Johannes Kepler University Linz (JKU)
Research Institute for Symbolic Computation (RISC)

Lecture Notes

Formal Modelling

Wolfgang Windsteiger

Summer Semester 2020

Contents

Introduction

These lecture notes cover examples of formal modelling in three areas:

1. geometry,
2. graph theory, and
3. combinatorial optimization.

By *formal modelling* we mean in this context the *exact formal specification of problems*, where the domains over which the problems are formulated are *discrete* or *symbolic*, in any case non-continuous.

In the area of *geometry*, we will describe a method, how *geometrical prove problems* can be formulated in such a way that they can be transformed into *systems of algebraic equations*, and the solvability of these systems then corresponds to the truth of the original statement. In this case, the domain is *symbolic* because we work on logical formulas expressing geometrical configurations.

In the area of *graph theory*, we introduce the basic concepts of graph theory and formulate one famous problem in graph theory, namely the *shortest path problem*. We present *Dijkstra's algorithm* for computing shortest paths in graphs with non-negative weights and *prove* why it always finds a shortest path. In this case, the domain is *discrete* because we work with graphs with finitely many vertices, edges, and paths.

In the area of *combinatorial optimization* we discuss the *bin packing problem* and outline different flavours of the problem: the *plain version of the problem*, i.e. finding a number of bins into which the objects can be packed, the *decision problem* derived from it, i.e. deciding whether the objects can be packed into a given number of bins, and the *optimization problem* derived from it, finding the minimal number of bins into which the objects can be packed. We compare an *exact solultion algorithm* based in *integer linear programming* with a *heuristic approximation algorithm*. In this case, the domain is *discrete* because we work with finitely objects being packed into finitely many bins.

Bibliography

[1] B. Korte, J. Vygen. *Kombinatorische Optimierung, Theorie und Algorithmen*. Springer Spektrum, 3. Auflage, ISBN 978-3-662-57690-8, `https://doi.org/10.1007/978`, 2018.

[2] A. Schrijver. *Combinatorial Optimization, Polyhedra and Efficiency*. Volume A, Chapters 1–38, Springer Berlin Heidelberg New York, ISBN 978-3-540-44389-6, 2003.

Chapter 1

Modelling in Geometry

In this chapter we present an example of an algebraic modeling of logical statements derived from geometrical problems. More precisely, we show how geometrical problems can be translated to systems of polynomial equations, and how the truth or falsity of the original statement corresponds to solvability of the resulting system of equations. Finally, we present an algorithm that can decide the solvability of systems of algebraic equations.

1.1 An Introductory Example

Consider the very simple geometrical configuration illustrated in Figure 1.1:

- Given two points A and C and the line passing through A and C.
- Given a point B such that the line *AB* is perpendicular to the line *AC*.
- Given a point D such that the line *CD* is perpendicular to the line *AC*.

Then

- the lines *AB* and *CD* must be parallel.

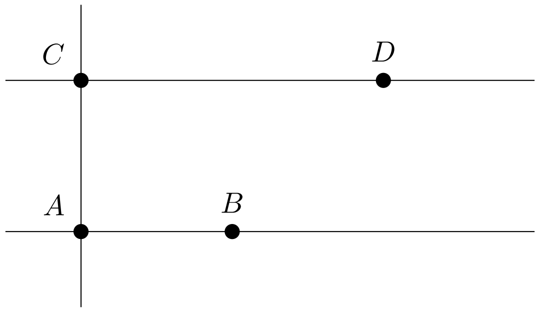

Figure 1.1: Parallel lines

The logical statement describing the geometric situation is

$$\forall_{A,B,C,D} ((\text{perpendicular}(A,B,A,C) \wedge \text{perpendicular}(C,A,C,D)) \Rightarrow \text{parallel}(A,B,C,D)) \tag{1.1}$$

with appropriate predicates ‘perpendicular’ and ‘parallel’.

1.2 Modelling Geometry in Algebra

The first step in modelling geometry is to introduce a coordinate system. Using coordinates we will then be able to describe properties such as 'perpendicular' and 'parallel' by polynomial equalities and negated equalities. Continuing the example from above, let

$$A = (0,0) \qquad B = (b_1, b_2) \qquad C = (c_1, c_2) \qquad D = (d_1, d_2).$$

1.2.1 First Approach

Using the coordinates as described above,

$$\text{perpendicular}(A,B,A,C) \quad \text{means} \quad \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \cdot \begin{pmatrix} c1 \\ c_2 \end{pmatrix} = b_1c_1 + b_2c_2 = 0 \tag{1.2}$$

$$\text{perpendicular}(C,A,C,D) \quad \text{means} \quad \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \cdot \begin{pmatrix} d_1 - c_1 \\ d_2 - c_2 \end{pmatrix} = c_1(d_1 - c_1) + c_2(d_2 - c_2) = 0 \tag{1.3}$$

$$\text{parallel}(A,B,C,D) \quad \text{means} \quad \begin{pmatrix} b_2 \\ -b_1 \end{pmatrix} \cdot \begin{pmatrix} d_1 - c_1 \\ d_2 - c_2 \end{pmatrix} = b_2(d_1 - c_1) - b_1(d_2 - c_2) = 0 \tag{1.4}$$

For short we write the equalities in (1.2), (1.3), and (1.4) as $p_1 = 0$, $p_2 = 0$, and $p_3 = 0$, respectively. Essentially, formula (1.1) is now

$$\underset{b_1,b_2,c_1,c_2,d_1,d_2}{\forall} (p_1 = 0 \wedge p_2 = 0 \Rightarrow p_3 = 0),$$

which is by de'Morgan's rule equivalent to

$$\neg \underset{b_1,b_2,c_1,c_2,d_1,d_2}{\exists} p_1 = 0 \wedge p_2 = 0 \wedge p_3 \neq 0. \tag{1.5}$$

Now the trick: the inequality $p_3 \neq 0$ is equivalent to $\underset{\alpha_0}{\exists}\, \alpha_0 p_3 - 1 = 0$, hence, (1.5) is equivalent to

$$\neg \underset{b_1,b_2,c_1,c_2,d_1,d_2}{\exists} p_1 = 0 \wedge p_2 = 0 \wedge \underset{\alpha_0}{\exists}\, \alpha_0 p_3 - 1 = 0, \tag{1.6}$$

and, under the assumption that α_0 is a variable different from all previously used variables, we finally arrive at

$$\neg \underset{b_1,b_2,c_1,c_2,d_1,d_2,\alpha_0}{\exists} p_1 = 0 \wedge p_2 = 0 \wedge \alpha_0 p_3 - 1 = 0. \tag{1.7}$$

It is easy to see, that (1.7) just expresses that there is *no solution* for the system of equations

$$p_1 = 0 \qquad p_2 = 0 \qquad \alpha_0 p_3 - 1 = 0, \tag{1.8}$$

substituting back the original expressions for p_1, p_2, and p_3 we have a *system of polynomial (algebraic) equations*

$$\begin{aligned} b_1c_1 + b_2c_2 &= 0 \\ c_1d_1 - c_1^2 + c_2d_2 - c_2^2 &= 0 \\ \alpha_0 b_2 d_1 - \alpha_0 b_2 c_1 - \alpha_0 b_1 d_2 + \alpha_0 b_1 c_2 - 1 &= 0. \end{aligned}$$

Solving this system (e.g. with Mathematica) gives us solutions, e.g.

$$c_1 = 0 \qquad c_2 = 0 \qquad b_1 = 0 \qquad b_2 = 1 \qquad d_1 = 1 \qquad d_2 = 0 \qquad \alpha_0 = 1, \tag{1.9}$$

which means that this does not constitute a proof of the original statement (1.1). In fact, the statement is *not true*, because the solution of the system of equations gives a counterexample. If we take $A = C$, $B = (0, 1)$, and $D = (1, 0)$, then the 'line passing through A and C' degenerates to a point such that the hypotheses 'perpendicular(A, B, A, C)' and 'perpendicular(C, A, C, D)' trivially become true whereas the conclusion 'parallel(A, B, C, D)' is false because *AB* and *CD* are perpendicular (and not parallel).

1.2.2 Improved Approach

Obviously, something must have gone wrong in the previous section, because the statement under investigation *is true*. The system of equations derived in the previous section, whose solvability should be equivalent to the statement that we want to prove, however, was an inaccurate model, because it allowed the 'wrong solution' (1.9). Note, however, that clearly C should be different from A when we talk about 'the line passing through A and C', but our model did not contain any hypothesis expressing $C \neq A$. Strictly speaking, (1.9) is of course a correct solution of the system of equations (check by substitution!), but the equations are a *wrong model* for the original proof problem.

Using coordinates $C \neq A$ means $c_1 \neq 0 \vee c_2 \neq 0$. Applying the trick like in (1.6) again, this is equivalent to

$$\underset{\alpha_1}{\exists}\, \alpha_1 c_1 - 1 = 0 \vee \underset{\alpha_2}{\exists}\, \alpha_2 c_2 - 1 = 0$$

which is equivalent to

$$\underset{\alpha_1, \alpha_2}{\exists}\, q_1 = 0 \quad \text{with} \quad q_1 = (\alpha_1 c_1 - 1)(\alpha_2 c_2 - 1).$$

In other words, (1.1) is equivalent to the unsolvability of

$$\begin{aligned} b_1 c_1 + b_2 c_2 &= 0 \\ c_1 d_1 - c_1^2 + c_2 d_2 - c_2^2 &= 0 \\ (\alpha_1 c_1 - 1)(\alpha_2 c_2 - 1) &= 0 \\ \alpha_0 b_2 d_1 - \alpha_0 b_2 c_1 - \alpha_0 b_1 d_2 + \alpha_0 b_1 c_2 - 1 &= 0 \end{aligned}$$

If we pass this system of equations to Mathematica, it will in fact tell us that there is no solution, so the original statement (1.1) is proved.

1.2.3 The General Model

Let us assume we have a geometrical configuration described by

$$p_1 = 0 \qquad \dots \qquad p_n = 0 \qquad q_1 \neq 0 \qquad \dots \qquad q_m \neq 0$$

and a conclusion described by

$$c = 0,$$

where $p_i, q_j, c \in \mathbb{Q}[x_1, \dots, x_l]$ for $1 \le i \le n$ and $1 \le j \le m$. Then

$$\underset{x_1,\dots,x_l}{\forall} (p_1 = 0 \wedge \ldots \wedge p_n = 0 \wedge q_1 \neq 0 \wedge \ldots \wedge q_m \neq 0 \Rightarrow c = 0)$$

is equivalent to

$$\neg \underset{x_1,\dots,x_l}{\exists} \neg(p_1 = 0 \wedge \ldots \wedge p_n = 0 \wedge q_1 \neq 0 \wedge \ldots \wedge q_m \neq 0 \Rightarrow c = 0),$$

which is in turn equivalent to

$$\neg \underset{x_1,\dots,x_l}{\exists} p_1 = 0 \wedge \ldots \wedge p_n = 0 \wedge q_1 \neq 0 \wedge \ldots \wedge q_m \neq 0 \wedge c \neq 0.$$

The negated equalities can then be turned into equalities by the so-called *Rabinovich-Trick*

$$\neg \underset{x_1,\dots,x_l}{\exists} p_1 = 0 \wedge \ldots \wedge p_n = 0 \wedge \underset{\alpha_1}{\exists} \alpha_1 q_1 - 1 = 0 \wedge \ldots \wedge \underset{\alpha_m}{\exists} \alpha_m q_m - 1 = 0 \wedge \underset{\alpha_0}{\exists} \alpha_0 c - 1 = 0,$$

and since we assume that $\alpha_0, \alpha_1, \dots, \alpha_m$ are new variables distinct from $x_1, \dots, x_n$ this is equivalent to

$$\neg \underset{x_1,\dots,x_l,\alpha_0,\alpha_1,\dots,\alpha_m}{\exists} p_1 = 0 \wedge \ldots \wedge p_n = 0 \wedge \alpha_1 q_1 - 1 = 0 \wedge \ldots \wedge \alpha_m q_m - 1 = 0 \wedge \alpha_0 c - 1 = 0.$$

This is nothing else than saying that the system of polynomial equations in the variables $x_1, \dots, x_l, \alpha_0, \alpha_1, \dots, \alpha_m$

$$\begin{aligned} p_1 &= 0 \\ &\vdots \\ p_n &= 0 \\ \alpha_1 q_1 - 1 &= 0 \\ &\vdots \\ \alpha_m q_m - 1 &= 0 \\ \alpha_0 c - 1 &= 0 \end{aligned}$$

has *no solutions* for $x_1, \dots, x_l, \alpha_0, \alpha_1, \dots, \alpha_m$.

EXAMPLE 1.1: THEOREM OF THALES

Let A and B be two points and M the midpoint between A and B. Let c be the circle with center M through A and B, and let C be any point on c. Then *AC* and *BC* are perpendicular.

We introduce coordinates and fix $M = (0, 0)$. We have $A = (a_1, a_2)$, $B = (b_1, b_2)$, and $C = (c_1, c_2)$. Since M is the midpoint between A and B, we have

$$a_1 + b_1 = 0 \qquad a_2 + b_2 = 0,$$

and since C and A are on a circle, their distance to the center M must be equal, i.e.

$$a_1^2 + a_2^2 - c_1^2 - c_2^2 = 0.$$

AC and *BC* are perpendicular can be formulated as

$$(c_1 - a_1)(c_1 - b_1) + (c_2 - a_2)(c_2 - b_2) = 0.$$

Using the model from above with $n = 3$, $m = 0$, and $l = 6$ we have the following system of equations in the variables $a_1, a_2, b_1, b_2, c_1, c_2$:

$$\begin{aligned} a_1^2 + a_2^2 - c_1^2 - c_2^2 &= 0 \\ a_1 + b_1 &= 0 \\ a_2 + b_2 &= 0 \\ \alpha_0((c_1 - a_1)(c_1 - b_1) + (c_2 - a_2)(c_2 - b_2)) - 1 &= 0 \end{aligned}$$

In this example it is easy to convince oneself that this system has *no solution*, because equations 2 and 3 mean

$$a_1 = -b_1 \qquad\qquad a_2 = -b_2,$$

and substituting in equation 4 yields

$$\alpha_0(c_1^2 - a_1^2 + c_2^2 - a_2^2) - 1 = 0. \tag{1.10}$$

From equation 1 we get $a_1^2 + a_2^2 = c_1^2 + c_2^2$, which turns (1.10) into

$$\alpha_0 \cdot 0 - 1 = 0, \quad \text{i.e.} \quad -1 = 0,$$

hence, the above system of polynomial equations has no solutions.

1.2.4 Deciding Solvability of a System of Polynomial Equations

In general, it is not as easy as in Example 1.1 to decide, whether a system of polynomial equations has a solution or not. The theory of Gröbner bases plays a key role in this field, but we will not go into much detail.

Given a set of polynomials G in $x_1, \ldots, x_n$, a *Gröbner basis of* G is a set of polynomials B, such that

$$\underset{x_1,\ldots,x_n}{\forall} \left(\underset{g \in G}{\forall} g(x_1, \ldots, x_n) = 0 \Leftrightarrow \underset{b \in B}{\forall} b(x_1, \ldots, x_n) = 0 \right),$$

i.e. the zero set of G equals the zero set of B, and B has some special properties that make the system $\underset{b \in B}{\forall} b = 0$ 'easier to solve' than the original system $\underset{g \in G}{\forall} g = 0$. In many respects, a Gröbner basis of G is the polynomial analogy to the triangular form of a matrix representing a system of linear equations. Fortunately[1], there is an algorithm that computes a Gröbner basis for any given set of polynomials G. Similar to Gaussian elimination, the Gröbner basis algorithm subsequently eliminates variables by a process called *polynomial reduction*, which is a generalization of the *univariate polynomial division* to multivariate polynomials. Every

[1]The *concept of Gröbner bases* and, most importantly, the *first algorithm to compute a Gröbner basis* for arbitrary G were invented by *Bruno Buchberger*, the founder of RISC, the Research Institute for Symbolic Computation at JKU Linz.

computer algebra system (like Mathematica, Maple, or Sage) offers a command to compute Gröbner bases, in Mathematica this command is called `GroebnerBasis`.

THEOREM 1.2

A system of polynomial equations

$$g_1 = 0, \quad \ldots \quad , g_n = 0$$

has no solutions over $\mathbb{C}$ *if and only if the Gröbner basis of* $\{g_1, \ldots, g_n\}$ *contains a constant polynomial unequal to* 0*.*

EXAMPLE 1.3: THALES WITH GRÖBNER BASIS

Using Mathematica, we compute

$$\texttt{GroebnerBasis}[\{a_1^2 + a_2^2 - c_1^2 - c_2^2, a_1 + b_1, a_2 + b_2, \alpha_0((c_1 - a_1)(c_1 - b_1) + (c_2 - a_2)(c_2 - b_2)) - 1\}, \{a_1, a_2, b_1, b_2, c_1, c_2, \alpha_0\}]$$

and the answer is $\{1\}$, thus, the Gröbner basis contains the constant polynomial 1 and the system of equations corresponding to the Theorem of Thales is unsolvable, therefore the theorem is proven.

1.2.5 Describing Frequently Used Geometrical Properties by Polynomials

In this section we assume some coordinate system. We will collect some polynomial equations describing frequently used properties in geometry.

THEOREM 1.4

Let

$$X_1 = \binom{x_1}{y_1} \qquad X_2 = \binom{x_2}{y_2} \qquad X_3 = \binom{x_3}{y_3} \qquad X_4 = \binom{x_4}{y_4}.$$

1. X_1*,* X_2*, and* X_3 *are collinear if and only if*

$$\det\begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix} = 0.$$

2. X_1*,* X_2*,* X_3*, and* X_4 *are collinear if and only if*

$$\det\begin{pmatrix} 1 & x_1 & y_1 & {x_1}^2 + {y_1}^2 \\ 1 & x_2 & y_2 & {x_2}^2 + {y_2}^2 \\ 1 & x_3 & y_3 & {x_3}^2 + {y_3}^2 \\ 1 & x_4 & y_4 & {x_4}^2 + {y_4}^2 \end{pmatrix} = 0.$$

3. X_1X_2 *and* X_3X_4 *are perpendicular if and only if*

$$(x_2 - x_1)(x_4 - x_3) + (y_2 - y_1)(y_4 - y_3) = 0.$$

4. *X_1X_2 and X_3X_4 are parallel if and only if*

$$(y_2 - y_1)(x_4 - x_3) - (x_2 - x_1)(y_4 - y_3) = 0.$$

Example 1.5: Theorem of Pappus

Given one set of collinear points R, S, and T, and another set of collinear points U, V, and W, s.t. R, S, and U and R, S, and V, respectively, are not collinear. Then the intersection points X, Y, and Z of line pairs *RV* and *SU*, *RW* and *TU*, *SW* and *TV* are collinear, see Figure 1.2. We introduce coordinates:

$$\begin{array}{ccc} R = \binom{r_1}{r_2} & S = \binom{s_1}{s_2} & T = \binom{t_1}{t_2} \\ U = \binom{u_1}{u_2} & V = \binom{v_1}{v_2} & W = \binom{w_1}{w_2} \\ X = \binom{x_1}{x_2} & Y = \binom{y_1}{y_2} & Z = \binom{z_1}{z_2}. \end{array}$$

That point X is the intersection of *RV* and *SU* means that both R, V, and X as well as S, U, and X are collinear, and by Theorem 1.4 this can be described by

$$\det(\begin{pmatrix} 1 & r_1 & r_2 \\ 1 & v_1 & v_2 \\ 1 & x_1 & x_2 \end{pmatrix}) = -r_2v_1 + r_1v_2 + r_2x_1 - r_1x_2 - v_2x_1 + v_1x_2 = 0$$

$$\det(\begin{pmatrix} 1 & s_1 & s_2 \\ 1 & u_1 & u_2 \\ 1 & x_1 & x_2 \end{pmatrix}) = -s_2u_1 + s_1u_2 + s_2x_1 - s_1x_2 + u_1x_2 - u_2x_1 = 0$$

Applying the same technique for the remaining hypotheses and the conclusion of the theorem we arrive at a model with $n = 8$, $m = 2$, and $l = 18$

$$\begin{aligned}
-r_2v_1 + r_1v_2 + r_2x_1 - r_1x_2 + v_1x_2 - v_2x_1 &= 0 \\
-s_2u_1 + s_1u_2 + s_2x_1 - s_1x_2 + u_1x_2 - u_2x_1 &= 0 \\
-r_2w_1 + r_1w_2 + r_2y_1 - r_1y_2 + w_1y_2 - w_2y_1 &= 0 \\
-t_2u_1 + t_1u_2 + t_2y_1 - t_1y_2 + u_1y_2 - u_2y_1 &= 0 \\
-s_2w_1 + s_1w_2 + s_2z_1 - s_1z_2 + w_1z_2 - w_2z_1 &= 0 \\
-t_2v_1 + t_1v_2 + t_2z_1 - t_1z_2 + v_1z_2 - v_2z_1 &= 0 \\
-r_2s_1 + r_1s_2 + r_2t_1 - r_1t_2 + s_1t_2 - s_2t_1 &= 0 \\
-u_2v_1 + u_1v_2 + u_2w_1 - u_1w_2 + v_1w_2 - v_2w_1 &= 0 \\
\alpha_1\left(-r_2s_1 + r_1s_2 + r_2u_1 - r_1u_2 + s_1u_2 - s_2u_1\right) - 1 &= 0 \\
\alpha_2\left(-r_2s_1 + r_1s_2 + r_2v_1 - r_1v_2 + s_1v_2 - s_2v_1\right) - 1 &= 0 \\
\alpha_0\left(-x_2y_1 + x_1y_2 + x_2z_1 - x_1z_2 + y_1z_2 - y_2z_1\right) - 1 &= 0
\end{aligned}$$

The Gröbner basis of the set of left-hand sides of these equations is in fact $\{1\}$, hence, the Theorem of Pappus is proved.

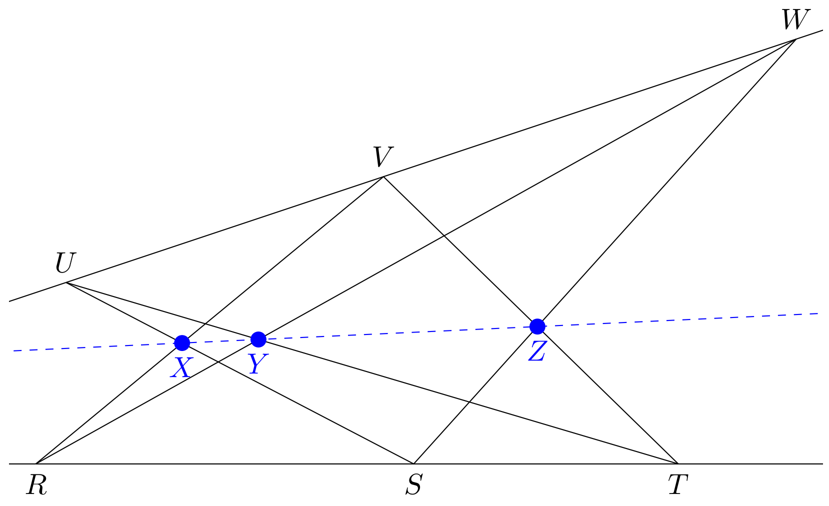

Figure 1.2: Theorem of Pappus

1.3 A Generalization Beyond Geometry

Statements derived from geometrical theorems usually have the form of universally quantified implications. We will now see that not only these can be translated into systems of polynomial equations. The same technique as shown in the previous section can be applied to *arbitrary universally quantified boolean combinations* of polynomial equalities[2]. We want to prove

$$\underset{x_1,\ldots,x_l}{\forall} \Phi, \tag{1.11}$$

where Φ is a boolean combination of polynomial equations with polynomials in $\mathbb{Q}[x_1, \ldots, x_l]$. First we rewrite the statement as

$$\neg \underset{x_1,\ldots,x_l}{\exists} \neg\Phi,$$

and then we convert $\neg\Phi$ into *conjunctive normal form*, thus, (1.11) can be written as

$$\neg \underset{x_1,\ldots,x_l}{\exists} (\Phi_{1,1} \vee \ldots \vee \Phi_{1,j_1}) \wedge \ldots \wedge (\Phi_{n,1} \vee \ldots \vee \Phi_{n,j_n}), \tag{1.12}$$

where each $\Phi_{i,j}$ has the form either $P_{i,j} = 0$ or $\neg(P_{i,j} = 0)$. Now we introduce new polynomials

$$Q_{i,j} := \begin{cases} P_{i,j} & \text{if } \Phi_{i,j} \text{ has the form } P_{i,j} = 0 \\ \alpha_{i,j} P_{i,j} - 1 & \text{if } \Phi_{i,j} \text{ has the form } \neg(P_{i,j} = 0) \end{cases}$$

with new variables $\alpha_{i,j}$. Note that the $\alpha_{i,j}$ can be thought of as existentially quantified in $Q_{i,j} = 0$, compare to the Rabinovich-Trick explained in the previous section. Since the $\alpha_{i,j}$

[2] Note that inequations of the form $p \neq 0$ are covered in this setting as well because $p \neq 0 \equiv \neg(p = 0)$, hence, an inequation is a boolean combination of an equality.

are all new and distinct from $x_1, \ldots, x_l$ the existential quantifiers can be pushed outside such that (1.12) can be written as

$$\neg \underset{x_1,\ldots,x_l,\{\alpha_{i,j}\}}{\exists} (Q_{1,1} = 0 \vee \ldots \vee Q_{1,j_1} = 0) \wedge \ldots \wedge (Q_{n,1} = 0 \vee \ldots \vee Q_{n,j_n} = 0).$$

The $\{\alpha_{i,j}\}$ in the existential quantifier should indicate that we quantify over all $\alpha_{i,j}$ that occur in the $Q_{i,j}$. Now remember that a product is zero if and only if one of the factors is zero[3], in other words

$$Q_{i,1} = 0 \vee \ldots \vee Q_{i,j_i} = 0 \quad \text{if and only if} \quad Q_{i,1} \cdot \ldots \cdot Q_{i,j_i} = 0,$$

thus, finally

$$\neg \underset{x_1,\ldots,x_l,\{\alpha_{i,j}\}}{\exists} (Q_{1,1} \cdot \ldots \cdot Q_{1,j_1} = 0) \wedge \ldots \wedge (Q_{n,1} \cdot \ldots \cdot Q_{n,j_n} = 0).$$

Hence, the original statement (1.11) is equivalent to the unsolvability of the system of polynomial equations

$$\begin{array}{rcl} Q_{1,1} \cdot \ldots \cdot Q_{1,j_1} & = & 0 \\ \vdots & \vdots & \vdots \\ Q_{n,1} \cdot \ldots \cdot Q_{n,j_n} & = & 0, \end{array}$$

which can be decided by computing

$$B = \texttt{GroebnerBasis}[\{Q_{1,1} \cdot \ldots \cdot Q_{1,j_1}, \ldots, Q_{n,1} \cdot \ldots \cdot Q_{n,j_n}\}]$$

and checking, whether B contains a constant polynomial unequal to 0.

EXAMPLE 1.6

We come back to our introductory example (1.1). It is easy to see that this statement can be generalized: if we have two perpendicular lines, then being parallel to one of them is obviously the same as being perpendicular to the other. To make a theorem out of that we need two side-conditions, which guarantee that the given lines will not degenerate to points, in other words,

$$\underset{A,B,C,D}{\forall} A \neq C \wedge A \neq B \wedge \text{perpendicular}(A,B,A,C) \Rightarrow$$
$$\text{perpendicular}(C,A,C,D) \Leftrightarrow \text{parallel}(A,B,C,D)$$

After introducing coordinates

$$A = (0,0) \qquad B = (b_1, b_2) \qquad C = (c_1, c_2) \qquad D = (d_1, d_2)$$

[3]We used the product trick also when we expressed the condition $c_1 \neq 0 \vee c_2 \neq 0$ in Section 1.2.2.

the conjunctive normal form of the negated expression inside the quantifier gives

$$\begin{gathered}(b_1 \neq 0 \vee b_2 \neq 0) \wedge (c_1 \neq 0 \vee c_2 \neq 0) \wedge \\ \wedge (\neg \text{parallel}(A, B, C, D) \vee \neg \text{perpendicular}(C, A, C, D)) \wedge \\ \wedge (\text{parallel}(A, B, C, D) \vee \text{perpendicular}(C, A, C, D)) \wedge \\ \wedge \text{perpendicular}(A, B, A, C)\end{gathered}$$

Using Theorem 1.4 we get the following combination of equations an inequations

$$\begin{gathered}(b_1 \neq 0 \vee b_2 \neq 0) \wedge (c_1 \neq 0 \vee c_2 \neq 0) \wedge \\ \wedge (b_2 (d_1 - c_1) - b_1 (d_2 - c_2) \neq 0 \vee -c_1 (d_1 - c_1) - c_2 (d_2 - c_2) \neq 0) \wedge \\ \wedge (b_2 (d_1 - c_1) - b_1 (d_2 - c_2) = 0 \vee -c_1 (d_1 - c_1) - c_2 (d_2 - c_2) = 0) \wedge \\ \wedge b_1 c_1 + b_2 c_2 = 0.\end{gathered}$$

Applying the Rabinovich-Trick and combining disjunctions to products results in the following set of polynomials

$$\begin{gathered}\{-\alpha_0 b_1 + \alpha_1 \alpha_0 b_1 b_2 - \alpha_1 b_2 + 1, -\alpha_2 c_1 + \alpha_3 \alpha_2 c_1 c_2 - \alpha_3 c_2 + 1, \\ -\alpha_4 \alpha_5 b_2 c_1^3 + \alpha_4 \alpha_5 b_1 c_2 c_1^2 + \alpha_4 b_2 c_1 - \alpha_4 \alpha_5 b_2 c_2^2 c_1 - \alpha_4 b_1 c_2 + \alpha_4 \alpha_5 b_1 c_2^3 + \\ 2\alpha_4 \alpha_5 b_2 c_1^2 d_1 - \alpha_4 \alpha_5 b_1 c_1^2 d_2 - \alpha_4 \alpha_5 b_2 c_1 d_1^2 - \alpha_4 \alpha_5 b_1 c_2 c_1 d_1 + \alpha_4 \alpha_5 b_2 c_2 c_1 d_2 + \\ \alpha_4 \alpha_5 b_1 c_1 d_1 d_2 + \alpha_4 \alpha_5 b_1 c_2 d_2^2 + \alpha_4 \alpha_5 b_2 c_2^2 d_1 - 2\alpha_4 \alpha_5 b_1 c_2^2 d_2 - \alpha_4 \alpha_5 b_2 c_2 d_1 d_2 - \alpha_4 b_2 d_1 + \\ \alpha_4 b_1 d_2 - \alpha_5 c_1^2 - \alpha_5 c_2^2 + \alpha_5 c_1 d_1 + \alpha_5 c_2 d_2 + 1, \\ 2b_2 c_1^2 d_1 - b_1 c_1^2 d_2 - b_2 c_1 d_1^2 - b_1 c_2 c_1 d_1 + b_2 c_2 c_1 d_2 + b_1 c_1 d_1 d_2 + b_1 c_2 d_2^2 + b_2 c_2^2 d_1 - \\ 2b_1 c_2^2 d_2 - b_2 c_2 d_1 d_2 - b_2 c_1^3 + b_1 c_2 c_1^2 - b_2 c_2^2 c_1 + b_1 c_2^3, \\ b_1 c_1 + b_2 c_2\},\end{gathered}$$

whose Gröbner basis is again $\{1\}$, hence, the statement is proved.

Chapter 2

Modelling in Graph Theory

Many practical problems concerning relationships, networks, scheduling, or assignments can be modelled mathematically in the language of graph theory. In this chapter, we present some basic concepts of graph theory and one example problem with an elegant algorithmic solution.

2.1 An Introductory Example

In GoogleMaps as well as in many modern cars you have the possibility to enter two geographical locations A and B and get "the shortest route" from A to B. Usually there are different choices for how to interpret "shortest". Of course, it will in most cases not mean "shortest euclidean distance", because in that case the solution would be trivial, hence uninteresting, namely the straight line through A and B. What makes the problem interesting and non-trivial is the underlying "network of streets", where a "route" from A to B is only allowed to follow these streets. Moreover, "shortest" can then mean "minimal w.r.t. length" or "minimal w.r.t. duration", i.e. the "quickest" route. Last but not least, various criteria can be applied for choosing appropriate streets, e.g. avoiding toll-road, avoiding highways, prefer scenic roads, calculate bicycle routes, etc. As an example see Figure 2.1, which shows the quickest bicycle route from JKU Linz to the RISC Institute in Hagenberg[1].

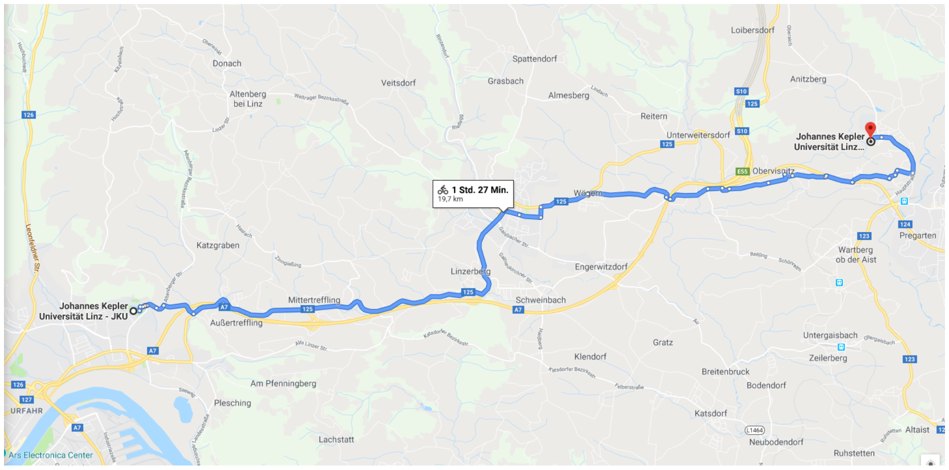

Figure 2.1: Google Maps Route Planner

[1]Note, however, that $1h27$ for not even 20km is a bit exaggerated.

2.2 Graph Theory

Graphs will turn out to be an appropriate mathematical model for a "network of streets" as needed in the routing example in Section 2.1. Informally speaking, in the heart of the concept of a graph is a collection of entities, called *vertices*, with connections between them, called *edges* or *arcs*, see Figure 2.2. There are different types of graphs, depending on whether for instance

- the connections are oriented or not,
- there can be more than one edge between to vertices,
- vertices connected by an edge must be distinct,
- edges have values associated,
- vertices have values associated,
- etc. etc.

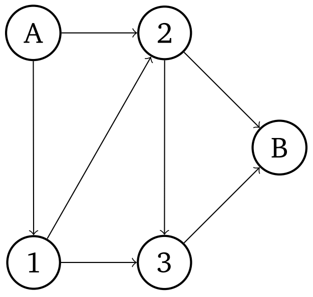

Figure 2.2: A typical graph

Definition 2.1: Undirected Simple Graph, Directed Simple Graph

Let V be a set. The pair $G = (V, E)$ is called an *undirected simple graph* if and only if

$$E \subseteq P(V) \text{ and } \underset{a \in E}{\forall} |a| = 2.$$

The pair (V, E) is called a *directed simple graph* if and only if

$$E \subseteq V^2 \text{ and } \underset{a \in E}{\forall} a_1 \neq a_2.$$

We call G a *simple graph* if and only if it is an undirected or a directed graph. If G is a graph, then $V(G) := G_1$ and $E(G) := G_2$ are called the *vertices* and *edges* of G, respectively.

Note that we use *sets* for representing edges, which implies that there cannot be multiple edges between two vertices. For certain applications, however, it is necessary to allow multiple edges between vertices. In these cases, the edges E can be defined as *multisets* $E = (A, m)$, where

$$A = \{a \in P(V) \mid |a| = 2\} \qquad \text{(for undirected graphs)}$$
$$A = \{a \in V^2 \mid a_1 \neq a_2\} \qquad \text{(for directed graphs)}$$

and

$$m\colon A \to \mathbb{N}_0.$$

The set A is the set of potential edges, and the function m gives the multiplicity of each edge, including the case $m(a) = 0$ meaning that the graph does not contain the edge a. For a multiset $E = (A, m)$ we write $a \in E$ if and only if $a \in A$ and $m(a) \geq 1$.

In the multiset representation, the multiplicity of an edge tells us only, how many connections we have, but these are indistinguishable. If we want to have distinct edges, then we define the edges as $E = (X, e)$, where X is a set and

$$e\colon X \to A \qquad \text{with } A \text{ as above.}$$

For $E = (X, e)$ we write $a \in E$ if and only if $\underset{x \in X}{\exists}\, e(x) = a$.

EXAMPLE 2.2

Figure 2.3 shows different types of graphs. The leftmost is an undirected simple graph

$$G_1 = (\{1,2,3\}, \{\{1,2\},\{2,3\}\}),$$

second from left is a directed simple graph

$$G_2 = (\{1,2,3\}, \{(1,2),(2,1),(1,3),(2,3)\}).$$

Second from right is not an undirected simple graph because it contains a double edge between 1 and 2, which can be defined as

$$G_3 = (\{1,2,3\}, (\{\{1,2\},\{1,3\},\{2,3\}\}, m)) \quad \text{with } m \text{ defined by}$$

a	$m(a)$
$\{1,2\}$	2
$\{1,3\}$	0
$\{2,3\}$	1

Note that one cannot distinguish the two edges between 1 and 2. The rightmost is not a simple directed graph either because it contains two distinct edges a and b from 1 to 2, which can be defined as

$$G_4 = (\{1,2,3\}, (\{a,b,c\}, e)) \quad \text{with } e \text{ defined by}$$

x	$e(x)$
a	$(1,2)$
b	$(1,2)$
c	$(2,3)$

By definition, a simple graph cannot contain *loops*, i.e. edges connecting a vertex with itself. In an undirected graph, this would be an edge $\{u,u\}$, but $|\{u,u\}| = 1$, whereas in a

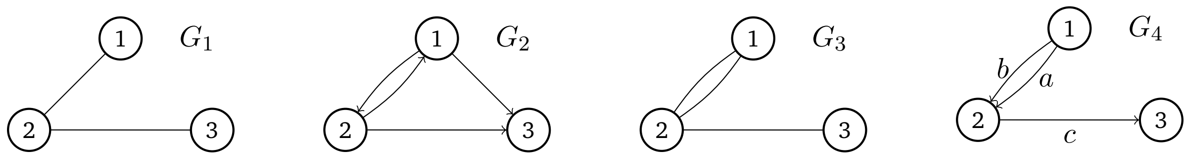

Figure 2.3: Different types of graphs

directed graph it would be an edge $a = (u, u)$, but $a_1 = u = a_2$. If loops should be permitted the restrictions $|a| = 2$ and $a_1 \neq a_2$, respectively, have to be relaxed.

From now on, if we say G is a graph then we mean a graph in one of the representations mentioned above. If we say G is a simple graph then we mean a simple undirected or a simple directed graph. If G is an undirected graph then we use uv as an abbreviation for an edge $\{u, v\} \in E(G)$, if G is directed then uv stands for an edge $(u, v) \in E(G)$.

DEFINITION 2.3: NEIGHBOUR, NEIGHBOURHOOD

Let G be a graph. The vertex v is a *neighbour of* vertex u if and only if $uv \in E(G)$. Furthermore, we call

$$N_G(u) := \{v \in V(G) \mid v \text{ is a neighbour of } u\}$$

the *neighbourhood* of u (in G).

EXAMPLE 2.4

Using the graphs from Figure 2.3,

$N_{G_1}(1) = \{2\}$	$N_{G_1}(2) = \{1, 3\}$	$N_{G_1}(3) = \{2\}$
$N_{G_2}(1) = \{2, 3\}$	$N_{G_2}(2) = \{1, 3\}$	$N_{G_2}(3) = \{\}$
$N_{G_3}(1) = \{2\}$	$N_{G_3}(2) = \{1, 3\}$	$N_{G_3}(3) = \{2\}$
$N_{G_4}(1) = \{2\}$	$N_{G_4}(2) = \{3\}$	$N_{G_4}(3) = \{\}$

DEFINITION 2.5: WALK, PATH

Let G be a graph, $n \geq 1$, and $a, b \in V(G)$. A finite sequence[a] $w\colon \mathbb{N}_{1,2n+1} \to V(G) \cup E(G)$ is called a *walk* of length n from a to b in G if and only if

1. $\underset{0 \leq i \leq n}{\forall} w_{2i+1} \in V(G)$,
2. $\underset{1 \leq i \leq n}{\forall} w_{2i} \in E(G)$,
3. $w_1 = a$ and $w_{2n+1} = b$, and
4. $\underset{1 \leq i \leq n}{\forall} w_{2i} = w_{2i-1} w_{2i+1}$.

Let w be a walk of length n from a to b in G, then the subsequences

$$V(w)\colon \mathbb{N}_{1,n+1} \to V(G), i \mapsto w_{2i-1} \qquad E(w)\colon \mathbb{N}_{1,n} \to E(G), i \mapsto w_{2i}$$

are the *sequence of vertices* of w and the *sequence of edges* of w, respectively.

A finite sequence $t\colon \mathbb{N}_{1,2n+1} \to V(G) \cup E(G)$ is called a *trail* of length n from a to b in G if and only if t is a walk of length n from a to b in G and $E(t)$ is injective (from $\mathbb{N}_{1,n}$ to $E(G)$).

A finite sequence $p\colon \mathbb{N}_{1,2n+1} \to V(G) \cup E(G)$ is called a *path* of length n from a to b in G if and only if p is a trail of length n from a to b in G and $V(p)$ is injective (from $\mathbb{N}_{1,n+1}$ to $V(G)$).

[a]Reminder: $\mathbb{N}_{m,n} := \{i \in \mathbb{N}_0 \mid m \leq i \leq n\}$.

A walk is an arbitrary sequence of vertices connected by appropriate edges. A trail does not use any edge twice and a path in addition does not visit any vertex twice. A walk that is not a trail must use at least one edge more than once, which implies that it also visits some vertices more than once, namely those vertices that are connected by that edge.

The length n of a walk/trail/path addresses the number of edges in that walk/trail/path. If we omit the length or the start and end then we assume the respective entities existentially quantified, e.g. if p is a path from a to b in G then we mean a path *of some length* from a to b in G, or if p is a path of length n in G then we mean a path of length n *from somewhere to somewhere* in G, or if p is a path in G then we mean a path *of some length from somewhere to somewhere* in G.

Example 2.6

$w = (2,(2,1),1,(1,2),2,(2,1),1,(1,3),3)$ is a walk of length 4 from 2 to 3 in G_2.

$$V(w) = (2,1,2,1,3) \qquad E(w) = ((2,1),(1,2),(2,1),(1,3)),$$

w is neither a trail nor a path in G_2.
$t = (2,(2,1),1,(1,2),2,(2,3),3)$ is a trail of length 3 from 2 to 3 in G_2.

$$V(t) = (2,1,2,3) \qquad E(t) = ((2,1),(1,2),(2,3)),$$

t is not a path in G_2.
$p = (2,(2,1),1,(1,3),3)$ is a path of length 2 from 2 to 3 in G_2.

$$V(p) = (2,1,3) \qquad E(p) = ((2,1),(1,3)).$$

Definition 2.7: Weighted Graph

The triple $G = (V,E,c)$ is called a *weighted graph* if and only if (V,E) is a graph and $c\colon E \to \mathbb{R}$. We call c the *cost function* of G and $c(e)$ the *costs* of an edge e.

All special properties of the graph (V,E), e.g. being simple, directed, undirected, translate directly to its weighted variant (V,E,c). A sequence is a walk/trail/path in (V,E,c) if and only if it is a walk/trail/path in (V,E).

Definition 2.8: Extended Cost Function, Distance

Let $G = (V, E, c)$ be a weighted graph and $F \subseteq E$. Then we extend the cost function c to

$$c(F) := \sum_{e \in F} c(e).$$

Let w be a walk of length n from a to b in G, then[a]

$$c(w) := \sum_{e \in E(w)} c(e).$$

The *distance* from a to b in G is

$$\text{dist}_G(a, b) := \min(\{c(w) \mid w \text{ is a walk from } a \text{ to } b \text{ in } G\}).$$

[a]Strictly speaking, $e \in E(w)$ does not make sense, because $E(w)$ is, by definition, a finite sequence and not a set. $e \in E(w)$ is meant as a conveniant shortcut notation for $\underset{1 \le i \le n}{\exists} E(w)_i = e$.

2.3 Modelling the Shortest-Connection-Problem

Our original problem from Section 2.1 can now be nicely formulated in the language of graph theory. Given a network of streets, we define the vertices

$$V := \{C \mid \text{there are streets } s \text{ and } t \text{ crossing at } C\}.$$

Then all streets split into *street segments* between crossings, in other words, a street segment is characterized by its endpoints, i.e. two crossings c_1 and c_2 on the same street such that no other crossing lies between c_1 and c_2. The street segments will become the edges connecting their endpoint crossings. If a street segment between c_1 and c_2 can be travelled in both directions, then an undirected edge $\{c_1, c_2\}$ is an appropriate model. If it can only be used in one direction, say from c_1 to c_2, then a directed edge (c_1, c_2) is an appropriate model. For typical street networks, directed edges are the better choice because these allow to model one-way streets and in case both directions are available we can use (c_1, c_2) and (c_2, c_1), thus

$$E := \{(c_1, c_2) \in V^2 \mid \text{there is a street segment from } c_1 \text{ to } c_2\}.$$

Every street segment (c_1, c_2) has costs associated, which can be the geographical distance between the c_1 and c_2 or the time required to travel from c_1 to c_2, depending on the application. Note, that the model with directed edges gives also more flexibility in this regard, because it allows to assign different times for the two directions, which is useful for instance when the street segment is a mountain road, where it is usually faster going down than going up. Note that the costs will always be non-negative, i.e.

$$c\colon E \to \mathbb{R}_0^+, (x, y) \mapsto \text{“costs” for going from } x \text{ to } y$$

The question of finding the “shortest connection” from A to B on the underlying network of roads is then just the following problem:

Problem 2.9: Shortest Path Problem

Given: The graph $G = (V, E, c)$ with appropriate cost function c and $A, B \in V$.

Find: $d = \text{dist}_G(A, B)$ and p such that p is a path from A to B in G and $c(p) = d$.

Note that it is sufficient to find a path, we need not search for walks or trails. Assume a walk

$$w = (A, e_1, \ldots, e_k, x, \ldots, x, e_l, \ldots, e_n, B)$$

from A to B in G containing vertex x twice. Then take

$$\bar{w} = (A, e_1, \ldots, e_k, x, e_l, \ldots, e_n, B),$$

which certainly gives also a walk from A to B in G with $c(\bar{w}) \leq c(w)$, such that we can always construct a walk with costs at most $c(w)$ avoiding multiple vertices. Such a walk must be a path.

Note also that we modeled the graph above as a simple graph, i.e. a graph without multiple edges and loops. If the original street network contains multiple edges, which might be the case in real world (think of examples!), then we choose the one with least cost and only put this edge into E, hence avoiding multiple edges. Consider a and b being both edges from x to y and $c(a) \leq c(b)$ and assume a path

$$p = (A, e_1, \ldots, x, b, y, \ldots, e_n, B)$$

from A to B in G. Then take

$$\bar{p} = (A, e_1, \ldots, x, a, y, \ldots, e_n, B),$$

which certainly gives also a path from A to B in G with $c(\bar{p}) \leq c(p)$, such that we can always construct a path with costs at most $c(p)$ avoiding edge b. Similary, we can eliminate loops contained in G. Consider a being a loop from x to x and assume a trail

$$p = (A, e_1, \ldots, e_k, x, a, x, e_l, \ldots, e_n, B)$$

from A to B in G. Then take

$$\bar{p} = (A, e_1, \ldots, e_k, x, e_l, \ldots, e_n, B),$$

which certainly gives also a path from A to B in G with $c(\bar{p}) \leq c(p)$ since $c(a) \geq 0$, such that we can always construct a path with costs at most $c(p)$ avoiding loop a.

2.4 An Algorithm for Solving the Shortest Path Problem

We will now briefly describe *Dijkstra's algorithm* for solving the shortest path problem. In fact, the algorithms solves a slightly more general problem, namely it computes $\text{dist}_G(A, v)$ for all $v \in V \setminus \{A\}$ and it allows to reconstruct shortest paths from A to v for all $v \in V \setminus \{A\}$.

The basic idea of the algorithm is to maintain two sets of vertices C and $O = V \setminus C$, where

- C contains the *closed vertices* v, for which $\text{dist}_G(A, v)$ is *already known* and

- O are the remaining *open vertices* v, for which only a *tentative distance* $l(v)$ *from* A *is known*. In fact, $l(v)$ is the shortest distance from A on a path containing only vertices in C except the final vertex v.

```
Data: $G = (V, E, c)$ a simple weighted directed graph, $A \in V$.
Result: $\text{dist}_G(A, v)$ for all $v \in V$,
        $\text{pre}_G(v)$ for all $v \in V$ such that[a] $P \asymp ((\text{pre}_G(v), v), v)$ is a shortest path from
        $A$ to $v$ in $G$, where $P$ is a shortest path from $A$ to $\text{pre}_G(v)$ in $G$.
$C = \emptyset$, $O = V$, $l(A) = 0$
for $v \in V \setminus \{A\}$ do
    $l(v) = \infty$
end
while $O \neq \emptyset$ do
    $v =$ such an $o \in O$ with $l(o) \leq l(x)$ for all $x \in O$
    $C = C \cup \{v\}$, $O = O \setminus \{v\}$
    $\text{dist}_G(A, v) = l(v)$
    for $w \in N_G(v)$ do
        if $l(w) > \textit{dist}_G(A, v) + c(vw)$ then
            $l(w) = \text{dist}_G(A, v) + c(vw)$
            $\text{pre}_G(w) = v$
        end
    end
end
```

Algorithm 1: Dijkstra's Algorithm

[a]We write $a \asymp b$ for the concatenation of tuples a and b.

Upon termination of the algorithm $\text{dist}_G(A, v)$ contains the shortest distance from A to v in G for all vertices v. If $\text{dist}_G(A, v) = \infty$ then there is no path from A to v in G.

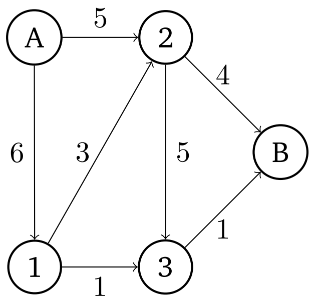

Figure 2.4: A weighted graph

EXAMPLE 2.10

Let us consider the graph depicted in Figure 2.4. After initialization we have $C = \emptyset$, $O = V$ and

v	A	1	2	3	B
$l(v)$	0	∞	∞	∞	∞

In the first run through the while-loop, we choose $v = \text{A}$, hence $C = \{\text{A}\}$ and $O = \{1, 2, 3, \text{B}\}$. We set $\text{dist}_G(\text{A}, \text{A}) = 0$ and compute $N_G(\text{A}) = \{1, 2\}$, hence we update

v	A	1	2	3	B
$l(v)$	0	6	5	∞	∞

and set $\text{pre}_G(1) = \text{A}$ and $\text{pre}_G(2) = \text{A}$. Next we choose $v = 2$, hence $C = \{\text{A}, 2\}$ and $O = \{1, 3, \text{B}\}$. We set $\text{dist}_G(\text{A}, 2) = 5$ and compute $N_G(2) = \{3, \text{B}\}$, hence we update

v	A	1	2	3	B
$l(v)$	0	6	5	10	9

and set $\text{pre}_G(3) = 2$ and $\text{pre}_G(\text{B}) = 2$. Next we choose $v = 1$, hence $C = \{\text{A}, 2, 1\}$ and $O = \{3, \text{B}\}$. We set $\text{dist}_G(\text{A}, 1) = 6$ and compute $N_G(1) = \{2, 3\}$, hence we update

v	A	1	2	3	B
$l(v)$	0	6	5	7	9

and set $\text{pre}_G(3) = 1$. Next we choose $v = 3$, hence $C = \{\text{A}, 2, 1, 3\}$ and $O = \{\text{B}\}$. We set $\text{dist}_G(\text{A}, 3) = 7$ and compute $N_G(3) = \{\text{B}\}$, hence we update

v	A	1	2	3	B
$l(v)$	0	6	5	7	8

and set $\text{pre}_G(\text{B}) = 3$. Finally, we take $v = \text{B}$, hence $C = \{\text{A}, 2, 1, 3, \text{B}\}$ and $O = \emptyset$. We set $\text{dist}_G(\text{A}, \text{B}) = 8$, no more updates necessary and we exit the while-loop.

We have all distances from A to v in $\text{dist}_G(\text{A}, v)$, and we can reconstruct the shortest path using $\text{pre}_G(v)$. The final path is

$$(\text{A}, (\text{A}, 1), 1, (1, 3), 3, (3, \text{B}), \text{B}).$$

Why does Dijkstra's algorithm work? The algorithm maintains a loop invariant, namely

$$\underset{x \in C, u \in N_G(x)}{\forall} l(u) \leq \text{dist}_G(A, x) + c(xu) \tag{2.1}$$

$$\underset{x \in C}{\forall} l(x) = \text{dist}_G(A, x) \tag{2.2}$$

$$\underset{o \in O}{\forall} l(o) \geq \text{dist}_G(A, o). \tag{2.3}$$

Before the algorithm enters the while-loop for the first time, (2.1) and (2.2) clearly hold due to $C = \emptyset$, and (2.3) holds because of $O = V$ and $l(A) = 0 = \text{dist}_G(A, A)$ and $l(o) = \infty \geq \text{dist}_G(A, o)$ for all $o \neq A$. Now assume (2.1), (2.2), and (2.3) hold at the beginning of one pass through the loop, we will show that (2.1), (2.2), and (2.3) then also hold at the end of that pass, i.e. we have to show

$$\underset{x \in C \cup \{v\}, u \in N_G(x)}{\forall} l(u) \leq \text{dist}_G(A, x) + c(xu) \tag{2.4}$$

$$\underset{x \in C \cup \{v\}}{\forall} l(x) = \text{dist}_G(A, x) \tag{2.5}$$

$$\underset{o \in O \setminus \{v\}}{\forall} l(o) \geq \text{dist}_G(A, o). \tag{2.6}$$

Proof of (2.4): Let $x \in C \cup \{v\}$ and $u \in N_G(x)$. In case $x \in C$, $l(u) \leq \text{dist}_G(A, x) + c(xu)$ is true by assumption (2.1). Now let $x = v$. By the algorithm, we have $l(w) \leq \text{dist}_G(A, v) + c(vw)$ for all $w \in N_G(v)$ after finishing the for-loop, hence $l(u) \leq \text{dist}_G(A, x) + c(xu)$.

Proof of (2.5): Let $x \in C \cup \{v\}$. In case $x \in C$, $l(x) = \text{dist}_G(A, x)$ is true by assumption (2.2). Now let $x = v$. First of all we show $l(x) \leq \text{dist}_G(A, x)$ by contradiction, hence, we assume $l(x) > \text{dist}_G(A, x)$, i.e. there must be a path P from A to x in G with $c(P) = \text{dist}_G(A, x) < l(x)$. P contains at least one vertex in O since $x = v \in O$, hence, there is a minimal i such that $p_i := V(P)_i \in O$. We then have $l(p_i) \leq \text{dist}_G(A, p_i)$ because

case $i = 1$: then $p_1 = A$ and $l(p_1) = l(A) = 0 = \text{dist}_G(A, A) = \text{dist}_G(A, p_1)$ and

case $i > 1$: from the minimality of i we get $p_{i-1} := V(P)_{i-1} \in C$ and clearly $p_i \in N_G(p_{i-1})$, hence

$$l(p_i) \overset{(2.1)}{\leq} \text{dist}_G(A, p_{i-1}) + c(p_{i-1}p_i) = c(P_{1:2i-1}) = \text{dist}_G(A, p_i).$$

Note that the last equality holds, because P is a path with lowest costs from A to x. Therefore, any subpath of P from A to b must be one with lowest costs to b, because otherwise P would not have lowest costs to x.

Finally, we have the contradiction

$$l(x) = l(v) \overset{\text{choice of } v}{\leq} l(p_i) \leq \text{dist}_G(A, p_i) \overset{\text{non-negative weights}}{\leq} \text{dist}_G(A, x) < l(x).$$

Thus, $l(x) \leq \text{dist}_G(A, x)$ and together with $l(x) = l(v) \geq \text{dist}_G(A, v) = \text{dist}_G(A, x)$ by assumption (2.3) since $v \in O$ we have $l(x) = \text{dist}_G(A, x)$.
Proof of (2.6): Let $o \in O \setminus \{v\}$. In case $l(o) = \infty$ then $l(o) = \infty \geq \text{dist}_G(A, o)$ is trivial. Otherwise $l(o)$ reflects the costs of a concrete path from A to o, hence $l(o) \geq \text{dist}_G(A, o)$, by definition of $\text{dist}_G(A, o)$.

REMARK 2.11

As already mentioned earlier the algorithm computes the shortest distances from A to all v in G. If one is only interested in the shortest distances from A to B then the while-loop can be terminated as soon as $v = B$ has been chosen and $\text{dist}_G(A, B)$ has been set. Those $\text{dist}_G(A, v)$ that have been set until then are the true shortest distances, but there may be vertices v, for which $\text{dist}_G(A, v)$ is still undefined.

REMARK 2.12

For a real implementation of Dijkstra's algorithm one should use special data-structures for storing O such that the choice of v as the $o \in O$ with minimal tentative distance can be performed efficiently. Keywords in this respect are *k-heaps* or *Fibonacci heaps*.

2.5 A Real-World Problem

As an example, we used a graph representing the street network of New York City, which is freely available from `http://users.diag.uniroma1.it/challenge9/download.shtml`. Geographical coordinates of all vertices are also available such that the street crossings can actually be visualized on a map of New York City. We have only used a small fragment of the

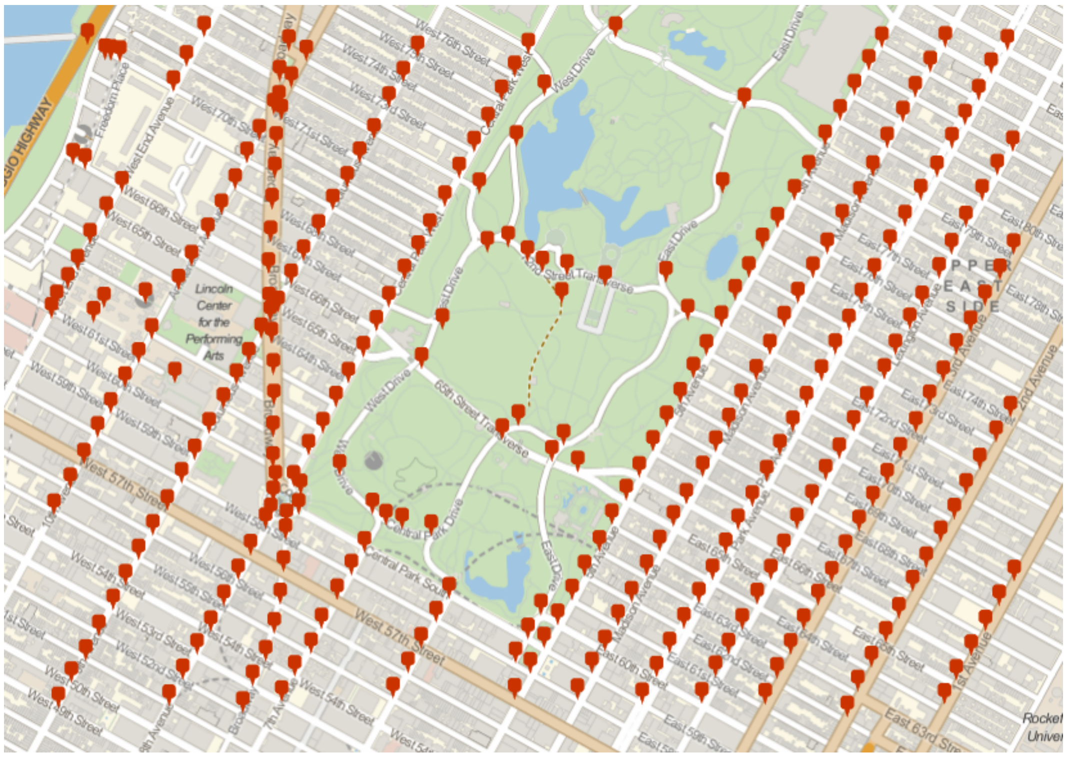

Figure 2.5: Street crossings defining street segments

vertives and edges such that computation of shortest paths was still feasible in Mathematica. Figures 2.5 shows the street crossings with their real-world positions on a map and Figure 2.6 shows the whole graph with all connections between vertices.

We then chose the west-most point as starting point A and the east-most point as the end point B and compute a shortest path using the mathematica command `FindShortestPath`, which computes a path of length 22. The abstract graph with the edges of the shortest path highlighted and the corresponding path on the New Yourk City map are shown in Figures 2.7 and 2.8.

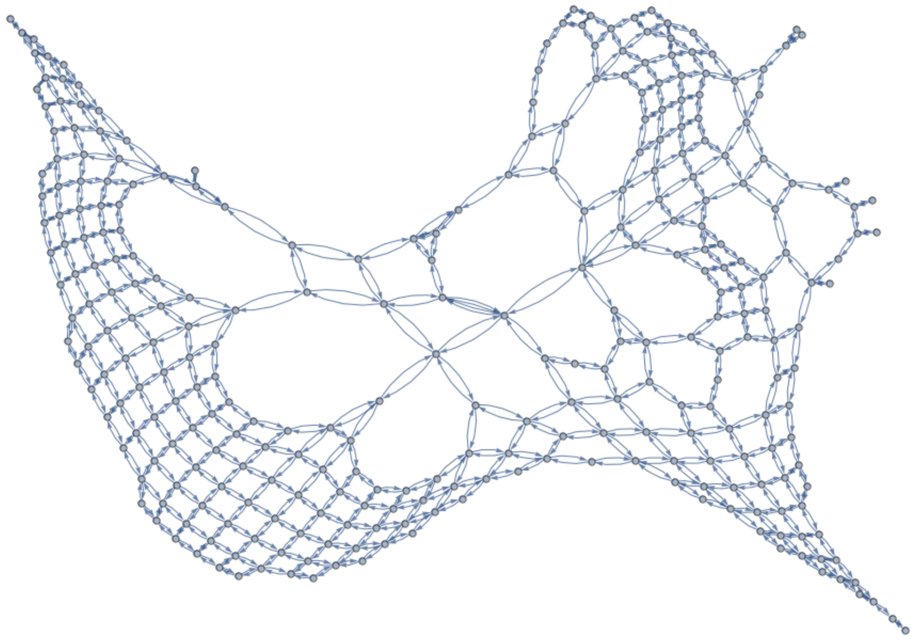

Figure 2.6: Street crossings and street segments as a directed graph

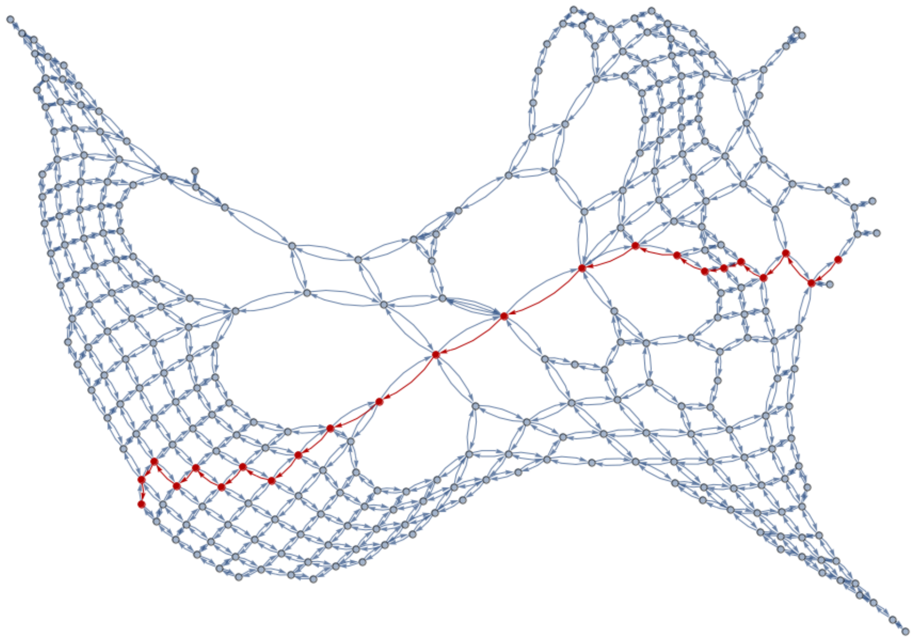

Figure 2.7: Street crossings defining street segments and the shortest path between two points

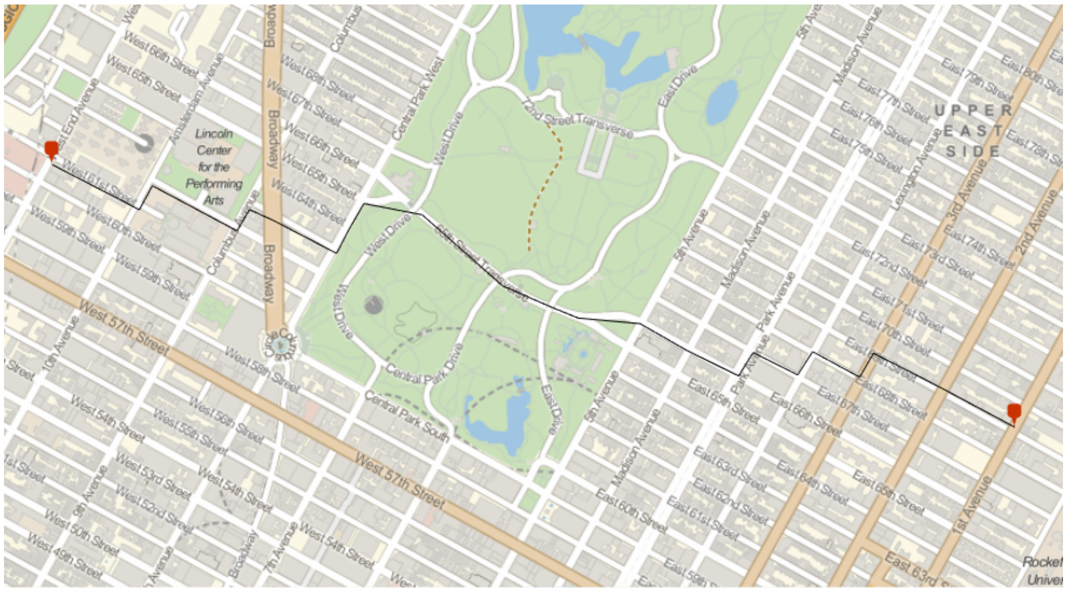

Figure 2.8: The shortest path on the map

Chapter 3

Modelling in Combinatorial Optimization

Combinatorial optimization usually deals with the optimization of an objective function over a *finite domain*. In most of the cases the variables are restricted to *integers or natural numbers* and there are restrictions that allow only finitely many *feasible solutions*. Although in principle possible, exhaustive search through all finitely many feasible solutions is practically not an option because of their huge number. Probably the most famous combinatorial optimization problem is the *travelling salesman problem*, other examples are the *minimum spanning tree problem*, the *knapsack problem*, or the *bin packing problem*.

3.1 An Introductory Example

Suppose we run an online shop and we deliver our goods in boxes of maximum capacity C. We have a concrete order with n items of sizes $s_1, \dots, s_n$, respectively. The size of an item could be for instance the weight or the volume and the capacity would then be the maximum weight allowed or the maximum space available in the box. The question is how to pack the items into a minimal number of boxes such that all capacity restrictions are still satisfied. In case the capacity is interpreted as volume we neglect the geometrical problem of fitting items of a certain shape into the boxes having a certain shape, i.e. if we have items with a total volume of x then we assume these items fit into a box with volume $V \geq x$. It is clear that in practice, by the geometry of the box and the items, this might lead to inappropriate solutions. As a simple example, consider a sphere of volume 1, which has a diameter of ≈ 1.24. Clearly, the spere dos not fit into a unit cube with side lenghts 1, although the volume restriction would be satisfied.

3.2 Modelling the Packing Problem

The problem described in Section 3.1 is known in literature as the *bin packing problem*.

Problem 3.1: Bin Packing Problem

Given: Positive numbers $a_1, \dots, a_n, A$.

Find: $k \in \mathbb{N}$ and $p \colon \mathbb{N}_{1,n} \to \mathbb{N}_{1,k}$ such that

$$\underset{1 \leq j \leq k}{\forall} \sum_{\substack{i \\ p(i)=j}} a_i \leq A.$$

We consider n items $I_1, \dots, I_n$ and need to find a number k of bins $B_1, \dots, B_k$ such that each I_i goes into one of the B_j. The packing function p assigns each item index i a bin index j, and $p(i) = j$ means that item I_i is packed into bin B_j. If k and p satisfy the above conditions we call the pair (k, p) a *feasible solution* (for the bin packing problem with respect to $a_1, \dots, a_n, A$).

Often, in particular in the context of complexity theory, problems are stated as decision problems. For the bin packing problem, this variant is the following.

PROBLEM 3.2: BIN PACKING DECISION PROBLEM

Given: Positive numbers $a_1, \dots, a_n, A$ and $k \in \mathbb{N}$.

Question: Does there exist a function $p\colon \mathbb{N}_{1,n} \to \mathbb{N}_{1,k}$ such that (k, p) a feasible solution for the bin packing problem with respect to $a_1, \dots, a_n, A$?

If there is a feasible solution (k, p) for the bin packing problem then of course (m, q) is again a feasible solution for all $m > k$ and $q\colon \mathbb{N}_{1,n} \to \mathbb{N}_{1,m}$ such that $q(i) = p(i)$ for all $1 \leq i \leq n$, hence there are infinitely many solutions. It is then pretty natural to ask for the best possible solution.

PROBLEM 3.3: OPTIMAL BIN PACKING PROBLEM

Given: Positive numbers $a_1, \dots, a_n, A$.

Find: $k \in \mathbb{N}$ and $p\colon \mathbb{N}_{1,n} \to \mathbb{N}_{1,k}$ such that

1. (k, p) a feasible solution for the bin packing problem w.r.t. $a_1, \dots, a_n, A$ and
2. k is minimal, i.e.

$$\underset{m<k}{\forall} \ \underset{q\colon \mathbb{N}_{1,n}\to\mathbb{N}_{1,m}}{\forall} (m, q) \text{ is not a feasible solution for the bin packing problem with respect to } a_1, \dots, a_n, A.$$

3.2.1 Bin Packing as a Linear Optimization Problem

A first approach for solving the optimal bin packing problem is to re-formulate the problem as a linear programming (linear optimization) problem. First we observe that $k \leq n$. We introduce *binary decision variables* x_{ij} for $1 \leq i, j \leq n$ and y_j for $1 \leq j \leq n$ with the following meaning:

$$x_{ij} := \begin{cases} 1 & \text{item } I_i \text{ goes into bin } B_j \\ 0 & \text{otherwise} \end{cases} \qquad y_j := \begin{cases} 1 & \text{bin } B_j \text{ will be occupied} \\ 0 & \text{otherwise} \end{cases}$$

The feasibility of a solution for the bin packing problem can then be described by

$$\underset{1\leq j\leq n}{\forall} \sum_{i=1}^{n} x_{ij} a_i \leq A y_j \tag{3.1}$$

$$\underset{1\leq i\leq n}{\forall} \sum_{j=1}^{n} x_{ij} = 1, \tag{3.2}$$

where (3.1) captures the capacity restrictions for each bin and (3.2) enforces that each article goes into exactly one bin. The number of bins used is obviously $\sum_{j=1}^{n} y_j$, hence, (3.1) and (3.2) together with

$$\sum_{j=1}^{n} y_j \longrightarrow \text{Min} \tag{3.3}$$

form an *integer linear programming problem* with the integer variables

$$x_{ij} \in \{0,1\} \text{ for } 1 \leq i,j \leq n \quad \text{and} \quad y_j \in \{0,1\} \text{ for } 1 \leq j \leq n.$$

We write *BPLP*(a, A) for this problem. It is easy to reconstruct a solution for the optimal bin packing problem from a solution (x, y) of *BPLP*(a, A). For this, let $k := \sum_{j=1}^{n} y_j$ and let $\pi\colon \mathbb{N}_{1,n} \to \mathbb{N}_{1,n}$ be a permutation of $\mathbb{N}_{1,n}$ such that

$$\underset{1\leq i\leq k}{\forall} y_{\pi(i)} = 1 \wedge \underset{k<i\leq n}{\forall} y_{\pi(i)} = 0. \tag{3.4}$$

π permutes the bins in such a way that the first k bins $y_{\pi(1)}, \ldots, y_{\pi(k)}$ will be occupied and the remaining $n-k$ bins $y_{\pi(k+1)}, \ldots, y_{\pi(n)}$ are empty. From (3.2) it follows immediately that for every $1 \leq i \leq n$ there is a unique $1 \leq j \leq n$ with $x_{ij} = 1$, thus the function

$$p\colon \mathbb{N}_{1,n} \to \mathbb{N}_{1,k}, i \mapsto \text{the unique } j \text{ with } x_{i\pi(j)} = 1$$

is well-defined, since π just permutes the columns of (x_{ij}) in such a way that zero-columns are moved to the end. In order to see that (k, p) is feasible, we take $1 \leq j \leq k$ arbitrary but fixed and now

$$\sum_{\substack{i \\ p(i)=j}} a_i = \sum_{\substack{i \\ x_{i\pi(j)}=1}} a_i = \sum_{i=1}^{n} x_{i\pi(j)} a_i \overset{(3.1)}{\leq} A y_{\pi(j)} \overset{(3.4)}{=} A.$$

The minimality of k follows easily from the definition of k together with (3.3).

EXAMPLE 3.4

Consider 7 articles with weights

$$a_1 = 0.2 \qquad a_2 = 0.5 \qquad a_3 = 0.4 \qquad a_4 = 0.7 \qquad a_5 = 0.1 \qquad a_6 = 0.3 \qquad a_7 = 0.8$$

and bins with maximum capacity of 1. The formulation as a linear programming problem now assumes at most $n = 7$ bins and introduces 49 variables $x_{11}, \ldots, x_{77}$ plus 7 variables $y_1, \ldots, y_7$, hence, a total of 56 variables. We have 7 capacity restrictions (3.1)

$$0.2x_{1j} + 0.5x_{2j} + 0.4x_{3j} + 0.7x_{4j} + 0.1x_{5j} + 0.3x_{6j} + 0.8x_{7j} \leq y_j \quad \text{for } 1 \leq j \leq 7$$

and 7 unicity restrictions (3.2)

$$x_{i1} + x_{i2} + x_{i3} + x_{i4} + x_{i5} + x_{i6} + x_{i7} = 1 \quad \text{for } 1 \leq i \leq 7,$$

which results in a solution

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

which translates into $k =$ and a mapping p for the articles

$$p(1) = 1 \qquad p(2) = 2 \qquad p(3) = 2 \qquad p(4) = 3 \qquad p(5) = 2 \qquad p(6) = 3 \qquad p(7) = 1.$$

3.2.2 Heuristic Approximation Algorithms for the Bin Packing Problem

For big instances of bin packing problems the integer linear programming problem can be computationally very expensive. Therefore, approximation algorithms based on heuristics are very popular for solving huge bin packing problems.

In the following, we write P for a problem and $P(x)$ for an instance of problem P with input x. If A is an algorithm, then $A(x)$ denotes the result of algorithm A applied to input x.

DEFINITION 3.5: APPROXIMATION ALGORITHM

Let P be an optimization problem and $\bar{P}$ the relaxed problem ignoring optimality. We call A an *approximation algorithm* for problem P if and only if every $y = A(x)$ is still a solution of $\bar{P}(x)$ but not necessarily a solution of $P(x)$.

DEFINITION 3.6: APPROXIMATION QUALITY

Let P be an optimization problem and $y(x)$ a solution for $P(x)$. Let furthermore $k \geq 1$. We call A a *k-approximation algorithm* for problem P if and only if for all admissible inputs x of P

$$A(x) \begin{cases} \leq k y(x) & \text{if } P \text{ is a minimization problem} \\ \geq \frac{1}{k} y(x) & \text{if } P \text{ is a maximization problem} \end{cases}$$

THEOREM 3.7

If $P \neq NP$[a], then there is no k-approximation algorithm for the optimal bin packing problem with $k < \frac{3}{2}$.

[a] we do not go into details

```
Data: Positive numbers $a_1, \dots, a_n, A$.
Result: $k, p$ such that $(k, p)$ is a solution for the bin packing problem
$k = 0$
sort $a$ into decreasing order
for $j = 1, \dots, n$ do
    $c_j = A$
end
for $i = 1, \dots, n$ do
    $m = 0$
    for $j = 1, \dots, k$ do
        if $a_i \le c_j$ then
            $m = j$
            $p(i) = j$
            $c_j = c_j - a_i$
        end
    end
    if $m = 0$ then
        $k = k + 1$
        $p(i) = k$
        $c_k = c_k - a_i$
    end
end
```

Algorithm 2: First Fit Decreasing Heuristic (FFD)

THEOREM 3.8

FFD is a $\frac{3}{2}$-approximation algorithm for the optimal bin packing problem.

THEOREM 3.9

For all instances x of the bin packing problem with solution $y(x)$ we have

$$\mathit{FFD}(x) \le \frac{11}{9} y(x) + \frac{2}{3}.$$

EXAMPLE 3.10

Consider again 7 articles with weights

$$a_1 = 0.2 \qquad a_2 = 0.5 \qquad a_3 = 0.4 \qquad a_4 = 0.7 \qquad a_5 = 0.1 \qquad a_6 = 0.3 \qquad a_7 = 0.8$$

and bins with maximum capacity of 1. Running the FFD-algorithm on this instance of the bin packing problem gives the same solution as shown in Example 3.4.