1 & 0 & 0 & \ldots & 0 & y_{1,m+1} & \ldots & y_{1,n} & y_{0,1} \\ 0 & 1 & 0 & \ldots & 0 & y_{2,m+1} & \ldots & y_{2,n} & y_{0,2} \\ \ldots & & & & & & & & \\ 0 & 0 & 0 & \ldots & 1 & y_{m,m+1} & \ldots & y_{m,n} & y_{0.m} \end{array}\right| \tag{4.11}$$

where the $y_{0,i}$ are the values on the right-hand side after applying the steps and where the first m columns have been "cleaned out" (i.e. they contain only one entry with 1 and all other entries are zero). In this "canonical form" the first m variables $x_1, \ldots, x_m$ related to the first m columns are the basic variables, and the other $n-m$ variables are the non-basic variables. Then clearly the basic feasible solution corresponding to this tableau is:

$$\begin{array}{rcl} x_1 & = & y_{0,1} \\ x_2 & = & y_{0,2} \\ \ldots & & \\ x_m & = & y_{0,m} \\ x_{m+1} & = & x_{m+2} = \ldots = x_n = 0 \end{array}$$

and the basis are just the unit vectors $\mathbf{e}_1, \ldots, \mathbf{e}_m$.

Now suppose we have decided that we want to change our basis by replacing one of the basis vectors $\mathbf{e}_p$ (for $1 \leq p \leq m$) against a non-basis vector $\mathbf{y}_q = (y_{1,q}, y_{2,q}, \ldots, y_{m,q})^T$ (for $m+1 \leq q \leq n$), thus turning the basic variable x_p into a non-basic variable and turning the non-basic variable x_q into a basic one. When $y_{p,q}$ is non-zero, then this can be accomplished by:

- first multiplying the p-th row by $\frac{1}{y_{p,q}}$ so that in the resulting tableau the coefficient $y_{p,q}$ is replaced by 1.
- then subtract suitable multiples of row p from each other row $j \neq p$ so that as a result row j has a zero entry in column q.

As a result, the q-th column in the new tableau is equal to the unit vector $\mathbf{e}_p$, and the columns for the other basic variables $x_1, \ldots, x_{p-1}, x_{p+1}, \ldots, x_m$ are not affected at all. The coefficients $y'_{i,j}$ of the resulting tableau can be expressed as:

$$y'_{i,j} = \begin{cases} y_{i,j} - \frac{y_{p,j}}{y_{p,q}} y_{i,q} & : \quad i \neq p \\ \frac{y_{p,j}}{y_{p,q}} & : \quad i = p \end{cases}$$

When $y_{p,q}$ is zero, then it is not possible to replace the basic variable x_p against the non-basic variable x_q. The element $y_{p,q}$ is also called the **pivot element** and the entire procedure for exchanging a basic variable against a non-basic variable is called **pivoting**.

Let us work in some detail through a pivoting example. We consider the following set of equation constraints:

$$\begin{array}{rcrcrcrcrcrcr} 2x_1 & + & 6x_2 & + & x_3 & + & 2x_4 & + & 5x_5 & + & 8x_6 & = & 5 \\ 5x_1 & + & x_2 & - & 2x_3 & + & 2x_4 & - & 9x_5 & + & 13x_6 & = & 7 \\ x_1 & + & 4x_2 & + & 3x_3 & & & + & 2x_5 & + & 4x_6 & = & 9 \end{array}$$

The resulting tableau is

$$\left|\begin{array}{rrrrrrr} 2 & 6 & 1 & 2 & 5 & 8 & 5 \\ 5 & 1 & -2 & 2 & -9 & 13 & 7 \\ 1 & 4 & 3 & 0 & 2 & 4 & 9 \end{array}\right|$$

Our first aim is to reach the form 4.11 for the first three columns. We start by dividing the first row by 2 to get a 1 on the diagonal place, giving:

$$\left|\begin{array}{rrrrrrr} 1 & 3 & 1/2 & 1 & 5/2 & 4 & 5/2 \\ 5 & 1 & -2 & 2 & -9 & 13 & 7 \\ 1 & 4 & 3 & 0 & 2 & 4 & 9 \end{array}\right|$$

We next create the required zeros in the first column by:

- Subtracting five times the first row from the second row
- Subtracting the first row from the third row

and the result is:

$$\left|\begin{array}{rrrrrrr} 1 & 3 & 1/2 & 1 & 5/2 & 4 & 5/2 \\ 0 & -14 & -9/2 & -3 & -43/2 & -7 & -11/2 \\ 0 & 1 & 5/2 & -1 & -1/2 & 0 & 13/2 \end{array}\right|$$

Next we consider the second column. We multiply the second row by $-1/14$ to get

$$\left|\begin{array}{rrrrrrr} 1 & 3 & 1/2 & 1 & 5/2 & 4 & 5/2 \\ 0 & 1 & 9/28 & 3/14 & 43/28 & 1/2 & 11/28 \\ 0 & 1 & 5/2 & -1 & -1/2 & 0 & 13/2 \end{array}\right|$$

and create the required zeros in the second column by:

- Subtracting three times the second row from the first row.
- Subtracting the second row from the third row.

to get:

$$\left|\begin{array}{ccccccc} 1 & 0 & -13/28 & 5/14 & -59/28 & 5/2 & 37/28 \\ 0 & 1 & 9/28 & 3/14 & 43/28 & 1/2 & 11/28 \\ 0 & 0 & 61/28 & -17/14 & -57/28 & -1/2 & 171/28 \end{array}\right|$$

Cleaning out the third column in a similar fashion gives the end result

$$\left|\begin{array}{ccccccc} 1 & 0 & 0 & 6/61 & -155/61 & 146/61 & 160/61 \\ 0 & 1 & 0 & 24/61 & \boxed{112/61} & 35/61 & -31/61 \\ 0 & 0 & 1 & -34/61 & -57/61 & -14/61 & 171/61 \end{array}\right|$$

and we can read off the basic solution for the first three variables x_1, x_2 and x_3 as:

$$\begin{aligned} x_1 &= \frac{160}{61} \\ x_2 &= \frac{-31}{61} \\ x_3 &= \frac{171}{61} \end{aligned}$$

(which is not a basic feasible solution though, but that does not matter for our example), and the corresponding basis members are $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$.

We now want to use the pivoting procedure to swap the basic variable x_2 against the non-basic variable x_5, which is possible since $y_{2,5} = 112/61$ in this tableau (highlighted) is non-zero. We multiply the second row by $61/112$ to get the new tableau:

$$\left|\begin{array}{ccccccc} 1 & 0 & 0 & 6/61 & -155/61 & 146/61 & 160/61 \\ 0 & 61/112 & 0 & 3/14 & 1 & 5/16 & -31/112 \\ 0 & 0 & 1 & -34/61 & -57/61 & -14/61 & 171/61 \end{array}\right|$$

and clear out the fifth column by:

- Adding the second row $\frac{155}{61}$ times to the first row.
- Adding the second row $\frac{57}{61}$ times to the third row.

giving the result tableau

$$\left|\begin{array}{ccccccc} 1 & 155/112 & 0 & 9/14 & 0 & 51/16 & 215/112 \\ 0 & 61/112 & 0 & 3/14 & 1 & 5/16 & -31/112 \\ 0 & 57/112 & 1 & -5/14 & 0 & 1/16 & 285/112 \end{array}\right|$$

and a new basic solution

$$\begin{aligned} x_1 &= \frac{215}{112} \\ x_3 &= \frac{285}{112} \\ x_5 &= \frac{-31}{112} \end{aligned}$$

So, now we have the new set of basic variables x_1, x_3 and x_5 with corresponding basis vectors $\mathbf{e}_1$, $\mathbf{e}_3$ and $\mathbf{e}_2$, respectively. You can see that the mechanics of the pivoting procedure do not differ from the sort of calculations you need to do when solving a linear system.

Problem 4.5.1 (Pivoting)**.**
The canonical form of the system is given by

$$\begin{array}{rrrrrrrrrrr} x_1 & & & + & x_4 & + & x_5 & - & x_6 & = & 5 \\ & x_2 & & + & 2x_4 & - & 3x_5 & + & x_6 & = & 3 \\ & & x_3 & - & x_4 & + & 2x_5 & - & x_6 & = & -1 \end{array}$$

The basic variables are x_1, x_2 and x_3. We want to exchange these step by step so that in the end we will have a tableau with x_4, x_5 and x_6 as the basic variables. Give the final tableau.

Now that we have understood the mechanics of swapping a basic variable against a non-basic variable (and thus what needs to be done to change the current basis) we will consider next the following two questions:

- Suppose we have chosen a non-basic variable x_q, which basic variable x_p will be replaced by it?
- Which non-basic variable x_q should we choose for bringing into the basis?

We continue with the first of these questions.

4.5.2 Which Basis Element to Remove from Basis?

In the previous section we have seen how a basic variable from a basic solution $\mathbf{x}$ can be swapped against a non-basic variable to get a new basic solution $\mathbf{x}'$. The new basic solution will by design satisfy the equality constraint $\mathbf{A}\mathbf{x}' = \mathbf{b}$, but in general there is a-priori no guarantee that non-negativity of the basic variables will be preserved (i.e. while $\mathbf{x} \geq \mathbf{0}$ may hold, it is not guaranteed that $\mathbf{x}' \geq \mathbf{0}$ will be true after the pivoting procedure). In this section we will see that for an arbitrarily chosen non-basic variable x_q with non-zero basis vector $\mathbf{a}_q$ it is possible either:

- to find a basic variable x_p from the basis to replace against, so that the non-negativity condition for the new basic solution $\mathbf{x}'$ is satisfied, or
- to conclude that the set of feasible solutions is unbounded, in which case the algorithm stops.

There is no other alternative.

To simplify notations, we assume that the current basic feasible solution is $\mathbf{x} = (x_1, \ldots, x_m, 0, \ldots, 0)$ so that

$$x_1\mathbf{a}_1 + \ldots + x_m\mathbf{a}_m = \mathbf{b} \tag{4.12}$$

holds, where again the $\mathbf{a}_i$ are the columns of the coefficient matrix $\mathbf{A}$.[5] Furthermore, since we assume that we are working with a basic feasible solution the values $x_1, \ldots, x_m$ are non-negative, and because of the non-degeneracy assumption we assume in fact that they are all positive.

We are given a non-basic variable x_q ($m{+}1 \leq q \leq n$) and the corresponding column vector $\mathbf{a}_q \neq \mathbf{0}$ from matrix $\mathbf{A}$, and since the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m$ are actually a basis of $\mathbb{R}^m$, we can express $\mathbf{a}_q$ as a linear combination of $\mathbf{a}_1, \ldots, \mathbf{a}_m$:

$$\mathbf{a}_q = y_{1,q}\mathbf{a}_1 + y_{2,q}\mathbf{a}_2 + \ldots + y_{m,q}\mathbf{a}_m$$

[5] Note that the matrix $\mathbf{A}$ and the right-hand side vector $\mathbf{b}$ given in the original LP change over time when we apply pivoting procedures. The matrix $\mathbf{A}$ and the vector $\mathbf{b}$ that we refer to in this section are the "current" ones, i.e. the ones we got after carrying out the pivoting operations a number of times, so they can differ from the original matrix/right-hand side.

where at least one of the $y_{i,q}$ is non-zero. When we multiply this equation by some scalar $\epsilon \geq 0$ and subtract it from Equation 4.12 we get

$$(x_1 - \epsilon y_{1,q})\mathbf{a}_1 + \ldots + (x_m - \epsilon y_{m,q})\mathbf{a}_m + \epsilon \mathbf{a}_q = \mathbf{b} \tag{4.13}$$

which holds for arbitrary $\epsilon \geq 0$. Since all the x_i are strictly positive, it is possible to choose an ϵ that is positive but small enough so that all the terms $(x_i - \epsilon y_{i,q})$ are still strictly positive and 4.13 gives a feasible solution which is not basic (because it involves $m+1$ variables with a positive contribution). Now everything depends on the sign of the values $y_{i,q}$:

- If one or more of the $y_{i,q}$ are strictly positive, then by choosing[6]

$$\epsilon = \min_i \left\{ \frac{x_i}{y_{i,q}} : y_{i,q} > 0 \right\} = \min_i \left\{ \frac{y_{0,i}}{y_{i,q}} : y_{i,q} > 0 \right\} \tag{4.14}$$

  we can make at least one of the coefficients $(x_i - \epsilon y_{i,q})$ zero while having a positive coefficient for $\mathbf{a}_q$. If exactly one of these coefficients, say coefficient $(x_p - \epsilon y_{p,q})$ for $1 \leq p \leq m$, becomes zero for this choice, then we have found a new basic feasible solution in which vector $\mathbf{x}_p$ is replaced by vector $\mathbf{x}_q$ (note that the coefficients for vectors $\mathbf{a}_1, \ldots, \mathbf{a}_{p-1}$,$\mathbf{a}_{p+1}$,$\ldots$,$\mathbf{a}_m$,$\mathbf{a}_q$ are strictly positive for our choice of ϵ) and we can proceed to use the pivoting procedure to swap these two and create an updated tableau. If more than one of the coefficients $(x_i - \epsilon y_{i,q})$ becomes zero for our choice of ϵ, then we can pick any of these for swapping with $\mathbf{a}_q$, but the new solution will be a degenerate basic feasible solution.

- If none of the $y_{i,q}$ is positive then at least one $y_{i,q}$ is strictly negative, say $y_{p,q}$, and by making ϵ arbitrarily large we can make the coefficient $(x_p - \epsilon y_{p,q})$ arbitrarily large in Equation 4.13. Hence, there exists feasible solutions with arbitrarily large coefficients and therefore the feasible set is unbounded, the algorithm stops.

In summary, we have argued that if we are given a current basic feasible solution $(x_1, \ldots, x_m, 0, \ldots, 0)$ and a new non-zero non-basis vector $\mathbf{a}_q$ to bring into the basis, we can either identify a basis vector $\mathbf{a}_p$ to remove from the basis such that the pivoting procedure is guaranteed to give a new basic feasible solution $\mathbf{x}'$, or we can conclude that the set of feasible solutions is unbounded. There are no other options.

4.5.3 Which New Non-Basis Element to Bring into Basis?

In the previous section we have seen how, for a given non-basis vector $\mathbf{a}_q$, we can find a basis vector $\mathbf{a}_p$ against which to swap (so that we get a new basic feasible solution), or conclude that the feasible set is unbounded and stop.

We next figure out how to choose the non-basis vector $\mathbf{a}_q$ to actually put into the basis. To simplify presentation, we again assume that the current basis vectors are just the first m column vectors of the coefficient matrix $\mathbf{A}$ (possibly modified by previous pivoting procedures), the current tableau is

$$\left| \begin{array}{cccccccc} 1 & 0 & 0 & \ldots & 0 & y_{1,m+1} & \ldots & y_{1,n} & y_{0,1} \\ 0 & 1 & 0 & \ldots & 0 & y_{2,m+1} & \ldots & y_{2,n} & y_{0,2} \\ \ldots & & & & & & & & \\ 0 & 0 & 0 & \ldots & 1 & y_{m,m+1} & \ldots & y_{m,n} & y_{0.m} \end{array} \right| \tag{4.15}$$

and the current basic feasible solution is $\mathbf{x}_B = (x_1, \ldots, x_m, 0, \ldots, 0)^T = (y_{0,1}, \ldots, y_{0,m}, 0, \ldots, 0)$. The current value of the objective function is

$$z_0 := \mathbf{c}_B^T \mathbf{x}_B$$

[6]Remember that in the canonical form of the tableau in Equation 4.11 we have that $x_i = y_{0,i}$ for the basic variables x_i $(1 \leq i \leq n)$.

where the cost vector $\mathbf{c}_B$ is the restriction of the cost vector $\mathbf{c}$ from our LP specification to the first m components, i.e. $\mathbf{c}_B = (c_1, \ldots, c_m)^T$. Now we want to answer the following question: suppose we change the contribution of a non-basic variable x_q ($m+1 \leq q \leq n$) by assigning a positive value to it, how will the value of the objective function change relative to z_0? To address this question, we aim to express the current costs z_0 only in terms of the non-basic variables $x_{m+1}, \ldots, x_n$.

To do this we proceed as follows. Suppose that in the tableau 4.15 we assign arbitrary values to the non-basis variables $x_{m+1}, \ldots, x_n$, then we can solve for the basic variables x_p ($1 \leq p \leq m$) as follows:

$$\begin{aligned} x_1 &= y_{0,1} - \sum_{j=m+1}^{n} y_{1,j} x_j \\ x_2 &= y_{0,2} - \sum_{j=m+1}^{n} y_{2,j} x_j \\ &\ldots \\ x_m &= y_{0,m} - \sum_{j=m+1}^{n} y_{m,j} x_j \end{aligned} \tag{4.16}$$

Plugging this into the general expression for the objective function

$$z = c_1 x_1 + \ldots + c_n x_n$$

gives us: (with remembering that $x_1 = y_{0,1}, \ldots, x_m = y_{0,m}$):

$$\begin{aligned} z &= c_1 x_1 + \ldots + c_m x_m + c_{m+1} x_{m+1} + \ldots + c_n x_n \\ &= c_1 \left(y_{0,1} - \sum_{j=m+1}^{n} y_{1,j} x_j \right) + \ldots + c_m \left(y_{0,m} - \sum_{j=m+1}^{n} y_{m,j} x_j \right) + c_{m+1} x_{m+1} + \ldots + c_n x_n \\ &= z_0 + \left(c_{m+1} - \sum_{j=1}^{m} c_j y_{j,m+1} \right) x_{m+1} + \left(c_{m+2} - \sum_{j=1}^{m} c_j y_{j,m+2} \right) x_{m+2} + \ldots + \left(c_n - \sum_{j=1}^{m} c_j y_{j,n} \right) x_n \\ &=: z_0 + (c_{m+1} - z_{m+1}) x_{m+1} + \ldots + (c_n - z_n) x_n \end{aligned} \tag{4.17}$$

where in the last equation we have used the abbreviation

$$z_k = \sum_{j=1}^{m} c_j y_{j,k} \tag{4.18}$$

for $m+1 \leq k \leq n$. We introduce the further abbreviation

$$r_k = c_k - z_k \tag{4.19}$$

for $m+1 \leq k \leq n$. Now observe that Equation 4.17 is already what we have been looking for: if any of the terms r_k (which only depend on the entries of the current tableau and the cost vector $\mathbf{c}$) is strictly negative, say: term r_q for $m+1 \leq q \leq n$, then by making x_q positive and all other x_r ($m+1 \leq r \leq n$, $r \neq q$) zero, we can possibly decrease the value of the objective function. So, x_q is a good candidate for bringing into the basis. If there are several r_k with strictly negative values, then one could select among these the non-basic variable for which r_k has the largest magnitude. The next theorem states that if any of the r_k ($m+1 \leq k \leq n$) is strictly negative, then the value of the objective function will indeed decrease:

Theorem 4.3 (Improvement of basic feasible solution [13])**.** *We are given a non-degenerate basic feasible solution, say* $(x_1, \ldots, x_m, 0, \ldots, 0)$*, with corresponding objective value* z_0*, and one of the* $\{r_{m+1}, r_{m+2}, \ldots, r_n\}$

is strictly negative, say r_j ($m+1 \leq j \leq n$). Then there is a feasible solution (and thus, by theorem 4.1 also a basic feasible solution) with objective value $z < z_0$. In particular, if the non-basis vector $\mathbf{a}_j$ can be swapped against some vector in the basis (using the pivoting procedure), then $z < z_0$ holds for this new solution, otherwise the feasible set is unbounded.

Proof. The last part (the new vector $\mathbf{a}_j$ cannot be swapped) follows from our considerations in Section 4.5.2.

Without loss of generality suppose that our given basic feasible solution is $(x_1, \ldots, x_m, 0, \ldots, 0)^T$, and furthermore that $r_{m+1} < 0$. It is possible to construct a new feasible solution $(x'_1, \ldots, x'_m, x'_{m+1}, 0, \ldots, 0)$ with completely positive entries by choosing a small enough positive x'_{m+1} and use Equations 4.16 to calculate updated values for the x_i (for $1 \leq i \leq m$). By doing this, the equality constraint $\mathbf{Ax} = \mathbf{b}$ remains valid.

With this new solution we get from Equation 4.17 that

$$z - z_0 = r_{m+1} x'_{m+1} < 0$$

so that the new feasible solution has indeed a smaller objective value, note also that it only depends on r_{m+1} (which does not depend on any of the variables but only on the matrix/tableau and cost coefficients) and x'_{m+1}. The other variables $x'_1, \ldots, x'_m$ either decrease, remain the same or increase as x'_{m+1} is increased. Similar to the arguments in Section 4.5.2 we can increase x'_{m+1} now to the point where one or more of the x'_i ($1 \leq i \leq m$) becomes zero. If exactly one x'_i becomes zero, we have found a new basic feasible solution with an objective value that is truly smaller than the previous objective value z_0 and we can swap $\mathbf{a}_{m+1}$ into the basis. If more than one x'_i becomes zero, the same is true but now we have a degenerate basic feasible solution. If none of the x'_i decreases, then the feasible set is unbounded and the algorithm stops. □

On the other hand, if all of the r_k ($m+1 \leq k \leq n$) are positive, it is not possible anymore to find another basic solution with smaller value of the objective function, and therefore (invoking theorem 4.1) we already have found an optimal basic feasible solution and hence an optimal feasible solution to the LP. The simplex algorithm can now stop and output this solution.

To summarize what we have achieved in this section:

- For the tableau corresponding to the current basic feasible solution $\mathbf{x}$ we can calculate the values r_k (for $m+1 \leq k \leq n$) according to Equations 4.18 and 4.19 – these depend only on the entries of the current tableau and the coefficients of the objective function, and not on the current basic feasible solution $\mathbf{x}$. The coefficients r_k are also called the **relative cost coefficients**.
- When none of the r_k is strictly negative, then $\mathbf{x}$ is already an optimal basic feasible solution (and thus an optimal feasible solution).
- When one or more of the r_k are strictly negative, we can pick one of the non-basic variables corresponding to these (perhaps the one for which r_k has the largest magnitude) and swap it into the basis using the pivoting procedure. We are guaranteed that the resulting basic feasible solution $\mathbf{x}'$ has a smaller objective value than $\mathbf{x}$.

4.5.4 Finding the First Basic Feasible Solution

As the last remaining piece of the puzzle, we need to find an initial basic feasible solution, which we then can try to improve iteratively following the steps discussed in the previous sections. Sometimes an initial basic feasible solution can be guessed, but here we discuss a technique which works in general, and which actually uses a modified problem.

Recall that the constraints in the standard form read:

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

where, by multiplying a row of the linear equality constraints by minus one, we can pose the additional requirement that $\mathbf{b} \geq 0$.

To find an initial basic feasible solution $\mathbf{x}$ for the problem

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \geq 0 \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{4.20}$$

$$\tag{4.21}$$

we introduce artificial variables and consider the modified problem

$$\begin{aligned} \text{minimize}_{[\mathbf{x},\mathbf{y}]} \quad & (1,\ldots,1)\cdot\mathbf{y} = \sum_{i=1}^{m} y_i \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} + \mathbf{y} = \mathbf{b} \geq 0 \\ & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{4.22}$$

This is again an LP with coefficient matrix $[\mathbf{A}, \mathbf{I}_m]$ and decision variable vector $(x_1,\ldots,x_n,y_1,\ldots,y_m)^T$. This new LP has an obvious basic feasible solution, namely:

$$y_i = b_i$$

for $1 \leq i \leq m$. Furthermore, if our initial problem 4.20 has a feasible solution, then the modified problem 4.22 will have a solution where all y_i are zero, i.e. $\mathbf{y} = \mathbf{0}$.

We can now apply the simplex steps to the modified problem, using the basic feasible solution

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix}$$

as a starting point. If we cannot find a solution for which $\mathbf{y} = \mathbf{0}$ holds, then the original problem 4.20 has no feasible solution. On the other hand, if we can find a solution of 4.22 where all y_i are zero and none of the y_i is a basic variable anymore, then automaticallly the basis must involve m columns from $\mathbf{A}$, and this is then a basic feasible solution we can use as a starting point for the simplex algorithm applied to the original problem 4.20.

4.5.5 Summary of Simplex Algorithm

We are now in a position to put together the main steps of the simplex algorithm for solving an LP in standard form 4.8 (however, we do not deal with some technical details like the occurence of degenerate basic feasible solutions or the detection of empty feasible sets). We assume that some initial basic feasible solution $\mathbf{x} = (x_1,\ldots,x_m,0,\ldots,0)^T$ is given (if not, then run the following procedure on the modified problem introduced in Section 4.5.4 for which an initial bfs can be trivially guessed, until we have found a solution in which all the auxiliary variables y_i are zero). Then go through the following steps:

1. Form an initial tableau of the following form

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\ldots$	$\mathbf{a}_m$	$\mathbf{a}_{m+1}$	$\ldots$	$\mathbf{a}_n$	$\mathbf{b}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$\ldots$	$a_{1,m}$	$a_{1,m+1}$	$\ldots$	$a_{1,n}$	b_1
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$\ldots$	$a_{2,m}$	$a_{2,m+1}$	$\ldots$	$a_{2,n}$	b_2
$\ldots$								
$a_{m,1}$	$a_{m,2}$	$a_{m,3}$	$\ldots$	$a_{m,m}$	$a_{m,m+1}$	$\ldots$	$a_{m,n}$	b_m
c_1	c_2	c_3	$\ldots$	c_m	c_{m+1}	$\ldots$	c_n	0

   Note that we have appended a row at the bottom of the tableau, initialized with the cost vector and an initial value 0 in the last column. As we proceed with the calculations, the values in this row actually represent the relative cost coefficients r_k introduced in Section 4.5.3. The first row with the vector names does not belong to the tableau proper.

2. Now bring the current tableau into the following form[7] by using the same elementary steps as in the Gaussian elimination procedure:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\ldots$	$\mathbf{a}_m$	$\mathbf{a}_{m+1}$	$\ldots$	$\mathbf{a}_n$	$\mathbf{b}$
1	0	0	$\ldots$	0	$y_{1,m+1}$	$\ldots$	$y_{1,n}$	$y_{0,1}$
0	1	0	$\ldots$	0	$y_{2,m+1}$	$\ldots$	$y_{2,n}$	$y_{0,2}$
$\ldots$								
0	0	0	$\ldots$	1	$y_{m,m+1}$	$\ldots$	$y_{m,n}$	$y_{0.m}$
0	0	0	$\ldots$	0	r_{m+1}	$\ldots$	r_n	$-z_0$

(4.23)

   In Section 4.5.3 we have explained that the relative cost coefficients r_j ($m+1 \leq j \leq n$) are calculated according to Equations 4.18 and 4.19. It is in fact possible to update the relative cost coefficients on the go, simply by **treating the last line as an additional line in the tableau** and applying the same steps to it when running the pivoting procedure. Then the values r_k in the last row are correctly updated, and the lower-right element of this tableau then always contains the negative of the value of the objective function for the current basic feasible solution. Note that when "clearing out" the first m columns to bring the tableau into form 4.23, you also have to include the last row!

3. If each relative cost coefficient r_k ($m+1 \leq k \leq n$) is non-negative, then stop and output the current basic feasible solution $\mathbf{x}$ as an optimal solution.

4. At least one relative cost coefficient r_k is strictly negative. From among these, pick one of the corresponding variables (and the corresponding column), say x_q, as the non-basic variable to bring into the basis. We pick the column showing the "most negative" relative cost (i.e. having the largest magnitude).

5. In column q we check for all rows i but the last (i.e. $i \in \{1, 2, \ldots, m\}$) whether the element $y_{i,q}$ is strictly positive, and if so we calculate the ratio

$$\frac{y_{0,i}}{y_{i,q}}$$

   of the right-hand side of this row, $y_{0,i}$ to the considered row element $y_{i,q}$ in column q (compare Equation 4.14). If there is no i for which $y_{i,q} > 0$, then we stop and output that the problem has an unbounded set of feasible solutions. Otherwise, we pick the row p which has the smallest ratio and hence have identified $y_{p,q}$ as our new pivot element.

6. Run the pivot procedure to swap the p-th and q-th basic variables from the basis (this effectively happens by completely cleaning out column q while keeping $y_{p,q}$ as the pivot element). Go back to step 2.

[7] As before, this corresponds to the assumption that the basic variables are just $x_1, \ldots, x_m$. If other variables are basic then columns need to be swapped appropriately.

4.5.6 Example

This example is taken from [13]. We want to solve the following problem:

$$\begin{array}{ll} \text{minimize} & 4x_1 + x_2 + x_3 \\ \text{subject to} & 2x_1 + x_2 + 2x_3 = 4 \\ & 3x_1 + 3x_2 + x_3 = 3 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{array} \tag{4.24}$$

As no initial basic feasible solution springs to mind, we first have to identify one, following the recipe given in Section 4.5.4. So, we introduce two new auxiliary variables x_4 and x_5 (as we have $m = 2$ constraint equations) and consider the auxiliary problem

$$\begin{array}{ll} \text{minimize} & x_4 + x_5 \\ \text{subject to} & 2x_1 + x_2 + 2x_3 + x_4 = 4 \\ & 3x_1 + 3x_2 + x_3 + x_5 = 3 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0 \end{array}$$

for which we establish the following initial tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{b}$
2	1	2	1	0	4
3	3	1	0	1	3
0	0	0	1	1	0

where the last row corresponds to the coefficients of the involved decision variables $x_1, \ldots, x_5$ in the objective function $x_4 + x_5$. Clearly, a basic feasible solution for this tableau is $x_4 = 4$ and $x_5 = 3$. In this form, the two columns for variables x_4 and x_5 have not yet been completely cleaned out (as we wish to have for basic variables), so we clear them by:

- Subtracting row 1 from row 3.
- Subtracting row 2 from row 3.

The resulting tableau becomes

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{b}$
2	1	2	1	0	4
$\boxed{3}$	3	1	0	1	3
-5	-4	-3	0	0	-7

Our next goal is to bring one of the non-basis variables x_1, x_2 or x_3 into the basis. The last line contains the relative cost values r_1, r_2 and r_3, all of which are strictly negative, and we pick the first column as the new column to bring into the basis, as this has the largest magnitude. To pick a pivot, we calculate for each row the ratio of the positive values in the first column (for x_1, in this example both are positive) and the last column (for the right-hand side, see Equation 4.14) to get:

$$\frac{y_{0,1}}{y_{1,1}} = \frac{4}{2} = 2 \qquad , \qquad \frac{y_{0,2}}{y_{2,1}} = \frac{3}{3} = 1$$

and pick the row with the minimum such ratio as our pivot (highlighted in the previous tableau). As a result, we will swap variables x_1 and x_5 by applying the pivoting procedure. Doing this (i.e. clearing out the first

column) gives the next tableau

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{b}$
0	-1	4/3	1	$-2/3$	2
1	1	$1/3$	0	$1/3$	1
0	1	$-4/3$	0	$5/3$	-2

We still need to remove x_4 out of the basis, and the only variable which has a negative relative cost coefficient is x_3. Fortunately, we can find a pivot element (highlighted) and we can swap these two variables. The resulting tableau is

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{b}$
0	$-3/4$	1	$3/4$	$-1/2$	$3/2$
1	$5/4$	0	$-1/4$	$1/2$	$1/2$
0	0	0	1	1	0

So, we have found a new basic feasible solution of the auxiliary problem which involves variables x_1 and x_3 and not our auxiliary variables x_4 and x_5 anymore. Furthermore, the objective function value for this solution (to be found in the lower right corner) is zero, indicating that indeed the original problem will be solvable. From this last tableau we can read off the basic feasible solution

$$\begin{aligned} x_1 &= \frac{1}{2} \\ x_2 &= 0 \\ x_3 &= \frac{3}{2} \end{aligned}$$

and we can now run the second phase, in which we return to the original problem 4.24, equipped with our freshly found basic feasible solution. We set up a new tableau for this problem, in which we have to use the updated vector

$$\mathbf{b} = \begin{pmatrix} 1/2 \\ 3/2 \end{pmatrix}$$

on the right-hand side and the updated coefficient matrix $\mathbf{A}$ – this new tableau hence reads:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{b}$
0	$-3/4$	1	$3/2$
1	$5/4$	0	$1/2$
4	1	1	0

Note that in the last row we have entered the coefficients for the original objective function and an initial objective value of 0. Again, the first step is to fully clear out columns 1 and 3 for our basic variables x_1 and x_3, which we do by:

- Subtracting four times row 2 from row 3.
- Subtracting row 1 from row 3.

The resulting tableau is:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{b}$
0	$-3/4$	1	$3/2$
1	5/4	0	$1/2$
0	$-13/4$	0	$-7/2$

Only one relative cost coefficient is strictly negative, the one for variable x_2. The only eligible pivot in the second column (by virtue of being positive) is indicated in the previous tableau. So, we swap variables x_2 and

x_1 through the pivoting procedure, and the resulting tableau becomes:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{b}$
$3/5$	0	1	$9/5$
$4/5$	1	0	$2/5$
$13/5$	0	0	$-11/5$

In this tableau there are no strictly negative relative cost values, so we have found the optimal solution. In this solution we have:

$$\begin{aligned} x_1 &= 0 \\ x_2 &= 2/5 \\ x_3 &= 9/5 \end{aligned}$$

and the minimal feasible value of the objective function is $11/5$ (recall that the general intermediate tableau 4.23 contains the negative of the current value of the objective function in its lower right corner).

Note that in this example we had to go through two phases: the first phase to identify an initial basic feasible solution using an auxiliary problem, and in the second phase we have solved the original problem with a starting basic feasible solution.

Problem 4.5.2 (Further Examples)**.**
Please solve the following linear programs manually, using the simplex algorithm.

1.

$$\begin{aligned} &\text{minimize} && -2x_1 - 5x_2 \\ &\text{subject to} && x_1 \leq 4 \\ &&& x_2 \leq 6 \\ &&& x_1 + x_2 \leq 8 \\ &&& x_1 \geq 0, x_2 \geq 0 \end{aligned}$$

2.

$$\begin{aligned} &\text{minimize} && 2x_1 + 3x_2 \\ &\text{subject to} && 4x_1 + 2x_2 \geq 12 \\ &&& x_1 + 4x_2 \geq 6 \\ &&& x_1 \geq 0, x_2 \geq 0 \end{aligned}$$

Chapter 5

Network Planning Problems

In Chapter 2 we have introduced a particular class of problems called **network flow problems**, in which we have determined how to route different demand volumes in a capacitated network to optimize given performance criteria.

In this chapter we continue this line of study in a much more rigorous way and will furthermore introduce other classes of problems which occur in the planning and operation of networks. It draws on material from [14] and [16]. The focus is more on problem formulation (which is an art to some extent and certainly requires lots of practice) than on numerical solution of actual instances.

5.1 Setup and Notations

We now introduce the general setup and a more systematic notation for formulating a range of network flow and related problems. The type of setup we use is known as the "link-path formulation", as it puts the focus more on paths for flows. There are other approaches for problem formulation, but we do not concern ourselves with these.

5.1.1 Demand Volumes

We are given a network graph with N nodes and L links. A number K of demand volumes are placed on the network, and we simply index these by natural numbers without relating them in any way to the identities of the nodes involved in the demand volume. The demand volumes are described by a table in the following form:

Index	**Involved nodes**	**Demand volume**
1	1:2	h_1
2	3:7	h_2
...	...	...
K	44:1	h_K

where the first column gives the index of the demand volume, the second lists the source and destination nodes involved, and the third column gives the required long-term average data rate.

5.1.2 Paths and Links

An important thing to understand is that particularly in flow planning problems no attempt is made to *find* ideal paths for a demand volume x between i and j out of all available links as part of the problem formulation. Rather, a set of candidate paths between i and j is identified up-front, and then the problem reduces to deciding which fraction of a demand volume h_x between i and j to put on which of these paths.[1] This initial selection of paths can either be done manually by an administrator, or one could run a k-shortest path algorithm according to some cost criterion, e.g. to find the k shortest paths in terms of hop-count or per-link transmission delay.

Excursion (k-Shortest-Path Problems)

It is sometimes useful to have not only a single (the best!) connection between a source and a destination node in a network, but several, for example k paths. This can be useful in different ways:

- To have backup paths available for fault tolerance.
- For load balancing over multiple paths (compare the Equal-Cost Multipath feature of OSPF).

One typically does not want *any* k paths, but the k best ones. A heuristic approach for finding k **link-disjoint** paths goes as follows:

- Start with the full network graph, called G_0 and identify the single best path $\mathcal{P}_1$ in this using some standard shortest-path algorithm (e.g. Bellman-Ford or Dijkstra).
- Compute the next network graph G_1 by removing all links (while keeping the nodes) involved in path $\mathcal{P}_1$ from G_0.
- Identify best single best path $\mathcal{P}_2$ in G_1, and so on, until k paths have been identified or some network G_i becomes disconnected.

Some variations of this basic algorithm include:

- Instead of removing all links from a previously identified path $\mathcal{P}_i$ from network graph G_{i-1}, remove just one of the links.
- Remove all intermediate nodes of $\mathcal{P}_i$ (and their direct links) from G_{i-1}, this then creates **node-disjoint paths**.

As a result of initial path selection, we have identified P_k paths for demand volume k (with $k \in \{1, \ldots, K\}$). We now introduce decision variables x_{kp}, indicating the rate of flow of the k-demand volume ($1 \leq k \leq K$) that is being sent over the p-th available path for this demand volume ($1 \leq p \leq P_k$). With this notation the demand constraints take the form

$$\begin{aligned} \sum_{p=1}^{P_1} x_{1p} &= h_1 \\ \sum_{p=1}^{P_2} x_{2p} &= h_2 \\ &\cdots \\ \sum_{p=1}^{P_K} x_{Kp} &= h_K \end{aligned} \tag{5.1}$$

As the next ingredient we consider links. Similar to the demand volumes, these are simply numbered from 1 to L as we have L links in total in the network. The numbering is arbitrary and there does not need to be any

[1] With this, any solution to an optimization problem does not reflect the truly global optimum, but rather only the optimum with respect to the available paths – if more paths are included into the set of candidate paths then the optimum may change. However, increasing the number of paths increases computational complexity.

relationship between a link index (a number between 1 and L) and the nodes involved in this link. The links and their capacities are described by a table in the following form:

Link index	**Involved nodes**	**Link capacity**
1	6:9	c_1
2	74:75	c_2
...	...	...
L	2:4	c_L

As a final piece of input data we need to establish the relationship between links, paths and demand volumes. This is done by providing a table showing which links are involved in which paths for which volumes:

Link index	Flow $k = 1, p = 1$	...	Flow $k = 1$,$p = P_1$	Flow $k = 2, p = 1$	...	Flow $k = K, p = P_K$
1	1	...	0	1	...	0
2	0	...	1	0	...	1
...	...	...	...	...	...	...
L	0	...	1	1	...	0

where the value in row l and column k/p is used to indicate whether or not link l is used in the p-th path for demand volume k (where $1 \leq k \leq K$ and $1 \leq p \leq P_k$). When the link is used, the table contains the value 1, otherwise it contains the value 0. The table contents are represented as constants

$$\delta_{kpl} = \begin{cases} 1 & : \text{ link } l \text{ is used on the } p\text{-th path for demand volume } k \\ 0 & : \text{ otherwise} \end{cases}$$

With this notation in place we can now introduce dependent variables y_l (for $1 \leq l \leq L$) expressing the total amount of flow on link l, it is given by:

$$y_l = \sum_{k=1}^{K} \sum_{p=1}^{P_k} x_{kp} \delta_{kpl}$$

and capacity constraints then can be expressed as

$$y_l \leq c_l$$

for $1 \leq l \leq L$.

5.1.3 Revisiting Network Flow Problems

We briefly sketch how the different network flow problems we have discussed in Chapter 2 can be phrased with the notation we have just introduced.

For the minimum-cost routing problem we still assume that the costs are given per path. Since for each demand volume k we have P_k paths available, numbered from 1 to P_k, we will need cost values with double indices:

$$\phi_{kp}$$

indicates the cost per unit of data flow for the p-th path chosen for demand volume k ($1 \leq k \leq K, 1 \leq p \leq P_k$).

With these cost coefficients the overall minimum cost routing problem becomes:

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \sum_{k=1}^{K}\sum_{p=1}^{P_k} \phi_{kp} x_{kp} \\ \text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1,\ldots,K\} \\ & \sum_{k=1}^{K}\sum_{p=1}^{P_k} x_{kp}\delta_{kpl} \le c_l \qquad \text{for } l \in \{1,\ldots,L\} \\ & x_{kp} \ge 0 \qquad \text{for } k \in \{1,\ldots,K\}, p \in \{1,\ldots,P_k\} \end{aligned} \tag{5.2}$$

This problem has K demand constraints and L capacity constraints, and hence a total number of $K + L$ constraints. Since the simplex algorithm only considers basic feasible solutions, any solution will use at most $K + L$ different paths, which can be substantially less than the total number of paths considered, $\sum_{k=1}^{K} P_k$. There are $\sum_{k=1}^{K} P_k$ decision variables x_{kp}, to which a further L decision variables s_l ($1 \le l \le L$) have to be added as slack variables for converting the problem into standard form (see Section 4.2).

We will later on consider a variation of this problem where costs are assigned to individual links and the cost of a path is assumed to be the sum of the costs of the involved links – this is consistent with the standard assumption in shortest-path routing. If we denote by $\widehat{\phi}_l$ the cost of link l (for $1 \le l \le L$), then the cost of the p-th path of demand volume k becomes

$$\phi_{kp} = \sum_{l=1}^{L} \delta_{kpl}\widehat{\phi}_l \tag{5.3}$$

and the overall minimum-cost problem formulation becomes

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \sum_{k=1}^{K}\sum_{p=1}^{P_k} \left(\sum_{l=1}^{L} \delta_{kpl}\widehat{\phi}_l\right) x_{kp} \\ \text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1,\ldots,K\} \\ & \sum_{k=1}^{K}\sum_{p=1}^{P_k} x_{kp}\delta_{kpl} \le c_l \qquad \text{for } l \in \{1,\ldots,L\} \\ & x_{kp} \ge 0 \qquad \text{for } k \in \{1,\ldots,K\}, p \in \{1,\ldots,P_k\} \end{aligned} \tag{5.4}$$

For the load-balancing problem it is straightforward to carry over the formulation from Section 2.2.2 to get the following formulation:

$$\begin{aligned} \text{minimize}_{[\mathbf{x},\mathbf{y},r]} \quad & r \\ \text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1,\ldots,K\} \\ & \sum_{k=1}^{K}\sum_{p=1}^{P_k} x_{kp}\delta_{kpl} = y_l \qquad \text{for } l \in \{1,\ldots,L\} \\ & y_l \le c_l r \qquad \text{for } l \in \{1,\ldots,L\} \\ & x_{kp} \ge 0 \qquad \text{for } k \in \{1,\ldots,K\}, p \in \{1,\ldots,P_k\} \end{aligned} \tag{5.5}$$

$$y_l \geq 0 \qquad \text{for } l \in \{1, \ldots, L\}$$
$$r \geq 0$$

The auxiliary variables y_l are not strictly necessary, they can be removed.

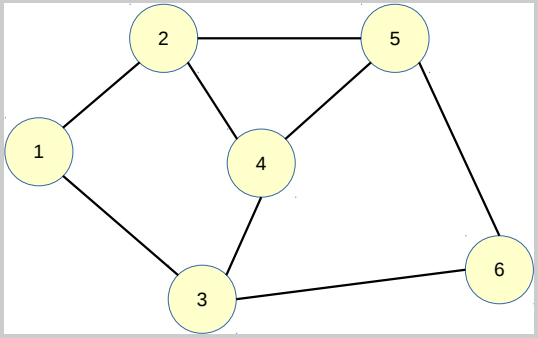
To carry over the problem for average delay, one needs to decide on how many points to use for the piecewise linear approximation of the function

$$f(y) = \frac{y}{c - y},$$

as we have discussed in Section 2.2.3. Re-using the approximation developed in that section would lead us to the formulation

$$\begin{aligned}
\text{minimize}_{[\mathbf{x},\mathbf{y},\mathbf{r}]} \quad & \sum_{l=1}^{L} \frac{r_l}{c_l} && (5.6)\\
\text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1, \ldots, K\} \\
& \sum_{k=1}^{K} \sum_{p=1}^{P_k} x_{kp} \delta_{kpl} = y_l \qquad \text{for } l \in \{1, \ldots, L\} \\
& - - - - - - \\
& r_l \geq \frac{3}{2} y_l \qquad \text{for } l \in \{1, \ldots, L\} \\
& r_l \geq \frac{9}{2} y_l - c_l \qquad \text{for } l \in \{1, \ldots, L\} \\
& \ldots \\
& r_l \geq 4000 y_l - 3781 c_l \qquad \text{for } l \in \{1, \ldots, L\} \\
& - - - - - - \\
& x_{kp} \geq 0 \qquad \text{for } k \in \{1, \ldots, K\}, p \in \{1, \ldots, P_k\} \\
& y_l \geq 0 \qquad \text{for } l \in \{1, \ldots, L\} \\
& r_l \geq 0 \qquad \text{for } l \in \{1, \ldots, L\}
\end{aligned}$$

Problem 5.1.1 (Setting up a Problem)**.**



We consider the network shown above. There are three demand volumes:

- 20 units between nodes 1 and 6, available paths are $1-2-5-6$ and $1-3-6$
- 37 units between nodes 2 and 3, available paths are $2-1-3$ and $2-4-3$.
- 16 units between nodes 3 and 5, available paths are $3-4-5$ and $3-6-5$.

The links are labeled as follows:

Link Index	Involved Nodes	Capacity
1	$1:2$	60
2	$1:3$	70
3	$2:5$	80
4	$2:4$	60
5	$3:4$	70
6	$3:6$	80
7	$4:5$	90
8	$5:6$	100

Furthermore, we denote the cost of routing one unit of flow on link l (with $1 \leq l \leq 8$) by $\widehat{\phi}_l$.
For the given network solve the following problems:

1. Give a table with all the path-link constants δ_{kpl} (where k represents the k-th demand volume and p the p-th path for this demand volume).
2. In the following you will be asked write down problem specifications. When doing so, give all equations and all involved terms explicitly, avoid summation signs ($\sum$).
   (a) Write down the minimum-cost routing problem and explain its components.
   (b) Write down the load-balancing problem.

5.2 Further Network Flow Problems

In this section we will discuss a range of further network flow problems.

5.2.1 Capacity Design

Broadly, in a capacity design problem we are given a number of nodes (e.g. routers) and demand volumes $h_1, h_2, \ldots, h_K$ between the nodes (as well as the set of available paths for each demand volume, see Section 5.1), but we actually have to lease the links between the nodes (for example from some ISP) and decide on their capacity c_l ($1 \leq l \leq L$). We assume that the ISP can offer links of arbitrary capacity c_l. Links between different nodes can have different prices ξ_l ($1 \leq l \leq L$) per unit capacity, e.g. reflecting the fact that an operator running a sea cable charges higher rates than another operator using land-based microwave links. The path-link indicators δ_{kpl} are given.

Problem 5.2.1 (Formulate the capacity design problem)**.**
Formulate an optimization problem in which you have to pay the minimum total link cost while satisfying your demand volumes.

A variation of this problem is the following one: suppose we are already given a capacitated network (i.e. the links have already capacities c_l assigned to them), but the network has been designed some time ago and the demand volumes have changed over time to become $h_1, \ldots, h_k$. Unfortunately it has turned out that the network cannot cope with the traffic anymore and needs an upgrade. You have the option to simultaneously increase the capacity c_l of all links in your network by the same amount z, and the cost of this upgrade are linearly proportional to z.

Problem 5.2.2 (Formulate the capacity increase problem)**.**
Formulate an optimization problem in which you minimize your upgrade cost while satisfying the new demand volumes.

Another variation of Problem 5.2.1 is a version where you want to design link capacities, but there is an upper bound C_l on the capacity of each link l, e.g. due to technological limits.

Problem 5.2.3 (Formulate the capacity design problem)**.**
Formulate a variation of Problem 5.2.1 in which there are maximum capacities C_l on each link.

5.2.2 Unsplittable Flows

In the unconstrained network flow problems discussed in Section 5.1.3 there has been no restriction on the number of paths that a demand volume can use in parallel with flows of positive rate. Sometimes it is desirable to restrict a demand volume to a single path, for example when the demand volume consists of a number of TCP connections using older TCP versions based on the Goback-N protocol. These TCP versions have problems when the network introduces packet reorderings (which is almost guaranteed to happen if multiple paths are used in parallel in a round-robin fashion), since the receiver will throw away out-of-order packets (which otherwise can be perfectly fine), triggering a lot of retransmissions.

To express restrictions like single-path restrictions, we replace the decision variables x_{kp} (which can take on any non-negative real value) by integer-valued decision variables u_{kp}. In this particular case, we use u_{kp} as so-called **indicator variables** or **binary variables**, which can only assume the values 0 or 1. When the p-th

path of demand volume k is used to carry the data, then $u_{kp} = 1$, otherwise we have $u_{kp} = 0$. The constraint that only one path is used can then for a given demand volume k be expressed as

$$\sum_{p=1}^{P_k} u_{kp} = 1$$

and with that the amount of flow on the p-th path is simply given by

$$u_{kp} h_k$$

The link capacity constraints become (by summing the amount of flow that demand volumes put on link l):

$$\sum_{k=1}^{K} \sum_{p=1}^{P_k} \delta_{kpl} u_{kp} h_k \leq c_l$$

for $1 \leq l \leq L$, and the overall minimum cost routing problem can then be formulated as:

$$\begin{aligned}
\text{minimize}_{[\mathbf{u}]} \quad & \sum_{k=1}^{K} \sum_{p=1}^{P_k} \phi_{kp} h_k u_{kp} && (5.7)\\
\text{subject to} \quad & \sum_{p=1}^{P_k} u_{kp} = 1 \qquad \text{for } k \in \{1, \ldots, K\} \\
& \sum_{k=1}^{K} \sum_{p=1}^{P_k} u_{kp} h_k \delta_{kpl} \leq c_l \qquad \text{for } l \in \{1, \ldots, L\} \\
& u_{kp} \in \{0, 1\} \qquad \text{for } k \in \{1, \ldots, K\}, p \in \{1, \ldots, P_k\}
\end{aligned}$$

This is an integer linear program, and as already explained in Section 3.6 this is generally much harder to solve than "normal" linear programs. In general, integer linear programs can be NP-hard problems with no substantially better solution strategy available than to iterate over all possible allocations of integers to decision variables (which in problem 5.7 would amount to 2^M candidate allocations with $M = \sum_{k=1}^{K} P_k$), checking for each allocation whether it satisfies the constraints and, if so, whether it improves on the smallest value found so far.[2]

In CPLEX integer constraints are expressed in an `Integer` section, see Section 7.1.4.

5.2.3 Influencing Routing

It is sometimes desirable to be somewhat robust against failures of links or routers, and to split up a demand volume delibately over a certain minimum number of paths. More precisely, we assume as usual that for demand volume k ($1 \leq k \leq K$) we have identified P_k different paths, out of which at least $1 \leq n_k \leq P_k$ paths have to be used with a positive amount of flow assigned to them. The links and their capacities c_l, as well as the path-link indicators δ_{kpl} are given.

Problem 5.2.4 (Formulate the flow diversity problem)**.**
Formulate this problem. How can you express the constraint that "at least n_k paths have to be used for a demand volume"? Do you need an objective function?

[2]Not every linear program with integer constraints is automatically NP-complete though.

With the right way of expressing the "use-at-least-n_k-paths" constraint the previous problem 5.2.4 could be formulated as a standard LP problem. Next we consider a variation of this problem, in which we ask that any flow with non-zero rate actually has a rate at least being a minimum pre-defined rate. It is important to understand that we are not asking to ensure that **all** flows of a demand volume have a given minimum rate, but only the non-zero flows – in other words, a path is either not used at all (with a zero flow) or it is used with a minimum rate. This actually turns the problem into a mixed-integer-programming problem (i.e. we have both real-valued and integer-valued decision variables) and can be seen as an extenstion to the problem of unsplittable flows in Section 5.2.2.

Problem 5.2.5 (Formulate the non-zero flow problem)**.**
Formulate this problem with the aim of finding feasible flows x_{kp}. Assume that each flow of demand volume k has to have a minimum rate of b_k, but can have a larger rate. Use binary indicator variables $u_{kp} \in \{0, 1\}$ telling whether path p for demand volume k is used ($u_{kp} = 1$) or not ($u_{kp} = 0$).

Another variation of Problem 5.2.4 is the requirement to split demand volume k into **exactly** n_k different flows instead of just asking to split it **at least** into n_k flows. This will lead us to another variation of the unsplittable-flows problem (Section 5.2.2).

Problem 5.2.6 (Formulate the exactly-n_k-flows problem)**.**
Formulate a problem with the aim of finding feasible flows x_{kp}. Assume that demand volume k needs to be split over exactly $n_k \leq P_k$ flows. Come up with a formulation where each of the n_k flows has an arbitrary amount of flow, and another formulation where all flows have an equal amount of flow.

Solution 5.2.1.

We will again use binary indicator variables $u_{kp} \in \{0, 1\}$. Again, we are only looking for a feasible allocation and do not need an objective function. For arbitrary amounts of flow the problem becomes:

$$
\begin{aligned}
&\textit{minimize}_{[\mathbf{x},\mathbf{u}]} \\
&\textit{subject to} \quad \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \textit{for } k \in \{1, \ldots, K\} \\
&\qquad\qquad \sum_{p=1}^{P_k} u_{kp} = n_k \qquad \textit{for } k \in \{1, \ldots, K\} \\
&\qquad\qquad x_{kp} \leq h_k u_{kp} \qquad \textit{for } k \in \{1, \ldots, K\},\ p \in \{1, \ldots, P_k\} \\
&\qquad\qquad \sum_{k=1}^{K} \sum_{p=1}^{P_k} x_{kp} \delta_{kpl} \leq c_l \qquad \textit{for } l \in \{1, \ldots, L\} \\
&\qquad\qquad x_{kp} \geq 0 \qquad \textit{for } k \in \{1, \ldots, K\},\ p \in \{1, \ldots, P_k\} \\
&\qquad\qquad u_{kp} \in \{0, 1\} \qquad \textit{for } k \in \{1, \ldots, K\},\ p \in \{1, \ldots, P_k\}
\end{aligned}
$$

and for equal split we get:

$$
\begin{aligned}
&\textit{minimize}_{[\mathbf{x},\mathbf{u}]} \\
&\textit{subject to} \quad \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \textit{for } k \in \{1, \ldots, K\}
\end{aligned}
$$

$$\sum_{p=1}^{P_k} u_{kp} = n_k \qquad \textit{for } k \in \{1, \ldots, K\}$$

$$x_{kp} = \frac{u_{kp} h_k}{n_k} \qquad \textit{for } k \in \{1, \ldots, K\},\ p \in \{1, \ldots, P_k\}$$

$$\sum_{k=1}^{K} \sum_{p=1}^{P_k} x_{kp} \delta_{kpl} \leq c_l \qquad \textit{for } l \in \{1, \ldots, L\}$$

$$x_{kp} \geq 0 \qquad \textit{for } k \in \{1, \ldots, K\},\ p \in \{1, \ldots, P_k\}$$

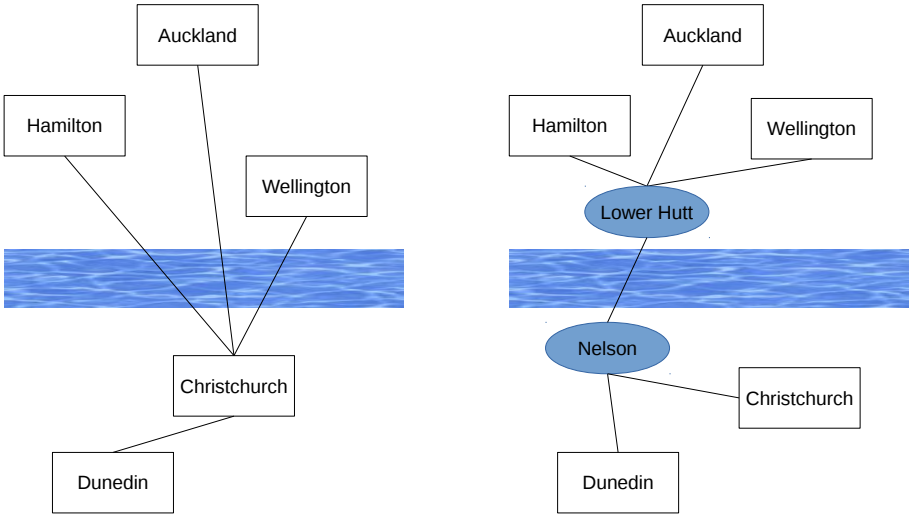
$$u_{kp} \in \{0, 1\} \qquad \textit{for } k \in \{1, \ldots, K\},\ p \in \{1, \ldots, P_k\}$$

5.3 Topology Design and Related Problems

In all the problems considered so far the network topology, i.e. the set of nodes and the links between them, have been given in advance, although in the capacity design (or dimensioning) problems of Section 5.2.1 we had to choose the transmission capacity of the given links.

In this section we consider some topology design problems. In the very early stages of planning a network, an operator has to make a number of decisions:

- Among the first decisions for an operator is in which locations it actually wants to establish a presence – for example, in which NZ cities should a new ISP install routers and access capacity? This sort of decisions is influenced by a number of factors, including the ISPs estimations of the size of the customer population and the actual capacity-independent cost of establishing a presence (for example for real-estate, installation of suitable water and electricity supply, etc).
- Once the locations have been chosen and initial estimates of the demand volumes are available, the transport network between the locations needs to be designed. Suppose that the ISP has decided to open offices in Auckland, Hamilton, Christchurch, Wellington and Dunedin, then the main routers in these offices need to be connected by a transport network. For reasons of reliability or cost it may be beneficial to install further routers which are not a source or destination of any user traffic. An example is shown in the following figure:



In the first example only the "access routers" in the five mentioned cities are connected to each other, whereas in the second example two additional "transit" routers in Lower Hutt (on the North Island) and Nelson (on the South Island) are introduced, so that only one under-sea cable in the Cook strait has to be installed. With reference to this example, in the following we call nodes which originate traffic or which are the destination of traffic **access nodes** (the five cities Auckland, Hamilton, Wellington, Christchurch and Dunedin in the above figure), and routers that are used for transit without being involved in any demand flow are called **transit nodes** (the two routers in Lower Hutt and Nelson).

5.3.1 Placement of Transit Routers

The first problem we describe in some more detail is the placement of transit routers. Suppose we are given N different access nodes $1, \ldots, N$ and we want to connect these access nodes exclusively through transit routers (i.e. no direct connection between access nodes is considered). We have M different candidate locations for transit routers (called transit locations) and we can place either zero or one transit routers into any of these transit locations, but no more. Furthermore, we assume that in each transit location j we can handle only up to K_j different access nodes (e.g. due to limitations on the number of ports we can afford in the transit routers). The cost of connecting an access router in location i to a transit router in location j is given by ξ_{ij}, and the cost for installing a transit router in transit location j is η_j. Each access router shall be connected to exactly one transit router. We use the following decision variables:

- $x_j \in \{0, 1\}$ (for $1 \leq j \leq M$) is a binary decision variable which tells whether there will be a transit router installed in transit location j ($x_j = 1$) or not ($x_j = 0$).
- $u_{ij} \in \{0, 1\}$ (for $1 \leq i \leq N$ and $1 \leq j \leq M$) is a binary decision variable which tells whether access router i will be connected to transit router j ($u_{ij} = 1$) or not ($u_{ij} = 0$).

Note that in this problem we only consider connecting the access routers to the transit routers and ignore the problem of interconnecting the transit routers.

Problem 5.3.1 (Minimum cost problem)**.**
Formulate the problem for minimizing the installation cost (or capital cost).

5.3.2 Placement and Interconnection of Transit Routers

The next problem is an extension of the previous problem, in which we take into account the interconnection of the transit routers as well. This interconnection also has installation costs, and the objective will become to minimize the total cost for:

- placing transit routers into transit locations,
- connecting access routers to transit routers, and
- interconnecting transit routers.

In addition to the cost values ξ_{ij} and η_j introduced above, we also introduce cost values ζ_{mn} for interconnecting a transit router in location m ($1 \leq m \leq M$) with a transit router in location n ($1 \leq n \leq M$). We assume that the interconnection cost are symmetric, i.e. $\zeta_{mn} = \zeta_{nm}$ and that the "self-connection" costs are $\zeta_{mm} = 0$ for $1 \leq m \leq M$. We furthermore assume that there are exactly $P \in \mathbb{N}$ transit routers to be placed on P transit locations.

Problem 5.3.2 (Minimum cost problem, fully connected transit routers)**.**
Formulate the problem for minimizing the installation cost (or capital cost), assuming that the P transit routers are fully interconnected, i.e. each transit router is connected to all $P - 1$ other transit vectors. More precisely:

- Do we need new decision variables beyond the ones already used in Problem 5.3.1 to cover the interconnection of transit routers?
- What does the objective function look like?
- Establish all required constraints.

The requirement posed in the previous problem that all transit routers have to be fully interconnected is reasonable for small numbers P of transit routers to consider, but for larger numbers P the number of interconnection links can become prohibitive. Suppose instead that we want to use just $L \leq P(P-1)/2$ links. Then we can use the following heuristic procedure:

- Solve the full problem 5.3.2 with the full set of interconnections.
- List all the interconnection links used in the solution of 5.3.2 in decreasing order of link costs.
- Traverse this list from the highest- to the lowest-cost link, for each link do the following:
  - Remove the link from the interconnection network and check whether the remaining network is still (strongly) connected.
  - If so, remove the link and proceed with the next link.
  - Otherwise, keep the link and proceed with the next link.

5.3.3 Adding Traffic Demands

In this section we add traffic demands to the picture. More precisely, we assume that the access nodes are already in place and that we have a first idea of the demand volumes to be transported between these access

nodes. The access nodes are connected through a network of transit nodes and this transit network needs to be designed. We will consider a few relevant design problems.

In the first design problem the number and locations of access nodes have already been decided, and the access routers have already been connected to the transit routers. The main design objective concerns the interconnection links between the transit routers. The candidate links are indexed from 1 to L. We assume that there are two different types of costs associated to a link: the **capital cost** κ_l of link l refers to the one-time cost for the installation of the link, whereas the **operational cost** of link l is calculated per unit flow for this link and adds up to $c_l\omega_l$, where ω_l is the unit cost per unit of flow and c_l is the capacity of this link. We want to design the interconnection network (including the link capacities) such that the operational cost are minimized, subject to the constraint that the total installation cost does not exceed a given budget B.

We proceed to formulate this problem. As usual, the demand volumes are labeled from 1 to K, and for each demand volume k we have P_k different candidate paths, which are realized through the links $1, \ldots, L$, where the constants δ_{kpl} indicate whether link l is used on the p-th path for demand volume k ($\delta_{kpl} = 1$) or not. Clearly, we have to assume that the transit network is connected when sufficiently many of the given links are activated. A link l can be used or not. To reflect this, we introduce a binary decision variable $u_l \in \{0, 1\}$ for each link, and when a link is used, then it incurs installation cost. When a link is used, we have to decide its capacity c_l, and furthermore we have to decide for each demand volume k the amount of flow x_{kp} to send over path p ($1 \leq p \leq P_k$). The objective function for this problem is given by:

$$\sum_{l=1}^{L} c_l \omega_l$$

as we wanted to minimize the operational cost. We have the following constraints:

- The usual demand constraints:

$$\sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1, \ldots, K\}$$

  expressing that each demand volume should indeed be satisfied.

- The capacity constraints:

$$\sum_{k=1}^{K} \sum_{p=1}^{P_k} \delta_{kpl} x_{kp} \leq c_l \qquad \text{for } l \in \{1, \ldots, L\}$$

  saying that the capacity of a link l should at least be as large as the aggregate rate of all flows using this link.

- The constraint on capital costs:

$$\sum_{l=1}^{L} \kappa_l u_l \leq B$$

  saying that the sum of all link-installation costs should not exceed the budget B.

- We finally need a constraint which links the binary decision variable u_l to the capacity c_l, in particular expressing that the capacity of an un-installed link ($u_l = 0$) has to be zero as well. To facilitate this, we introduce further parameters C_l, which express a maximum available capacity for a link l, and the constraint becomes:

$$c_l \leq C_l u_l$$

  This does what we want:

- When $u_l = 0$, this forces $c_l = 0$ as well.
- When $u_l = 1$ then c_l cannot exceed some upper bound.

Summarizing, we have the following problem:

$$
\begin{aligned}
\text{minimize}_{[\mathbf{u},\mathbf{x},\mathbf{c}]} \quad & \sum_{l=1}^{L} c_l \omega_l \\
\text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1,\dots,K\} \\
& \sum_{k=1}^{K}\sum_{p=1}^{P_k} \delta_{kpl} x_{kp} \le c_l \qquad \text{for } l \in \{1,\dots,L\} \\
& \sum_{l=1}^{L} \kappa_l u_l \le B \\
& c_l \le C_l u_l \qquad \text{for } l \in \{1,\dots,L\} \\
& u_l \in \{0,1\} \qquad \text{for } l \in \{1,\dots,L\} \\
& c_l \ge 0 \qquad \text{for } l \in \{1,\dots,L\} \\
& x_{kp} \ge 0 \qquad \text{for } k \in \{1,\dots,K\}, p \in \{1,\dots,P_k\}
\end{aligned}
$$

This is a mixed-integer problem, as it contains both (binary) integer variables and (non-negative) real variables.

Problem 5.3.3 (Connectivity?)**.**
In putting together the previous optimization problem we seemingly have not considered the issue of connectivity in the transit network, i.e. are we guaranteed that the transit network is indeed suitably connected? Argue whether or not in the previous problem formulation there is an issue with connectivity in the transit network.

Problem 5.3.4 (A variation of this problem)**.**
Give a variation of the previous problem in which the capital cost are not constrained anymore by a budget, but you still want to minimize them. How can you introduce relative weights for the operational and the capital costs?

We will finally consider a problem in which the number and positions of the transit nodes, the links between them and their capacities have to be decided jointly. More precisely, the following is given:

- The access nodes are already in place.
- There are M potential positions for transit nodes and the capital cost of installing a transit router in position $m \in \{1,\dots,M\}$ is ϕ_m. Furthermore, a transit node in location m can have at most G_m interconnections to other transit nodes.
- There are L potential interconnection links between transit routers that can be installed. The installation / capital cost for link l is κ_l, the operational cost per unit flow for link l is ω_l. Each potential link interconnects two transit locations that have been decided in advance. The information of whether a

link l is available at transit location m is captured in constants β_{ml} ($1 \leq m \leq M$, $1 \leq l \leq L$), where $\beta_{ml} = 1$ indicates whether link l is available in location m ($\beta_{ml} = 1$) or not ($\beta_{ml} = 0$). Furthermore, there is an upper bound C_l for the capacity of link l. The capacity c_l of a link needs to be decided, and the operational costs of a link are given by $c_l\omega_l$.

- There are K traffic demands, with h_k being the demand volume for demand k ($1 \leq k \leq K$). For each traffic demand k a number P_k different paths have been identified beforehand. The relationship between paths for flows and involved links is again given by the constants δ_{kpl}.

Problem 5.3.5 (Formulating the problem)**.**
Give a formulation of this problem. In particular:

- Minimize both operational and capital cost.
- Which decision variables will you need?

5.4 Comments

In this chapter we have seen a variety of flow and network planning problems. Some of these problems are genuine linear programs and can hence be efficiently solved using the simplex or other algorithms for solving linear programs. Other problems, however, are of either the integer- or the mixed-integer type, and some of these problems (like the unsplittable-flows problem discussed in Section 5.2.2) can in fact be shown to be NP-hard. For other (mixed-)integer problems efficient algorithms generating either approximate or precise solutions exist and can be found in the literature.

We conclude this chapter with another observation. For flow planning problems we have always tacitly assumed that we actually can achieve any given split $x_{k1}, x_{k2}, \ldots, x_{kP_k}$ for a given flow k. For example, if we are given a demand volume of $h_k = 50$ units and our network flow problem has determined that $x_{k1} = 20$ units go over the first path and $10, 5, 5, 3, 3, 4$ units go over the next six paths, respectively, then we actually need to influence our underlying routing protocol to make this split work as intended. We will see in the next (optional) Chapter 6 that things are not so easy with OSPF, as OSPF has a tendency to only use the single best path between a source and a destination, and the paths chosen by OSPF depend entirely on the weights or cost values we assign to links. Even with the equal-cost-multipath (ECMP) facility not all intended flow allocations can be achieved. So, the best we can hope for is to find systems of link metrics which approximate a given flow allocation as closely as possible. However, there is another technology that is used in the Internet (in particular in core areas) and which can split up demand volumes in a very fine-grained manner. This is the Multi-Protocol Label-Switching (MPLS) protocol [7].

Chapter 6

Excursion: Link-Weight Determination for Shortest-Path Routing

This entire chapter is an excursion and can be skipped, in particular the proofs. We will have a first look at the problem of how to achieve a given network flow allocation (see Chapter 5) on an OSPF network, which is based on shortest-path routing and where the only available control knob is the set of weights or cost values assigned to the available links. As a preparation, we need to introduce a concept related to linear programming, the concept of duality. The presentation in this chapter is sketchy and based on a number of sources, including [14], [16], [13], [6].

6.1 Duality

We now pick up a more theoretical topic in linear programming. Suppose we are given a linear program in the following form (which for now is **not** the standard form):

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{6.1}$$

To this LP we can associate a related optimization problem called the **dual problem** as:

$$\begin{aligned} \text{maximize}_{[\mathbf{z}]} \quad & \mathbf{b}^T\mathbf{z} \\ \text{subject to} \quad & \mathbf{A}^T\mathbf{z} \leq \mathbf{c} \\ & \mathbf{z} \geq \mathbf{0} \end{aligned} \tag{6.2}$$

where, as compared to the original problem 6.1 we have exchanged the cost vector $\mathbf{c}$ and the right-hand side $\mathbf{b}$, have transposed the coefficient matrix $\mathbf{A}$, have exchanged the "greater-than-or-equal" constraint to a "smaller-than-or-equal" constraint, and now want to maximize instead of minimize. In this context, the original problem 6.1 is also called the **primal problem**. It is not hard to see that for this particular problem formulation the dual problem of the dual problem gives again the primal problem (after changing the direction of the inequality constraints suitably).

Before developing some of the relationships between the primal and the dual problems, we first describe how

to obtain the dual for other types of problem formulations and then use an example to develop some intuition on how dual problems might arise.

6.1.1 Finding the Dual Problem

We have chosen to start this section with a primal problem formulation of the form 6.1 because the associated dual problem 6.2 shows a particularly pleasing symmetry with respect to the primal problem. For a primal problem with mixed constraints (equality and inequality constraints, non-zero or free variables) the following rules apply when finding the dual:

- When the primal has an $\leq$ constraint inequality, multiply this (and only this!) inequality by -1 on both sides to turn it into an $\geq$ constraint inequality. This gives a new version of the primal.
- When the primal has a constraint equality $\mathbf{a}^i\mathbf{x} = b_i$ (where $\mathbf{a}^i$ is a row vector containing the i-th row of coefficient matrix $\mathbf{A}$), then replace this (still in the primal) by two constraint inequalities:

$$\begin{aligned} \mathbf{a}^i\mathbf{x} &\geq b_i \\ \mathbf{a}^i\mathbf{x} &\leq b_i \end{aligned}$$

which by the first rule gets translated into

$$\begin{aligned} \mathbf{a}^i\mathbf{x} &\geq b_i \\ -\mathbf{a}^i\mathbf{x} &\geq -b_i \end{aligned}$$

In particular, for a primal problem given in standard form

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

(with $\mathbf{A}$ being an $m \times n$ matrix as usual) we get the revised primal

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \begin{bmatrix} \mathbf{A} \\ -\mathbf{A} \end{bmatrix} \mathbf{x} \geq \begin{bmatrix} \mathbf{b} \\ -\mathbf{b} \end{bmatrix} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

which is in the right "symmetric" form but has $2m$ constraints instead of m constraints. We write the dual decision vector $\mathbf{z}$ as

$$\mathbf{z} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}$$

where $\mathbf{u}$ and $\mathbf{v}$ are m-dimensional vectors. With that the dual problem becomes

$$\begin{aligned} \text{maximize}_{[\mathbf{u},\mathbf{v}]} \quad & [\mathbf{b}^T, -\mathbf{b}^T] \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \\ \text{subject to} \quad & [\mathbf{A}^T, -\mathbf{A}^T] \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \leq \mathbf{c} \\ & \mathbf{u} \geq \mathbf{0}, \mathbf{v} \geq \mathbf{0} \end{aligned}$$

or after re-factoring

$$\begin{aligned} \text{maximize}_{[\mathbf{u},\mathbf{v}]} \quad & \mathbf{b}^T(\mathbf{u}-\mathbf{v}) \\ \text{subject to} \quad & \mathbf{A}^T(\mathbf{u}-\mathbf{v}) \leq \mathbf{c} \\ & \mathbf{u} \geq \mathbf{0}, \mathbf{v} \geq \mathbf{0} \end{aligned}$$

We realize that $\mathbf{u}$ and $\mathbf{v}$ are both m-dimensional vectors and the dual problem depends only on their difference $\boldsymbol{\lambda} = \mathbf{u} - \mathbf{v}$, and recalling from Section 4.2.3 that free variables can be represented as the difference of two non-negative variables, we arrive at the final form of the dual as

$$\begin{aligned} \text{maximize}_{[\boldsymbol{\lambda}]} \quad & \mathbf{b}^T\boldsymbol{\lambda} \\ \text{subject to} \quad & \mathbf{A}^T\boldsymbol{\lambda} \leq \mathbf{c} \end{aligned} \tag{6.3}$$

where the variable vector $\boldsymbol{\lambda}$ is not constrained to be non-negative.

The following table contains a summary of the translation rules:

Comment	Primal	Dual
Primal in $\geq$ form	$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$	$\begin{aligned} \text{maximize}_{[\mathbf{z}]} \quad & \mathbf{b}^T\mathbf{z} \\ \text{subject to} \quad & \mathbf{A}^T\mathbf{z} \leq \mathbf{c} \\ & \mathbf{z} \geq \mathbf{0} \end{aligned}$
Primal in standard ("=") form	$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$	$\begin{aligned} \text{maximize}_{[\mathbf{z}]} \quad & \mathbf{b}^T\mathbf{z} \\ \text{subject to} \quad & \mathbf{A}^T\mathbf{z} \leq \mathbf{c} \end{aligned}$
Objective	$\min \mathbf{c}^T\mathbf{x}$	$\max \mathbf{b}^T\mathbf{z}$
Constraint equation	$\mathbf{a}^i\mathbf{x} = b_i$	Define an unrestricted dual variable z_i
Constraint inequality	$\mathbf{a}^i\mathbf{x} \geq b_i$	Define a non-negative dual variable z_i
Constrained variable	$x_j \geq 0$	$\mathbf{a}^j\mathbf{z} \leq c_j$
Unconstrained variable	x_j	$\mathbf{a}^j\mathbf{z} = c_j$

6.1.2 Interpretation as Prices

Consider again the diet problem discussed in Section 4.1.1, which led to a problem of the type

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where c_j is the unit cost of food j ($1 \leq j \leq n$), b_i is the minimum requirement for nutrient i ($1 \leq i \leq m$) and $a_{i,j}$ indicates how many units of nutrient i are contained in a unit of food j.

Now suppose you run a pharmaceutical company and have a new business idea. Instead of forcing customers to buy various sorts of food to collectively cover nutrition requirements, you produce m different pills, such that the i-th type of pill contains exactly b_i units of nutrient i (for $1 \leq i \leq m$). Your main goal now is to determine at which price z_i you want to sell one unit of pills for nutrient i. More precisely, you want to set the prices such that:

- Your revenue is maximized.
- For customers it is cheaper to buy your pills than buying the food (for otherwise most customers would probably still prefer the food).

So, the company wants to maximize

$$\mathbf{b}^T\mathbf{z}$$

which gives the total revenue when setting the unit pill prices to $z_j \geq 0$. With respect to the pricing, we would like to preserve the property that for any particular food j it should be cheaper to obtain the nutrient mixture in j through buing the pills rather than buying the food, i.e. we should have:

$$a_{1,j}z_1 + a_{2,j}z_2 + \ldots + a_{m,j}z_m \leq c_j$$

for $1 \leq j \leq n$, which in matrix/vector notation just leads to the constraints:

$$\mathbf{A}^T\mathbf{z} \leq \mathbf{c}$$

Furthermore, the prices should clearly be non-negative, i.e. $\mathbf{z} \geq \mathbf{0}$. Hence, we just arrive at the dual problem for the diet problem, and the quantities z_j can be interpreted as optimal prices or optimal cost.

6.1.3 Duality Theorems

For the following theorems we assume that the primal problem is given in standard form:

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \tag{6.4}$$

and the dual then is:

$$\begin{array}{ll} \text{maximize} & \mathbf{b}^T\mathbf{z} \\ \text{subject to} & \mathbf{A}^T\mathbf{z} \leq \mathbf{c} \end{array} \tag{6.5}$$

Our main goal in this section is to prove the **duality theorem**, which roughly states that if either of the problems 6.4 and 6.5 has a solution, then so has the other and the optimal values of their objective functions are the same!

We start with a simple observation.

Theorem 6.1 (Weak duality theorem)**.** *When $\mathbf{x}$ is a feasible solution of the primal 6.4 and $\mathbf{z}$ is a feasible solution of the dual 6.5 then we have*

$$\mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{z}$$

As a corollary, when

$$\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{z}$$

holds, then $\mathbf{x}$ and $\mathbf{z}$ must already be the optimal solutions of problems 6.4 and 6.5, respectively.

Proof. We have:

$$\mathbf{b}^T\mathbf{z} = \mathbf{x}^T\mathbf{A}^T\mathbf{z} \leq \mathbf{x}^T\mathbf{c} = \mathbf{c}^T\mathbf{x}$$

where the next-to-last equation is true because $\mathbf{x}$ is a non-negative vector. The corollary is immediate. □

In other words, any feasible solution of the dual has an objective value that is smaller-than or equal to the objective value of any feasible solution of the primal. So, as a minimum, by solving the dual we get lower bounds on the minimum achievable value of the primal.

Excursion (The separating hyperplane theorem)

Before we state and prove the duality theorem we will first develop an auxiliary result about convex sets (remember that the set of feasible points of a linear program is always a convex set).

Theorem 6.2 (Separating hyperplane theorem)**.** *Be $C \subset \mathbb{R}^n$ a non-empty closed convex set and $\mathbf{y} \in \mathbb{R}^n$ a point with positive distance to C, i.e. we have*

$$\delta = \inf\left\{\|\mathbf{x} - \mathbf{y}\|_2 : \mathbf{x} \in C\right\} > 0$$

Then there exists a vector $\mathbf{a} \in \mathbb{R}^n$ such that

$$\mathbf{a}^T\mathbf{y} < \inf\left\{\mathbf{a}^T\mathbf{x} : \mathbf{x} \in C\right\} \tag{6.6}$$

A subset $C \subset \mathbb{R}^n$ is called **closed**, when it contains all its limit points, i.e. when $(\mathbf{x}_n)_{n\in\mathbb{N}} \subset C$ is a sequence of points from C which converges to some $\mathbf{x} \in \mathbb{R}^n$, then it follows that $\mathbf{x} \in C$.

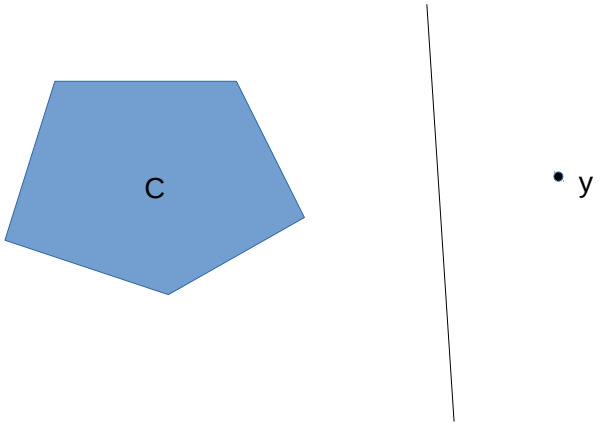
To understand the separating hyperplane theorem, recall from Section 4.3 that in the two-dimensional case a set of the form

$$\left\{\mathbf{x} \in \mathbb{R}^2 : \mathbf{a}^T\mathbf{x} = c\right\}$$

for given $\mathbf{a}$ describes a line in $\mathbb{R}^2$. Geometrically, the line given by this set is perpendicular to the line from the origin to point $\mathbf{a}$. We can do a similar thing in higher dimensions, where for given $\mathbf{a} \in \mathbb{R}^n$ the set

$$\left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T\mathbf{x} = c\right\}$$

forms a so-called hyperplane, which is an $n-1$ dimensional set perpendicular to the vector $\mathbf{a}$. The separating hyperplane theorem now says: given a convex set C and an external point $\mathbf{y}$ we can find a so-called "separating hyperplane", i.e. a hyperplane which truly sits "between" the set and the point and which touches neither of them. A two-dimensional example is shown in the following figure:



The separating hyperplane is normally not uniquely determined.

Proof. The first important observation is that there actually exists a $\mathbf{x}_0 \in C$ which has a distance to $\mathbf{y}$ of exactly δ, i.e.

$$\|\mathbf{x}_0 - \mathbf{y}\|_2 = \delta$$

To see this, take the closed ball $\widehat{B}_{2\delta}(\mathbf{y}) := \left\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{y}\|_2 \leq 2\delta\right\}$ of radius 2δ around $\mathbf{y}$ and intersect it with C to get a closed and bounded set $\widehat{C}$. The function $\mathbf{x} \mapsto \|\mathbf{x} - \mathbf{y}\|_2$ from $\widehat{C}$ into the non-negative real numbers is continuous, and since a continuous function on a bounded and closed subset of $\mathbb{R}^n$ actually assumes a minimum and maximum value, the claim follows, i.e. there must be at least one $\mathbf{x}_0 \in \widehat{C}$ for which $\|\mathbf{x}_0 - \mathbf{y}\|_2 = \delta$ indeed holds.

In the remaining proof we set $\mathbf{a} = \mathbf{x}_0 - \mathbf{y}$ and show that this particular choice of $\mathbf{a}$ satisfies the condition 6.6. Be $\mathbf{x} \in C$. Since C is convex, the convex combination

$$\alpha\mathbf{x} + (1-\alpha)\mathbf{x}_0 = \mathbf{x}_0 + \alpha(\mathbf{x} - \mathbf{x}_0)$$

for some $0 \leq \alpha \leq 1$ is in C as well and since $\mathbf{x}_0$ has the smallest possible distance to C, we have that:

$$\|\mathbf{x}_0 + \alpha(\mathbf{x} - \mathbf{x}_0) - \mathbf{y}\|_2^2 \geq \|\mathbf{x}_0 - \mathbf{y}\|_2^2$$

Using that $\|\mathbf{u}\|_2 = \sqrt{(\mathbf{u}, \mathbf{u})}$ and the properties of the inner product (see Appendix, Section A.1) we can expand both sides, cancel and re-arrange to get:

$$2\alpha(\mathbf{x}_0 - \mathbf{y}, \mathbf{x} - \mathbf{x}_0) + \alpha^2 \|\mathbf{x} - \mathbf{x}_0\|_2^2 \geq 0$$

If we now let α go to 0 from above (i.e. $\alpha \geq 0$) then the second term will vanish much more quickly than the first one, which then implies that

$$(\mathbf{x}_0 - \mathbf{y}, \mathbf{x} - \mathbf{x}_0) \geq 0$$

holds. Expanding this further gives

$$\begin{aligned}(\mathbf{x}_0 - \mathbf{y}, \mathbf{x}) &\geq (\mathbf{x}_0 - \mathbf{y}, \mathbf{x}_0) \\ &= (\mathbf{x}_0 - \mathbf{y}, \mathbf{x}_0 + \mathbf{y} - \mathbf{y}) = (\mathbf{x}_0 - \mathbf{y}, \mathbf{y}) + (\mathbf{x}_0 - \mathbf{y}, \mathbf{x}_0 - \mathbf{y}) = (\mathbf{x}_0 - \mathbf{y}, \mathbf{y}) + \delta^2\end{aligned}$$

or, recalling that we have set $\mathbf{a} = \mathbf{x}_0 - \mathbf{y}$, this is the same as

$$(\mathbf{a}, \mathbf{x}) \geq (\mathbf{a}, \mathbf{y}) + \delta^2$$

Since $\mathbf{x} \in C$ was arbitrary, we can conclude that

$$\inf_{\mathbf{x} \in C} (\mathbf{a}, \mathbf{x}) > (\mathbf{a}, \mathbf{y})$$

which is what we set out to prove.

□

Theorem 6.3 (Duality theorem)**.** *The following holds:*

1. *If either of problems 6.4 and 6.5 has a finite optimal solution, then the other one does as well, and the corresponding optimal objective values are equal.*
2. *If either problem has an unbounded objective function, then the other problem has no feasible solution.*

This theorem is one of the main reasons why dual problems are important: you can learn something about one problem by looking at a related problem (the dual), which might be simpler. Furthermore, there are variants of the simplex algorithm which make good use of dual problems.

Excursion (Proof of duality theorem)

Proof. We only prove the first statement of the theorem. We assume that the primal problem has a finite optimal solution $\mathbf{x}_0$ with objective value z_0, and we will show that the dual has a solution with the same objective value. By exchanging the roles of the primal

and the dual (recall: the dual of the dual is the primal, after converting some signs) this also implies the reverse direction in which we would conclude that the primal has a solution with the same objective value when an optimal solution of the dual is given.

We introduce the auxiliary set

$$C = \left\{(r, \mathbf{w}) : r = tz_0 - \mathbf{c}^T\mathbf{x}, \mathbf{w} = t\mathbf{b} - \mathbf{A}\mathbf{x}, \mathbf{x} \geq \mathbf{0}, t \geq 0\right\}$$

and collect a few observations about this set:

- The set C is convex. To check this, assume that

$$\begin{aligned}(r_1, \mathbf{w}_1) &= (t_1 z_0 - \mathbf{c}^T\mathbf{x}_1, t_1\mathbf{b} - \mathbf{A}\mathbf{x}_1)\\ (r_2, \mathbf{w}_2) &= (t_2 z_0 - \mathbf{c}^T\mathbf{x}_2, t_2\mathbf{b} - \mathbf{A}\mathbf{x}_2)\end{aligned}$$

  are vectors from C and we have picked some scalar $0 \leq \lambda \leq 1$ to form their convex combination

$$(r, \mathbf{w}) = \lambda(r_1, \mathbf{w}_1) + (1-\lambda)(r_2.\mathbf{w}_2) = (\lambda r_1 + (1-\lambda)r_2, \lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)$$

  For the first component of this vector, $r = \lambda r_1 + (1-\lambda)r_2$, we have to show that we can write it in the form

$$r = tz_0 - \mathbf{c}^T\mathbf{x}$$

  for some $t \geq 0$ and $\mathbf{x} \geq \mathbf{0}$. Plugging the definitions of r_1 and r_2 gives:

$$\begin{aligned}\lambda r_1 + (1-\lambda)r_2 &= \lambda(t_1 z_0 - \mathbf{c}^T\mathbf{x}_1) + (1-\lambda)(t_2 z_0 - \mathbf{c}^T\mathbf{x}_2)\\ &= (\lambda t_1 + (1-\lambda)t_2)z_0 - \mathbf{c}^T(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2)\end{aligned}$$

  and observing that $\lambda t_1 + (1-\lambda)t_2 \geq 0$ and $\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \geq \mathbf{0}$ verifies membership in C. One proceeds in a similar fashion for the second component of $(r, \mathbf{w})$ given by $\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2$.

- The set C is a **cone**, where a non-empty set $C \subset \mathbb{R}^n$ is a cone if $\mathbf{x} \in C$ implies that any non-negative multiple of $\mathbf{x}$ is also in C, i.e. $\mathbf{x} \in C \implies \alpha\mathbf{x} \in C$ for all $\alpha \geq 0$. To check this property, assume that $(r, \mathbf{w}) = (tz_0 - \mathbf{c}^T\mathbf{x}, t\mathbf{b} - \mathbf{A}\mathbf{x}) \in C$, then $\alpha(r, \mathbf{w}) = ((\alpha t)z_0 - \mathbf{c}^T(\alpha\mathbf{x}), (\alpha t)\mathbf{b} - \mathbf{A}(\alpha\mathbf{x})) \in C$.

- The vector $(1, \mathbf{0})$ is **not** in C. To see this, assume in the contrary that there are values $t_1 \geq 0$ and $\mathbf{x}_1 \geq \mathbf{0}$ such that

$$(1, \mathbf{0}) = (t_1 z_0 - \mathbf{c}^T\mathbf{x}_1, t_1\mathbf{b} - \mathbf{A}\mathbf{x}_1)$$

  We distinguish two cases:

  - When $t_1 > 0$ then we divide by t_1 on both sides to get:

$$\left(\frac{1}{t_1}, \mathbf{0}\right) = \left(z_0 - \mathbf{c}^T\frac{\mathbf{x}_1}{t_1}, \mathbf{b} - \mathbf{A}\frac{\mathbf{x}_1}{t_1}\right)$$

    which says that $\mathbf{x}_1/t_1$ becomes a feasible solution. However, since z_0 is the minimal objective value, the objective value $\mathbf{c}^T\frac{\mathbf{x}_1}{t_1}$ for this feasible solution can be no smaller than z_0 and we must have

$$z_0 - \mathbf{c}^T\frac{\mathbf{x}_1}{t_1} \leq 0$$

    which is a contradiction to $z_0 - \mathbf{c}^T\frac{\mathbf{x}_1}{t_1} = \frac{1}{t_1} > 0$.

  - When $t_1 = 0$ then $\mathbf{x}_1$ cannot be the all-zero vector and we furthermore must have

$$(1, \mathbf{0}) = (-\mathbf{c}^T\mathbf{x}_1, -\mathbf{A}\mathbf{x}_1)$$

    or

$$\mathbf{c}^T\mathbf{x}_1 = -1$$

    Because of $\mathbf{A}\mathbf{x}_1 = \mathbf{0}$ we can, starting from our optimal solution $\mathbf{x}_0$ construct new feasible solutions as

$$\mathbf{x}_0 + \alpha\mathbf{x}_1$$

    for any $\alpha \in \mathbb{R}$, which will have objective value

$$\mathbf{c}^T(\mathbf{x}_0 + \alpha\mathbf{x}_1) = z_0 + \alpha\mathbf{c}^T\mathbf{x}_1 = z_0 - \alpha$$

    which by choosing positive α we can make as small as we want, contradicting optimality of z_0.

- The vector $(0, \mathbf{0})$ is in C: pick $t = 1$ and $\mathbf{x} = \mathbf{x}_0 \geq \mathbf{0}$.

- C is a closed set. You can take this for granted.

In summary, C is a closed convex set and point $(1, \mathbf{0})$ is not in that set. By the separating hyperplane theorem 6.2 there hence exists a vector $(s, \mathbf{z})$ such that

$$(s, \mathbf{z})^T \cdot (1, \mathbf{0}) < \inf \left\{ (s, \mathbf{z})^T \cdot (r, \mathbf{w}) : (r, \mathbf{w}) \in C \right\}$$

or

$$s < c := \inf \left\{ sr + \mathbf{z}^T \cdot \mathbf{w} : (r, \mathbf{w}) \in C \right\}$$

where the proof of the separating hyperplane theorem has shown that the infimum is actually assumed. We will next show that $c = 0$:

- We have that $c \geq 0$, as otherwise there would be some $(r, \mathbf{w})$ in C for which $sr + \mathbf{z}^T\mathbf{w} < 0$ holds. Since C is a cone, the point $\alpha(r, \mathbf{w})$ for $\alpha \geq 0$ would be in C as well and we would get

  $$\alpha(sr + \mathbf{z}^T\mathbf{w}) < 0$$

  and by choosing α large we can drive the left-hand side of this inequality towards negative infinity. However, this contradicts the assumption that c is always larger than s.
- By plugging the special point $(0, \mathbf{0}) \in C$ into $sr + \mathbf{z}^T \cdot \mathbf{w}$ we get that $c \leq 0$.

So, we have $c = 0$ which then implies that $s < 0$. The fact that $c = 0$ then implies

$$0 = \inf \left\{ sr + \mathbf{z}^T \cdot \mathbf{w} : (r, \mathbf{w}) \in C \right\}$$

or in other words

$$sr + \mathbf{z}^T\mathbf{w} \geq 0$$

for all $(r, \mathbf{w}) \in C$. When using the definition of C, this last condition becomes

$$\begin{aligned} 0 &\leq sr + \mathbf{z}^T\mathbf{w} \\ &= s(tz_0 - \mathbf{c}^T\mathbf{x}) + \mathbf{z}^T(t\mathbf{b} - \mathbf{A}\mathbf{x}) \\ &= (-s\mathbf{c}^T - \mathbf{z}^T\mathbf{A})\mathbf{x} + t\mathbf{z}^T\mathbf{b} + stz_0 \end{aligned} \tag{6.7}$$

which is true for all $t \geq 0$ and $\mathbf{x} \geq \mathbf{0}$. With the particular choice $t = 0$ we get

$$0 \leq (-s\mathbf{c}^T - \mathbf{z}^T\mathbf{A})\mathbf{x}$$

or equivalently

$$\mathbf{z}^T\mathbf{A}\mathbf{x} \leq -s\mathbf{c}^T\mathbf{x}$$

Since this holds for any non-negative vector $\mathbf{x}$, we can in fact conclude that

$$\mathbf{z}^T\mathbf{A} \leq -s\mathbf{c}^T$$

or after transposition

$$\mathbf{A}^T\mathbf{z} \leq -s\mathbf{c}$$

which then implies that in fact the vector $\frac{\mathbf{z}}{-s}$ is a feasible solution of the dual problem. On the other hand, with the special choice $\mathbf{x} = \mathbf{0}$ and $t = 1$ in Equation 6.7 we get

$$0 \leq \mathbf{z}^T\mathbf{b} + sz_0$$

or (remember $s < 0$)

$$(-s)z_0 \leq \mathbf{z}^T\mathbf{b} = \mathbf{b}^T\mathbf{z}$$

which says

$$\mathbf{b}^T\frac{\mathbf{z}}{-s} \geq z_0$$

which shows that the feasible solution $\frac{\mathbf{z}}{-s}$ of the dual has an objective value larger larger than or equal to z_0. The weak duality theorem 6.1 then actually implies that $\frac{\mathbf{z}}{-s}$ is optimal for the dual problem. This is what we had to show.

□

We conclude this section with another useful theorem.

Theorem 6.4 (Complementary slackness theorem)**.** *The feasible solutions* $\mathbf{x}$ *and* $\mathbf{z}$ *to a primal problem and its associated dual are optimal if and only if:*

1. $(\mathbf{c}^T - \mathbf{z}^T\mathbf{A})\mathbf{x} = 0$

2. $\mathbf{z}^T(\mathbf{Ax} - \mathbf{b}) = 0$

hold simultaneously.

Excursion (Proof of complementary slackness theorem)

Proof. We first assume that the primal is given in standard form

$$\begin{array}{rl} \text{minimize}_{[\mathbf{x}]} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

‘ $\Longrightarrow$ ‘ : Assume that $\mathbf{x}$ and $\mathbf{z}$ are optimal feasible solutions to the primal and the dual, respectively. Then the second condition holds trivially, as $\mathbf{x}$ is a feasible solution for the primal.

Since $\mathbf{x}$ and $\mathbf{z}$ are optimal, by the duality theorem 6.3 and using $\mathbf{Ax} = \mathbf{b}$ we have

$$\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{z} = \mathbf{z}^T\mathbf{b} = \mathbf{z}^T\mathbf{Ax}$$

which then implies $(\mathbf{c}^T - \mathbf{z}^T\mathbf{A})\mathbf{x} = 0$ as claimed.

‘ $\Longleftarrow$ ’ : suppose that $(\mathbf{c}^T - \mathbf{z}^T\mathbf{A})\mathbf{x} = 0$ holds. Then $\mathbf{c}^T\mathbf{x} = \mathbf{z}^T\mathbf{Ax} = \mathbf{z}^T\mathbf{b}$, and from the weak duality theorem 6.1 this implies optimality.

We give a second proof for the case where the primal is given in the “symmetric form”

$$\begin{array}{rl} \text{minimize}_{[\mathbf{x}]} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{Ax} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

‘ $\Longrightarrow$ ‘ : Assume that $\mathbf{x}$ and $\mathbf{z}$ are optimal feasible solutions to the primal and the dual, respectively. Then by the (strong) duality theorem 6.3 we have $\mathbf{c}^T\mathbf{x} = \mathbf{z}^T\mathbf{b}$. Using that $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{z} \geq \mathbf{0}$ we get

$$\begin{array}{rcl} (\mathbf{c}^T - \mathbf{z}^T\mathbf{A})\mathbf{x} & = & \mathbf{c}^T\mathbf{x} - \mathbf{z}^T\mathbf{Ax} \\ & = & \mathbf{z}^T\mathbf{b} - \mathbf{z}^T\mathbf{Ax} \\ & = & \mathbf{z}^T(\mathbf{b} - \mathbf{Ax}) \\ & \leq & 0 \end{array}$$

At the same time, since we have $\mathbf{z}^T\mathbf{A} \leq \mathbf{c}^T$ and $\mathbf{x} \geq \mathbf{0}$ we get $(\mathbf{c}^T - \mathbf{z}^T\mathbf{A})\mathbf{x} \geq 0$ which then implies $(\mathbf{c}^T - \mathbf{z}^T\mathbf{A})\mathbf{x} = 0$, giving the first equation. For the second condition, recall that $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{z} \geq \mathbf{0}$, giving $\mathbf{z}^T(\mathbf{Ax} - \mathbf{b}) \geq 0$. Furthermore, $\mathbf{z}^T\mathbf{A} \leq \mathbf{c}^T$ and $\mathbf{x} \geq \mathbf{0}$ implies

$$\mathbf{z}^T(\mathbf{Ax} - \mathbf{b}) = \mathbf{z}^T\mathbf{Ax} - \mathbf{z}^T\mathbf{b} = \mathbf{z}^T\mathbf{Ax} - \mathbf{c}^T\mathbf{x} = (\mathbf{z}^T\mathbf{A} - \mathbf{c}^T)\mathbf{x} \leq 0$$

proving the second equation.

‘ $\Longleftarrow$ ’ : if we put together both equations we get

$$\mathbf{c}^T\mathbf{x} = \mathbf{z}^T\mathbf{Ax} = \mathbf{zb} \tag{6.8}$$

and we can invoke the weak duality theorem 6.1 to conclude optimality.

□

6.2 Link-Weight Determination

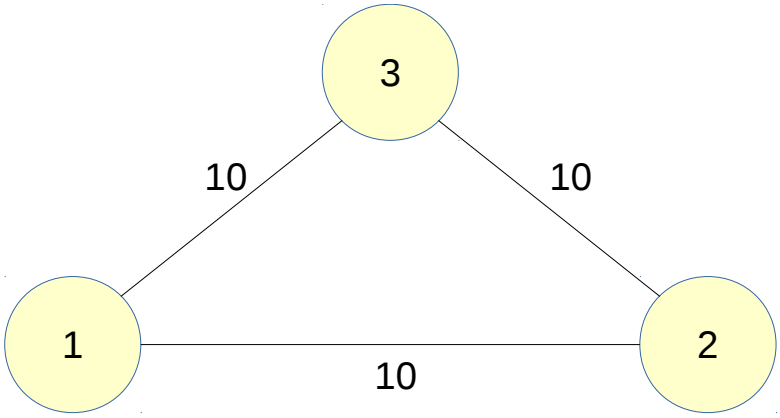
In the network flow problems we have discussed so far in Chapters 5 and 2 we have assumed that for a particular demand volume k any split over different paths was possible, e.g. to split one-third of the volume over the first

path and two-thirds over another one. However, shortest-path routing protocols like RIP, OSPF, IS-IS and others do not easily allow this kind of behaviour. In particular, in OSPF the routing of packets is completely determined by the **link weights** (or metric values) of the different hops, and the chosen routes / paths will always be the ones which have the smallest sum of link weights of all involved links. A key question then becomes: **How to choose the link weights to achieve or at least approximate an optimal solution for a given network flow problem?** This is the main question we address in this section, and the (partial) answer to it relies on the concept of duality introduced in Section 6.1.

6.2.1 Impact of Shortest-Path Routing

We will illustrate the impact of OSPF-style shortest-path routing using an example. In a system with L links indexed from 1 to L, we assume that the weight of link l is given by w_l.

We consider the load-balancing problem in the three-node network discussed in Section 2.1, repeated for reference here:



There is only one demand volume between nodes 1 and 2, indexed as demand volume 1, and two different paths: the direct path $1-2$ indexed as path 1 and the indirect path $1-3-2$ indexed as path 2. The link capacities are shown along the edges, and we are given a demand volume of $h_1 = 15$. Link $1-2$ is indexed as link 1, link $1-3$ as link 2 and link $3-2$ as link 3. If our objective is load-balancing, then the optimal choice is to send $x_{11} = 7.5$ units of flow through path 1 and $x_{12} = 7.5$ units of flow through path 2. Now suppose the following cases for the allocation of link weights:

- Assume we just assign the same weight to all links to achieve minimum-hop routing, e.g. $w_1 = w_2 = w_3 = 1$. Then OSPF will assign a total cost of 1 to path $1-2$ and a total cost of $1+1=2$ to path $1-3-2$, as OSPF adds up the link weights of a path to get the cost of a path. In this case node 1 will pick path $1-2$ as the shortest path to node 2 and will send all packets over this path, which becomes overloaded.
- When the equal-cost multi-path (ECMP) option of OSPF is activated, then we can achieve the desired result by making sure that both paths have the same total cost, for example by assigning the following link weights:

$$w_1 = 2 \quad , \quad w_2 = 1 \quad , \quad w_3 = 1$$

As a variation of this problem we consider the case that the link capacities are not all the same, but rather link 1 (between $1-2$) has a capacity of 20 units ($c_1 = 20$), whereas links 2 (between $1-3$) and 3 (between $2-3$)

have capacities of $c_2 = c_3 = 10$ each. Under the load-balancing objective the optimal allocation is to send $x_{11} = 10$ units of flow through the direct path (giving a utilization of $\frac{x_{11}}{c_1} = 0.5$) and $x_{12} = 5$ units of flow through the indirect path (with the same utilization $\frac{x_{12}}{c_2} = 0.5$, using that both links along this path have the same capacity). In other words, unequal amounts of flow are sent over the two different paths. With OSPF we have two distinct options for assigning link weights:

- We can assign them such that paths $1-2$ and $1-3-2$ have different total costs. When path $1-3-2$ has the smaller total cost, then all traffic will be routed along that path, which, however, will not have sufficient capacity, making this solution $x_{12} = 15$ infeasible. When path $1-2$ has the smaller total cost, the solution $x_{11} = 15$ becomes feasible with a maximum utilization of $\frac{x_{11}}{c_1} = 0.75$.
- We can assign them such that both paths have the same total cost, in which case the ECMP facility will lead to $x_{11} = x_{12} = 7.5$ units of flow per path, and the utilizations among both paths are:

$$\frac{x_{11}}{c_1} = \frac{7.5}{20} = 0.375 \qquad , \qquad \frac{x_{12}}{c_2} = \frac{7.5}{10} = 0.75$$

In other words, we have no chance to achieve the theoretical optimum utilization of 0.5 on both paths, but we nonetheless have to find a system of link weights which satisfies all the given constraints (demand and capacity constraints) and which at least approximates the optimal, unconstrained flow.

6.2.2 Problem Formulation

For reference we repeat here the unconstrained formulation for the general load-balancing problem in the notation introduced in Chapter 5 (compare Equation 5.5, here we have eliminated the dependent variables y_l):

$$\begin{aligned} \text{minimize}_{[\mathbf{x},r]} \quad & r \\ \text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1,\ldots,K\} \\ & \sum_{k=1}^{K}\sum_{p=1}^{P_k} x_{kp}\delta_{kpl} \le c_l r \qquad \text{for } l \in \{1,\ldots,L\} \\ & x_{kp} \ge 0 \qquad \text{for } k \in \{1,\ldots,K\}, p \in \{1,\ldots,P_k\} \\ & r \ge 0 \end{aligned} \tag{6.9}$$

The key point to realize is that with introducing shortest-path routing like in OSPF we have to change our decision variables: the only thing we can adjust are the link weights w_l, and the former decision variables x_{kp} (the amount of flow of demand volume k to send over the p-th path for this demand volume) now become **dependent variables**, i.e. they become a function of the link weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_L)^T$, and conceptually the problem formulation changes into

$$\begin{aligned} \text{minimize}_{[\mathbf{w},r]} \quad & r \\ \text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp}(\mathbf{w}) = h_k \qquad \text{for } k \in \{1,\ldots,K\} \\ & \sum_{k=1}^{K}\sum_{p=1}^{P_k} x_{kp}(\mathbf{w})\delta_{kpl} \le c_l r \qquad \text{for } l \in \{1,\ldots,L\} \\ & x_{kp}(\mathbf{w}) \ge 0 \qquad \text{for } k \in \{1,\ldots,K\}, p \in \{1,\ldots,P_k\} \end{aligned} \tag{6.10}$$

$$
\begin{aligned}
& r \geq 0 \\
& w_l \in \mathcal{W} \qquad \text{for } l \in \{1, \ldots, L\}
\end{aligned}
$$

where the set $\mathcal{W}$ is the set of allowable values for the link weights. In OSPF for example link weights are restricted to be integers between 0 and 65,535.

We make some important observations about the new problem 6.10:

- This problem is not a linear program anymore, as the dependency of the x_{kp} on the weight vector $\mathbf{w}$ will in general be more complex than just linear. This dependency has not been made explicit in the formulation and will generally be hard to obtain, not to mention finding an explicit formula for it.
- When the link-weight set $\mathcal{W}$ is discrete, the problem might turn into a nonlinear and possibly NP-hard integer optimization problem. In the case of OSPF with its 64k different values one might (through scaling and translation) reach a situation where the weights can be approximated by real numbers in some interval (which could be handled easily in an LP), but for a protocol like RIP with just 15 allowed values for the link weight this illusion becomes harder to maintain.
- Even when we assume that problem 6.10 is still linear at the bottom of its heart, we are nonetheless **introducing new constraints** as compared to the original problem 6.9. In broad generality, introducing new constraints to a problem in addition to already existing ones can only reduce the size of the feasible set, and hence the optimally achievable objective value can increase (as options are being taken away). So, we generally have to expect that the shortest-path versions of our network flow problems (minimum-cost, load-balancing, average delay) will not achieve the same optimal values as the "free" versions of the problems (i.e. the ones without the constraints introduced by shortest-path routing).
- In the light of the previous point, we can actually regard Problem 6.9 as a **relaxation** of Problem 6.10.

6.2.3 Duality-Based Link-Weight Determination

We will consider the dual problem of Problem 6.9. To do so, we first re-write the smaller-than-or-equal constraint in the primal to get the new primal

$$
\begin{aligned}
\text{minimize}_{[\mathbf{x},r]} \quad & r \qquad (6.11) \\
\text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1, \ldots, K\} \\
& -\sum_{k=1}^{K} \sum_{p=1}^{P_k} x_{kp} \delta_{kpl} + c_l r \geq 0 \qquad \text{for } l \in \{1, \ldots, L\} \\
& x_{kp} \geq 0 \qquad \text{for } k \in \{1, \ldots, K\}, p \in \{1, \ldots, P_k\} \\
& r \geq 0
\end{aligned}
$$

We introduce the dual variables v_k ($1 \leq k \leq K$) for the equality constraints (which will be un-restricted) and the dual variables π_l ($1 \leq l \leq L$) for the greater-than-or-equal constraints. With this, the dual problem

becomes:

$$\begin{aligned} \text{maximize}_{[\mathbf{v},\boldsymbol{\pi}]} \quad & \sum_{k=1}^{K} h_k v_k \\ \text{subject to} \quad & v_k - \sum_{l=1}^{L} \delta_{kpl}\pi_l \leq 0 \qquad \text{for } k \in \{1,\ldots,K\}, p \in \{1,\ldots,P_k\} \\ & \sum_{l=1}^{L} c_l \pi_l \leq 1 \\ & \pi_l \geq 0 \qquad \text{for } l \in \{1,\ldots,L\} \end{aligned} \tag{6.12}$$

Note that in this problem only the decision variables v_k feature in the objective, as the right-hand side for the inequality constraints in 6.11 is zero. After re-writing the first constraint we get

$$\begin{aligned} \text{maximize}_{[\mathbf{v},\boldsymbol{\pi}]} \quad & \sum_{k=1}^{K} h_k v_k \\ \text{subject to} \quad & v_k \leq \sum_{l=1}^{L} \delta_{kpl}\pi_l \qquad \text{for } k \in \{1,\ldots,K\}, p \in \{1,\ldots,P_k\} \\ & \sum_{l=1}^{L} c_l \pi_l \leq 1 \\ & \pi_l \geq 0 \qquad \text{for } l \in \{1,\ldots,L\} \end{aligned} \tag{6.13}$$

Now, suppose that $(\mathbf{x}^*, r^*)$ is an optimal solution of the primal 6.11 and $(\mathbf{v}^*, \boldsymbol{\pi}^*)$ is an optimal solution of the dual. Then the following must be true:

- The vector $(\mathbf{x}^*, r^*)$ must satisfy all constraints of the primal and similarly $(\mathbf{v}^*, \boldsymbol{\pi}^*)$ satisfies all constraints of the dual.
- The complementary slackness conditions (compare Theorem 6.4) give:

$$r^* \left(\sum_{l=1}^{L} c_l \pi_l^* - 1 \right) = 0 \qquad \text{for } 1 \leq l \leq L \tag{6.14}$$

$$x_{kp}^* \left(v_k^* - \sum_{l=1}^{L} \delta_{kpl}\pi_l^* \right) = 0 \qquad \text{for } 1 \leq k \leq K, 1 \leq p \leq P_k \tag{6.15}$$

With reference to the interpretation of the dual variables as prices or "commodity costs" (compare Section 6.1.2) the second complementary slackness equation 6.15 implies that for any path p of demand volume k for which $x_{kp}^* > 0$ holds (i.e. for any of the available paths that is actually used), we need to have

$$v_k^* = \sum_{l=1}^{L} \delta_{kpl}\pi_l^*$$

so the "cost" of each path is the sum of all π_i^* of all the links involved in this path. This is eerily similar to the way the costs of a path are calculated in shortest-path settings (sum of the costs of all involved links), and this actually suggests to use the link-weight system

$$w_l = \pi_l^* \tag{6.16}$$

for $l = 1, \ldots, L$. In other words, in this example **we obtain the solution to the dual problem 6.12 and use this to build our link weights!**

We will use a second example to illustrate this approach. We consider the minimum-cost routing problem

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \sum_{k=1}^{K} \sum_{p=1}^{P_k} \phi_{kp} x_{kp} \\ \text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1, \ldots, K\} \\ & \sum_{k=1}^{K} \sum_{p=1}^{P_k} x_{kp} \delta_{kpl} \leq c_l \qquad \text{for } l \in \{1, \ldots, L\} \\ & x_{kp} \geq 0 \qquad \text{for } k \in \{1, \ldots, K\}, p \in \{1, \ldots, P_k\} \end{aligned} \tag{6.17}$$

where we introduce the additional assumption that the cost ϕ_{kp} of the p-th path for the k-demand volume is actually the sum of per-link costs $\widehat{\phi}_l$ for links $1 \leq l \leq L$, i.e. we have

$$\phi_{kp} = \sum_{l=1}^{L} \delta_{kpl} \widehat{\phi}_l$$

With this and with negating the smaller-than-or-equal constraints the problem formulation becomes

$$\begin{aligned} \text{minimize}_{[\mathbf{x}]} \quad & \sum_{k=1}^{K} \sum_{p=1}^{P_k} \left(\sum_{l=1}^{L} \delta_{kpl} \widehat{\phi}_l \right) x_{kp} \\ \text{subject to} \quad & \sum_{p=1}^{P_k} x_{kp} = h_k \qquad \text{for } k \in \{1, \ldots, K\} \\ & -\sum_{k=1}^{K} \sum_{p=1}^{P_k} x_{kp} \delta_{kpl} \geq -c_l \qquad \text{for } l \in \{1, \ldots, L\} \\ & x_{kp} \geq 0 \qquad \text{for } k \in \{1, \ldots, K\}, p \in \{1, \ldots, P_k\} \end{aligned} \tag{6.18}$$

Again we introduce unrestricted dual variables v_k ($1 \leq k \leq K$) for the demand constraints (which are equality constraints) and non-negative dual variables π_l ($1 \leq l \leq L$) for the link capacity constraints. The dual of this problem is given by:

$$\begin{aligned} \text{maximize}_{[\mathbf{v}, \boldsymbol{\pi}]} \quad & \sum_{k=1}^{K} h_k v_k - \sum_{l=1}^{L} c_l \pi_l \\ \text{subject to} \quad & v_k - \sum_{l=1}^{L} \delta_{kpl} \pi_l \leq \sum_{l=1}^{L} \delta_{kpl} \widehat{\phi}_l \qquad \text{for } k \in \{1, \ldots, K\} \\ & \pi_l \geq 0 \qquad \text{for } l \in \{1, \ldots, L\} \end{aligned} \tag{6.19}$$

and re-writing the inequality constraint in the dual gives

$$\begin{aligned} \text{maximize}_{[\mathbf{v}, \boldsymbol{\pi}]} \quad & \sum_{k=1}^{K} h_k v_k - \sum_{l=1}^{L} c_l \pi_l \\ \text{subject to} \quad & v_k \leq \sum_{l=1}^{L} \delta_{kpl} (\pi_l + \widehat{\phi}_l) \qquad \text{for } k \in \{1, \ldots, K\} \\ & \pi_l \geq 0 \qquad \text{for } l \in \{1, \ldots, L\} \end{aligned} \tag{6.20}$$

When $\mathbf{x}^*$ is an optimal solution of the primal 6.18 and $(\mathbf{v}^*, \boldsymbol{\pi}^*)$ is an optimal solution of the dual 6.20 then:

- $\mathbf{x}^*$ and $(\mathbf{v}^*, \boldsymbol{\pi}^*)$ satisfy the constraints of the primal and dual, respectively.
- Relevant complementary slackness conditions are:

$$\pi_l^* \left(c_l - \sum_{k=1}^{K} \sum_{p=1}^{P_k} \delta_{kpl} x_{kp}^* \right) = 0 \qquad \text{for } 1 \in \{1, \ldots, L\} \tag{6.21}$$

$$x_{kp}^* \left(v_k^* - \sum_{l=1}^{L} \delta_{kpl} (\pi_l^* + \widehat{\phi}_l) \right) = 0 \qquad \text{for } k \in \{1, \ldots, K\}, p \in \{1, \ldots, P_k\} \tag{6.22}$$

Referring again to the interpretation of v_k^* as the "cost" of demand volume k, we get from the complementary slackness condition that for a path for which $x_{kp}^* > 0$ holds we must have

$$v_k^* = \sum_{l=1}^{L} \delta_{kpl} (\pi_l^* + \widehat{\phi}_l) \qquad \text{for } k \in \{1, \ldots, K\}, p \in \{1, \ldots, P_k\}$$

Similar to above, this suggests to set the link weights as

$$w_l = \pi_l^* + \widehat{\phi}_l \tag{6.23}$$

A few comments are in order:

- In both cases we can suggest a link weight system from the optimal solution of the dual problem. This is in general not hard to obtain. In fact, CPLEX routinely solves both the given primal problem and the associated dual problem, and you can print the optimal solution values for the dual by giving the command:

  ```
  display solution dual
  ```

- Note that the link-weight systems 6.16 and 6.23 are not identical. In general, the link weight system depends on the objective function.
- All paths induced by the link-weight system are just the shortest paths with respect to this system.
- The calculated link-weight systems 6.16 and 6.23 might possibly need to be scaled or translated to get values which comply with the allowable set of link weights for a given routing protocol.

Part II

Labs

Chapter 7

Labs

In the labs for this module you will first solve some instances of the problems introduced in Section 4.1 using the CPLEX solver by IBM, an industrial-strength optimization tool.[1] The main purpose of this first part is to give you some first experience with this tool and with the formulation of linear programs. Next, you will go manually through some example problems to deepen your understanding of the simplex algorithm and its application. Finally, you will formulate some network flow and other problems and solve these using CPLEX. You will also use CPLEX for the second assignment. We will use CPLEX only in the most basic way.

It is important that you read this chapter **before** the actual lab, since otherwise you will risk to have insufficient time to get everything done!! However, you should not expect that the notes below will contain *everything* you need – in fact, you are expected to figure out several details on your own.

This chapter contains all the information and problems about the lab work for this module. I expect that you will need between three or four weeks to complete the problems. There is no need to finish a certain amount of problems in any one week, but at the end of this module you should have completed all of them.

7.1 The CPLEX Solver

7.1.1 Installation

There are two options for you to obtain and install the CPLEX software. We will only use the command line version of CPLEX, the Linux version also provides an Eclipse-based GUI.

The Community Edition IBM provides a community edition of CPLEX under the following URL:

`http://www-01.ibm.com/software/websphere/products/optimization/cplex-studio-community-edition/`

(which you could also find simply by googling for "IBM CPLEX"). This version contains all the functionality of the full version but is restricted (at the time of writing this) to 1,000 decision variables and 1,000 constraints, which will be sufficient for our purposes.

Follow the instructions on the IBM website to register with IBM (in doing that it is apparently necessary to set the region to "New Zealand", otherwise registration might fail), download the community edition and install it

[1] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

on the platform of your choice – the installation itself is essentially the same as for the full version, described below. I recommend to download using http, otherwise IBM insists on using their own download tool, which is Java-based and requires a java plugin being activated in your browser. The download might take a while and is a bit voluminous. If you work on your Linux home directories in the labs, you might not have sufficient quota to download the installer and run it. In that case please get in touch with the programmers to increase your quota.

The Full Version The second option is through me. I am registered with IBM's Academic Initiative and through this have obtained the full version of CPLEX for research and class-room purposes – by a full version here I mean a version that has no limitations in the number of variables or constraints. I can supply you with a binary for CPLEX version 12.6.1 on either 64-bit Linux, 64-bit Windows or 64-bit Mac OS X, but I have to register your name and you have to agree in writing that you do not distribute the software any further and do not use it for anything other than this course.

The installation of the software is platform-dependent. Under Linux, you will get from the file

```
CPLEX_OPTIM_STUDIO_12.6.1_LNXX86-.bin
```

which you need to store in some temporary place. Next run the installer using the commands

```
$ chmod +x CPLEX_OPTIM_STUDIO_12.6.1_LNXX86-.bin
$ ./CPLEX_OPTIM_STUDIO_12.6.1_LNXX86-.bin
```

(where the dollar sign indicates the command prompt and is not typed). Follow the instructions of the installer. You need to choose your locale (or preferred language), accept the license agreement, and choose an installation directory, preferrably somewhere under your home directory (you need to have write permission). The installation might take some time. Once the installer has finished, it is a good idea to add the directory containing the main CPLEX binary to your search path, so that you can just use the command `cplex` instead of typing the whole path each time. This works as follows for the `bash` shell. Suppose I have chosen `/home/awillig/cplex` as the installation directory. Then perform the following steps:

- Load the file `/.bashrc` into an editor (e.g. `emacs` or `vi`).
- Go to the bottom of the file, add a new line

  ```
  export PATH=$PATH:/home/awillig/cplex/cplex/bin/x86-64_linux
  ```

- Save the file.
- Stop the shell and start a new one so that the changes take effect.
- Use the command `which cplex` to check whether the `cplex` executable is now in your path.

CPLEX also comes with documentation in HTML format. Open a browser and point it to

```
file:///home/awillig/cplex/doc/html/en-US/documentation.html
```

where `/home/awillig/cplex` is my choice of installation directory and `en-US` is my choice of locale / language, which you must replace by your choices. On this site especially the pages under "CPLEX engine" are relevant. You should have a look at the tutorials, in particular the first two (the navigation needs some getting used to).

7.1.2 Interactive Usage

We will be working with the interactive / command line version of CPLEX, in which you enter commands into CPLEX directly and which you start by entering the command `cplex` on the command line (assuming you have the `cplex` binary in your search path, otherwise you must specify the full path). Note that CPLEX is also usable as a library in the C++, Java and Python programming languages, where you use these languages to build a problem description and call into the CPLEX library to set up the problem, run the optimization and extract results.

We have already seen an example CPLEX session in Section 2.1.1, where we have provided the problem specification in an external file (called `tm.lp` in that example), and have used the following commands:

```
CPLEX> read tm.lp
Problem 'tm.lp' read.
Read time = 0.00 sec. (0.00 ticks)
CPLEX> optimize
Tried aggregator 1 time.
LP Presolve eliminated 3 rows and 2 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec. (0.00 ticks)

Dual simplex - Optimal:  Objective =  1.3400000000e+02
Solution time =    0.00 sec.  Iterations = 0 (0)
Deterministic time = 0.00 ticks  (3.08 ticks/sec)

CPLEX> display solution variables -
Variable Name           Solution Value
x12                          10.000000
x132                          7.000000
CPLEX>
```

Note that this output shows the commands given after the `CPLEX>` prompt and the response of CPLEX follows in the next line.

The interactive optimizer (the `cplex` program) actually allows to construct optimization problems on the fly, you can add constraints or variables interactively or change parameters. I recommend to put the problem specification into a separate file and only use the following commands in the interactive optimizer:

- The command `read <filename>` instructs CPLEX to read a problem specification from a given file. Under Linux the command line editor of `cplex` unfortunately does not have all the convenient features of the `bash` like history, completion etc. So I recommend to keep filenames short to reduce the amount of typing.
- The command `optimize` runs the actual calculations for performing the optimization. CPLEX has a range of algorithms, including variants of the simplex algorithm.
- The command `display` in general outputs various bits and pieces of information, the specific incarnation

  ```
  display solution variables -
  ```

  from above shows the values of **all** decision variables after the optimization. If you want to see only some, then you can enter

```
display solution variables x1*
```

which will display all solution variables starting with `x1`.

- With the command `quit` (or simply by pressing Control+D under Linux) you can leave the interactive optimizer.

Names of Input Files
You can put a problem specification into external input files and read these into CPLEX. CPLEX supports different input formats and distinguishes these based on the file name suffix / extension. We will only use the "LP format" (see below), and the files need to have extension `.lp`

Problem 7.1.1 (A First Test)**.**
Go back to Section 2.1.1, create an LP file with the model specification given there, start `cplex` and run through the same sequence of commands. You should see the same outputs.

7.1.3 Batch Usage

There is a second, highly useful way to use the `cplex` program: instead of entering commands interactively you can supply them in the command line, and after executing them CPLEX stops. To run the above model, you can run the following invocation:

```
$ cplex -c "read tm.lp" "optimize" "display solution variables -"

Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.6.1.0
with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2014.  All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX> Problem 'tm.lp' read.
Read time = 0.29 sec. (0.00 ticks)
CPLEX> Tried aggregator 1 time.
LP Presolve eliminated 3 rows and 2 columns.
All rows and columns eliminated.
Presolve time = 0.01 sec. (0.00 ticks)

Dual simplex - Optimal:  Objective =  1.3400000000e+02
Solution time =    0.07 sec.  Iterations = 0 (0)
Deterministic time = 0.00 ticks  (0.04 ticks/sec)
```

```
CPLEX> Variable Name           Solution Value
x12                           10.000000
x132                           7.000000
CPLEX> $
```

The `cplex` executable writes its output to `stdout`. This means that you can put your `cplex` invocations into a script and run this in batch mode!

7.1.4 LP File Format

The file format is best explained using an example:

```
Minimize
  5 x12 + 12 x132
Subject to
  demandflow:   x12 + x132 = 17
  capp1:        x12  <= 10
  capp2:        x132 <= 12
Bounds
  0 <= x12
  0 <= x132
End
```

The file is sub-divided into a number of sections:

- In the `Minimize` section you give the objective function, i.e. the function to minimize. The function consists of a linear combination (or simple sum) of terms, where each term is a constant value (you can leave that away if the constant is one) and a variable name. Do not use an asterisk between the constant and the variable.
- Next comes the mandatory constraints section, which usually starts with the string `Subject to`. A constraint has the following components:
  - An optional constraint name (which is useful for documentation purposes), which starts with a letter, followed by a sequence of letters and numbers, and ends with a colon.
  - A linear combination of the decision variables, following the same format as for the objective function.
  - A "sense indicator" for the constraint, which can be one of =, <, <=, >, or >=.
  - The "right hand side", which is a constant.
- The `Bounds` section is optional and can be used to introduce constraints on the smallest and largest value that a decision variable may take. If a decision variable is not mentioned, it is unconstrained.
- If some of the variables have to be integer variables, then you can specify this in the optional `Integer` section (which must come after a `Bounds` section). An integer variable must always be bounded (i.e. have a lower and upper bound). If no bounds are specified in the `Bounds` section, then the integer variable is automatically assumed to be binary, i.e. can only take the values 0 and 1.
- The problem specification must be ended with the keyword `End` on a separate line. You can in fact add further text after the `End` line, which will be ignored by CPLEX and which you can use for documentation purposes.

A more detailed explanation of the file format can be found here:

`http://lpsolve.sourceforge.net/5.0/CPLEX-format.htm`

7.2 First Steps with CPLEX

The problems in this section serve as a "warm-up" to the work with CPLEX. The problems will be relatively simple and the focus is on establishing the LP files and extracting results out of them. We will also use a bit of shell-scripting to extract and process results.

Problem 7.2.1 (A first problem)**.**
Consider again the single-commodity minimum-cost routing problem from Section 2.1.1, and in particular the problem posed in Problem 2.1.2, where for definiteness we assume that $c_{12} = c_{132} = 10$, $\phi_{132} = 5$ and $\phi_{12} = 10$.

- Write an LP file for the specific demand volume $h = 12$.
- Write a program / script in a programming language of your choice (you should be able to invoke shell commands from it and read back the standard output) which:
  - Loop over a range of values for h, with $h \in \{1, 1.1, 1.2, \ldots, 18.9, 19.0\}$
  - For each value of h create an LP file, invoke CPLEX to solve it and extract the optimal value for the decision variable x_{12}.
  - Write the value of h and the resulting x_{12} as a single line to `stdout`.
- Run your script and redirect `stdout` to some file (the "output file"). Plot the contents of the output file with h on the x-axis and the corresponding x_{12} on the y-axis. You can use a plotting tool of your choice. One option is `gnuplot`, which has a somewhat particular syntax but has the big advantage that it is scriptable.

The aim of this problem is not to solve a terribly interesting or demanding problem, but rather to get you used to the idea of constructing LPs (or in our case: LP files) from within another program on the fly, and do something with the results.
As for the programming language you can use python. The python module `subprocess` actually allows you to stitch together command line incantations from within python and retrieve the `stdout` for further processing.

Problem 7.2.2 (An instance of the diet problem)**.**
We consider the diet problem described in Section 4.1.1. Write an LP file to solve the following problem instance:

$$\begin{aligned} (c_1, c_2, c_3, c_4) &= (1, 9, 3.3, 1.7) \\ (b_1, b_2, b_3) &= (1, 6, 8) \\ \mathbf{A} &= \begin{pmatrix} 1/2 & 1/5 & 1/3 & 0 \\ 0 & 0 & 1/10 & 1/3 \\ 0 & 1/2 & 0 & 1/2 \end{pmatrix} \end{aligned}$$

Problem 7.2.3 (An instance of the wireless problem)**.**
We consider the wireless communications problem described in Section 4.1.2 and in particular in Problem 4.1.2 (but without upper bounds on the transmit powers). Write an LP file to solve the following problem instance:

$$\begin{aligned}(\gamma_1, \gamma_2, \gamma_3) &= (1, 2, 3)\\ h_{i,i} &= 0.5\\ h_{i,j} &= 0.02\\ \sigma^2 &= 0.01\end{aligned}$$

i.e. all the "direct attenuation factors" $h_{i,i}$ have the same value 0.5, and all the "interference attenuation factors" have the (much smaller!) value 0.02. The noise power is also the same for all base stations.

Problem 7.2.4 (An instance of the transportation problem)**.**
We consider the first transportation problem described in Section 4.1.3, more specifically in Problem 4.1.3. Write an LP file to solve the following problem instance:

$$\begin{aligned}(a_1, a_2, a_3) &= (3, 4, 5)\\ (b_1, b_2, b_3) &= (2, 4, 6)\\ c_{i,j} &= 1\end{aligned}$$

Before you run this, think about the following question: this problem will have six constraints and nine decision variables. How many decision will at most be non-zero, and why? And is the solution to this instance degenerate or not?

7.3 Manually Solving LPs

In this section you should solve a number of LPs manually by running through the steps of the simplex algorithm. Where necessary, go through both phases for all the problems, where:

- First phase: find an initial basic feasible solution $\mathbf{x}$ for the given problem by applying the simplex procedure to the auxiliary problem as described in Section 4.5.4.
- Second phase: use the basic feasible solution found in the previous step as your starting solution for the simplex procedure run on the original problem, and solve it.

Problem 7.3.1 (Applying the simplex algorithm)**.**
Solve the following problem manually:

$$\begin{aligned} \text{minimize} \quad & -x_1 + x_2 \\ \text{subject to} \quad & x_1 - x_2 \leq 2 \\ & x_1 + x_2 \leq 6 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$

Show your workings. In particular, carefully write out the problem which you get after transformation to standard form, and the auxiliary problem for finding an initial basic feasible solution. Show all relevant tableaus.

Solution 7.3.1.

As we have "smaller-than" constraints in the problem, we first need to re-write it to put it in standard form. Applying the rule from Section 4.2.2, we introduce two slack variables x_3 and x_4, and with these the problem becomes:

$$\begin{aligned} \textit{minimize} \quad & -x_1 + x_2 \\ \textit{subject to} \quad & x_1 - x_2 + x_3 = 2 \\ & x_1 + x_2 + x_4 = 6 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_3 \geq 0 \\ & x_4 \geq 0 \end{aligned}$$

We are now in the very lucky situation that for this revised problem we do not have to go through the first phase, as an initial basic feasible solution of this problem is available immediately:

$$\begin{aligned} x_3 &= 2 \\ x_4 &= 6 \end{aligned}$$

At this point you can realize that in fact the approach to obtain an initial basic feasible solution described in Section 4.5.4 is really the same technique as introducing slack variables for "smaller-than" problems!

With this, our initial tableau becomes:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{b}$
$\boxed{1}$	-1	1	0	2
1	1	0	1	6
-1	1	0	0	0

where x_3 and x_4 are currently our basic variables, and their respective columns are already cleared out. From looking at the relative cost values (initialized with the coefficients of the cost vector) we see that only the first variable x_1 has a strictly negative value, so we pick x_1 to bring into the basis. The pivot is highlighted: both entries in the first column are positive, but the first one has the smaller ratio to the corresponding value in the right-hand-side column:

$$\frac{y_{0,1}}{y_{1,1}} = \frac{2}{1} = 2 \qquad , \qquad \frac{y_{0,2}}{y_{2,1}} = \frac{6}{1} = 6$$

and as a result we will swap x_1 against x_3. Cleaning out the first column by:

- *subtracting the first row from the second*
- *adding the first row to the third*

gives the updated tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{b}$
1	-1	1	0	2
0	$\boxed{2}$	-1	1	4
0	0	1	0	2

This tableau does not contain strictly negative relative cost values, so we cannot improve the value of the objective function (-2) anymore. In this particular case we have $x_1 = 2$, and the variable x_2 (which features in the objective function) is not actually in our current basis, so we must set it to 0 (CPLEX agrees to this). We can finish the simplex procedure here.

We could actually continue the calculations to finally bring x_2 into the basis, replacing x_4. The pivot element (highlighted in the previous tableau) indicates that this can be done. In this particular case, both columns 2 and 4 have relative cost values of 0 so that by exchanging them we do not change our objective cost. We can exchange them through the pivoting procedure by first dividing the second row by 2, giving the tableau

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{b}$
1	-1	1	0	2
0	1	$-1/2$	$1/2$	2
0	0	1	0	2

and then we clean out the second column by adding the second row to the first row, giving us the tableau

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{b}$
1	0	$1/2$	$1/2$	4
0	1	$-1/2$	$1/2$	2
0	0	1	0	2

So, we have another solution $x_1 = 4$ and $x_2 = 2$, which again gives the value -2 for the objective function.

So, we have seen here that there can be several solutions which attain the minimum value of the objective function.

Problem 7.3.2 (Applying the simplex algorithm)**.**
Solve the following problem manually:

$$\begin{aligned} \text{minimize} \quad & -3x_1 + x_2 + 3x_3 - x_4 \\ \text{subject to} \quad & x_1 + 2x_2 - x_3 + x_4 = 0 \\ & 2x_1 - 2x_2 + 3x_3 + 3x_4 = 9 \\ & x_1 - x_2 + 2x_3 - x_4 = 6 \\ & x_1 \geq 0 \\ & \dots \\ & x_4 \geq 0 \end{aligned}$$

Show your workings. In particular, carefully write out the auxiliary problem for finding an initial basic feasible solution. Show all relevant tableaus.

Solution 7.3.2.

This problem is already in standard form, and there appears to be no obvious guess for an initial feasible solution, so we have to go through the two-phase procedure.

Since we have three constraints, we need to add three auxiliary variables x_5, x_6 and x_7, and the auxiliary problem becomes

$$\begin{array}{ll} \textit{minimize} & x_5 + x_6 + x_7 \\ \textit{subject to} & x_1 + 2x_2 - x_3 + x_4 + x_5 = 0 \\ & 2x_1 - 2x_2 + 3x_3 + 3x_4 + x_6 = 9 \\ & x_1 - x_2 + 2x_3 - x_4 + x_7 = 6 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0 \\ & x_5 \geq 0, x_6 \geq 0, x_7 \geq 0 \end{array}$$

The initial tableau for this problem is:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
1	2	−1	1	1	0	0	0
2	−2	3	3	0	1	0	9
1	−1	2	−1	0	0	1	6
0	0	0	0	1	1	1	0

We first need to clean out the columns for variables x_5, x_6 and x_7 in this tableau to make them proper basic variables. We do this by:

- *subtracting row 1 from row 4*
- *subtracting row 2 from row 4*
- *subtracting row 3 from row 4*

The resulting tableau is:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
$\boxed{1}$	2	−1	1	1	0	0	0
2	−2	3	3	0	1	0	9
1	−1	2	−1	0	0	1	6
−4	1	−4	−3	0	0	0	−15

We now swap variables x_5 and x_1 (since x_1 has the smallest relative cost value, and within this column the pivot could be uniquely identified) to bring x_1 into the basis, and so we clean out the first column by:

- *subtracting two times the first row from the second row*
- *subtracting the first row from the third row*
- *adding four times the first row to the fourth row*

to give the new tableau

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
1	2	−1	1	1	0	0	0
0	−6	$\boxed{5}$	1	−2	1	0	9
0	−3	3	−2	−1	0	1	6
0	9	−8	1	4	0	0	−15

By choosing the highlighted pivot (x_3 is the only variable with a negative relative cost value and the highlighted positive entry has the smaller ratio to the last column as compared to the other entry in this column), we can next exchange variables x_3 and x_6 by first dividing the second row by 5, giving

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
1	2	-1	1	1	0	0	0
0	$-6/5$	1	$1/5$	$-2/5$	$1/5$	0	$9/5$
0	-3	3	-2	-1	0	1	6
0	9	-8	1	4	0	0	-15

and then cleaning out the third column by:

- *adding second row to the first one*
- *subtracting three times the second rom from the third one*
- *adding eight times the second row to the fourth one*

giving the next tableau

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
1	$4/5$	0	$6/5$	$3/5$	$1/5$	0	$9/5$
0	$-6/5$	1	$1/5$	$-2/5$	$1/5$	0	$9/5$
0	$\boxed{3/5}$	0	$-13/5$	$1/5$	$-3/5$	1	$3/5$
0	$-3/5$	0	$13/5$	$4/5$	$8/5$	0	$-3/5$

By the chosen pivot, we can now exchange variables x_2 and x_7. We first divide the third row by $3/5$ to get

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
1	$4/5$	0	$6/5$	$3/5$	$1/5$	0	$9/5$
0	$-6/5$	1	$1/5$	$-2/5$	$1/5$	0	$9/5$
0	1	0	$-13/3$	$1/3$	-1	$5/3$	1
0	$-3/5$	0	$13/5$	$4/5$	$8/5$	0	$-3/5$

and then clean out the second column by:

- *subtracting $4/5$ times the third row from the first row*
- *adding $6/5$ times the third row to the second row*
- *adding $3/5$ times the third row to the fourth row*

giving the final tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
1	0	0	$14/3$	$1/3$	1	$-4/3$	1
0	0	1	-5	0	-1	2	3
0	1	0	$-13/3$	$1/3$	-1	$5/3$	1
0	0	0	0	1	1	1	0

Now we have removed the auxiliary variables x_5, x_6 and x_7 from the basis and have a basic feasible solution of

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \\ x_3 &= 3 \end{aligned}$$

which we can use as a starting initial solution for our original problem. For this we set up the new tableau

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{b}$
1	0	0	*14/3*	1
0	0	1	-5	3
0	1	0	$-13/3$	1
-3	1	3	-1	0

which we got from the previous tableau by removing the columns for x_5, x_6 and x_7 and filling in the last row according to the original objective function. Again, we first have complete the cleanout of the columns corresponding to our basic variables, which we achieve by:

- *adding three times the first row to the fourth row*
- *subtracting three times the second row from the fourth row*
- *subtracting the third row from the fourth row*

giving the new tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{b}$
1	0	0	*14/3*	1
0	0	1	-5	3
0	1	0	$-13/3$	1
0	0	0	$97/3$	-7

which has no strictly negative relative cost values anymore, and hence we have already found the optimal solution

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \\ x_3 &= 3 \\ x_4 &= 0 \end{aligned}$$

Problem 7.3.3 (Applying the simplex algorithm)**.**
Solve the following problem manually:

$$\begin{aligned} \text{minimize} \quad & 2x_1 + 4x_2 + x_3 + x_4 \\ \text{subject to} \quad & x_1 + 3x_2 + x_4 \geq 4 \\ & 2x_1 + x_2 \geq 3 \\ & x_2 + 4x_3 + x_4 \leq 3 \\ & x_1 \geq 0 \\ & \dots \\ & x_4 \geq 0 \end{aligned}$$

Show your workings. In particular, carefully write out the problem which you get after transformation to standard form, and the auxiliary problem for finding an initial basic feasible solution. Show all relevant tableaus.

Solution 7.3.3.

In the model answer to this problem I will be slightly less pedestrian than for the previous problems.

This problem has "less-than" and "greater-than" constraints, for which we first need to introduce slack and surplus variables to arrive at the revised problem:

$$\begin{aligned}
&\textit{minimize} && 2x_1 + 4x_2 + x_3 + x_4 && (7.1)\\
&\textit{subject to} && x_1 + 3x_2 + x_4 - x_5 = 4 && (7.2)\\
& && 2x_1 + x_2 - x_6 = 3 && (7.3)\\
& && x_2 + 4x_3 + x_4 + x_7 = 3 && (7.4)\\
& && x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0 && (7.5)\\
& && x_5 \geq 0, x_6 \geq 0, x_7 \geq 0 && (7.6)
\end{aligned}$$

In contrast to Problem 7.3.1, the presence of "larger-than" constraints does not allow us to come up with a first guess of a basic feasible solution, as the "obvious candidate"

$$\begin{aligned}
x_5 &= -4\\
x_6 &= -3\\
x_7 &= 3
\end{aligned}$$

is not feasible. We therefore have no choice than to introduce another three auxiliary variables x_8, x_9 and x_{10} for finding an initial basic feasible solution. We are hence led to the auxiliary problem

$$\begin{aligned}
&\textit{minimize} && x_8 + x_9 + x_{10}\\
&\textit{subject to} && x_1 + 3x_2 + x_4 - x_5 + x_8 = 4\\
& && 2x_1 + x_2 - x_6 + x_9 = 3\\
& && x_2 + 4x_3 + x_4 + x_7 + x_{10} = 3\\
& && x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0\\
& && x_5 \geq 0, x_6 \geq 0, x_7 \geq 0\\
& && x_8 \geq 0, x_9 \geq 0, x_{10} \geq 0
\end{aligned}$$

for which we have the initial tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{a}_8$	$\mathbf{a}_9$	$\mathbf{a}_{10}$	$\mathbf{b}$
1	3	0	1	−1	0	0	1	0	0	4
2	1	0	0	0	−1	0	0	1	0	3
0	1	4	1	0	0	1	0	0	1	3
0	0	0	0	0	0	0	1	1	1	0

Completely cleaning out the columns for our initial basic variables x_8, x_9 and x_{10} gives the new tableau

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{a}_8$	$\mathbf{a}_9$	$\mathbf{a}_{10}$	$\mathbf{b}$
1	*3*	0	1	−1	0	0	1	0	0	4
2	1	0	0	0	−1	0	0	1	0	3
0	1	4	1	0	0	1	0	0	1	3
−3	−5	−4	−2	1	1	−1	0	0	0	−10

Picking the indicated pivot and exchanging variables x_2 and x_8 gives the new tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{a}_8$	$\mathbf{a}_9$	$\mathbf{a}_{10}$	$\mathbf{b}$
$1/3$	1	0	$1/3$	$-1/3$	0	0	$1/3$	0	0	$4/3$
$5/3$	0	0	$-1/3$	$1/3$	-1	0	$-1/3$	1	0	$5/3$
$-1/3$	0	$\boxed{4}$	$2/3$	$1/3$	0	1	$-1/3$	0	1	$5/3$
$-4/3$	0	-4	$-1/3$	$-2/3$	1	-1	$5/3$	0	0	$-10/3$

Picking the indicated pivot (x_2 is the variable with the smallest relative cost value) and swapping this against x_{10} gives

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{a}_8$	$\mathbf{a}_9$	$\mathbf{a}_{10}$	$\mathbf{b}$
$1/3$	1	0	$1/3$	$-1/3$	0	0	$1/3$	0	0	$4/3$
$\boxed{5/3}$	0	0	$-1/3$	$1/3$	-1	0	$-1/3$	1	0	$5/3$
$-1/12$	0	1	$1/6$	$1/12$	0	$1/4$	$-1/12$	0	$1/4$	$5/12$
$-5/3$	0	0	$1/3$	$-1/3$	1	0	$4/3$	0	1	$-5/3$

Picking the indicated pivot (x_1 has the smallest relative cost value) and swapping this against x_9 gives:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{a}_8$	$\mathbf{a}_9$	$\mathbf{a}_{10}$	$\mathbf{b}$
0	1	0	$2/5$	$-2/5$	$1/5$	0	$2/5$	$-1/5$	0	1
1	0	0	$-1/5$	$1/5$	$-3/5$	0	$-1/5$	$3/5$	0	1
0	0	1	$3/20$	$1/10$	$-1/20$	$1/4$	$-1/10$	$1/20$	$1/4$	$1/2$
0	0	0	0	0	0	0	1	1	1	0

Therefore we have found an initial basic feasible solution of the auxiliary problem as

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \\ x_3 &= 1/2 \end{aligned}$$

and we can now go back to solving our revised problem 7.1, for which we get as the initial tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
0	1	0	$2/5$	$-2/5$	$1/5$	0	1
1	0	0	$-1/5$	$1/5$	$-3/5$	0	1
0	0	1	$3/20$	$1/10$	$-1/20$	$1/4$	$1/2$
2	4	1	1	0	0	0	0

Completely cleaning out the first three columns gives:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
0	1	0	$\boxed{2/5}$	$-2/5$	$1/5$	0	1
1	0	0	$-1/5$	$1/5$	$-3/5$	0	1
0	0	1	$3/20$	$1/10$	$-1/20$	$1/4$	$1/2$
0	0	0	$-7/20$	$11/10$	$9/20$	$-1/4$	$-13/2$

Picking the indicated pivot and bringing x_4 into the basis instead of x_2 gives the new tableau:

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
0	$5/2$	0	1	-1	$1/2$	0	$5/2$
1	$1/2$	0	0	0	$-1/2$	0	$3/2$
0	$-3/8$	1	0	$1/4$	$-1/8$	$\boxed{1/4}$	$1/8$
0	$7/8$	0	0	$3/4$	$5/8$	$-1/4$	$-45/8$

Picking the indicated pivot and bringing x_7 into the basis instead of x_3 gives the final tableau

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{a}_7$	$\mathbf{b}$
0	5/2	0	1	−1	1/2	0	5/2
1	1/2	0	0	0	−1/2	0	3/2
0	−3/2	4	0	1	−1/2	1	1/2
0	1/2	1	0	1	1/2	0	−11/2

in which all relative cost values are positive. So our final basic feasible solution of the revised problem 7.1 is given by:

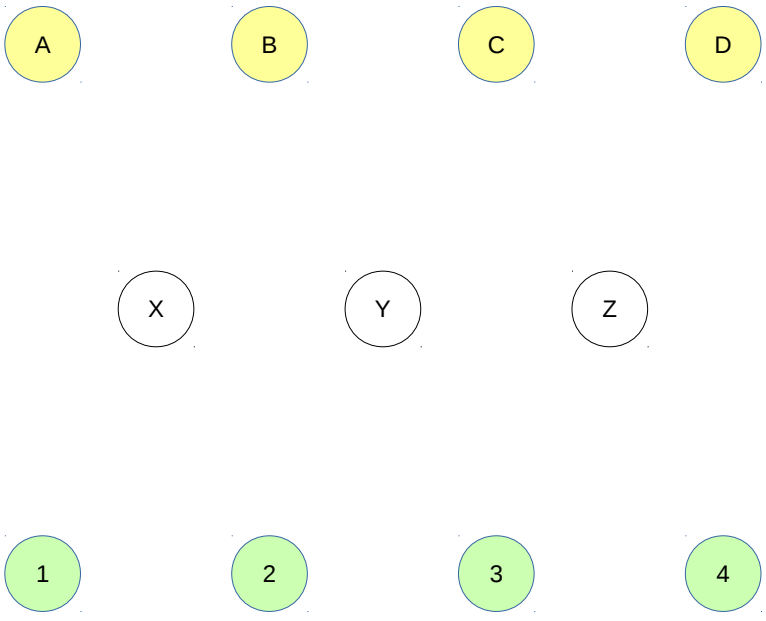
$$\begin{aligned} x_1 &= 3/2 \\ x_4 &= 5/2 \\ x_7 &= 1/2 \end{aligned}$$

and furthermore $x_2 = x_3 = x_5 = x_6 = 0$. This also implies that the optimal solution of our initial problem is given by:

$$\begin{aligned} x_1 &= 3/2 \\ x_2 &= 0 \\ x_3 &= 0 \\ x_4 &= 5/2 \end{aligned}$$

7.4 Further Problems

In the problems of this section we consider the network shown in the following figure:



Here, nodes A, B, C, and D are source nodes (or access nodes) generating traffic towards destination nodes 1, 2, 3 and 4. The demand volumes are given in the following table:

	1	**2**	**3**	**4**
A	40	30	20	10
B	10	60	20	40
C	20	20	20	20
D	70	30	50	10

Between the source and destination nodes some transit routers X, Y and Z are placed.

In all of the following problems it is advisable to write a program / script which **generates** the LP file to be used.

Problem 7.4.1 (Load balancing)**.**
Assume that sources do not have direct links to destinations, but each source has a link to each transit node, and each transit node has a link to each destination. A source then has as many paths available towards a destination as there are transit nodes, and instead of using the δ_{kpl} notation introduced in Section 5.1.2 it might be easier to use decision variables of the form x_{ikj}, referring to the part of the demand volume between source node i and destination node j that is routed through transit node k. All the links have the same maximum capacity of $C = 100$ units.

- Formulate the load-balancing problem for this scenario.
- Solve it numerically with CPLEX.

What is the minimum common link capacity C for which the problem is still feasible?

Problem 7.4.2 (Finding link capacities)**.**
Suppose that each source node $s \in \{A, B, C, D\}$ is connected to all three transit routers, and each transit router is connected to all four destination nodes $t \in \{1, 2, 3, 4\}$.
We want to find the capacities of the links (compare Section 5.2.1).

- Formulate the problem of designing the link capacities between the source nodes, transit nodes and destination nodes, assuming that one unit of flow on any link has a cost of 1, and that a demand volume can be split up into an arbitrary number of flows.
- Solve it numerically with CPLEX.

Part III

Appendices

Appendix A

Vectors and Matrices

In this appendix we briefly recall some of the fundamentals of linear algebra: matrices, linear independence and the rank of matrices. Matrices and operations on them are closely related to solving systems of linear equations (Gaussian elimination, reducing to row echolon form and the like). I assume you already know how to do this and will not repeat that here.

This appendix is meant to serve as a reference and to introduce the notation, not as an introduction. If you lack background in linear algebra, you should consult standard texts like [12].

A.1 Vectors and the $\mathbb{R}^n$

We start with the concept of vectors.

Vectors and $\mathbb{R}^n$: For $n \in \mathbb{N}$ a real n-vector is an n-tuple of real numbers, which we normally write in column form, or as a column vector

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$$

where $x_i \in \mathbb{R}$ for all $i \in \{1, \dots, n\}$. The x_i are called the **components** or **coordinates** of the vector. Another way to write a real n vector is as a row vector:

$$(x_1, x_2, \dots, x_n)$$

Row vectors and column vectors are related through an operation called "transposition", which will be explained in the context of matrices. A row vector is just the transpose of a column vector and vice versa, i.e. we have

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}^T = (x_1, x_2, \dots, x_n)$$

and

$$(x_1, x_2, \ldots, x_n)^T = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}$$

We will usually assume vectors to be column vectors and we will also often use shorthands for vectors, typically lowercase letters typeset in bold, e.g.:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}$$

specifies a column vector $\mathbf{x}$. For a fixed n the set of all possible vectors

$$\left\{ \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix} : x_i \in \mathbb{R}, i \in \{1, \ldots, n\} \right\}$$

is denoted as the space $\mathbb{R}^n$. Note that $\mathbb{R}^1$ is simply the set of real numbers, $\mathbb{R}^2$ is called the plane and $\mathbb{R}^3$ is our "3D space". A vector $\mathbf{x} \in \mathbb{R}^n$ is also said to be a **point** in the $\mathbb{R}^n$ space.

Vector equality, non-negativity : Two vectors $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^n$ are said to be equal if they are equal in all their components simultaneously, i.e. when $u_i = v_i$ for all $i \in \{1, \ldots, n\}$. They are unequal if they differ in at least one coordinate. A vector $\mathbf{v} \in \mathbb{R}^n$ is said to be non-negative when all of its components are non-negative, i.e. $v_i \geq 0$ for all $i \in \{1, \ldots, n\}$. Similarly for (strictly) positive vectors, non-positive vectors, and (strictly) negative vectors.

Vector addition and scalar multiplication : There are two important operations on vectors from the $\mathbb{R}^n$: a vector addition and scalar multiplication. Given two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ we define an addition operation $+ : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^n$ on vectors as follows:

$$(\mathbf{u}, \mathbf{v}) = \left(\begin{pmatrix} u_1 \\ u_2 \\ \ldots \\ u_n \end{pmatrix}, \begin{pmatrix} v_1 \\ v_2 \\ \ldots \\ v_n \end{pmatrix} \right) \mapsto \mathbf{u} + \mathbf{v} = \begin{pmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \ldots \\ u_n + v_n \end{pmatrix}$$

and the operation of scalar multiplication $\cdot : \mathbb{R} \times \mathbb{R}^n \mapsto \mathbb{R}^n$ as follows:

$$(\lambda, \mathbf{u}) = \left(\lambda, \begin{pmatrix} u_1 \\ u_2 \\ \ldots \\ u_n \end{pmatrix} \right) \mapsto \lambda \cdot \mathbf{u} = \begin{pmatrix} \lambda \cdot u_1 \\ \lambda \cdot u_2 \\ \ldots \\ \lambda \cdot u_n \end{pmatrix}$$

The $\mathbb{R}^n$ is a vector space : The space $\mathbb{R}^n$ together with these two operations satisfies the following laws (which make it an instance of an abstract algebraic structure called a vector space):

- $\mathbb{R}^n$ is algebraically closed with respect to addition, i.e. the sum $\mathbf{u} + \mathbf{v}$ of two vectors $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^n$ is again a real vector, i.e. $\mathbf{u} + \mathbf{v} \in \mathbb{R}^n$. $\mathbb{R}^n$ is also algebraically closed with respect to scalar multiplication.

- Addition of vectors is associative and commutative, i.e. for three vectors $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ we have:

$$\begin{aligned} \mathbf{u} + (\mathbf{v} + \mathbf{w}) &= (\mathbf{u} + \mathbf{v}) + \mathbf{w} \\ \mathbf{u} + \mathbf{v} &= \mathbf{v} + \mathbf{u} \end{aligned}$$

  These properties are directly inherited from their counterparts in the real numbers.

- For the zero vector $\mathbf{0}$ and any vector $\mathbf{v} \in \mathbb{R}^n$ we have that

$$\mathbf{0} + \mathbf{v} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \end{pmatrix} + \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix} = \mathbf{v}$$

  The zero vector is also often called the origin.

- For each vector $\mathbf{v} \in \mathbb{R}^n$ there exists a (unique) inverse vector $-\mathbf{v}$ such that $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$. This last equation is more conveniently written as $\mathbf{v} - \mathbf{v} = \mathbf{0}$ and the inverse vector to

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix}$$

  is given by

$$-\mathbf{v} = \begin{pmatrix} -v_1 \\ -v_2 \\ \dots \\ -v_n \end{pmatrix}$$

- Scalar multiplication of a vector $\mathbf{v}$ with the scalar (real) number 0 simply gives the zero vector $\mathbf{0}$, i.e. $0 \cdot \mathbf{v} = \mathbf{0}$. We will leave away the multiplication dot from here on.

- Scalar multiplication fulfills the following consistency requirements with addition: for scalars λ and μ and vectors $\mathbf{u}$ and $\mathbf{v}$ we have that:

$$\begin{aligned} (\lambda + \mu)\mathbf{v} &= \lambda\mathbf{v} + \mu\mathbf{v} \\ \lambda(\mathbf{u} + \mathbf{v}) &= \lambda\mathbf{u} + \lambda\mathbf{v} \\ \lambda(\mu\mathbf{v}) &= (\lambda\mu)\mathbf{v} \\ 1\mathbf{v} &= \mathbf{v} \end{aligned}$$

From these properties it is very simple to further prove the following rules:

- $\lambda\mathbf{0} = \mathbf{0}$
- $\lambda\mathbf{v} = \mathbf{0} \implies \lambda = 0$ or $\mathbf{v} = \mathbf{0}$
- $(-1)\mathbf{v} = -\mathbf{v}$

Inner Product, (Euclidean) Norm : We are given two vectors $\mathbf{u} = (u_1, \dots, u_n)^T$ and $\mathbf{v} = (v_1, \dots, v_n)^T$ from $\mathbb{R}^n$. Then the **inner product** of these two vectors is defined as

$$(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{n} u_i v_i$$

and the (euclidean) norm of a vector $\mathbf{u}$ is defined as

$$\|\mathbf{u}\|_2 = \sqrt{(\mathbf{u},\mathbf{u})} = \sqrt{\sum_{i=1}^{n} u_i^2}$$

Geometrically, the norm of a vector measures the "distance" of the point represented by the vector to the origin (this should be familiar to you at least for the two-dimensional case). The inner product satisfies the following laws in $\mathbb{R}^n$:

- $(\mathbf{u},\mathbf{v}) = (\mathbf{v},\mathbf{u})$
- $(\lambda\mathbf{u},\mathbf{v}) = \lambda(\mathbf{u},\mathbf{v})$ for any scalar $\lambda \in \mathbb{R}$
- $(\mathbf{u},\mathbf{v}+\mathbf{w}) = (\mathbf{u},\mathbf{v}) + (\mathbf{u},\mathbf{w})$

By applying these rules we furthermore get:

$$\|\mathbf{x}+\mathbf{y}\|_2^2 = (\mathbf{x}+\mathbf{y},\mathbf{x}+\mathbf{y}) = (\mathbf{x},\mathbf{x}) + 2(\mathbf{x},\mathbf{y}) + (\mathbf{y},\mathbf{y}) = \|\mathbf{x}\|_2^2 + 2(\mathbf{x},\mathbf{y}) + \|\mathbf{y}\|_2^2$$

The euclidean norm satisfies the following properties for any vectors $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^n$ and any scalar value $\lambda \in \mathbb{R}$:

- $\|\mathbf{u}\|_2 = 0 \quad \Longrightarrow \quad \mathbf{u} = 0$
- $\|\lambda \cdot \mathbf{u}\|_2 = |\lambda| \cdot \|\mathbf{u}\|_2$
- $\|\mathbf{u}+\mathbf{v}\|_2 \leq \|\mathbf{u}\|_2 + \|\mathbf{v}\|_2$

A.2 Linear Independence, Dimension and Basis

In the following we are given some vector space $\mathbb{R}^n$ for a fixed but arbitrary choice of n.

Unit vectors : We introduce the following special n-dimensional vectors:

$$\mathbf{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \dots \\ 0 \\ 0 \end{pmatrix}, \qquad \mathbf{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \dots \\ 0 \\ 0 \end{pmatrix}, \qquad \mathbf{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \dots \\ 0 \\ 0 \end{pmatrix}, \dots, \qquad \mathbf{e}_n = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \dots \\ 0 \\ 1 \end{pmatrix}$$

and the vector $\mathbf{e}_i$ is called the i-th unit vector.

Linear combinations : a linear combination of vectors $\mathbf{u}$ and $\mathbf{v}$ is given by

$$\lambda\mathbf{u} + \mu\mathbf{v}$$

i.e. each vector is multiplied by a scalar and the resulting vector are summed up. The concept of a linear combination can readily be extended to m vectors being combined. Now note that we can write an arbitrary vector $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ as a linear combination of the unit vectors:

$$\mathbf{x} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + \ldots + x_n\mathbf{e}_n$$

Linear independence : We are given some set of m vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m$ from $\mathbb{R}^n$. Intuitively, this set of vectors is said to be linearly independent, when none of these vectors can be written as a linear combination of the others. More formally, the vectors are linearly independent when the relation

$$x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \ldots + x_m\mathbf{a}_m = \mathbf{0}$$

implies that $x_1 = x_2 = \ldots = x_m = 0$ holds. Let us check that this formal definition is equivalent to the given intuition:

- Suppose that
$$x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \ldots + x_m\mathbf{a}_m = \mathbf{0}$$
holds with at least one non-zero x_i – for simplicity let us just assume that $x_1 \neq 0$. Then we can re-arrange and find that:
$$\mathbf{a}_1 = -\left(\frac{x_2}{x_1}\mathbf{a}_2 + \ldots + \frac{x_m}{x_1}\mathbf{a}_m\right),$$
i.e. we can express $\mathbf{a}_1$ as a linear combination of the others, violating linear independence.
- Conversely, assume that $\mathbf{a}_1 \neq \mathbf{0}$ can be expressed as a linear combination of the others, i.e.:
$$\mathbf{a}_1 = x_2\mathbf{a}_2 + \ldots + x_m\mathbf{a}_m$$
with at least one of the x_i being non-zero. Then we clearly have
$$x_2\mathbf{a}_2 + \ldots + x_m\mathbf{a}_m - \mathbf{a}_1 = 0$$
with not all the coefficients of the linear combination being zero.

Basis and Dimension : A basis of the vector space $\mathbb{R}^n$ is a set of linearly independent vectors $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$ such that any vector $\mathbf{x} \in \mathbb{R}^n$ can be written as a (it will show: unique) linear combination of the basis vectors $\mathbf{b}_i$, i.e. we can find coefficients $x_i = x_{i,\mathcal{B}}$ (which will generally depend on the basis $\mathcal{B}$) such that

$$\mathbf{x} = x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + \ldots + x_m\mathbf{b}_m$$

Note that $\mathcal{E} = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ is a basis of $\mathbb{R}^n$. There are in general many different bases, but one can prove that in fact any basis of $\mathbb{R}^n$ will have exactly $m = n$ vectors in it, and this number n is then also called the **dimension** of $\mathbb{R}^n$. This is consistent with everyday's usage of the word "dimension", which loosely refers to the number of degrees of freedom in a space. In three-dimensional space $\mathbb{R}^3$ you have three degrees of freedom and each basis of $\mathbb{R}^3$ has three elements.

Bonus problem A.2.1 (Find another basis of $\mathbb{R}^3$)**.**
The three unit vectors $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$ are a basis of $\mathbb{R}^n$. Find another basis and verify that it indeed is a basis.

A.3 Matrices

Matrix : A matrix $\mathbf{A}$ is a rectangular array of numbers with m rows and n columns:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \ldots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \ldots & a_{2,n-1} & a_{2,n} \\ \ldots & & & & & \\ a_{m,1} & a_{m,2} & a_{m,3} & \ldots & a_{m,n-1} & a_{m,n} \end{pmatrix}$$

where the individual coefficients $a_{i,j}$ (of row i and column j) are real numbers. Such a matrix is called an $m \times n$ matrix and m and n are also called the dimensions of the matrix. We write $\mathbf{A} \in \mathbb{R}^{m \times n}$ to indicate that $\mathbf{A}$ is an $m \times n$ matrix, and when we want to highlight the entries of $\mathbf{A}$ we also write $\mathbf{A} = ((a_{ij})) = ((a_{ij}))_{i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\}}$, preferring the short form $((a_{ij}))$ when the dimensions m and n are understood from the context. We will usually denote matrices by bold capital letters.

Quadratic matrices and identity matrix : A matrix $\mathbf{A}$ is quadratic when it has the same number of rows and columns, i.e. we have $\mathbf{A} \in \mathbb{R}^{n \times n}$. A special quadratic matrix of dimension n is the **identity matrix**, denoted by $\mathbf{I}_n$ or simply by $\mathbf{I}$ when the dimension n is understood from the context. It is given by:

$$\mathbf{I}_n = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ \ldots & & & & & \\ 0 & 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix}$$

i.e. it has ones on the diagonal and zeros everywhere else.

Matrix addition and scalar multiplication : just as for vectors we introduce addition between matrices and scalar multiplication. Suppose that we have two matrices $\mathbf{A} = ((a_{ij}))$ and $\mathbf{B} = ((b_{ij}))$ of the same dimensions. Then the sum $\mathbf{C} = \mathbf{A} + \mathbf{B}$ is defined component-wise, i.e. we have

$$\mathbf{C} = ((c_{ij})) = ((a_{ij} + b_{ij}))$$

and the scalar product between a matrix $\mathbf{A}$ and some real number $\lambda \in \mathbb{R}$ is another matrix given by

$$\lambda \mathbf{A} = ((\lambda a_{ij}))$$

If you look at these definitions you might find that they are similar in spirit to the way we have defined addition and scalar multiplications for vectors. It is in fact perfectly reasonable to think of $\mathbb{R}^{m \times n}$ as a vector space of dimension mn!

Transposition : there is another important operation on matrices, called transposition:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \ldots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \ldots & a_{2,n-1} & a_{2,n} \\ \ldots & & & & & \\ a_{m,1} & a_{m,2} & a_{m,3} & \ldots & a_{m,n-1} & a_{m,n} \end{pmatrix}^T = \begin{pmatrix} a_{1,1} & a_{2,1} & \ldots & a_{m,1} \\ a_{1,2} & a_{2,2} & \ldots & a_{m,2} \\ a_{1,3} & a_{2,3} & \ldots & a_{m,3} \\ \ldots & & & \\ a_{1,n-1} & a_{2,n-1} & \ldots & a_{m,n-1} \\ a_{1,n} & a_{2,n} & \ldots & a_{m,n} \end{pmatrix}$$

i.e. the rows now become columns and an $m \times n$ matrix is changed into an $n \times m$ matrix. When $\mathbf{A}$ and $\mathbf{B}$ are matrices of the same dimension and $\lambda \in \mathbb{R}$ is a scalar, then the following relations hold for transposition:

$$\begin{aligned} (\mathbf{A} + \mathbf{B})^T &= \mathbf{A}^T + \mathbf{B}^T \\ (\lambda \mathbf{A})^T &= \lambda \mathbf{A}^T \end{aligned}$$

Vectors as matrices : we can regard n-dimensional column vectors as an $n \times 1$ matrix and an n-dimensional row vector as an $1 \times n$ matrix – so, **vectors are a special case of matrices**. With this in mind one can see that vector transposition is really just a special case of matrix transposition.

Matrices as vectors : A matrix could also be regarded as a row vector of column vectors:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n-1} & a_{2,n} \\ \dots & & & & & \\ a_{m,1} & a_{m,2} & a_{m,3} & \dots & a_{m,n-1} & a_{m,n} \end{pmatrix} = \left[\begin{pmatrix} a_{1,1} \\ a_{2,1} \\ \dots \\ a_{m,1} \end{pmatrix}, \begin{pmatrix} a_{1,2} \\ a_{2,2} \\ \dots \\ a_{m,2} \end{pmatrix}, \dots, \begin{pmatrix} a_{1,n} \\ a_{2,n} \\ \dots \\ a_{m,n} \end{pmatrix} \right] = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n]$$

where the last two expressions introduce some notation for this viewpoint. By symmetry, one can introduce a matrix as a column vector of row vectors, but this point of view will be less relevant to us.

Rank of a matrix : I will not give the precise definition here, but the **rank** of a matrix $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n]$ is a number indicating the size of the largest subset of column vectors that is linearly independent. When $\mathbf{A} \in \mathbb{R}^{m \times n}$ then the relation

$$\operatorname{rank} \mathbf{A} \leq \min \{m, n\}$$

holds. One furthermore has:

$$\operatorname{rank} \mathbf{A} = \operatorname{rank} \mathbf{A}^T$$

Matrix multiplication : Given two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times k}$ (note here that $\mathbf{A}$ must have as many columns as $\mathbf{B}$ has rows) then we can calculate their matrix product $\mathbf{C} = ((c_{ij})) = \mathbf{A} \cdot \mathbf{B}$ with $\mathbf{C} \in \mathbb{R}^{m \times k}$ by the prescription

$$c_{ij} = \sum_{\nu=1}^{n} a_{i\nu} b_{\nu j}$$

i.e. by calculating the inner product of the i-th row of $\mathbf{A}$ with the j-th column of $\mathbf{B}$. A special case of matrix multiplication is matrix-vector multiplication: be $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{x} \in \mathbb{R}^n = \mathbb{R}^{n \times 1}$ be a column vector, then the result $\mathbf{y} = \mathbf{A} \cdot \mathbf{x}$ is an m-dimensional column vector $\mathbf{y} \in \mathbb{R}^m = \mathbb{R}^{m \times 1}$. An important rule clarifying the relationship between matrix multiplication and transposition is

$$(\mathbf{A} \cdot \mathbf{B})^T = \mathbf{B}^T \cdot \mathbf{A}^T$$

Note that if $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a quadratic matrix, then we have that:

$$\mathbf{A} \cdot \mathbf{I}_n = \mathbf{I}_n \cdot \mathbf{A} = \mathbf{A}$$

Note furthermore that given two column vectors $\mathbf{u}$ and $\mathbf{v}$, we can also re-interpret the inner product of these two vectors as

$$(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T \cdot \mathbf{v}$$

Matrices as linear maps on vector spaces : We consider the n-dimensional vector space $\mathbb{R}^n$ and some quadratic $n \times n$ matrix $\mathbf{A}$. Then the new vector $\mathbf{y} = \mathbf{A}\mathbf{x}$ is again a vector from $\mathbb{R}^n$ and is the image of a map that takes an input vector $\mathbf{x}$ and performs matrix multiplication $\mathbf{A}\mathbf{x}$. This map (and in fact any such map defined by multiplication with some $n \times n$ matrix $\mathbf{A}$) has the following important property: When $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^n$ are two vectors from $\mathbb{R}^n$ and $\lambda \in \mathbb{R}$ is a real number, then we have:

$$\begin{aligned} \mathbf{A} \cdot (\mathbf{u} + \mathbf{v}) &= \mathbf{A} \cdot \mathbf{u} + \mathbf{A} \cdot \mathbf{v} \\ \mathbf{a} \cdot (\lambda \mathbf{u}) &= \lambda (\mathbf{A} \cdot \mathbf{u}) \end{aligned}$$

This in fact says that $\mathbf{A}$ is a **linear map** from $\mathbb{R}^n$ to $\mathbb{R}^n$. So, each matrix $\mathbf{A}$ induces a linear map on $\mathbb{R}^n$. It is a remarkable fact that the converse is true as well: any linear map L on $\mathbb{R}^n$ can in fact be described by a matrix! In particular, the identity map $L(\mathbf{x}) = \mathbf{x}$ on $\mathbb{R}^n$ can be represented by the identity matrix $\mathbf{I}_n$.

Matrix inverse : Suppose we are given some quadratic matrix $\mathbf{A} \in \mathbb{R}^n$. If we can find another matrix $\mathbf{B} \in \mathbb{R}^n$ such that

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{I}_n$$

holds, then the matrix $\mathbf{A}$ is called **non-singular**, the matrix $\mathbf{B}$ is called the **(matrix-)inverse** of $\mathbf{A}$ and is denoted by $\mathbf{A}^{-1}$. Not all quadratic matrices are non-singular (find an example 2×2 matrix!), but if $\mathbf{A}$ is non-singular, then we have

$$\mathbf{I}_n = \mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{A}^{-1} \cdot \mathbf{A}$$

and the inverse $\mathbf{A}^{-1}$ for matrix $\mathbf{A}$ is actually uniquely determined. We furthermore have

$$(\mathbf{A}^{-1})^T = (\mathbf{A}^T)^{-1}$$

An inverse for an $n \times n$ matrix $\mathbf{A}$ is guaranteed to exist when the matrix has full rank, i.e. when $\operatorname{rank} \mathbf{A} = n$ holds.

Matrix representation for solving a linear system of equations : Suppose we are given a linear system of equations:

$$\begin{array}{ccccccccc} a_{1,1}x_1 & + & a_{1,2}x_2 & + & \ldots & + & a_{1,n}x_n & = & b_1 \\ a_{2,1}x_1 & + & a_{2,2}x_2 & + & \ldots & + & a_{2,n}x_n & = & b_2 \\ \ldots & & & & & & & & \\ a_{m,1}x_1 & + & a_{m,2}x_2 & + & \ldots & + & a_{m,n}x_n & = & b_m \end{array}$$

then from inspection it is clear that we can represent this as a vector-matrix equation as follows:

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\mathbf{b} = (b_1, \ldots, b_m)^T \in \mathbb{R}^m$ and $\mathbf{A} = ((a_{ij})) \in \mathbb{R}^{m \times n}$ are given, and the vector $\mathbf{x} = (x_1, \ldots, x_n)^T$ is the vector of unknowns. We solve this in general by using the Gaussian elimination procedure. In the particular case that $\mathbf{A}$ is a quadratic $m \times m$ matrix and has a matrix inverse $\mathbf{A}^{-1}$, we can find the solution $\mathbf{x}$ immediately by multiplying this equation on both sides from the left with $\mathbf{A}^{-1}$ to find:

$$\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b}$$

However, it is in practice often simpler to just use Gaussian elimination than trying to find the inverse of a matrix.

Bibliography

[1] Dimitri P. Bertsekas. *Linear Network Optimization – Networks and Codes*. MIT Press, Cambridge, Massachusetts, 2nd edition, 1991.

[2] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, Massachusetts, 2nd edition, 1999.

[3] Dimitri P. Bertsekas, Angelia Nedic, and Asuman E. Ozdaglar. *Convex analysis and optimization*. Athena Scientific, Belmont, Massachusetts, 2003.

[4] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains – Modeling and Performance Evaluation with Computer Science Applications*. John Wiley and Sons, New York, 1998.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[6] Edwin K. P. Chong and Stanislaw H. Zak. *An Introduction to Optimization*. John Wiley & Sons, New York, second edition, 2001.

[7] Luc De Ghein. *MPLS Fundamentals – A Comprehensive Introduction to MPLS Theory and Practice*. Cisco Press, Indianapolis, In, 2007.

[8] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, 4(4):373–395, 1984.

[9] Leonard Kleinrock. *Queueing Systems – Volume 1: Theory*, volume 1. John Wiley and Sons, New York, 1975.

[10] Leonard Kleinrock. *Queueing Systems – Volume 2: Computer Applications*, volume 2. John Wiley and Sons, New York, 1976.

[11] Bernhard Korte and Jens Vygen. *Combinatorial Optimization – Theory and Algorithms*. Springer, Berlin, third edition, 2005.

[12] David C. Lay, Steven R. Lay, and Judi J. McDonald. *Linear Algebra and Its Applications*. Pearson Education, fifth edition, 2015.

[13] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer, New York, 2010.

[14] Deepankar Medhi and Karthikeyan Ramasamy. *Network Routing – Algorithms, Protocols, and Architectures*. Morgan Kaufmann, San Francicso, California, 2007.

[15] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization – Algorithms and Complexity*. Dover Publications, Mineola, New York, 1998.

[16] Michal Pioro and Deepankar Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Elsevier / Morgan Kaufmann, Amsterdam, 2004.

[17] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, UK, 1998.

[18] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, second edition, 2001.