Linear & Combinatorial Optimization

401-3901-00

Rico Zenklusen

Fall term 2022

This script has profited enormously from the input and help of many people. In particular, I would like to thank Simon Bruggmann, Tom Häring, Martin Nägele, Elias Wirth, and Ivan Sergeev for their help with this script in various ways. Moreover, I am grateful to all students who provided very valuable feedback on a wide variety of aspects, which significantly increased the quality of this script.

Rico Zenklusen

Contents

1 Linear Programming and Polyhedra

1.1 Introduction to linear programming

Linear programming captures one of the most canonical and influential constrained optimization problems. More precisely, it asks to maximize or minimize a linear objective under linear inequality and equality constraints. Below is a concrete example of a linear programming problem:

$$\begin{array}{rrcrcrcr}
\max & 6x_1 & + & 5x_2 & + & 5.5x_3 & & \\
\text{subject to} & 10x_1 & - & x_2 & - & 2.5x_3 & \geq & 11.5\\
 & -21x_1 & + & x_2 & - & 6x_3 & \geq & -104\\
 & 4.25x_1 & + & 2.75x_2 & - & x_3 & \leq & 24\\
 & & & x_2 & & & \geq & 0\\
 & 10x_1 & - & x_2 & + & 35x_3 & \geq & 49\\
 & x_1 & & & + & 2x_3 & = & 12\ ,
\end{array}$$

where the variables x_1, x_2, x_3 all take real values. Such a problem is often called a *linear program* or simply *LP*. As highlighted in the above example, the objective depends linearly on the variables, and each constraints imposes either a lower bound, an upper bound, or an equality condition on a linear form of the variables. Finally, in a linear program one can either ask to maximize or minimize the objective. Hence, formally, a general linear program is of the following form:

$$\begin{array}{rrcl}
\max / \min & c^\top x & & \\
 & Ax & \leq & e\\
 & Bx & \geq & f\\
 & Cx & = & g\ ,
\end{array} \tag{general LP}$$

where A, B, and C are real matrices, and c, e, f, and g are real column vectors of the appropriate dimensions.

Linear programs can be reformulated in various equivalent forms. When talking about structural results or algorithms, it is often convenient to fix one particular form of writing a linear program, like the *canonical form*, which looks as follows:

$$\begin{array}{rrcl}
\max & c^\top x & & \\
 & Ax & \leq & b\\
 & x & \geq & 0\ ,
\end{array} \tag{LP in canonical form}$$

where, if $n \in \mathbb{Z}_{\geq 0}$ is the number of variables and $m \in \mathbb{Z}_{\geq 0}$ the number of constraints, we have $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and the variables x take values in $\mathbb{R}^n$. Indeed, any linear program can be transformed into canonical form as follows:

(i) Any constraint of type $a^\top x \geq \beta$ can be rewritten as $(-a)^\top x \leq -\beta$.

(ii) Any constraint of type $a^\top x = \beta$ can be replaced by a pair of equivalent constraints $a^\top x \leq \beta$ and $(-a)^\top x \leq -\beta$.
(iii) Any variable x_i that is not bound by a non-negativity constraint $x_i \geq 0$ can be replaced as follows. Introduce two new variables x_i^+ and x_i^- with non-negativity constraints $x_i^+ \geq 0$ and $x_i^- \geq 0$, and replace all occurrences of x_i by $x_i^+ - x_i^-$.
(iv) If the problem is a minimization problem, then replace the objective $\min c^\top x$ by $\max -c^\top x$. This will flip the sign of all solution values. However, maximizers of the new problem are minimizers of the original problem and vice versa.

Notice that linear programming can also be equivalently described as the task to maximize a linear function over a polyhedron, which is the intersection of finitely many half-spaces in finite-dimensional Euclidean space.

1.1.1 Different types of LPs and goal of LP algorithms

Linear programs are often divided into the following three types, depending on their optimal value:

LP with finite optimum An LP with a finite optimal value. The optimum is either attained at a unique point or the LP may have multiple optimal solutions.

Unbounded LP An LP with feasible solutions of arbitrarily large value in case of a maximization problem, or of arbitrarily small value in case of a minimization problem.

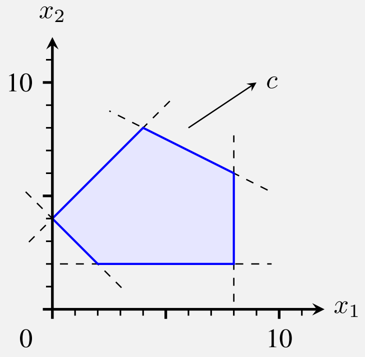
Infeasible LP An LP without any feasible solutions. If it is a maximization problem, its optimal value is often set to $-\infty$ by convention, and to ∞ for minimization problems.

In the three examples below, we exemplify the above notions on LPs in 2 dimensions.

Example 1.1: LP with finite optimum

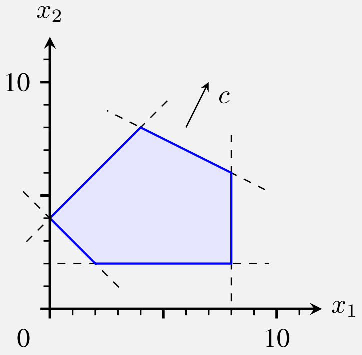
Below is an LP with a unique optimum, attained at the point $(x_1, x_2) = (8, 6)$.

$$
\begin{array}{lrcrcr}
\max & 3x_1 & + & 2x_2 & & \\
 & -x_1 & + & x_2 & \leq & 4 \\
 & x_1 & + & 2x_2 & \leq & 20 \\
 & x_1 & & & \leq & 8 \\
 & & & x_2 & \geq & 2 \\
 & x_1 & + & x_2 & \geq & 4
\end{array}
$$



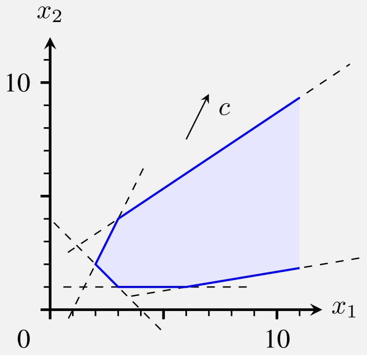
Taking the same example as above with a different objective function that is perpendicular to one of the sides of the polygon describing the feasible region, an LP with multiple optimal solutions is obtained.

$$\begin{array}{lrcrcr}
\max & x_1 & + & 2x_2 & & \\
 & -x_1 & + & x_2 & \leq & 4 \\
 & x_1 & + & 2x_2 & \leq & 20 \\
 & x_1 & & & \leq & 8 \\
 & & & x_2 & \geq & 2 \\
 & x_1 & + & x_2 & \geq & 4
\end{array}$$



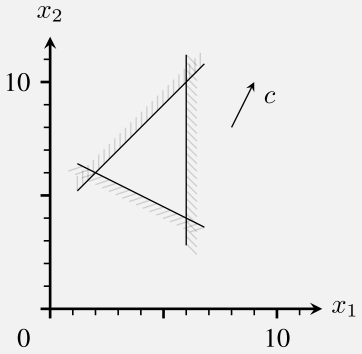
Example 1.2: Unbounded LP

$$\begin{array}{lrcrcr}
\max & x_1 & + & 2x_2 & & \\
 & x_1 & - & 6x_2 & \leq & 0 \\
 & & & x_2 & \geq & 1 \\
 & x_1 & + & x_2 & \geq & 4 \\
 & 2x_1 & - & x_2 & \geq & 2 \\
 & -2x_1 & + & 3x_2 & \leq & 6
\end{array}$$



Example 1.3: Infeasible LP

$$\begin{array}{lrcrcr}
\max & x_1 & + & 2x_2 & & \\
 & -x_1 & + & x_2 & \geq & 4 \\
 & x_1 & + & 2x_2 & \leq & 14 \\
 & x_1 & & & \geq & 6
\end{array}$$



When talking about an algorithm that solves linear programs, we expect the algorithm to provide answers to the following questions. First, the algorithm should detect the type of LP we are dealing with, i.e., whether it is an LP with finite optimum, an unbounded LP, or an infeasible one. Normally, in case of an LP with finite optimum, one does not require an LP algorithm to distinguish between whether there is a unique optimal solution or multiple ones. Second, the algorithm should return the following.

- If the LP has a finite optimum, the algorithm should return an optimal solution.
- If the LP is unbounded, the algorithm should return a half-line pointing in an improving unbounded direction. More precisely, assuming that we want to maximize the objective c, this consists of a feasible point $y \in \mathbb{R}^n$ and a non-zero vector $v \in \mathbb{R}^n$ such that
  (i) $y + \lambda v$ is feasible for any $\lambda \in \mathbb{R}_{\geq 0}$, and
  (ii) $c^\top v > 0$.

Hence, in the first problem of Example 1.1, an LP algorithm must return the unique optimal solution $\binom{8}{6}$, whereas in the second LP of the same example, any point on the segment connecting $\binom{8}{6}$ and $\binom{4}{8}$ can be returned. For the unbounded LP shown in Example 1.2, an LP algorithm needs to return a strictly improving half-line, for example $\binom{5}{2} + \lambda \cdot \binom{3}{1}$ for $\lambda \in \mathbb{R}_{\geq 0}$.

Whereas the above highlights minimal requirements that we expect LP algorithms to fulfill, sometimes, additional information is desired, as for example:

- In case of an LP with finite optimum such that there is a corner of the feasible region at which the optimum is attained, we may want that the LP algorithm returns an optimal solution that is a corner of the feasible region.
- In case of an LP with finite optimum, we may want a certificate of optimality.
- In case of an infeasible LP, we may want to obtain a certificate of infeasibiliy.

Informally speaking, a certificate of optimality or infeasibility is a piece of information with which one can check quickly that a certain solution is optimal or a certain LP is infeasible, respectively. We formally provide such certificates later on. The Simplex Method, which is an algorithm to solve linear programs, together with the concept of linear duality that we introduce later, allows for obtaining the above-mentioned additional information.

1.1.2 Example applications

Before we talk about structural and algorithmic results related to linear programming, we present some of its numerous, and sometimes surprising, applications.

Production planning

Linear programs, and variations and generalizations thereof, are often used in Operations Research problems coming from industrial applications. In this example, we consider an extreme simplification of such a real-world problem, to provide a glimpse of why linear programs can be relevant in such contexts, and to introduce some terminology that is prevalent in Operations Research when dealing with linear programs.

To this end, consider the following heavily simplified production planning example focusing on a recycling facility that recycles a raw material into a purified final form, which we simply call the *recycled material*. We are interested in determining the maximum number of tons of recycled material that the facility can produce per day. There are two different types of raw materials, type A and type B, both being recycled to the same final recycled material. These raw materials have different types and levels of impurity.

At most 20 t of total raw material of both types together can be transported to the recycling facility per day. For every ton of raw material of type A, 720 kg of recycled material can be produced, whereas for every ton of raw material of type B, 600 kg of recycled material can be obtained. Moreover, if a mix of raw material A and B is used, then the total kg of recycled material that can be produced behaves linearly with respect to the above values.

The recycling process can handle 16 t/d (tons per day) of raw material A, if only raw material A is used, and 24 t/d of raw material B, if only raw material B is used. We assume that the number of tons per day of a mix of raw materials A and B that can be processed behaves linearly with respect to the above values, e.g., the recycling process can simultaneously handle 8 t/d of raw material A and 12 t/d of raw material B.

The purity of the recycled material depends on the proportions of raw material A and B that are being used. This is measured though an *impurity coefficient* which should be no more than 30 for the recycled material. Only using raw material A will lead to an impurity coefficient of 18 for the recycled material, and the obtained impurity coefficient when only using B is 38. A mix of both raw materials leads to an impurity coefficient that behaves linearly with respect to the above values, e.g., when using the same amount of raw material A and B, the recycled material has impurity coefficient 28.

Finally, the recycling process leads to smoke emissions, and the recycling facility wants to adhere to strict emission limits by not emitting more than 12 kg of smoke per day. For every ton of raw material A that is used, 0.5 kg of smoke are emitted, and for every ton of raw material B the smoke emission is 1 kg.

To determine the maximum number of tons of recycled material that the recycling facility can produced per day subject to the above conditions, we set up a linear program.

Decision variables The key parameters to be determined are the total number of tons per day that are used of raw material A and B, respectively. Because these are the parameters that can be controlled in this decision problem, the following two variables, which correspond to these "free" parameters, are called *decision variables*:

- $x_1 \in \mathbb{R}_{\geq 0}$: amount of raw material A used per day (in t/d), and
- $x_2 \in \mathbb{R}_{\geq 0}$: amount of raw material B used per day (in t/d).

Objective function The total number of tons of recycled material per day depends on the number of tons of raw material A and B that are recycled daily. Because each ton of raw material A and B lead to 0.72 t and 0.6 t of recycled material, respectively, and, as described, mixtures behave linearly, the objective is to maximize the following linear form:

$$\underbrace{0.72\, x_1 + 0.6\, x_2}_{\text{objective function}} \quad = \quad \underbrace{z}_{\substack{\text{value of the}\\ \text{objective function}}} \quad .$$

Hence, in the two-dimensional (x_1, x_2)-space, the level curve for every fixed value of z is a line.

Constraints In addition to the non-negativity requirements, the variables are also constrained by further conditions that can be of various types, like physical, economic, or legal conditions.

Smoke emission constraints. The maximum smoke emission must be no more than 12 kg/d. We know that 0.5 kg of smoke are emitted per ton of raw material A and 1 kg of smoke per ton of raw material B. When x_1 tons of A and x_2 tons of B are being used per day, we have

$$0.5\,x_1 \;+\; 1\,x_2$$

kilograms of smoke emitted per day. The following constraint makes sure that this value does not exceed the emission limit of 12 kg/d:

$$\begin{array}{lllll} 0.5x_1 & + & 1x_2 & \leq & 12\ . \\ \uparrow & & \uparrow & & \uparrow \\ \multicolumn{3}{l}{\text{coefficients of the “left-hand side (lhs)”}} & \multicolumn{2}{l}{\text{“right-hand side (rhs)”}} \end{array}$$

Transport capacities. The transport capacity of 20 t of total raw material per day is captures by the following constraint:

$$x_1 + x_2 \leq 20\ .$$

Recycling performance. The facility can recycle 16 t/d of raw material A and 24 t/d of raw material B. In other words, $^1\!/_{16}$ d is needed to recycle one ton of A and $^1\!/_{24}$ d to recycle one ton of B. Because the processing of a mix of A and B behaves linearly, processing x_1 tons of raw material A and x_2 tons of raw material B takes

$$\frac{1}{16}x_1 + \frac{1}{24}x_2 \quad \text{days.}$$

Thus we get the following inequality for the recycling performance constraint:

$$\frac{1}{16}x_1 + \frac{1}{24}x_2 \leq 1\ .$$

Expressing the maximum recycling performance in time units per ton, we were able to link them with the time constraints.

Purity constraint. We recall that the required impurity coefficient of the recycled material must not exceed 30 and depends on the proportions of the raw materials used. Raw material A alone leads to recycled material of impurity coefficient 18, whereas raw material B alone leads to an impurity coefficient of 38. The proportion of raw material A used among all the used raw material is $\frac{x_1}{x_1+x_2}$ and, accordingly, the proportion of raw material B is $\frac{x_2}{x_1+x_2}$. The final impurity coefficient is the weighted average of the impurity obtained when only using A or B, respectively, which leads to the following purity constraint:

$$18 \cdot \frac{x_1}{x_1+x_2} + 38 \cdot \frac{x_2}{x_1+x_2} \leq 30\ .$$

Even though this constraint is not linear, it can be linearized by multiplying both sides by x_1+x_2. This leads to the linear constraint

$$12x_1 - 8x_2 \geq 0\ ,$$

which ensures that the required purity level is achieved for the recycled material.

Mathematical formulation Summarizing the above discussion, we obtain the following linear program, which is a mathematical model of the described problem.

$$
\begin{array}{rrcrcrl}
\max & 0.72x_1 & + & 0.6x_2 & & & \\
 & \frac{1}{2}x_1 & + & x_2 & \leq & 12 & \text{(smoke emission)} \\
 & x_1 & + & x_2 & \leq & 20 & \text{(transport capacity)} \\
 & \frac{1}{16}x_1 & + & \frac{1}{24}x_2 & \leq & 1 & \text{(recycling performance)} \\
 & 12x_1 & - & 8x_2 & \geq & 0 & \text{(purity)} \\
 & x_1 & & & \geq & 0 & \text{(non-negativity)} \\
 & & & x_2 & \geq & 0 & \text{(non-negativity),}
\end{array}
$$

or, in matrix notation in canonical form, $\max\ c^\top x$ subject to $Ax \leq b$ and $x \geq 0$, where

$$
c = \begin{pmatrix} 24 \\ 20 \end{pmatrix}, \quad A = \begin{pmatrix} \frac{1}{2} & 1 \\ 1 & 1 \\ \frac{1}{16} & \frac{1}{24} \\ -12 & 8 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \text{and} \quad b = \begin{pmatrix} 12 \\ 20 \\ 1 \\ 0 \end{pmatrix} .
$$

Note that, to obtain canonical form, we multiplied the purity constraint by -1 to convert the inequality type from "$\geq$" into "$\leq$".

Graphical solution method As our example is modeled by two variables, every constraint can be graphically represented in the (x_1, x_2)-space (see Figure 1.1). In order for a combination of activities to be *feasible*, it has to satisfy all constraints *simultaneously*, including the non-negativity constraints. We will call such points *feasible solutions* and the set of all feasible solutions is called *feasible region*. In our example, the feasible region is the shaded area in Figure 1.1.

If removing a constraint does not change the feasible region, then this constraint is called *redundant*. Thus, removing redundant constraints does not change the problem. In our example, the transport capacity constraint is redundant (see Figure 1.1). Whereas removing redundant constraints may sometimes lead to a desirable simplification of the problem, finding and eliminating them in large LPs may take considerable time, and is therefore often impractical. Moreover, when considering slight modifications of the entries of an LP—which is for example done in sensitivity analysis—it may be that a redundant constraint becomes non-redundant. In any case, the approaches we will study to solve LPs work irrespectively of the presence of redundant constraints.

When superposing the level curves of the objective function with the feasible region, we immediately obtain a graphical solution method. Moving the level curves in parallel with increasing value as far as possible while still intersecting the feasible region, we can read off the optimal value and solution from the graphical illustration (see Figure 1.2): $z = 12.24$ is the maximum value of the objective function, which is achieved at the corner of the feasible region that is named q in Figure 1.2 and which is given by the coordinates $x_1 = 12$ and $x_2 = 6$. Hence, a combination of 12 tons of raw material A and 6 tons of raw material B per day maximizes the total number of tons of recycled material under the constraints discussed above.

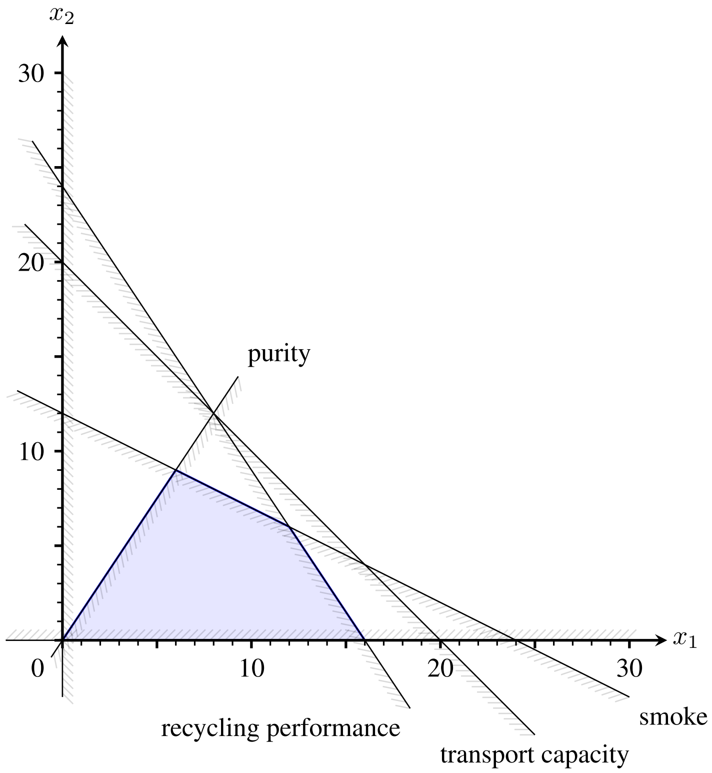


Figure 1.1: Constraints of the recycling example.

ℓ_1 regression

In statistics, regression analysis is used to estimate the relationships between different variables. In this section, we consider linear ℓ_1 regression. Suppose we have a set of input points $x_1, \dots, x_k \in \mathbb{R}^n$, and for each input point x_i we have a response variable $y_i \in \mathbb{R}$ that we observed. In linear regression, we consider a model where we assume that the response variables are an affine transformation of the input points plus some random error. This leads to the question of finding an affine function $f(x) := a^\top x + \beta$, where $a \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ such that $f(x_1), \dots, f(x_k)$ is a good approximation of $y_1, \dots y_k$. To quantify how good the approximation is, ℓ_1 regression uses the ℓ_1 norm (contrary to the more common square error, which uses the ℓ_2 norm), leading to the following optimization problem

$$\min \left\{ \sum_{i=1}^{k} |y_i - a^\top x_i - \beta| \colon a \in \mathbb{R}^n, \beta \in \mathbb{R} \right\} \quad . \tag{1.1}$$

Notice that because the response variables are scalars, the ℓ_1 norm of the error $y_i - a^\top x_i - \beta$ is simply its absolute value. More generally, we could have response variables in a higher-dimensional space $\mathbb{R}^m$; the discussion here extends to this generalization in a straightforward way.

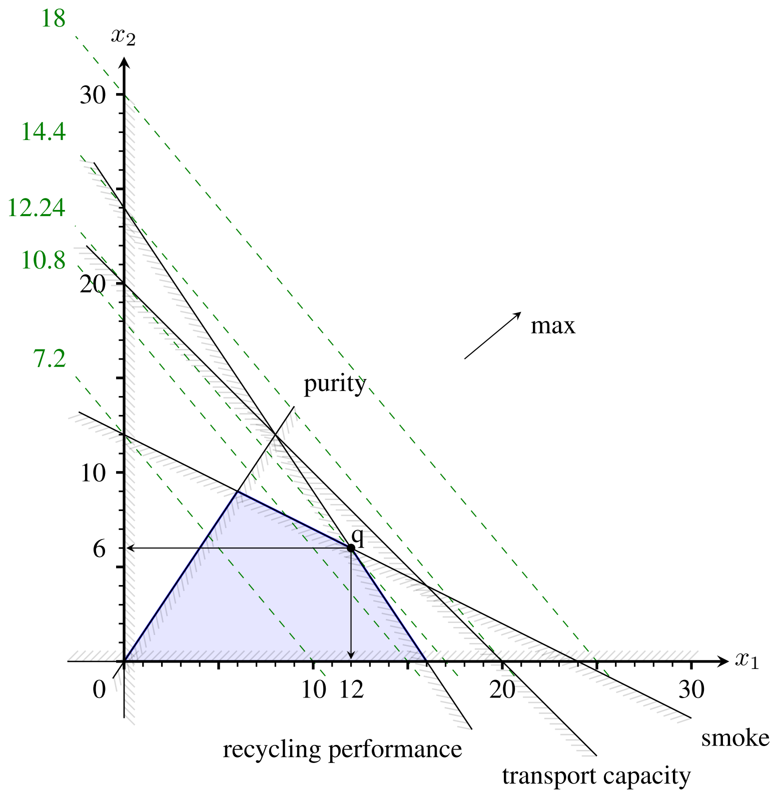


Figure 1.2: Graphical solution method. The green dashed lines show level curves, which are perpendicular to the direction in which we maximize. Among all level curves touching the feasible region, the one corresponding to the largest objective value reveals the optimal value of the LP.

Problem (1.1) is not a linear program in its current form because the absolute value is not an affine function. However, it can be converted into one by introducing, for $i \in [k]$, a variable z_i that captures the ℓ_1 error $|y_i - a^\top x_i - \beta|$:

$$
\begin{array}{rrcrcrcll}
\min & \displaystyle\sum_{i=1}^{k} z_i & & & & & & & \\
 & y_i & - & a^\top x_i & - & \beta & \leq & z_i & \forall i \in [k] \\
 & -y_i & + & a^\top x_i & + & \beta & \leq & z_i & \forall i \in [k] \\
 & & & & & a & \in & \mathbb{R}^n & \\
 & & & & & \beta & \in & \mathbb{R} & \\
 & & & & & z & \in & \mathbb{R}^k \ . &
\end{array}
\tag{1.2}
$$

Notice that the constraints imposed on the variables z_i only require $z_i \geq |y_i - a^\top x_i - \beta|$ and not the ostensibly more natural $z_i = |y_i - a^\top x_i - \beta|$. The reason is that requiring $z_i = |y_i - a^\top x_i - \beta|$

would lead to a non-convex set of feasible solutions; however, linear programs always optimize over a polyhedron, which is convex. Nevertheless, because we minimize the sum of the z_i, the fact that the z_i will be at least as large as the absolute value is sufficient. They will automatically satisfy $z_i = |y_i - a^\top x_i - \beta|$ in an optimal solution to the LP, because setting them to a larger value would just lead to a strictly worse objective value. Hence, finding the optimal coefficients in linear ℓ_1 regression can be reduced to solving a linear program.

Figure 1.3 shows an example of linear ℓ_1 regression with unidimensional input points, i.e., $n = 1$, applied to a small random data set.

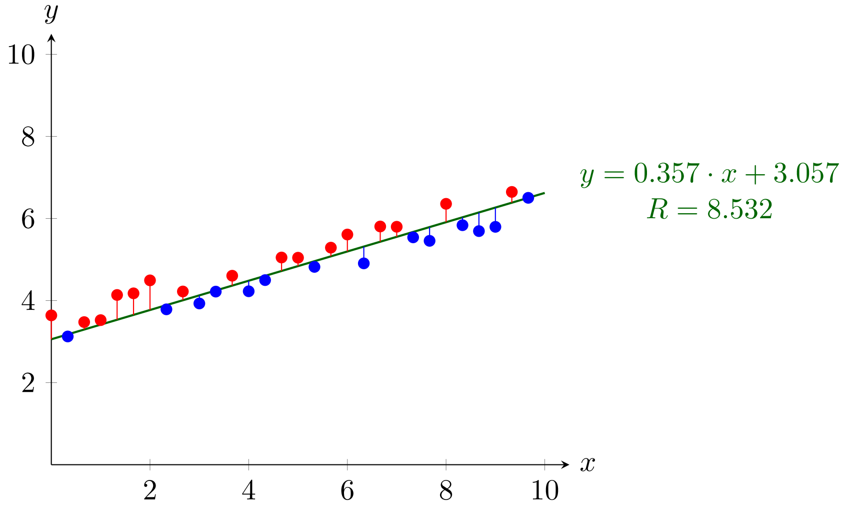


Figure 1.3: Example with 30 pairs $(x_i, y_i) \in \mathbb{R}^2$. The green line is the optimal linear ℓ_1 regression computed by solving the corresponding linear program (1.2). The sum of the vertical distances between the points and the line is the total error R that got minimized.

Resource allocation

Resource allocation is one of the classical problem settings in Operations Research. The goal is to distribute scarce resources among different activities, commonly with the objective of maximizing profit.

To exemplify this problem class, consider a fictitious Italian coffee-shop that wants to edge the competition by brewing its own special coffee types. They are doing this by combining different sorts of coffee beans into a single type. Moreover, they know how many beans of which sort they need to produce one liter of a certain coffee type. Now they need to figure out how much to produce of each coffee type (resource allocation) to maximize profit. Assume that they have fixed amounts of coffee beans of each sort (scarce resource) to be used for the next brewing cycle. Moreover, the coffee types need very different compositions of beans and each type gives a different profit per liter. This is shown in Table 1.1, where we assume that there are four bean sorts (Liberica, Excelsa, Arabica, and Robusta), and three coffee types (Prego, Barzo, Pronto).

Bean sort	Available amount	Prego	Barzo	Pronto
Liberica	300 000	100	100	300
Excelsa	350 000	200	300	100
Arabica	250 000	100	100	250
Robusta	500 000	300	200	200
profit/liter	-	10	10	20

Table 1.1: Beans needed per liter of each coffee type.

To approach this problem with linear programming, we first introduce a variable for each coffee type, capturing the amount of coffee to be brewed of the corresponding type:

$$c_j := \text{production quantity of coffee type } j \text{ (in liters)} \qquad j \in \{1,2,3\} \ ,$$

where $j = 1$ corresponds to coffee type Prego, $j = 2$ to Barzo, and $j = 3$ to Pronto.

Now we introduce one constraint per bean type to ensure that the total number of beans needed of each type is not exceeded by the production plan.

$$\begin{array}{lrcrcrcr}
\text{Liberica:} & 100c_1 & + & 100c_2 & + & 300c_3 & \leq & 300\,000 \\
\text{Excelsa:} & 200c_1 & + & 300c_2 & + & 100c_3 & \leq & 350\,000 \\
\text{Arabica:} & 100c_1 & + & 100c_2 & + & 250c_3 & \leq & 250\,000 \\
\text{Robusta:} & 300c_1 & + & 200c_2 & + & 200c_3 & \leq & 500\,000
\end{array}$$

Finally, we have non-negativity constraints:

$$c_j \geq 0 \quad \forall j \in \{1,2,3\} \ .$$

The objective is to maximize the profit:

$$\max \ 10c_1 + 10c_2 + 20c_3 \ .$$

Hence, this completes the description of the problem in form of a linear program. Notice that, to keep things as simple as possible, we assumed that unused beans do not have any value. However, we could also assume that there is a salvage value per bean type for unused beans, and this could still be modeled as a linear program.

Project scheduling

Consider a project comprising of a finite set of n tasks, where task i needs t_i time for completion. Additionally, some tasks depend on others and cannot be started before all the prerequisite tasks are finished. Our goal is to minimize the makespan, i.e., the total time required to finish all tasks and, thus, the whole project. Here we assume that any number of tasks can be processed simultaneously without compromising performance or quality.

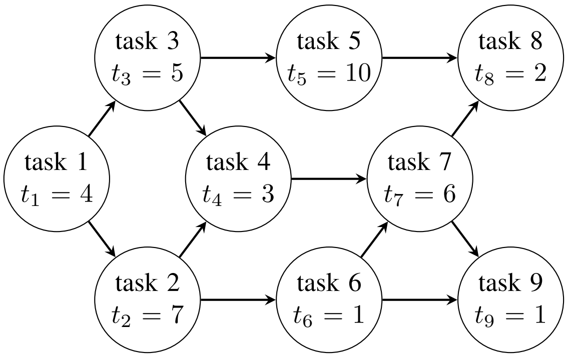


Figure 1.4: Example project scheduling problem.

Figure 1.4 shows an example project with 9 tasks and the dependencies between the tasks. For example, the arrow from task 3 to task 4 indicates that task 4 can only be started once task 3 has been completed.

We now show how this project scheduling question can be modeled by a linear program. Assuming that work on the projects starts at time 0, denote by s_v the starting time of task v. Moreover, let z be a variable we introduce to denote the completion time of the whole project, which is the completion time of the last task. We set up the following linear program.

$$\begin{array}{rrcrcll} \min & z & & & & & \\ & z & \geq & s_v & + & t_v & \text{for every task } v \\ & s_v & \geq & s_u & + & t_u & \text{for every dependency } u \text{ of } v \\ & s_v & \geq & 0 & & & \text{for every task } v \\ & z & \geq & 0 & & & \end{array}$$

Let us check that solving (1.1.2) indeed yields an optimal task schedule (minimizing the makespan). On the one hand, the optimal solution of the LP is a scheduling of the tasks satisfying the precedence constraints, so the optimal value is greater or equal to the minimal makespan. On the other hand, every optimal schedule satisfies the constraints of the LP, hence it is a feasible solution, and thus the optimal objective value is less or equal to the minimal makespan. Therefore, the optimal solution of the LP is precisely the minimal makespan of the whole project. Note that by backtracking from the end of the project and following tight constraints, one can construct the "bottleneck" chain of tasks that determine the project completion time (see Figure 1.5). This reasoning leads to the following interesting property: The minimum makespan is the same as the length of a longest path in the task graph, as shown in Figure 1.4, where the length of a path is the sum of the processing times of the tasks on the path.

Shortest s-t path

The problem of finding a shortest path from one site to another frequently occurs in various contexts, such as planning the quickest route from one physical location to another (from point A to point B), finding the shortest chain of social connections between two people to test the six

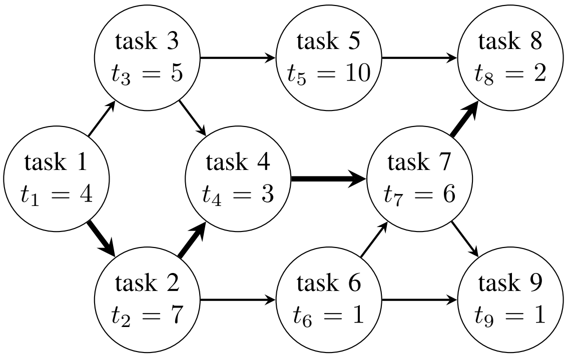


Figure 1.5: The highlighted path shows a bottleneck chain of tasks. This is a longest path in the above graph, where the length of the path is measured by summing up the processing times of the tasks on the path. Hence, in the shown example, the highlighted path has total length 22, which is the optimal makespan.

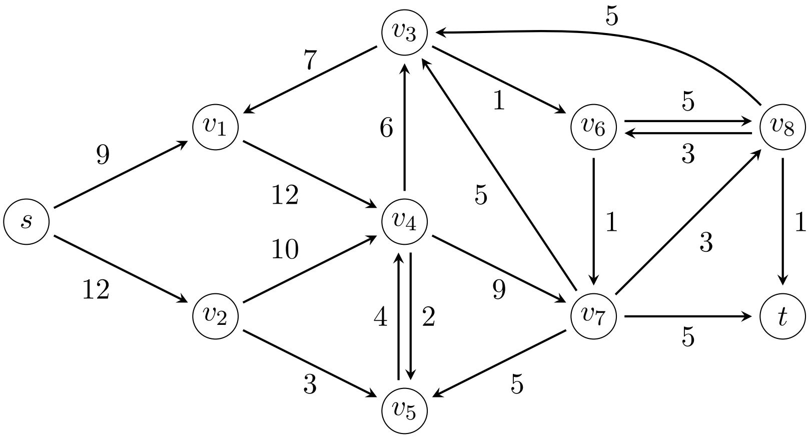


Figure 1.6: Example of finding a shortest s-t path.

degrees of separation hypothesis, or solving the Rubik's Cube in the least number of moves. In this section, we show that the shortest path problem can be reformulated in terms of an LP.

To formalize the setting of the shortest path problem, we are given a graph as shown in Figure 1.6, where a finite number of nodes is connected by arcs. Each arc a has a non-negative length $\ell(a) \in \mathbb{R}_{\geq 0}$ assigned to it; these are the numbers written next to the arcs in Figure 1.6. Moreover, we have a designated start node s and end node t. The goal is to find a shortest way to go from s to t, where we can only follow arcs from tail to head. (We formally introduce graph theoretic notions and terminology later in this course.)

To reformulate the shortest s-t path problem as an LP, we introduce a non-negative variable d_v for each node v with which we want to capture the distance from s to v. In particular, the distance from s to itself is zero, i.e., we can set $d_s = 0$. Another example distance, for the shown problem, is the distance from s to v_5, which is 15, because the shortest path from s to v_5 is $s \to v_2 \to v_5$ of length 15. Notice that the following is a valid constraint for the distances: for any arc a from u to v, the distance from s to v is no greater than the distance from s to u plus the length of the arc a. Indeed, to reach u from s, one option is to first visit v and then go from v to u over the arc a. Due to this, we will impose the constraints

$$d_v \leq d_u + \ell(a) \quad \text{for every arc } a \text{ from } u \text{ to } v,$$

on the variables d_v. In turns out that these constraints ensure that any d_v is never strictly larger then the distance from s to v. However, the discussed constraints allow for choosing distances that are significantly shorter. For example, $d_v = 0$ for every node v is a feasible solution. Hence, our linear program will actually not minimize d_t, but maximize it, which might sound counter-intuitive at first sight. In summary, below is the LP formulation we are using.

$$\begin{array}{llcll} \max & d_t & & & \\ & d_s & = & 0 & \\ & d_v & \leq & d_u + \ell(a) & \text{for every arc } a \text{ from } u \text{ to } v \\ & d_v & \geq & 0 & \text{for every node } v \end{array}$$

Let us justify why the optimal value of the above LP is indeed the shortest path distance from s to t. First, for any path P, we can sum up all the constraints corresponding to arcs on the path to obtain the inequality $d_t \leq d_s + \ell(P) = \ell(P)$. Thus the maximum value of d_t we obtain through the LP is not greater than the shortest path length. Moreover, by setting d_v, for each node v, to the actual distances from s to v, all of our constraints are fulfilled. Hence, d_t is not smaller than the shortest path length from s to t, implying that the optimal LP value is indeed the distance from s to t.

Additionally, a shortest s-t path can be inferred from the optimal solution of the LP: starting from t, on each step choose an arc to the current node for which the respective constraint is tight and transition to the beginning of that arc. Figure 1.7 indicates in green next to each node v the value d_v. This is an optimal solution to the above LP. (Notice there are further optimal solutions; for example one can exchange $d_{v_1} = 9$ by $d_{v_1} = 8$. In particular, this shows that there may be vertices $v \in V \setminus \{s, t\}$ for which, even in an optimal LP solution, d_v is strictly smaller than the s-v distance.)

1.2 Polyhedra and basic convex geometry

As discussed, linear programming is about maximizing or minimizing a linear (or affine) function over a polyhedron. Not surprisingly, the study of polyhedra is key in understanding and solving linear programs. Moreover, as we will see later, polyhedra also play a central role in Combinatorial Optimization.

1.2.1 Basic notions

We start with some basic notions and terminology.

Definition 1.4: Half-space & hyperplane

A *half-space* in $\mathbb{R}^n$ is a set of the form $\{x \in \mathbb{R}^n : a^\top x \leq \beta\}$ for $a \in \mathbb{R}^n \setminus \{0\}$ and $\beta \in \mathbb{R}$. Moreover, $\{x \in \mathbb{R}^n : a^\top x = \beta\}$ is called a *hyperplane*.

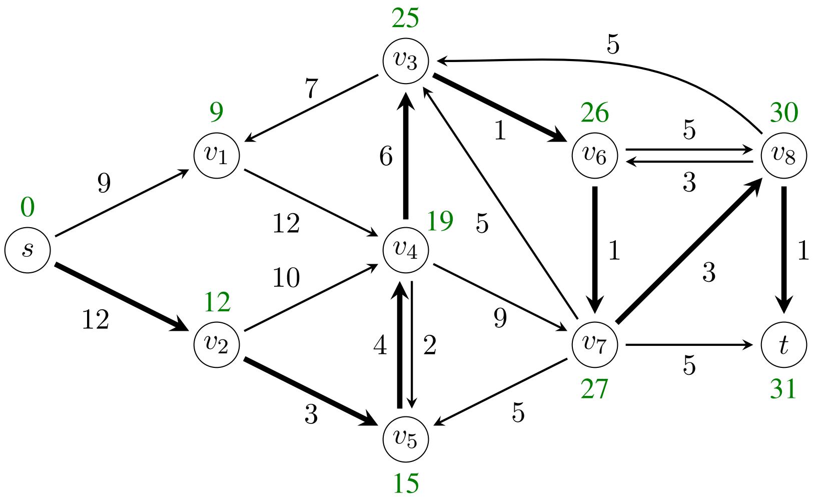


Figure 1.7: The green numbers show the d_v-values of one optimal LP solution. A shortest s-t path can be found by starting at t and backtracking along arcs that correspond to tight LP constraints.

Definition 1.5: Polyhedron & polytope

A *polyhedron* $P \subseteq \mathbb{R}^n$ is a finite intersection of half-spaces. Moreover, a bounded polyhedron is called a *polytope*.

An immediate consequence is that a polyhedron can be described by a finite set of linear inequalities, which is generally known as an *inequality description*. In other words, for a polyhedron $P \subseteq \mathbb{R}^n$, we can always write $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ for some $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ with $m \in \mathbb{Z}_{\geq 0}$. Even though, as we already discussed in the context of linear programs, we can always describe P in terms of less-than-or-equal inequalities, we allow an inequality description of a polyhedron to have inequalities of both types and also equality constraints. Clearly, there are many equivalent inequality descriptions for the same polyhedron.

Notice that polyhedra are convex sets.[1] This follows immediately from the fact that half-spaces are convex and the intersection of any family of convex sets is convex.

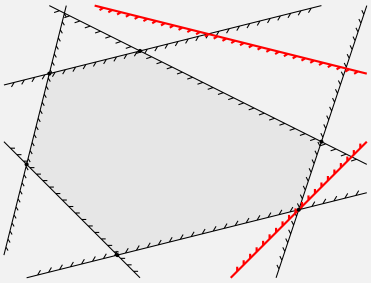
Definition 1.6: Redundancy

A linear inequality or equality of an inequality description of a polyhedron is called *redundant* if removing it from the description does not change the polyhedron.

[1] We recall that a set $X \subseteq \mathbb{R}^n$ is convex if, for any $x_1, x_2 \in X$ and $\lambda \in [0,1]$, we have $\lambda x_1 + (1-\lambda)x_2 \in X$.

Example 1.7: Redundant constraints

The picture below shows a polytope with two redundant constraints, highlighted in red.



Definition 1.8: Dimension of a polyhedron

The dimension $\dim(P)$ of a polyhedron $P \subseteq \mathbb{R}^n$ is the dimension of a smallest-dimensional affine subspace containing P, i.e.,

$$\dim(P) := \min\{k \in \mathbb{Z}_{\geq 0} \colon \exists A \in \mathbb{R}^{n \times n} \text{ with } \operatorname{rank}(A) = n - k \text{ \& } Ax = Ay \; \forall x, y \in P\} \ .$$

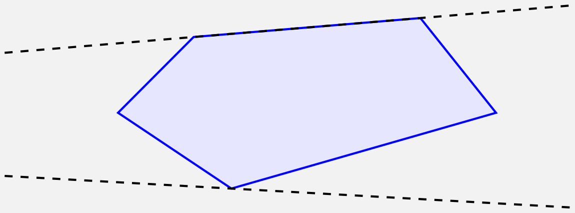
In particular, P is called *full-dimensional* if $\dim(P) = n$.

Definition 1.9: Supporting hyperplane

Let $P \subseteq \mathbb{R}^n$ be a polyhedron. A hyperplane $H = \{x \in \mathbb{R}^n \colon a^\top x = \beta\}$ is called P-*supporting*—or simply *supporting*, if P is clear from context—if $P \cap H \neq \emptyset$ and P is contained in one of the two half-spaces defined by H, i.e., either $P \subseteq \{x \in \mathbb{R}^n \colon a^\top x \leq \beta\}$ or $P \subseteq \{x \in \mathbb{R}^n \colon a^\top x \geq \beta\}$.

Example 1.10

The figure below shows a 2-dimensional polytope with two supporting hyperplanes.



Definition 1.11: Face, vertex, edge, and facet

Let $P \subseteq \mathbb{R}^n$ be a non-empty polyhedron.

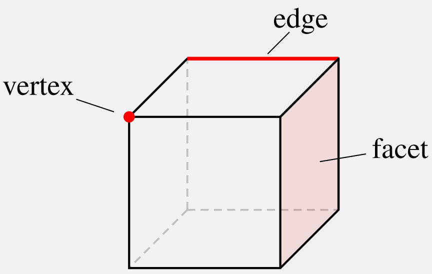
(i) A *face* of P is either P itself or the intersection of P with a supporting hyperplane.
(ii) A *vertex* of P is a 0-dimensional face of P.
(iii) An *edge* of P is a 1-dimensional face of P.
(iv) A *facet* of P is a $(\dim(P) - 1)$-dimensional face of P.

The empty polyhedron has only one face, which is the empty set. We denote by $\mathrm{vertices}(P)$ the set of all vertices of P.

Notice that a face of a polyhedron P is also a polyhedron, because it is the intersection of P with a hyperplane; indeed, this follows because a hyperplane is the intersection of two half-spaces and a polyhedron is by definition a finite intersection of half-spaces. Thus, when talking about the dimension of a face, we refer to the notion of dimension that we introduced for polyhedra.

Example 1.12

Below is a cube with three of its faces highlighted in red: a vertex, an edge, and a facet.



Note that vertices are the smallest (inclusion-wise and in terms of dimension) possible faces of a non-empty polyhedron P, and edges are the next-smallest faces P can have. On the other end, P is its largest face, and facets are the next-largest faces of P. A non-empty polyhedron does not need to have any vertices or edges; for example, a hyperplane in $\mathbb{R}^n$ for $n \geq 3$ is a polyhedron without vertices or edges. Moreover, the empty polyhedron, the whole space $\mathbb{R}^n$, as well as polytopes consisting of a single point, do not have any facets. However, all other polyhedra do have facets.

Proposition 1.13

Let $P = \{x \in \mathbb{R}^n \colon Ax \leq b\} \subseteq \mathbb{R}^n$ be a non-empty polyhedron with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, and let $F \subseteq P$. Then the following statements are equivalent.

(i) F is a face of P.
(ii) $\exists c \in \mathbb{R}^n$ such that $\delta := \max\{c^\top x \colon x \in P\}$ is finite and $F = \{x \in P \colon c^\top x = \delta\}$.
(iii) $F = \{x \in P \colon \overline{A}x = \overline{b}\} \neq \emptyset$ for a subsystem $\overline{A}x \leq \overline{b}$ of $Ax \leq b$.

Proof. Points (i) and (ii) are clearly equivalent by the definition of a face. We complete the proof by showing the equivalence between (iii) and (ii).

(iii) ⇒ (ii): Let $c = \overline{A}^\top \mathbf{1}$, where $\mathbf{1} \in \mathbb{R}^m$ denotes the all-ones vector. We show that this vector c fulfills the condition stated in (ii). Indeed, for any $x \in P$ we have

$$c^\top x = \mathbf{1}^\top \overline{A} x \leq \mathbf{1}^\top \overline{b} \ ,$$

with equality if and only if $\overline{A}x = \overline{b}$. Hence, for $\delta := \mathbf{1}^\top \overline{b} = \max\{c^\top x \colon x \in P\}$, we have

$$\{x \in P \colon c^\top x = \delta\} = \{x \in P \colon \overline{A}x = \overline{b}\} = F \ ,$$

as desired.

(ii) ⇒ (iii): Let $\overline{A}x \leq \overline{b}$ be the subsystem of all constraints of $Ax \leq b$ that are tight for all points in F. We will show that these constraints satisfy $F = \{x \in P \colon \overline{A}x = \overline{b}\}$, thus implying (iii).

Let $\widetilde{A}x \leq \widetilde{b}$ be all other constraints of $Ax \leq b$. Hence, for each constraint $\widetilde{a}^\top x \leq \widetilde{\beta}$ of $\widetilde{A}x \leq \widetilde{b}$, there is a point $x_{\widetilde{a}} \in F$ with $\widetilde{a}^\top x_{\widetilde{a}} < \beta$. By summing up all of these points $x_{\widetilde{a}}$ for all rows $\widetilde{a}$ of $\widetilde{A}$ and dividing by the number of rows of $\widetilde{A}$, a point $z \in \mathbb{R}^n$ is obtained such that

(i) $z \in F$ and hence $c^\top z = \delta$, because z is a convex combination of the $x_{\widetilde{a}}$, and
(ii) $\widetilde{A}z < \widetilde{b}$ component-wise.

We complete the proof by showing that any point $y \in P \setminus F$ fulfills $\overline{A}y \neq \overline{b}$. Because $y \in P \setminus F$, we have

$$c^\top y < \delta = c^\top z \ . \tag{1.3}$$

We now define for any $\varepsilon > 0$ the point

$$x_\varepsilon := z + \varepsilon \cdot (z - y) \ .$$

Notice that for any $\varepsilon > 0$ we have

$$c^\top x_\varepsilon > \delta$$

because of (1.3). Hence, $x_\varepsilon \notin P$ as $\delta = \max\{c^\top x \colon x \in P\}$. Thus, x_ε violates one of the constraints of $Ax \leq b$, no matter how small $\varepsilon > 0$ is. For small enough $\varepsilon > 0$, only constraints of $\overline{A}x \leq \overline{b}$ can be violated, because no other constraint of $Ax \leq b$ is tight at z. Hence, there is some constraint $\overline{a}^\top x \leq \overline{\beta}$ of $\overline{A}x \leq \overline{b}$ such that $\overline{a}^\top x_\varepsilon > \overline{\beta}$ for any $\varepsilon > 0$, which, by the definition of x_ε and the fact that $\overline{a}^\top z = \overline{\beta}$, implies $\overline{a}^\top y < \overline{\beta}$. Thus, $\overline{A}y \neq \overline{b}$, as desired. □

Proposition 1.13 reveals a basic property of faces, namely that the face relationship is transitive.

Corollary 1.14

Let P be a polyhedron. Then a face of a face of P is itself a face of P.

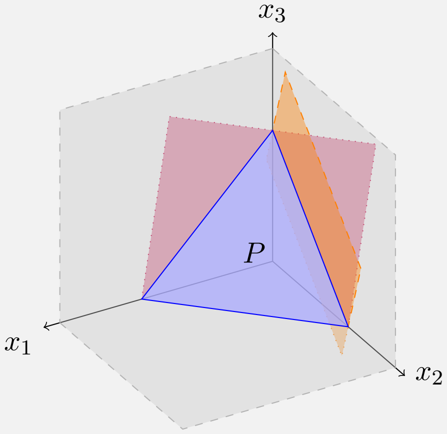
Proof. This follows immediately from point (iii) of Proposition 1.13. □

Definition 1.15: Facet-defining inequality

Let $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$ be a polyhedron. An inequality $a^\top x \leq \beta$ of the system $Ax \leq b$ is *facet-defining* if $F := P \cap \{x \in \mathbb{R}^n \colon a^\top x = \beta\}$ is a facet of P. We also say that $a^\top x \leq \beta$ is a constraint *defining* the facet F.

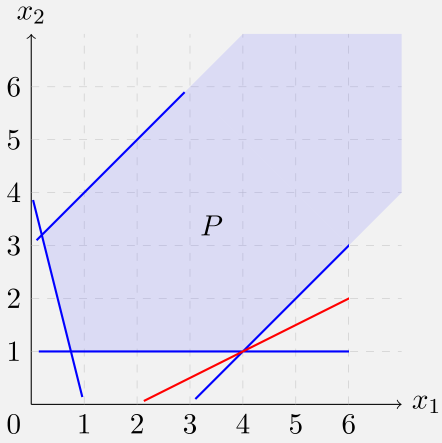
Example 1.16: Facet-defining inequalities

The example below shows a polytope $P \subseteq \mathbb{R}^3$ with $\dim(P) = 2$. The two red constraints imply an equality constraint and are responsible for P not being full-dimensional. Notice that neither of these two constraints is facet-defining even though they are both not redundant. Moreover, the orange constraint and the non-negativity constraint $x_1 \geq 0$ (highlighted in gray) both define the same facet of P.



$$P = \left\{ \begin{array}{rcrcrcl} x_1 & + & x_2 & + & x_3 & \leq & 2 \\ x_1 & + & x_2 & + & x_3 & \geq & 2 \\ x_1 & & & & & \geq & 0 \\ & & x_2 & & & \geq & 0 \\ & & & & x_3 & \geq & 0 \\ -4x_1 & + & x_2 & + & x_3 & \leq & 2 \end{array} \right\}$$

In the 2-dimensional example shown below, all constraints except for the red one are facet-defining. Whenever P is full-dimensional, every non-facet-defining constraint is redundant. However, as highlighted by the example above, this does not hold for lower-dimensional polyhedra.



$$P = \left\{ \begin{array}{rcrcr} x_1 & - & x_2 & \geq & -3 \\ 4x_1 & + & x_2 & \geq & 4 \\ & & x_2 & \geq & 1 \\ x_1 & - & 2x_2 & \leq & 2 \\ x_1 & - & x_2 & \leq & 3 \end{array} \right\}$$

Proposition 1.17

Let $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$ be a polyhedron. For any facet $F \subseteq P$ of P, there is at least one inequality $a^\top x \leq \beta$ of $Ax \leq b$ that defines F.

Proof. Let $F \subseteq P$ be a facet of P. By Proposition 1.13, F can be written as

$$F = \left\{x \in P \colon \overline{A}x = \overline{b}\right\} \quad , \tag{1.4}$$

where $\overline{A}x = \overline{b}$ is a subsystem of $Ax \leq b$. Because F is a facet of P we have $\dim(F) = \dim(P) - 1$. Hence, in particular $F \neq P$, which implies that there is an equality $\overline{a}^\top x = \overline{\beta}$ among $\overline{A}x = \overline{b}$ such that $P \not\subseteq \{x \in \mathbb{R}^n \colon \overline{a}^\top x = \overline{\beta}\}$. Hence

$$\dim(\{x \in P \colon \overline{a}^\top x = \overline{\beta}\}) < \dim(P) \quad . \tag{1.5}$$

We finish the proof by observing in the following that

$$F = \{x \in P \colon \overline{a}^\top x = \overline{\beta}\} \quad .$$

Notice that (1.4) implies

$$F \subseteq \{x \in P : \overline{a}^\top x = \overline{\beta}\} \quad . \tag{1.6}$$

Furthermore, it is impossible that there is a point $y \in \{x \in P \colon \overline{a}^\top x = \overline{\beta}\} \setminus F$, as this would imply $\dim(\{x \in P \colon \overline{a}^\top x = \overline{\beta}\}) > \dim(F)$, which, together with (1.5) implies $\dim(F) \leq \dim(P) - 2$, thus contradicting $\dim(F) = \dim(P) - 1$. □

Definition 1.18: Extreme point

Let P be a polyhedron. A point $y \in P$ is an *extreme point* of P if it is not the midpoint of two distinct points of P, i.e., there are no $p_1, p_2 \in P$ with $p_1 \neq p_2$ and $y = \frac{1}{2}(p_1 + p_2)$.

Note that one can equivalently define an extreme point $y \in P$ of a polyhedron P by requiring that y is not a *non-trivial convex combination* of two distinct points $p_1, p_2 \in P$, i.e., we cannot write $y = \lambda p_1 + (1 - \lambda)p_2$ with $\lambda \in (0, 1)$. This follows by observing that if a point in a convex set C is a non-trivial convex combination of two distinct points in C, then it is also the midpoint of two distinct points in C. In short, this modified definition as well as the one stated in Definition 1.18 state that for y to be an extreme point in P, it should not be in the interior of a segment that lies in P.

Proposition 1.19

Let $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$ be a polyhedron and $y \in P$. Then the following statements are equivalent.

(i) y is a vertex of P.
(ii) y is the unique solution to $\overline{A}x = \overline{b}$, where $\overline{A}x \leq \overline{b}$ is the subsystem of $Ax \leq b$ containing all y-tight constraints.
(iii) y is an extreme point of P.

Proof. We show the statement by showing (i) ⇒ (iii) ⇒ (ii) ⇒ (i).

(i) ⇒ (iii): We show this statement by showing that if y is not an extreme point of P, then it is also not a vertex of P. The point y not being an extreme point implies that there are two distinct points $p_1, p_2 \in P$ with $y = \frac{1}{2}p_1 + \frac{1}{2}p_2$. For y to be a vertex, there needs to be a vector $c \in \mathbb{R}^n$ such that y is the unique maximizer of $\max\{c^\top x \colon x \in P\}$. However, this is impossible no matter how c is chosen, because $c^\top y = \frac{1}{2}c^\top p_1 + \frac{1}{2}c^\top p_2$. Hence, at least one of $c^\top p_1$ and $c^\top p_2$ is no smaller than y. Thus, y is not the unique maximizer of $\max\{c^\top x \colon x \in P\}$.

(iii) ⇒ (ii): We show this statement by showing the contraposition, i.e., we assume that y is not the unique solution to $\overline{A}x = \overline{b}$, where $\overline{A}x \leq \overline{b}$ is the subsystem of $Ax \leq b$ containing all y-tight constraints, and show that y is not an extreme point. Indeed, if y is not the unique solution of $\overline{A}x = \overline{b}$, then the dimension of $\{x \in \mathbb{R}^n \colon \overline{A}x = \overline{b}\}$ is at least one; hence, there is a non-zero vector $v \in \mathbb{R}^n$ such that $\overline{A}v = 0$. Moreover, by choosing $\varepsilon > 0$ small enough we have $y + \varepsilon v \in P$ and $y - \varepsilon v \in P$, because all constraints of $Ax \leq b$ except for those corresponding to $\overline{A}x \leq \overline{b}$ are not y-tight, and $y \pm \varepsilon v$ does not violate any y-tight constraints because $\overline{A}(y \pm \varepsilon v) = \overline{A}y = \overline{b}$. However, this implies that y is the midpoint of $y + \varepsilon v$ and $y - \varepsilon v$, and thus, y is not an extreme point of P.

(ii) ⇒ (i): If y is the unique solution to $\overline{A}x = \overline{b}$, where $\overline{A}x \leq \overline{b}$ is the subsystem of $Ax \leq b$ containing all constraints that are y-tight, then

$$\{y\} = \{x \in P \colon \overline{A}x = \overline{b}\} \ .$$

By Proposition 1.13, $\{y\}$ is therefore a face of P and, because it is 0-dimensional, the point y is a vertex of P. □

Even though we focus here on polyhedra, we want to highlight that some natural results like Proposition 1.19 may not hold anymore when extending the involved notions to closed convex sets. For example, when defining extreme points and vertices for closed convex sets analogously to the polyhedral case, then an extreme point of a closed convex set does not need to be a vertex of it. See Figure 1.8 for an illustration.

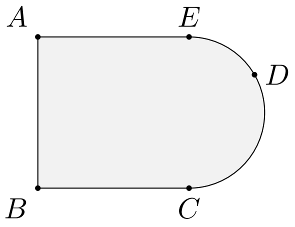


Figure 1.8: A closed and bounded convex set with an infinite number of extreme points and vertices, due to the curved boundary on the right of the object, which forms a half-circle. All of the five named points are extreme points. However, among these points, only A, B, and D are vertices. The points C and E are extreme points but not vertices because there is no supporting hyperplane that touches the set only at C or E, respectively.

Lemma 1.20

Let $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$ be a full-dimensional polyhedron, then each inequality $a^\top x \leq \beta$ of $Ax \leq b$ that is not facet-defining for P is redundant.

Proof. We prove the contraposition. Hence, assume that $a^\top x \leq \beta$ is not redundant, and we will show that it is facet-defining for P. Let $\overline{A}x \leq \overline{b}$ be all constraints of $Ax \leq b$ except for $a^\top x \leq \beta$. We define

$$Q \coloneqq \left\{x \in \mathbb{R}^n \colon \overline{A}x \leq \overline{b}\right\} \enspace .$$

Because $a^\top x \leq \beta$ is not redundant, there is a point $y \in Q \setminus P$, and hence $a^\top y > \beta$. Moreover, let $z \in P$ be a point in the interior of P, which exists because P is full-dimensional. Consequently, there is some $\varepsilon > 0$ such that the ball $B(z, \varepsilon)$ around z of radius ε, i.e.,

$$B(z, \varepsilon) = \{x \in \mathbb{R}^n \colon \|x - z\|_2 \leq \varepsilon\} \enspace ,$$

is contained in P, i.e., $B(z, \varepsilon) \subseteq P$. Notice that $a^\top z < \beta$, because z is in the interior of P. Furthermore, as $B(z, \varepsilon) \subseteq P \subseteq Q$ and $y \in Q$, we have that the set

$$C \coloneqq \operatorname{conv}(B(z, \varepsilon) \cup \{y\})$$

satisfies

$$C \subseteq Q \enspace .$$

Because C is full-dimensional and contains points in the interior of each of the sides of the hyperplane defined by $a^\top x = \beta$, namely the points y and z, we have that

$$C \cap \{x \in \mathbb{R}^n \colon a^\top x = \beta\}$$

is $(n-1)$-dimensional. Finally, $C \cap \{x \in \mathbb{R}^n \colon a^\top x = \beta\} \subseteq P$ because $C \subseteq Q$. Hence, $\{x \in P \colon a^\top x = \beta\}$ is $(n-1)$-dimensional, which implies that $a^\top x \leq \beta$ is facet-defining for P. □

Corollary 1.21

Let $P \subseteq \mathbb{R}^n$ be a polyhedron, and let f be the number of facets of P. Then, every inequality description of P requires at least f inequalities. Moreover, if P is full-dimensional, then an inequality description of P with f inequalities exists.

Proof. By Proposition 1.17, any inequality description of P needs at least one inequality per facet of P. It remains to show that there is an inequality description of P with facet-many inequalities if P is full-dimensional.

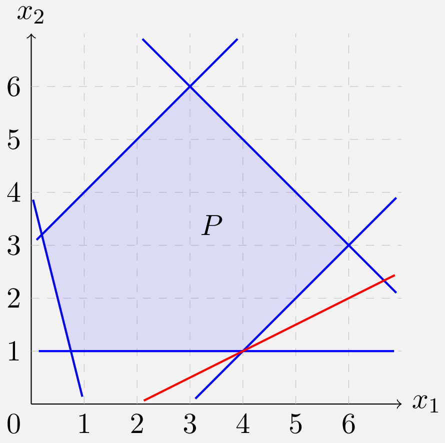
To this end, we start with an arbitrary inequality description $P = \{x \in \mathbb{R}^n \colon \overline{A}x \leq \overline{b}\}$ of P, and successively remove redundant constraints. We will be left with only facet-defining constraints due to Lemma 1.20. Let $Ax \leq b$ be the subsystem of $\overline{A}x \leq \overline{b}$ of remaining constraints after all redundant constraints have been successively removed from $\overline{A}x \leq \overline{b}$. Because only redundant constraints have been successively removed, we have $P = \{x \in \mathbb{R} \colon Ax \leq b\}$. We claim that this description has facet-many inequalities. Because each inequality of $Ax \leq b$ is facet-defining, it remains to show that no two different inequalities $a_1^\top x \leq \beta_1$ and $a_2^\top x \leq \beta_2$ of $Ax \leq b$ define the same facet $F \subseteq P$ of P. Assume for sake of contradiction that this is not the case, i.e., $F = \{x \in P \colon a_1^\top x = \beta_1\} = \{x \in P \colon a_2^\top x = \beta_2\}$. Now, because $P \subseteq \mathbb{R}^n$ is full-dimensional, there are n affinely independent points in F. As the hyperplanes $\{x \in \mathbb{R}^n \colon a_i^\top x = \beta_i\}$ for $i \in [2] := \{1, 2\}$ both contain these n affinely independent points, and must be such that $P \subseteq \{x \in \mathbb{R}^n \colon a_i^\top x \leq \beta_i\}$, the half-spaces $\{x \in \mathbb{R}^n \colon a_i^\top x \leq b_i\}$ are the same for $i \in [2]$. This contradicts the fact that we removed successively all redundant constraints, because the two constraints $a_i^\top x \leq \beta_i$ for $i \in [2]$ are redundant. (Even though this is not relevant for this proof, notice that after removing one of them, the other one may not be redundant anymore.) □

Definition 1.22: Degeneracy with respect to linear inequality descriptions

Consider a linear inequality description of a polyhedron $P \subseteq \mathbb{R}^n$. A vertex $y \in \mathrm{vertices}(P)$ is called *degenerate* (w.r.t. the linear inequality description) if the number of y-tight constraints in the linear inequality description is strictly larger than n. The inequality description is called *degenerate* if there is a degenerate vertex with respect to it.

Example 1.23: Degenerate vertex & degenerate inequality description

Below is a 2-dimensional polytope with a corresponding inequality description that is degenerate, because the vertex $\binom{4}{1}$ is degenerate with respect to the given inequality description.



$$P = \left\{ \begin{array}{rcrcr} x_1 & - & x_2 & \geq & -3 \\ 4x_1 & + & x_2 & \geq & 4 \\ & & x_2 & \geq & 1 \\ x_1 & - & 2x_2 & \leq & 2 \\ x_1 & - & x_2 & \leq & 3 \\ x_1 & + & x_2 & \leq & 8 \end{array} \right\}$$

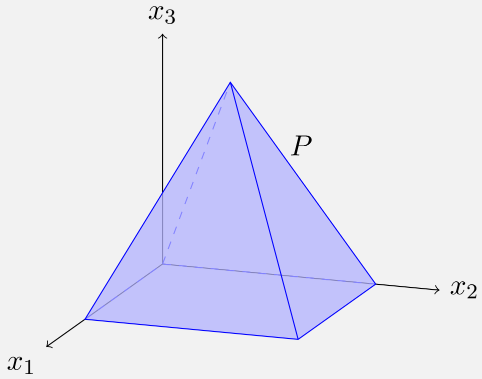
Definition 1.24: Degeneracy of a polyhedron

A polyhedron $P \subseteq \mathbb{R}^n$ is called *degenerate* if it has a vertex $y \in P$ contained on strictly more than $\dim(P)$ many facets.

Notice that even though the inequality description of the polytope in Example 1.23 is degenerate, the polytope itself is not degenerate.

Example 1.25: Degenerate polyhedron

Below is an example of a degenerate 3-dimensional polytope, namely a pyramid with a square base. It is degenerate because the apex of the pyramid is a vertex contained in 4 facets.



$$P = \left\{ \begin{array}{rrrcl} & & x_3 & \geq & 0 \\ 2x_1 & & -\ x_3 & \geq & 0 \\ 2x_1 & & +\ x_3 & \leq & 2 \\ & 2x_2 & -\ x_3 & \geq & 0 \\ & 2x_2 & +\ x_3 & \leq & 2 \end{array} \right\}$$

In particular, if a polyhedron $P \subseteq \mathbb{R}^n$ is degenerate, then any inequality description for it is degenerate too. This is a consequence of Proposition 1.17, which implies that for each facet there must be a facet-defining inequality.

Definition 1.26: Dominant

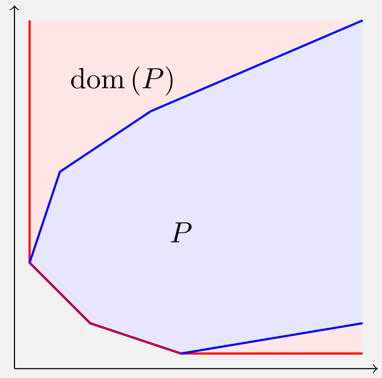
For a set $X \subseteq \mathbb{R}^n$, its dominant $\mathrm{dom}(X) \subseteq \mathbb{R}^n$ is defined by

$$\mathrm{dom}(X) := X + \mathbb{R}^n_{\geq 0} = \{x + y \colon x \in X, y \in \mathbb{R}^n_{\geq 0}\} \ .$$

For $X, Y \subseteq \mathbb{R}^n$, the set addition $X + Y := \{x + y \colon x \in X, y \in Y\}$, which we used above in the definition of the dominant, is called the *Minkowski sum*.

Example 1.27: Dominant

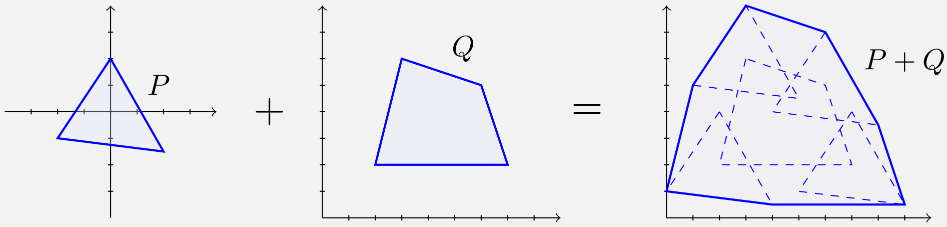
The figure below shows an unbounded polyhedron $P \subseteq \mathbb{R}^2$ together with its corresponding dominant $\mathrm{dom}(P)$.



Before moving over to the representation of polyhedra, we provide an example of a Minkowski sum, because this simple yet useful operation will be relevant also later on. Hence, it is helpful to develop some intuition for it.

Example 1.28: Minowski sum

The graphic below shows the Minkowski sum $P + Q$ of two 2-dimensional polytopes P and Q. The third pictures illustrates that $P + Q$ can be obtained by "shifting" one of the polytopes, say P, to each vertex of the other one—such that the origin in the figure showing P lies at the vertices of Q—and then taking the convex hull of the shifted versions of P. In particular, each vertex of $P + Q$ is the sum of a vertex of P and one of Q. However, not all sums of vertices in P and Q lead to a vertex of $P + Q$.



1.2.2 Representation of polyhedra

As mentioned, polyhedra are a basic family of convex sets. Not surprisingly, convexity is exploited in a multitude of results and algorithms linked to polyhedra. At the heart of convexity is the notion of a convex combination.

Definition 1.29: Convex combination

A *convex combination* of $x_1, \dots, x_k \in \mathbb{R}^n$ is a point described by $\sum_{i=1}^k \lambda_i x_i$, where $\lambda \in \mathbb{R}^k_{\geq 0}$ and $\sum_{i=1}^k \lambda_i = 1$.

In this course, convex combinations are always with respect to a finite number of elements. The fact that linear combinations with the above properties are called convex combinations is motivated by the observation that a set $X \subseteq \mathbb{R}^n$ is convex if and only if any convex combination of finitely many points in X lies in X. The convex combinations of an arbitrary point set are known as its convex hull.

Definition 1.30: Convex hull

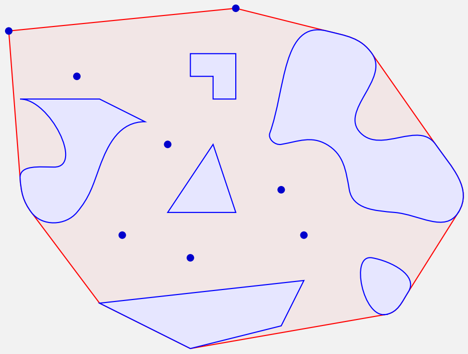
Let $X \subseteq \mathbb{R}^n$. The *convex hull* $\operatorname{conv}(X) \subseteq \mathbb{R}^n$ of X are all convex combinations of finitely many points in X, i.e.,

$$\operatorname{conv}(X) := \left\{ \sum_{i=1}^{k} \lambda_i x_i \;\middle|\; \begin{array}{l} k \in \mathbb{Z}_{\geq 1},\ x_1, \dots, x_k \in X, \text{ and} \\ \lambda_1, \dots, \lambda_k \in \mathbb{R}_{\geq 0} \text{ with } \sum_{i=1}^{k} \lambda_i = 1 \end{array} \right\} .$$

Hence, a set $X \subseteq \mathbb{R}^n$ is convex if and only if $\operatorname{conv}(X) = X$. This, together with the fact that the convex hull operator is monotone, i.e., $\operatorname{conv}(X) \subseteq \operatorname{conv}(Y)$ for any $X \subseteq Y \subseteq \mathbb{R}^n$, implies that $\operatorname{conv}(X)$ is the smallest convex set containing X.

Example 1.31: Convex hull

Example of the convex hull in 2 dimensions. The red area is the convex hull of the blue objects.



The following proposition highlights how polytopes are described by their vertices. The statement will be proven in the problem sets.

Proposition 1.32

A polytope is the convex hull of its vertices.

Moreover, as stated below, the convex hull of any finite point set is a polytope. Again, the statement will be shown in one of the problem sets.

Proposition 1.33

Let $X \subseteq \mathbb{R}^n$ be a finite set. Then $\operatorname{conv}(X)$ is a polytope.

To move from polytopes to polyhedra, we have to consider unbounded sets. One of the most basic building blocks when dealing with unbounded sets are cones, which we define below. They allow for obtaining a nice characterization of polyhedra that we mention later.

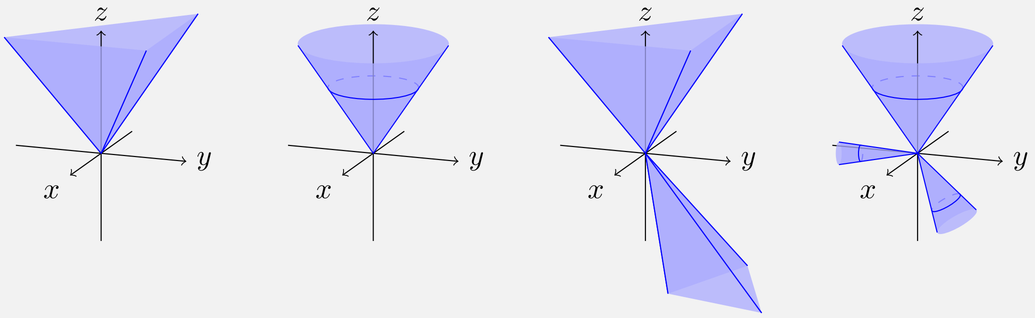
Definition 1.34: (Polyhedral) cone

A *cone* is a set $C \subseteq \mathbb{R}^n$ such that for any $x \in C$ and $\lambda \in \mathbb{R}_{\geq 0}$ we have $\lambda \cdot x \in C$. A cone that is a polyhedron is called a *polyhedral cone*.

We highlight that a cone does not need to be convex. Nevertheless, we will mostly be interested in convex cones. These can be characterized as the sets that are invariant under *conic combinations*, which are linear combinations with exclusively non-negative coefficients.

Example 1.35

Below are four examples of cones in $\mathbb{R}^3$. The first two of them are convex cones whereas the last two are not. Moreover, only the first one is a polyhedral cone. Notice that the third is not a polyhedral cone because it is not a polyhedron as it is not even convex.



Proposition 1.36

If $C \subseteq \mathbb{R}^n$ is a non-empty polyhedral cone, then

$$C = \{x \in \mathbb{R}^n \colon Ax \leq 0\} \ , \tag{1.7}$$

for some matrix $A \in \mathbb{R}^{m \times n}$, where $m \in \mathbb{Z}_{\geq 0}$. Vice-versa, any set C with a description as in (1.7) is a polyhedral cone.

Proof. Let $C \subseteq \mathbb{R}^n$ be a polyhedral cone. Because C is a polyhedron, we have

$$C = \{x \in \mathbb{R}^n \colon Ax \leq b\}$$

for some $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, and we assume without loss of generality that none of the constraints $Ax \leq b$ are redundant; for otherwise, we could first successively remove redundant constraints. We claim that $b = 0$. For otherwise, there is some constraint $a^\top x \leq \beta$ with $\beta \neq 0$ in the system $Ax \leq b$. Because $a^\top x \leq \beta$ is not redundant, there is a point $y \in C$ with $a^\top y = \beta$.

However, because C is a cone, we must have $\frac{1}{2}y, 2y \in C$; but one of these points violates the constraint $a^\top x \leq \beta$. (If $\beta > 0$, then $2y$ will violate it; otherwise $\frac{1}{2}y$ will violate the constraint.) Hence, $b = 0$, and we have $C = \{x \in \mathbb{R}^n \colon Ax \leq 0\}$ as desired.

Conversely, any set $C \subseteq \mathbb{R}^n$ with $C = \{x \in \mathbb{R}^n \colon Ax \leq 0\}$ for $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ is clearly a polyhedron and a cone, and thus a polyhedral cone. □

Another way to characterize polyhedral cones is by the fact that they have a finite set of generators, as formalized in the statement below. This will be shown in the problem sets.

Proposition 1.37

If $C \subseteq \mathbb{R}^n$ is a polyhedral cone, then

$$C = \left\{ \sum_{i=1}^{k} \lambda_i x_i \colon \lambda_i \geq 0 \ \forall i \in [k] \right\} \quad , \tag{1.8}$$

for some finite set of points $x_1, \dots, x_k \in \mathbb{R}^n$. The points $x_1, \dots, x_k$ are called a *set of generators* of C. Vice-versa, any set C as described in (1.8) is a polyhedral cone.

Polyhedral cones can be thought of as the most basic unbounded polyhedra. The statement below underlines this intuition by showing that any polyhedron can be written as a Minkowski sum of a polytope and a polyhedral cone. The proof of it will be covered in the problem sets.

Proposition 1.38

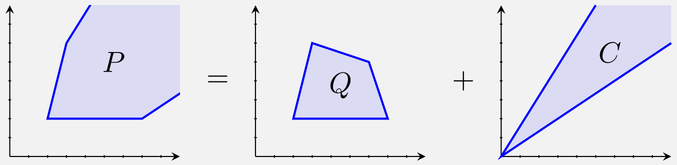
Let $P \subseteq \mathbb{R}^n$ be a polyhedron. Then

$$P = Q + C \quad ,$$

where $Q \subseteq \mathbb{R}^n$ is a polytope and $C \subseteq \mathbb{R}^n$ is a polyhedral cone. Vice-versa, the Minkowski sum of a polytope and a polyhedral cone is always a polyhedron.

Example 1.39

The graphic below shows an unbounded 2-dimensional polyhedron P and how it can be written as the Minkowski sum of a polytope Q and a polyhedral cone C.



The above characterizations of polytopes and polyhedra often allow for deriving short and elegant proofs of further important statements. One such statement, shown below, states that the family of polyhedra is invariant under affine transformations. This property is at the heart of various techniques in Combinatorial Optimization, in particular when a complex polytope, say one with a very large number of facets, is described as the affine projection of a much simpler one.

Proposition 1.40

An affine image of a polyhedron is a polyhedron, i.e., for any polyhedron $P \subseteq \mathbb{R}^n$ and any affine function $\varphi\colon \mathbb{R}^n \to \mathbb{R}^m$, the set $\varphi(P) := \{\varphi(x) \colon x \in P\}$ is a polyhedron.

Proof. We first observe that it suffices to prove the statement only for linear functions $\varphi\colon \mathbb{R}^n \to \mathbb{R}^m$ instead of affine ones. This follows from the fact that a translation of a polyhedron is again a polyhedron. Indeed, a general affine function $\varphi\colon \mathbb{R}^n \to \mathbb{R}^m$ can always be written as $\varphi(x) = \phi(x) + t$, where $\phi\colon \mathbb{R}^n \to \mathbb{R}^m$ is a linear function and $t \in \mathbb{R}^m$. Now assume that for any polyhedron $P \subseteq \mathbb{R}^n$, the image $\phi(P) \subseteq \mathbb{R}^m$ is again a polyhedron. We then have

$$\begin{aligned}\varphi(P) &= \phi(P) + t\\ &= \{x \in \mathbb{R}^m \colon Ax \leq b\} + t\\ &= \{x + t \colon x \in \mathbb{R}^m, Ax \leq b\}\\ &= \{y \in \mathbb{R}^m \colon A(y-t) \leq b\}\\ &= \{y \in \mathbb{R}^m \colon Ay \leq b + At\} \enspace ,\end{aligned} \tag{1.9}$$

where the second equality follows from the assumption that $\phi(P)$ is a polyhedron, and can thus be described by a linear inequality system $\phi(P) = \{x \in \mathbb{R}^m \colon Ax \leq b\}$, and the penultimate equality substitutes $x + t$ by y. Indeed, (1.9) shows that $\varphi(P)$ is a finite intersection of half-spaces and therefore a polyhedron, as desired.

Hence, let $P \subseteq \mathbb{R}^n$ and it remains to show the statement for linear functions $\varphi\colon \mathbb{R}^n \to \mathbb{R}^m$. To highlight that the function is linear, we denote it again by $\phi\colon \mathbb{R}^n \to \mathbb{R}^m$. By Proposition 1.38, any polyhedron P can be written as a Minkowski sum

$$P = Q + C \enspace ,$$

where $Q \subseteq \mathbb{R}^n$ is a polytope and $C \subseteq \mathbb{R}^n$ is a polyhedral cone. Observe that

$$\begin{aligned}\phi(P) &= \phi(Q + C)\\ &= \{\phi(q + c) \colon q \in Q, c \in C\}\\ &= \{\phi(q) + \phi(c) \colon q \in Q, c \in C\}\\ &= \phi(Q) + \phi(C) \enspace .\end{aligned} \tag{1.10}$$

We finish the proof by showing that $\phi(Q) \subseteq \mathbb{R}^m$ is a polytope and $\phi(C) \subseteq \mathbb{R}^m$ is a polyhedral cone, which, by Proposition 1.38 then implies that $\phi(Q) + \phi(C)$ is a polyedron, thus finishing the proof because $\phi(P) = \phi(Q) + \phi(C)$ by (1.10).

Because Q is a polytope, we have

$$Q = \operatorname{conv}(\operatorname{vertices}(Q))$$

by Proposition 1.32. Let $\text{vertices}(Q) = \{q_1, \dots, q_k\}$. Hence,

$$\begin{aligned}\phi(Q) &= \{\phi(x) \colon x \in \text{conv}(\text{vertices}(Q))\}\\ &= \left\{\phi\left(\sum_{i=1}^{k} \lambda_i q_i\right) \colon \lambda \in \mathbb{R}^k_{\geq 0}, \sum_{i=1}^{k} \lambda_i = 1\right\}\\ &= \left\{\sum_{i=1}^{k} \lambda_i \phi(q_i) \colon \lambda \in \mathbb{R}^k_{\geq 0}, \sum_{i=1}^{k} \lambda_i = 1\right\}\\ &= \text{conv}(\{\phi(q) \colon q \in \text{vertices}(Q)\}) \enspace ,\end{aligned}$$

implying that $\phi(Q)$ is the convex hull of the finitely many points $\phi(q_1), \dots, \phi(q_k)$, and thus is a polytope due to Proposition 1.32.

It remains to show that $\phi(C) \subseteq \mathbb{R}^m$ is a polyhedral cone. By Proposition 1.37, there are some points $x_1, \dots, x_\ell \subseteq \mathbb{R}^n$ such that

$$C = \left\{\sum_{i=1}^{\ell} \lambda_i x_i \colon \lambda \in \mathbb{R}^\ell_{\geq 0}\right\} \enspace .$$

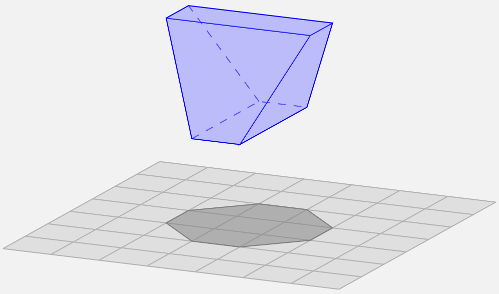
Hence,

$$\begin{aligned}\phi(C) &= \left\{\phi\left(\sum_{i=1}^{\ell} \lambda_i x_i\right) \colon \lambda \in \mathbb{R}^\ell_{\geq 0}\right\}\\ &= \left\{\sum_{i=1}^{\ell} \lambda_i \phi(x_i) \colon \lambda \in \mathbb{R}^\ell_{\geq 0}\right\} \enspace ,\end{aligned}$$

which implies by Proposition 1.37 that $\phi(C) \subseteq \mathbb{R}^m$ is a polyhedral cone, as desired. □

Example 1.41: Projection of polytope

The illustration below shows an axis-parallel projection of a 3-dimensional polytope leading to a 2-dimensional octagon. First, this exemplifies that the projection of a polytope is a polytope.



Moreover, we highlight that the number of facets of the projection, which is 8, is strictly larger than the number of facets of the original polytope, which is only 6. The fact that polyhedra

with many facets can sometimes be represented as projections of polyhedra with much fewer facets is a crucial property that can sometimes be exploited to obtain compact representations of polyhedra with exponentially many facets in terms of the dimension. More precisely, there are polytopes $P \subseteq \mathbb{R}^n$ whose number of facets is $2^{\Omega(n)}$, which can be obtained as a projection of a polytope $Q \subseteq \mathbb{R}^m$ for some $m \geq n$, where the number of facets of Q is bounded by a polynomial in n.
Moving to a higher-dimensional space and then projecting is crucial to obtain a compact representation. Indeed, due to Corollary 1.21, any inequality description of a polyhedron with exponentially many facets needs exponentially many constraints; actually, at least one per facet.

Moreover, Proposition 1.40 can be leveraged to show that certain sets are polyhedra by showing that they are affine projections of polyhedra. This allows us to provide a simple proof of the following basic fact on dominants.

Proposition 1.42

The dominant of a polyhedron is a polyhedron.

Proof. Let $P \subseteq \mathbb{R}^n$ be a polyhedron. We recall the definition of the dominant of P, i.e.,

$$\mathrm{dom}(P) = P + \mathbb{R}^n_{\geq 0} \ .$$

By Proposition 1.38, the polyhedron P can be written as

$$P = Q + C \ ,$$

where $Q \subseteq \mathbb{R}^n$ is a polytope and $C \subseteq \mathbb{R}^n$ is a polyhedral cone. Hence,

$$\mathrm{dom}(P) = Q + C + \mathbb{R}^n_{\geq 0} \ .$$

We prove the result by showing that $C + \mathbb{R}^n_{\geq 0}$ is a polyhedral cone. This indeed finishes the proof because $\mathrm{dom}(P) = Q + (C + \mathbb{R}^n_{\geq 0})$ is the Minkowski sum of a polytope Q and a polyhedral cone $C + \mathbb{R}^n_{\geq 0}$, and thus a polyhedron due to Proposition 1.38.

By Proposition 1.37, there are points $x_1, \ldots, x_k \in \mathbb{R}^n$ such that

$$C = \left\{ \sum_{j=1}^k \mu_j x_j \colon \mu \in \mathbb{R}^k_{\geq 0} \right\} \ .$$

Hence, by denoting by $e_i \in \mathbb{R}^n$ the i-th unit vector in $\mathbb{R}^n$, we have

$$\begin{aligned} C + \mathbb{R}^n_{\geq 0} &= \left\{ \sum_{j=1}^k \mu_j x_j \colon \mu \in \mathbb{R}^k_{\geq 0} \right\} + \left\{ \sum_{i=1}^n \lambda_i e_i \colon \lambda \in \mathbb{R}^n_{\geq 0} \right\} \\ &= \left\{ \sum_{j=1}^k \mu_j x_j + \sum_{i=1}^n \lambda_i e_i \colon \mu \in \mathbb{R}^k_{\geq 0}, \lambda \in \mathbb{R}^n_{\geq 0} \right\} \ , \end{aligned}$$

which implies by Proposition 1.37 that $C + \mathbb{R}^n_{\geq 0}$ is a polyhedral cone, as desired. □

1.2.3 Convex separation theorems

Definition 1.43: (Strictly) separating hyperplanes

Let $Y, Z \subseteq \mathbb{R}^n$ be two sets. A hyperplane $H = \{x \in \mathbb{R}^n \colon a^\top x = \beta\}$ is called a (Y, Z)-*separating hyperplane*, or simply *separating hyperplane*, if Y is contained in one of the half-spaces defined by H and Z in the other one, i.e., either

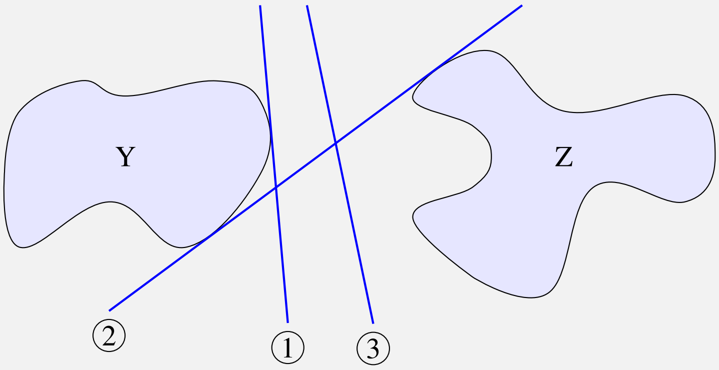
$$\begin{aligned} a^\top y \leq \beta \leq a^\top z \qquad &\forall y \in Y, z \in Z \enspace, \text{ or} \\ a^\top y \geq \beta \geq a^\top z \qquad &\forall y \in Y, z \in Z \enspace. \end{aligned}$$

The hyperplane is called *strictly* (Y, Z)*-separating*, or simply *strictly separating*, if the above inequalities are strict.

If $Y = \{y\}$ is a single point, we also write (y, Z)-separating hyperplane instead of $(\{y\}, Z)$-separating hyperplane.

Example 1.44: Separating two sets

The illustration below shows two sets $Y, Z \subseteq \mathbb{R}^2$ together with three separating hyperplanes. Hyperplane 3 is strictly separating the sets whereas hyperplanes 1 and 2 do not separate Y and Z in a strict sense.



Theorem 1.45: Separating a point from a polyhedron

Let $P \subseteq \mathbb{R}^n$ be a polyhedron and $y \in \mathbb{R}^n \setminus P$. Then there is a strictly (y, P)-separating hyperplane.

Proof. Consider an inequality description of P, i.e., $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$. Because $y \in \mathbb{R}^n \setminus P$, there is an inequality $a^\top x \leq \beta$ among the system $Ax \leq b$ such that $a^\top y > \beta$. Then, the hyperplane

$$\left\{x \in \mathbb{R}^n \colon a^\top x = \frac{1}{2}\left(\beta + a^\top y\right)\right\}$$

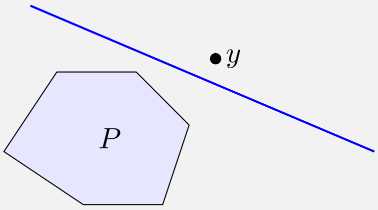
is strictly (y, P)-separating because

$$\begin{aligned} a^\top x \leq \beta &< \frac{1}{2}\left(\beta + a^\top y\right) \qquad \forall x \in P, \text{ and} \\ a^\top y &> \frac{1}{2}\left(\beta + a^\top y\right) \quad , \end{aligned}$$

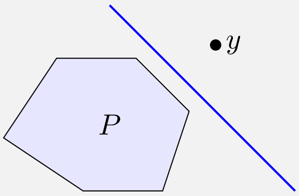
where the last inequality follows from $a^\top y > \beta$. □

Example 1.46: Separating a point from a polyhedron

The illustration below shows a hyperplane strictly separating a point from a polytope. Whenever a point is not contained in a polytope, it can always be strictly separated.



Moreover, as the proof of Theorem 1.45 highlights, one can always choose a separating hyperplane with the same normal vector as one of the constraints in any inequality description of P. In particular, if P is full-dimensional, like in the example above, we can use an inequality description that has one inequality per facet (see Corollary 1.21), and therefore separate y with a hyperplane that is parallel to a facet of P, as illustrated below.



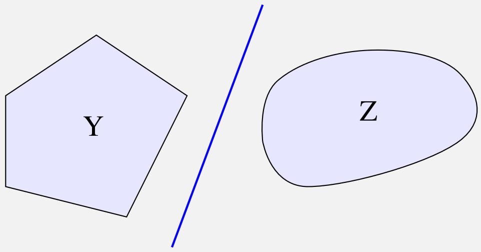
The above separation theorem is a very special case of the following much more general convex separation theorem, which we state below without proof.

Theorem 1.47

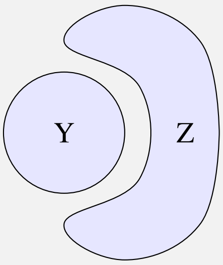
Let $Y, Z \subseteq \mathbb{R}^n$ be two disjoint closed convex sets with at least one of them being compact, then there exists a strictly (Y, Z)-separating hyperplane.

Example 1.48: Separating two sets

The illustration below shows two convex sets strictly separated by a hyperplane, which is always possible.



However, when at least one of the sets is not convex, separation may not be possible anymore as illustrated in the example below.



1.3 Simplex Method

What is known as Simplex Methods, or simply the Simplex Method, is a class of approaches to solve linear programs. Simplex Methods maintain a vertex solution that they iteratively improve by moving along the edges of the feasible region to strictly better vertex solutions. The Simplex Method is widely used in practice, despite the fact that there is no known realization of it that is guaranteed to run efficiently, i.e., in polynomial time (see Chapter 2 for more details on Computational Complexity). Moreover, most variants of the Simplex Method are even known to run in exponential time on some instances. Nevertheless, the Simplex Method shows excellent performance when applied to practical problems. Moreover, its study allows us to obtain a deeper understanding of linear programming in general and comes in handy to discuss further concepts like linear duality.

1.3.1 Geometric idea

We start with a high-level description of the main geometric idea behind the Simplex Method. Consider a linear program $\max\{c^\top x \colon x \in P\}$, where $P \subseteq \mathbb{R}^n$ is a polyhedron. Assume that we know a vertex $y \in \mathrm{vertices}(P)$. The Simplex Method seeks to improve the solution y through a local-search approach, that goes along one of the edges incident with y. In case of P being a polytope, each of these edges will lead to another vertex of P; such vertices that can be reached from y by following an edge are called *polyhedral neighbors*, or simply *neighbors*, of y (with respect to P). A key observation behind the Simplex Method is the following:

(i) Either y is an optimal solution to $\max\{c^\top x \colon x \in P\}$, or

(ii) there is an edge incident with y such that by going along the edge, the objective value strictly increases. In particular, if another vertex z is encountered along this edge, then $c^\top z > c^\top y$; otherwise, if one can continue indefinitely along the edge without leaving P, then the linear program is unbounded because we can get solutions with arbitrarily high objective value.

To illustrate the above, consider a linear program on a polytope P. Here, the Simplex Method will do a walk along the edges of P, going from one vertex to a strictly better one until an optimal vertex is reached. See Figure 1.9 for an illustration.

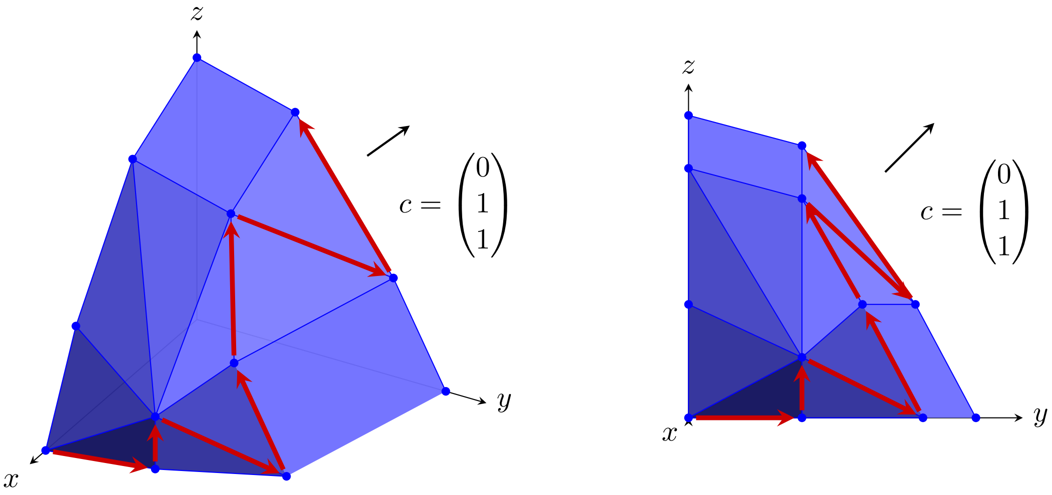

z
y
x
$c = \begin{pmatrix}0\\1\\1\end{pmatrix}$
z
y
x
$c = \begin{pmatrix}0\\1\\1\end{pmatrix}$

Figure 1.9: Illustration of a possible walk performed by the Simplex Method in a 3-dimensional linear program. The feasible region is the polytope depicted in blue and the task is to maximize the objective $y + z$. The right-hand side shows the projection of the feasible region onto the (y, z)-space. In this projection, one can nicely see that whenever we move from a vertex to a neighboring one, the objective function value strictly increases.

In case of an unbounded LP, which implies in particular that P is an unbounded polyhedron, the walk along the edges of P continues until $y \in \mathrm{vertices}(P)$ is found together with an improving edge direction $v \in \mathbb{R}^n$, i.e., $c^\top v > 0$, such that $y + \lambda v \in P$ for $\lambda \in \mathbb{R}_{\geq 0}$. See Figure 1.10 for an illustration.

This simple geometric intuition of the Simplex Method leaves several technical questions unanswered, including:

- Given $y \in \mathrm{vertices}(P)$, how to find an improving edge direction?
- How to find a starting vertex $y \in \mathrm{vertices}(P)$?
- How to use such a method to show that a linear program is infeasible?

We start by addressing the first question, for which the so-called *phase II* of the Simplex Method provides an answer. Once this is understood, a simple yet elegant auxiliary construction allows for answering the other two questions; this is known as *phase I* of the Simplex Method.

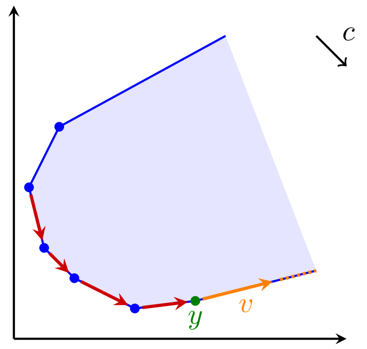


Figure 1.10: In the above unbounded example, the red arrows highlight the vertex-to-vertex walk performed by the Simplex Method. At the last vertex y there is an edge-direction v such that $y + \lambda v$ for $\lambda \geq 0$ is a feasible half-line with points of arbitrarily large objective value.

To algebraically realize the geometric idea of moving from vertex to vertex along edges, we will start with an LP in canonical form and then transform this description of the LP into the so-called *standard form*. The standard form only has equality constraints on top of the usual non-negativity requirements on the variables. This form allows us to highlight a particular vertex, and can be brought into equivalent standard forms highlighting a neighboring vertex. This way, we can capture the walk from vertex to vertex that is performed by the Simplex Method through changing the standard form. We therefore start by introducing the standard form and showing how to move between equivalent standard forms.

1.3.2 From canonical to standard form

A linear program is described in *standard form* if it only has equality constraints and its variables are required to be non-negative. Hence, it is of the following form:

$$\begin{array}{rrcl} \max & c^\top x & & \\ & Ax & = & b \\ & x & \geq & 0 \ . \end{array} \tag{LP in standard form}$$

In the exposition that follows to introduce the Simplex Method, we will actually start with a general LP in canonical form:

$$\begin{array}{rrcl} \max & c^\top x & & \\ & Ax & \leq & b \\ & x & \geq & 0 \ , \end{array}$$

where $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. From this canonical form, we build the corresponding standard form by introducing a new set of m *slack variables* $y \in \mathbb{R}^m$ in the following way:

$$\begin{array}{rrcrcl} \max & c^\top x & & & & \\ & Ax & + & y & = & b \\ & x & & & \geq & 0 \\ & & & y & \geq & 0 \ . \end{array} \tag{1.11}$$

Indeed, the feasible values that $x \in \mathbb{R}^n$ can take in the above LP in standard form are the same as in the original LP in canonical form. Hence, the two linear programs are equivalent. Going from canonical to standard form is a very common transformation. The variables y are called *slack variables* because they measure the slack of x with respect to the originally given inequalities in canonical form.

One may wonder why we started with an LP in canonical form, and only then moved to standard form, instead of right-away starting with an LP in standard form. The main reason is that even though we will do operations in standard form from now on, we interpret these operations as walking from vertex to vertex on the polyhedron $P := \{x \in \mathbb{R}^n_{\geq 0} \colon Ax \leq b\}$, i.e., the feasible region of the original LP given in canonical form. Hence, we will use the standard form of a canonical LP description as a convenient algebraic way to represent a walk along vertices of P. Thus, the geometric interpretation is with respect to the original LP in canonical form.

Some terminology

A pair $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m$ satisfying the equation system $Ax + y = b$ defined in (1.11) is called a *solution* of the LP. If a solution also satisfies $x \geq 0$ and $y \geq 0$ it is called *feasible*.

We will first discuss solutions to the system of equations $Ax + y = b$, with m equations and $n + m$ unknowns. Notice that this system has full row rank due to the variables y. Hence, it admits solutions for any right-hand side $b \in \mathbb{R}^m$.

In what follows, we use the simple LP below, in canonical form, to exemplify some notation and terminology.

$$\begin{array}{rrcrcrl} \max z = & 400x_1 & + & 900x_2 & & & \\ & x_1 & + & 4x_2 & \leq & 40 & \text{(constraint 1)} \\ & 2x_1 & + & x_2 & \leq & 42 & \text{(constraint 2)} \\ & 1.5x_1 & + & 3x_2 & \leq & 36 & \text{(constraint 3)} \\ & x_1 & & & \geq & 0 & \text{(non-negativity for } x_1\text{)} \\ & & & x_2 & \geq & 0 & \text{(non-negativity for } x_2\text{)} \end{array} \tag{1.12}$$

Figure 1.11 highlights the feasible region of LP (1.12).

To transform the above LP to the corresponding standard form, we introduce three slack variables y_1, y_2, and y_3, leading to the following system, represented in tabular form below.

$$\boxed{\text{I}} \qquad \begin{array}{|ccccc|c} y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\ \hline 1 & 0 & 0 & 1 & 4 & 40 \\ 0 & 1 & 0 & 2 & 1 & 42 \\ 0 & 0 & 1 & 1.5 & 3 & 36 \\ \multicolumn{5}{c}{\underbrace{\hphantom{y_1 \quad y_2 \quad y_3 \quad x_1 \quad x_2}}_{\text{coefficient matrix}}} & \underbrace{\hphantom{00}}_{\text{right-hand side}} \end{array}$$

An important property of the standard LP form built from a canonical LP form is that there is a one-to-one correspondence between variables in the standard form and constraints in the original

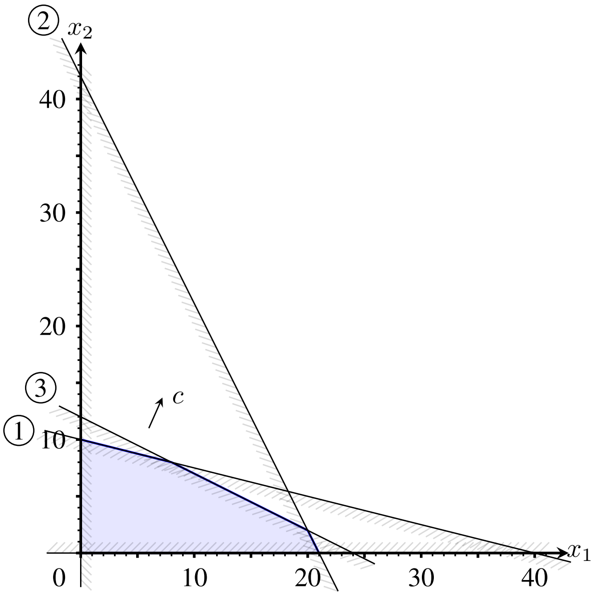


Figure 1.11: Graphical representation of the linear program (1.12).

canonical form (including also the non-negativity constraints). More precisely, even though only the variables y_1, y_2, and y_3 are called *slack variables* in the above example, also the variables x_1 and x_2 measure slack, namely the slack with respect to the non-negativity constraints in (1.12).

Due to the particular form of the linear equation system in the standard LP above, we can easily obtain a parametrized form of the set of all solutions, namely:

$$\begin{array}{rcrcrcr} y_1 & = & 40 & - & x_1 & - & 4x_2 \\ y_2 & = & 42 & - & 2x_1 & - & x_2 \\ y_3 & = & 36 & - & 1.5x_1 & - & 3x_2 \ . \end{array} \tag{1.13}$$

Indeed, every choice of x_1 and x_2 determines a unique solution to the system. Correspondingly,

$$\begin{array}{rl} (x_1, x_2) & \text{are called } \textit{free variables}\text{, and} \\ (y_1, y_2, y_3) & \text{are called } \textit{dependent variables.} \end{array}$$

Clearly, the dimension of the solution space is determined by the number of free variables; hence, 2 for our example. Moreover, to each parameterized equation system, like (1.13), we associate a *basic solution*, namely the one obtained by setting all free variables to zero. Thus, the basic solution for (1.13) is $(x_1, x_2, y_1, y_2, y_3) = (0, 0, 40, 42, 36)$. By construction, such a basic solution lies on the intersection of n linearly independent tight constraints of the initial canonical form LP, i.e., the normal vectors of the constraints are linearly independent. Hence, if the basic solution is feasible, it corresponds to a vertex of the feasibility region $P = \{x \in \mathbb{R}^n_{\geq 0} \colon Ax \leq b\}$ of the original problem in canonical form by Proposition 1.19. Thus, parameterized forms are a way to represent the equation system in a form highlighting a particular vertex solution. By moving from one parameterized form to another equivalent one, with different free and dependent variables, we can highlight different basic solutions. This is the way how the geometric idea of the Simplex Method of moving from vertex to vertex along edges will be realized algebraically.

Example 1.49

The tableau below corresponds to an equation system that is equivalent to system $\boxed{\text{I}}$, but with different dependent and free variables.

$$\boxed{\text{II}} \qquad \begin{array}{c|ccccc|c}
 & y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\
\hline
 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 1 & 10 \\
 & -\frac{1}{4} & 1 & 0 & \frac{7}{4} & 0 & 32 \\
 & -\frac{3}{4} & 0 & 1 & \frac{3}{4} & 0 & 6
\end{array}$$

The columns corresponding to the variables $B = (x_2, y_2, y_3)$ form an identity matrix. The solution space in its parametrized form with free variables (y_1, x_1) can directly be read from the tableau:

$$\begin{array}{rcrcrcr}
x_2 & = & 10 & - & \frac{1}{4}y_1 & - & \frac{1}{4}x_1 \\
y_2 & = & 32 & + & \frac{1}{4}y_1 & - & \frac{7}{4}x_1 \\
y_3 & = & 6 & + & \frac{3}{4}y_1 & - & \frac{3}{4}x_1 \ .
\end{array}$$

1.3.3 Equivalent linear systems

To move from one equation system to another equivalent one, *elementary row operations* can be used.

Definition 1.50: Elementary row operations

(i) Change the order of the equations.
(ii) Multiply an equation with a non-zero real number.
(iii) Add a multiple of one equation to another one.

One can easily observe that these operations do not change the set of solutions to a linear system.

Example 1.51

We illustrate these operations using our example, to show that the linear equation systems I and II indeed have the same set of solutions.

$$\boxed{\text{I}} \qquad \begin{array}{c|ccccc|c}
 & y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\
\hline
 & 1 & 0 & 0 & 1 & 4 & 40 \\
 & 0 & 1 & 0 & 2 & 1 & 42 \\
 & 0 & 0 & 1 & 1.5 & 3 & 36
\end{array}$$

Multiply the first equation by $\frac{1}{4}$:

$$\rightarrow \boxed{\text{Ia}} \qquad \begin{array}{|ccccc|c} y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\ \hline \frac{1}{4} & 0 & 0 & \frac{1}{4} & 1 & 10 \\ 0 & 1 & 0 & 2 & 1 & 42 \\ 0 & 0 & 1 & 1.5 & 3 & 36 \end{array}$$

Add $(-1)\times$ first row to the second row, and add $(-3)\times$ first row to the third row:

$$\boxed{\text{II}} \qquad \begin{array}{|ccccc|c} y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\ \hline \frac{1}{4} & 0 & 0 & \frac{1}{4} & 1 & 10 \\ -\frac{1}{4} & 1 & 0 & \frac{7}{4} & 0 & 32 \\ -\frac{3}{4} & 0 & 1 & \frac{3}{4} & 0 & 6 \end{array}$$

Definition 1.52: Equivalent equation systems

Systems of linear equations, which can be transformed to each other using elementary row operations, are called *equivalent*.

$\boxed{\text{I}}$, $\boxed{\text{Ia}}$, and $\boxed{\text{II}}$ are equivalent systems and have therefore the same set of solutions.

Conversely, it is not hard to see that two linear equation systems with full row rank and identical solutions can be obtained from one another through elementary row operations.

As discussed, the equation system we obtained by transforming an LP given in canonical form into one in standard form has the special property that the coefficients of the slack variables in the system form an identity matrix. This form, which leads to a natural parameterization of the solution space as highlighted previously and allows us to talk about basic solutions, is typically referred to as the *tableau form* in the context of linear programming.

Definition 1.53: Tableau form

An equation system $Ax = b$ with m equations in $n+m$ variables is in *tableau form* if its coefficient matrix contains an identity matrix. In this case:

- A tuple of variables whose corresponding columns—listed in the same order in which they appear in the tuple—form an identity matrix, are called a *basis*.
- The variables used in the basis are called *basic variables*.
- All other variables are called *non-basic variables*.
- The columns corresponding to basic variables are called *basic columns*.
- The columns corresponding to non-basic variables are called *non-basic columns*.
- The *basic solution* corresponding to a basis is the unique solution obtained by setting all non-basic variables to zero; hence, the basic variables will be set to b.

Notice that, given an equation system in tableau form, it may have more than one basis. This is the reason why a basic solution is defined with respect to a basis and not with respect to an equation system in tableau form. Nevertheless, if the basis of an equation system in tableau form is unique, we also talk about *the basic solution* of the system.

As mentioned, for any values of the non-basic variables, there is a unique set of values for the basic ones that leads to a solution to the equation system. Hence, the non-basic variables correspond to what we called *free* variables and the basic ones to *non-free* variables.

Example 1.54

The following system of linear equations in tableau form has a unique basis, namely (y_1, y_2, y_3).

y_1	y_2	y_3	x_1	x_2	1
1	0	0	1	4	40
0	1	0	2	1	42
0	0	1	1.5	3	36

The system of linear equations shown below is not in tableau form because it does not contain an identity matrix.

y_1	y_2	y_3	x_1	x_2	1
0	1	0	2	1	42
$-\frac{1}{4}$	1	0	$\frac{7}{4}$	0	32
$-\frac{3}{4}$	0	1	$\frac{3}{4}$	0	6

However, it turns out to be equivalent to the first system. To check this, one can apply row operations to the second system to obtain the first one or vice versa. Applying row operations to a matrix is the same as multiplying the matrix from the left with a full-rank matrix. Clearly, up to column permutations, the only matrix $S \in \mathbb{R}^{3\times 3}$ that, when being multiplied with the above matrix, can lead to the first system, is the inverse of

$$\begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{4} & 1 & 0 \\ -\frac{3}{4} & 0 & 1 \end{pmatrix} ,$$

i.e., the matrix formed by the columns corresponding to y_1, y_2, and y_3. Hence, we need to choose

$$S = \begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{4} & 1 & 0 \\ -\frac{3}{4} & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 4 & -4 & 0 \\ 1 & 0 & 0 \\ 3 & -3 & 1 \end{pmatrix} .$$

Finally, by multiplying S with the 3×6 matrix consisting of the entries (including its right-hand side) of the second tableau, which is not in standard form, we obtain the matrix corresponding to the first tableau. Hence, the two systems are indeed equivalent.

Moreover, the following system of equations in tableau form does not have a unique basis.

$$
\begin{array}{|ccccc|c}
y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 40 \\
0 & 1 & 0 & 1 & 2 & 50 \\
0 & 0 & 1 & 0 & 3 & 60
\end{array}
$$

Indeed, there are two possible bases, namely (y_1, y_2, y_3) and (y_1, x_1, y_3).

Example 1.55

Consider the three equation systems I, Ia, and II in Example 1.51. System I is in tableau form with basis $B = (y_1, y_2, y_3)$. Hence, y_1, y_2, and y_3 are the basic variables. System II is in tableau form with basis $B = (x_2, y_2, y_3)$, whereas system Ia is *not* in tableau form.

Example 1.56

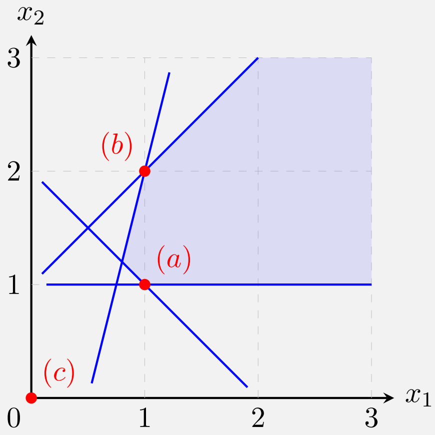
Consider the following LP, given in canonical form.

$$
\begin{array}{lrcrcr}
\max & -x_1 & - & x_2 & & \\
 & & - & x_2 & \leq & -1 \\
 & -x_1 & - & x_2 & \leq & -2 \\
 & -4x_1 & + & x_2 & \leq & -2 \\
 & -x_1 & + & x_2 & \leq & 1 \\
 & x_1 & & & \geq & 0 \\
 & & & x_2 & \geq & 0
\end{array}
$$

By adding slack variables we obtain the equivalent standard form shown below.

$$
\begin{array}{lccccrcrcrcr}
\max & & & & & - & x_1 & - & x_2 & & \\
 & y_1 & & & & & & - & x_2 & = & -1 \\
 & & y_2 & & & - & x_1 & - & x_2 & = & -2 \\
 & & & y_3 & & - & 4x_1 & + & x_2 & = & -2 \\
 & & & & y_4 & - & x_1 & + & x_2 & = & 1 \\
 & & & & & & & & x & \in & \mathbb{R}^2_{\geq 0} \\
 & & & & & & & & y & \in & \mathbb{R}^4_{\geq 0}
\end{array}
$$

The graphic below shows the feasible region of the LP in the (x_1, x_2)-space.



Below, we present three equivalent equation systems, (a), (b), and (c) in tableau form that all correspond to the above-written LP in standard form. Each of these systems has a unique basis, and hence, each system has a well-defined basic solution, which is highlighted in the graphic above.

System (a) in tableau form with basic solution $(x_1, x_2, y_1, y_2, y_3, y_4) = (1, 1, 0, 0, 1, 1)$:

$$
(a) \qquad \begin{array}{|rrrrrr|r}
y_1 & y_2 & y_3 & y_4 & x_1 & x_2 & 1 \\
\hline
-1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & -1 & 0 & 0 & 1 & 0 & 1 \\
5 & -4 & 1 & 0 & 0 & 0 & 1 \\
2 & -1 & 0 & 1 & 0 & 0 & 1
\end{array}
$$

System (b) in tableau form with basic solution $(x_1, x_2, y_1, y_2, y_3, y_4) = (1, 2, 1, 1, 0, 0)$:

$$
(b) \qquad \begin{array}{|rrrrrr|r}
y_1 & y_2 & y_3 & y_4 & x_1 & x_2 & 1 \\
\hline
0 & 0 & -\frac{1}{3} & \frac{4}{3} & 0 & 1 & 2 \\
0 & 0 & -\frac{1}{3} & \frac{1}{3} & 1 & 0 & 1 \\
0 & 1 & -\frac{2}{3} & \frac{5}{3} & 0 & 0 & 1 \\
1 & 0 & -\frac{1}{3} & \frac{4}{3} & 0 & 0 & 1
\end{array}
$$

System (c) in tableau form with basic solution $(x_1, x_2, y_1, y_2, y_3, y_4) = (0, 0, -1, -2, -2, 1)$:

$$
(c) \qquad \begin{array}{|rrrrrr|r}
y_1 & y_2 & y_3 & y_4 & x_1 & x_2 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & -1 & -1 \\
0 & 1 & 0 & 0 & -1 & -1 & -2 \\
0 & 0 & 1 & 0 & -4 & 1 & -2 \\
0 & 0 & 0 & 1 & -1 & 1 & 1
\end{array}
$$

1.3.4 Exchange step/pivoting

Transforming one tableau into another can be done using *exchange steps*, which are also called *pivoting steps*. An exchange step is a sequence of elementary row operations that are uniquely determined by the variables "entering" and "leaving" the basis. Consider a tableau with basis B. Let $k \in [m+n]$ be the index of a non-basic column, and let $i \in [m]$ be a row such that $A_{ik} \neq 0$. The variable corresponding to column k will enter the basis and its column will be transformed to the i-th canonical vector $e_i \in \mathbb{R}^m$, i.e., for $j \in [m]$ we have $e_i(j) = 1$ if $i = j$ and $e_i(j) = 0$ otherwise. Therefore, the basic variable whose column is e_i will leave the basis, leading to a new basis B'.

The coefficient matrix A' and right-hand side b' of an equivalent tableau with basis B' can be obtained through the following rules, which simply correspond to a sequence of elementary row operations achieving the above exchange:

(i) $A'_{i\ell} = \dfrac{A_{i\ell}}{A_{ik}}$ $\qquad \ell \in [m+n]$

(ii) $A'_{j\ell} = A_{j\ell} - \dfrac{A_{jk} \cdot A_{i\ell}}{A_{ik}}$ $\qquad \ell \in [m+n], \qquad j \neq i, j \in [m]$

(iii) $b'_i = \dfrac{b_i}{A_{ik}}$

(iv) $b'_j = b_j - \dfrac{A_{jk} \cdot b_i}{A_{ik}}$ $\qquad j \in [m]$ with $j \neq i$,

where

- A_{ik} is called *pivot element*;
- the row with index i is called *pivot row*;
- the column with index k is called *pivot column*.

Typically, when talking about a pivot element A_{ik}, we do not just refer to the actual scalar A_{ik}, but also to the corresponding row index i and column index k. Due to this, it makes sense to talk about "performing an exchange step on the pivot A_{ik}", even if the matrix A contains multiple entries with the same value as A_{ik}.

Example 1.57

The unique basis of the tableau below is $B = (x_2, y_2, y_3)$.

$$\boxed{\text{II}} \qquad \begin{array}{c|ccccc|c}
 & y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\
\hline
 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 1 & 10 \\
 & -\frac{1}{4} & 1 & 0 & \frac{7}{4} & 0 & 32 \\
 & -\frac{3}{4} & 0 & 1 & \boxed{\frac{3}{4}} & 0 & 6
\end{array}$$

After an exchange step with $k = 4, i = 3$ one obtains the following equivalent system:

$$\boxed{\text{III}} \qquad \begin{array}{|ccccc|c}
y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\ \hline
\frac{1}{2} & 0 & -\frac{1}{3} & 0 & 1 & 8 \\
\frac{6}{4} & 1 & -\frac{7}{3} & 0 & 0 & 18 \\
-1 & 0 & \frac{4}{3} & 1 & 0 & 8
\end{array}$$

The basis of this system is $B' = (x_2, y_2, x_1)$, and its parameterized solution set is the following:

$$\begin{array}{rcrcrcr}
x_2 & = & 8 & - & \frac{1}{2}y_1 & + & \frac{1}{3}y_3 \\
y_2 & = & 18 & - & \frac{6}{4}y_1 & + & \frac{7}{3}y_3 \\
x_1 & = & 8 & + & y_1 & - & \frac{4}{3}y_3
\end{array}$$

Moreover, the basic solution of $\boxed{\text{III}}$ is $(x_1, x_2, y_1, y_2, y_3) = (8, 8, 0, 18, 0)$.

1.3.5 Short tableau

The short tableau is a more compact way of representing the tableau forms we have seen so far and has further important advantages. First, as already mentioned, it reduces the amount of data that has to be carried along when doing operations on the tableau. Second, it will prove very useful when we talk about the dual of a linear program, many of whose properties can be read more naturally from the short tableau. Last but not least, the short tableau avoids ambiguities in terms of what variables are used in the basis. To clearly distinguish the tableau form we already introduced from the short tableau, we will also sometimes talk about the long tableau to refer to the former.

For simplicity, we introduce the structure of a short tableau through an example. Consider the following long tableau with unique basis $B = (x_2, y_2, y_3)$.

$$\boxed{\text{II}} \qquad \begin{array}{|ccccc|c}
y_1 & y_2 & y_3 & x_1 & x_2 & 1 \\ \hline
\frac{1}{4} & 0 & 0 & \frac{1}{4} & 1 & 10 \\
-\frac{1}{4} & 1 & 0 & \frac{7}{4} & 0 & 32 \\
-\frac{3}{4} & 0 & 1 & \frac{3}{4} & 0 & 6
\end{array}$$

By marking the i-th *row* of the tableau with the (basic) variable whose column is e_i and by deleting the corresponding unit vectors, we obtain a *short tableau with basis B*:

$$\boxed{\text{IIb}} \qquad \begin{array}{c|cc|c}
 & y_1 & x_1 & 1 \\ \hline
x_2 & \frac{1}{4} & \frac{1}{4} & 10 \\
y_2 & -\frac{1}{4} & \frac{7}{4} & 32 \\
y_3 & -\frac{3}{4} & \frac{3}{4} & 6
\end{array} \qquad\qquad B = (x_2, y_2, y_3)$$

1.3.6 Exchange step in short tableau

The exchange step we introduced for the long tableau can now be translated to the short one. For this consider a short tableau and let $A \in \mathbb{R}^{m\times n}$ be the matrix defining its entries, which all correspond to non-basic variables, and let b be the right-hand side of the short tableau. Hence, for the example $\boxed{\text{IIb}}$ above we have

$$A = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{7}{4} \\ -\frac{3}{4} & \frac{3}{4} \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 10 \\ 32 \\ 6 \end{pmatrix} \ .$$

An exchange step on the pivot $A_{ik} \neq 0$ in the short tableau leads to a new and equivalent short tableau with constraint matrix A' and right-hand side b' given by

$$\begin{aligned}
A'_{ik} &= \frac{1}{A_{ik}} && (pivot\ element) \\
A'_{jk} &= -\frac{A_{jk}}{A_{ik}} && (pivot\ column) \quad j \in [m], \quad j \neq i \\
A'_{i\ell} &= \frac{A_{i\ell}}{A_{ik}} && (pivot\ row) \quad \ell \in [n], \quad \ell \neq k \\
A'_{j\ell} &= A_{j\ell} - \frac{A_{jk} \cdot A_{i\ell}}{A_{ik}} && j \in [m], \quad j \neq i \\
& && \ell \in [n], \quad \ell \neq k \\
b'_i &= \frac{b_i}{A_{ik}} && \\
b'_j &= b_j - \frac{A_{jk} \cdot b_i}{A_{ik}} && j \in [m], \quad j \neq i \ .
\end{aligned} \tag{1.14}$$

Moreover, this pivoting step does not just change the entries of the short tableau, but also leads to a modification of the boundary, i.e., the variables assigned with the rows and columns. More precisely, the basic variable at row i swaps its role with the non-basic variable at column k.

Example 1.58

Consider performing an exchange step in system $\boxed{\text{IIb}}$ at the pivot element A_{32} as highlighted below.

$$\boxed{\text{IIb}} \qquad \begin{array}{c|cc|c} & y_1 & \boxed{x_1} & 1 \\ \hline x_2 & \frac{1}{4} & \frac{1}{4} & 10 \\ y_2 & -\frac{1}{4} & \frac{7}{4} & 32 \\ \boxed{y_3} & -\frac{3}{4} & \boxed{\frac{3}{4}} & 6 \end{array} \qquad B = (x_2, y_2, y_3)$$

This leads to the following equivalent system.

$$\begin{array}{c|cc|c} & y_1 & y_3 & 1 \\ \hline x_2 & \frac{1}{2} & -\frac{1}{3} & 8 \\ y_2 & \frac{6}{4} & -\frac{7}{3} & 18 \\ x_1 & -1 & \frac{4}{3} & 8 \end{array} \qquad \text{with basis } B' = (x_2, y_2, x_1)$$

Exercise 1.59

Verify the validity of the rules for performing an exchange step in a short tableau, i.e., show that they indeed give an equivalent linear system in tableau form with the claimed basis, by using the long tableau.

Example 1.60

For the following system of equations we can read off a tableau for the basis $B = (x_1, x_2, x_3)$:

$$\begin{array}{ccccccccccc} x_1 & & & & & + & 2x_4 & + & x_5 & = & 3 \\ & & x_2 & & & + & x_4 & - & x_5 & = & 0 \\ & & & & x_3 & + & x_4 & & & = & 2 \end{array}$$

$$B = (x_1, x_2, x_3) \qquad \begin{array}{c|cc|c} & x_4 & x_5 & 1 \\ \hline x_1 & 2 & \boxed{1} & 3 \\ x_2 & 1 & -1 & 0 \\ x_3 & 1 & 0 & 2 \end{array} \qquad \text{(a)}$$

The corresponding basic solution for this basis is:

$$(x_1, x_2, x_3, x_4, x_5) = (3, 0, 2, 0, 0) \ .$$

We perform exchange steps to compute further basic solutions (the pivot element is marked).

(b):

$$\begin{array}{c|cc|c} & x_4 & x_1 & 1 \\ \hline x_5 & 2 & 1 & 3 \\ x_2 & \boxed{3} & 1 & 3 \\ x_3 & 1 & 0 & 2 \end{array}$$

(c):

$$\begin{array}{c|cc|c} & x_2 & x_1 & 1 \\ \hline x_5 & -\frac{2}{3} & \frac{1}{3} & 1 \\ x_4 & \frac{1}{3} & \boxed{\frac{1}{3}} & 1 \\ x_3 & -\frac{1}{3} & -\frac{1}{3} & 1 \end{array}$$

(d):

$$\begin{array}{c|cc|c} & x_2 & x_4 & \\ \hline x_5 & \text{-1} & \text{-1} & 0 \\ x_1 & 1 & 3 & 3 \\ x_3 & 0 & 1 & 2 \end{array}$$

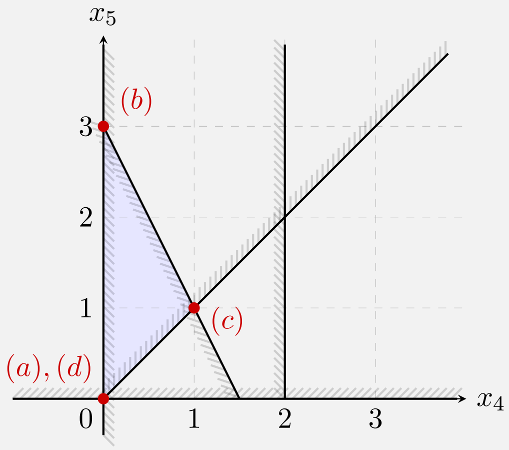
$(0, 3, 2, 0, 3)$ $(0, 0, 1, 1, 1)$ $(3, 0, 2, 0, 0)$

basic solutions as tuples $(x_1, x_2, x_3, x_4, x_5)$

Notice that the initial equation system corresponds to the standard form of an LP in canonical form with constraints given by

$$\begin{array}{rcrcl} 2x_4 & + & x_5 & \leq & 3 \\ x_4 & - & x_5 & \leq & 0 \\ x_4 & & & \leq & 2 \\ x_4 & & & \geq & 0 \\ & & x_5 & \geq & 0\ . \end{array}$$

Hence, we can visualize the feasible region of the equation system we started with, including non-negativity constraints, in the (x_4, x_5)-space:



The basic solutions corresponding to the tableaus (a), (b), (c), and (d) are highlighted in red in the above picture.

Example 1.60 also nicely exemplifies equivalent ways to look at an LP in terms of a different set of basic variables. More precisely, even though the original equation system corresponds to a 2-dimensional canonical LP in the variables x_4 and x_5, the other tableaus we obtained correspond to equivalent 2-dimensional LPs in a different set of variables. For example, tableau (b) corresponds to a 2-dimensional LP in the variables x_4 and x_1.

Exercise 1.61

Draw the 2-dimensional region of tuples $(x_2, x_3) \in \mathbb{R}^2$ that correspond to a feasible solution to the equation system in Example 1.60. Moreover, highlight the four basic solutions obtained in tableaus (a)–(d) in Example 1.60 in the (x_2, x_3)-space.

1.3.7 Simplex Method: phase II

The second phase of the Simplex Method requires a feasible basic solution to start with. It then performs exchange steps in a systematic way to either reach an optimal solution or to show that the given LP is unbounded.

To introduce the method, consider a general linear program in standard form obtained from one in canonical form, where we allow the objective function to contain additionally a constant term $q \in \mathbb{R}$, i.e.,

$$\begin{array}{rcccccl} \max & c^\top x & & & + & q & \\ & Ax & + & y & & = & b \\ & x & & & & \in & \mathbb{R}^n_{\geq 0} \\ & & & y & & \in & \mathbb{R}^m_{\geq 0}\ , \end{array}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. By introducing a variable $z \in \mathbb{R}$ for the objective, i.e., $z = c^\top x + q$, we can rewrite the problem as:

$$\begin{array}{rcccccl} \max & z & & & & & \\ & z & - & c^\top x & & & = & q \\ & & & Ax & + & y & = & b \\ & & & x & & & \in & \mathbb{R}^n_{\geq 0} \\ & & & & & y & \in & \mathbb{R}^m_{\geq 0}\ , \end{array} \tag{1.15}$$

which leads to a system of $m + 1$ equations in the variables z, x, and y. Consider the tableau corresponding to (1.15) in matrix form:

$$\begin{array}{c|ccc|ccc|c} z & y_1 & \dots & y_m & x_1 & \dots & x_n & 1 \\ \hline 1 & 0 & \dots & 0 & -c_1 & \dots & -c_n & q \\ \hline 0 & & I & & & A & & b \end{array} \tag{1.16}$$

We will work with the short form of this tableau, which is given below.

$$\begin{array}{c|ccc|c} & x_1 & \dots & x_n & 1 \\ \hline z & -c_1 & \dots & -c_n & q \\ \hline y_1 & A_{11} & \dots & A_{1n} & b_1 \\ \vdots & \vdots & & \vdots & \vdots \\ y_m & A_{m1} & \dots & A_{mn} & b_m \end{array} \tag{1.17}$$

In the following, the auxiliary variable z will always stay in the basis. Therefore, when we refer to a *basic solution* or *basic variables*, this will always be with respect to the remaining variables, i.e., x and y. We highlight the special role of the variable z in the tableau by separating the equation for the objective function with a horizontal line. Still, when we perform an exchange step on a tableau as shown in (1.17), the exchange step is performed as we previously introduced it, i.e., over all rows of the tableau including the first row $-c^\top$, which corresponds to the objective.

Definition 1.62: Feasible tableau

The tableau (1.17) is called *feasible* if $b_1, \dots, b_m \geq 0$, i.e., if the corresponding basic solution is feasible.

Definition 1.63: Value of tableau

The entry q in a tableau (see 1.17) is called the *value of the tableau*. This is the objective value of the basic solution that corresponds to the tableau. We use this terminology both for feasible and infeasible tableaus.

Example 1.64: Feasible tableau

The tableau below corresponds to the linear program shown in (1.12). It is a feasible tableau.

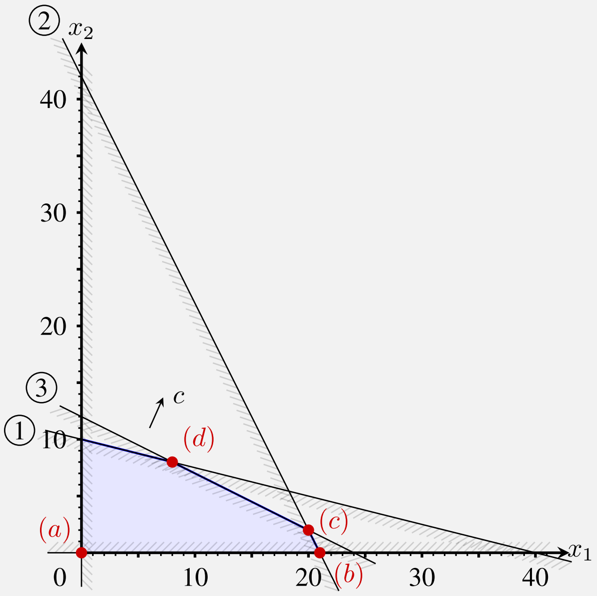
	x_1	x_2	1
z	-400	-900	0
y_1	1	4	40
y_2	2	1	42
y_3	1.5	3	36

The second phase of the Simplex Method starts with a feasible tableau and iteratively computes new feasible tableaus. Hence, the method never encounters an infeasible tableau. Given a feasible tableau, the goal is to move to another one with strictly higher objective value through a single exchange step. This will not always be possible. Hence, the second phase of the Simplex Method may sometimes pivot from one feasible tableau to another one with identical objective value. However, it will never pivot to a tableau with strictly worse objective value.

Before introducing a formal rule of how to perform exchange steps, we show in the example below how a well-chosen sequence of exchange steps solves the example linear program highlighted in (1.12).

Example 1.65

We recall the linear program shown in (1.12) in canonical form together with a graphical representation of its solution set.



The following sequence of tableaus, obtained through successive exchange steps, leads to the unique optimal solution of this LP. The corresponding basic feasible solutions are highlighted in the graphic above.

$$\begin{array}{c|cc|c} & x_1 & x_2 & 1 \\ \hline z & -400 & -900 & 0 \\ \hline y_1 & 1 & 4 & 40 \\ y_2 & \boxed{2} & 1 & 42 \\ y_3 & \frac{3}{2} & 3 & 36 \end{array} \quad \longrightarrow \quad \begin{array}{c|cc|c} & y_2 & x_2 & 1 \\ \hline z & 200 & -700 & 8400 \\ \hline y_1 & -\frac{1}{2} & \frac{7}{2} & 19 \\ x_1 & \frac{1}{2} & \frac{1}{2} & 21 \\ y_3 & -\frac{3}{4} & \boxed{\frac{9}{4}} & \frac{9}{2} \end{array}$$

(a) (b)

$$\downarrow$$

$$\begin{array}{c|cc|c} & y_1 & y_3 & 1 \\ \hline z & 50 & \frac{700}{3} & 10400 \\ \hline y_2 & \frac{3}{2} & -\frac{7}{3} & 18 \\ x_1 & -1 & \frac{4}{3} & 8 \\ x_2 & \frac{1}{2} & -\frac{1}{3} & 8 \end{array} \quad \longleftarrow \quad \begin{array}{c|cc|c} & y_2 & y_3 & 1 \\ \hline z & -\frac{100}{3} & \frac{2800}{9} & 9800 \\ \hline y_1 & \boxed{\frac{2}{3}} & -\frac{14}{9} & 12 \\ x_1 & \frac{2}{3} & -\frac{2}{9} & 20 \\ x_2 & -\frac{1}{3} & \frac{4}{9} & 2 \end{array}$$

(d) (c)

Notice that in this example, each of the 3 exchange steps strictly improved the objective value of the tableau, with the last tableau revealing the optimal objective value of 10400. Moreover, it is not hard to see that the last tableau corresponds to an optimal solution. Indeed, the objective row of the last tableau reveals that the objective value z satisfies

$$z = 10400 - 50y_1 - \frac{700}{3}y_3 \ .$$

Because all variables, including y_1 and y_3, are non-negative, this implies that no feasible solution of value higher than 10400 can be obtained. Finally, 10400 is the objective value attained by the basic solution that corresponds to the last tableau; hence, this solution is optimal.

To formally introduce a rule for selecting the pivot, consider a general feasible tableau as shown below, where we denote by x_B and x_N the basic and non-basic variables, respectively. Notice that there are always m many basic variables and n many non-basic variables. The location of the variables x_B and x_N is called the *boundary* of the tableau.

$$\begin{array}{c|ccc|c} & & x_N^\top & & 1 \\ \hline z & c_1 & \dots & c_n & q \\ \hline & A_{11} & \dots & A_{1n} & b_1 \\ x_B & \vdots & & \vdots & \vdots \\ & A_{m1} & \dots & A_{mn} & b_m \end{array} \tag{1.18}$$

When we obtain the tableau from the canonical form of an LP by transforming it first into standard form, then the variables x_B correspond to the slack variables, which we often denote by y. Moreover, because we consider a feasible tableau, we have that the right-hand side vector $b \in \mathbb{R}^m$ is non-negative.

Definition 1.66: Legal pivot element for phase II of Simplex Method

Given a feasible tableau, as shown in (1.18), a tuple $(i,k) \in [m] \times [n]$ corresponds to a legal pivot element A_{ik} for phase II of the Simplex Method if

(i) $c_k < 0$, and
(ii) $i \in \operatorname{argmin}\left\{\frac{b_j}{A_{jk}} : j \in [m] \text{ with } A_{jk} > 0\right\}$.

Pivot elements in phase II of the Simplex Method are typically chosen by first selecting the pivot column k, satisfying condition (i), and then the pivot row i to fulfill (ii). Condition (ii) is often called the *quotient rule*. Notice that by first choosing the pivot column k before choosing the row i, it may be that there is no row i that fulfills the quotient rule for column k, even though there may be a legal pivot element in another column. As we will see later, if this happens, then we do not have to look for another pivot column k, but can stop right-away because such a column turns out to prove unboundedness of the LP. Before expanding on this and further aspects, the following result motivates the definition of a legal pivot element.

Theorem 1.67

Consider a feasible tableau with a legal pivot element A_{ik} for phase II of the Simplex Method. Then the new tableau obtained after pivoting on A_{ik} satisfies the following:

(i) The new tableau is feasible.

(ii) The value of the new tableau is no less than that of the original one.

(iii) If $b_i > 0$, the value of the new tableau is strictly larger than that of the original one.

Proof.

(i) To check feasibility of the new tableau, we have to show that its right-hand side vector $b' \in \mathbb{R}^m$ is non-negative. We recall that b' is given by the pivot rules (1.14). Hence, for the pivot row we have

$$b'_i = \frac{b_i}{A_{ik}} \geq 0 \ ,$$

as $A_{ik} > 0$ because it is a legal pivot element and $b_i \geq 0$ because the original tableau is feasible. Moreover, for any other row $j \in [m] \setminus \{i\}$, the new right-hand side b'_j is given by

$$b'_j = b_j - \frac{A_{jk} \cdot b_i}{A_{ik}} \ . \tag{1.19}$$

If $A_{jk} \leq 0$, then we have $b'_j \geq b_j \geq 0$ as desired. Otherwise, if $A_{jk} > 0$, then, by the fact that A_{ik} is a legal pivot, we have

$$\frac{b_i}{A_{ik}} \leq \frac{b_j}{A_{jk}} \ ,$$

which, together with (1.19), again implies $b'_j \geq 0$, as desired.

(ii) When computing an exchange step, the row corresponding to the objective is treated in the same way as the other non-pivot rows. Hence, to compute the new objective value q', the same pivot rule applies as the one used to compute the new right-hand side of a non-pivot row:

$$q' = q - \frac{c_k \cdot b_i}{A_{ik}} \geq q \ , \tag{1.20}$$

where the inequality follows from $c_k < 0$ (legal pivot), $b_i \geq 0$ (feasibility of original tableau), and $A_{ik} > 0$ (legal pivot).

(iii) The above reasoning also immediately implies that $q' > q$ if $b_i > 0$. □

Notice that the conditions for a pivot element to be legal, i.e., Definition 1.66, leave significant freedom in terms of which pivot to choose. Indeed, different rules of which legal pivot to choose lead to different variations of the Simplex Method. One very common rule is to choose the pivot column k such that c_k is the most negative value among all columns. This choice guarantees the largest *marginal increase* in the objective in the sense of how the objective changes per unit of increase of the non-basic variable corresponding to the pivot column.

Moreover, it may be that no legal pivot element exists. This can happen either because $c_k \geq 0$ for $k \in [n]$, or because for every column $k \in [n]$ with $c_k < 0$, there is no row $i \in [m]$ with $A_{ik} > 0$. In this case, the statement below implies that either the current basic solution is optimal or the problem is unbounded.

Theorem 1.68

Consider a feasible tableau (given as in (1.18)).

(i) If $c_k \geq 0\ \forall k \in [n]$, then the basic solution corresponding to (1.18) is an *optimal solution* to the corresponding LP.

(ii) If $\exists k \in [n]$ with $c_k < 0$ and $A_{jk} \leq 0\ \forall j \in [m]$, then the underlying LP is unbounded.

Moreover, a certificate of unboundedness can be obtained as follows. For $\lambda \in \mathbb{R}_{\geq 0}$, let x_B^λ, x_N^λ be the following assignments to the basic and non-basic variables, respectively:

$$\begin{aligned} x_B^\lambda &= b - \lambda A_{\cdot k} \\ (x_N^\lambda)_k &= \lambda \\ (x_N^\lambda)_\ell &= 0 \quad \forall \ell \in [n] \setminus \{k\} \ , \end{aligned}$$

where $A_{\cdot k}$ is the k-th column of A and $(x_N^\lambda)_k$ is the value of the non-basic variable corresponding to that column. Then, for any $\lambda \in \mathbb{R}_{\geq 0}$, $x^\lambda := \binom{x_B^\lambda}{x_N^\lambda}$ is feasible with objective value going to ∞ as $\lambda \to \infty$.

Proof.

(i) We can read off from the tableau that the objective value z satisfies

$$z = q - c^\top x_N \ .$$

Notice that the above expression is always at most q because $c \geq 0$ and $x_N \geq 0$. Hence, no feasible solution can have objective value strictly larger than q. Because the basic solution for the current feasible tableau has objective value equal to q, it is optimal.

(ii) To show this point, we only have to show that the claimed certificate of unboundedness is correct. To this end, we first observe that for any $\lambda \in \mathbb{R}_{\geq 0}$, the point x^λ is a solution by construction, i.e., it fulfills all equality constraints. Moreover, x^λ is non-negative. This is obvious for the non-basic variables, and follows for the basic variables by observing that $b \geq 0$, due to feasibility of the tableau, and $A_{\cdot k} \leq 0$ by assumption. Finally, the objective value achieved by x^λ is

$$z = q - \lambda c_k \ ,$$

which indeed goes to ∞ for $\lambda \to \infty$. □

Theorem 1.68 covers the case where no legal pivot element exists anymore. Nevertheless, Theorem 1.68 may apply even though there still is a legal pivot element in the tableau. More precisely, there may be a column $k \in [n]$ satisfying point (ii) of Theorem 1.68, even though a legal pivot element exists in a different column. The fact that Theorem 1.68 also applies in this case is convenient, because it allows us to first fix a column k with $c_k < 0$ and then only focus on that column for legal pivots. As we will see, this is precisely what the Simplex Method does.

Theorem 1.68 naturally leads to the following terminology.

Definition 1.69: Optimal and unbounded tableaus

A feasible tableau (as in 1.18) is called *optimal* if $c_k \geq 0\ \forall k \in [n]$. It is called *unbounded* if $\exists k \in [n]$ with $c_k < 0$ and $A_{jk} \leq 0\ \forall j \in [m]$.

Example 1.70

To exemplify the above statements, we start with the tableau below and perform legal pivots as long as possible.

The first tableau below is a feasible starting tableau, and we choose x_2 as the variable entering the basis, i.e., we want to pivot on the column corresponding to x_2. The corresponding quotients for this column are indicated to the right of the tableau.

	x_1	x_2	1	quotient $\frac{b_j}{A_{jk}}$		
z	-400	-900	0			
y_1	1	$\boxed{4}$	40	10	←	minimum
y_2	2	1	42	42		
y_3	1.5	3	36	12		

After performing an exchange step with respect to the indicated pivot, we obtain the tableau below, which, due to Theorem 1.67 (i), is feasible because the pivot was legal.

	x_1	y_1	1	quotient		
z	$-\frac{700}{4}$	$\frac{900}{4}$	9000			
x_2	$\frac{1}{4}$	$\frac{1}{4}$	10	40		
y_2	$\frac{7}{4}$	$-\frac{1}{4}$	32	128/7		
y_3	$\boxed{\frac{3}{4}}$	$-\frac{3}{4}$	6	8	←	minimum

The current feasible basic solution is

$$(x_1, x_2, y_1, y_2, y_3) = (0, 10, 0, 32, 6)$$

with objective value $z = 9000$. Notice that the objective value strictly increased from the value of the original tableau, which was 0. The strict increase in objective value is guaranteed by Theorem 1.67 (iii) in this case.

We continue with a pivot on the circled element to obtain the optimal tableau below.

$$
\begin{array}{c|cc|r}
 & y_3 & y_1 & 1 \\
\hline
z & \frac{700}{3} & 50 & 10400 \\
\hline
x_2 & -\frac{1}{3} & \frac{1}{2} & 8 \\
y_2 & -\frac{7}{3} & \frac{6}{4} & 18 \\
x_1 & \frac{4}{3} & -1 & 8
\end{array}
$$

The corresponding basic solution of this tableau, which is optimal due to Theorem 1.68 (i), is

$$(x_1, x_2, y_1, y_2, y_3) = (8, 8, 0, 18, 0)$$

with optimal objective value $z = 10400$. Again, the strict increase in objective value compared to the previous tableau was guaranteed by Theorem 1.67 (iii).

We are now ready to formally state phase II of the Simplex Method.

Algorithm 1: Phase II of Simplex Method

Input: Feasible tableau as shown in (1.18).

1. **Choice of pivot:**
   - (a) *Choice of pivot column* (variable entering basis):
     If $c_k \geq 0 \; \forall k \in [n]$, then **stop**. (The current basic solution is optimal due to Theorem 1.68 (i).) Otherwise, choose $k \in [n]$ with $c_k < 0$.
   - (b) *Choice of pivot row* (variable leaving basis):
     If $A_{jk} \leq 0 \; \forall j \in [m]$, then **stop**. (The problem is unbounded, see Theorem 1.68 (ii).) Otherwise, choose

$$i \in \operatorname{argmin} \left\{ \frac{b_j}{A_{jk}} : j \in [m] \text{ with } A_{jk} > 0 \right\} \quad .$$

2. **Exchange step:**
   Perform an exchange step on pivot element A_{ik} and **go back** to step 1.

As previously mentioned, the above description leaves open which pivot column or row to choose if there are several options fulfilling the above conditions. One of the most important questions in the field is whether there exists a good way to select the pivot column and row in the Simplex Method to obtain a polynomial-time algorithm. Interestingly, depending on how the pivot column and row are chosen in the Simplex Method, it is not even clear whether the algorithm terminates. We will get back to that point in Section 1.3.8.

Example 1.71: Phase II of Simplex Method

In this example, we apply phase II of the Simplex Method to solve the following LP given in canonical form.

$$\begin{array}{rrcrcl}
\max & -3x_1 & + & 2x_2 & & \\
 & x_1 & - & x_2 & \leq & 1 \\
 & -2x_1 & + & x_2 & \leq & 1 \\
 & -x_1 & + & x_2 & \leq & 2 \\
 & x_1 & & & \geq & 0 \\
 & & & x_2 & \geq & 0
\end{array}$$

We first transform it into standard form by introducing three slack variables y_1, y_2, and y_3.

$$\begin{array}{rcccccrcrcl}
\max & z = & & & & - & 3x_1 & + & 2x_2 & & \\
 & & y_1 & & & + & x_1 & - & x_2 & = & 1 \\
 & & & y_2 & & - & 2x_1 & + & x_2 & = & 1 \\
 & & & & y_3 & - & x_1 & + & x_2 & = & 2 \\
 & & & & & & & & x & \in & \mathbb{R}^2_{\geq 0} \\
 & & & & & & & & y & \in & \mathbb{R}^3_{\geq 0}
\end{array}$$

We now apply phase II of the Simplex Method. For this example, there is a unique legal pivot element at each step, which is circled in the tableaus below.

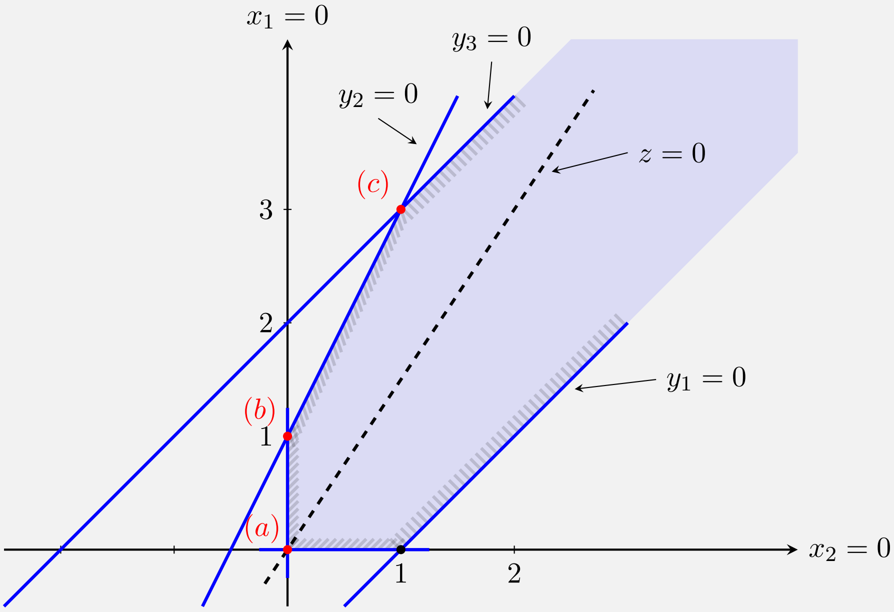
tableau (a) tableau (b) tableau (c)

$$\begin{array}{c|rr|r}
 & x_1 & x_2 & 1 \\ \hline
z & 3 & -2 & 0 \\ \hline
y_1 & 1 & -1 & 1 \\
y_2 & -2 & \boxed{1} & 1 \\
y_3 & -1 & 1 & 2
\end{array}
\quad \rightarrow \quad
\begin{array}{c|rr|r}
 & x_1 & y_2 & 1 \\ \hline
z & -1 & 2 & 2 \\ \hline
y_1 & -1 & 1 & 2 \\
x_2 & -2 & 1 & 1 \\
y_3 & \boxed{1} & -1 & 1
\end{array}
\quad \rightarrow \quad
\begin{array}{c|rr|r}
 & y_3 & y_2 & 1 \\ \hline
z & 1 & 1 & 3 \\ \hline
y_1 & 1 & 0 & 3 \\
x_2 & 2 & -1 & 3 \\
x_1 & 1 & -1 & 1
\end{array}$$

The last tableau is an optimal tableau because the coefficients of the non-basic variables in the objective row, namely two 1s, are both non-negative. Hence, the corresponding basic solution is optimal. It is

$$(x_1, x_2, y_1, y_2, y_3) = (1, 3, 3, 0, 0) \ .$$

The graphic below highlights, in the (x_1, x_2)-space, the vertices of the feasible region that got traversed by this run of the Simplex Method.



If we replace the objective function and maximize $\overline{z} = x_2$ instead of $-3x_1 + 2x_2$, then the algorithm will stop with the tableau below.

tableau (c')

	y_3	y_2	
$\overline{z}$	2	-1	3
y_1	1	0	3
x_2	2	-1	3
x_1	1	-1	1

This tableau is unbounded. As highlighted in Theorem 1.68, we can read off an unbounded half-line as follows. The half-line starts at the current basic solution, which is

$$\begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 3 \\ 0 \\ 0 \end{pmatrix} .$$

Moreover, the unbounded improving direction can be read off the column corresponding to y_2, and is

$$\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} .$$

Hence, the following improving half-line, described in the space $(x_1, x_2, y_1, y_2, y_3)$, is an unboundedness certificate:

$$\begin{pmatrix}1\\3\\3\\0\\0\end{pmatrix} + \lambda \cdot \begin{pmatrix}1\\1\\0\\1\\0\end{pmatrix} \qquad \text{for } \lambda \in \mathbb{R}_{\geq 0} \enspace .$$

By only considering the first two coordinates, we obtain an unboundedness certificate in the original (x_1, x_2)-space:

$$\begin{pmatrix}1\\3\end{pmatrix} + \lambda \cdot \begin{pmatrix}1\\1\end{pmatrix} \qquad \text{for } \lambda \in \mathbb{R}_{\geq 0} \enspace .$$

Exercise 1.72

Verify that one can indeed obtain tableau (c') in Example 1.71 through pivoting steps when starting with the modified objective $\max \overline{z} = x_2$.

Degeneracy

One of the most central questions we left open so far, is whether the Simplex Method terminates in a finite number of steps. It turns out that this is closely related to the notion of degeneracy, which we already saw in the context of polyhedra, and can be rephrased in terms of tableaus and basic solutions as follows.

Definition 1.73

The basic solution to a tableau (given as in 1.18) is called *degenerate* if there is an index $i \in [m]$ such that $b_i = 0$. In this case, the tableau is also called *degenerate*.

This notion of degeneracy, defined in the context of tableaus, is indeed tightly linked to the notion of degeneracy that we encountered in the context of polyhedra, as highlighted in the following statement.

Proposition 1.74

A feasible tableau is degenerate if and only if its basic solution is a degenerate vertex of the feasible region of the problem (with respect to both the canonical or standard form).

Proof. Assume that

$$\begin{array}{rrcl} \max & c^\top x & & \\ & Ax & \leq & b \\ & x & \geq & 0 \end{array} \tag{1.21}$$

is the LP we started with in canonical form, and

$$\begin{array}{rrcrcl} \max & c^\top x & & & & \\ & Ax & + & y & = & b \\ & x & & & \geq & 0 \\ & & & y & \geq & 0 \end{array} \tag{1.22}$$

is the standard form of the LP we used to build the tableau. Now let $(\overline{x}, \overline{y})$ be the basic solution to a feasible tableau, and let η be the total number of zeros in $\overline{x}$ and $\overline{y}$. Notice that the number of tight constraints among the original constraints in canonical form, i.e., $Ax \leq b$ and $x \geq 0$, is equal to η. Hence, $\overline{x}$, which is a vertex of

$$P := \{x \in \mathbb{R}^n \colon Ax \leq b, x \geq 0\} \enspace ,$$

is degenerate if and only if $\eta > n$. Notice that in any basic solution, all n-many non-basic variables are set to zero. Hence, we have $\eta > n$ if and only if at least one basic variable is set to zero. This is the case if and only if the tableau is degenerate.

Finally, we highlight that the same statement also holds when considering the point $\left(\begin{smallmatrix}\overline{x}\\ \overline{y}\end{smallmatrix}\right)$ as a vertex of

$$Q := \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^{n+m} \colon Ax + y = b, x \geq 0, y \geq 0 \right\} \enspace ,$$

i.e., the feasible region for the LP in standard form. Indeed, the number of tight constraints for $\left(\begin{smallmatrix}\overline{x}\\ \overline{y}\end{smallmatrix}\right)$ in the description of Q is equal to $m + \eta$, which leads to degeneracy in the $(n+m)$-dimensional space $\mathbb{R}^{n+m}$ if and only if $\eta > n$, as in the canonical form. □

We now observe that without degeneracy, any realization of phase II of the Simplex Method, as described in Algorithm 1, will be successful, i.e., it will terminate. We recall that when talking about a "realization" of the algorithm, we mean a specific way to choose the pivot column and row in case of multiple options.

Theorem 1.75: Finiteness of Simplex Method in non-degenerate case

If phase II of the Simplex Method never encounters a degenerate tableau, then it will stop after a finite number of steps. (This holds no matter how the pivot column and row is chosen if there are several legal options.)

Proof. Whenever phase II of the Simplex Method does an exchange step, the value of the tableau strictly improves because of Theorem 1.67 (iii). (Notice that non-degeneracy of the tableau guarantees that the right-hand side b_i of the pivot row is strictly positive, which is required by Theorem 1.67 (iii).) This implies that we will never encounter the same tableau twice. However, for any given problem, there is only a finite number of possible tableaus, because there are only finitely many options to partition all variables into a basic and non-basic group. Hence, the algorithm must terminate. □

Thus, if the inequality description $Ax \leq b, x \geq 0$ of an LP in canonical form is not degenerate, then phase II of the Simplex Method will always terminate in a finite number of steps. This is a consequence of Theorem 1.75 together with Proposition 1.74.

To summarize what we discussed so far, if phase II of the Simplex Method never encounters a degenerate tableau, then it will stop, which implies that we will either have found an optimal solution or shown that the problem is unbounded. However, to obtain a working algorithm for solving general linear programs, we also need to be able to handle the case of degenerate basic solutions.

1.3.8 Cycling and Bland's rule

Even though, due to Theorem 1.75, we know that phase II of the Simplex Method will stop if there is no degeneracy, it is not yet clear whether degeneracy is actually a problem for the algorithm or whether it is merely an issue for the analysis we employed. It turns out that degeneracy can indeed lead to Simplex Methods that do not terminate. Notice that if the Simplex Method does not terminate, then it will visit some tableaus an infinite number of times. This is called *cycling*. The terminology of cycling is even more perspicuous when considering a Simplex Method with a deterministic rule for choosing the pivot row and column that does not terminate. In this case, the first time the same tableau is encountered twice, the method will keep cycling through the precise same exchange steps performed between these two encounters.

We start with an example that highlights the issue of cycling in degenerate tableaus.

Example 1.76: Cycling of Simplex Method

Consider the following linear program.

$$
\begin{array}{lccccccccccc}
\max\ z = & & & 2x_3 & + & 2x_4 & - & 8x_5 & - & 2x_6 & & \\
& & x_2 & - 7x_3 & - & 3x_4 & - & 7x_5 & + & 2x_6 & = & 0 \\
& x_1 & & + 2x_3 & + & x_4 & - & 3x_5 & - & x_6 & = & 0 \\
& & & & & & & & & x & \in & \mathbb{R}^6_{\geq 0}
\end{array}
$$

We will apply phase II of the Simplex Method to it with the following rule for choosing a pivot element. When there are several options for choosing the pivot column or row, we always choose the leftmost or topmost one, respectively. This leads to the following tableaus.

(a)

	x_3	x_4	x_5	x_6	1
z	-2	-2	8	2	0
x_2	-7	-3	7	2	0
x_1	$\boxed{2}$	1	-3	-1	0

(b)

	x_1	x_4	x_5	x_6	1
z	1	-1	5	1	0
x_2	$\frac{7}{2}$	$\boxed{\frac{1}{2}}$	$-\frac{7}{2}$	$-\frac{3}{2}$	0
x_3	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{3}{2}$	$-\frac{1}{2}$	0

(c)

	x_1	x_2	x_5	x_6	1
z	8	2	-2	-2	0
x_4	7	2	-7	-3	0
x_3	-3	-1	$\boxed{2}$	1	0

(d)

	x_1	x_2	x_3	x_6	1
z	5	1	1	-1	0
x_4	$-\frac{7}{2}$	$-\frac{3}{2}$	$\frac{7}{2}$	$\boxed{\frac{1}{2}}$	0
x_5	$-\frac{3}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0

(e)

	x_1	x_2	x_3	x_4	1
z	-2	-2	8	2	0
x_6	-7	-3	7	2	0
x_5	$\boxed{2}$	1	-3	-1	0

(f)

	x_5	x_2	x_3	x_4	1
z	1	-1	5	1	0
x_6	$\frac{7}{2}$	$\boxed{\frac{1}{2}}$	$-\frac{7}{2}$	$-\frac{3}{2}$	0
x_1	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{3}{2}$	$-\frac{1}{2}$	0

(g)

	x_5	x_6	x_3	x_4	1
z	8	2	-2	-2	0
x_2	7	2	-7	-3	0
x_1	-3	-1	2	1	0

After six exchange steps, we again find a tableau with basis $B = (x_2, x_1)$. This is not the precise same tableau as (a) because the order of the columns is different. However, one can check that the next six exchange steps will choose the precise same entering and leaving variables as the first six and result in the original tableau (a). Hence, the tableaus will keep changing in this cyclic order with the employed pivot rule.

Another quick way to see that the above example leads to cycling, is by observing that the tableau (e) has the precise same entries as tableau (a), only the boundary is different. Because our pivoting rule chooses the pivot elements only depending on the entries and not the boundaries, we will obtain a tableau with identical entries after every 4 exchange steps. Hence, we will never find an optimal or unbounded tableau in this example. Thus, the method cycles.

One of the easiest pivot selection rules to avoid cycling is *Bland's rule*, which is defined as follows.

Definition 1.77: Bland's pivot rule

All variables are first ordered in an arbitrary way (e.g., according to increasing index). Whenever a pivot column or row is to be selected and there are several options, we choose the column or row corresponding to the variable appearing first in the fixed order.

Hence, if we are given a tableau with variable $x_1, \ldots, x_k$ and apply Bland's rule (using the order induced by the indices), then, among all possible pivot columns, we select the one whose corresponding variable has smallest index, and do the same in the next step for selecting the pivot row.

Theorem 1.78

When applying Bland's rule for choosing the pivot, phase II of the Simplex Method does not cycle. More precisely, with Bland's rule, the Simplex Method will never encounter two tableaus with the same set of basic (and therefore also non-basic) variables.

Example 1.79

Using Bland’s rule in Example 1.76 leads to a different pivot choice in tableau (f):

$$(f)\quad \begin{array}{c|cccc|c} & x_5 & x_2 & x_3 & x_4 & 1 \\ \hline z & 1 & -1 & 5 & 1 & 0 \\ \hline x_6 & \frac{7}{2} & \frac{1}{2} & -\frac{7}{2} & -\frac{3}{2} & 0 \\ x_1 & \frac{1}{2} & \boxed{\frac{1}{2}} & -\frac{3}{2} & -\frac{1}{2} & 0 \end{array}$$

This results in the following optimal tableau:

$$\begin{array}{c|cccc|c} & x_5 & x_1 & x_3 & x_4 & 1 \\ \hline z & 2 & 2 & 2 & 0 & 0 \\ \hline x_6 & 3 & -1 & -2 & -1 & 0 \\ x_2 & 1 & 2 & -3 & -1 & 0 \end{array}$$

Notice that both in Example 1.76 and also here we always pivoted on a row whose right-hand side was zero. Consequently, all tableaus we encountered have identical basic solutions. (Though they have different bases.) Hence, in this special case, the basic solution of tableau (a) was already optimal. (Even though the tableau is not an optimal one!) However, it is not hard to modify the example such that after breaking out of the cycling with Bland’s rule, one can still find tableaus with strictly higher value through further exchange steps.

Proof of Theorem 1.78. Assume for the sake of obtaining a contradiction that we cycle with Bland’s rule. Recall that Bland’s rule is deterministic, hence we will always cycle in the same order through the same tableaus. Some variables will sometimes be basic and sometimes non-basic, whereas others may always be either basic or non-basic. We first observe that we can assume that no variables of the latter type exist, i.e., no variable will always be basic or always be non-basic. Indeed, consider one tableau we encounter when cycling, and assume that such a variable exists, say an always basic one. Then we can simply delete the row corresponding to the variable from the tableau and obtain a smaller tableau where Bland’s rule cycles. Clearly, because the pivot elements in the cycle are never chosen on the row corresponding to the deleted basic variable, the selection of the pivot elements does not change. Analogously, we can delete from a tableau any column corresponding to a variable that is always non-basic. Hence, we can assume that we have a cycling example for Bland’s rule where each variable is sometimes basic and sometimes non-basic in the tableaus encountered in the cycle.

An important observation is that when cycling, the basic solution never changes. Because each variable is at least once non-basic on the cycle, and hence has value 0 in the corresponding basic solution, this implies that the basic solution to all tableaus encountered in the cycle is the all-zeros vector. In particular, the right-hand side of any tableau encountered in the cycle consists of zeros only. Moreover, the constant entry in the objective function row always remains the same; hence, we can assume without loss of generality that it is zero.

Let $x_1, \dots, x_k$ be all variables involved in the tableau, and the numbering is chosen such that Bland's rule always chooses the variable with the smallest index if there are several options. Now consider a tableau in the cycle in which x_k is basic and is chosen as the basis-leaving variable, i.e., the pivot element is chosen in the row of x_k. We denote this tableau as tableau (a) and let x_i be the basis-entering variable. Figure 1.12 highlights the signs of the entries in the column corresponding to x_i.

$$(a) \qquad \begin{array}{c|c|c}
 & x_i & \\
\hline
z & - & 0 \\
\hline
 & \ominus & 0 \\
 & \vdots & \vdots \\
 & \ominus & 0 \\
x_k & + & 0 \\
 & \ominus & 0 \\
 & \vdots & \vdots \\
 & \ominus & 0
\end{array}$$

Figure 1.12: Tableau (a) shown above is a tableau encountered during cycling where, in the next exchange step, the basis-leaving variable is x_k. We denote by x_i the basis-entering variable. As discussed, all right-hand sides are 0. Moreover, we highlight the signs of the values in the column corresponding to x_i. Here, '$-$' stands for a strictly negative number, '$+$' for a strictly positive one, and '$\ominus$' for a non-positive one. The signs follow by the fact that we choose a legal pivot element, which implies the '$+$' and '$-$' signs, and because we apply Bland's rule, which implies in this case that no other choice than x_k for basis-leaving variable was possible, because x_k is the variable with the largest index and all quotients are 0. Due to this, all the '$\ominus$' signs above follow.

Additionally, there is a tableau in the cycle where x_k is non-basic but will enter the basis during the next exchange step. We denote this tableau as tableau (b).

$$(b) \qquad \begin{array}{c|ccccccc|c}
 & & & & x_k & & & & \\
\hline
z & \oplus & \cdots & \oplus & - & \oplus & \cdots & \oplus & 0 \\
\hline
 & & & & & & & & 0 \\
 & & & & & & & & \vdots \\
 & & & & & & & & 0
\end{array}$$

Figure 1.13: Tableau (b) shown above is a tableau encountered during cycling where, in the next exchange step, the basis-entering variable is x_k. Because Bland's rule chooses x_k as last option when selecting a basis-entering variable, this implies that it is the only non-basic variable with a strictly negative entry in the objective row, as highlighted above.

Consider now the following point $d \in \mathbb{R}^k$, where we think of the k coordinates of d as values for $(x_1, x_2, \ldots, x_k)$. All entries of d are zero except for d_i and for the entries corresponding to variables in tableau (a) that are basic. We set $d_i = 1$ and then set the entries corresponding to the basic variables of tableau (a) to the unique values such that d is a solution. Due to the sign pattern highlighted in Figure 1.12, we have

(i) $d_k < 0$,
(ii) $d_j \geq 0 \; \forall j \in [k-1]$, and
(iii) the objective value of the solution d is strictly better than the all-zeros solution because the objective coefficient of x_i in tableau (a) is strictly negative.

Even though this is not crucial for this proof, we highlight that d is not feasible because $d_k < 0$.

Note that we can evaluate the objective value of the solution d by plugging its values into the objective row of any equivalent tableau, which must lead to the same value. However, if we evaluate the objective value of d by plugging its values into the objective row of tableau (b), we obtain—due to the sign pattern of d and of the objective row of tableau (b) as highlighted in Figure 1.13—that its objective value is strictly worse than the one of the all-zeros solution. This contradicts point (iii) above and finishes the proof. □

Practical experience shows that Bland's pivot selection rule often significantly increases the running time of the Simplex Method. On the other hand, countless practical examples also show that the Simplex Method with the simple pivot rule that chooses the basis-entering variable (pivot column) of largest marginal increase very rarely gets stuck in a degeneracy. Consequently, many common implementations of the Simplex Method mainly follow the latter rule and only deviate from it if necessary. One way to get a practical pivot rule that is guaranteed not to get stuck is to only change to a different rule that does not cycle, like Bland's rule, if another rule that is typically faster, like the one based on marginal improvements, does not strictly improve the objective value during a predefined number of steps.

It is important to know that degeneracy is not a rare phenomenon. In particular, as we will see later, most classical combinatorial optimization problems can naturally be rephrased as highly degenerate linear programs. In these problems, degeneracy appears very naturally and is not just an artefact of a badly-chosen inequality description with redundant constraints that lead to degeneracy.

1.3.9 Simplex Method: phase I

Phase II of the Simplex Method assumes that we start with a feasible tableau, i.e., we know a basic feasible solution. We now consider the problem of obtaining a basic feasible solution. This can be achieved with what is called *phase I of the Simplex Method*, which is a technique to reduce the problem back to the one solved by phase II of the Simplex Method. More precisely, in phase I, an auxiliary problem is defined for which we know a trivial basic feasible solution and, by optimizing the auxiliary problem, we either obtain a feasible basis for the original problem—and can then start phase II with that basis—or learn that the original problem is infeasible.

Consider a general LP in canonical form for which we want to find a basic feasible solution (of the corresponding problem in standard form).

$$\begin{array}{rrcl} \max & c^\top x & & \\ & Ax & \leq & b \\ & x & \in & \mathbb{R}^n_{\geq 0} \ , \end{array} \tag{1.23}$$

where, as usual, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$.

We construct the following auxiliary problem, which uses one new auxiliary variable x_0; moreover, we denote by $\mathbf{1}$ the all-ones vector in $\mathbb{R}^m$.

$$\begin{array}{rrcl} \max & -x_0 & & \\ & Ax - \mathbf{1} \cdot x_0 & \leq & b \\ & x & \in & \mathbb{R}^n_{\geq 0} \\ & x_0 & \in & \mathbb{R}_{\geq 0} \end{array} \tag{1.24}$$

Notice that problem (1.24) is always feasible. Actually, one can even fix $x \in \mathbb{R}^n$ arbitrarily and still obtain a feasible solution by choosing $x_0 \geq 0$ large enough. In particular, x_0 can be thought of as allowing a violation in the original constraints. By maximizing $-x_0$, we aim at minimizing this violation. This leads to the following relation between the two problems.

Observation 1.80

LP (1.23) is feasible $\Leftrightarrow$ LP (1.24) has optimal value 0.

We now show the following two points:

(i) One can easily determine a feasible basis for the auxiliary LP (1.24) such that phase II of the Simplex Method can be applied to it, and

(ii) if the auxiliary LP has optimal value zero, then an optimal tableau for it allows for reading off a feasible starting basis for the original LP, which can be used to solve the original problem with phase II of the Simplex Method.

To apply phase II of the Simplex Method to the auxiliary problem (1.24), we first write it in standard form by introducing the slack variables $x_s \in \mathbb{R}^m$:

$$\begin{array}{rrcl} \max & -x_0 & & \\ & x_s + Ax - \mathbf{1} \cdot x_0 & = & b \\ & x & \in & \mathbb{R}^n_{\geq 0} \\ & x_0 & \in & \mathbb{R}_{\geq 0} \\ & x_s & \in & \mathbb{R}^m_{\geq 0} \ , \end{array} \tag{1.25}$$

and get the corresponding tableau, where we use $\tilde{z} = -x_0$ as the variable for the auxiliary objective function:

$$\begin{array}{c|c|r|c} & x & x_0 & 1 \\ \hline \tilde{z} & 0 & 1 & 0 \\ \hline & & -1 & \\ x_s & A & -1 & b \\ & & -1 & \end{array} \tag{1.26}$$

We assume that b is not non-negative; for otherwise, there is no need to apply phase I of the Simplex Method because one could simply right-away apply phase II with the slack variables as basis. Hence, tableau (1.26) is not feasible. However, it is easy to transform the above tableau into a feasible one through the following well-chosen exchange step.

(i) Choose x_0 as the variable entering the basis, i.e., as the pivot column.
(ii) Choose as basis-leaving variable a row with most negative b-value, i.e., a row with index $i \in \operatorname{argmin}\{b_\ell \colon \ell \in [m]\}$.

Note that in the second step above, we can indeed choose a row with negative b-value because we assumed that (1.26) is not feasible. Hence, $b_i < 0$.

Proposition 1.81

Performing an exchange step on tableau (1.26) using a pivot element as described above leads to a feasible tableau.

Proof. To verify that the new tableau, after performing an exchange step on a pivot element as described above, is feasible, we have to check that the new right-hand side b' is non-negative. For this, let $\overline{A} = [A \ \ -\mathbf{1}]$ be the constraint matrix of the auxiliary problem (1.24), which includes the negative all-ones column corresponding to x_0. Let $\overline{A}_{ik}$ be a pivot element as described above. In particular, $k = n+1$.

The new right-hand side b' is expressed by the pivot rules (1.14). Notice that $\overline{A}_{ik} = -1$. Hence, the new right-hand side b'_i of the pivot row is indeed non-negative:

$$b'_i = \frac{b_i}{\overline{A}_{ik}} = -b_i > 0 \ .$$

Moreover, also for any non-pivot row $j \in [m]\setminus\{i\}$, we have non-negativity of the new right-hand side:

$$b'_j = b_j - \frac{\overline{A}_{jk} \cdot b_i}{\overline{A}_{ik}} = b_j - b_i \geq 0 \ ,$$

where the second equality is due to $\overline{A}_{jk} = \overline{A}_{ik} = -1$, and the inequality follows from our choice of the pivot row. □

Hence, Proposition 1.81 provides the last missing step to solve the auxiliary problem with phase II of the Simplex Method. Moreover, the optimal tableau of the auxiliary problem can be used to solve the original problem with phase II of the Simplex Method. We show how this can be done in the example below.

Example 1.82

We start with the following linear program in canonical form. Notice that there are two constraints with strictly negative right-hand sides. Hence, we cannot start phase II of the Simplex Method by transforming this problem into standard form and then using the slack variables as basis.

$$\begin{array}{lrcrcrcr} \max & x_1 & - & x_2 & + & x_3 & & \\ & 2x_1 & - & x_2 & + & 2x_3 & \leq & 4 \\ & 2x_1 & - & 3x_2 & + & x_3 & \leq & -5 \\ & -x_1 & + & x_2 & - & 2x_3 & \leq & -1 \\ & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

We now start with phase I of the Simplex Method and build the auxiliary problem.

$$\begin{array}{lrcrcrcrcr} \max & & & & & & - & x_0 & & \\ & 2x_1 & - & x_2 & + & 2x_3 & - & x_0 & \leq & 4 \\ & 2x_1 & - & 3x_2 & + & x_3 & - & x_0 & \leq & -5 \\ & -x_1 & + & x_2 & - & 2x_3 & - & x_0 & \leq & -1 \\ & & & & & & & x_0, x_1, x_2, x_3 & \geq & 0 \end{array}$$

Again, we use $\widetilde{z} = -x_0$ for the auxiliary objective function, and write the problem in tableau form:

	x_1	x_2	x_3	x_0	1
$\tilde{z}$	0	0	0	1	0
x_4	2	−1	2	−1	4
x_5	2	−3	1	$\boxed{-1}$	−5
x_6	−1	1	−2	−1	−1

This tableau is not feasible, however it will turn feasible after performing an exchange step on the above-explained pivot element. More precisely, the pivot element, which is circled in the table above, is in the column of the auxiliary variable x_0 and in the row of the most negative b_i entry. After performing the exchange step, the following tableau is obtained, which is feasible by choice of the pivot element.

	x_1	x_2	x_3	x_5	1
$\tilde{z}$	2	−3	1	1	−5
x_4	0	2	1	−1	9
x_0	−2	3	−1	−1	5
x_6	−3	$\boxed{4}$	−3	−1	4

Starting from the tableau above, we perform phase II of the Simplex Method. The chosen pivot elements are circled.

	x_1	x_6	x_3	x_5	1
$\tilde{z}$	−0.25	0.75	−1.25	0.25	−2
x_4	1.5	−0.5	2.5	−0.5	7
x_0	0.25	−0.75	$\boxed{1.25}$	−0.25	2
x_2	−0.75	0.25	−0.75	−0.25	1

	x_1	x_6	x_0	x_5	1
$\tilde{z}$	0	0	1	0	0
x_4	1	1	-2	0	3
x_3	0.2	-0.6	0.8	-0.2	1.6
x_2	-0.6	-0.2	0.6	-0.4	2.2

(1.27)

The tableau above is optimal with value 0. Hence, the original linear program is feasible. Moreover, the basic solution to the above tableau, which is

$$(x_0, x_1, x_2, x_3, x_4, x_5, x_6) = (0, 0, 2.2, 1.6, 3, 0, 0) \ ,$$

corresponds to a feasible solution to the original problem by dropping the x_0 coordinate, i.e.,

$$(x_1, x_2, x_3, x_4, x_5, x_6) = (0, 2.2, 1.6, 3, 0, 0) \ .$$

Moreover, the basis of (1.27) is a feasible basis for the original LP. Indeed, by simply removing the x_0 column from tableau (1.27), a tableau for the original problem is obtained, modulo the fact that the objective function $\tilde{z}$ is the auxiliary one and not the original one. We can obtain the original objective, expressed in the basis (x_4, x_3, x_2), as follows. By setting $x_0 = 0$ in tableau (1.27), we obtain the relations

$$\begin{array}{rcrcrcrcrl} x_3 & = & -0.2x_1 & + & 0.6x_6 & + & 0.2x_5 & + & 1.6 & \text{, and} \\ x_2 & = & 0.6x_1 & + & 0.2x_6 & + & 0.4x_5 & + & 2.2 & . \end{array}$$

By plugging in the above expressions for x_2 and x_3 into the original objective function $z = x_1 - x_2 + x_3$, we get the objective in the new basis (x_4, x_3, x_2):

$$\begin{array}{rrcrcrcrcrl} z = & x_1 & - & (0.6x_1 & + & 0.2x_6 & + & 0.4x_5 & + & 2.2) & \\ & & + & (-0.2x_1 & + & 0.6x_6 & + & 0.2x_5 & + & 1.6) & \\ = & & & 0.2x_1 & + & 0.4x_6 & - & 0.2x_5 & - & 0.6 & . \end{array}$$

Hence, the tableau for the original problem in the basis (x_4, x_3, x_2) is the following:

	x_1	x_6	x_5	1
z	-0.2	-0.4	0.2	-0.6
x_4	1	1	0	3
x_3	0.2	-0.6	-0.2	1.6
x_2	-0.6	-0.2	-0.4	2.2

Starting from this *feasible tableau*, we can now use phase II of the Simplex Method to solve the original problem.

Remark 1.83

Note that the optimal tableau obtained after solving the auxiliary problem, i.e., tableau (1.27) in Example 1.82, may be such that x_0 is a basic variable of value 0, due to degeneracy. In this case, the basis of the auxiliary problem, which contains x_0, is clearly not a feasible basis of the original one, where no x_0 variable exists. However, whenever this happens, one can simply perform an additional exchange step on any non-zero pivot element in the row of x_0, to obtain another optimal tableau where x_0 is non-basic.
Observe that, whenever x_0 is basic, there must always be a non-zero element in the row corresponding to x_0. For otherwise, all solutions to the auxiliary problem would have the same x_0 value. However, this is not true, because there are feasible solutions for arbitrarily high values of x_0.

Finally, we highlight that one can avoid the reconstruction of the original objective function after obtaining an optimal tableau for phase I of the Simplex Method by carrying along the original objective function in the tableau. More precisely, when constructing the tableau for the auxiliary LP for phase I of the Simplex Method, one can add one additional row corresponding to the original objective z as follows:

$$
\begin{array}{rc|c|c|c|cl}
 & & & \overset{\text{auxiliary column}}{\downarrow} & & & \\
 & & x & x_0 & 1 & & \\ \cline{2-5}
\text{auxiliary row} \rightarrow & \widetilde{z} & 0 & 1 & 0 & \leftarrow & \text{objective function for phase I} \\ \cline{2-5}
\text{objective function} \rightarrow & z & -c & 0 & 0 & \leftarrow & \text{objective function for phase II} \\ \cline{2-5}
 & & & -1 & & & \\
 & x_s & A & \vdots & b & & \\
 & & & -1 & & & \\ \cline{2-5}
\end{array}
$$

Starting from this tableau, we continue as before. More precisely, we first perform an exchange step to obtain a feasible tableau where x_0 is a basic variable. We then perform phase II of the Simplex Method with the special rule that we never choose a pivot on the row corresponding to z. More precisely, the row corresponding to z is ignored for choosing pivot elements or for determining whether the tableau is feasible. Nevertheless, whenever the pivot element is fixed and an exchange step is performed, the row corresponding to z is treated like any other row.

This way, the original objective remains expressed in the current non-basic variables throughout phase I of the Simplex Method. Hence, when phase I of the Simplex Method terminates with an optimal tableau of value 0, and with x_0 being non-basic, then it suffices to delete the column corresponding to x_0 and the row corresponding to $\widetilde{z}$ to obtain a feasible tableau for the original problem. From this tableau, we can then start phase II of the Simplex Method.

1.4 Linear duality

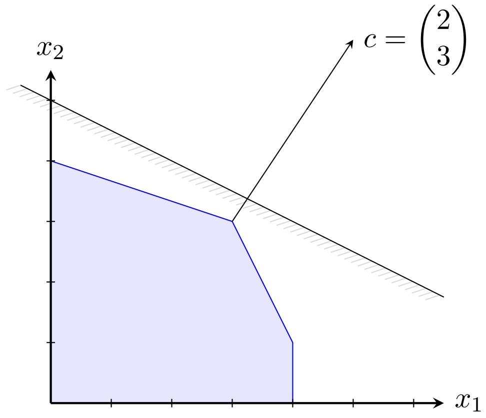
It turns out that for every linear program there exists a closely related linear program, called the *dual problem*. The dual program can be seen as a different way to look at the original problem, which is typically referred to as the *primal*. The close relation between primal and dual can be exploited in several ways. In particular, the dual provides quick ways for showing optimality of an optimal primal solution. Moreover, given an optimal solution for either the primal or the dual often allows for quickly obtaining an optimal solution to the other problem. This connection can sometimes be used to find an optimal solution to the primal more quickly. More precisely, sometimes it is faster to use for example the Simplex Method to solve the dual problem and then derive from an optimal dual solution an optimal primal one. Duality is also key in the context of sensitivity analysis, and has various algorithmic applications, for example in so-called *primal-dual* procedures, which solve a problem by simultaneously exploiting properties of the primal and dual of a problem.

In the next section, we introduce linear duality as a way to find strong upper bounds for the optimal value of a linear program.

1.4.1 Motivation: finding bounds on optimal value

Consider the following linear program in canonical form. Its feasible region is highlighted below on the right-hand side together with the objective function, which is maximized at the point $(x_1, x_2) = (3, 3)$.

$$\begin{array}{rrcrcl}
\max & 2x_1 & + & 3x_2 & & \\
 & x_1 & + & 3x_2 & \leq & 12 \\
 & 2x_1 & + & x_2 & \leq & 9 \\
 & x_1 & & & \leq & 4 \\
 & x_1 & + & 2x_2 & \leq & 10 \\
 & x_1 & & & \geq & 0 \\
 & & & x_2 & \geq & 0
\end{array}$$



Assume we want to find an upper bound on the optimal value of the above linear program. One way to obtain such a bound is by multiplying the constraint $x_1 + 2x_2 \leq 10$ by 2 to obtain

$$2x_1 + 4x_2 \leq 20 \ .$$

Because any solution to the LP satisfies $x_1, x_2 \geq 0$, we have

$$2x_1 + 3x_2 \leq 2x_1 + 4x_2 \leq 20 \ .$$

Hence, 20 is an upper bound on the optimal LP value. An obvious questions is whether we can obtain a better bound by a similar technique. Indeed, by adding up the constraints $x_1+3x_2 \leq 12$ and $x_1 \leq 4$ we get

$$2x_1 + 3x_2 \leq 16 \ ,$$

showing that the optimal LP value is upper bounded by 16. In general, to obtain a valid bound for the objective $2x_1 + 3x_2$, one can try to multiply each '$\leq$'-constraint by some non-negative factor such that a linear constraint $\alpha x_1 + \beta x_2 \leq \gamma$ is obtained with $\alpha \geq 2$, $\beta \geq 3$. Then γ is an upper bound to the optimal value of the LP.

Hence, finding the best such upper bound is itself an LP where we have a variable for each constraint of the original LP, except for the non-negativity constraints. Those variables describe the coefficients in the conic combination of the constraints. Furthermore, there is a constraint for each variable in the original LP. For the example above it looks as follows.

$$\begin{array}{rrcrcrcrcl}
\min & 12y_1 & + & 9y_2 & + & 4y_3 & + & 10y_4 & & \\
 & y_1 & + & 2y_2 & + & y_3 & + & y_4 & \geq & 2 \\
 & 3y_1 & + & y_2 & & & + & 2y_4 & \geq & 3 \\
 & y_1 & & & & & & & \geq & 0 \\
 & & & y_2 & & & & & \geq & 0 \\
 & & & & & y_3 & & & \geq & 0 \\
 & & & & & & & y_4 & \geq & 0
\end{array}$$

This problem is called the *dual problem* to the original problem. The unique optimal solution of this dual problem is $(y_1, y_2, y_3, y_4) = (\frac{4}{5}, \frac{3}{5}, 0, 0)$ and leads to an optimal value of 15. The way how we set up this dual problem implies that 15 is an upper bound to the value of the original LP. As it turns out, the optimal value of the dual LP is even equal to the optimal value of the original LP, which is achieved at the point $(3, 3)$. This is no coincidence. It is at the heart of one of the most important results in linear programming, namely strong duality.

In the following, we start by making the above discussion on constructing the dual more formal before covering some key results linked to linear duality, like strong duality.

1.4.2 Dual of a linear program

Let us revisit the above discussion in more generality. Consider a general linear program in canonical form, which we will call the *primal problem* to distinguish it later from the *dual problem*:

$$\begin{array}{rrcl}
\max & c^\top x & & \\
 & Ax & \leq & b \\
 & x & \geq & 0 \ .
\end{array} \tag{PLP}$$

Following the above discussion, its corresponding dual problem is:

$$\begin{array}{rrcl}
\min & b^\top y & & \\
 & A^\top y & \geq & c \\
 & y & \geq & 0 \ .
\end{array} \tag{DLP}$$

Notice that for each constraint in the primal there is a variable in the dual and vice versa. In particular, if the primal problem has n variables and m constraints (not counting the non-negativity constraints) then the dual problem has m variables and n constraints (not counting the non-negativity constraints).

Dualization of LPs not in canonical form

So far, we focussed on dualizing linear programs in canonical form. However, dualitzation can be applied to a linear program in an arbitrary form. Of course, one can simply bring an arbitrary linear program first into canonical form and then dualize it. However, one can also dualize any linear program directly, without first transforming it into canonical form. For example, if we have a constraint of type '$\geq$' in the primal, then we can use a dual variable for this constraint that is non-positive instead of non-negative.

Recall that if we first transform a problem into canonical form, then this can lead to extra variables and/or constraints, which leads to extra constraints and/or variables in the dual. Nevertheless, one can simplify the resulting dual to avoid the introduction of extra variables or constraints.

In the problem sets, we will expand on the above discussion about dualizing linear programs that are not in canonical form. In what follows, we focus on the case of primal problems that are in canonical form.

Dual of dual is primal

An interesting property of linear duality is that the dual of the dual is the primal problem. One way to see this is to first transform (DLP) into canonical form:

$$\begin{array}{rrcl} -\max & -b^\top y & & \\ & -A^\top y & \leq & -c \\ & y & \geq & 0\ , \end{array} \tag{1.28}$$

and then dualizing the above problem, which leads to

$$\begin{array}{rrcl} -\min & -c^\top x & & \\ & -Ax & \geq & -b \\ & x & \geq & 0\ , \end{array}$$

which is again the primal problem (PLP). As discussed, it is not necessary to first transform (DLP) into (1.28) to dualize it. One can immediately dualize it by again deriving how one can bound the objective value $b^\top y$ with the available constraints.

1.4.3 Weak and strong linear duality

Notice that by the way how we constructed the dual problem (DLP) from the primal problem (PLP), the value of any feasible solution to the dual upper bounds the value of any feasible solution to the primal. This result is known as *weak duality* and is easy to prove formally.

Theorem 1.84: Weak duality

Let x, y be feasible solutions to (PLP) and (DLP), respectively. Then

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

Proof. The result follows from

$$\underbrace{c^\top}_{\leq y^\top A} x \leq y^\top \underbrace{Ax}_{\leq b} \leq y^\top b = b^\top y \ ,$$

where the first inequality holds due to $c \leq A^\top y$ and $x \geq 0$, and the second inequality follows from $Ax \leq b$ and $y \geq 0$. □

Weak duality comes already with some interesting implications. It implies that if the primal is unbounded, then the dual must be infeasible. Since the primal-dual relation is symmetric, it also implies that if the dual is unbounded then the primal is infeasible. Furthermore, if one can find a primal feasible solution x and a dual feasible solution y such that $c^\top x = b^\top y$, then both x and y are optimal solutions for the primal and dual, respectively. Hence, in this case, the dual solution is an optimality certificate of the primal solution and vice versa.

Notice that weak duality does not rule out that both the primal and dual are infeasible, which is indeed possible.

Example 1.85: Infeasible primal and dual LP

Below is a pair of a primal and dual linear program that are both infeasible.

$$\begin{array}{rrcrcr}
\max & 2x_1 & - & x_2 & & \\
 & x_1 & - & x_2 & \leq & 1 \\
 & -x_1 & + & x_2 & \leq & -2 \\
 & x_1 & & & \geq & 0 \\
 & & & x_2 & \geq & 0
\end{array}
\qquad\qquad
\begin{array}{rrcrcr}
\min & y_1 & - & 2y_2 & & \\
 & y_1 & - & y_2 & \geq & 2 \\
 & -y_1 & + & y_2 & \geq & -1 \\
 & y_1 & & & \geq & 0 \\
 & & & y_2 & \geq & 0
\end{array}$$

Notice that the existence of such an example is not surprising due to the following. Consider two independent linear programs in canonical form, one that is infeasible and one for which the dual is infeasible. Then we can simply combine them into a single linear program maximizing the sum of both objectives. It is not hard to check that this leads to a new linear program for which both the primal and dual are infeasible.

Interestingly, when the primal and dual are both feasible—which is often the most interesting case—then a cornerstone result in linear duality implies that their optimal values are equal. This result is known as *strong duality*.

Theorem 1.86: Strong duality

If the primal has a finite optimum, then also the corresponding dual problem has a finite optimum and their values are equal, i.e., there exists a feasible solution x for (PLP) and a feasible solution y for (DLP) such that $c^\top x = b^\top y$.

We provide a proof of Theorem 1.86 later in Section 1.4.4, after having discussed the dual interpretation of the simplex tableau, which allows us to simultaneously make statements about the primal and dual problem using the same tableau.

Strong duality implies that if an LP has a finite optimum, then the optimality of a solution to that LP can *always* be certified by exhibiting a dual solution with the same value. Moreover, strong duality has numerous implications in various combinatorial optimization problems. In particular, most max-min relations in Combinatorial Optimization, like the max-flow min-cut theorem, can be interpreted as a consequence of strong duality.

We recall that every linear program either (i) has a finite optimum, (ii) is unbounded, i.e., allows for solutions with arbitrarily strong objective values, or (iii) is infeasible. The table below shows in which of those 3 states an LP and its corresponding dual can be, where a check mark (✓) indicates that the corresponding combination is possible and a cross (✗) that it is not.

			dual	
		finite	unbounded	infeasible
primal	finite	✓	✗	✗
	unbounded	✗	✗	✓
	infeasible	✗	✓	✓

The above table is symmetric because the dual of the dual is the primal. The first row and first column follow from strong duality. The second row and column follow from weak duality. Finally, the check mark at the bottom right follows from the fact that there are pairs of primal and dual LPs that are both infeasible (see Example 1.85).

1.4.4 Dual interpretation of simplex tableau

We now discuss how the simplex tableau for the dual problem is tightly related to the one of the primal problem. More precisely, by reading the tableau differently, one can interpret the primal tableau as one for the dual problem. To highlight this connection, consider again the dual problem (DLP) in canonical form, i.e.,

$$\begin{array}{rrcl} \max & -b^\top y & & \\ & -A^\top y & \leq & -c \\ & y & \geq & 0\ , \end{array}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. Introducing the objective function variable $w = b^\top y$ for the dual and the dual slack variables $y^s \in \mathbb{R}^n$, we write the dual problem in standard form:

$$\begin{array}{rrrcr} \max & -w & & = & -b^\top y \\ & & y^s \quad -A^\top y & = & -c \\ & & y, y^s & \geq & 0\ . \end{array}$$

Given an initial primal tableau for the primal, with objective $z = c^\top x$ and slack variables x^s, we can directly read the dual relations by reading the tableau column-wise with the equality sign next to the variables:

$$\begin{array}{c|c|c} & x\ \ (y^s) & 1\ \ (w) \\ \hline z\ \ (1) & -c^\top & 0 \\ \hline x^s\ \ (y) & A & b \end{array} \quad \begin{array}{c} (\shortparallel) \\ (+) \end{array}$$

Here, x^s and y^s denote the primal and dual slack variables, respectively. When reading the tableau column-wise, we interpret the boundary of the tableau differently, as highlighted by the parenthetical expressions in the tableau above:

(i) The role of the primal variables x is replaced by the dual slack variables y^s.
(ii) The primal slack variables x^s are replaced by the dual variables y.
(iii) The primal objective z takes the role of the dual constant 1.
(iv) The primal constant column 1 takes the role of the dual objective w.

This so-called *dual reading* or *dual interpretation* leads to:

$$\begin{array}{rcrcl} y^s & = & -c & + & A^\top y \\ w & = & 0 & + & b^\top y\ , \end{array}$$

which indeed corresponds to the dual constraints in standard form. In particular, this way of reading uses the following one-to-one correspondence between the primal slack variables and the dual variables:

Relation between primal/dual variables and slacks

x_j^s	primal slack variable	$\longleftrightarrow$	y_j	dual variable
x_i	primal variable	$\longleftrightarrow$	y_i^s	dual slack variable

A crucial fact is that the dual reading method works for any primal tableau, i.e., it always leads to an equivalent equation system for the dual. Hence, any primal tableau can be interpreted as a dual tableau when using dual reading. This key property, which readily leads to a proof of the strong duality theorem, is formally stated below.

Lemma 1.87

For any primal tableau, the dual reading of the tableau leads to an equivalent equation system for the dual (including the objective function).

Proof. To set notation, assume that the original primal problem in standard form is described as usual by:

$$\begin{array}{rrcrcl} \max & z & & & = & c^\top x \\ & & x^s & + \quad Ax & = & b \\ & & & x, x^s & \geq & 0 \ , \end{array} \tag{1.29}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. We recall that the corresponding dual problem in standard form is given by:

$$\begin{array}{rrcrcl} \max & -w & & & = & -b^\top y \\ & & y^s & -A^\top y & = & -c \\ & & & y, y^s & \geq & 0 \ . \end{array} \tag{1.30}$$

Now consider a primal tableau for (1.29) (with respect to an arbitrary basis):

$$\begin{array}{c|c|c} & x_N & 1 \\ \hline z & -\overline{c}^\top & \overline{q} \\ \hline x_B & \overline{A} & \overline{b} \end{array} \ , \tag{1.31}$$

where x_B and x_N are just names for the vectors of basic and non-basic variables, respectively, in the above tableau. Our task is to show that by dual reading the tableau (1.31), one obtains an equation system for the dual problem that is equivalent to the one in (1.30). Notice that the dual equation system obtained from dual reading (1.31) clearly has full rank because it is in tableau form. Hence, one only has to show that each of the dual constraints stemming from (1.31) is implied by the equations in (1.30), i.e., it is a linear combination of the equations in (1.30).

Now consider an arbitrary column of (1.31); this can either be one of the n columns corresponding to the non-basic variables x_N or the last column, which corresponds to the constant 1. Let us call the considered column the *selected column*. Notice that any such column corresponds to a dual equation when using dual reading. To represent the entries of the selected column in a way that we can naturally interpret both in the primal and the dual, we define a vector in the space $Q := \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}$, where we interpret the $m + n + 2$ coordinates of the space Q as follows:

(i) The first entry corresponds to the primal objective z and thus to the dual constant term.
(ii) The next m entries correspond to the primal slacks x^s and thus to the dual variables y.
(iii) The next n entries correspond to the primal variables x and thus to the dual slacks y^s.
(iv) The last entry corresponds to the primal constant term and thus to the dual objective w.

Hence, a primal solution with values $x \in \mathbb{R}^n$ for the original primal variables, $x^s \in \mathbb{R}^m$ for the primal slack variables, and $z \in \mathbb{R}$ for the objective is represented by the vector

$$\begin{pmatrix} z \\ x^s \\ x \\ 1 \end{pmatrix} \in Q \ . \tag{1.32}$$

Because this is just another way to represent a primal solution, we also call the above vector a *primal solution*.

Now let $\eta \in Q$ be the vector defined as follows:

(i) If the selected column corresponds to a variable of x_N, then we set the η-entry that corresponds to this variable to -1. Otherwise, if the selected column corresponds to the constant column, then we set the η-entry that corresponds to the constant term to 1.

(ii) For any row of the tableau (1.31), which either corresponds to z or a variable in x_B, we set the corresponding entry of η to the value that the selected column has in this row.

(iii) All other entries of η are set to zero.

We now observe a few properties of the vector η. First, by reading the tableau (1.31) in the primal way, we see that adding or subtracting η to any primal solution given in the space Q, as shown in (1.32), another primal solution (in space Q) is obtained. Notice that whether z, x^s, and x is a primal solution can be rephrased as follows:

$$z \in \mathbb{R}, x^s \in \mathbb{R}^m, x \in \mathbb{R}^n \text{ is a primal solution} \quad \Leftrightarrow \quad \begin{pmatrix} 1 & 0^\top & -c^\top & 0 \\ 0 & I & A & -b \end{pmatrix} \cdot \begin{pmatrix} z \\ x^s \\ x \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \ .$$

Hence, this implies

$$\begin{pmatrix} 1 & 0^\top & -c^\top & 0 \\ 0 & I & A & -b \end{pmatrix} \cdot \eta = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \ .$$

Thus, η is in the null space of $\left(\begin{smallmatrix} 1 & 0^\top & -c^\top & 0 \\ 0 & I & A & -b \end{smallmatrix}\right)$. This null space is spanned by the columns of

$$\begin{pmatrix} c^\top & 0 \\ -A & b \\ I & 0 \\ 0 & 1 \end{pmatrix} ,$$

which implies that there exists some $\lambda \in \mathbb{R}^{n+1}$ such that

$$\eta = \begin{pmatrix} c^\top & 0 \\ -A & b \\ I & 0 \\ 0 & 1 \end{pmatrix} \cdot \lambda \ . \tag{1.33}$$

Moreover, observe that the dual equation stemming from dual reading the tableau (1.31) is given by

$$\eta^\top \begin{pmatrix} 1 \\ y \\ y^s \\ -w \end{pmatrix} = 0 \ . \tag{1.34}$$

To show that the above constraint is implied by the equations in (1.30), we have to show that any dual solution $y \in \mathbb{R}^m$, $y^s \in \mathbb{R}^n$, and $w \in \mathbb{R}$ fulfills (1.34). Observe that the fact of y, y^s, and w being a dual solution, i.e., fulfilling the equations of (1.30), can be written in matrix form as follows:

$$\begin{pmatrix} c & -A^\top & I & 0 \\ 0 & b^\top & 0^\top & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ y \\ y^s \\ -w \end{pmatrix} = 0 \ . \tag{1.35}$$

Finally, we conclude that (1.34) indeed holds for any dual solution y, y^s, w because

$$\eta^\top \begin{pmatrix} 1 \\ y \\ y^s \\ -w \end{pmatrix} = \lambda^\top \begin{pmatrix} c & -A^\top & I & 0 \\ 0 & b^\top & 0^\top & 1 \end{pmatrix} \begin{pmatrix} 1 \\ y \\ y^s \\ -w \end{pmatrix} = 0 \ ,$$

where the first equality follows from (1.33) and the second one from (1.35). □

Lemma 1.87 allows for providing a quick proof of the strong duality theorem of linear programming, i.e., Theorem 1.86. Indeed, the dual interpretation of the simplex tableau, together with Proposition 1.87, show that whenever we have an optimal (primal) tableau, it also reveals an optimal dual solution. This fact is at the heart of the proof below.

Proof of Theorem 1.86. Assume that we are given a linear (primal) program with a finite optimum. Now consider an optimal tableau for the problem. Notice that such a tableau exists because first there is a feasible tableau due to feasibility of the problem. (Indeed, phase I of the Simplex Method implies this.) Then there is also an optimal tableau because of phase II of the Simplex Method which, when run with Bland's rule, must either return an optimal tableau or an unbounded direction, which does not exist because the primal optimum is finite. In an optimal tableau the row corresponding to the primal objective is a non-negative vector. Hence, when reading the tableau using dual reading, we obtain a feasible basic dual solution. Moreover, because the objective value obtained through dual reading is the same as through primal reading, we obtain that the dual basic solution y for the optimal tableau satisfies $b^\top y = c^\top x$, where x is the primal basic solution of the optimal tableau. Hence, by weak duality, x and y are optimal primal and dual solutions, respectively. □

Example 1.88

Consider the following (primal) linear program in canonical form.

$$\begin{array}{rrcrcrcrcl}
\max & 4x_1 & + & x_2 & + & 5x_3 & + & 3x_4 & & \\
 & x_1 & - & x_2 & - & x_3 & + & 3x_4 & \leq & 1 \\
 & 5x_1 & + & x_2 & + & 3x_3 & + & 8x_4 & \leq & 55 \\
 & -x_1 & + & 2x_2 & + & 3x_3 & - & 5x_4 & \leq & 3 \\
 & & & & & \multicolumn{3}{r}{x_1, x_2, x_3, x_4} & \geq & 0
\end{array}$$

Below is an optimal tableau that corresponds to the above LP.

	$x_1\,(y_1^s)$	$x_3\,(y_3^s)$	$x_1^s\,(y_1)$	$x_3^s\,(y_3)$	$1\,(w)$
$(1)\,z$	1	2	11	6	29
$(y_2^s)\,x_2$	2	4	5	3	14
$(y_4^s)\,x_4$	1	1	2	1	5
$(y_2)\,x_2^s$	-5	-9	-21	-11	1

Because the tableau is optimal, the basic primal solution, i.e.,

$$(x_1, x_2, x_3, x_4) = (0, 14, 0, 5) \ ,$$

is an optimal primal solution to the LP, and the basic dual solution of the above tableau, i.e.,

$$(y_1, y_2, y_3) = (11, 0, 6) \ ,$$

which is obtained by dual reading the tableau, is an optimal dual solution. Both solutions are obtained from the primal and dual reading, respectively, when setting the corresponding non-basic variables to zero. Notice that if a slack variable x_i^s (y_i^s) is in the optimal basis, then the corresponding dual variable y_i (x_i) is non-basic and therefore equal to zero. This relationship is known as *complementary slackness* and will be discussed in more detail in Section 1.4.5.

Remark 1.89: Notation of primal & dual variables and slacks

Due to the strong correspondence between the primal (dual) slack variables and the dual (primal) variables, both variables are often denoted by the same symbol. This simplifies the dual reading, but also involves the risk of mistaking the primal slack variables for the dual variables and vice versa. For instance, let us consider the following tableau, corresponding to a primal problem in canonical form:

	x	1
z	$-c^\top$	0
y	A	b

In the primal interpretation, x denotes the variables and y the slack variables, whereas in the dual interpretation, x denotes the slack variables and y the variables. Moreover, z is the primal objective but dual constant, whereas 1 is the primal constant but dual objective.

Moreover, all terminology we introduced for (primal) tableaus, like degeneracy, optimality, or feasibility, can be defined analogously with respect to the dual. More precisely, given a primal tableau, we call it *dual feasible* if its corresponding dual basic solution (obtained through dual reading) is feasible, i.e., all entries in the objective row are non-negative. It is called *dual degenerate* if at least one of the objective row entries is zero. Finally, there is no difference between the notions of a primal and dual optimal tableau.

1.4.5 Complementary slackness

The complementary slackness theorem formalizes a crucial relation between optimal primal and dual solutions, which we already observed in Example 1.88.

Theorem 1.90: Complementary slackness theorem

Consider a pair of primal and dual linear programs with finite optima:

$$\begin{array}{rcl} \max\ c^\top x & & \\ Ax & \leq & b \\ x & \geq & 0 \end{array} \qquad\qquad \begin{array}{rcl} \min\ b^\top y & & \\ A^\top y & \geq & c \\ y & \geq & 0\ . \end{array}$$

Let $\overline{x}$ be a feasible primal solution and $\overline{y}$ a feasible dual solution. Then both $\overline{x}$ and $\overline{y}$ are optimal solutions (for the primal and dual, respectively) if and only if

(i) $(b - A\overline{x})^\top \overline{y} = 0$, and
(ii) $(A^\top \overline{y} - c)^\top \overline{x} = 0$.

Proof. Consider again the chain of relations that we used to derive the weak duality theorem:

$$c^\top \overline{x} \ \leq\ \overline{y}^\top A\overline{x} \ \leq\ \overline{y}^\top b\ , \tag{1.36}$$

which is based on the relations $c \leq A^\top \overline{y}$, $A\overline{x} \leq b$, and $\overline{x}, \overline{y} \geq 0$. Now, by strong duality, $\overline{x}$ and $\overline{y}$ are optimal if and only if $c^\top \overline{x} = b^\top \overline{y}$, i.e., the left-hand side and right-hand side of (1.36) are equal. This happens if and only if both inequalities in (1.36) are equalities, which is equivalent to

(i) $(b - A\overline{x})^\top \overline{y} = 0$, and
(ii) $(A^\top \overline{y} - c)^\top \overline{x} = 0$,

as desired. □

As we will see in the problem sets, apart from providing an important insight on the relation between optimal primal and dual solutions, the complementary slackness theorem can often be used to derive an optimal dual solution from a primal one or vice versa.

2 Brief Introduction to Computational Complexity

Typically, there are many different ways to tackle a computational problem. Different approaches lead to different algorithms that solve the same problem. Consequently, one faces the task of comparing these algorithms with each other. What should be the criteria under which we evaluate an algorithm? Depending on the application, different criteria may be important: Sometimes *simplicity* is crucial to enable a quick and error-free implementation. For complex problems, a clean and easy-to-understand structure is often in the foreground: A mathematical language, mathematical tools, and a mathematical formalism should be used in such a way that complicated aspects are cleanly expressed in a comprehensible way. This avoids errors in implementation and interpretation. When the focus is on dealing with large problem instances, however, the *efficiency* of the algorithm is often decisive: we want to solve the problem with the least amount of computation. This leads us to the runtime analysis of algorithms. When we examine the running time of an algorithm, we want to know how much time it needs to solve a particular problem instance. We are interested in the dependence of this duration on the size of the problem instance. From a formal point of view, we have to be clear on what we mean by the "size" of a problem instance and how we measure the "running time" (or "runtime") of an algorithm. When we talk about a problem class in the context of Complexity Theory, we limit ourselves to instances of a problem for which the algorithm we are examining was designed. For example, for the problem of finding the largest among n numbers, a list of n numbers comprises a problem instance.

The size of a problem is the number of bits needed to store it. For example, an integer $a \in \mathbb{Z}_{\geq 0}$ can be saved in a standard representation with $\lceil \log_2(a+1) \rceil$ bits, where $\lceil q \rceil$ for $q \in \mathbb{R}$ represents the smallest integer $r \in \mathbb{Z}$ that satisfies $r \geq q$. The number 9 can therefore be represented as a binary code in the form `1001`. Because we often want to encode negative numbers, we can use an extra bit to encode the sign. More generally, a (possibly negative) integer $a \in \mathbb{Z}$ can be represented in $\lceil \log_2(|a|+1) \rceil + 1$ bits.

Unfortunately, it is difficult to give a good definition of the "runtime" of an algorithm. A first approach could be to implement the algorithm in a common programming language, run it on different sized problem instances and analyze the measured running times. However, such measurements are highly dependent on the underlying hardware, the programming language used, and many details linked to the implementation. But all these aspects have no relation to the efficiency of the algorithm that we actually want to measure. For the running time of an algorithm, a much less fine-grained measure is employed in Complexity Theory, with the goal of achieving independence of the above-described factors: The running time of an algorithm is measured by counting *elementary operations*. Elementary operations include additions, subtractions, multiplications, divisions, and comparisons of two numbers. In particular, we want to measure the number of elementary operations performed depending on the size of the problem instance, which we refer to as $\langle\text{input}\rangle$ and which represents the number of bits used to encode input.

Example 2.1: Maximum element

Consider the problem of finding the largest number in a given list $a_1, \ldots, a_n \in \mathbb{Z}_{\geq 0}$ of numbers. One possible algorithm that solves this problem is as follows: Start by comparing a_1 and a_2 and store the larger of the two numbers as x, i.e., $x = \max\{a_1, a_2\}$. Compare x with a_3. If a_3 is the larger of the two numbers, override x with a_3, and so on. This algorithm performs $n - 1$ comparisons, so its running time is $n - 1$. Because we need at least n bits to store the numbers $a_1, \ldots, a_n$, the running time of this algorithm is linearly (upper) bounded by the size of the problem instance.
This example shows a situation typical of running time analyses: Often, one can describe the running time as a function of the number of given elements (in this case n), without having to look at the exact size of the input instance in binary encoding. If a polynomial bound of this type is possible on the number of elementary operations, we talk about a *strongly polynomial algorithm.*

In the example above, it was easy to specify an upper bound on the runtime of the algorithm. Note that the actual running time of an algorithm, i.e., the number of elementary operations, depends not only on the size of the concrete problem instance but also on the problem instance itself. For example, consider an algorithm that calculates the largest common divisor of two numbers $a_1, a_2 \in \mathbb{Z}_{>0}$. Such an algorithm can terminate much faster on some problem instances (e.g., if $a_1 = a_2$) than on others. Classical running time analysis in Complexity Theory focusses on so-called *worst case analysis*. That is, for each possible input size $s = \langle\text{input}\rangle$, we want to know an upper bound on the running time of the algorithm that holds for *any* instance of input size at most s. The runtime of an algorithm is thus captured by a function $f\colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$, such that every problem instance of input size at most s is solved in at most $f(s)$ elementary operations.

To further simplify the analysis and avoid dependencies on unimportant details, we are not interested in the exact number of elementary operations, but only in their *order*. More specifically, we want to determine the runtime up to a constant factor. For example, we call an algorithm *linear* if there is a constant $c > 0$ such that, for each problem instance, the algorithm needs at most $c \cdot \langle\text{input}\rangle$ elementary operations, where $\langle\text{input}\rangle$ is the size of the problem instance. Thus, for an algorithm to have linear running time, it does not matter how large the constant c is, as long as its runtime is bounded by $c \cdot \langle\text{input}\rangle$. To succinctly describe the order of a term without including constant factors, we use the Landau O notation. In this notation, the runtime $f(\langle\text{input}\rangle) \leq c \cdot \langle\text{input}\rangle$ of a linear algorithm is written as $f(\langle\text{input}\rangle) = O(\langle\text{input}\rangle)$, that of a quadratic algorithm as $f(\langle\text{input}\rangle) = O(\langle\text{input}\rangle^2)$. Analogously to the Landau O notation, which describes upper bounds up to constant factors, the notations Ω and Θ are commonly used for lower bounding expressions and relating expressions of the same order (up to constant factors), respectively. They are defined as follows.

Definition 2.2: Landau notation

Let f and g be two functions.

(i) We write $f = O(g)$ if and only if

$$\exists M > 0, c > 0 \quad \text{such that} \quad |f(s)| \leq c \cdot |g(s)| \quad \forall s \geq M \ .$$

(ii) We write $f = \Omega(g)$ if and only if $g = O(f)$.

(iii) We write $f = \Theta(g)$ if and only if $f = O(g)$ and $f = \Omega(g)$.

In words, $f = O(g)$ means that, for values of s that are large enough, $f(s)$ does not grow faster than $g(s)$, up to a constant factor. Similarly, $f = \Omega(g)$ means that, for s being large enough, $f(s)$ does not grow more slowly than $g(s)$, up to a constant factor. The combination of both properties, that is $f = \Theta(g)$, means that f and g grow equally fast, up to constant factors, for large values.

It turns out that subdividing algorithms into fast and slow algorithms, according to the criterion of whether their runtime can be bounded by a polynomial in the size of the input instance, is a natural way that proved to be highly influential both in theory and practice. As highlighted below, algorithms that run in polynomial time are called *efficient* algorithms and, equally importantly, this notion can be used to classify problems.

Definition 2.3: Polynomial algorithms and problems

An algorithm is *polynomial* or *efficient* if its running time $f(\langle \text{input} \rangle)$ is bounded by a polynomial in the size of the input, i.e., there is a polynomial g such that

$$f = O(g) \ .$$

A problem is *solvable in polynomial time* if it can be solved by a polynomial algorithm.

Example 2.4: Systems of linear equations

Consider a linear equation system of the form $Ax = b$ with $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$. Note that the special case of integer systems that we consider also covers the more general case of rational systems of equations, since each rational system of equations can be converted into an equivalent integer equation system by multiplying it by a common multiple of all occurring denominators. The size of the problem instance is approximately given by

$$\langle \text{input} \rangle \approx \sum_{i=1}^{m} \sum_{j=1}^{n} (\lceil \log_2(|A_{ij}| + 1) \rceil + 1) + \sum_{i=1}^{m} (\lceil \log_2(|b_i| + 1) \rceil + 1) \ . \qquad (2.1)$$

We do not write an actual equality in the above statement, because we normally cannot just write all encodings consecutively without additional overhead. Indeed, we have to make sure that we are able to separate the encodings of the involved entries. From a technical point of view, this is not hard to achieve, for example when using additional syntax for delimiters to

mark the beginning and end of each encoded entry. However, the number of these extra bits is typically small compared to the expression in (2.1). Because, as explained above, we are only interested in the order of magnitude of the running time as a function of the input size, constant factors do not play a role in the analysis. It is therefore sufficient to estimate the size of the problem instance up to a constant factor. In this case, we see that (2.1) satisfies this requirement. In addition, we emphasize that due to the double sum in (2.1) we have $\langle \text{input} \rangle \geq m \cdot n$.
The system $Ax = b$ can be solved through Gaussian elimination in $O(mn^2)$ elementary operations. In particular, this means that the number of elementary operations is restricted by a polynomial in $\langle \text{input} \rangle$, because $\langle \text{input} \rangle \geq m \cdot n$. It can also be shown that the encoding length of any matrix entry encountered during the calculations is upper bounded by a polynomial in $\langle \text{input} \rangle$. Hence, Gaussian elimination is an efficient algorithm for the solution of linear systems of equations.

We are particularly interested in the complexity of finding optimal solutions to mathematical optimization problems. When talking about the complexity of problems, one typically considers so-called *decision problems*. A decision problem is a problem that asks a yes-or-no question. For example, one could consider a linear program and ask whether its optimal value is at least some number q. This way, optimization problems can naturally be transformed into decision problems.

Complexity Theory divides decision problems into different classes. The following are the arguably two best known complexity classes:

- The class $\mathcal{P}$, which contains all polynomially-solvable problems (or the corresponding decision problems). We have seen that both finding the maximum in a list of given numbers (Example 2.1) and solving linear systems of equations (Example 2.4) are included in $\mathcal{P}$. For problems in $\mathcal{P}$, also large problem instances can very often be solved in a relatively short time.
- The class $\mathcal{NP}$, which contains all the decision problems for which the answer "yes" can be *verified* by a cleverly chosen polynomial-size *certificate* in polynomial time. The *certificate* is often a solution to the problem itself, but may also contain other (polynomially bounded) information. The idea of a certificate is that you can quickly convince yourself that a "yes" instance really is a "yes" instance. For example, consider the decision problem whether a number $a \in \mathbb{Z}_{>0}$ is *not* a prime. If a is not a prime, you could choose as a certificate a divisor $t \in \{2, \dots, a-1\}$ of a. In fact, given such a divisor t of a, we can convince ourselves very quickly (in polynomial time) that a is a "yes" instance, i.e., it is not a prime: it is enough to check that a is actually divisible by t. This algorithmic process of convincing yourself with a certificate is called *verification*. Hence, problems in $\mathcal{NP}$ allow for fast verification of "yes" instances, but that does not imply that a "no" instance can be quickly verified, nor does it imply that a problem class can be solved efficiently, i.e., that it lies in $\mathcal{P}$.
  Another example of a problem in $\mathcal{NP}$ is the so-called *knapsack problem*: Given is a set $[n]$ of n objects, where each object $i \in [n]$ has a weight $w_i \in \mathbb{Z}_{\geq 0}$ and a value $p_i \in \mathbb{Z}_{\geq 0}$, and a global weight bound $W \in \mathbb{Z}_{\geq 0}$. The task is to select a subset $I \subseteq [n]$ of the objects with total weight not exceeding W, i.e., $\sum_{i \in I} w_i \leq W$ and total value as large as possible,

i.e., it should maximize $\sum_{i\in I} p_i$. The natural decision version of this problem adds to the input of the problem a value $P \in \mathbb{Z}_{\geq 0}$ and asks whether there is a knapsack solution of value at least P. A knapsack solution of value at least P clearly is a certificate that the answer is “yes”. However, it is not clear how, in case of a “no”-instance, one could provide a quickly checkable proof showing that no knapsack solution with value at least P exists.

One may wonder, whether by moving from an optimization problem to its natural decision version, like in the knapsack example above, one may end up with a significantly easier problem. This is not the case. For example, assume we had an efficient algorithm for the decision version of the knapsack problem. Then we could apply binary search with respect to P to find a knapsack solution of largest possible value, thus solving the optimization problem. The range over which we have to apply binary search can be bounded by $[0, \sum_{i=1}^{n} p_i]$. Binary search needs a number of steps that is logarithmic in the width of this search interval, which leads to a number of steps that is linear in the input size. Hence, in particular, an efficient algorithm to solve the decision version of the knapsack problem can easily be transformed to efficiently solve the optimization version.

Intriguingly, it remains unknown whether the two complexity classes $\mathcal{P}$ and $\mathcal{NP}$ are different, i.e., whether $\mathcal{P} \neq \mathcal{NP}$. The question of whether $\mathcal{P} \neq \mathcal{NP}$ is one of the most famous open problems in Mathematics and Computer Science. It is one of the 7 Millennium Prize Problems, stated by the Clay Mathematics Institute, who awards a prize of 1 000 000 USD for its resolution.[1] The prevailing opinion of experts in the field is that $\mathcal{P} \neq \mathcal{NP}$.

Notice that we have $\mathcal{P} \subseteq \mathcal{NP}$, because a provably correct algorithm for a problem in $\mathcal{P}$ can be used to certify in polynomial time the correctness of a yes answer. The problems in $\mathcal{NP}$ are a very big problem class with simple and extremely complex problems. To better understand how difficult problems can be in $\mathcal{NP}$, a subclass of $\mathcal{NP}$ has been defined, the so-called *$\mathcal{NP}$-complete problems*. $\mathcal{NP}$-complete problems can be interpreted as the most difficult problems in $\mathcal{NP}$. In particular, it can be shown that if you can efficiently solve any single $\mathcal{NP}$-complete problem, you can efficiently solve all the problems in $\mathcal{NP}$, which would imply $\mathcal{P} = \mathcal{NP}$. The existence of $\mathcal{NP}$-complete problems is a very central and profound result in Complexity Theory that led to a significantly better understanding of the problems that the class $\mathcal{NP}$ captures. If $\mathcal{P} \neq \mathcal{NP}$ is true, then no $\mathcal{NP}$-complete problem can be solved efficiently. An obvious proof plan for $\mathcal{P} \neq \mathcal{NP}$ would thus start with an $\mathcal{NP}$-complete problem and show that it is not efficiently solvable.

It is crucial to understand that the number of solutions typically does not reflect the complexity of a problem. Many combinatorial problems have solution spaces that are exponentially large in the size of the input but can nevertheless be solved efficiently. One such example is sorting n different integers in ascending order. To algorithmically deal with a problem, it is crucial to know which complexity class the problem belongs to. In some cases, this is anything but simple—and there remain many interesting problem classes for which this question remains open.

Complexity Theory has strongly influenced research on algorithms. Especially for problems that are known to belong to difficult complexity classes (for example, $\mathcal{NP}$-complete problems),

[1] See https://www.claymath.org/millennium-problems for more information on the Millennium Prize Problems.

Complexity Theory gives clear indications that there are good chances that no polynomial algorithms exist to solve these problems. Hence, different approaches need to be explored. This is a central motivation of so-called *approximation algorithms*. These are efficient algorithms that return solutions of a guaranteed quality (in a well-defined sense) with respect to an optimal solution. Also, when we know from a complexity-theoretic point of view that a problem is very difficult to solve, this also justifies the use of *heuristic* algorithms, which typically do not come with any solution guarantee but are often very fast.

3 Basics on Graphs

Graphs are a central structure in Discrete Mathematics with numerous real-world applications. This comes at no surprise, as graphs and networks are ubiquitous in everyday life and can be used to model transport networks, social networks, telecommunications networks, and neural networks, just to name a few examples. By exploiting the underlying graph-theoretical structures, problems in such networks can often be solved fast even for enormously large instances.

The aim of this chapter is to provide a very brief introduction to some of the basics linked to graphs and networks. After some motivational examples, we first introduce terminology and notation commonly used in Graph Theory. Moreover, we talk about the most common data structures to store graphs. This allows us to understand the input size of a problem that involves graphs and also clarifies how much time certain simple graph operations take. Finally, we discuss one of the most basic graph algorithms, namely breadth-first search, and some of its implications. This first algorithmic example also helps us to combine and better understand the previously introduced concepts, and is a first non-trivial example of how to analyze a graph algorithm.

We would like to emphasize that the terms "graph" and "network" are used interchangeably in many contexts. Sometimes, however, a network is used to describe graphs with additional information, for example on its edges or vertices.

3.1 Some motivational examples

In this introductory section we describe some classical problems that can be modeled and solved using graphs.

Example 3.1: Road network

Five cities, A, B, C, D, and E, are linked by the road network shown in Figure 3.1. The cities are represented by *vertices*, the streets as connections between two vertices, which are also called *edges*. The edges are labeled with the respective distances (in km) between the connected cities.

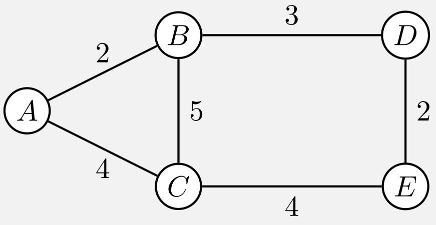


Figure 3.1: A road network represented as a graph.

One typical problem is to find the distance (i.e., the length of a shortest path) between any two cities as well as a corresponding path of that length. For example, the distance between city A and city E is 7 kilometers, and A-B-D-E is a possible path between A and E with this length.

The schematic representation of the road network in Figure 3.1, consisting of vertices and edges, is a *graph*. The edges here are labeled with numbers—representing the distances in the example— so we are talking about a *weighted graph*. If Example 3.1 contained a one-way street, we could illustrate this in the graph by adding an arrow pointing in the allowed direction. Such an edge is called *directed edge*, or simply *arc*. If every edge is directed in a graph, we are talking about a *directed graph*.

Example 3.2: Utility network

Between five locations, a supply network is to be set up (e.g., for water, gas, electricity, or internet access). The goal is that each pair of locations is connected, where a connection can go through other intermediate stations. In Figure 3.2, the locations are represented as vertices and the possible connections as edges between the vertices. The numbers on the edges correspond to the installation cost for the respective connection. The problem is to find a minimum cost set of edges that connect all locations. The highlighted edges in Figure 3.2 show an optimal solution for this particular example.

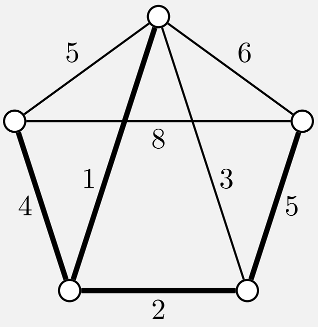


Figure 3.2: A utility network respresented as a graph.

Example 3.3: Assignment problem

A set of n jobs $\{j_1, \dots, j_n\}$ are to be completed by m workers $\{a_1, \dots, a_m\}$. Not every job can be completed by every worker: for each job, there is a given list of workers who are qualified to perform it. The question is whether it is possible to assign to each job one worker who is qualified for it. Each worker can be assigned to at most one job. Figure 3.3 shows a toy problem of this type with five jobs and six workers. The problem is represented as a graph. The top row of vertices corresponds to the jobs and the bottom one to the available workers. There is an edge between a job and a worker if and only if the job can be performed by the worker. The highlighted edges indicate a possible allocation assigning to each job one worker.

job	qualified workers
j_1	a_1, a_3, a_4
j_2	a_1, a_2, a_3, a_5
j_3	a_4, a_5, a_6
j_4	a_3, a_4
j_5	a_4, a_5, a_6

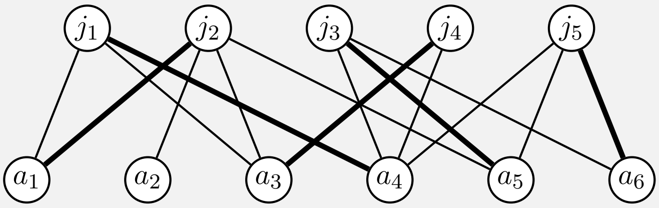


Figure 3.3: An assignment problem and its representation as a graph.

Example 3.4: Flow problem

In a system of water pipes there is a source s and a sink t. The system consists of junctions and pipes connecting these points. Each pipe can be used to let water flow in either one of the two directions. Moreover, each pipe has a maximum throughput, i.e., a maximum amount of water that can flow through the pipe per time unit. Junctions cannot store water. Hence, for each junction, the water inflow must be equal to its outflow. A canonical optimization problem in this setting is to determine the largest so-called s-t flow, which is the water flow—in terms of water per time unit—from s to t.

Figure 3.4 shows an example of a pipe network. The edges represent pipes and the vertices junctions, except for the two labelled vertices, which represent the source s and the sink t. The black numbers on the edges, i.e., the ones on the right-hand side, are the known maximum throughput rates for the pipes. This concrete example allows a maximum s-t flow of 17 units. The blue numbers, which are to the left of the throughput values, indicate a possible flow in the direction of the blue arrows that realizes this maximum.

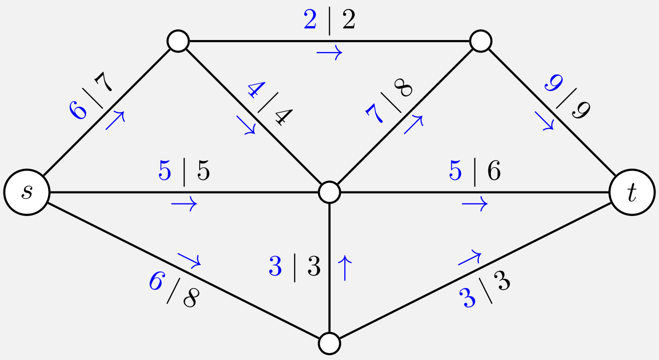


Figure 3.4: A flow problem together with a maximum s-t flow represented as a graph.

Example 3.5: Connection problem

In a telecommunications network, there is a central server in a network of locations. There are connections between the locations of different bandwidths. At certain stations are customers who would like to each have a connection of bandwidth 1 to the server. The task is to determine a maximum number of customers that can simultaneously be served by the server with a unit bandwidth each. The left part of Figure 3.5 shows such a problem where the server and locations are represented as vertices of a graph and the edges correspond to the connections. The numbers on the edges indicate the bandwidth of the respective connections. The tokens above vertices represent the clients. As highlighted on the right-hand side of Figure 3.5, a maximum of 4 clients can be connected to the server simultaneously.

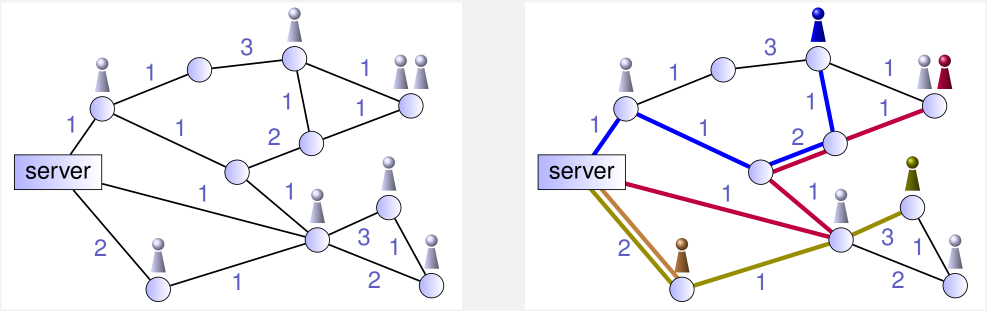


Figure 3.5: A connection problem and an optimal solution for it.

Example 3.6: Round trip

Our goal is to plan a road trip through the USA going through 20 given cities. The road trip should be a round trip, i.e., start and endpoint must coincide. The task is to determine a road trip that minimizes the driving time. The driving time from any of the 20 cities to any other one is known upfront. Figure 3.6 shows an example.

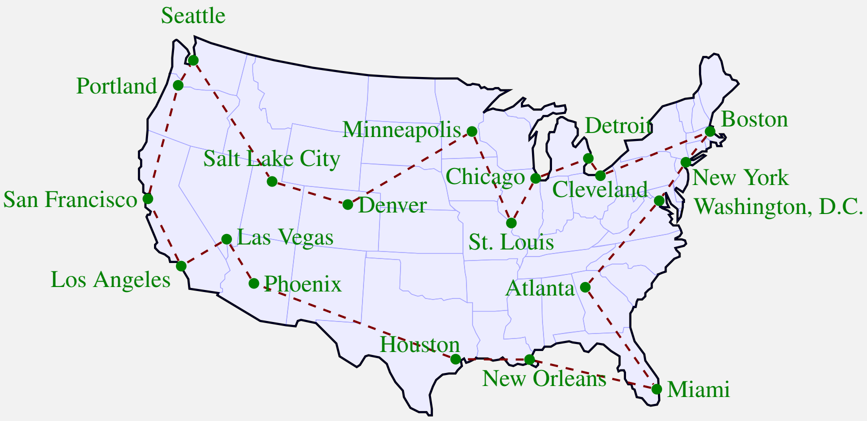


Figure 3.6: A possible road trip through 20 given cities in the USA.

Here, the problem is presented as a graph with 20 vertices, one per city, and all possible edges between these vertices. Each edge is assigned a value, representing the travel time between its endpoints. With this abstraction, the shortest round trip problem, also famously known as the Traveling Salesman Problem, can be formulated as follows: Find a shortest cycle in the graph that goes through all vertices.

Example 3.7: Drilling circuit boards

One of the steps in the production process of an electronic circuit board is the drilling of holes. This is performed by automatic drilling machines, whose drill head wanders over the circuit board and drills one hole after the other. In order to optimize the time required for this task, the distance traveled by the drill head should be minimized. This problem is very similar to the round trip problem depicted in Example 3.6 and can be modeled in an analogous way. However, in contrast to the round trip problem, we do not need that the drill head returns back to the starting point after completion of the drilling. Figure 3.7 shows a section of a circuit board and a possible drilling order for that area.

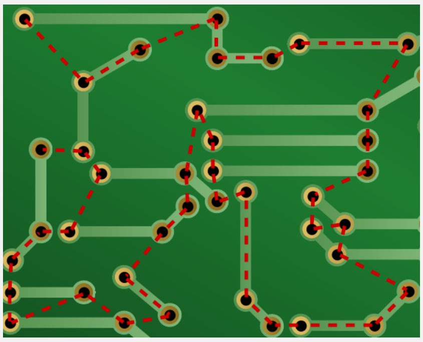
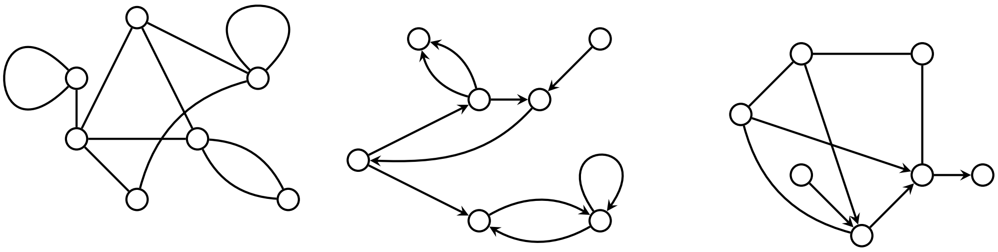


Figure 3.7: Section of a circuit board with a possible drilling order.

3.2 Basic terminology and notation

A graph consists of a set of *vertices* and connections between pairs of vertices, which are called *edges*. Edges can be *undirected* or *directed*, that is, oriented from one endpoint to another. A graph containing only undirected edges is an *undirected graph* (Figure 3.8a). If all edges are directed, we speak of a *directed graph* (Figure 3.8b). It is also possible that undirected and directed edges occur simultaneously. In this case we are talking about a *mixed graph* (Figure 3.8c).

Our main focus will be on directed and undirected graphs, which we will formally introduce shortly. In the context of graphs, various notations and terminology are used throughout the literature. We focus on one widespread formalization in the following.



(a) An undirected graph.

(b) A directed graph.

(c) A mixed graph.

Figure 3.8: Different types of graphs.

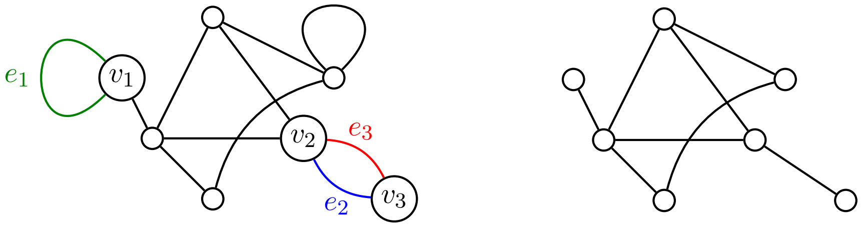
Undirected graphs

An *undirected* graph $G = (V, E)$ is a pair consisting of a finite set V of *vertices* and a finite set E of *edges*. Each edge $e \in E$ has two endpoints $u, v \in V$, which may be identical. The endpoints of an edge are denoted by $\mathrm{endpoints}(e) = \{u, v\}$.

An edge $e \in E$ is called a *loop* if its two endpoints are identical, or in other words if $|\mathrm{endpoints}(e)| = 1$. Two different edges $e_1, e_2 \in E$ are called *parallel* if $\mathrm{endpoints}(e_1) = \mathrm{endpoints}(e_2)$. An (undirected) graph without loops and parallel edges is called a *simple graph*. For example, in Figure 3.9a, we have

$$\mathrm{endpoints}(e_1) = \{v_1\} \quad \text{and} \quad \mathrm{endpoints}(e_2) = \mathrm{endpoints}(e_3) = \{v_2, v_3\} \ ;$$

hence e_1 is a loop and the edges e_2 and e_3 are parallel. Thus, the graph in Figure 3.9a is not a simple graph. The graph in Figure 3.9b, on the other hand, has neither loops nor parallel edges and is thus simple.



(a) A non-simple graph.

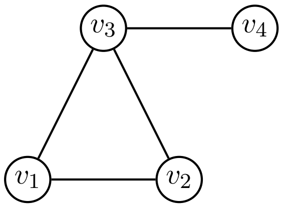
(b) A simple graph.

Figure 3.9: Non-simple and simple graphs.

In simple graphs, we often describe an edge as the set of its endpoints, i.e.,

$$E \subseteq \binom{V}{2} := \{\{u, v\} \colon u, v \in V,\ u \neq v\} \ .$$

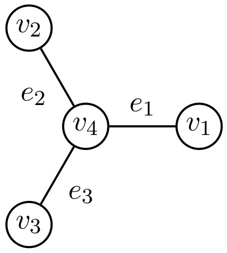
Figure 3.10 illustrates this notation, which is unambiguous in simple graphs, because for any set of endpoints there is at most one edge corresponding to it. In non-simple graphs, however, ambiguities may appear when referring to one of several parallel edges by its endpoints. Nevertheless, we also frequently use the above notation in the context of non-simple graphs in cases where there is little danger of confusion.



$$G = (V, E)$$
$$V = \{v_1, v_2, v_3, v_4\}$$
$$E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}, \{v_3, v_4\}\}$$

Figure 3.10: Notation for simple graphs. Due to its convenience, this notation is also often used in the context of non-simple graphs.

An edge $e \in E$ and a vertex $v \in V$ are called *incident* if v is one of the endpoints of e, i.e., if $v \in \text{endpoints}(e)$. Two distinct vertices $u, v \in V$ are called *adjacent* (or *neighboring*) if there is an edge e with $\text{endpoints}(e) = \{u, v\}$. With the notation introduced above, u and v are adjacent if and only if $\{u, v\} \in E$. Figure 3.11 shows examples of incident and adjacent vertices.



v_1 is incident to e_1, but not to e_2 or e_3.

v_4 is adjacent to v_1, v_2, and v_3.

v_1 and v_2 are not adjacent.

Figure 3.11: Incidence and adjacency in the context of vertices.

Directed graphs

An *directed graph* $G = (V, A)$ consists of a finite set V of vertices and a finite set A of *directed edges*, which we call *arcs* to more easily distinguish them from their undirected counterparts. Every arc $a \in A$ has a *tail* $\text{tail}(a) \in V$ and a *head* $\text{head}(a) \in V$ and points from tail to head. In particular, the endpoints are given by $\text{endpoints}(a) = \{\text{tail}(a), \text{head}(a)\}$.

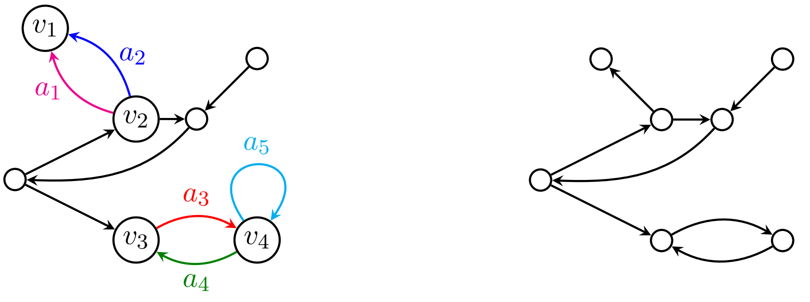
An arc a is a *loop* if $\text{tail}(a) = \text{head}(a)$. Two distinct arcs a_1 and a_2 are called *parallel* if $\text{tail}(a_1) = \text{tail}(a_2)$ and $\text{head}(a_1) = \text{head}(a_2)$. They are called *antiparallel* if $\text{tail}(a_1) = \text{head}(a_2)$ and $\text{head}(a_1) = \text{tail}(a_2)$. As for undirected graphs, a directed graph $G = (V, A)$ is called *simple* if it contains neither loops nor parallel arcs. However, a simple directed graph may contain antiparallel arcs. Figure 3.12 shows examples of a non-simple and a simple directed graph. In Figure 3.12a, we have

$$\left.\begin{aligned} \text{head}(a_1) = \text{head}(a_2) = v_1 \\ \text{tail}(a_1) = \text{tail}(a_2) = v_2 \end{aligned}\right\} \quad \Longrightarrow \quad a_1 \text{ and } a_2 \text{ are parallel,}$$

$$\left.\begin{aligned}\operatorname{tail}(a_3) = \operatorname{head}(a_4) = v_3\\ \operatorname{head}(a_3) = \operatorname{tail}(a_4) = v_4\end{aligned}\right\} \quad\Longrightarrow\quad a_3 \text{ and } a_4 \text{ are antiparallel,}$$

$$\left.\begin{aligned}\operatorname{head}(a_5) = \operatorname{tail}(a_5) = v_4\\ \text{or, equivalently: } \operatorname{endpoints}(a_5) = \{v_4\}\end{aligned}\right\} \quad\Longrightarrow\quad a_5 \text{ is a loop.}$$

Hence, because the directed graph in Figure 3.12a has both parallel edges and moreover a loop, it is not simple. The directed graph in Figure 3.12b is simple.



(a) A non-simple directed graph.

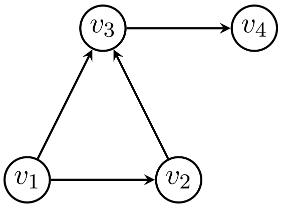
(b) A simple directed graph.

Figure 3.12: Non-simple and simple directed graphs.

Analogously to the undirected case, we can use a simplified notation for arcs that is unambiguous in simple graphs. More precisely, we can describe an arc $a \in A$ as an ordered pair $a = (u, v)$ of vertices, where $u = \operatorname{tail}(a)$ and $v = \operatorname{head}(a)$. With this notation it holds that

$$A \subseteq \{(u,v) \colon u, v \in V,\ u \neq v\} \enspace ,$$

where A is the arc set of a simple directed graph $G = (V, A)$. As before, this notation is unambiguous only if no parallel arcs exist. Similar to the undirected case, we nevertheless use this convenient notation also for directed graphs with parallel arcs, as long as there is no danger of ambiguity. Figure 3.13 illustrates this notation.



$$G = (V, A)$$
$$V = \{v_1, v_2, v_3, v_4\}$$
$$A = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_3, v_4)\}$$

Figure 3.13: Notation for simple directed graphs. Due to its convenience, this notation is also used in the context of non-simple graphs.

Basic notation and terminology

We continue with some very common and basic notation and terminology in the context of graphs. Because of the close relation between undirected and directed graphs, when it comes to terminology and notation, we discuss both cases together. In parenthesis, we indicate whether the introduced notation is commonly used for undirected (undir.) graphs, directed (dir.) graphs, or both (undir./dir.).

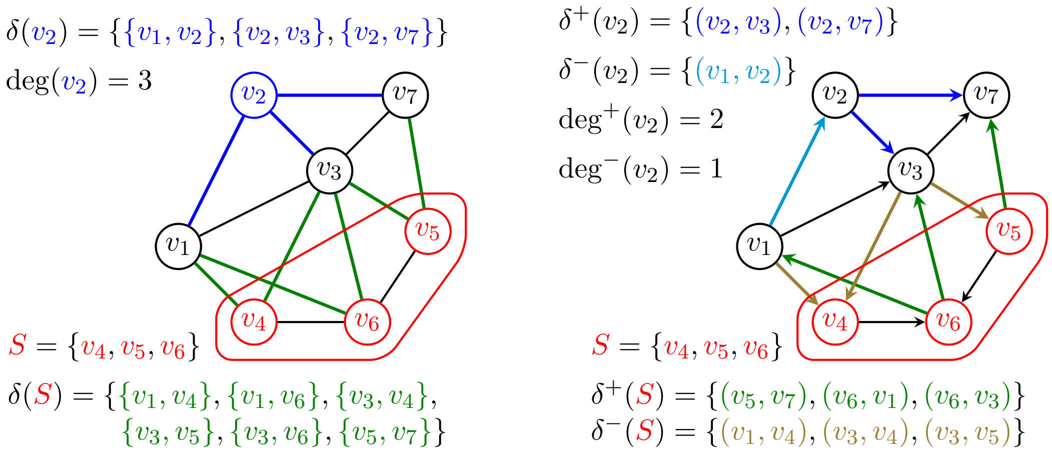
- (undir./dir.) Edges that *cross* the subset of vertices $S \subseteq V$:
$$\delta(S) := \left< \begin{array}{l} \{\{u,v\} \in E \colon |\{u,v\} \cap S| = 1, u \neq v\} \ , \\ \{(u,v) \in A \colon |\{u,v\} \cap S| = 1, u \neq v\} \ . \end{array} \right.$$
- (dir.) Arcs leaving the vertex set $S \subseteq V$:
$$\delta^+(S) := \{(u,v) \in A \colon u \in S,\ v \notin S\} \ .$$
- (dir.) Arcs entering the vertex set $S \subseteq V$:
$$\delta^-(S) := \{(u,v) \in A \colon u \notin S,\ v \in S\} \ .$$

In particular, for a directed graph $G = (V, A)$ and a subset of vertices $S \subseteq V$, it always holds that $\delta(S) = \delta^+(S) \cup \delta^-(S)$.

For a single vertex $v \in V$, we use the following shorthands of the above notation:
$$\delta(v) := \delta(\{v\}) \ , \quad \delta^+(v) := \delta^+(\{v\}) \ , \quad \text{and} \quad \delta^-(v) := \delta^-(\{v\}) \ .$$

- (undir./dir.) *Degree* of v: $\deg(v) := |\delta(v)| + 2 \cdot |\{\text{loops at } v\}|$.
- (dir.) *Outdegree* of v: $\deg^+(v) := |\delta^+(v)| + |\{\text{loops at } v\}|$.
- (dir.) *Indegree* of v: $\deg^-(v) := |\delta^-(v)| + |\{\text{loops at } v\}|$.

Figure 3.14 exemplifies the introduced notation.


$\delta(v_2) = \{\{v_1,v_2\},\{v_2,v_3\},\{v_2,v_7\}\}$
$\deg(v_2) = 3$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
$S = \{v_4,v_5,v_6\}$
$\delta(S) = \{\{v_1,v_4\},\{v_1,v_6\},\{v_3,v_4\},\{v_3,v_5\},\{v_3,v_6\},\{v_5,v_7\}\}$
$\delta^+(v_2) = \{(v_2,v_3),(v_2,v_7)\}$
$\delta^-(v_2) = \{(v_1,v_2)\}$
$\deg^+(v_2) = 2$
$\deg^-(v_2) = 1$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
$S = \{v_4,v_5,v_6\}$
$\delta^+(S) = \{(v_5,v_7),(v_6,v_1),(v_6,v_3)\}$
$\delta^-(S) = \{(v_1,v_4),(v_3,v_4),(v_3,v_5)\}$

(a) Undirected graph.

(b) Directed graph.

Figure 3.14: Crossing edges and vertex degrees.

Having introduced the most important notations for dealing with graphs, we now present two simple yet very useful properties of the vertex degrees occurring in a graph.

Property 3.8: Sum of all vertex degrees

In an undirected graph $G = (V, E)$, the sum of the degrees of all vertices is even.

Proof of Property 3.8. If we sum up the vertex degrees, each edge is counted twice, once for each of its endpoints. Consequently

$$\sum_{v \in V} \deg(v) = 2|E| \ ,$$

implying that the sum of all vertex degrees is even. □

The proof of Property 3.8 is an example of double counting. A direct implication of Property 3.8 is the so-called *handshaking lemma*.

Property 3.9: Handshaking lemma

In every undirected graph, the number of odd-degree vertices is even.

Proof of Property 3.9. We partition the vertex set $V = V_1 \dot\cup V_2$, where

$$\begin{aligned} V_1 &:= \{v \in V \colon \deg(v) \text{ odd}\} = \{\text{odd-degree vertices}\} \ , \text{ and} \\ V_2 &:= \{v \in V \colon \deg(v) \text{ even}\} = \{\text{even-degree vertices}\} \ . \end{aligned}$$

We now have

$$\sum_{v \in V_1} \deg(v) = \sum_{v \in V} \deg(v) - \sum_{v \in V_2} \deg(v) \ .$$

On the right-hand side of the last equation, the first sum is even by Property 3.8, and the second one is even as well because each summand is even. Thus it follows that the sum on the left-hand side is even. Notice that each term in the sum on the left-hand side is odd by definition of V_1; hence, the number of summands must be even, i.e., $|V_1|$ is even. □

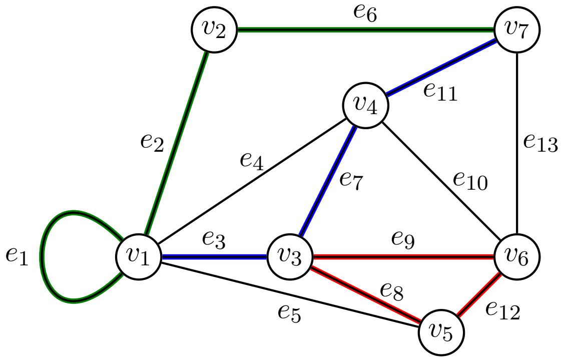
The reason why Property 3.9 is also known as the handshaking lemma is the following: Property 3.9 states that at every party the number of people who shake hands with an odd number of guests has to be even. To see this, consider a graph that has the party's guests as its vertices. Two people are connected by an edge if and only if they shake hands. Thus, the degree of a vertex is the number of guests with whom the corresponding person shook hands. The statement now follows immediately from Property 3.9.

A *walk* in an undirected graph $G = (V, E)$ is a sequence $v_1, e_1, v_2, \dots, v_{k-1}, e_{k-1}, v_k$ (possibly with repetitions) consisting of $k \in \mathbb{Z}_{\geq 1}$ vertices $v_1, \dots, v_k \in V$ and $k - 1$ edges $e_1, \dots, e_{k-1} \in E$ with $e_i = \{v_i, v_{i+1}\}$ for all $i \in [k-1]$, where we recall that $[\ell] := \{1, \dots, \ell\}$ for $\ell \in \mathbb{Z}_{\geq 1}$, and we use the convention $[0] = \emptyset$. We call such a walk a *walk between v_1 and v_k*. For an undirected walk, we consider the description $v_1, e_1, v_2, \dots, v_{k-1}, e_{k-1}, v_k$ and the description $v_k, e_{k-1}, v_{k-1}, \dots, v_2, e_1, v_1$ in the opposite direction as equivalent, i.e., they describe the same walk.

In a directed graph $G = (V, A)$, a walk is a sequence $v_1, a_1, v_2, \dots, v_{k-1}, a_{k-1}, v_k$ (possibly with repetitions) consisting of vertices $v_1, \dots, v_k \in V$ and arcs $a_1, \dots, a_{k-1} \in A$ with $a_i =$

$(v_i, v_{i+1}) \in A$ for all $i \in [k-1]$. In the directed case we are talking about a *walk from* v_1 *to* v_k or a v_1-v_k *walk*. This notation is also used for undirected walks, where in undirected graphs, each v_1-v_k walk is also a v_k-v_1 walk.

In both cases we define the *length* of a given walk with k vertices—where a vertex is counted as many times as it appears in the walk—as $k-1$, which is the number of edges in the walk, also counted with multiplicities if an edge appears multiple times. A walk is called a *path* if it contains no vertex more than once, i.e., $v_i \neq v_j$ for all $i, j \in [k]$ with $i \neq j$. A walk is called *closed*, if $v_1 = v_k$. A closed walk, in which all vertices except for start and endpoint are pairwise distinct, is called a *cycle*. Figure 3.15 illustrates the new terminology.



$v_1, e_1, v_1, e_2, v_2, e_6, v_7, e_6, v_2$ is a walk of length 4, but not a path, since v_1 and v_2 appear multiple times.

$v_1, e_3, v_3, e_7, v_4, e_{11}, v_7$ is a v_1-v_7-path of length 3.

$v_3, e_8, v_5, e_{12}, v_6, e_9, v_3$ is a closed walk with pairwise distinct vertices (up to start and endpoint), hence it is a cycle of length 3.

Figure 3.15: Walks, paths, closed walks, cycles, and their lengths.

To simplify the notation, we often denote a path or cycle simply by the set of edges it traverses, for example, a cycle C in a graph $G = (V, E)$ is represented as an edge set $C \subseteq E$. In addition, '$+$' and '$-$' are used for adding or deleting a single element to or from a set, i.e., $S + u - w := (S \cup \{u\}) \setminus \{w\}$.

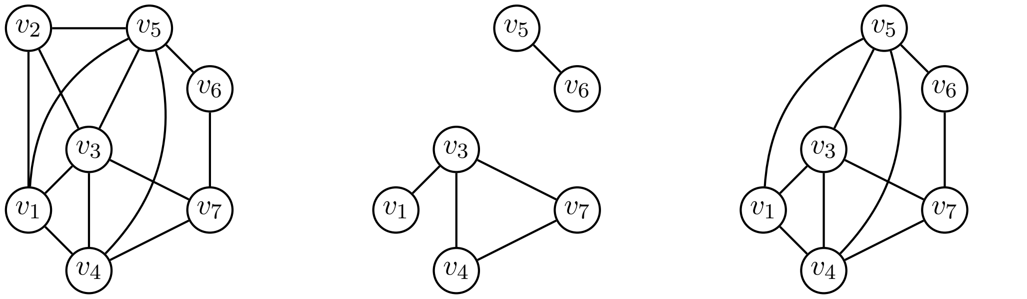
A *subgraph* of an undirected graph $G = (V, E)$ is a graph $H = (W, F)$ with $W \subseteq V$ and $F \subseteq \{e \in E \colon e \subseteq W\}$, i.e., F is a subset of those edges of G that have both endpoints in W. For a vertex set $W \subseteq V$, the subgraph *induced* by W is the subgraph of $G = (V, E)$ defined by

$$G[W] = (W, E[W]) \ ,$$

where $E[W] := \{e \in E \colon e \subseteq W\}$ is the set of all edges of G with both endpoints in W. The definitions of subgraph and induced subgraph are analogous for directed graphs. Figure 3.16 exemplifies these notions for undirected graphs.

Exercise 3.10: Seating arrangements

You invite n guests over for dinner. For each pair of guests it is known whether they are friends or not. The task is to create a seating plan for a round table such that guests sitting next to each other are friends. Formulate this problem as the graph-theoretic problem of finding a certain cycle in a well-defined graph.



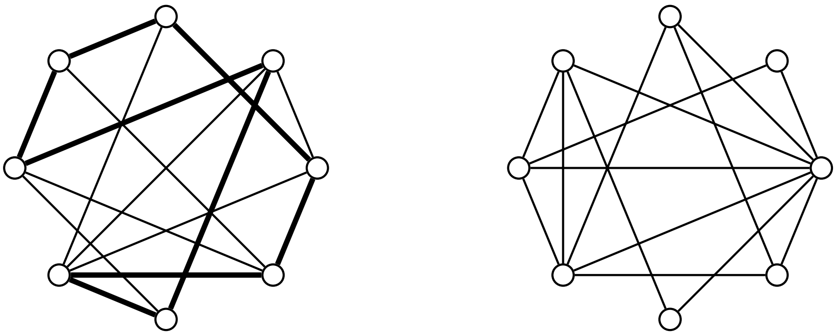
(a) A graph $G = (V, E)$.

(b) A subgraph of G.

(c) Induced subgraph $G[V \setminus \{v_2\}]$.

Figure 3.16: Graphs, subgraphs, and induced subgraphs.

Solution
We construct a graph $G = (V, E)$ in which each vertex corresponds to a guest. Two vertices are connected by an edge if and only if the two corresponding guests are friends. A desired seating plan now corresponds to a cycle in G that contains all the vertices in V exactly once (see Figure 3.17a). Such a cycle is called a *Hamilton cycle*. Of course, not every graph contains a Hamilton cycle. For example, for the graph in Figure 3.17b, it can be shown that it does not contain a Hamilton cycle.

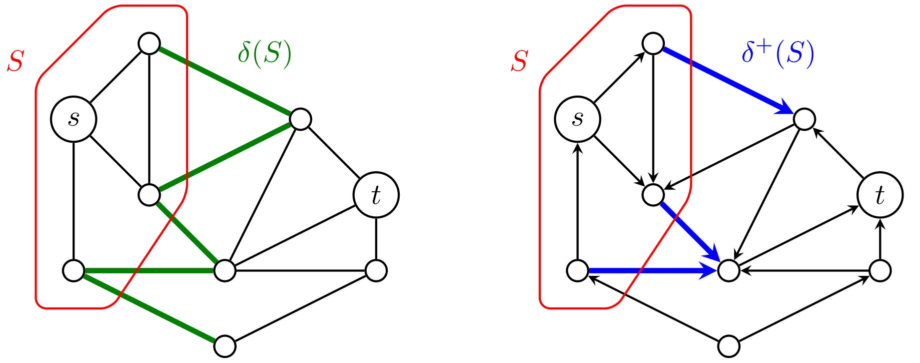


(a) A graph with a Hamilton cycle.

(b) A graph with no Hamilton cycle.

Figure 3.17: Hamilton cycles in graphs.

A *cut* in a graph $G = (V, E)$ is a nonempty subset $S \subsetneq V$ of vertices, where $S \subsetneq V$ is a shorthand notation for $S \subseteq V$ and $S \neq V$. One can also imagine a cut as a non-trivial partition of the vertex set V into the sets S and $V \setminus S$. This partition is non-trivial because both S and $V \setminus S$ are nonempty. For this reason, a cut is sometimes also written as the pair $(S, V \setminus S)$. Cuts are defined analogously in directed and undirected graphs. For a cut S in an undirected graph, the edges in $\delta(S)$ are also referred to as the *edges in the cut* S or as the edges *crossing* S. In the case of a directed graph, we denote the edges in $\delta^+(S)$ as the *edges in the cut* S. For two distinct vertices $s, t \in V$, an s-t *cut* in G is a cut $S \subseteq V$ such that $s \in S$ and $t \notin S$. In other words, an s-t cut separates s from t. Figure 3.18 illustrates the difference between the undirected and the directed case.



(a) Undirected graph.

(b) Directed graph.

Figure 3.18: An s-t cut S and the edges in the cut $\delta(S)$ (for the undirected case) and $\delta^+(S)$ (for the directed case).

Exercise 3.11

Let $G = (V, A)$ be a directed graph, S a cut in G, and $C \subseteq A$ a directed cycle in G. Show that $|\delta^+(S) \cap C| = |\delta^-(S) \cap C|$.

Solution

Let $H = (V, C)$ be the subgraph of G only containing the arcs in C. For each vertex set $W \subseteq V$, let $\delta^+_H(W) := \delta^+(W) \cap C$ and $\delta^-_H(W) := \delta^-(W) \cap C$. Using this notation, we therefore need to prove $|\delta^+_H(S)| = |\delta^-_H(S)|$. First we show

$$|\delta^+_H(S)| - |\delta^-_H(S)| = \sum_{v \in S} \left(\deg^+_H(v) - \deg^-_H(v)\right) \quad . \tag{3.1}$$

This equality can be verified by checking that each arc $a \in C$ contributes to both sides of the equation equally. An easy way to perform this check is by distinguishing the following three types of arcs: (i) $a \in \delta^+_H(S)$, (ii) $a \in \delta^-_H(S)$, and (iii) $a \notin \delta^+_H(S) \cup \delta^-_H(S)$.
Because C is a directed cycle, every vertex in H has the same indegree as outdegree, i.e., $\deg^+_H(v) = \deg^-_H(v)$ for all $v \in V$. Consequently, every single summand on the right-hand side of (3.1) is equal to zero. Hence, the difference on the left-hand side is zero, too. It follows that $|\delta^+_H(S)| = |\delta^-_H(S)|$, as desired.

3.3 Data structures for graphs

One of our main goals is to develop and analyze algorithms for graph optimization problems. To this end, we need to clarify how a graph is stored in a computer and how graph-related data can be accessed. This will both clarify the input size of a graph and the time we need to perform basic graph operations, both of which are key to talk about the running time of graph algorithms.

In this section we address these points by talking about data structures for graphs.

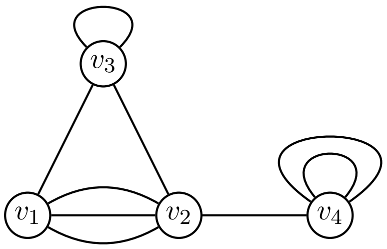
Two of the most widely used data structures for graphs are the representation as an *adjacency matrix* (sometimes called *neighborhood matrix*) and the representation as an *incidence list*. Unless explicitly indicated otherwise, we will always assume that the underlying data structure is an incidence list. However, we start by presenting the adjacency matrix before expanding on the incidence list, due to its simplicity. The choice of data structures is a crucial and often highly non-trivial step in algorithmics that comes with various trade-offs. Introducing another graph data structure besides the incidence list allows us to discuss and compare advantages and disadvantages of these two data structures in comparison to each other, and to highlight some example trade-offs faced when choosing a graph data structure.

The adjacency matrix

The *adjacency matrix* of an undirected graph $G = (V, E)$ is the symmetric matrix $M \in \mathbb{Z}_{\geq 0}^{V \times V}$ whose entries for all $u, v \in V$ are defined by

$$M(u,v) := \begin{cases} |\{e \in E\colon \text{ endpoints}(e) = \{u,v\}\}| & \text{if } u \neq v \ , \\ 2 \cdot |\{\text{loops at } v\}| & \text{if } u = v \ . \end{cases}$$

Figure 3.19 shows the adjacency matrix of an undirected graph with 4 vertices.



$$M = \begin{array}{c c} & \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} & \begin{pmatrix} 0 & 3 & 1 & 0 \\ 3 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 \\ 0 & 1 & 0 & 4 \end{pmatrix} \end{array}$$

Figure 3.19: The adjacency matrix of an undirected graph.

The adjacency matrix of a directed graph $G = (V, A)$ is the matrix $M \in \mathbb{Z}_{\geq 0}^{V \times V}$ whose entries for all $u, v \in V$ are given by

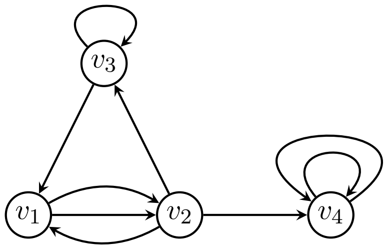
$$M(u,v) := |\{a \in A\colon \text{ tail}(a) = u, \text{head}(a) = v\}| \ .$$

Figure 3.20 shows the adjacency matrix of a directed graph with 4 vertices. As the example in Figure 3.20 highlights, the adjacency matrix of a directed graph is not necessarily symmetric.

For simple graphs, the corresponding adjacency matrices are $\{0, 1\}$-matrices, i.e., matrices where each entry is either 0 or 1. In addition, for simple graphs, all entries on the diagonal of the adjacency matrix are 0.

The incidence list

An *incidence list* contains for each vertex a doubly linked list of those (directed or undirected) edges incident to the corresponding vertex. To access these lists, we can use an array indexed



$$M = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 2 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \end{array}$$

Figure 3.20: The adjacency matrix of a directed graph.

by the vertices of the graph, which contains the pointers to the respective lists. More precisely, these pointers point to the first element of the respective lists. For each vertex v, accessing the first element of the list corresponding to v requires constant time. Figure 3.21 schematically shows the incidence list of a directed graph.

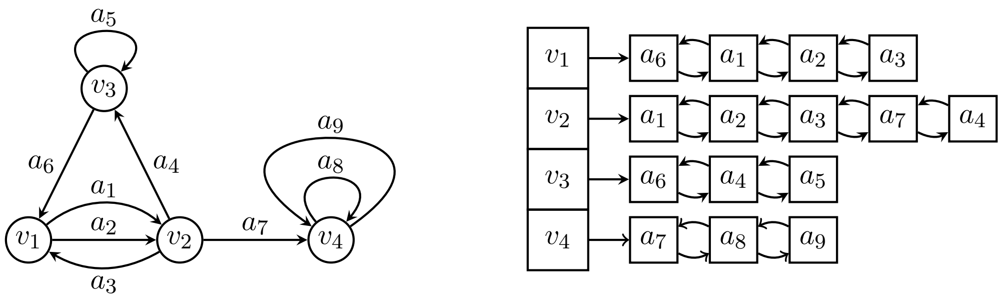


Figure 3.21: The incidence list of a directed graph.

For undirected edges e we use a data structure in which the set $\mathrm{endpoints}(e)$ is stored and can be queried. In the case of an arc a, it is possible to query $\mathrm{tail}(a)$ and $\mathrm{head}(a)$ in constant time. Further information about the edges, for example weights, can also be stored in the data structure for the edges.

Comparison: adjacency matrix and incidence list

Space requirements We consider a simple undirected graph $G = (V, E)$ (the directed case is analogous). Let $n := |V|$ be the number of vertices and $m := |E|$ the number of edges of G. An important difference between the adjacency matrix and the incidence list of G is the space required:

- Adjacency matrix: $\Theta(n^2)$.
- Incidence list: $\Theta(m + n)$.

Thus, for simple graphs the incidence list is typically a more compact encoding of the graph, because in this case $m = O(n^2)$. The reason for the significantly larger space requirement of the adjacency matrix is that even if two vertices are not connected by an edge, it stores a 0 to capture this information. This difference is particularly pronounced if the graph is *sparse*, that is, if it has only few edges, for example $m = O(n)$. Here, the adjacency matrix needs quadratic space in n whereas the incidence list only requires linear space. This difference is crucial when dealing with graphs with millions of vertices but constant average degree—a regime that is very common in many applications.

Algorithmic complexity of some operations the Table 3.1 lists, for both the adjacency matrix and the incidence list, the running times of several simple computational tasks in an undirected graph $G = (V, E)$, namely: calculating the degree of a vertex or listing all incident edges, deciding whether a particular edge is part of the graph, and computing the number of edges $|E|$.

	$\deg(v)$ or $\delta(v)$	$\exists?\{u,v\} \in E$	$\|E\|$
adjacency matrix	$O(n)$	$O(1)$	$O(n^2)$
incidence list	$O(\deg(v))$	$O(\min\{\deg(u), \deg(v)\})$	$O(m+n)$

Table 3.1: Running time of simple computations in graphs.

Exercise 3.12

Let $G = (V, E)$ be a graph given by an incidence list and let $u, v \in V$ two vertices. Show that in time $O(\min\{\deg(u), \deg(v)\})$ it can be decided whether G contains an edge $\{u, v\}$.

We recall that, unless explicitly stated otherwise, we will always assume that graphs are given as incidence lists. The size of a graph with n vertices and m edges is therefore $\Theta(m+n)$. Thus, an algorithm whose input is a single such graph is polynomial if and only if its running time is upper bounded by a polynomial in $m + n$.

3.4 Breadth-first search (BFS): shortest paths and more

We now discuss one of the best-known search methods on graphs, namely breadth-first search, or BFS for short. To be exact, breadth-first search is an algorithmic idea that can be adapted to solve many different problems. We introduce it in the context of finding shortest paths in undirected graphs. Later we will see how the algorithm can be modified for directed graphs.

Let $G = (V, E)$ be an undirected graph. We define a distance function $d\colon V \times V \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$ as follows:

$$d(u, v) := \min\{|P| \colon P \subseteq E,\ P \text{ is a walk between } u \text{ and } v\} \qquad \forall u, v \in V \ , \tag{3.2}$$

i.e., $d(u, v)$ is the length of the shortest u-v-walk in G. If there is no walk between u and v, then $d(u, v)$, as defined above, is the minimum over an empty set. For this case we set by convention $d(u, v) = \infty$. Notice that one can impose in (3.2) the additional requirement that P must be a path (instead of a walk) without changing the definition. Indeed, this would not change the value of $d(u, v)$ because a shortest walk is always a path. Figure 3.22 shows a graph $G = (V, E)$ that contains next to each vertex v the distance to the fixed vertex v_1, i.e., the length of a shortest path to v_1.

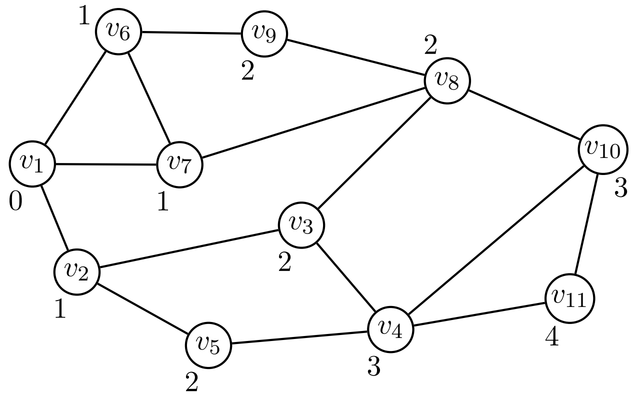


Figure 3.22: The number next to each vertex gives the distance to v_1.

Breadth-first search is an algorithm that efficiently calculates the distances $d(v_1, v)$ for all vertices $v \in V$ for a fixed vertex $v_1 \in V$. Algorithm 2 describes BFS in pseudocode.

Algorithm 2: Breadth-first search: computation of distances from a fixed vertex v_1

```
Input: $G = (V, E), v_1 \in V$.
Output: $d(v_1, v)$ for all $v \in V$.
  1. Initialization:
     $d(v_1, v) = \begin{cases} \infty & \text{if } v \in V \setminus \{v_1\}, \\ 0 & \text{if } v = v_1. \end{cases}$
     $L = \{v_1\}$.                          // vertices to be processed
     $k = 1$.                     // shortest possible assignable distance
  2. while ($L \neq \emptyset$) do:
       $L_{\text{new}} = \emptyset$.
       for $v \in N(L) := \{u \in V : \exists w \in L$ with $\{w, u\} \in E\}$ do:
         if $d(v_1, v) = \infty$ then:
           $d(v_1, v) = k$.
           $L_{\text{new}} = L_{\text{new}} \cup \{v\}$.
       $k = k + 1$.
       $L = L_{\text{new}}$.
  3. return $d(v_1, v)$ for all $v \in V$.
```

The idea of breadth-first search is simple: We first consider the vertex v_1. For this vertex, we know that $d(v_1, v_1) = 0$, because an empty walk has no edges and therefore has length 0. In the following, we consider all vertices in the *neighborhood* $N(\{v_1\})$ of v_1, that is, all vertices connected to v_1 by an edge. For these vertices v we set $d(v_1, v) = 1$. In the next step we consider all vertices that are adjacent to one of the vertices of distance 1 and have not yet been considered. For these vertices v we set $d(v_1, v) = 2$, and so on. Table 3.2 shows the sets L and $N(L)$ for each iteration of the while-loop Algorithm 2 runs through for the graph in Figure 3.22. The set L contains the vertices newly considered in the previous iteration, $N(L)$ their neighbors.

k	L	$N(L)$
1	$\{v_1\}$	$\{v_2, v_6, v_7\}$
2	$\{v_2, v_6, v_7\}$	$\{v_1, v_3, v_5, v_6, v_7, v_8, v_9\}$
3	$\{v_3, v_5, v_8, v_9\}$	$\{v_2, v_3, v_4, v_6, v_7, v_8, v_9, v_{10}\}$
4	$\{v_4, v_{10}\}$	$\{v_3, v_4, v_5, v_8, v_{10}, v_{11}\}$
5	$\{v_{11}\}$	$\{v_4, v_{10}\}$
6	$\{\}$	$\{\}$

Table 3.2: Breadth-first search table for the graph in Figure 3.22.

In the following we show that the function d computed by breadth-first search indeed corresponds to the distances from v_1. In addition, we show that the runtime of BFS is $O(m + n)$, where $n := |V|$ and $m := |E|$, which is linear in the size of the graph.

Theorem 3.13: Correctness of breadth-first search

For every graph $G = (V, E)$ and every vertex $v_1 \in V$, Algorithm 2 calculates the values $d(v_1, v)$ correctly for all $v \in V$.

Proof. To avoid confusion, let $d'(v_1, v)$ for $v \in V$ be the distances computed by Algorithm 2, and d the true distances in G as defined in (3.2). Furthermore, let $D'_\ell = \{v \in V : d'(v_1, v) = \ell\}$ and $D_\ell = \{v \in V : d(v_1, v) = \ell\}$ for all $\ell \in \mathbb{Z}_{\geq 0}$.

We show for every $k \in \mathbb{Z}_{\geq 0}$ that Algorithm 2, after k iterations of the while-loop, computed the set of all vertices $v \in V$ satisfying $d(v_1, v) \leq k$ as well as their distances correctly, i.e., $D_\ell = D'_\ell$ for all $\ell \leq k$. For this we use induction on k. For $k = 0$ the statement is certainly true, because only $v = v_1$ satisfies $d(v_1, v) = 0$ and only $v = v_1$ satisfies $d'(v_1, v) = 0$, So, suppose the statement holds for iteration k and consider iteration $k+1$ of the while-loop. In this iteration, we set $d'(v_1, v) = k + 1$ for all v that are adjacent to at least one vertex in $D'_k = D_k$ and have not yet been considered, hence $d'(v_1, v) = \infty$. Consequently,

$$D'_{k+1} = \{v \in V : d(v_1, v) > k, v \in N(D_k)\} \ .$$

We need to show that $D'_{k+1} = D_{k+1}$. For this we prove both inclusions $\subseteq$ and $\supseteq$.

$\boldsymbol{D'_{k+1} \subseteq D_{k+1}}$**:** Let $v \in D'_{k+1}$. Because $v \in N(D_k)$, it holds that $d(v_1, v) \leq k + 1$: At least one neighbor w of v has distance k from v_1, thus there exists a v_1-w walk of length at most k. This walk can be extended to a v_1-v walk of length at most $k + 1$, which yields $d(v_1, v) \le k+1$. In addition, we have $d(v_1, v) > k$ because, due to the inductive assumption, all vertices v with $d(v_1, v) \le k$ have already been considered. Together it follows that $d(v_1, v) = k+1$, so $v \in D_{k+1}$.

$\boldsymbol{D_{k+1} \subseteq D'_{k+1}}$**:** Let $v \in D_{k+1}$. Hence, there is a v_1-v walk of length $k+1$ with vertices $v_1, \dots, v_{k+1}, v$. It follows that $v_{k+1} \in D_k$ needs to hold; for otherwise we could construct a v_1-v-path of length $< k+1$. Thus, $v \in N(D_k)$, and consequently $v \in D'_{k+1}$.

This completes the inductive step and thus implies the statement. □

Theorem 3.14: Running time of breadth-first search

For every graph $G = (V, E)$ and every vertex $v_1 \in V$, Algorithm 2 has a running time bounded by $O(m+n)$.

Proof. The initialization in the first step of Algorithm 2 requires $O(n)$ time. For the second step, we first determine the time needed to compute the neighborhoods $N(L)$ calculated at the beginning of the for-loop. Let $L_1 = \{v_1\}, L_2, \dots, L_{k-1}, L_k = \emptyset$ be the sets L occurring during the algorithm. Then L_i contains exactly those vertices of V that have distance $i-1$ from v_1. Thus, $L_1, \dots, L_k$ is a partition of all vertices that are in the same connected component as v_1.

The calculation of $N(L)$ takes $O\left(|L| + \sum_{v \in L} \deg(v)\right)$ time, because all neighbors of vertices in L are explored. Thus, the running time to determine all neighborhoods $N(L_1), \dots, N(L_k)$ is bounded by

$$O\left(\sum_{i=1}^{k}\left(|L_i| + \sum_{v \in L_i} \deg(v)\right)\right) = O\left(|V| + \sum_{v \in V} \deg(v)\right) = O(n+m) \ ,$$

where we use that $L_1, \dots, L_k$ is a partition of V, and that the sum of all vertex degrees is exactly $2m$ (see proof of Property 3.8). Furthermore, each vertex $v \in V$ is contained in at most 3 different neighborhoods $N(L_j)$, namely in the iteration i such that $v \in L_i$, as well as in the one before and after. Indeed, a vertex can only be a neighbor of vertices with either the same distance to v_1 or with a distance to v_1 differing from $d(v_1, v)$ by exactly one unit. Thus, $\sum_{i=1}^{k} |N(L_i)| = O(n)$, and because every operation within the for-loop only requires constant time per iteration, the total running time of all operations within the for-loop is bounded by $O(n)$. The output of the distances $d(v_1, v)$ in the last step requires $O(n)$ time. Overall, the running time is therefore bounded by $O(m+n)$, as desired. □

Algorithm 2 can easily be adapted for directed graphs $G = (V, A)$. For this we define the *out-neighborhood* $N^+(L)$ as

$$N^+(L) = \{u \in V \colon \exists w \in L \text{ with } (w, u) \in A\} \ .$$

It suffices to replace $N(L)$ by $N^+(L)$ in Algorithm 2 to adapt it to the directed case.

Connectivity and connected components

A canonical application of BFS is to study the connectivity of a graph.

Definition 3.15: Connectivity in undirected graphs

Let $G = (V, E)$ be an undirected graph.

(i) Two vertices $s, t \in V$ are called *connected* in G if G contains an s-t walk.

(ii) The graph G is callled *connected* if each pair of vertices in G is connected.

With breadth-first search it is easy to check if a graph $G = (V, E)$ is connected. It suffices to perform a single breadth-first search with an arbitrary starting vertex $v_1 \in V$, because G is connected if and only if $d(v_1, v) < \infty$ holds for all $v \in V$.

In directed graphs $G = (V, A)$, connectivity can be defined analogously by “forgetting” the directions of the arcs. Moreover, there is a stronger notion of connectivity for directed graphs.

Definition 3.16: Connectivity in directed graphs

Let $G = (V, A)$ be a directed graph.

(i) G is called *connected*, if the undirected graph G', obtained from G by ignoring arc directions, is connected.

(ii) Let $s, t \in V$. The vertex t can be *reached* from s in G if G contains a directed s-t walk.

(iii) The graph G is called *strongly connected* if every vertex in G can be reached from every other vertex.

Also strong connectivity (of a directed graph) can be checked through BFS, using only two BFS calls. To this end, we fix an arbitrary vertex v_1 and start BFS with v_1 as starting vertex. This run allows us to determine whether every vertex can be reached from v_1, that is, if $d(v_1, v) < \infty \ \forall v \in V$. If a vertex cannot be reached from v_1, then G is not strongly connected. Otherwise, we call BFS a second time with starting vertex v_1 but this time on the graph $\overline{G}$ obtained from G by reversing all arc directions. This allows us to determine whether in G, the vertex v_1 can be reached from all other vertices. If not, then G is not strongly connected. Otherwise, G is strongly connected. Indeed, if there is a vertex that can reach any other vertex and can be reached by any other vertex then the graph must be strongly connected. This follows from the fact that for any two vertices $u, w \in V$ there is a u-w walk because such a walk can be obtained by concatenating a u-v_1 walk with a v_1-w walk.

Definition 3.17: Connected components in undirected graphs

Let $G = (V, E)$ be an undirected graph. A *connected component* of G is an induced subgraph $G[W]$ such that $G[W]$ is connected and $W \subseteq V$ is maximal with respect to this property, i.e., for every $X \subseteq V$ with $X \supsetneq W$ the graph $G[X]$ is not connected.

We emphasize the crucial notion of maximality, which is ubiquitous in Mathematical Optimization. Namely, a set W is *maximal* with respect to a particular property if there is no strict superset of W that also fulfills the property. This does not necessarily mean that W is a set of maximum cardinality among all the sets with the property. The converse, however, is true: If a set among all sets with a given property has maximum cardinality, then it is also maximal with respect to that property.

Each graph can be decomposed into its connected components. Figure 3.23 shows a graph with four connected components.

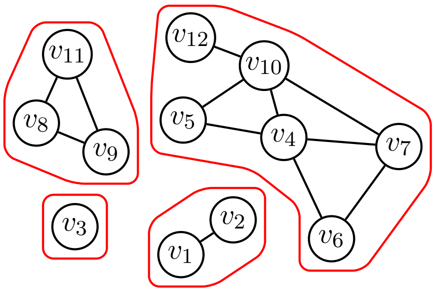


Figure 3.23: The above graph G has four connected components: $G[\{v_1, v_2\}]$, $G[\{v_3\}]$, $G[\{v_4, v_5, v_6, v_7, v_{10}, v_{12}\}]$, and $G[\{v_8, v_9, v_{11}\}]$.

Using BFS, one can also find the connected components of a graph. Starting from a starting vertex v_1, breadth-first search reaches exactly those vertices that are in the same connected component as v_1. Indeed, the vertices in the same connected component are precisely those that can be reached from v_1, i.e., those vertices v satisfying $d(v_1, v) < \infty$. So, if V_1 is the set of vertices reached by the breadth-first search, then $G[V_1]$ is a connected component of G. We can repeat this procedure for another starting vertex $v_2 \in V \setminus V_1$ to find a second connected component $G[V_2]$, and so on. A crude way to bound the running time of this procedure is by observing that we make $O(n)$ calls to BFS—because a graph has at most $n := |V|$ connected components—each of which uses $O(n + m)$ time by Theorem 3.14. This leads to an overall running time bound of $O(n(m + n))$.

Exercise 3.18: Running time improvement

Show that the connected components of an undirected graph $G = (V, E)$ can be found in running time $O(m + n)$. To achieve this, improve the running time analysis presented above and, if necessary, adapt BFS (Algorithm 2) for this setting.

We can analogously define connected components in directed graphs by disregarding arc directions. Again, for directed graphs, there is moreover a natural stronger notion of connected components.

Definition 3.19: Connected components in directed graphs

Let $G = (V, A)$ be a directed graph.

(i) The *connected components* of G are the connected components of the undirected graph G', which results from G by ignoring arc directions.

(ii) A *strongly connected component* of G is an induced directed subgraph $G[W]$ such that $G[W]$ is strongly connected and $W \subseteq V$ is maximal with respect to this property.

Figure 3.24 shows a directed graph with five strongly connected components.

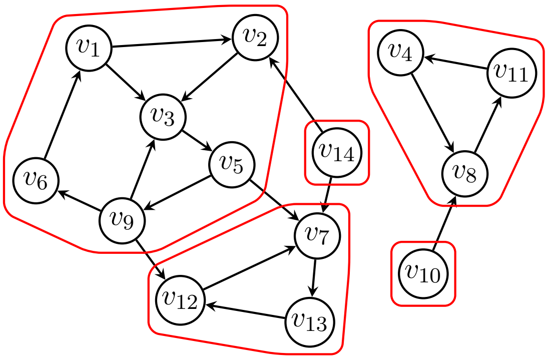


Figure 3.24: A graph G with two connected components and five strongly connected components (in red): $G[\{v_{10}\}]$, $G[\{v_1, v_2, v_3, v_5, v_6, v_9\}]$, $G[\{v_4, v_8, v_{11}\}]$, $G[\{v_7, v_{12}, v_{13}\}]$, and $G[\{v_{14}\}]$.

4 Flows and Cuts

Flows and cuts allow for modeling and solving a surprisingly large set of real-world problems. Furthermore, they can be used to answer many foundational questions related to graphs, for example, whether a graph admits k arc-disjoint paths between two vertices s and t. We consider flows in directed graphs. An undirected version can easily be derived from this. Throughout this chapter, we denote by $G = (V, A)$ a directed graph with arc capacities $u\colon A \to \mathbb{Z}_{\geq 0}$. Notice that we assume that arc capacities are integral. Non-negative rational arc-capacities can be transformed into integral ones by scaling, leading to properties and results analogous to the ones we discuss in the following.

4.1 Basic notions and relations

We start by defining the notion of an s-t flow, where $s, t \in V$, $s \neq t$. Intuitively, one can imagine G to be a network of water pipes where one wants to continuously send water from s to t. Each arc $a = (v, w) \in A$ is a pipe that can have a maximum throughput of $u(a)$ from vertex v to w. The quantity $u(a)$ is called the *capacity* of a. Assume that one can decide how water entering a vertex v is sent out on the outgoing arcs $\delta^+(v)$, subject to respecting the arc capacities u. An s-t flow then simply captures a possible steady-state mode for continuously sending water from s to t, by indicating how much water flows on each arc per time unit.

Definition 4.1: s-t flow / flow

Let $s, t \in V$, $s \neq t$. An s-t flow in G is a function $f\colon A \to \mathbb{R}_{\geq 0}$ satisfying the following conditions.

(i) *Capacity constraints*: $f(a) \leq u(a)\ \forall a \in A$.

(ii) *Balance constraints*: for $v \in V$,

$$f(\delta^+(v)) - f(\delta^-(v)) \begin{cases} = 0 & \text{if } v \in V \setminus \{s,t\} \ , \\ \geq 0 & \text{if } v = s \ , \\ \leq 0 & \text{if } v = t \ . \end{cases}$$

The *value* of a flow f is $\nu(f) := f(\delta^+(s)) - f(\delta^-(s))$.

The capacity constraints assure that the flow on each arc does not exceed the capacity. For a vertex $v \in V$, we call the quantity $f(\delta^+(v))$ the *outflow* of v. Similarly, $f(\delta^-(v))$ is called the *inflow* of v. Hence, the balance constraints can be rephrased as imposing that for each vertex $v \in V \setminus \{s, t\}$, the outflow of v equals the inflow into v. The balance constraints guarantee that whatever flows into some vertex $v \in V \setminus \{s,t\}$ also has to flow out of it. Furthermore, the outflow of s is at least the inflow into s, and the outflow of t is at most the inflow into t. In other words, no flow is created or destroyed at any vertex $v \in V \setminus \{s,t\}$. Furthermore, flow can be created at vertex s, i.e., there may be more outgoing flow than incoming one, and flow may be absorbed at vertex t. Therefore, the vertex s is often called the *source* and t is called the *sink* of the flow network. Also note that the value of a flow f in the analogy based on water pipes indicates how much flow is sent from s to t per time unit. Figure 4.1 shows an example of an s-t flow.

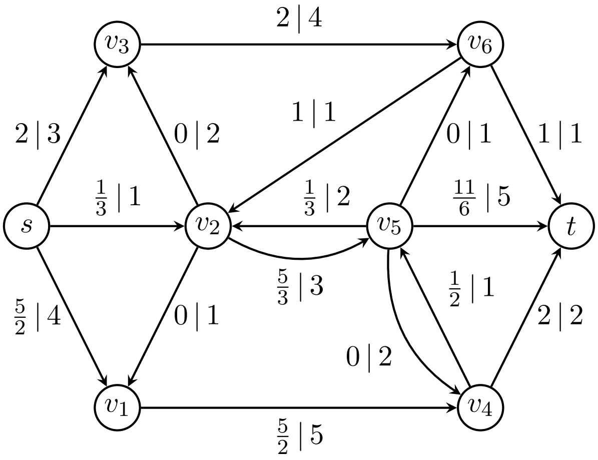


Figure 4.1: Example of an s-t flow. Next to each arc $a \in A$, one can find the value of the flow on a, i.e., $f(a)$, followed by its capacity $u(a)$, written as $f(a) \mid u(a)$. The value of the shown flow f is $\nu(f) = \frac{29}{6}$.

Notice that in the water pipe analogy we used, flow values correspond to how much water is sent *per time unit*, and the capacities limit the throughput, which is also measured in terms of water *per time unit*. Nevertheless, in the context of network flows, we simply talk about the *flow value* on an arc, and the *flow sent* from s to t, without repeatedly referring to a time component that may underlie the problem.

An s-t flow in G is called *maximum* if it has maximum value among all s-t flows in G. This leads to the maximum s-t flow problem, or simply the maximum flow problem, which is formally defined as follows.

Maximum flow problem, or maximum s-t flow problem

Input: A directed graph $G = (V, A)$, arc capacities $u\colon A \to \mathbb{Z}_{\geq 0}$, and $s, t \in V$, $s \neq t$.

Task: Find a maximum s-t flow in G, i.e., an s-t flow f that maximizes $\nu(f)$.

Hence, in the analogy of the water pipes, the maximum s-t flow problem asks to send water from s to t with maximum throughput. Before we present algorithms to solve this problem, we will derive some further results regarding different ways to measure the value of an s-t flow.

This will play an important role in understanding how to verify that a flow has maximum value, which is useful in the design of maximum s-t flow algorithms.

The value of a flow can be measured in several ways. We defined it as the difference between outflow of s and inflow into s, which is also called the *net outflow of* s. Since all vertices except for s and t have the same outflow as inflow, we know that the only place where the net outflow of s can be absorbed is at t. Hence, the net inflow of t must be equal to the net outflow of s, and is therefore also equal to the value of the flow $\nu(f)$. In the following, we want to formalize and generalize this reasoning, which will also play an important role in understanding how to verify that a flow has maximum value among all possible s-t flows. For this we recall the notion of s-t cut, to which we assign a *value* in the context of flow problems.

Definition 4.2: s-t cut

An s-t cut is a set $C \subseteq V$ such that $s \in C$ and $t \notin C$. Furthermore, in the context of a maximum flow problem with capacities $u \colon A \to \mathbb{Z}_{\geq 0}$, the *value* of an s-t cut C is defined as $u(\delta^+(C))$. An s-t cut C is called *minimum* if it has minimum value among all s-t cuts.

The following lemma shows that any s-t cut $C \subseteq V$ can be used to measure the value of an s-t flow f .

Lemma 4.3: Value of a flow expressed via an s-t cut

Let f be an s-t flow and $C \subseteq V$ an s-t cut. Then

$$\nu(f) = f(\delta^+(C)) - f(\delta^-(C)) \ .$$

Proof. By condition (ii) (balance constraints) in the definition of an s-t flow f, we have $f(\delta^+(v)) - f(\delta^-(v)) = 0$ for all $v \in V \setminus \{s,t\}$. Thus

$$\begin{aligned}\nu(f) &= f(\delta^+(s)) - f(\delta^-(s)) + \sum_{v \in C \setminus \{s\}} \underbrace{\Big(f(\delta^+(v)) - f(\delta^-(v))\Big)}_{=0} \\ &= \sum_{v \in C} \Big(f(\delta^+(v)) - f(\delta^-(v))\Big) \\ &= f(\delta^+(C)) - f(\delta^-(C)) \ ,\end{aligned}$$

as required. □

Exercise 4.4: Value of a flow

Show that the value of any s-t flow f is equal to the difference between inflow into t and outflow of t, i.e., $\nu(f) = f(\delta^-(t)) - f(\delta^+(t))$.

Additionally to being a means of measuring the value of an s-t flow, an s-t cut also naturally leads to an upper bound on the value of *any* s-t flow. This central result is known as the *weak max-flow min-cut theorem* and can be seen to be a special case of weak duality of linear programming.

Theorem 4.5: Weak max-flow min-cut theorem

Let f be an s-t flow and let $C \subseteq V$ be an s-t cut. Then

$$\nu(f) \leq u(\delta^+(C)) \ .$$

In other words, the value of a maximum s-t flow is upper bounded by the value of a minimum s-t cut.

Proof. Using Lemma 4.3, we obtain

$$\nu(f) = \underbrace{f(\delta^+(C))}_{\leq u(\delta^+(C))} - \underbrace{f(\delta^-(C))}_{\geq 0} \leq u(\delta^+(C)) \ ,$$

where $f(\delta^+(C)) \leq u(\delta^+(C))$ and $f(\delta^-(C)) \geq 0$ follow from $0 \leq f(a) \leq u(a)\ \forall a \in A$. □

Hence, if we find an s-t flow and an s-t cut with identical values, then Theorem 4.5 implies that the s-t flow is maximum and the s-t cut is minimum. The (strong) max-flow min-cut theorem, which we will see later, guarantees that such a pair of s-t flow and s-t cut with matching values always exists.

Remark 4.6: Infinite capacities

Even though in our definition of the maximum s-t flow problem, capacities are assumed to be non-negative integers, it is common to allow that some arcs $a \in A$ have infinite capacity, i.e., $u(a) = \infty$. This case can easily be reduced to the one with finite capacities. Notice that when allowing infinite capacities, an s-t flow of infinite value may exist, which can easily be checked by using BFS only over the arcs with infinite capacity. If no infinitely large s-t flow exists, then one can for example replace the infinite capacities by a sufficiently large capacity that is at least as large as the value of a maximum s-t flow. One can use for example the value of any finite-valued s-t cut, because this is an upper bound on the maximum flow value by Theorem 4.5. All vertices reachable from s by arcs of infinite capacity form such a cut that can easily be found by BFS. Due to this simple reduction, we will sometimes use infinite capacities to model problems later on, because they express in a clean and easy-to-parse way that some arcs do not have any relevant upper bound on the flow that may traverse them.

We next discuss a first algorithm to solve maximum flow problems, namely the Ford-Fulkerson algorithm. Theorem 4.5 plays a crucial role in proving that this algorithm indeed works. More precisely, one can exhibit from the Ford-Fulkerson algorithm not only an s-t flow f , but also an s-t cut C, such that $\nu(f) = u(\delta^+(C))$. By Theorem 4.5, this will immediately imply that the s-t flow is maximum and that the s-t cut is minimum. Since the Ford-Fulkerson algorithm always terminates with such a flow/cut pair, we can then derive from Theorem 4.5 the strong max-flow min-cut theorem mentioned above.

4.2 Algorithm of Ford-Fulkerson and strong max-flow min-cut theorem

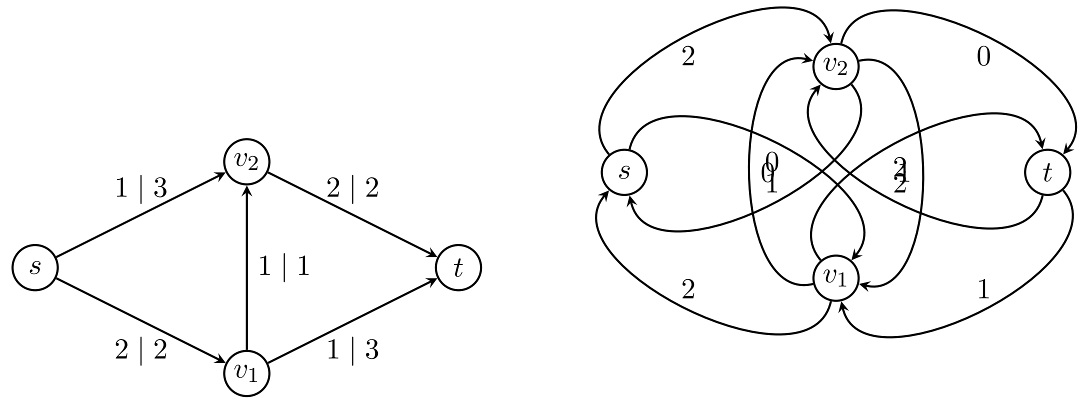
Ford and Fulkerson's max-flow algorithm starts with the *zero flow*, which is the s-t flow f defined by $f(a) = 0\ \forall a \in A$, and modifies it iteratively. In each iteration, the value of the flow strictly increases by at least one unit until a maximum flow is obtained. To find a way to augment an s-t flow f to an s-t flow with strictly larger value, Ford and Fulkerson's algorithm works on a *residual graph*, defined as follows.

Definition 4.7: f-residual graph & f-residual capacities

Let f be an s-t flow in G. The *f-residual graph* $G_f = (V, B)$ with *f-residual capacities* $u_f \colon B \to \mathbb{Z}_{\geq 0}$ is defined as follows. The set of arcs $B := A \cup A^R$ contains all original arcs A together with reverse arcs A^R , where for $a \in A$, the set A^R contains an arc a^R that is *antiparallel* to a, i.e., the head of a^R is the tail of a and vice versa. Furthermore,

$$u_f(b) := \begin{cases} u(b) - f(b) & \text{if } b \in A \ , \\ f(a) & \text{if } b = a^R \in A^R \ . \end{cases}$$

Figure 4.2 Illustrates the notion of f-residual graph and f-residual capacities.



(a) A graph $G = (V, A)$ with flow f and capacities u specified in the form $f(a) \mid u(a)$.

(b) The f-residual graph $G_f = (V, B)$ with f-residual capacities u_f.

Figure 4.2: A flow f and the corresponding f-residual graph G_f.

Notice, an arc $a \in A$ has a residual capacity of zero, i.e., $u_f(a) = 0$, if and only if $f(a) = u(a)$. Such an arc is called *f-saturated*, or simply *saturated*.

A key observation used in the Ford-Fulkerson algorithm is the fact that if there is an s-t path $P \subseteq B$ in the residual graph $G_f = (V, B)$ such that each arc of P has strictly positive residual capacity, then there is a simple operation to increase the flow f by at least one unit. Such an s-

t path in G_f is called an *f-augmenting path*, or simply *augmenting path*, and is formally defined as follows.

Definition 4.8: *f*-augmenting path/augmenting path

Let f be an s-t flow in G. An *f-augmenting* path $P \subseteq B$ is an s-t path in $G_f = (V, B)$ with $u_f(b) > 0 \; \forall b \in P$.

Once an f-augmenting path is found, one can augment the value of the flow f as follows.

Definition 4.9: Augmentation

The *augmentation* of an s-t flow f in G along an f-augmenting path $P \subseteq B$, where $G_f = (V, B)$ is the f-residual graph, is the flow f' in G defined as

$$f'(a) = \begin{cases} f(a) + \gamma & \text{if } a \in P \;\;, \\ f(a) - \gamma & \text{if } a^R \in P \;\;, \\ f(a) & \text{if } a, a^R \notin P \;\;, \end{cases}$$

where $\gamma := \min\{u_f(b) \colon b \in P\} > 0$. We call the value γ the *augmentation volume* of the augmenting path P.

One can easily verify that the augmentation f' of f is indeed a flow in G of value $\nu(f') = \nu(f) + \gamma$. Furthermore, finding an augmenting path is easy. This can be done with breadth-first search, as an augmenting path is simply an s-t path in the network (V, B'), where $B' = \{b \in B \colon u_f(b) > 0\}$ are all arcs of the residual graph with strictly positive residual capacities. We thus obtain the following.

Lemma 4.10: Running time for finding *f*-augmenting paths

Let f be an s-t flow in G and denote the number of arcs and vertices in G by m and n, respectively. If there is an f-augmenting path, then such a path can be found in $O(m+n)$ time via breadth-first search.

Ford and Fulkerson's maximum flow algorithm starts with a zero flow and augments that flow via augmenting paths for as long as possible. Algorithm 3 describes Ford and Fulkerson's procedure. Figure 4.3 illustrates the Ford-Fulkerson algorithm on an example.

There are still several points to be discussed concerning the correctness and running time of Ford and Fulkerson's algorithm. A priori, it is not even clear whether the algorithm terminates, because one may never leave the while-loop in the second step and always find new augmenting paths. It turns out that if the initial capacities are not required to be integral but can be any non-negative real numbers—in particular, irrational ones—then this can indeed happen. In such bad examples one can indefinitely find augmenting paths and augment along them. This seems somehow counter-intuitive, as the value of the flow increases each time we augment. The prob-

Algorithm 3: Ford and Fulkerson's algorithm to find a maximum s-t flow

```
Input  : Directed graph $G = (V, A)$ with arc capacities $u\colon A \to \mathbb{Z}_{\geq 0}$ and $s, t \in V$, $s \neq t$.
Output: A maximum $s$-$t$ flow $f$.
  1. Initialization:
     $f(a) = 0 \; \forall a \in A$.
  2. while ($\exists$ $f$-augmenting path $P$ in $G_f$) do:
        Augment $f$ along $P$ and set $f$ to be the augmented flow.
  3. return $f$.
```

lem lies in the fact that, when having arbitrary real capacities, a bad example can be built such that the augmentation volumes become smaller over the iterations and the value of the s-t flow remains bounded, despite an infinite number of iterations. However, when dealing with integral (or rational) capacities $u\colon A \to \mathbb{Z}_{\geq 0}$—which is the case we are interested in and we recall that integrality of the capacities is assumed in our definition of the maximium flow problem—this cannot happen as we will show next.

Lemma 4.11: Positive integral augmentation volume

Each augmentation step of Ford and Fulkerson's algorithm has an integral augmentation volume of at least one unit.

Proof. By definition, the augmentation volume is strictly positive. Thus, it suffices to prove that it is additionally integral. We claim that each flow f encountered at the beginning and end of one iteration of step 2 is integral. This can easily be seen by induction. We start with the zero flow, which is integral. Whenever we construct the residual graph G_f for an integral flow f, the residual capacities u_f are integral, because f and u are integral. This implies that no matter which augmenting path we choose, the augmentation volume is integral. Therefore, the updated flow will be integral too.

Hence, all encountered flows f during Ford and Fulkerson's algorithm are integral, and therefore, all augmentation volumes are integral too, as already discussed above. □

From Lemma 4.11 we can easily derive bounds on the number of augmentation steps of Ford and Fulkerson's algorithm, and therefore also on its running time.

Corollary 4.12: Running time bound for Ford and Fulkerson's algorithm

The number of augmentation steps in Ford and Fulkerson's algorithm is bounded by the value $\alpha \in \mathbb{R}_{\geq 0}$ of a maximum s-t flow in G. Hence, the running time of Ford and Fulkerson's algorithm is bounded by $O(\alpha \cdot (m + n))$, assuming $\alpha \geq 1$.

Notice that the above running time bound depends on an unknown quantity $\alpha > 0$. However, there are different ways how we can upper bound α to obtain a running time bound without unknowns. A straightforward bound is $\alpha \leq u(A) := \sum_{a \in A} u(a)$. Often, stronger bounds can be obtained through Theorem 4.5 by using the value of a well-chosen s-t cut to bound α. A canonical candidate is often the singleton s-t cut $C = \{s\}$.

When using the trivial bound $\alpha \leq u(A)$, we therefore obtain a running time bound of $O(u(A) \cdot (m+n))$, where we assume $u(A) > 0$. It turns out that this bound can be tight for some examples, i.e., there are examples where Ford and Fulkerson's algorithm indeed needs $\Theta(u(A) \cdot (m+n))$ elementary operations. Notice that this running time bound is not polynomial in the input size, because $u(A)$ is not polynomial in the input size. We recall that to store a positive number, like $u(a)$ for some $a \in A$, we only need $\Theta(\log u(a))$ bits. Hence, $u(a)$ can be exponential in the input size. Still, whenever the maximum capacity is small, say bounded by a polynomial in n, then the bound $O(\alpha \cdot (m+n))$ is polynomial in the input size, implying that Ford and Fulkerson's algorithm runs efficiently on such instances. This covers many interesting applications as we will see later. In the following, we will show the correctness of Ford and Fulkerson's algorithm. We will prove this together with further properties that imply the (strong) max-flow min-cut theorem.

Theorem 4.13

Let f be an s-t flow in G. Then the following are equivalent:

(i) f is a maximum s-t flow.
(ii) There does not exist an f-augmenting path in G_f.
(iii) There exists an s-t cut $C \subseteq V$ with $u(\delta^+(C)) = \nu(f)$.

Furthermore, a minimum s-t cut can be found in linear time given a maximum s-t flow.

Before proving Theorem 4.13, we discuss some of its various implications. First, Theorem 4.13 together with Theorem 4.5 implies that the value of a maximum s-t flow is always equal to the value of a minimum s-t cut. This is called the *strong max-flow min-cut theorem*, or simply the *max-flow min-cut theorem*.

Corollary 4.14: Strong max-flow min-cut theorem

The value of a maximum s-t flow in G is equal to the value of a minimum s-t cut in G:

$$\max\{\nu(f) \colon f \text{ is } s\text{-}t \text{ flow in } G\} = \min\left\{u(\delta^+(C)) \colon C \subseteq V, s \in C, t \notin C\right\} \enspace .$$

Consequently, Corollary 4.14 implies that the maximality of an s-t flow can always be shown by specifying a minimum s-t cut because the maximum value of an s-t flow is equal to the value of a minimum s-t cut.

Furthermore, Theorem 4.13 also implies that, as soon as one obtains an s-t flow f such that there are no f-augmenting paths, then f has maximum value. This immediately implies that Ford and Fulkerson's algorithm indeed returns a maximum s-t flow, because we already know that Ford and Fulkerson's algorithm terminates, as stated in Corollary 4.12, and we recall that the only way for the algorithm to terminate is the inexistence of further f-augmenting paths.

Corollary 4.15: Correctness of Ford and Fulkerson's algorithm

Ford and Fulkerson's algorithm returns a maximum s-t flow.

We will now provide a proof of Theorem 4.13.

Proof of Theorem 4.13. In order to prove the three equivalences in Theorem 4.13 we show the implications

$$\text{(i)} \implies \text{(ii)} \implies \text{(iii)} \implies \text{(i)} \enspace .$$

(i) ⇒ (ii): We show the contraposition. The existence of an f-augmenting path P in G implies that the flow f is not maximum because augmenting f along P leads to a larger s-t flow.

(ii) ⇒ (iii): Let $C \subseteq V$ be the set of all vertices that can be reached in G_f from s via the arcs $b \in B = A \cup A^R$ with $u_f(b) > 0$. Obviously, $s \in C$, and moreover $t \notin C$ because by assumption there are no f-augmenting paths in G_f. Hence, C is an s-t cut. By definition of C, there cannot be an arc b in G_f that is leaving C and satisfies $u_f(b) > 0$. By definition of u_f, this is equivalent to all arcs $a \in \delta^+_G(C)$ being f-saturated and all arcs $a \in \delta^-_G(C)$ satisfying $f(a) = 0$. Hence, the value of the s-t cut C satisfies

$$u(\delta^+_G(C)) = f(\delta^+_G(C)) = f(\delta^+_G(C)) - \underbrace{f(\delta^-_G(C))}_{=0} = \nu(f) \enspace .$$

(iii) ⇒ (i): By the weak max-flow min-cut theorem (Theorem 4.5), point (iii) implies that f is a maximum s-t flow and C is a minimum s-t cut.

This proves the desired equivalences. Finally, the construction of a minimum s-t cut $C \subseteq V$ can be performed by following the approach used in the second part of this proof, i.e., C can be chosen to be all vertices reachable from s in G_f by using only arcs with strictly positive f-residual capacity. This can be done in $O(m + n)$ time via BFS. □

4.3 Integrality of s-t flows

The above discussion of Ford and Fulkerson's maximum s-t flow algorithm has a further important implication that is heavily exploited in Combinatorial Optimization, namely that there is a maximum s-t flow that is integral, and we can find such an s-t flow with Ford and Fulkerson's algorithm.

Theorem 4.16: Integral maximum flows

Let $G = (V, A)$ be a directed graph with capacities $u \colon A \to \mathbb{Z}_{\geq 0}$, and let $s, t \in V$, $s \neq t$. Ford and Fulkerson's algorithm finds a maximum s-t flow that is integral.

Proof. The s-t flow f returned by Ford and Fulkerson's algorithm is maximum by Corollary 4.15. Furthermore, f is integral because each augmentation increases the flow by an integral amount. □

The fact that maximum s-t flows can be chosen to be integral allows for using s-t flows to model a wide variety of combinatorial optimization problems through flow problems. A crucial advantage of having integral flows lies in the ability to interpret an integral s-t flow combinatorially. For example, if all capacities are either 0 or 1, then a maximum s-t flow can be interpreted as a subset of the arcs, namely the ones that carry one unit of flow. We will give applications of this in the next section. Finally, we want to highlight that the integrality of s-t flows is one of the main reasons why s-t flows are studied so heavily in Combinatorial Optimization.

4.4 Applications of s-t flows

There is a very broad set of applications of flows and cuts. As we already mentioned, one reason for this is the existence of optimal integral flows, but this is by far not the only one. The existence of extremely fast algorithms to solve even very large flow problems is another key aspect. In general, state-of-the-art flow algorithms run significantly faster in practice then their worst-case guarantees, often reaching a roughly linear running time. Due to this, flows and cuts have found numerous applications in large-scale problems. Additionally, several classical theoretical results linked to graphs and graph optimization can be elegantly derived via flows, their integrality, and the strong max-flow min-cut theorem. In the following, we present a selection of several applications of flows and cuts that reflect their versatility and broad applicability.

4.4.1 Arc-connectivity

We already discussed basic connectivity properties of graphs. Often, one is not only interested in knowing whether a directed graph is strongly connected, i.e., whether every vertex can be reached from every other vertex, but also how "well" two vertices are connected. A typical way to measure this is by checking how many arc-disjoint paths there are between two vertices s and t. Using this stronger notion of connectivity leads to the notion of *k-arc-connected* graphs.

Definition 4.17: k-arc-connectivity

A directed graph $G = (V, A)$ is *k-arc-connected* if for any two vertices $s, t \in V, s \neq t$, there are at least k arc-disjoint s-t paths in G.

Notice that a 1-arc-connected graph is simply a strongly connected graph. We can check whether a graph is strongly connected via breadth-first search. A key question is how to check whether a graph is k-arc-connected. This can be reduced to the question of computing a maximum number of arc-disjoint paths from s to t for two arbitrary distinct vertices $s, t \in V$, by checking whether each pair of vertices is k-arc-connected. Interestingly, this question can be solved easily via maximum s-t flows, by introducing unit capacities on all the arcs and computing a maximum s-t flow, as highlighted in Algorithm 4. We will show that the value of this s-t flow corresponds to the number of arc-disjoint s-t paths.

We show in two steps that Algorithm 4 works correctly. First, we observe that whenever G contains k arc-disjoint s-t paths, then there is an s-t flow of value k in the flow network used in

Algorithm 4: Determining maximum number of arc-disjoint s-t paths.

```
Input: A directed graph $G = (V, A)$ and vertices $s, t \in V$, $s \neq t$.
Output: Maximum number of arc-disjoint $s$-$t$ paths in $G$.
    1. Define unit capacities $u\colon A \to \mathbb{Z}_{\geq 0}$, i.e., $u(a) = 1\ \forall a \in A$.
    2. Compute a maximum $s$-$t$ flow $f$ in $G$ with capacities given by $u$.
    3. return $\nu(f)$, the value of $f$.
```

Algorithm 4.

Observation 4.18: From arc-disjoint paths to flows

Let $G = (V, A)$ be a directed graph and $s, t \in V$ with $s \neq t$. Let $u\colon A \to \mathbb{Z}_{\geq 0}$ be unit capacities, i.e., $u(a) = 1\ \forall a \in A$. If there are k arc-disjoint s-t paths $P_1, \dots, P_k \subseteq A$ in G, then an s-t flow f of value k is obtained by setting

$$f(a) := \begin{cases} 1 & \text{if } a \in \bigcup_{i=1}^k P_i \ , \\ 0 & \text{otherwise} \ . \end{cases}$$

Second, we show that if there is an s-t flow of value k in the flow network used in Algorithm 4, then there exist k arc-disjoint s-t paths. This is implied by the following constructive theorem, which shows how to convert an integral s-t flow of value k in a unit-capacity graph into k arc-disjoint s-t paths.

Theorem 4.19: Path decomposition

Let f be an integral s-t flow in G with unit capacities u and let $k = \nu(f)$. Then, one can find in linear time (i.e., $O(m+n)$ time), k arc-disjoint s-t paths $P_1, \dots, P_k \subseteq A$ with $\bigcup_{i=1}^k P_i \subseteq \{a \in A\colon f(a) = 1\}$.

Proof. Let $U = \{a \in A\colon f(a) = 1\}$. We will find k arc-disjoint s-t walks within U. The result then follows by shortcutting the walks into paths, i.e., by deleting directed cycles within a walk. We use induction on k. The case $k = 0$ is trivial; hence, we assume $k \in \mathbb{Z}_{\geq 1}$. We construct the first s-t walk $W \subseteq U$ in our family of arc-disjoint s-t walks by using Algorithm 5.

To show that Algorithm 5 indeed finds an arc-disjoint s-t walk, we have to show that whenever we are at the beginning of a new iteration of the while-loop and $v \neq t$, then $\exists (v, w) \in U \setminus W$. This then implies that the algorithm will stop with an s-t walk.

When considering a vertex $v \in V \setminus \{s, t\}$ in Algorithm 5, then, by construction, $W \subseteq U$ is an s-v walk. The set W contains one more incoming arc into v than outgoing ones. Because U stems from a $\{0, 1\}$ s-t flow, the number of arcs in U that enter v is equal to the number of arcs in U that leave v. Therefore, there must exist an arc $(v, w) \in U \setminus W$, as desired.

We can follow a similar argument for $v = s$. Whenever the algorithm considers the vertex $v = s$, then $W \subseteq U$ is a closed walk containing s. Hence, W contains as many arcs entering s

Algorithm 5: Determining an s-t walk W

```
Input: A directed graph $G = (V, U)$, vertices $s, t \in V$, $s \neq t$, and an $s$-$t$ flow $f\colon U \to \{0,1\}$
  with value $\nu(f) \geq 1$.
Output: An arc-disjoint $s$-$t$ walk $W \subseteq U$.
  1. Initialization:
     $v = s$.
     $W = \emptyset$.
  2. while $v \neq t$ do:
       Choose an arc $(v, w) \in U \setminus W$.
       $W = W \cup \{(v, w)\}$.
       $v = w$.
  3. return $W$.
```

as there are arcs leaving s. However, because U stems from a $\{0,1\}$ s-t flow with value $k > 0$, the arc set U contains exactly k arcs more that leave s than arcs entering s. As $k > 0$, there exists an arc $(s, w) \in U \setminus W$.

Thus, we can find an s-t walk in $G' = (V, U)$ through Algorithm 5. The other $k-1$ required s-t walks can be found inductively among the remaining arcs $U \setminus W$ because these arcs correspond to the flow f', who is defined as follows:

$$f'(a) := \begin{cases} 1 & \text{if } a \in U \setminus W \ , \\ 0 & \text{otherwise} \ , \end{cases}$$

and has value $k-1$. Hence, the result follows by induction. The bound for the running time applies because every arc in U occurs at most once as an arc (v, w) in step 2 of a call to Algorithm 5 in the course of the construction of s-t walks according to Algorithm 5. □

Combining Observation 4.18, Theorem 4.19, and the fact that there is an integral maximum s-t flow due to Theorem 4.16, we obtain the following result.

Corollary 4.20: Correctness of Algorithm 4

Algorithm 4 correctly determines the maximum number of arc-disjoint s-t paths. Furthermore, by Theorem 4.19, one can find a maximum set of arc-disjoint s-t paths in linear time once an integral maximum s-t flow is found.

Notice that we need the existence of an integral maximum s-t flow to be able to invoke Theorem 4.19. However, this is just part of the proof. If we only want to determine the maximum number of arc-disjoint s-t paths, then any maximum s-t flow algorithm can be used, even if a non-integral maximum flow is returned. However, if we want to compute a maximum number of arc-disjoint s-t paths, then we need to obtain an integral flow to invoke Theorem 4.19 constructively.

Moreover, notice that Algorithm 4 is an efficient procedure when using Ford and Fulkerson's maximum s-t flow procedure. This is despite the fact that for general maximum s-t flow problems, Ford and Fulkerson's algorithm is not efficient, as already discussed. The reason for this is that the maximum flow in a unit-capacity network with m arcs is at most m. Hence, by Corollary 4.12, the running time of Algorithm 4, when implemented with Ford and Fulkerson's maximum flow algorithm, is bounded by $O(m(m+n))$. Thus, we can answer the question whether a given directed graph is k-arc-connected in polynomial time.

Notice that Corollary 4.20 states that the arc-connectivity between s and t is equal to the value of a maximum s-t flow in the unit-capacity graph considered in Algorithm 4. Hence, by the strong max-flow min-cut theorem, this leads to the following version of Menger's Theorem.

Theorem 4.21: Menger's Theorem

Let $G = (V, A)$ be a directed graph with $s, t \in V, s \neq t$. Then the maximum number of arc-disjoint s-t paths is equal to the number of arcs in a minimum cardinality s-t cut.

In the above theorem, a minimum cardinality s-t cut refers to a minimum s-t cut with respect to unit capacities. Hence, its value is $\min\{|\delta^+(C)| \colon C \subseteq V, s \in C, t \notin C\}$.

There are many versions of Menger's Theorem, which can be derived through variations of the above reasoning. In particular, the analogous theorems can be stated for undirected graphs and also for vertex-disjoint paths. We will get back to variations of Menger's Theorem in the problem sets.

Arc-connectivity can be generalized to undirected graphs. In this case, k-edge-connectivity is defined as follows.

Definition 4.22: k-edge-connectivity in undirected graphs

An undirected graph $G = (V, E)$ is *k-edge-connected* if for any two vertices $s, t \in V$ with $s \neq t$ there exist at least k edge-disjoint s-t paths in G.

Exercise 4.23: Checking k-edge-connectivity in undirected graphs

Reduce the problem of checking k-edge-connectivity in undirected graphs to the problem of checking k-arc-connectivity in directed graphs.

4.4.2 Maximum cardinality matchings in undirected graphs

Consider the following problem. Assume that there is a set of workers X that have to perform a set of tasks Y. Not every worker can perform every task, and each task is a one day job for a worker with the appropriate skill set. We can represent the relation of which workers can perform which tasks as a graph $G = (V, E)$, where $V = X \mathbin{\dot\cup} Y$ and an edge $\{x, y\} \in E$ between worker $x \in X$ and task $y \in Y$ means that x has the right skill set to perform task y. Our goal is to find an assignment of tasks to workers such that a maximum number of tasks is performed within the considered day. This problem can be solved by computing a maximum s-t flow in an appropriate auxiliary network, as we discuss next.

We start with some basic terminology to highlight some important structure of the problem and its solution. First, notice that the graph G is not a general undirected graph, but it has an additional property. Namely, each edge goes from a vertex in X to a vertex in Y, with $X \cap Y = \emptyset$. Such graphs are called *bipartite*, and one can easily check that they cannot have any odd cycles. It turns out that the non-existence of odd cycles even characterizes them.

Definition 4.24: Bipartite graph

An undirected graph $G = (V, E)$ is called *bipartite* if there is a bipartition of its vertices $V = X \mathbin{\dot\cup} Y$ such that each edge has one endpoint in X and the other in Y.

Figure 4.4 shows an example of a bipartite graph. If G has several connected components, then there are several ways to partition V into $X \mathbin{\dot\cup} Y$ such that all edges go between X and Y. For simplicity, we sometimes say that $G = (V, E)$ is a bipartite graph *with bipartition* $V = X \mathbin{\dot\cup} Y$ to denote an arbitrary bipartition of the vertices such that all edges go between X and Y.

Exercise 4.25

Show that a graph is bipartite if and only if it does not contain an odd cycle. We recall that a loop is also considered to be an odd cycle.

Notice that a solution to our assignment problem can be represented by a subset $M \subseteq E$ of the edges that indicates which tasks are assigned to which workers. More formally, an edge $\{x, y\} \in M$ is interpreted as an assignment of worker x to task y. Notice that because every worker can be assigned to at most one task and every task can be performed by at most one worker, a solution set M must be such that no worker or task has two edges of M incident with it, i.e., no vertex in the subgraph (V, M) has degree bigger than 1. Such an edge set M is called a *matching*.

Definition 4.26: Matching

Let $G = (V, E)$ be an undirected graph. A set $M \subseteq E$ is a *matching* in G if M does not contain loops and no two edges of M share a common endpoint.

Hence, finding a maximum assignment of workers to possible jobs can be rephrased as finding a maximum cardinality matching in the bipartite graph $G = (X \mathbin{\dot\cup} Y, E)$. In the following, we show how to solve such problems efficiently by computing an integral maximum s-t flow. For completeness, we first give a formal description of the problem.

Maximum cardinality matching problem in bipartite graphs

Input: An undirected bipartite graph $G = (V, E)$.

Task: Find a maximum cardinality matching $M \subseteq E$ in G.

In the following discussion, where we show how to solve the maximum cardinality bipartite matching problem, we reuse the notation and terminology that we introduced for the worker-task assignment problem. Hence, our bipartite graph is $G = (V, E)$ with bipartition $V = X \dot{\cup} Y$. Consider the following auxiliary graph $H = (W, F)$, which is directed. It is obtained from G by directing all edges of G from X to Y. Furthermore, we introduce two new vertices s and t. For each vertex $x \in X$ we add an arc (s, x), and for each vertex $y \in Y$ we add an arc (y, t). Hence, formally, the graph H is defined as follows:

$$\begin{aligned} &W = V \cup \{s, t\} \ , \text{ and} \\ &F = \{(x, y) \colon x \in X, y \in Y, \{x, y\} \in E\} \cup \{(s, x) \colon x \in X\} \cup \{(y, t) \colon y \in Y\} \ . \end{aligned}$$

Figure 4.5 shows the graph H that corresponds to the example bipartite graph depicted in Figure 4.4.

The capacities $u \colon F \to \mathbb{Z}_{\geq 0}$ are set to $u(a) = 1$ for all $a \in F$. We now determine a maximum integral s-t flow f in H. Let

$$M := \{\{x, y\} \in E \colon x \in X, y \in Y, f((x, y)) = 1\}$$

be the set of all edges in G whose corresponding arcs in H carry one unit of flow. We prove the following claim.

Claim 4.27

M is a maximum cardinality bipartite matching in G.

Proof. First notice that M is a matching. For this we show that every vertex of V is incident with at most one edge of M. Indeed, consider any vertex $x \in X$; the reasoning for $y \in Y$ is analogous. Because x has only one incoming arc in the flow network H, namely the arc (s, x), which has capacity 1, it is impossible to have two arcs going out of x that both carry a unit of flow. This would contradict the balance constraint at x. Thus at most one of the outgoing arcs of x has flow 1., i.e., x has degree at most 1 in M. Thus, M is indeed a matching. Furthermore, we can compute the value $\nu(f)$ of f by using Lemma 4.3 with $C = \{s\} \cup X$, i.e.,

$$\nu(f) = f(\delta^+(C)) - f(\delta^-(C)) = f(\delta^+(C)) = |M| \ ,$$

where the second equality follows from $\delta^-(C) = \emptyset$, and the last one from the definition of M and the fact that f is a $\{0, 1\}$-flow due to its integrality and the unit capacities. In summary, any integral s-t flow in H of value k corresponds to a matching of cardinality k.

Conversely, every matching $M' \subseteq E$ of cardinality k can easily be transformed into an s-t flow of value k in the graph H. We start with the zero flow f, and for every arc $\{x, y\} \in M'$, we set the flow values on the arcs of the path $s \to x \to y \to t$ to one unit. Hence, whenever there is a matching of cardinality k in G, there is an s-t flow of value k in H.

The above showed that there is a one-to-one correspondence between matchings in G and integral s-t flows in H with the cardinality of a matching being equal to the value of the corresponding s-t flow. Hence, the matching M constructed via the s-t flow problem is indeed a maximum cardinality matching in G. □

Hence, we can solve the matching problem with Ford and Fulkerson's algorithm. With regard to the running time, observe that the auxiliary graph H has size polynomially bounded in the input size. Moreover, all arcs have capacity 1, and thus, the sum of all capacities is a polynomial upper bound for the value of a maximum flow. By Corollary 4.12, it follows that the proposed algorithm for finding a maximum cardinality bipartite matching is efficient when using Ford and Fulkerson's algorithm to find an integral maximum s-t flow.

Exercise 4.28: Two jobs per day

Consider our worker-task assignment problem. Assume that every worker can perform up to two tasks within the same day. As before, only one worker can be assigned per job. Show how this problem can still be solved efficiently, by adapting the approach presented above.

When interpreting the allocation of workers to jobs, it is a natural question under which circumstances it is possible to simultaneously assign all workers in X. Formally, in a bipartite graph $G = (V, E)$ with bipartition $V = X \mathbin{\dot\cup} Y$, we are looking for sufficient and necessary conditions for the existence of a matching $M \subseteq E$ that touches all vertices in X, i.e., matchings M such that every vertex x is incident with an edge in M. Obviously $|Y| \geq |X|$ is a necessary condition, although this condition is certainly not sufficient. It is not hard to see that the following restriction is also necessary: For each set of $X_0 \subseteq X$, the number of neighbors that X_0 has in Y must be at least $|X_0|$. If we define $N(X_0) \subseteq Y$ as the set of neighbors from X_0 in Y, i.e.,

$$N(X_0) \coloneqq \{y \in Y \colon \exists x \in X_0 \text{ with } \{x, y\} \in E\} \quad ,$$

then we can write the condition as $|N(X_0)| \geq |X_0|$ for all $X_0 \subseteq X$. The condition is necessary because $N(X_0)$ contains all jobs that workers in X_0 can perform. For everyone to work simultaneously, $N(X_0)$ must have at least as many jobs as there are workers in X_0. As it turns out, this condition, known as *Hall's condition*, is also sufficient. This is the statement of Hall's Theorem which, as we will see, can be considered as an implication of the (strong) max-flow min-cut theorem (Corollary 4.14).

Theorem 4.29: Hall's Theorem

Let $G = (V, E)$ be a bipartite graph with bipartition $V = X \mathbin{\dot\cup} Y$. Then there exists a matching $M \subseteq E$ in G that touches all vertices in X if and only if

$$|N(X_0)| \geq |X_0| \qquad \text{for all } X_0 \subseteq X \quad .$$

Proof. As discussed, Hall's condition is clearly necessary for the existence of a matching M that touches all vertices in X. We still have to prove sufficiency of the condition.

To this end, we consider an s-t flow problem in the same auxiliary graph $H = (W, F)$ that we constructed to solve the bipartite matching problem (see Figure 4.5). As already shown, the maximum cardinality matching in G is equal to the value of a maximum s-t flow in H. In order to prove Hall's Theorem, it suffices to conclude from Hall's condition that there exists an s-t flow of value $|X|$ in H. Note that no flow can have a value greater than $|X|$, because the total capacity of all arcs leaving s is $|X|$. The max-flow min-cut theorem (Corollary 4.14) implies that the value of a maximum s-t flow in H equals the value of a minimum s-t cut in H. In order to show the result, we thus prove that the value of a minimum s-t cut $C \subseteq W$ in H is $|X|$.

Let $C \subseteq W$ be a minimum s-t cut in H, and we define $C_X := C \cap X$ and $C_Y := C \cap Y$. We first show that, without loss of generality, we can assume that C fulfills

$$N(C_X) \subseteq C_Y \ . \tag{4.1}$$

If not, we can replace C by $C \cup N(C_X)$ without increasing the value of the cut. To show this, we observe how the value of the s-t cut C changes when we add $y \in N(C_X) \setminus C_Y$ to C to obtain $C' = C \cup \{y\}$. There is one new arc in the cut, namely $\delta^+(C') \setminus \delta^+(C) = \{(y, t)\}$. However, there is also at least one arc in $\delta^+(C) \setminus \delta^+(C')$, because y is a neighbor of a vertex in C_X and $y \notin C_Y$. Consequently, because all arcs have unit capacity, the value of the s-t cut C' cannot be greater than the value of the s-t cut C. By applying this reasoning iteratively to all vertices $y \in N(C_X) \setminus C_Y$, we obtain, as claimed, that the value of the s-t cut $C \cup N(C_X)$ is no greater than the value of the s-t cut C. We can thus assume that C fulfills (4.1).

The value of the s-t cut C is given by $u(\delta^+(C)) = |\delta^+(C)|$, because H has unit capacities. Consider the arcs $\delta^+(C)$. This set contains no arcs between X and Y, because C satisfies (4.1). Consequently, $\delta^+(C)$ consists of all arcs from s to $X \setminus C_X$ and all arcs from C_Y to t. Thus,

$$\begin{aligned}
u(\delta^+(C)) &= |\delta^+(C)| \\
&= |X| - |C_X| + |C_Y| \\
&\geq |X| - |C_X| + |N(C_X)| && \text{(because } |C_Y| \geq |N(C_X)| \text{ by (4.1))} \\
&\geq |X| \ . && \text{(by Hall's condition } |N(C_X)| \geq |C_X)|\text{)}
\end{aligned}$$

In summary, the minimum s-t cut in H has a value of at least $|X|$, and thus a value of exactly $|X|$, because the s-t cut $C = \{s\}$ has value $|X|$. According to the strong max-flow min-cut theorem (Corollary 4.14), the value of a maximum s-t flow is also $|X|$, and hence, as discussed above, a maximum cardinality matching $M \subseteq E$ has size $|X|$, thus touching all vertices in X. □

4.4.3 Multiple sources and sinks with limited supply and demand

So far we have only considered flow problems with a single source s and a single sink t. We now consider multiple sources and sinks. Here, there are different natural versions one can consider, many of which can easily be reduced to the single-source single-sink case. We consider a classical setting with multiple sources and sinks with limited supply and demand. More precisely, given is a directed graph $G = (V, A)$ with arc capacities $u\colon A \to \mathbb{Z}_{\geq 0}$, sources $s_1, \dots, s_k \in V$, and sinks $t_1, \dots, t_\ell \in V$. Suppose that sources and sinks are all distinct. This assumption is made only to simplify the exposition and can easily be lifted. In addition, each source has limited supply and each sink has a specific demand, captured by a function $h\colon \{s_1, \dots, s_k\} \cup \{t_1, \dots, t_\ell\} \to \mathbb{Z}$. More precisely, positive values of h represent supplies whereas negative values capture demands, i.e., the h-value $h(s_i)$ of a source s_i is positive and represents its supply, whereas $h(t_j)$ for a sink t_j is negative and $-h(t_j)$ represents the demand of t_j. The task is to find a flow f in G that simultaneously

(i) respects the capacity constraints,
(ii) has a net outflow of no more than $h(s_i)$ at source s_i, and
(iii) has a net inflow of precisely $-h(t_j)$ into sink t_j.

In other words, the flow $f\colon A \to \mathbb{R}_{\geq 0}$ shall satisfy the capacity constraints

$$f(a) \leq u(a) \qquad \forall a \in A \ ,$$

and the balance constraints

$$\begin{cases} f(\delta^+(s_i)) - f(\delta^-(s_i)) \leq h(s_i) & \forall i \in [k] \ , \\ \ f(\delta^+(v)) - f(\delta^-(v)) = 0 & \forall v \in V \setminus \{s_1, \dots, s_k, t_1, \dots t_\ell\} \ , \\ f(\delta^+(t_j)) - f(\delta^-(t_j)) = h(t_j) & \forall j \in [\ell] \ . \end{cases}$$

Figure 4.6 shows an example of such a flow problem with four sources and three sinks.

To solve the problem, we construct an auxiliary graph $H = (W, B)$, which is created from G by adding two vertices s and t and the arcs (s, s_i) for all $i \in [k]$ and (t_j, t) for all $j \in [\ell]$, i.e.,

$$W = V \cup \{s, t\} \qquad \text{and} \qquad B = A \cup \{(s, s_i)\colon i \in [k]\} \cup \{(t_j, t)\colon j \in [\ell]\} \ .$$

We define the capacities $u'\colon B \to \mathbb{Z}_{\geq 0}$ in the network H for all $b \in B$ by

$$u'(b) = \begin{cases} u(b) & \text{if } b \in A \ , \\ h(s_i) & \text{if } b = (s, s_i) \text{ for } i \in [k] \ , \\ -h(t_j) & \text{if } b = (t_j, t) \text{ for } j \in [\ell] \ . \end{cases}$$

Figure 4.7 shows the auxiliary graph H and the corresponding capacities u' that we construct to solve the flow problem from Figure 4.6.

We determine a maximum s-t flow f in the network H. The choice of capacities on the arcs (s, s_i) ensures that each source s_i has a net outflow of no more than $h(s_i)$ units over the original arcs. Similarly, the choice of capacities of the arcs (t_j, t) guarantees that each sink t_j has a net inflow of at most $-h(t_j)$ units over the original arcs. Notice that $\nu(f) \leq -\sum_{j=1}^{\ell} h(t_j)$, because the flow value is upper bounded by the value of the s-t cut $W \setminus \{t\}$. Thus, f has value $-\sum_{j=1}^{\ell} h(t_j)$ if and only if each sink t_j has a net inflow of $-h(t_j)$ on the arcs of the original graph. In summary, the original flow problem in G has a solution if and only if the maximum s-t flow in the auxiliary network H reaches the value $-\sum_{j=1}^{\ell} h(t_j)$. In this case we can read off a solution directly from a maximum s-t flow in H.

Figure 4.8 shows an optimal flow in the network from Figure 4.7, from which we can read off an optimal solution for the original problem.

4.4.4 Roster planning

A construction company has a set of workers A, a set of vehicles F grouped into vehicle types, and a set of projects P. The realization of a project requires a vehicle type suitable for the project and a worker allowed to operate the vehicle. The task is to simultaneously handle as many projects as possible.

The following graph $G = (V, E)$ highlights key parts of the available information. It contains one vertex per worker, one per vehicle type, and one per project. Let $A = \{a_1, \dots, a_n\}$ by the set of all workers, let $f_1, \dots, f_\ell$ be the different vehicle types, where $g(f_j) \in \mathbb{Z}_{\geq 0}$ for $j \in [\ell]$ denotes the number of vehicles of type f_j, and let $P = \{p_1, \dots, p_m\}$ be the set of all projects. Hence,

$$V := A \cup \{f_1, \dots, f_\ell\} \cup P \enspace .$$

Moreover, the edge set E contains two types of edges:

(i) There is an edge $\{a_i, f_j\}$ between a worker a_i and vehicle type f_j if a_i can operate vehicles of type f_j.

(ii) There is an edge $\{f_j, p_k\}$ between vehicle type f_j and project p_k if a vehicle of type f_j is suitable for the project p_k.

Figure 4.9 shows a graph with 7 workers, 3 vehicle types, and 8 projects.

The assignment of a worker to a vehicle and a project in the constructed graph G corresponds to a path of length 2 from a worker a_i via a vehicle f_j to a project p_k. Hence, we are interested in the maximum number of paths such that no two paths have a worker or a project in common. Additionally, for each vehicle type f_j, no more than $g(f_j)$ paths should pass through vertex f_j. To solve this problem, we again construct a directed auxiliary network $H = (W, D)$ with capacities $u\colon D \to \mathbb{Z}_{\geq 0}$ in which we will solve a maximum flow problem. We define H by

$$\begin{aligned} W &:= A \cup \{f_j \colon j \in [\ell]\} \cup \{f'_j \colon j \in [\ell]\} \cup P \cup \{s, t\} \enspace , \text{ and} \\ D &:= \{(s, a_i) \colon i \in [n]\} \cup \{(a_i, f_j) \colon \{a_i, f_j\} \in E\} \cup \{(f_j, f'_j) \colon j \in [\ell]\} \\ &\quad\; \cup \{(f'_j, p_k) \colon \{f_j, p_k\} \in E\} \cup \{(p_k, t) \colon k \in [m]\} \enspace . \end{aligned}$$

Furthermore, the capacities $u\colon D \to \mathbb{Z}_{\geq 0}$ are defined as follows: For $d \in D$ we set

$$u(d) = \begin{cases} g(f_j) & \text{if } d = (f_j, f'_j) \text{ for } j \in [\ell] \enspace , \\ 1 & \text{otherwise} \enspace . \end{cases}$$

The directed auxiliary network H resulting from the graph in Figure 4.9 is shown in Figure 4.10.

In the auxiliary graph H, the assignment of a worker a_i to a vehicle f_j and a project p_k corresponds to an s-t path $s \to a_i \to f_j \to f'_j \to p_k \to t$. Each assignment, which can be represented by such a path, is captured in the flow problem by a unit flow along the path. With this interpretation, it is easy to see that each possible roster corresponds to an integral s-t flow in H, whose value corresponds to the number of concurrently processed projects. We can simply add up the flows of the individual assignments. Thereby, no capacity is exceeded, because in a possible roster each worker and each project occur at most once—thus the unit capacities are respected—and each vehicle type f_j is used at most $g(f_j)$ times, that is, the arc (f_j, f'_j) is part of at most $g(f_j)$ single flows and so the total flow over (f_j, f'_j) is at most $g(f_j) = u((f_j, f'_j))$.

Conversely, given an integral s-t flow f in H, we can create a roster in which $\nu(f)$ projects can be completed simultaneously. This is done by splitting the flow into $\nu(f)$ many s-t paths, through each of which flows one unit. We can proceed in the same way as we did to determine a maximum number of arc-disjoint s-t paths in Section 4.4.1: We can iteratively select s-t paths from the arcs with non-negative flow and reduce the flow over the respective arcs by 1 for each selected s-t path.

Hence, there is a bijective assignment between the possible rosters and the integral s-t flows in H, where the value of the flow is equal to the number of simultaneously processed projects. A maximum number of simultaneously processed projects is therefore reached by a maximum integral s-t flow in H. By Theorem 4.16, we can use Ford and Fulkerson's algorithm to determine a maximum integral s-t flow in H. After transforming this flow into a corresponding optimal roster, the rostering problem is solved. Notice that Ford and Fulkerson's algorithm is efficient for this problem, because the maximum flow value is bounded by the number of workers, as this corresponds to the value of the singleton s-t cut $\{s\}$.

4.4.5 Optimal project selection

A company has a set of projects $P = \{p_1, \dots, p_m\}$ to choose from. Each project can either be selected to be performed or discarded. Each of the projects p_i has a profit of $g(p_i)$, where $g\colon P \to \mathbb{Z}$. If $g(p_i) \geq 0$, then p_i is interpreted as a profitable project, whereas $g(p_i) < 0$ indicates that the project is generating a loss. There are precedence constraints between the available projects, i.e., for each project p_i, there is a (possibly empty) set of other projects it depends on, which have to be completed before project p_i can be started. For example, this may model an investment in infrastructure needed for various projects. The question is which projects should be realized to maximize the total profit. Figure 4.11 shows an example of projects with precedence constraints modeled as a directed graph $G = (P, A)$. An arc (p_i, p_j) indicates that project p_i is a prerequisite for project p_j. The project requirements are such that there are no directed cycles in the graph $G = (P, A)$. Such a cycle would correspond to a set of projects that are cyclically dependent on each other and therefore could never be started.

To solve the problem, we construct an auxiliary graph $H = (V, B)$ with capacities $u\colon B \to \mathbb{Z}_{\geq 0}$. The graph H is obtained by starting from G and adding two vertices s, t and several arcs. In the previous examples, the auxiliary graph was built in a way that an integral flow could be interpreted in terms of the decisions that had to be taken. Here, we follow a different approach, in the sense that we build a graph where the minimum s-t cut will correspond to an optimal solution. Hence, we want to interpret certain s-t cuts as ways of how decisions can be taken. Cuts allow us to think about vertices that get selected, which we will interpret as selected projects. In short, we want to set the capacities so that a minimum s-t cut contains exactly the vertex s and an optimal selection of projects from the original graph. To do this we define the auxiliary graph $H = (V, B)$ by setting

$$V := P \cup \{s, t\} \qquad \text{and} \qquad B := A \cup A^R \cup \{(s, p_i)\colon g(p_i) \geq 0\} \cup \{(p_i, t)\colon g(p_i) < 0\} \ ,$$

where $A^R := \{(p_j, p_i)\colon (p_i, p_j) \in A\}$ contains all arcs that are antiparallel to the arcs in A. We also define capacities $u\colon B \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$ as follows: For $b \in B$ we set

$$u(b) := \begin{cases} 0 & \text{if } b \in A \ , \\ \infty & \text{if } b \in A^R \ , \\ g(p_i) & \text{if } b = (s, p_i) \text{ for } p_i \in P \ , \\ -g(p_i) & \text{if } b = (p_i, t) \text{ for } p_i \in P \ . \end{cases}$$

Figure 4.12 shows the graph resulting from the example in Figure 4.11. Notice that we set some capacities to ∞, which can easily be reduced to finite capacities as discussed in Remark 4.6.

The definition of this network is based on the following ideas.

- We first look at the contribution of the arcs of the form (s, p_i) and (p_i, t) for $p_i \in P$. Adding a project p_i to a given s-t cut C with $p_i \notin C$ changes its value by $u((p_i, t))$ if $g(p_i) < 0$ or by $-u((s, p_i))$ if $g(p_i) \geq 0$. By the way how we defined the capacities, the change in terms of value is $-g(p_i)$. Hence, if p_i is a project with non-negative profit $g(p_i) \geq 0$, then the cut value will decrease by $g(p_i)$ units. This is what we want because we are looking for a minimum s-t cut. Otherwise, if p_i incurs costs, then the cut value will increase.
- A minimum s-t cut C corresponds to a set of projects that fulfill the precedence constraints due to the arcs A^R, which have infinite capacities. Indeed, if C contains a project p_j but does not include some other project p_i that is a requirement for p_j, i.e., $(p_i, p_j) \in A$, then $(p_j, p_i) \in \delta^+(C)$. Hence, the value of the s-t cut C is thus $u(\delta^+(C)) \geq u((p_j, p_i)) = \infty$. Because the trivial s-t cut $\{s\}$ has finite capacity, a minimum s-t cut in H has finite capacity and, thus, fulfills all precedence constraints.

We are now formalizing these observations. Let

$$\begin{aligned} P^+ &:= \{p_i \in P\colon g(p_i) \geq 0\} \ , \text{ and} \\ P^- &:= \{p_i \in P\colon g(p_i) < 0\} \end{aligned}$$

be the projects generating profits and losses, respectively. We consider an s-t cut C with finite cut value and first express its value $u(\delta^+(C))$ in terms of project profits and losses. Due to the second of the above observations, $\delta^+(C)$ does not contain arcs with capacity ∞. Furthermore, when calculating the value of C we can neglect the arcs with capacity 0. The only remaining arcs are those of the form (s, p_i) with capacities $g(p_i) \geq 0$ for $p_i \in P^+$ and those of the form (p_i, t) with capacities $-g(p_i) > 0$ for $p_i \in P^-$. We therefore have

$$\begin{aligned} u(\delta^+(C)) &= u(B(\{s\}, P \setminus C)) + u(B(P \cap C, \{t\})) \\ &= g(P^+ \setminus C) - g(P^- \cap C) \\ &= g(P^+) - g(P^+ \cap C) - g(P^- \cap C) \\ &= g(P^+) - g(C \setminus \{s\}) \ , \end{aligned}$$

where $B(V_1, V_2)$, for two vertex sets $V_1, V_2 \subseteq V$, denotes the set of all arcs in B with tail in V_1 and head in V_2. The total profit of the selection $(C \setminus \{s\}) \subseteq P$ of projects is exactly $g(C \setminus \{s\})$. Hence, by minimizing the s-t cut value $u(\delta^+(C))$, we indeed maximize the total profit of the selected projects $C \setminus \{s\}$. Thus, to determine an optimal project selection, we first find a maximum s-t flow in the auxiliary graph H and then construct a minimum s-t cut C from it by Theorem 4.13. The projects $C \setminus \{s\}$ correspond to an optimal project selection. Observe that we do not need to use a maximum s-t flow algorithm that always returns an integral flow, because we are only interested in the minimum s-t cut, which can be derived in linear time from any maximum s-t flow (see Theorem 4.13).

In Figure 4.13, the vertices in a minimum s-t cut are marked in gray. The corresponding optimal selection of projects corresponds to a profit of 4 units for this example.

4.4.6 Open pit mining

In this section, we look at a planning problem inspired by open pit mining. We consider a fixed mining area and assume that, based on soil investigations, it is known—down to a certain depth—what mineral resources can be mined together with the costs incurred by mining certain areas. Figure 4.14 shows a possible soil profile, where the soil is divided into equal parts, which yield the indicated profit during mining. A negative profit indicates that the excavation costs exceed the value of the minerals contained in it. A section can only be mined if the two sections immediately above it have already been excavated. The task is to determine which parts to mine to maximize profit.

This problem can be rephrased as a special case of the optimal project selection problem discussed earlier, namely by considering each mining section as a project with the projects immediately above it as prerequisites. This results in the project dependency graph shown in Figure 4.15. We can thus solve these problems as described in the previous section.

4.4.7 Image segmentation

Image segmentation is a classical image processing problem. It is about splitting an image into different content-related parts, e.g., the separation of foreground and background. The version we consider is a typical problem variation faced by most image processing software. More precisely, our goal is to segment an image based on a rough manual preselection to outline the part of the image that should be separated. Figure 4.16 shows the extraction of the foreground on the example of the Mona Lisa by Leonardo da Vinci.

We discuss an approach for image segmentation that can be modeled and solved as a flow problem. For this we represent each pixel of the image as a vertex of a graph $G = (V, A)$, add two vertices s and t as well as arcs with well-defined capacities, and look for a minimum s-t cut C. Each such cut corresponds to a part of the image. Our goal is to set the arcs and capacities so that a minimum s-t cut contains exactly the desired part of the image. In the selection of the cut C we have to find a compromise between the following two goals, which lead us to the formal definition of our auxiliary graph:

- **Not deviating too much from the manual selection.** The given manual selection is a set $W \subseteq V$ of pixels. In terms of s-t cuts, we represent this selection by the s-t cut $W \cup \{s\}$. Any deviation from $W \cup \{s\}$ should affect the value of the cut negatively, i.e., the s-t cut value should increase. We model this by fixing a penalty $x \in \mathbb{Z}_{\geq 0}$ that has to be paid for every pixel in the returned solution that deviates from $W \cup \{s\}$. More precisely, an s-t cut

$C \subseteq V$, which corresponds to the segmentation given by the pixels $C \setminus \{s\}$, would be penalized by $x \cdot |W \triangle C| = x \cdot |(W \setminus C) \cup (C \setminus W)|$. To achieve this in terms of s-t cut values, we add, for each pixel $p \in W$, an arc (s, p) with capacity $u((s, p)) = x$, and for each pixel $p \in V \setminus W$, we add an arc (p, t) with $u((p, t)) = x$. Figure 4.17 shows this construction schematically. Each vertex in the graph, except for s and t, corresponds to one pixel.

- **Cut along color differences.** The primary goal of image segmentation is to separate different objects, which typically corresponds to areas of different color or color intensity. Hence, adjacent pixels with very similar colors should typically not be separated through the segmentation. Conversely, it is normally desirable to split the image between two adjacent pixels with very different colors. To achieve this behavior for a minimum s-t cut, we introduce directed arcs (p_1, p_2) and (p_2, p_1) for any two adjacent pixels p_1 and p_2. The capacities on these arcs should be large if the colors of the two pixels are very similar. Otherwise, if the colors are very different, the capacities should be small. Hence, this punishes s-t cuts—in the sense that they have a higher value—that separate very similarly colored adjacent pixels and rewards the separation of very differently colored pixels.

  This can for example be implemented by considering the RGB color values, say with 8 bit per color. One option is to set the capacity on an arc (p_1, p_2) depending on the ℓ_1-norm of the difference of the RGB values. Hence, for (r_1, g_1, b_1) and (r_2, g_2, b_2) being the RGB color values of the neighboring pixels p_1 and p_2, the corresponding ℓ_1-norm is

  $$\|(r_1, g_1, b_1) - (r_2, g_2, b_2)\|_1 = |r_1 - r_2| + |g_1 - g_2| + |b_1 - b_2| \ .$$

  This ℓ_1-norm is small if the color values are similar and large otherwise. For the capacity $u((p_1, p_2))$, we want to achieve exactly the opposite behavior. Because each color value is in $\{0, 1, \dots, 255\}$, we have $\|(r_1, g_1, b_1) - (r_2, g_2, b_2)\|_1 \in \{0, 1, \dots, 3 \cdot 255 = 765\}$. We set

  $$u((p_1, p_2)) := 765 - \|(r_1, g_1, b_1) - (r_2, g_2, b_2)\|_1 \ .$$

  The above choice makes sure that the capacities are non-negative (and even integral), which is required in a maximum s-t flow problem, and large capacities correspond to similar colors, as desired. Figure 4.18 illustrates the capacity calculation for arcs between neighboring pixels.

  The choice of our inter-pixel capacities was mostly to easily illustrate this modeling step. In practice, the capacities between pixels are typically not set to the ℓ_1 RGB distance, because equal RGB distances in terms of ℓ_1 norm are often not visually perceived as equal color distances.

After the implementation of these two ideas, a parameter, which we can vary, remains, namely $x \in \mathbb{Z}_{\geq 0}$. The above construction is parameterized by x, which allows for adjusting the weighing between the two penalizations introduced above, i.e., deviation from the manual selection and separation of similarly colored neighboring pixels. Through a suitable choice of x, a compromise between the two goals can be found.

Finally, the image segmentation problem can be approached via a maximum s-t flow algorithm. Again, because we are only interested in a minimum s-t cut, we do not need to find an integral maximum s-t flow.

4.4.8 Theoretical winning possibilities in sports competitions

In the course of a sports championship season it is often discussed which teams can still technically end up as (sole) leaders at the end of the season. We consider handball as an example where 2 points are awarded in every game: In case of a win, the winner gets 2 points and the loser none; in case of a draw, both get one point each. Consider a possible table of the Swiss Handball Championship shortly before the end of the season, see Table 4.1. We are interested in determining whether St. Gallen could still theoretically become sole leader of the table at the end of the season, i.e., whether it can have strictly more points at the end of the season than any other team.

rank	team	remaining games	points
1.	Schaffhausen	3	25
2.	Winterthur	3	23
3.	Thun	4	22
4.	Luzern	3	20
5.	St. Gallen	3	20
⋮	⋮	⋮	⋮

Table 4.1: A possible (partial) handball table.

To simplify the exposition, we restrict ourselves to only the first five teams as listed in Table 4.1 and assume that any other team cannot achieve strictly more than 25 points by the end of the season and, therefore, cannot exceed the currently highest score, which is held by Schaffhausen. Also, for simplicity's sake, assume that the remaining matches of the first five teams are only matches against each other. These restrictions are only assumed to simplify the exposition, and the presented technique could just as well be applied to the full table.

The remaining matches between the considered teams are given by the graph in Figure 4.19a, where an edge between two teams indicates that there is a remaining match between these teams.

To check whether St. Gallen can still become sole leader at the end of the season, we assume that it wins all remaining games. In Figure 4.19b, this is indicated by the orientation of the corresponding three arcs. In our toy example, St. Gallen then reaches 26 points, one more than the current leader Schaffhausen has. Whereas this makes it at least plausible that St. Gallen may still be able to become sole leader, a closer look reveals that this is not possible anymore. Indeed, the three teams Schaffhausen, Winterthur, and Thun together have $25 + 23 + 22 = 70$ points, and the three remaining games between them will increase this count by another 6 points. Hence, the average number of points that these teams will have at the end of the season is $(70 + 6)/3 = 25 + {}^1\!/\!_3 > 25$. This implies that at least one of these three teams will have 26 or more points at the end of the season, thus matching or even surpassing St. Gallen's score. This toy example illustrates that such questions are not always trivial to answer and require a careful consideration of what games still have to be played.

We now move to the general case. Assume we have n handball teams $h_1, \dots, h_n$, where team h_i has p_i points currently. Say that we want to decide whether h_n can become sole leader at the end of the season. As we did in our toy example, we assume h_n to win all remaining games. If r is the number of remaining games of h_n, then team h_n will have $p_n + 2r$ points at the end of the season. Let ℓ be the number of the remaining games $g_1, \dots, g_\ell$ not involving h_n. It remains to check whether there exist outcomes of these games such that none of the teams $h_1, \dots, h_{n-1}$ exceeds $p_n + 2r - 1$ points at the end of the season. We capture this question in terms of a flow problem on a graph $G = (V, A)$ with capacities $u \colon A \to \mathbb{Z}_{\geq 0}$ as follows:

- G has a vertex for each game $g_1, \dots, g_\ell$, a vertex for each team $h_1, \dots, h_{n-1}$, and two further vertices s and t.
- The arcs A and their capacities u are defined as follows:
  - For each game g_i, we add an arc (s, g_i) with capacity $u((s, g_i)) = 2$.
  - For each game g_i, say betweens teams h_{j_1} and h_{j_2}, we add arcs (g_i, h_{j_1}) and (g_i, h_{j_2}) with capacities $u((g_i, h_{j_1})) = u((g_i, h_{j_2})) = 2$.
  - For each team h_i with $i \in [n-1]$, we add an arc (h_i, t) with capacity $u((h_i, t)) = p_n + 2r - p_i - 1$. Here we assume without loss of generality that no team h_i for $i \in [n-1]$ has strictly more than $p_n + 2r - 1$ points; for otherwise, we can immediately observe that team h_n cannot become sole leader anymore because it cannot surpass the current score p_i of team h_i.

Figure 4.20 depicts the graph $G = (V, A)$ with capacities u for the toy example shown in Figure 4.19b.

In this network, s models the source of the game points still to be distributed. The capacities of 2 on arcs leaving s model that, for each remaining game, 2 points have to be distributed. These points can be split in any integral way among the two involved teams. This is captured by the arcs connecting games to teams. Finally, the flow on the arcs from teams to the sink model how many additional points each team will get out of the remaining games. The capacities on these arcs is set such that no team can get enough points to match or even exceed the $p_n + 2r$ points that team n will have at the end of the season. In terms of s-t flows, the problem of distributing the remaining points is the following. We need to find an integral s-t flow that saturates all arcs leaving s, because we need to assign 2 points per remaining game. This is equivalent to finding an integral s-t flow f of value $\nu(f) = 2\ell$, i.e., one that saturates all arcs leaving s. If we only want to decide whether a corresponding point assignment exists, without actually having to find the assignment, then the question reduces to checking whether an integral s-t flow of value 2ℓ exists. (Note that because 2ℓ is the sum of the capacities of all arcs leaving s, there is no flow of value strictly larger than 2ℓ.) Because, by Theorem 4.16, there always exists an s-t flow of maximum value that is integral, it suffices to check whether the maximum s-t flow in G, without any integrality requirement, has value at least 2ℓ. Hence, to perform this check, we do not need to employ a maximum s-t flow algorithm that returns an integral flow. If the maximum s-t flow value is 2ℓ, then team h_n can still become sole leader by the end of the season, otherwise this is not possible anymore.

4.5 Polynomial-time variations and extensions of Ford and Fulkerson's algorithm

We come back to the fact that the maximum s-t flow algorithm of Ford and Fulkerson is not guaranteed to run in polynomial time. We recall the running time bound of $O(\alpha \cdot (m+n))$, where α is the value of a maximum s-t flow and $m := |E|$, $n := |V|$ (see Corollary 4.12). As discussed, this is not polynomial time because α may be exponential in the input size. In this section, we discuss approaches inspired by Ford and Fulkerson's algorithm that find maximum s-t flows efficiently. These are procedures that, like Ford and Fulkerson's algorithm, augment flow successively until they obtain a maximum s-t flow. We highlight that there are other classes of efficient flow algorithms, most notably so-called *preflow push* algorithms.

For simplicity, we assume that $n = O(m)$. This is not really restrictive, because whenever $m < n-1$, then the graph is not even connected. In such a case we could first perform a BFS to check whether s and t are in the same connected component and then only work on that connected component. This leads to a graph fulfilling $n = O(m)$ because the graph is connected.

Hence, under this assumption, the running time of Ford and Fulkerson's procedure is $O(\alpha \cdot m)$, or, to avoid the dependence on the unknown quantity α, we can also bound it by $O(U \cdot m)$, where we use $U := u(A)$ as a shorthand for the sum of all capacities.

4.5.1 Capacity scaling algorithm

The capacity scaling algorithm is one of the simplest polynomial-time procedures for the maximum flow problem. It addresses the key issue of the running time bound of Ford and Fulkerson, namely the dependence on U. Scaling is an important technique with numerous applications in algorithmics. The capacity scaling algorithm for flows is a nice an instructive example of a scaling algorithm. In this context, the idea is to make augmentations along paths with large augmentation volume if possible. For this, we will consider subgraphs of the residual graph that only contain large capacities. To this end, we introduce the following notation $G_{f,\Delta}$.

Definition 4.30: $G_{f,\Delta}$

Let f be an s-t flow in the directed graph $G = (V, A)$ with capacities $u \colon A \to \mathbb{Z}_{\geq 0}$ and let $\Delta \in \mathbb{R}_{\geq 0}$. We denote by $G_{f,\Delta}$ the subgraph of the residual graph $G_f = (V, B)$ containing only the arcs with residual capacity of at least Δ.

Algorithm 6 describes the capacity scaling algorithm for maximum s-t flows. It runs in phases, defined by the outer while-loop. In each phase, it only considers augmenting paths with an augmentation volume of at least some value Δ; this is equivalent to looking for augmenting paths in $G_{f,\Delta}$. Once no more augmentations can be found during a phase, the parameter Δ is halved and the next phase begins.

For the running time analysis of Algorithm 6, we have to be specific on how precisely we find an f-augmenting path in $G_{f,\Delta}$. An easy way to perform this step is as follows. Instead of explicitly constructing $G_{f,\Delta}$, we construct G_f as in Ford and Fulkerson's algorithm. However,

Algorithm 6: Capacity scaling algorithm for maximum s-t flows

```
Input  : Directed graph $G = (V, A)$ with arc capacities $u\colon A \to \mathbb{Z}_{\geq 0}$ and $s, t \in V$, $s \neq t$.
Output: A maximum $s$-$t$ flow $f$.
$f(a) = 0\ \forall a \in A$.                    // We start with the zero flow.
$\Delta = 2^{\lfloor \log_2(U) \rfloor}$.
while $\Delta \geq 1$ do              // These iterations are called phases.
    while $\exists f$-augmenting path $P$ in $G_{f,\Delta}$ do
        Augment $f$ along $P$ and set $f$ to the augmented flow.
    $\Delta = \frac{\Delta}{2}$.
return $f$
```

to find an augmenting path in $G_{f,\Delta}$, we then perform BFS on G_f where we slightly modify BFS to ignore arcs with a capacity strictly below Δ. This does not affect the running time of the BFS procedure.

The correctness of the algorithm follows easily from arguments identical to the ones we used to show that Ford and Fulkerson's algorithm returns a maximum s-t flow.

Theorem 4.31

Algorithm 6 returns a maximum s-t flow.

Proof. Because the starting value of Δ is a power of two, Δ will always be integral. Consequently, the algorithm will finish after finitely many augmentations because each augmentation increases the flow value of f by at least one unit. Moreover, for the phase $\Delta = 1$, we have that $G_{f,\Delta} = G_{f,1}$ is identical to G_f, except that arcs with 0 capacity have been removed. Thus, a path is f-augmenting in G_f if and only if it is f-augmenting in $G_{f,1}$. Because the $\Delta = 1$ phase finishes with a residual graph $G_{f,1}$ without f-augmenting paths from s to t, we obtain an s-t flow f such that there is no f-augmenting path in G_f. Hence, by Theorem 4.13, the flow f is a maximum s-t flow. □

Notice that the maximum flow returned by the capacity scaling algorithm will be integral because all augmentations have an integral augmentation volume, as in the case of Ford and Fulkerson's algorithm.

We now show the crucial gain of capacity scaling in this context, namely that the running time dependence on U is only logarithmic.

Theorem 4.32

Algorithm 6 runs in $O(m^2 \log U)$ time.

Proof. The number of phases is bounded by $O(\log U)$. Hence, it suffices to show that each phase takes $O(m^2)$ time. To this end, consider a phase, which is defined by some value of Δ. Let f be the current flow at the start of the phase. Let $C \subseteq V$ be all vertices that can be reached from s in $G_{f,2\Delta}$. We claim that $t \notin C$. Indeed, if the phase we consider is the first one, i.e.,

$\Delta = 2^{\lfloor \log U \rfloor} > U/2$, then $2\Delta > U$; however, there is no arc with capacity strictly larger than U, implying that an augmentation volume strictly larger than U is not achievable. Otherwise, if the considered phase is not the first one, then there is no augmenting path in $G_{f,2\Delta}$ due to the termination criterion of the previous phase. Hence, the set $C \subseteq V$ of all vertices reachable from s in $G_{f,2\Delta}$ is an s-t cut, which, by definition, fulfills that for each arc $a \in \delta^+_{G_f}(C)$, we have $u_f(a) < 2\Delta$. Hence, the value of the s-t cut C in G_f is upper bounded by

$$u_f(\delta^+_{G_f}(C)) \leq 2m \cdot 2\Delta \ . \tag{4.2}$$

A crucial observation is that $u_f(\delta^+_{G_f}(C))$ is an upper bound on how much the value of f can still be augmented, i.e., the difference between the value of a maximum flow and $\nu(f)$. To see this, we recall that for each $a \in A$ we have $u_f(a) = u(a) - f(a)$, and for a reverse arc a^R of $a \in A$ we have $u_f(a^R) = f(a)$. Hence,

$$u_f(\delta^+_{G_f}(C)) = u(\delta^+(C)) - f(\delta^+(C)) + f(\delta^-(C)) = u(\delta^+(C)) - \nu(f) \ ,$$

implying, as claimed, that future augmentations will increase the value of the flow by at most $u_f(\delta^+_{G_f}(C))$, because $u(\delta^+(C))$ is an upper bound on the maximum s-t flow value by the weak max-flow min-cut theorem. Because each augmentation in the current iteration will have an augmentation volume of at least Δ, this implies, together with (4.2), that we can have at most

$$\frac{u_f(\delta^+_{G_f}(C))}{\Delta} \leq 4m$$

augmentations in the current phase, as desired. The result follows by observing that each augmentation takes $O(m)$ time. □

Notice that the proven running time bound of $O(m^2 \log U)$ is indeed polynomially bounded in the input size. For this, observe that to save the arc capacities in binary, the number of bits needed is

$$\Theta\left(m + \sum_{a\in A} \log(u(a)+1)\right) = \Theta\left(m + \log\left(\prod_{a\in A}(u(a)+1)\right)\right) = \Omega\left(m + \log U\right) \ .$$

Hence, the input size is at least $\Omega(m + \log U)$, and the capacity scaling algorithm is thus an efficient method. However, note that it is not strongly polynomial, because the number of elementary operations depends on numbers provided in the input. We next show a method to find a maximum s-t flow in strongly polynomial time.

4.5.2 Edmonds-Karp algorithm

The Edmonds-Karp algorithm is a particular, strongly polynomial, way to realize Ford and Fulkerson's algorithm. More precisely, the Edmonds-Karp algorithm will always augment on shortest augmenting paths, i.e., augmenting paths with a minimum number of arcs. Algorithm 7 describes the procedure.

Figure 4.21 shows an example run of the Edmonds-Karp algorithm.

Algorithm 7: Edmonds-Karp algorithm

```
Input  : Directed graph $G = (V, A)$ with arc capacities $u\colon A \to \mathbb{Z}_{\geq 0}$ and $s, t \in V$, $s \neq t$.
Output: A maximum $s$-$t$ flow $f$.
$f(a) = 0 \;\; \forall a \in A$.
while $\exists f$-augmenting path in $G_f$ do
    Find an $f$-augmenting path $P$ in $G_f$ minimizing $|P|$.
    Augment $f$ along $P$ and set $f$ to augmented flow.
return $f$
```

As mentioned, the Edmonds-Karp algorithm is a version of Ford and Fulkerson's algorithm with a more specific rule of how to choose augmenting paths. Thus, the correctness proof of Ford and Fulkerson's method also applies to Algorithm 7. It remains to discuss the running time. Notice that finding a shortest augmenting path does not take longer than computing any augmenting path if we use BFS, because BFS automatically identifies a shortest path. Hence, also the running time bound we derived for Ford and Fulkerson's algorithm applies to Algorithm 7. We show next that the running time of Algorithm 7 is even strongly polynomial.

We want to measure the progress of Algorithm 7 in terms of the lengths of the augmenting paths found. Intuitively, we expect that augmenting paths used by the algorithm are short at the beginning and get longer later on. The following lemma implies that this is indeed the case.

Lemma 4.33

Let $G = (V, A)$ be a directed graph with arc capacities $u\colon A \to \mathbb{Z}_{\geq 0}$, and let $s, t \in V$ with $s \neq t$. Moreover, let f_1 be an s-t flow in G, and let f_2 be an s-t flow obtained by augmenting f_1 along a shortest augmenting path P in G_{f_1}. Then,

$$\begin{aligned} d_{f_1}(s,v) &\leq d_{f_2}(s,v) \quad \forall v \in V \;\;, \text{ and} \\ d_{f_1}(v,t) &\leq d_{f_2}(v,t) \quad \forall v \in V \;\;, \end{aligned}$$

where $d_f(v, w)$ denotes, for $v, w \in V$ and an s-t flow f, the length (in terms of number of arcs) of a shortest v-w path in G_f that only uses arcs with strictly positive f-residual capacity.

Proof. We only prove the first part of the statement, i.e., $d_{f_1}(s, v) \leq d_{f_2}(s, v) \; \forall v \in V$. The second part can be reduced to the first one by reversing all arc directions and flows on the arcs, and applying the first part of the lemma.

Notice that G_{f_1} and G_{f_2} have the same vertices and arcs, but they have different residual capacities u_{f_1} and u_{f_2}. Indeed the vertex set is V and the arc set is $B = A \cup A^R$, i.e., it contains the original arcs A and a reverse arc a^R for each original arc $a \in A$. We define

$$B_i \coloneqq \{b \in B \colon u_{f_i}(b) > 0\} \qquad \forall i \in [2] \;\; .$$

Assume with the goal of obtaining a contradiction that there is a vertex $v \in V$ such that

$$d_{f_1}(s,v) > d_{f_2}(s,v) \;\; . \tag{4.3}$$

Among all such vertices, we choose one where $d_{f_2}(s,v)$ is smallest. Let P_2 be a shortest s-v path in (V,B_2) and let $(w,v)\in P_2$ be the last arc in P_2. (Notice that P_2 cannot be an empty path because $s\neq v$ due to (4.3).) By our choice of v we have

$$d_{f_1}(s,w)\leq d_{f_2}(s,w) \ . \tag{4.4}$$

In particular, this implies that $(w,v)\in B_2\setminus B_1$. For otherwise, we could extend a shortest s-w path in (V,B_1) with the arc (w,v), leading to an s-v walk in (V,B_1) of length $d_{f_1}(s,w)+1$. Together with (4.4) this would lead to

$$d_{f_1}(s,v)\leq d_{f_1}(s,w)+1\leq d_{f_2}(s,w)+1=d_{f_2}(s,v) \ ,$$

thus contradicting (4.3). Observe that $(w,v)\in B_2\setminus B_1$ implies $(v,w)\in P$, because only reverse arcs of arcs in P have higher u_{f_2}-capacity than u_{f_1}-capacity, which is necessary for an arc to be in $B_2\setminus B_1$. However, $(v,w)\in P$ is impossible because it implies $d_{f_1}(s,v)=d_{f_1}(s,w)-1$, which leads to the following contradiction:

$$d_{f_1}(s,v)<d_{f_1}(s,w)\leq d_{f_2}(s,w)=d_{f_2}(s,v)-1<d_{f_1}(s,v)-1 \ ,$$

where the second inequality is due to (4.4), the equality follows from the fact that $(w,v)\in P_2$, and the last inequality is due to (4.3). □

Theorem 4.34

Algorithm 7 runs in $O(nm^2)$ time.

Proof. We divide the different augmentation steps of Algorithm 7 into phases, where a phase corresponds to augmentations with augmenting paths of the same length. Because in a graph with n vertices the longest possible s-t path has length $n-1$ and the shortest one has length 1, there are at most $n-1=O(n)$ phases. Moreover, notice that Lemma 4.33 implies that the augmenting paths found in Algorithm 7 are non-decreasing in lengths during the course of the algorithm. Hence, a phase contains augmentations done in consecutive iterations of the while-loop of Algorithm 7. We finish the proof by showing that each phase performs at most $O(m)$ augmentations, which implies the result because each augmentation takes $O(m)$ time.

To this end, consider a single phase, say the one where all augmenting paths have length $k\in[n-1]$. We call this *phase* k. For any s-t flow f and vertices $v,w\in V$, we use the notation

$$d_f(v,w)=\min\{|P|\colon P\subseteq B \text{ is } v\text{-}w \text{ path with } u_f(b)>0\ \forall b\in P\}$$

for the v-w distance (in terms of number of arcs) in $G_f=(V,B)$ after deleting arcs with zero f-residual capacity.

We first show the following claim.

Claim: If an arc is used in an augmenting path in phase k, then its reverse arc will never be used in an augmenting path in phase k.

To see this, assume that $(v,w) \in B$ is an arc on an augmenting path in phase k, and let f be the s-t flow right before we perform this augmentation. Because we are in phase k, the corresponding augmenting path has length k, i.e.,

$$d_f(s,v) + 1 + d_f(w,t) = k \ . \tag{4.5}$$

Moreover, because subpaths of a shortest path are shortest paths, we have

$$\begin{aligned} d_f(s,w) &= d_f(s,v) + 1 \ , \text{ and} \\ d_f(v,t) &= 1 + d_f(w,t) \ . \end{aligned} \tag{4.6}$$

For the sake of obtaining a contradiction, assume that a later augmenting path P in phase k uses the arc (w,v). Hence, P consists of:

(i) An s-w path, which has length at least $d_f(s,w)$ by Lemma 4.33,
(ii) the arc (w,v), and
(iii) a v-t path, which has length at least $d_f(v,t)$ by Lemma 4.33.

Thus, P has length

$$|P| \geq d_f(s,w) + 1 + d_f(v,t) = d_f(s,v) + d_f(w,t) + 3 = k + 2 \ ,$$

where the first equality follows from (4.6) and the second one from (4.5). However, this contradicts the assumption that P is an augmenting path in phase k, which would require $|P| = k$, and thus shows the claim.

The claim implies that whenever an arc $b \in B$ gets saturated during an augmentation in phase k, then neither the reverse arc b^R of b nor the arc b itself will be used again during the same phase. (If $b \in B$ is already a reverse arc of an arc $a \in A$, i.e., $b = a^R$, then we denote by b^R the arc a.) Indeed, b^R cannot be used due to the claim, and b cannot be used anymore because its residual capacity is zero and will not increase during the same phase again because no flow will be sent over b^R. Hence, at most m arcs can get saturated in augmentations during phase k. However, each augmentation does saturate at least one arc. Hence, there are at most $O(m)$ augmentations in each phase, as desired. □

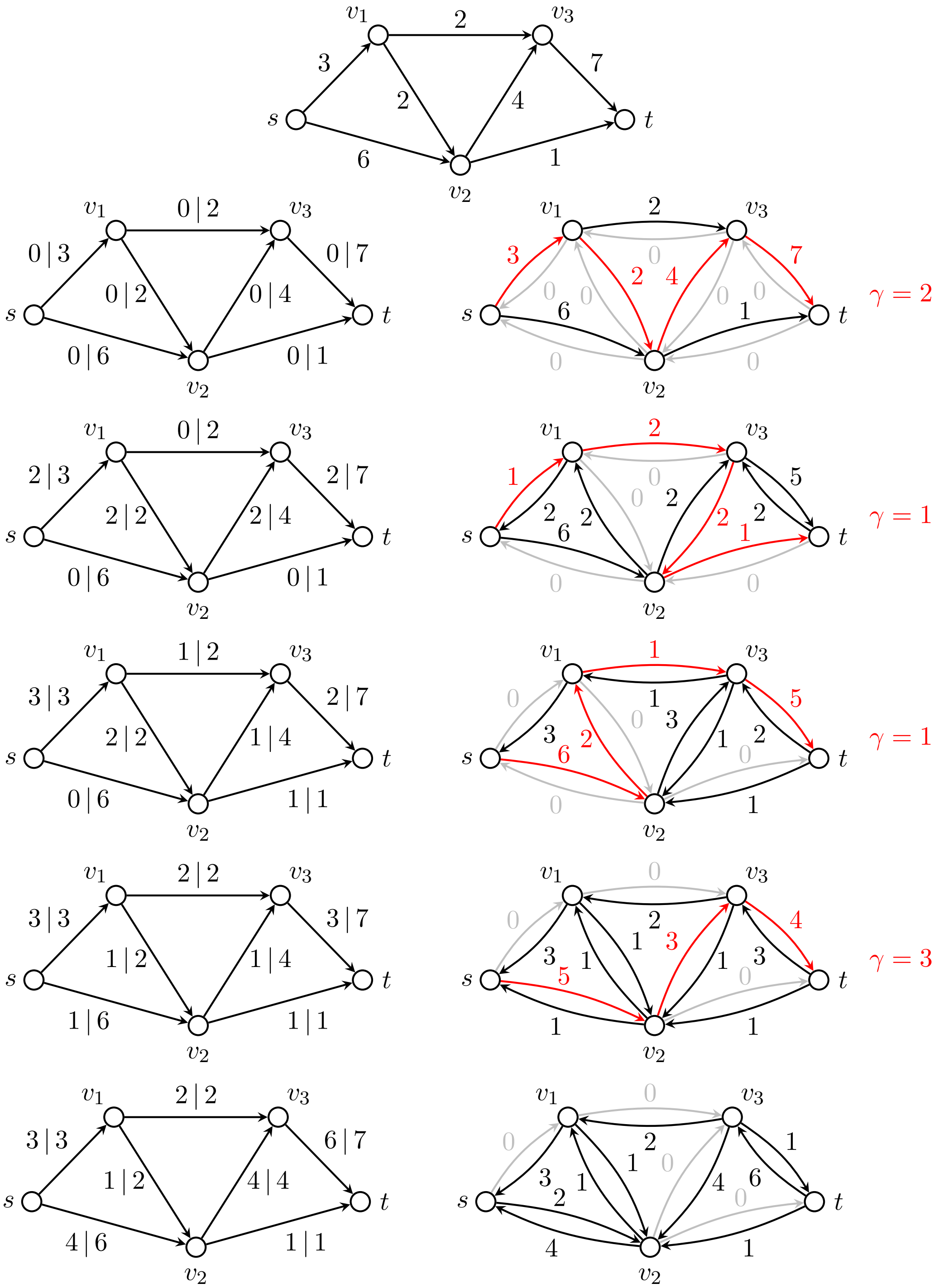


Figure 4.3: Example run of Ford and Fulkerson's algorithm. On top is the starting graph, left-hand side pictures show the current flow and right-hand side pictures the residual network with an augmenting path in red, if there is one.

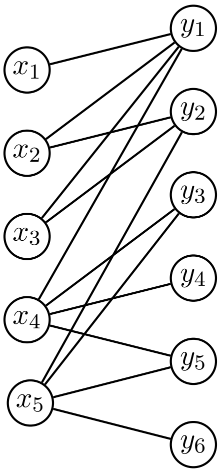


Figure 4.4: Example of a bipartite graph $G = (V, E)$ with bipartition $V = X \dot{\cup} Y$, where $X = \{x_1, \ldots, x_5\}$ and $Y = \{y_1, \ldots, y_6\}$.

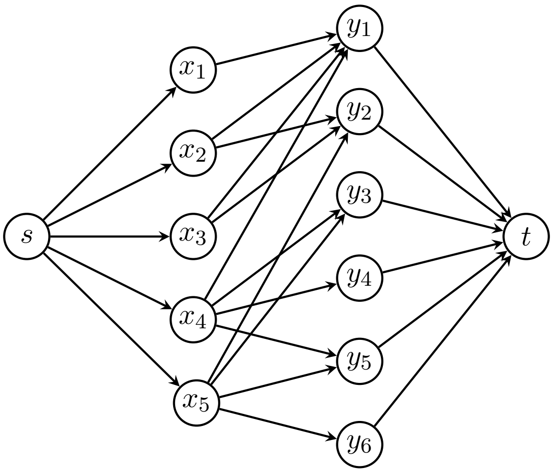


Figure 4.5: Example of the directed graph $H = (W, F)$ obtained from the bipartite graph $G = (V, E)$ shown in Figure 4.4.

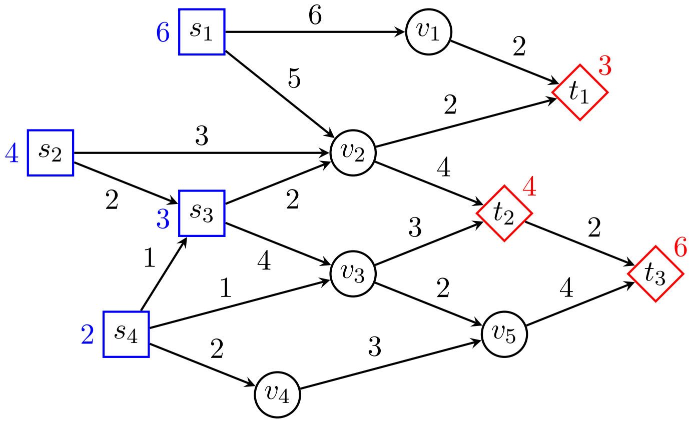


Figure 4.6: A flow problem with four sources $\{s_1, s_2, s_3, s_4\}$ and three sinks $\{t_1, t_2, t_3\}$. The supply of each source is indicated in blue, the demand of the sinks in red. The numbers on the arcs correspond to the respective capacities.

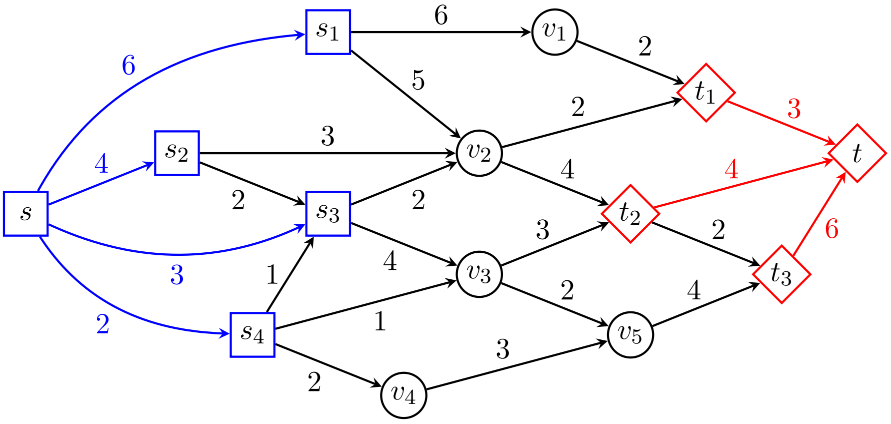


Figure 4.7: The auxiliary graph H with capacities u' used to solve the flow problem from Figure 4.6.

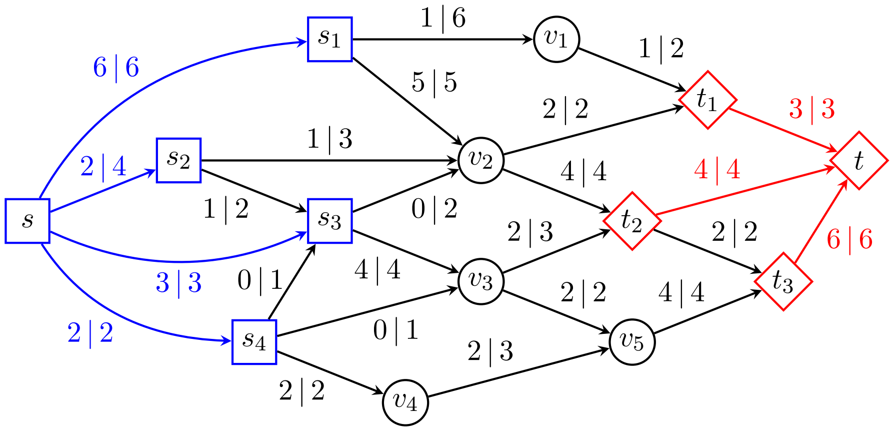


Figure 4.8: An optimal flow in the auxiliary graph H from Figure 4.7, given in the form $f(b) \mid u'(b)$ for all $b \in B$.

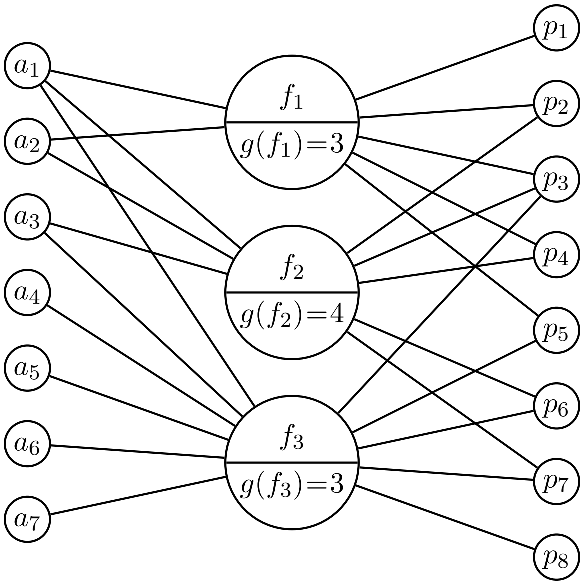


Figure 4.9: The graph $G = (V, E)$ representing a roster planning problem with 7 workers, 3 vehicle types, and 8 projects. The number of available vehicles of each type is specified in the corresponding vertex.

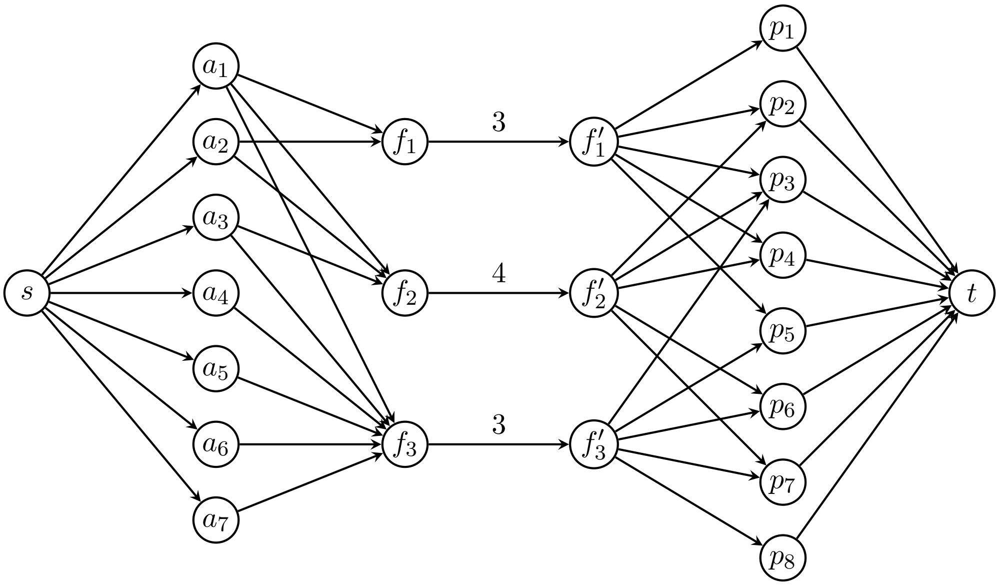


Figure 4.10: The auxiliary network H for the graph G from Figure 4.9. Arcs without explicitly specified capacity have capacity 1.

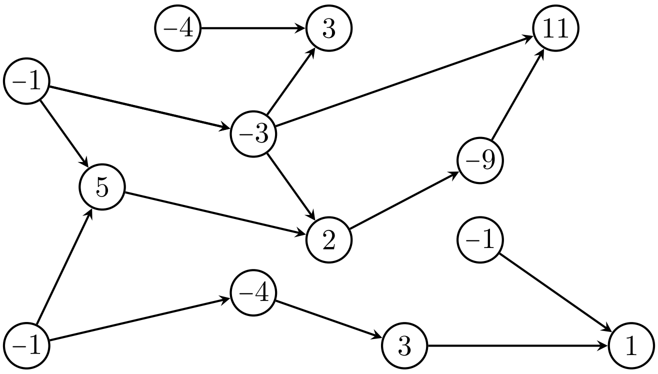


Figure 4.11: A graph G with projects and their precedence constraints. Profits (or costs) are indicated in the corresponding vertices.

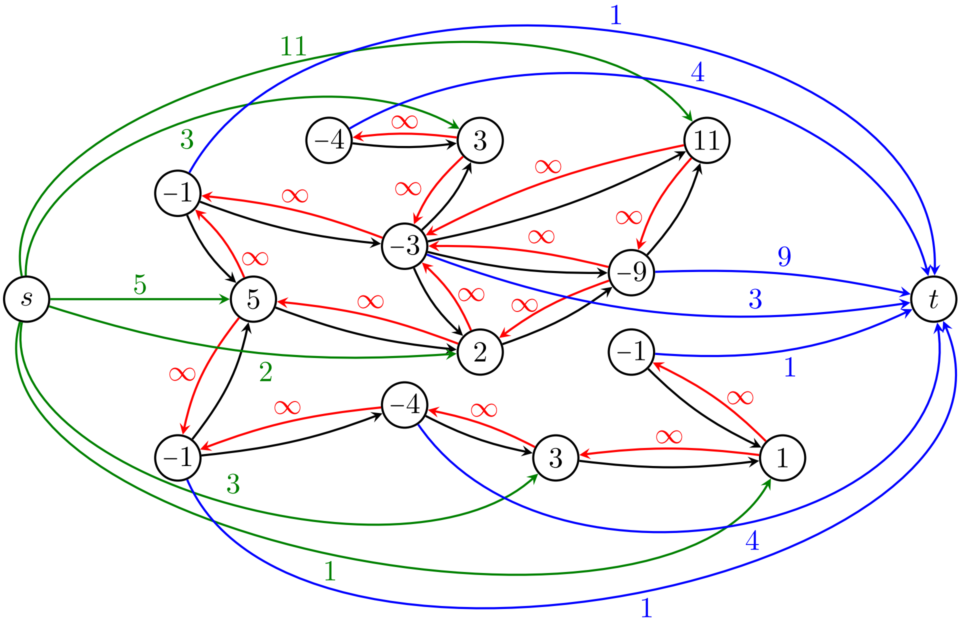


Figure 4.12: The auxiliary graph for the example in Figure 4.11. Arcs with no explicitly specified capacity have capacity 0.

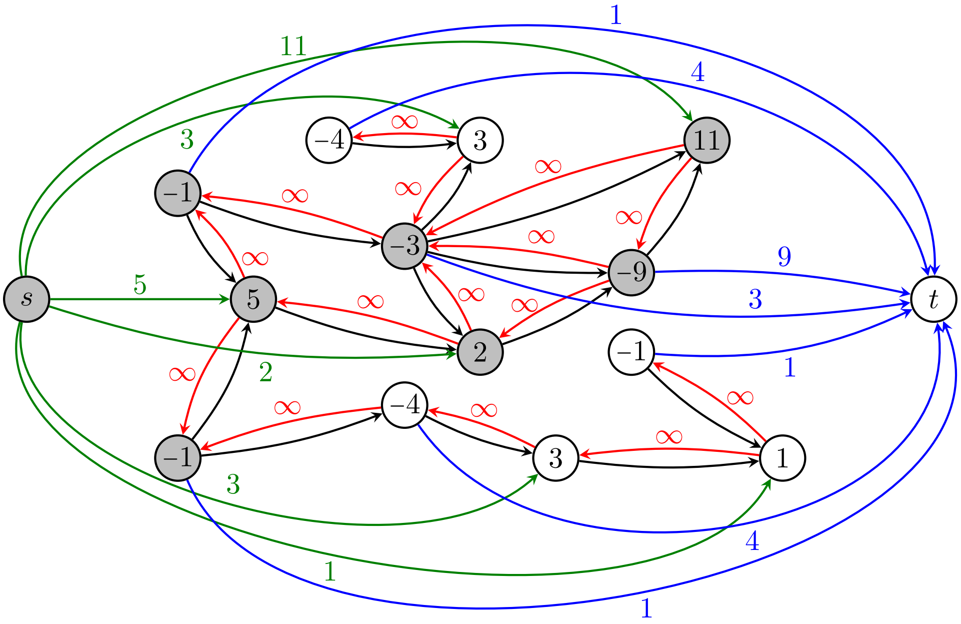


Figure 4.13: An optimal solution to the shown project selection problem. The vertices in a minimum s-t cut C are marked in gray, and the optimal set of projects to select is $C \setminus \{s\}$.

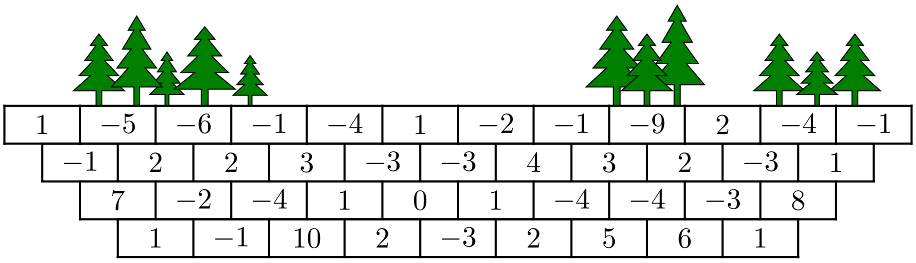


Figure 4.14: A possible soil profile with respective profits.

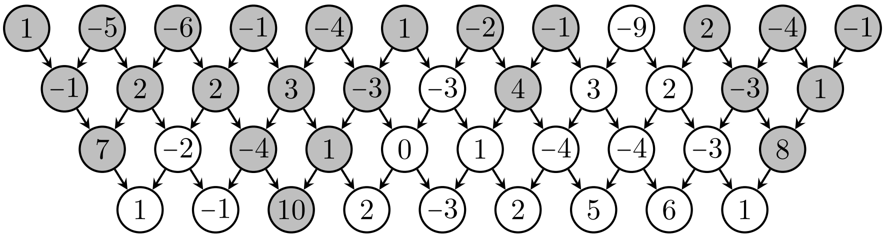
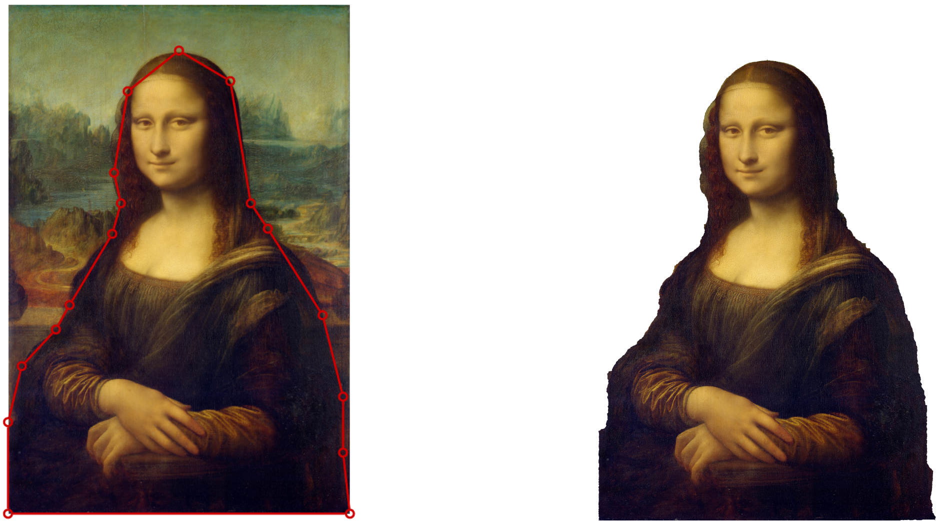


Figure 4.15: Reduction of the open pit mining problem shown in Figure 4.14 to an optimal project selection problem. The gray vertices correspond to an optimal solution.



(a) The Mona Lisa of Leonardo da Vinci together with a manual selection.

(b) The foreground of the Mona Lisa, extracted due to color differences and manual selection.

Figure 4.16: Extraction of the foreground from an image.

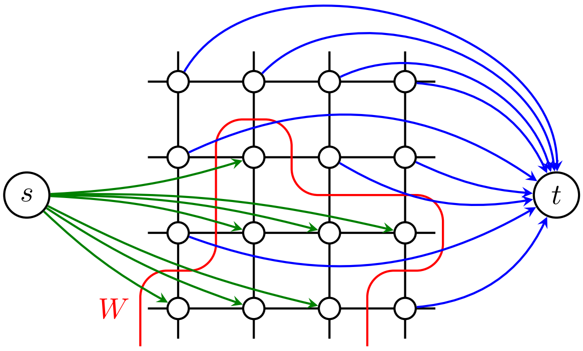
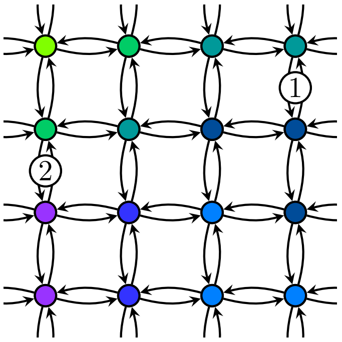


Figure 4.17: Excerpt of an image with manual segmentation W shown in red. The arcs (s, p) are shown in green and arcs (p, t) are highlighted in blue. Each of these colored arcs has equal capacity $x \in \mathbb{Z}_{\geq 0}$.



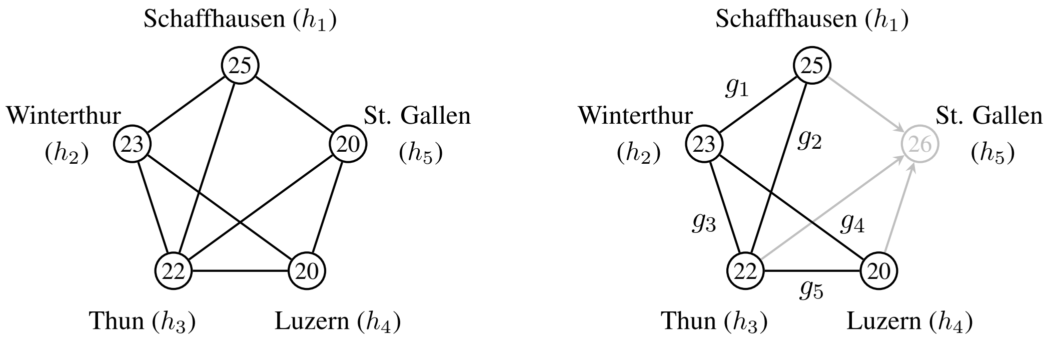
① 🟦 $= (0, 153, 153)$ 🟦 $= (0, 76, 153)$

$$\begin{aligned} u((\bullet, \bullet)) &= 765 - |0 - 0| - |153 - 76| - |153 - 153| \\ &= 765 - 0 - 77 - 0 = 688 \ . \end{aligned}$$

② 🟩 $= (0, 204, 102)$ 🟪 $= (153, 51, 255)$

$$\begin{aligned} u((\bullet, \bullet)) &= 765 - |0 - 135| - |204 - 51| - |102 - 255| \\ &= 765 - 135 - 153 - 153 = 324 \ . \end{aligned}$$

Figure 4.18: Calculation of color difference of adjacent pixels using two examples. The greater the color difference, the smaller the capacity u on the corresponding arcs.



(a) Remaining matches between teams: Each edge corresponds to one match.

(b) If we assume that St. Gallen wins all its games, then the 5 games $g_1, \ldots, g_5$ remain.

Figure 4.19: Remaining games displayed as a graph.

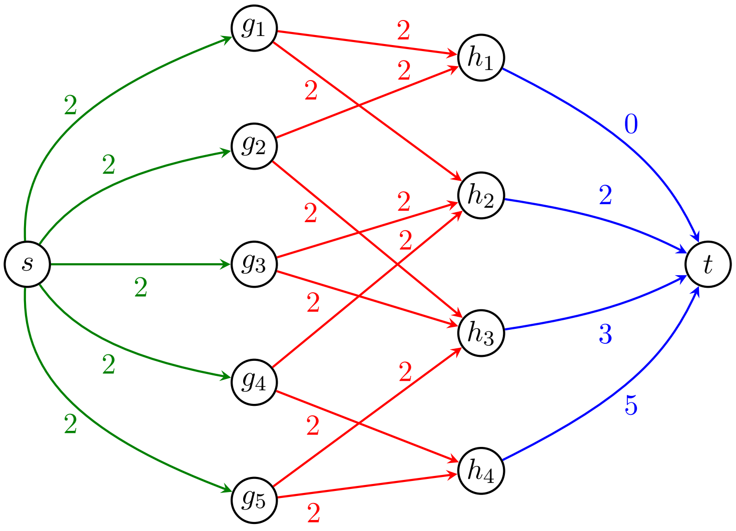


Figure 4.20: The auxiliary s-t flow network $G = (V, A)$ with corresponding capacities for the example shown in Figure 4.19b.

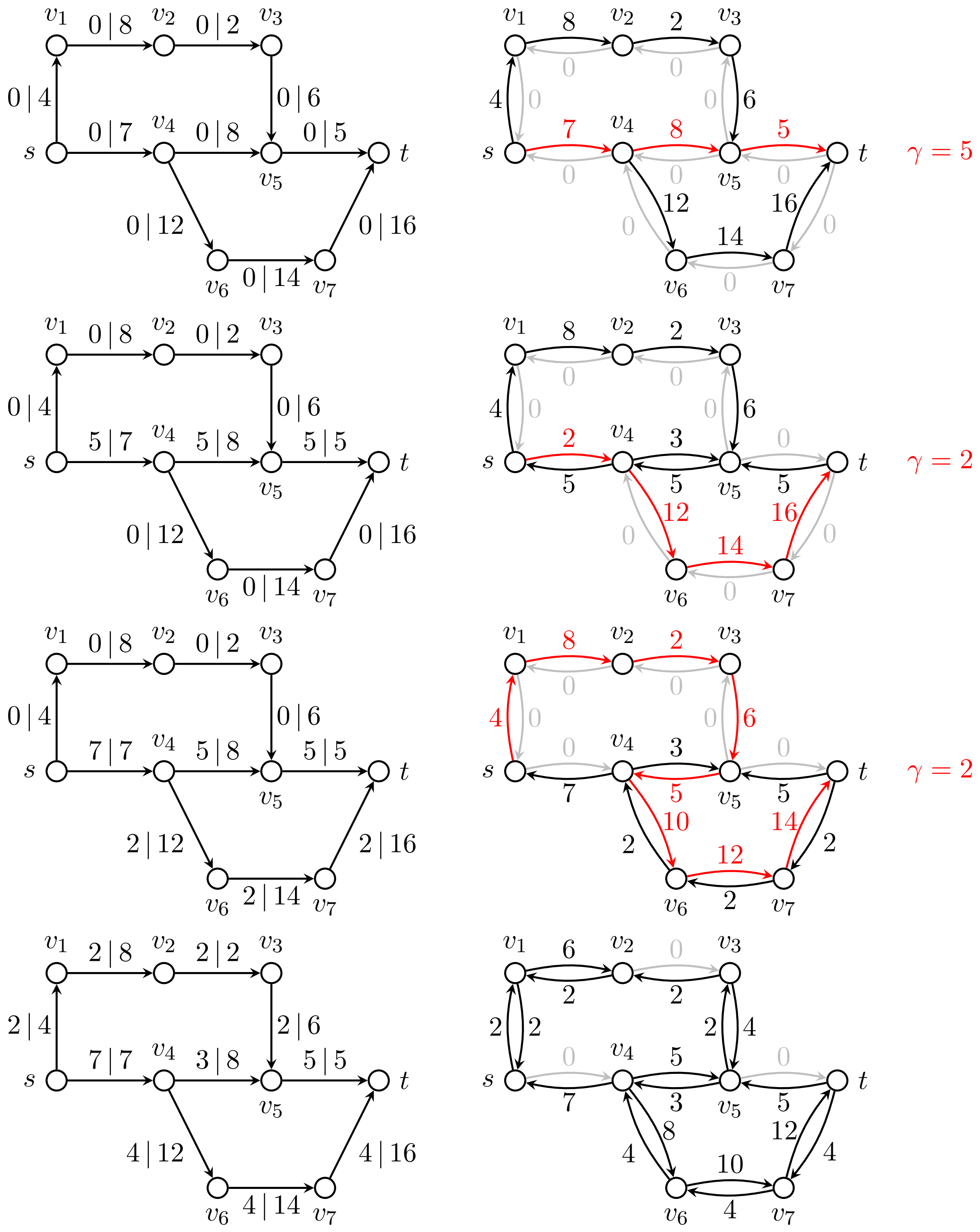


Figure 4.21: Example run of the Edmonds-Karp algorithm to find a maximum s-t flow. Each row of two figures corresponds to one iteration of the algorithm. The left-hand side graphs show the current flow values, and the right-hand side graphs are the corresponding residual graphs with a shortest augmenting path highlighted in red. The γ-values to the far right are the augmentation volumes for the highlighted augmenting paths.

5 Polyhedral Approaches in Combinatorial Optimization

Combinatorial optimization problems can often be described by:

- A finite set N, called *ground set*,
- a family $\mathcal{F} \subseteq 2^N$ of *feasible sets*, also called *solutions*, and
- an *objective function* $w\colon N \to \mathbb{Z}$ to maximize/minimize.

The corresponding combinatorial optimization problem is then given by:

$$\boxed{\begin{aligned} \max / \min \quad w(F) &:= \sum_{e \in F} w(e) \\ F &\in \mathcal{F} \end{aligned}} \tag{5.1}$$

Some examples:

(i) **Maximum weight matchings**. Given is an undirected graph $G = (V, E)$ with non-negative edge weights $w\colon E \to \mathbb{Z}_{\geq 0}$.

- Ground set: $N = E$.
- Feasible sets: $\mathcal{F}$ is the family of all loopless sets $M \subseteq E$ such that no two edges of M have a common endpoint. Such a set M is called a *matching*.
- Objective: maximize w.

Two important special cases of the problem:

- *Maximum cardinality matchings*: w is given by $w(U) = |U|$ for $U \subseteq E$, i.e., each edge has weight equal to one.
- *Maximum weight/cardinality bipartite matchings*: $G = (V, E)$ is a *bipartite graph*, i.e., there is a bipartition $V = X \dot{\cup} Y$ such that each $e \in E$ has one endpoint in X and one in Y.

(ii) **Shortest s-t path problem**. Given is an undirected or directed graph $G = (V, E)$, two vertices $s, t \in V$, and a length function $\ell\colon E \to \mathbb{Z}_{\geq 0}$.

- $N = E$,
- $\mathcal{F}$ are all subsets of edges corresponding to s-t paths.
- Objective: minimize $w = \ell$.

(iii) **Minimum weight spanning tree**. Given is an undirected graph $G = (V, E)$ and non-negative edge weights $w\colon E \to \mathbb{Z}_{\geq 0}$.

- $N = E$,
- $\mathcal{F}$ are all sets $F \subseteq E$ that correspond to spanning trees.
- Objective: minimize w.

5.1 Polyhedral descriptions of combinatorial optimization problems

Consider the family $\mathcal{F} \subseteq 2^N$ of feasible sets of a combinatorial optimization problem (5.1). We can describe each solution $F \in \mathcal{F}$ by its *incidence/characteristic vector* $\chi^F \in \{0,1\}^N$, where

$$\chi^F(e) = \begin{cases} 1 & \text{if } e \in F \ , \\ 0 & \text{if } e \in N \setminus F \ . \end{cases}$$

The polytope that *corresponds* to $\mathcal{F}$ is the polytope $P_{\mathcal{F}} \subseteq [0,1]^N$ whose vertices are precisely incidence vectors of solutions, i.e.,

$$P_{\mathcal{F}} = \operatorname{conv}(\{\chi^F \colon F \in \mathcal{F}\}) \ , \tag{5.2}$$

where we recall that for any set $X \subseteq \mathbb{R}^n$, $\operatorname{conv}(X) \subseteq \mathbb{R}^n$ is the *convex hull* of X, i.e., the unique smallest convex set containing X. Also, remember that for a set $X \subseteq \mathbb{R}^n$, the *convex hull* $\operatorname{conv}(X)$ of X can be described by all *convex combinations* of points in X, i.e.,

$$\operatorname{conv}(X) = \left\{ \sum_{i=1}^k \lambda_i x_i \;\middle|\; k \in \mathbb{Z}_{>0}, x_i \in X \text{ and } \lambda_i \in \mathbb{R}_{\geq 0} \ \forall i \in [k] \text{ and } \sum_{i=1}^k \lambda_i = 1 \right\} \ .$$

Furthermore, by Carathéodory's Theorem, one can fix k to be equal to $n+1$ in the above description of $\operatorname{conv}(X)$, still obtaining the same set.

Hence, $P_{\mathcal{F}}$ is a $\{0,1\}$-polytope, i.e., all its vertices only have coordinates within $\{0,1\}$. Notice that solving the original combinatorial optimization problem (5.1) is equivalent to finding a vertex solution—also called *basic feasible solution*—to the following LP, where we interpret $w\colon N \to \mathbb{Z}$ as a vector $w \in \mathbb{Z}^N$:

$$\boxed{\begin{array}{rl} \max / \min & w^\top x \\ & x \in P_{\mathcal{F}} \end{array}} \tag{5.3}$$

The main challenge with this approach is that the description of $P_{\mathcal{F}}$ given by (5.2) refers to all sets in $\mathcal{F}$, which are typically exponentially (in $n := |N|$) many. Problems where $|\mathcal{F}|$ is small are often not that interesting. In particular, if $|\mathcal{F}| = O(\operatorname{poly}(n))$, then one could try to solve the combinatorial optimization problem efficiently by simply checking all feasible solutions. Of course, such an approach would still require the availability of a method to enumerate all feasible solutions.

Our goal is to get an inequality-description of $P_{\mathcal{F}}$, i.e., write $P_{\mathcal{F}}$ as $P_{\mathcal{F}} = \{x \in \mathbb{R}^N \colon Ax \leq b\}$. This has many advantages:

- Often, $P_{\mathcal{F}}$ only has $O(\operatorname{poly}(n))$ facets, even though it has $2^{\Omega(n)}$ vertices, i.e., feasible solutions. In this case, $P_{\mathcal{F}}$ can be described compactly by its facets.

→ Think about $P = [0,1]^n$. P has 2^n vertices but only $2n$ facets:

$$P = \{x \in \mathbb{R}^n \colon 0 \leq x_i \leq 1 \ \forall i \in [n]\} \ ,$$

where $[n] \coloneqq \{1, \dots, n\}$.

- One good inequality description of $P_{\mathcal{F}}$ allows for optimizing any linear objective function.

  → Combinatorial algorithms are often less versatile. E.g., Edmonds' blossom shrinking algorithm is a very elegant combinatorial algorithm for finding a maximum cardinality matching. However, it is hard to generalize this approach to the weighted case.

- Even when $P_{\mathcal{F}}$ has exponentially many facets, a description of them can often be obtained. If so, we can often still solve the LP efficiently.

  → For example, this might be possible by employing the Ellipsoid Method. We will discuss the Ellipsoid Method later in class, when we talk about the "equivalence" between separation and optimization.

- An inequality-description of $P_{\mathcal{F}}$ is often helpful when trying to solve a related combinatorial optimization problem, e.g., the original one with some additional linear constraints.
- The LP dual of (5.3) can often be interpreted combinatorially. Possible implications:
  - Optimality certificates through strong duality. E.g., many classical certificates like max-flow min-cut can be derived this way.
  - Fast algorithms based on dual such as primal-dual algorithms.
- Get a better understanding of the original combinatorial optimization problem. In particular, one can use elegant polyhedral proof techniques to derive results.
- Leverage general polyhedral techniques to design strong algorithms. E.g., network simplex algorithms are a way to interpret the Simplex Method combinatorially.

5.2 Meta-recipe for finding inequality-descriptions

There is no one-fits-all method. However, one typical high-level approach is to "guess" the polytope and show that it is the right one, following the recipe below:

(i) Determine (guess) a candidate polytope $P \subseteq [0,1]^N$ defined by some linear inequality description $P = \{x \in \mathbb{R}^N \colon Ax \leq b\}$, which we want to prove to be equal to $P_{\mathcal{F}}$.

(ii) Prove that P contains the correct set of integral points, i.e.,

$$P \cap \{0,1\}^N = \{\chi^F \colon F \in \mathcal{F}\} \ .$$

This is normally quite easy. Notice that $\{\chi^F \colon F \in \mathcal{F}\} = P_{\mathcal{F}} \cap \{0,1\}^N$.

(iii) Show that P is a $\{0,1\}$-polytope. Because $P \subseteq [0,1]^N$, this is equivalent to showing that P is integral.

If one can show (ii) and (iii), then $P = P_{\mathcal{F}}$, because P only contains $\{0,1\}$-vertices by (iii), and those vertices are the same as the ones of $P_{\mathcal{F}}$ due to (ii).

Typically, the most difficult step of the above recipe is point (iii), i.e., to show that P is integral. Also, we remark that the above recipe can be used iteratively to correct a potentially wrong guess of the inequality description. More precisely, if we realize that the statement we have to prove in step (ii) or (iii) is incorrect, then we know that our candidate inequality description that we established in point (i) is erroneous. Moreover, the insights we gain when thinking about why point (ii) or (iii) is not correct often help to improve the inequality description coming from step (i). Iterating this procedure is one way to come up with the correct description.

5.2.1 Example: bipartite vertex cover

To exemplify the above recipe, we consider the bipartite vertex cover problem. We first define the notion of a vertex cover in a general graph.

Definition 5.1: Vertex cover

Let $G = (V, E)$ be an undirected graph. A vertex cover of G is a subset $S \subseteq V$ such that for every edge $e \in E$, at least one of its endpoints is in S.

The minimum vertex cover problem, where each vertex has a non-negative cost and the goal is to find a minimum cost vertex cover, is a classical combinatorial optimization problem. Our goal here is to describe the vertex cover polytope for a bipartite graph. Hence, let $G = (V, E)$ be a bipartite graph and let $\mathcal{F} \subseteq 2^V$ be the family of all vertex covers in G. Our goal is to obtain an inequality description of the vertex cover polytope $P_{\mathcal{F}}$, i.e.,

$$P_{\mathcal{F}} = \operatorname{conv}\left(\{\chi^F \colon F \in \mathcal{F}\}\right) \quad .$$

We follow the recipe and start with the following natural candidate:

$$P := \left\{x \in [0,1]^V \colon x(u) + x(v) \geq 1 \;\forall \{u,v\} \in E\right\} \quad .$$

Notice that this candidate is essentially a literal translation of the property of being a vertex cover into linear constraints. This completes point (i) of the recipe. Through step (ii) and (iii) of the recipe, we now prove that this is a correct inequality description for the bipartite vertex cover polytope.

Theorem 5.2

The vertex cover polytope of a bipartite graph $G = (V, E)$ can be described by

$$P = \left\{x \in [0,1]^V \colon x(u) + x(v) \geq 1 \;\forall \{u,v\} \in E\right\} \quad .$$

Proof. We now show point (ii) of the recipe, i.e., that P contains the correct set of integral points, which means $P \cap \{0,1\}^V = \{\chi^F \colon F \in \mathcal{F}\}$. Clearly, if $S \subseteq V$ is a vertex cover, then

$\chi^S \in P$. Moreover, consider a point $y \in P \cap \{0,1\}^V$. Because y is a $\{0,1\}$-point, it can be written as $y = \chi^S$, where $S \coloneqq \{v \in V : y(v) = 1\}$. Due to the constraints of P, we have

$$1 \leq y(u) + y(v) = |S \cap \{u,v\}| \quad \forall \{u,v\} \in E \enspace .$$

Hence, at least one of the endpoints of any edge $\{u,v\} \in E$ is in S. Thus, S is a vertex cover and hence $y = \chi^S \in \{\chi^F : F \in \mathcal{F}\}$.

It remains to show point (iii) of the recipe, i.e., that P is integral. For the sake of deriving a contradiction, assume that P is not integral. Hence, P contains a fractional vertex y, i.e., $y \in P \setminus \{0,1\}^V$. Let $V = A \dot\cup B$ be a bipartition of the vertices such that every edge has one endpoint in A and one in B. We define

$$\begin{aligned} W_A &\coloneqq \{u \in A : y(u) \in (0,1)\} \quad \text{and} \\ W_B &\coloneqq \{u \in B : y(u) \in (0,1)\} \end{aligned}$$

to be the vertices in A and B, respectively, with fractional y-value. Because y is not integral by assumption, we have $W_A \cup W_B \neq \emptyset$. We will derive a contradiction by showing that y is not an extreme point of P. Let

$$\varepsilon \coloneqq \min\left\{\min\{y(u), 1 - y(u)\} : u \in W_A \cup W_B\right\} \enspace .$$

Note that $\varepsilon > 0$. Now define for any $\delta \in \mathbb{R}$:

$$y^\delta \coloneqq y + \delta \cdot (\chi^{W_A} - \chi^{W_B}) \enspace .$$

We claim that $y^\varepsilon \in P$ and $y^{-\varepsilon} \in P$. Notice that this will lead to the desired contradiction because $y = \frac{1}{2}(y^\varepsilon + y^{-\varepsilon})$ and $y^\varepsilon \neq y^{-\varepsilon}$ because $\chi^{W_A} - \chi^{W_B}$ is not the all-zeros vector as $W_A \cup W_B \neq \emptyset$. Hence, this implies that y is not an extreme point of P and therefore also not a vertex of P due to Proposition 1.19. Hence, it remains to show $y^\varepsilon, y^{-\varepsilon} \in P$. In fact, it suffices to show $y^\varepsilon \in P$, which implies $y^{-\varepsilon} \in P$ by exchanging the roles of A and B.

By our choice of ε, we clearly have $y^\varepsilon \in [0,1]^V$. Hence, it remains to check the constraints of P induced by the edges. Let $\{u,v\} \in E$. If either $y(u) = 1$ or $y(v) = 1$, then $y^\varepsilon(u) + y^\varepsilon(v) \geq 1$ because if a y-value is 1 at a certain entry, then so is the value of y^ε at the same entry, because y^ε only modifies fractional entries of y. Hence, we only have to check whether $y^\varepsilon(u) + y^\varepsilon(v) \geq 1$ if both $y(u)$ and $y(v)$ are fractional. However, in this case one of these two y-values will increase by ε and the other one decrease by ε, and thus

$$y^\varepsilon(u) + y^\varepsilon(v) = y(u) + y(v) \geq 1 \enspace ,$$

where the inequality follows from $y \in P$. □

5.3 Total unimodularity

There are various techniques to show integrality of a polyhedron, and we cover many different approaches here. In the example of the bipartite vertex cover problem, we have already seen one method, where we disproved the existence of fractional extreme points (see proof of Theorem 5.2). One elegant way to prove that certain polyhedra are integral is based on showing that the constraint matrix A used in the linear inequality description has a very strong structural property, known as *total unimodularity*. Even though most constraint matrices of integral polyhedra are not totally unimodular, it is still important to understand when we deal with matrices exhibiting this strong property, as this leads to short and elegant proofs, and allows for deriving many additional interesting properties.

5.3.1 Definition and basic observations

Definition 5.3

A matrix is *totally unimodular* (TU) if the determinant of any square submatrix of it is either 0, 1, or -1.

Remark 5.4

$A \in \mathbb{R}^{m\times n}$ is TU $\Rightarrow A \in \{-1,0,1\}^{m\times n}$.

This follows by observing that each single entry of the matrix A is itself a square submatrix and its determinant is the entry itself.

Remark 5.5

A is TU $\Leftrightarrow A^\top$ is TU.

This follows by the definition of TU matrices and the fact that the determinant of any square matrix is equal to the determinant of its transpose.

Remark 5.6

If $A \in \mathbb{R}^{m\times n}$ is TU, then so is $[A\ -A]$, i.e., the $\mathbb{R}^{m\times 2n}$ matrix obtained by appending the columns of $-A$ to the columns of A.

Indeed, $[A\ -A]$ is TU because any square submatrix of $[A\ -A]$ either contains a column of A and its negation in $-A$, in which case the determinant is zero, or it is a square submatrix of A with some columns multiplied by -1, in which case we have a determinant within $\{-1,0,1\}$ due to TU-ness of A.

Remark 5.7

If $A \in \mathbb{R}^{m\times n}$ is TU, then so is $[A\ I]$, i.e., the $\mathbb{R}^{m\times(n+m)}$ matrix obtained by appending the columns of an $m\times m$ identity matrix I to the columns of A.

Indeed, any determinant of a submatrix of $[A\ I]$ is a determinant of a submatrix of A, which is TU by assumption.

5.3.2 Integrality of polyhedra with TU constraint matrices

The main algorithmic motivation for studying totally unimodular matrices comes from the following theorem.

Theorem 5.8

Let $A \in \mathbb{Z}^{m\times n}$. Then,

$$A \text{ is TU} \quad \Leftrightarrow \quad P = \{x \in \mathbb{R}^n \colon Ax \leq b, x \geq 0\} \text{ is integral } \forall\, b \in \mathbb{Z}^m.$$

Proof. $\Rightarrow$) Assume that A is TU and $b \in \mathbb{Z}^m$, and let $P = \{x \in \mathbb{R}^n \colon Ax \leq b, x \geq 0\}$. To show that P is integral consider a vertex $y \in \text{vertices}(P)$. By Proposition 1.19, y is the unique solution to some subsystem $Dx = d$ of the system

$$\begin{pmatrix} A \\ -I \end{pmatrix} x \leq \begin{pmatrix} b \\ 0 \end{pmatrix} \quad , \tag{5.4}$$

which defines P. Moreover, we can choose D to be a square matrix because any linear system with a unique solution contains a square subsystem with the same unique solution. Hence,

$$y = D^{-1}d \ .$$

Now observe that D^{-1} is an integral matrix, because D is a sub-matrix of the matrix in (5.4), which is TU by Remark 5.6 and 5.7; therefore, because D is full-rank, we have $\det(D) \in \{-1, 1\}$ which, together with $D \in \mathbb{Z}^{n\times n}$, implies integrality of D^{-1}. Moreover, also $d \in \mathbb{Z}^n$ is an integral vector because its entries are a subset of the entries of the right-hand side of (5.4). Hence, $y = D^{-1}d$ is integral, as desired.

$\Leftarrow$) We show this direction by proving the contraposition. Hence, assume that A is not TU, which implies that there is some square submatrix Q of A with $\det(Q) \notin \{-1, 0, 1\}$. Because the statement we want to prove is invariant with respect to row and column permutations, we assume without loss of generality that Q consists of the first k rows and k columns of A. Notice that this implies $k \leq \min\{m, n\}$ because $A \in \mathbb{R}^{m\times n}$. Consequently, when defining an $n \times n$ submatrix of $\left(\begin{smallmatrix} A \\ -I \end{smallmatrix}\right)$ consisting of the k first rows and $n-k$ last rows, a square matrix $H \in \mathbb{R}^{n\times n}$ is obtained with $\det(H) \notin \{-1, 0, 1\}$, namely

$$H = \begin{pmatrix} Q & W \\ 0 & -I \end{pmatrix} \ ,$$

where I in the matrix above is an $(n-k) \times (n-k)$ identity matrix and $W \in \mathbb{Z}^{k\times(n-k)}$. Hence, we have

$$H^{-1} = \begin{pmatrix} Q^{-1} & Q^{-1}W \\ 0 & -I \end{pmatrix} \ .$$

Notice that because $\det(Q) \notin \{-1, 0, 1\}$ we have that Q^{-1} is not an integral matrix. One easy way to see this is by observing that $\det(Q^{-1}) = {}^{1}\!/_{\det(Q)} \notin \mathbb{Z}$ and using that a matrix with a non-integral determinant must have at least one non-integral entry. Let $j \in [k]$ be the index of a column of Q^{-1} that is not integral. We now define a vector $b \in \mathbb{Z}^m$ such that $P = \{x \in \mathbb{R}^n \colon Ax \leq b, x \geq 0\}$ is not integral, which will finish the proof. For this we start by defining the first k coordinates of b, i.e., those that correspond to the rows of Q. Let $e_j \in \{0,1\}^k$ be the j-th canonical unit vector in dimension k, and let $\mathbf{1} \in \{0,1\}^k$ be the all-ones vector. Moreover, let $p \in \mathbb{Z}$ be an integer that is big enough such that

$$Q^{-1}e_j + p \cdot \mathbf{1} \geq 0 \ . \tag{5.5}$$

We set the first k entries of b to $e_j + p \cdot Q\mathbf{1}$. We will set the remaining coordinates of b such that the following vector $y \in \mathbb{R}^n$ is a vertex of P: The first k coordinates of y are $Q^{-1}e_j + p \cdot \mathbf{1}$, all other coordinates of y are 0. First observe that y is fractional because the j-th column of Q^{-1} is fractional. We also have $y \geq 0$ due to (5.5). Moreover, y is by construction the unique solution to the subsystem of $Ax \leq b, x \geq 0$ that was used to define H. This subsystem is of the form $Hx = d$, where the first k rows of d are $e_j + p \cdot Q\mathbf{1}$, and all the other entries of d are zero. Hence, by choosing the last $m - k$ entries of b large enough, to make sure that y fulfills $Ay \leq b$, we have by Proposition 1.19 that y is a vertex of P. □

Hence, if we have an inequality description of a polyhedron as in Theorem 5.8 with A being TU and b integral, then the above theorem immediately implies that the polyhedron is integral. However, we still need techniques to check whether a given matrix is totally unimodular.

5.3.3 The characterization of Ghouila-Houri

The definition of total unimodularity as well as Theorem 5.8 do not provide very useful means to check that some matrix is totally unimodular. One of the most useful characterization of total unimodularity is the following theorem by Ghouila and Houri.

Theorem 5.9: Characterization of Ghouila-Houri

A matrix $A \in \mathbb{R}^{m \times n}$ is TU if and only if for every subset of the rows $R \subseteq [m]$, there is a partition $R = R_1 \,\dot{\cup}\, R_2$ such that

$$\sum_{i \in R_1} A_{ij} - \sum_{i \in R_2} A_{ij} \in \{-1, 0, 1\} \quad \forall j \in [n] \ . \tag{5.6}$$

Proof. $\Rightarrow$) We start by assuming that A is TU and show that the stated row-partitioning property holds. Hence, let $R \subseteq [m]$ be a subset of the rows of A. Let $d \in \mathbb{R}^m$ be defined by

$$d_i = \begin{cases} 1 & \text{if } i \in R \ , \\ 0 & \text{if } i \in [m] \setminus R \ , \end{cases}$$

and we define the polytope

$$Q := \left\{ x \in \mathbb{R}^m \colon A^\top x \leq \left\lceil \frac{1}{2} A^\top d \right\rceil ,\ A^\top x \geq \left\lfloor \frac{1}{2} A^\top d \right\rfloor ,\ x \leq d,\ x \geq 0 \right\} \ .$$

By Theorem 5.8, Q is an integral polytope, because its constraint matrix (without including the non-negativity constraints) is

$$\begin{pmatrix} A^\top \\ -A^\top \\ I \end{pmatrix} \quad ,$$

which is TU due to Remark 5.5, 5.6, and 5.7. Furthermore, $Q \neq \emptyset$ because $\frac{d}{2} \in Q$. Therefore Q has a vertex, say y, which must be integral. Due to the upper and lower bounds in the definition of Q, we have $y \in \{0,1\}^m$. We now define the following partition of R into R_1 and R_2:

$$\begin{aligned} R_1 &:= \{i \in R \colon y_i = 0\} \quad , \\ R_2 &:= \{i \in R \colon y_i = 1\} \quad . \end{aligned}$$

This partition indeed fulfills (5.6) because

$$\sum_{i \in R_1} A_{ij} - \sum_{i \in R_2} A_{ij} = (d-2y)^\top A_{\cdot j} = (A^\top)_{j\cdot}(d-2y) \in \{-1,0,1\} \qquad \forall j \in [n] \quad ,$$

where the fact that the expression $(A^\top)_{j\cdot}(d-2y)$ is within $\{-1,0,1\}$ follows from integrality of $d-2y$ together with $y \in Q$.

$\Leftarrow$) We now show the converse by showing that for any square matrix A, if A fulfills the row-partition property (5.6), then $\det(A) \in \{-1,0,1\}$. Notice that it suffices to consider square matrices. Indeed, for a general matrix B satisfying the row-partitioning property of Ghouila-Houri, we need to show that each square submatrix A of B fulfills $\det(A) \in \{-1,0,1\}$. If B satisfies the row-partitioning property of Ghuila-Houri, then so does A. If the implication we aim at proving holds for square matrices, then this implies $\det(A) \in \{-1,0,1\}$ as desired.

We show this statement by induction on the size k of the square matrix $A \in \mathbb{R}^{k \times k}$. For $k=1$, the statement clearly holds. Hence, we consider a matrix $A \in \mathbb{R}^{(k+1)\times(k+1)}$, for $k \in \mathbb{Z}_{\geq 1}$, that fulfills the row-partition property (5.6), and we assume that the row-partition property on any square matrix of size at most k implies that its determinant is within $\{-1,0,1\}$.

By letting R be a single row of A, we obtain that each entry of A is within $\{-1,0,1\}$. If A is the all-zeros matrix, the statement trivially holds; hence, assume that A has some non-zero entry. Because the statement is invariant with respect to row and column permutations and multiplying rows and columns by -1, we can assume that $A_{k+1,k+1} = 1$. We denote by $\begin{pmatrix} r^\top & 1 \end{pmatrix}$ the last row of A, by $\left(\begin{smallmatrix} c \\ 1 \end{smallmatrix}\right)$ the last column of A, and by $Q \in \mathbb{R}^{k \times k}$ the submatrix of A consisting of the first k rows and columns, i.e.,

$$A = \begin{pmatrix} Q & c \\ r^\top & 1 \end{pmatrix} \quad .$$

Through elementary row operations—more precisely, by adding or subtracting the last row to some of the other rows—we obtain the following matrix:

$$\overline{A} = \begin{pmatrix} Q - c \cdot r^\top & 0 \\ r^\top & 1 \end{pmatrix} \quad .$$

Because these row operations do not change the determinant of a matrix, we have $\det(A) = \det(\overline{A}) = \det(Q - c \cdot r^\top)$. Hence, it suffices to show $\det(Q - c \cdot r^\top) \in \{-1,0,1\}$. To this end, we show that $Q - c \cdot r^\top$ fulfills the row-partition condition (5.6), which then implies the statement by the inductive hypothesis. Thus, let $R \subseteq [k]$ be a subset of the rows of $Q - c \cdot r^\top$. We distinguish two cases depending on the parity of $|R \cap \text{supp}(c)|$, where $\text{supp}(c) := \{i \in [k] : c_i \neq 0\}$ is the *support* of the vector c.

Case $|R \cap \text{supp}(c)|$ even: By assumption, the Ghuila-Houri criterion holds for the matrix $\begin{pmatrix} Q & c \end{pmatrix}$, because it is a submatrix of A. Hence, there is a vector $d \in \{-1,0,1\}^k$ with

(i) $\text{supp}(d) = R$,
(ii) $d^\top Q \in \{-1,0,1\}^k$, and
(iii) $d^\top c \in \{-1,0,1\}$.

More precisely, d encodes a partition $R = R_1 \,\dot{\cup}\, R_2$, where the 1-entries of d correspond to the rows in R_1 and the -1-entries to the rows in R_2. We show that this same partition of R can be used for $Q - c \cdot r^\top$. Note that because $|R \cap \text{supp}(c)|$ is even and $\text{supp}(d) = R$, we must have that $d^\top c$ is even. Thus, because $d^\top c \in \{-1,0,1\}$, we have $d^\top c = 0$. However, this, together with (ii), implies

$$d^\top (Q - c \cdot r^\top) = d^\top Q \in \{-1,0,1\}^k \ ,$$

showing that the partition of R defined by d fulfills (5.6), as desired.

Case $|R \cap \text{supp}(c)|$ odd: In this case we use the fact that the matrix A fulfills the row-partition condition for the rows $R \cup \{k+1\}$. Again, we can represent the partition of $R \cup \{k+1\}$ by a vector in $\{-1,0,1\}^{k+1}$. We can assume that row $k+1$ is in the second part of the partition, because the two parts are exchangeable, i.e., condition (5.6) is invariant with respect to exchanging R_1 and R_2. Hence, there is a partition of the rows $R \cup \{k+1\}$ of A fulfilling (5.6) that can be described by a vector $\binom{d}{-1} \in \{-1,0,1\}^{k+1}$, where $d \in \{-1,0,1\}^k$. Thus, we have

(i) $\text{supp}(d) = R$,
(ii) $d^\top Q - r^\top \in \{-1,0,1\}^k$, and
(iii) $d^\top c - 1 \in \{-1,0,1\}$.

We claim that d describes a desired partition of the rows R of the matrix $Q - c \cdot r^\top$. First observe that because $|R \cap \text{supp}(c)|$ is odd and $\text{supp}(d) = R$, we have that $d^\top c$ is odd. Due to (iii), this implies $d^\top c = 1$. Hence, together with (ii), this leads to

$$d^\top (Q - c \cdot r^\top) = d^\top Q - r^\top \in \{-1,0,1\}^k \ ,$$

showing that the partition of R defined by d fulfills (5.6), as desired. □

Remark 5.10

Because A is TU if and only if $A^\top$ is TU, one can exchange the roles of rows and columns in Theorem 5.9.

Example 5.11: Consecutive-ones matrices

To exemplify how the characterization of Ghuila-Houri can be used, we show that consecutive-ones matrices are TU. A *consecutive-ones* matrix is a $\{0,1\}$-matrix $A \in \{0,1\}^{m \times n}$ such that either in each column all the 1s appear consecutively, or in each row all the 1s appear consecutively. Below are two examples of consecutive-ones matrices.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

We will show that any consecutive-ones matrix is TU.

Consider a consecutive-ones matrix $A \in \mathbb{R}^{m \times n}$, where the 1s are consecutive in each column. (This covers the general case because a consecutive-ones matrix where the 1s are consecutive in each row is just the transpose of one where they are consecutive in each column.) Now consider any subset $R \subseteq [m]$ of the rows of A. We have to find a partitioning of them satisfying (5.6). Let $B \in \mathbb{R}^{r \times n}$ be the submatrix of A consisting of the rows in R; hence, $r := |R|$. Notice that B is also a consecutive-ones matrix where the 1s appear consecutively in each column. We partition its rows by alternatingly putting them into R_1 and R_2. More precisely, all rows with an odd row index in B are assigned to R_1 and the other ones to R_2. One can easily check that this partition fulfills (5.6) because in each column the difference between the number of 1s assigned to R_1 and the number of 1s assigned to R_2 is at most one.

5.4 Bipartite matching polytope

Let $G = (V, E)$ be an undirected, bipartite graph with corresponding bipartition $V = X \dot\cup Y$, and let $\mathcal{M} \subseteq 2^E$ be all matchings of G. Hence, using the notation introduced previously, we have $N = E$ and $\mathcal{F} = \mathcal{M}$.

Theorem 5.12

The bipartite matching polytope $P_{\mathcal{M}}$ is given by

$$P_{\mathcal{M}} = \{x \in \mathbb{R}^E_{\geq 0} \colon x(\delta(v)) \leq 1 \ \forall v \in V\} \ . \tag{5.7}$$

We will prove Theorem 5.12 by first showing (ii), and then presenting two general techniques that prove (iii). Let $P = \{x \in \mathbb{R}^E_{\geq 0} \colon x(\delta(v)) \leq 1 \ \forall v \in V\}$. Hence, we have to show $P = P_{\mathcal{M}}$.

Proof of point (ii). We prove (ii) by showing that for any $F \subseteq E$ we have

$$\chi^F \in P \cap \{0,1\}^N \Leftrightarrow F \in \mathcal{M} \ .$$

We distinguish between $F \in \mathcal{M}$ and $F \in 2^E \setminus \mathcal{M}$.

If $F \in \mathcal{M}$, then, as F is a matching, we have $|F \cap \delta(v)| \leq 1 \ \forall v \in V$. Notice that $|F \cap \delta(v)| = \chi^F(\delta(v))$. Hence, $\chi^F(\delta(v)) \leq 1 \ \forall v \in V$, and thus $\chi^F \in P$, as desired.

If $F \in 2^E \setminus \mathcal{M}$, then, as F is not a matching, there exists a vertex $v \in V$ such that $|F \cap \delta(v)| \geq 2$, which can be rephrased as $\chi^F(\delta(v)) \geq 2$. Hence, $\chi^F \notin P$. □

It remains to prove (iii).

We will see two proofs of (iii): $<$ one showing that the constraint matrix is totally unimodular,
one showing that P has no fractional extreme points.

5.4.1 Integrality through TU-ness

We can write P in the following form:

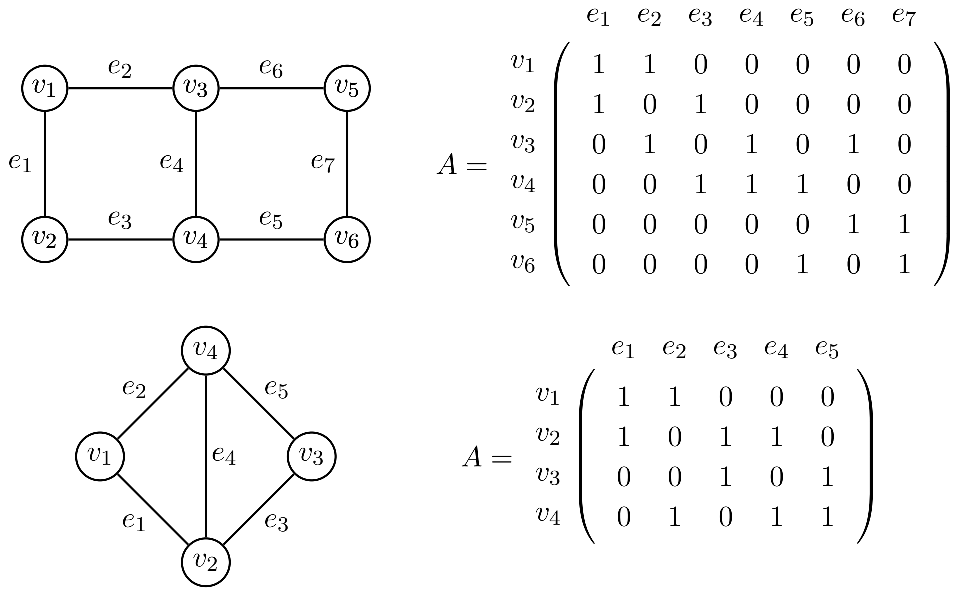
$$P = \{x \in \mathbb{R}^E \colon Ax \leq b, x \geq 0\} \enspace ,$$

where A and b are defined as follows. First observe that because P contains one constraint for each vertex, and one variable for each edges, we have $A \in \mathbb{R}^{V \times E}$ and $b \in \mathbb{R}^V$. The right-hand side of each constraint of P, that is not a non-negativity constraint, is 1, i.e., $b = \mathbf{1}$, where $\mathbf{1} \in \mathbb{R}^V$ is the all-ones vector.

The matrix A is the *vertex-edge incidence matrix* of G, i.e., for every $v \in V$ and $e \in E$,

$$A(v,e) = \begin{cases} 1 & \text{if } v \in e \enspace , \\ 0 & \text{if } v \notin e \enspace . \end{cases}$$

Figure 5.1 shows two example graphs—one bipartite and the other one non-bipartite—with their corresponding vertex-edge incidence matrices, denoted by A.



$$A = \begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline v_1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ v_2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ v_3 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ v_4 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ v_5 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ v_6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}$$

$$A = \begin{array}{c|ccccc} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline v_1 & 1 & 1 & 0 & 0 & 0 \\ v_2 & 1 & 0 & 1 & 1 & 0 \\ v_3 & 0 & 0 & 1 & 0 & 1 \\ v_4 & 0 & 1 & 0 & 1 & 1 \end{array}$$

Figure 5.1: Example graphs with corresponding vertex-edge incidence matrices.

Theorem 5.13

Let $G = (V, E)$ be an undirected graph with vertex-edge incidence matrix A. Then,

$$G \text{ is bipartite } \Leftrightarrow A \text{ is TU.}$$

Proof. $\Leftarrow$) Left as an exercise.

$\Rightarrow$) We use the characterization of Ghouila-Houri to show that A is totally unimodular if $G = (V, E)$ is bipartite with bipartition $V = X \dot\cup Y$.

Consider a subset of the rows of A. This subset corresponds to some subset of the vertices $R \subseteq V$. We partition R into $R_1 = R \cap X$ and $R_2 = R \cap Y$. For any column of A, which corresponds to some edge $e \in E$, we have

$$\underbrace{\sum_{v \in R_1} A_{v,e}}_{=|e \cap R_1| \leq |e \cap X| = 1} \quad - \quad \underbrace{\sum_{v \in R_2} A_{v,e}}_{=|e \cap R_2| \leq |e \cap Y| = 1} \quad \in \{-1, 0, 1\} \enspace .$$

Notice that the above statement follows by the fact that each of the two sums in the above expression is either 0 or 1. Thus, their difference is either -1, 0, or 1. This shows that the criterion of Ghouila and Houri holds for the matrix A, implying that A is totally unimodular. □

Combining Theorem 5.13 and Theorem 5.8, we obtain that $P = \{x \in \mathbb{R}^E \colon Ax \leq \mathbf{1}, x \geq 0\}$ is integral, and thus $P = P_{\mathcal{M}}$ as desired.

5.4.2 Integrality by disproving existence of fractional extreme points

Let $x \in P$ be a vertex of P. For sake of contradiction assume $x \notin \{0,1\}^E$. Let $U \subseteq E$ be all edges with fractional values, i.e.,

$$U = \{e \in E \colon 0 < x(e) < 1\} \enspace .$$

Since x is not integral, $U \neq \emptyset$. We distinguish two cases, depending on whether U contains a cycle.

First case: U contains a cycle $C \subseteq U$.

Hence, C must be even since G is bipartite. Let $C = \{e_1, \dots, e_k\} \subseteq U$ with k even be the edges along C. Let

$$\begin{aligned} W_1 &= \{e_i \colon i \in [k], i \text{ odd}\} \enspace , \\ W_2 &= \{e_i \colon i \in [k], i \text{ even}\} \enspace . \end{aligned}$$

For $\varepsilon \in \mathbb{R}$, let $x^\varepsilon \in \mathbb{R}^E$ be defined by

$$x^\varepsilon = x + \varepsilon \cdot \chi^{W_1} - \varepsilon \cdot \chi^{W_2} \enspace .$$

We will show that there is some $\rho > 0$ such that $x^\rho, x^{-\rho} \in P$. This contradicts x being a vertex, or equivalently, an extreme point; indeed, x would be the midpoint of the two distinct points

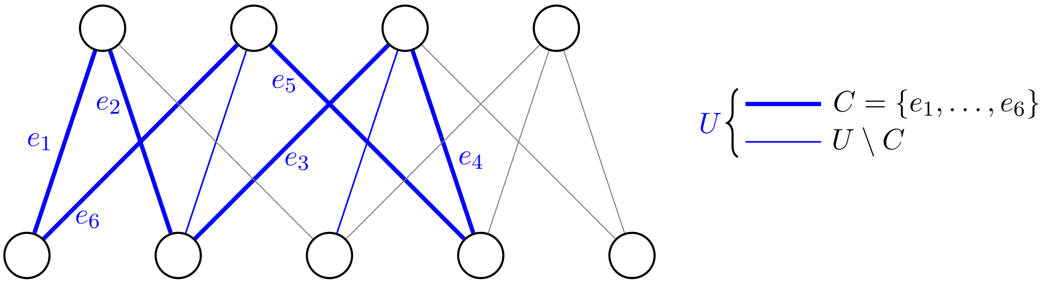


Figure 5.2: Example for first case of proof where U contains a cycle C.

$x^\rho \in P$ and $x^{-\rho} \in P$. Without loss of generality we only prove $x^\rho \in P$; the statement $x^{-\rho} \in P$ reduces to this case by exchanging the roles of W_1 and W_2, which can be done by renumbering the edges on C.

Let

$$\rho = \min_{e \in C}\{x(e)\} \ .$$

We clearly have $\rho > 0$ since $x(e) > 0 \ \forall e \in U$ and $C \subseteq U$. For any $v \in V$, we have

$$x^\rho(\delta(v)) = x(\delta(v)) \leq 1 \ ,$$

which is true even for any $\rho \in \mathbb{R}$. Furthermore, for $e \in E \setminus C$, we have $x^\rho(e) = x(e) \geq 0$, and for $e \in C$ we have

$$x^\rho(e) \geq x(e) - \underbrace{\rho}_{\leq x(e)} \geq 0 \ .$$

Thus, $x^\rho, x^{-\rho} \in P$, leading to a contradiction of x being a vertex.

Second case: U does not contain cycles.

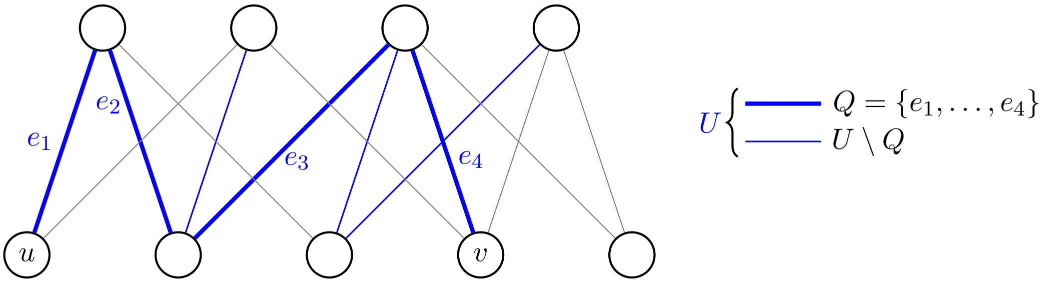


Figure 5.3: Example for second case of proof where U is a forest.

Hence U are the edges of a forest. Let $Q \subseteq U$ be any maximal path in U, i.e., Q is the unique path between some pair of leaf vertices u, v of U. We number the edges in $Q = \{e_1, \dots, e_k\}$ in the way they are encountered when traversing Q from u to v. Let

$$W_1 = \{e_i \colon i \in [k], i \text{ odd}\} \ ,$$
$$W_2 = \{e_i \colon i \in [k], i \text{ even}\} \ .$$

Again, we define for $\varepsilon \in \mathbb{R}$

$$x^{\varepsilon} = x + \varepsilon \cdot \chi^{W_1} - \varepsilon \cdot \chi^{W_2} \ ,$$

and this time we set

$$\rho = \min_{e \in Q}\{\min\{x(e), 1 - x(e)\}\} \ .$$

By definition of ρ, we clearly have $\rho > 0$ and $x^{\rho}, x^{-\rho} \in [0,1]^E$. Consider x^{ρ} (reasoning for $x^{-\rho}$ is identical). For all $w \in V \setminus \{u, v\}$ we have $x^{\rho}(\delta(w)) = x(\delta(w)) \leq 1$. It remains to show $x^{\rho}(\delta(u)) \leq 1$ and $x^{\rho}(\delta(v)) \leq 1$. W.l.o.g. we only show $x^{\rho}(\delta(u)) \leq 1$, since the roles of u and v can be exchanged by reversing the path P.

Notice that only one edge of U is adjacent to u since u was chosen to be a leaf vertex of the forest U. Thus all edges $e \in \delta(u) \setminus U$ satisfy $x(e) \in \{0, 1\}$. However, no edge $e \in \delta(u) \setminus U$ can satisfy $x(e) = 1$ since this would imply $x \notin P$ as $x(\delta(u)) \geq x(e) + x(e_1) = 1 + x(e_1) > 1$. Hence, of all edges $e \in \delta(u)$, e_1 is the only edge with $x(e) > 0$. Thus $x^{\rho}(\delta(u)) = x^{\rho}(e_1) \leq 1$, since $x^{\rho} \in [0,1]^E$. Hence $x^{\rho} \in P$.

Thus x can again be expressed as the midpoint of $x^{\rho}, x^{-\rho} \in P$ with $x^{\rho} \neq x^{-\rho}$ and therefore cannot be a vertex of P. Hence, P is integral, and therefore $P = P_{\mathcal{M}}$.

5.4.3 Some implications coming from the inequality description of $P_{\mathcal{M}}$

Often, one can derive interesting results from an inequality description of a polytope. We provide two examples below linked to the bipartite matching polytope. More precisely, we first show an example how an inequality description of one polytope can sometimes be used to derive an inequality description of a related one. The second example shows how the polyhedral description that we derived for the bipartite matching polytope can be used to obtain a very short and elegant proof of a classical theorem in Graph Theory.

Perfect bipartite matching polytope A matching $M \in \mathcal{M}$ is called *perfect*, if it touches all vertices, i.e., $|M| = |V|/2$. Of course, a perfect matching can only exist if $|V|$ is even. Furthermore, for a bipartite graph $G = (V, E)$ with corresponding bipartition $V = X \dot{\cup} Y$, one needs $|X| = |Y|$ for a perfect matching to exist.

Theorem 5.14

The perfect matching polytope of a bipartite graph $G = (V, E)$ is given by

$$P = \left\{x \in \mathbb{R}^E_{\geq 0} \colon x(\delta(v)) = 1 \ \forall v \in V\right\} \ .$$

Proof. Again, one can easily check that P contains the correct set of integral points. Its integrality follows by observing that P is a face of the matching polytope $P_{\mathcal{M}}$ or the empty set, and hence an integral polytope.

A key property we exploit here is that a F face of an integral polyhedron Q is itself integral. This is a consequence of Corollary 1.14, which states that a face of a face is a face. Indeed, a vertex y of F is a 0-dimensional face of F. By Corollary 1.14, y is also a face of Q. Because y is 0-dimensional, it must be a vertex of Q. We now can exploit that Q is integral, which implies integrality of y. □

Perfect matchings in bipartite *d*-regular graphs A graph is called *d-regular* for some $d \in \mathbb{Z}_{\geq 0}$ if every vertex has degree d.

Theorem 5.15

Let $d \in \mathbb{Z}_{\geq 1}$. Every d-regular bipartite graph admits a perfect matching.

Proof. Using Theorem 5.14, let $P = \{x \in \mathbb{R}^E_{\geq 0} \colon x(\delta(v)) = 1\ \forall v \in V\}$ be the perfect bipartite matching polytope of a d-regular bipartite graph. Notice that the point $x \in \mathbb{R}^E$ given by $x(e) = \frac{1}{d}$ for every $e \in E$ satisfies $x \in P$. Hence, $P \neq \emptyset$, and P therefore contains a vertex, which corresponds to a perfect matching. □

5.5 Polyhedral description of shortest *s*-*t* paths

Let $G = (V, A)$ be a directed graph, and let $s, t \in V$ with $s \neq t$. Recall that a path is by definition vertex-disjoint, i.e., the same vertex is encountered at most once when traversing the path.

No "good" polyhedral description is known of the s-t path polytope, i.e., one over which we could optimize any linear function in polynomial time. This is not surprising, because such a description would imply $\mathcal{P} = \mathcal{NP}$ since one could solve the longest s-t path problem, which is well-known to be $\mathcal{NP}$-hard. In particular, one can solve the Hamiltonian path problem if one can find longest s-t paths: It suffices to find a longest s-t path with unit edge lengths for all $O(n^2)$ pairs of vertices $s, t \in V$, $s \neq t$, and check whether one of them is Hamiltonian.

Still, when trying to find shortest paths with respect to any positive length function, it is not hard to find a polyhedral approach. Consider the polytope

$$P = \left\{ x \in [0,1]^A \;\middle|\; x(\delta^+(v)) - x(\delta^-(v)) = \begin{cases} 1 & \text{if } v = s, \\ -1 & \text{if } v = t, \\ 0 & \text{if } v \in V \setminus \{s,t\}, \end{cases} \quad \forall v \in V \right\} .$$

Notice that P is simply the flow polytope of an s-t flow problem where a unit flow has to be sent from s to t in a directed graph with all arc capacities being 1. Consider first the integral points in P. Notice that integral points in P do not correspond one-to-one to s-t paths. More precisely, each s-t path is an integral point in P, however, some integral points in P are not s-t paths. More precisely, one can easily prove (and we leave this as an exercise) that integral points in P correspond precisely to the disjoint union of an s-t path and possibly some additional cycles.

Hence, if we can show integrality of P, then for any positive weights (or lengths) $w \colon A \to \mathbb{Z}_{>0}$, a basic solution to $\min\{w^\top x \colon x \in P\}$ will correspond to a w-shortest s-t path. Thus, P

allows us to efficiently find shortest s-t paths with linear programming techniques. Notice that even if the weights w are non-negative instead of positive, i.e., $w\colon E \to \mathbb{Z}_{\geq 0}$, we can still find a shortest path via linear programming on P. An optimal vertex solution will correspond to a minimum weight set $U \subseteq E$ that is a disjoint union of an s-t path and cycles, where all cycles must have zero length by optimality of U. One can easily show that any s-t path $P \subseteq U$ is a shortest s-t path in G. The same approach works even if the weights are allowed to have negative values, but have the property of being *conservative*, which means that no cycle has strictly negative weight. Conservative weights cover the classical shortest path settings considered by specialized shortest path algorithms.

To show integrality of P, we will show that its corresponding constraint matrix is TU. First observe that P can be rewritten as

$$P = \left\{x \in \mathbb{R}^A \colon Dx = b, \mathbf{1} \geq x \geq 0\right\} \quad ,$$

where $\mathbf{1} = \chi^A$ is the all-ones vector, $b \in \{-1,0,1\}^V$ is defined by

$$b(v) = \begin{cases} 1 & \text{if } v = s \ , \\ -1 & \text{if } v = t \ , \\ 0 & \text{if } v \in V \setminus \{s,t\} \ , \end{cases}$$

and $D \in \{-1,0,1\}^{V\times A}$ is the *vertex-arc incidence matrix* of the *directed* (loopless) graph G, which is defined as follows: for $v \in V$, $a \in A$:

$$D(v,a) = \begin{cases} 1 & \text{if } a \in \delta^+(v) \ , \\ -1 & \text{if } a \in \delta^-(v) \ , \\ 0 & \text{otherwise} \ . \end{cases}$$

It thus suffices to show that D is TU to obtain integrality of P.

Theorem 5.16

The vertex-arc incidence matrix $D \in \{-1,0,1\}^{V\times A}$ of any directed (loopless) graph $G = (V, A)$ is TU.

Proof. We apply the Ghouila-Houri characterization to the rows of D. For any subset $R \subseteq V$ of the rows, we choose the partition $R_1 = R$ and $R_2 = \emptyset$. Since each column of D has only zeros except for precisely one 1 and one -1, summing any subsets of the elements of any column will lead to a total sum of either -1, 0, or 1. Hence,

$$\sum_{v\in R_1} D_{v,a} \in \{-1,0,1\} \quad \forall a \in A \ ,$$

as desired. Or, more formally, for $a = (u,v) \in A$, we have

$$\sum_{w\in R_1} D_{w,a} = \underbrace{\mathbf{1}_{u\in R_1}}_{\in\{0,1\}} - \underbrace{\mathbf{1}_{v\in R_1}}_{\in\{0,1\}} \in \{-1,0,1\} \ ,$$

where, for $w \in V$, $\mathbf{1}_{w\in R_1} = 1$ if $w \in R_1$ and $\mathbf{1}_{w\in R_1} = 0$ otherwise. □

5.6 Spanning trees and r-arborescences

Spanning trees and their directed counterparts, known as arborescences, are among the most basic combinatorial structures in Graph Theory. Contrary to the polytopes we have seen so far, the combinatorial polytopes corresponding to spanning trees and arborescences have an exponential number of facets. In this section, we only provide a brief discussion of their corresponding polytopes. We prove the correctness of the presented linear inequality descriptions later on when we introduce a powerful proof technique known as combinatorial uncrossing.

5.6.1 Spanning tree polytope

Let $G = (V, E)$ be an undirected graph. A *spanning tree* in G is an edge set $T \subseteq E$ that connects all vertices and does not contain a cycle.

For any set $S \subseteq V$, we denote by $E[S] \subseteq E$ all edges with both endpoints in S, i.e.,

$$E[S] \coloneqq \{e \in E \colon e \subseteq S\} \ .$$

Theorem 5.17

The spanning tree polytope of an undirected loopless graph $G = (V, E)$ is given by

$$P = \left\{ x \in \mathbb{R}^E_{\geq 0} \;\middle|\; \begin{array}{ll} x(E) = |V| - 1 & \\ x(E[S]) \leq |S| - 1 & \forall S \subsetneq V, |S| \geq 2 \end{array} \right\} \ .$$

Again, one can easily check that P contains the correct set of integral points, i.e., the $\{0,1\}$-points in P are precisely the incidence vectors of spanning trees. To check this, the following equivalent definition of spanning trees is useful: A set $T \subseteq E$ is a spanning tree if and only if $|T| = |V| - 1$ and T does no contain any cycle.

The constraints of the spanning tree polytope are often divided into two groups, namely the *non-negativity constraints* $x \geq 0$, and all the other constraints which are called *spanning tree constraints*.

5.6.2 The r-arborescence polytope

Definition 5.18: Arborescence, r-arborescence

Let $G = (V, A)$ be a directed graph. An *arborescence* in G is an arc set $T \subseteq A$ such that

(i) T is a spanning tree (when disregarding the arc directions), and

(ii) there is one vertex r from which all arcs are directed away, i.e., every vertex $v \in V$ can be reached from r using a directed path in T.

The vertex r in condition (ii) is called the *root* of the arborescence, and T is called an *r-arborescence*.

Figure 5.4 shows an example of an r-arborescence.

Notice that condition (ii) can equivalently be replaced by

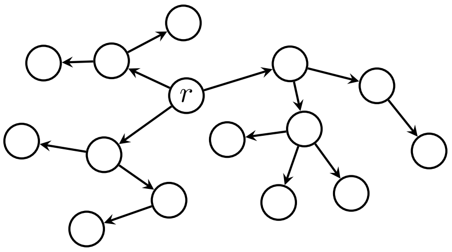


Figure 5.4: Example of an r-arborescence.

(ii') Every vertex has at most one incoming arc.

Theorem 5.19

The arborescence polytope of a directed loopless graph $G = (V, A)$ is given by

$$P = \left\{ x \in \mathbb{R}^A_{\geq 0} \;\middle|\; \begin{array}{ll} x(A) = |V| - 1 & \\ x(A[S]) \leq |S| - 1 & \forall S \subsetneq V, |S| \geq 2 \\ x(\delta^-(v)) \leq 1 & \forall v \in V \end{array} \right\} ,$$

where $A[S] \subseteq A$ for $S \subseteq V$ denotes all arcs with both endpoints in S.

A polyhedron that is closely related to the arborescence polytope and has a very elegant description is the *dominant of the r-arborescence polytope*. The dominant of the r-arborescence polytope will also provide an excellent example to show how integrality of a polyhedron can be proven using a technique called *combinatorial uncrossing*.

Apart from sometimes having a simpler description, the dominant of a polytope can also often be used for optimization. For example, consider the problem of finding a minimum weight r-arborescence with respect to some positive arc weights $w \in \mathbb{Z}^A_{>0}$. Let P be the r-arborescence polytope. Then this problem corresponds to minimizing $w^\top x$ over all $x \in P$. However, this is equivalent to minimizing $w^\top x$ over $x \in \text{dom}(P)$. Indeed, any $x \in \text{dom}(P)$ can be written as $x = y + z$, where $y \in P$ and $z \in \mathbb{R}^A_{\geq 0}$. Therefore for $x \in \text{dom}(P)$ to be a minimizer of $w^\top x$, we must have $z = 0$; for otherwise, $w^\top y < w^\top x$ and $y \in P \subseteq \text{dom}(P)$, violating that $x \in \text{dom}(P)$ minimizes $w^\top x$.

Theorem 5.20

The dominant of the r-arborescence polytope is given by

$$P = \left\{ x \in \mathbb{R}^A_{\geq 0} \colon x(\delta^-(S)) \geq 1 \quad \forall S \subseteq V \setminus \{r\}, S \neq \emptyset \right\} .$$

We will prove integrality of this polyhedron in the next section, when talking about combinatorial uncrossing. This particular inequality description of the dominant of the r-arborescence polytope will turn out to be useful later, when we design a separation oracle for it.

5.7 Non-bipartite matchings

We will start by introducing the perfect matching polytope and then derive therefrom the description of the matching polytope.

5.7.1 Perfect matching polytope

Theorem 5.21

The perfect matching polytope of an undirected graph $G = (V, E)$ is given by

$$P = \left\{ x \in \mathbb{R}^E_{\geq 0} \;\middle|\; \begin{array}{ll} x(\delta(v)) = 1 & \forall v \in V \\ x(\delta(S)) \geq 1 & \forall S \subseteq V, |S| \text{ odd} \end{array} \right\} .$$

Proof. It is easy to check that P contains the correct set of integral points. Thus, it remains to show integrality of P.

For the sake of contradiction assume that there are graphs $G = (V, E)$ for which P is not integral. Among all such graphs let $G = (V, E)$ be a one that minimizes $|V| + |E|$, i.e., we look at a smallest bad example, and let P be the corresponding polytope as defined in Theorem 5.21. Notice that we must have that $|V|$ is even. For otherwise the polytope P is indeed the perfect matching polytope because it is empty, which follows from the constraint $x(\delta(V)) \geq 1$, which is impossible to satisfy since $\delta(V) = \emptyset$.

Let $y \in P$ be a vertex of P that is fractional. We start by observing some basic properties following from the fact that we chose a smallest bad example. In particular, G is connected. For otherwise, a smaller bad example is obtained by only considering one of its connected components containing a y-fractional edge, which violates minimality of G. Moreover, there is no 0-edge, i.e., an edge $e \in E$ such that $y(e) = 0$, because such an edge could be deleted leading to a smaller bad example. Similarly, there is no 1-edge $e = \{u, v\} \in E$, because in this case there can be no other edge f except for e that is incident with either u or v, since we would have $y(f) = 0$ due to $y(\delta(u)) = 1$ and $y(\delta(v)) = 1$, and we already know that there is no 0-edge. Finally, $\delta(\{u, v\}) = \emptyset$ implies that G is not connected, because G must contain at least one edge besides e because there is an edge with fractional y-value. Hence, y is fractional on all edges, i.e., $y(e) \in (0, 1)$ $\forall e \in E$.

Since y is a vertex of P it can be defined as the unique solution to $|E|$ linearly independent constraints of P that are *y-tight*, i.e., tight for the point y. If it is clear from the context, we also often just talk about *tight* constraints. P has three types of constraints: *degree constraints* ($x(\delta(v)) = 1$ for $v \in V$), *cut constraints* ($x(\delta(S)) \geq 1$ for $S \subseteq V, |S|$ odd), and *non-negativity constraints* ($x(e) \geq 0$ for $e \in E$). Notice that none of the non-negativity constraints are tight since $y > 0$. Hence, y is the unique solution to a full-rank linear system of the following type:

$$\begin{aligned} x(\delta(v)) &= 1 \quad \forall v \in W \ , \\ x(\delta(S)) &= 1 \quad \forall S \in \mathcal{F} \ , \end{aligned} \tag{5.8}$$

where $W \subseteq V$, $\mathcal{F} \subseteq \{S \subseteq V \colon |S| \text{ odd}\}$, and $|W| + |\mathcal{F}| = |E|$ because the system is full-rank. Without loss of generality, we assume that $\mathcal{F}$ only contains sets $S \subseteq V$ such that $|S| \neq 1$ and

$|S| \neq |V| - 1$. Indeed, if $|S| = 1$, then the constraint $x(\delta(S)) = 1$ can be handled as a tight degree constraints. The same holds for $|S| = |V| - 1$, in which case the constraint $x(\delta(S)) = 1$ is identical to $x(\delta(V \setminus S)) = 1$, which is a degree constraint.

We now distinguish between two cases depending on whether $\mathcal{F} = \emptyset$ or not.

Case $\mathcal{F} = \emptyset$: We start with an observation that is independent of this case. Namely, each vertex must have degree at least two, since a degree-one vertex v would need to be adjacent to a 1-edge to satisfy the degree constraint $y(\delta(v)) = 1$. This implies $|E| \geq |V|$. Furthermore, since (5.8) is full-rank, we have $|W| = |E|$. Finally, $W \subseteq V$ implies $|W| \leq |V|$, and by combining all above inequalities we get $|V| \leq |E| = |W| \leq |V|$. Hence, $|E| = |W| = |V|$, i.e., the system (5.8) contains all degree constraints, and G is a graph where each vertex has degree precisely 2. Since G is connected, it must be a single cycle containing all vertices. Moreover, because $|V|$ is even, it is an even cycle. Let $V = \{v_1, \dots, v_n\}$ be a consecutive numbering of the vertices when going along the cycle. We now get a contradiction by showing that the system (5.8) is not full-rank. Indeed, we have

$$\sum_{i \in [n], i \text{ even}} \chi^{\delta(v_i)} = \sum_{i \in [n], i \text{ odd}} \chi^{\delta(v_i)} \quad ,$$

which shows that there is a linear dependence within the degree constraints.

Case $\mathcal{F} \neq \emptyset$: Let $S \in \mathcal{F}$. As discussed, we have $|S| > 1$ and $|V \setminus S| > 1$. We consider two graphs, $G_1 = G/S$ and $G_2 = G/(V \setminus S)$, which are obtained from G by contracting S and $V \setminus S$, respectively. Contracting S means that we replace all vertices in S by a single new vertex v_S. Furthermore, all edges with both endpoints in S are deleted, and any edge with one endpoint in S and the other one $v \in V \setminus S$ outside of S is replaced by an edge between v_S and v. Let y_1 and y_2 be the restriction of y to all non-deleted edges in G_1 and G_2, respectively. One can easily observe that $y_i \in P_{G_i}$ for $i \in \{1,2\}$, where P_{G_i} is the polytope as defined by Theorem 5.21 for the graph G_i. Notice that both graphs G_1 and G_2 are strictly smaller than G in terms of the sum of their number of vertices and edges. Since G was a smallest bad example, the two polytopes P_{G_1} and P_{G_2} are therefore the perfect matching polytopes of G_1 and G_2. Hence, we can write y_i for $i \in \{1,2\}$ as a convex combination of perfect matchings in G_i. In particular, there is some $N \in \mathbb{Z}_{>0}$ such that for $i \in \{1,2\}$

$$y_i = \frac{1}{N} \sum_{j=1}^{N} \chi^{M_i^j} \quad , \tag{5.9}$$

where M_i^j is a perfect matching in G_i for $i \in \{1,2\}$ and $j \in [N]$. Notice that both graphs G_1 and G_2 contain the edges $\delta(S)$. More precisely, for both graphs G_1, G_2, the set $\delta(S)$ consists of all edges incident with the vertex representing the contracted set S or $V \setminus S$. Hence, each perfect matching M_i^j contains precisely one edge of $\delta(S)$. Furthermore, for $i \in \{1,2\}$, each edge $e \in \delta(S)$ must be contained in precisely $N \cdot y_i(e)$ matchings of the family $\{M_i^j\}_{j \in [N]}$ for (5.9) to be true. Additionally, since both y_1 and y_2 are just restrictions of y, they coincide on the edges $\delta(S)$. Hence, $y(e) = y_1(e) = y_2(e)$ for each $e \in \delta(S)$. Thus, for every $e \in \delta(S)$, there is the same number of perfect matchings in $\{M_1^j\}_{j \in [N]}$ that contain e as there are perfect matchings in

$\{M_2^j\}_{j\in[N]}$ containing e. We can therefore choose the numberings of those matchings, i.e., the indices j, such that

$$M_1^j \cap \delta(S) = M_2^j \cap \delta(S) \quad \forall j \in [N] \ .$$

This implies that $M_1^j \cup M_2^j$ is a perfect matching in G. Hence, we have

$$y = \frac{1}{N}\sum_{j=1}^{N} \chi^{M_1^j \cup M_2^j} \ ,$$

which implies that y is a convex combination of perfect matchings in G. This violates the fact that y is a fractional vertex of P, because we were able to write y as a convex combination of $\{0,1\}$-points of P, thus finding a non-trivial convex combination that expresses y. □

5.7.2 Matching polytope

As we show in the proof of the theorem below, the description of the perfect matching polytope can now be used to prove integrality of the matching poytope.

Theorem 5.22

The matching polytope of an undirected graph $G = (V, E)$ is given by

$$P = \left\{ x \in \mathbb{R}^E_{\geq 0} \;\middle|\; \begin{array}{ll} x(\delta(v)) \leq 1 & \forall v \in V \\ x(E[S]) \leq \frac{|S|-1}{2} & \forall S \subseteq V, |S| \text{ odd} \end{array} \right\} \ .$$

Proof. As usual, it is easy to check that P contains the correct set of integral points, i.e., $P \cap \{0,1\}^E$ are all incidence vectors of matchings in G. Let $x \in P$. We have to show that x is a convex combination of matchings. We use the following proof plan. We will define an auxiliary graph $H = (W, F)$ that extends G, and we will also extend x to a point in $y \in [0,1]^F$, which we can show to be in the perfect matching polytope of H using Theorem 5.21. Hence, y can be written as a convex combination of perfect matchings of H. From this convex decomposition we then derive that x is a convex combination of matchings of G.

We start by constructing the auxiliary graph $H = (W, F)$. Let $G' = (V', E')$ be a copy of $G = (V, E)$, i.e., for every vertex $v \in V$ there is a vertex $v' \in V'$ and for every edge $e = \{u, v\} \in E$ there is an edge $e' = \{u', v'\} \in E'$. We define $H = (W, F)$ to be the disjoint union of G and G', where we add all edges of the type $\{v, v'\}$ for $v \in V$. More formally,

$$\begin{aligned} W &= V \cup V' \ , \\ F &= E \cup E' \cup \{\{v, v'\} \colon v \in V\} \ . \end{aligned}$$

Furthermore, we define $y \in [0,1]^F$ by

$$\begin{aligned} y(e) &= x(e) & \forall e \in E \ , \\ y(e') &= x(e) & \forall e \in E \ , \\ y(\{v, v'\}) &= 1 - x(\delta(v)) & \forall v \in V \ . \end{aligned}$$

We now show that y is in the perfect matching polytope of H, which is described by Theorem 5.21. First, we clearly have $y \geq 0$ and $y(\delta_H(w)) = 1$ for $w \in W$. Let $Q \subseteq W$ be a set with $|Q|$ odd. It remains to show $y(\delta_H(Q)) \geq 1$ for y to be in the perfect matching polytope of H. Let $A = Q \cap V$, and $B' = Q \cap V'$. Furthermore, we define $B = \{v \in V \colon v' \in B'\}$. We have

$$y(\delta_H(Q)) = \underbrace{y(\delta_H(Q) \cap E)}_{=\text{I}} + \underbrace{y(\delta_H(Q) \cap E')}_{=\text{II}} + \underbrace{y(\delta_H(Q) \cap \{\{v,v'\} \colon v \in V\})}_{=\text{III}} \ .$$

Now observe that

$$\begin{aligned}
\text{I} &= x(\delta(A)) \ , \\
\text{II} &= x(\delta(B)) \ , \\
\text{III} &= \sum_{v \in A \setminus B} (1 - x(\delta(v))) + \sum_{v \in B \setminus A} (1 - x(\delta(v))) \\
&= |A \setminus B| - 2x(E[A \setminus B]) - x(\delta(A \setminus B)) + |B \setminus A| - 2x(E[B \setminus A]) - x(\delta(B \setminus A)) \ .
\end{aligned}$$

Furthermore,

$$\begin{aligned}
x(\delta(A)) + x(\delta(B)) &= x(\delta(A \setminus B)) + x(\delta(B \setminus A)) + 2x(E(A \cap B, V \setminus (A \cup B))) \\
&\geq x(\delta(A \setminus B)) + x(\delta(B \setminus A)) \ ,
\end{aligned}$$

where for $S_1, S_2 \subseteq V$, the set $E(S_1, S_2) \subseteq E$ consists of all edges in G with one endpoint in S_1 and the other in S_2. Combining the above, we obtain

$$y(\delta_H(Q)) \geq |A \setminus B| - 2x(E[A \setminus B]) + |B \setminus A| - 2x(E[B \setminus A]) \ . \tag{5.10}$$

Now, notice that for any set $S \subseteq V$, we have

$$2x(E[S]) \leq \sum_{v \in S} x(\delta(v)) \leq |S| \ , \tag{5.11}$$

where the first inequality follows by observing that every edge $\{u, v\} \in E[S]$ appears twice in the middle term: once in $\delta(u)$ and once in $\delta(v)$. Furthermore, the second inequality follows by $x(\delta(v)) \leq 1 \ \forall v \in V$, which holds since $x \in P$.

Notice that

$$|Q| = |A \setminus B| + |B \setminus A| + 2|A \cap B| \ .$$

Hence, since $|Q|$ is odd, either $|A \setminus B|$ or $|B \setminus A|$ must be odd. Without loss of generality assume that $|A \setminus B|$ is odd. As $x \in P$, we have

$$x(E[A \setminus B]) \leq \frac{|A \setminus B| - 1}{2} \ . \tag{5.12}$$

We finally obtain

$$\begin{aligned}
y(\delta_H(Q)) &\geq |A \setminus B| - 2x(E[A \setminus B]) + |B \setminus A| - 2x(E[B \setminus A]) && \text{(by (5.10))} \\
&\geq |A \setminus B| - 2x(E[A \setminus B]) && \text{(by (5.11))} \\
&\geq 1 && \text{(by (5.12))},
\end{aligned}$$

thus implying that y is in the perfect matching polytope of H. Hence, y can be written as a convex combination of perfect matchings in H, i.e.,

$$y = \sum_{i=1}^{k} \lambda_i \underbrace{\chi^{M_i}}_{\in \{0,1\}^F} \quad ,$$

where $k \in \mathbb{Z}_{>0}$, $\lambda_i \geq 0$ for $i \in [k]$, $\sum_{i=1}^{k} \lambda_i = 1$, and $M_i \subseteq F$ for $i \in [k]$ is a perfect matching in $H = (W, F)$. Since x is the restriction of y to the edges in E, we get

$$x = y|_E = \sum_{i=1}^{k} \lambda_i \underbrace{\chi^{M_i \cap E}}_{\in \{0,1\}^E} \quad ,$$

where we use the characteristic vector notation χ in the equation above with respect to the ground set E instead of F, i.e., $\chi^{M_i \cap E} \in \{0,1\}^E$. Thus, x is a convex combination of the matchings $M_1 \cap E, \dots, M_k \cap E$ in the graph G. This proves that P is contained in the matching polytope. Furthermore, since P contains the correct integral points, it contains the matching polytope. Hence, P is the matching polytope. □

5.8 Combinatorial uncrossing

Combinatorial uncrossing is a technique to extract a well-structured set system out of a large family of sets. It can be used in various contexts, one of which is to show the integrality of polyhedra. Here, the main goal of combinatorial uncrossing can be summarized as follows:

> Given a heavily overdetermined linear system that uniquely defines a point, find a well-structured full-rank subsystem.

5.8.1 Integrality of spanning tree polytope

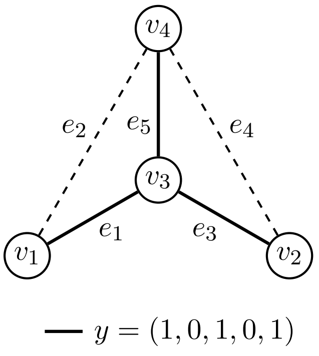
We recall the description of the spanning tree polytope claimed by Theorem 5.17.

$$P = \left\{ x \in \mathbb{R}^E_{\geq 0} \;\middle|\; \begin{array}{rl} x(E) = |V| - 1 & \\ x(E[S]) \leq |S| - 1 & \forall S \subsetneq V, |S| \geq 2 \end{array} \right\} \quad .$$

As it is common for many combinatorial optimization problems, the spanning tree polytope is highly degenerate. Figure 5.5 shows a degenerate vertex of the spanning tree polytope together with the tight constraints.

Notice that the constraint matrix corresponding to the spanning tree polytope is not TU. Moreover, even if we only consider tight constraints, the resulting linear subsystem may not be TU. A submatrix of the tight constraints having a determinant not within $\{-1, 0, 1\}$ is highlighted in blue in Figure 5.5.

However, it turns out that for any vertex y of the spanning tree polytope, there is always a full-rank linear subsystem among the y-tight constraints that is TU and thus implies integrality of y. We will prove the existence of such a subsystem using combinatorial uncrossing.



spanning tree constraints:

$$\begin{array}{cc} & \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{array}{r} \{v_1,v_2\} \\ \{v_1,v_3\} \\ \{v_1,v_4\} \\ \{v_2,v_3\} \\ \{v_2,v_4\} \\ \{v_3,v_4\} \\ \{v_1,v_2,v_3\} \\ \{v_1,v_2,v_4\} \\ \{v_1,v_3,v_4\} \\ \{v_2,v_3,v_4\} \\ \{v_1,v_2,v_3,v_4\} \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array} y \overset{=}{\leq} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \\ 3 \end{pmatrix}$$

non-negativity constraints:

$$\begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} y \overset{=}{\geq} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Figure 5.5: The vector y is the incidence vector of the spanning tree $\{e_1, e_3, e_5\}$. Tight spanning tree constraints (with respect to y) and tight non-negativity constraints are highlighted in red. There are 7 tight spanning tree constraints and 2 tight non-negativity constraints. Hence, the vertex y of the spanning tree polytope is degenerate because the total number of tight constraints, which is 9, is strictly larger then the dimension in which y lives, i.e., 5. Furthermore, the constraint matrix that corresponds to only the tight constraints is not TU. This can be verified by considering the subsystem of the tight spanning tree constraints that correspond to the columns e_1, e_3, e_5 and the rows $\{v_1, v_2, v_3\}, \{v_1, v_3, v_4\}, \{v_2, v_3, v_4\}$, as highlighted in blue in the above figure. The determinant of this 3×3 subsystem is -2.

Proof of integrality of P Let $y \in P$ be a vertex of P. Without loss of generality, we delete from $G = (V, E)$ all edges $e \in E$ with $y(e) = 0$, to obtain a smaller graph such that y restricted to $\{e \in E \mid y(e) > 0\}$ is still a vertex of the polytope P in the reduced graph. Hence, we assume from now on that G is this reduced graph and therefore $\mathrm{supp}(y) = E$.

Consider all y-tight, or simply tight, *spanning tree (ST) constraints*. Each tight ST constraint

$x(E[S]) = |S|-1$ corresponds to some set $S \subseteq V$, $|S| \geq 2$. We represent all tight ST constraints by their corresponding set S, thus obtaining the *family of tight sets* $\mathcal{F}$ given by

$$\mathcal{F} = \{S \subseteq V : |S| \geq 1,\ y(E[S]) = |S| - 1\} \ .$$

For technical reasons that will become clear later, we also include sets of size 1 in $\mathcal{F}$. Notice that they correspond to constraints that are trivially fulfilled.

Let $\mathcal{H} \subseteq \mathcal{F}$ be a maximal laminar subfamily of $\mathcal{F}$, and consider the system:

$$x(E[S]) = |S| - 1 \quad \forall S \in \mathcal{H} \ . \tag{5.13}$$

We will show that y is the unique solution to (5.13). Notice that this will imply integrality of y since (5.13) is a TU system with integral right-hand side, as it has the consecutive-ones property with respect to the rows.

As the set of all tight spanning tree constraints uniquely defines y, it suffices to show that any tight spanning tree constraint is implied by (5.13), i.e.,

$$\chi^{E[S]} \text{ is a linear combination of } \{\chi^{E[H]} \mid H \in \mathcal{H}\} \qquad \forall S \in \mathcal{F} \ . \tag{5.14}$$

We showed in the problem sets in a very general context that (5.14) is indeed equivalent to showing that every tight spanning tree constraint is implied by (5.13).

To this end, we need a better understanding of the structure of y-tight constraints, because we need to show that $\mathcal{H}$ contains a rich enough family of y-tight constraints such that (5.14) holds. The lemma below states an elementary yet crucial structural property that will allow us to make a concrete statement about the laminar family $\mathcal{H}$.

Lemma 5.23

For any sets $A, B \subseteq V$, we have

$$\chi^{E[A]} + \chi^{E[B]} + \chi^{E(A\setminus B, B\setminus A)} = \chi^{E[A\cup B]} + \chi^{E[A\cap B]} \ ,$$

which implies

$$\chi^{E[A]} + \chi^{E[B]} \leq \chi^{E[A\cup B]} + \chi^{E[A\cap B]} \ .$$

Proof. In words, the equality in the lemma states that any edge $e \in E$ appears the same number of times in the edge sets $E[A], E[B], E(A\setminus B, B\setminus A)$ as it appears in the sets $E[A\cup B], E[A\cap B]$. One can easily check that Lemma 5.23 holds by verifying the equality for each coordinate, i.e., each edge, where the edges can be grouped into the following *edge types*. The type of an edge is determined by where its endpoints lie among the four sets $A \setminus B, B \setminus A, A \cap B, E \setminus (A \cup B)$. Figure 5.6 shows the 10 different edge types. Clearly, two edges of the same edge type have the same contribution to the left-hand side and right-hand side of the equality in Lemma 5.23. Hence, it suffices to check whether each edge type has the same contribution on each side of the equality, which is easy to observe. □

Lemma 5.23 allows for deriving relations about tight sets. In particular, if two sets with non-empty intersection are tight, then so are their union and intersection.

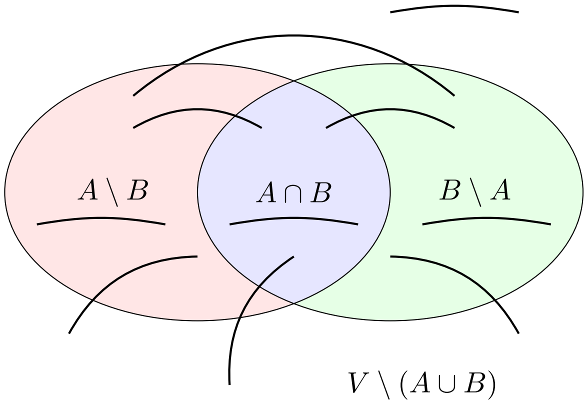


Figure 5.6: There are 10 types of edges in terms of where their endpoints lie with respect to the sets A and B, as shown above. One can easily check that each type appears precisely the same number of times in $E[A \cup B]$ and $E[A \cap B]$ as it appears in $E[A]$, $E[B]$, and $E(A \setminus B, B \setminus A)$.

Lemma 5.24

If $S_1, S_2 \in \mathcal{F}$ with $S_1 \cap S_2 \neq \emptyset$, then $S_1 \cap S_2, S_1 \cup S_2 \in \mathcal{F}$ and $E(S_1 \setminus S_2, S_2 \setminus S_1) = \emptyset$. In particular, this implies by Lemma 5.23

$$\chi^{E[S_1]} + \chi^{E[S_2]} = \chi^{E[S_1 \cup S_2]} + \chi^{E[S_1 \cap S_2]} \quad .$$

Proof. By Lemma 5.23 we have

$$y(E[S_1]) + y(E[S_2]) + y(E(S_1 \setminus S_2, S_2 \setminus S_1)) = y(E[S_1 \cup S_2]) + y(E[S_1 \cap S_2]) \quad . \tag{5.15}$$

Using (5.15), we obtain

$$\begin{aligned}
|S_1| + |S_2| - 2 &= \underbrace{|S_1 \cup S_2| - 1}_{\geq y(E[S_1 \cup S_2])} + \underbrace{|S_1 \cap S_2| - 1}_{\geq y(E[S_1 \cap S_2])} \\
&\geq y(E[S_1 \cup S_2]) + y(E[S_1 \cap S_2]) && \text{(using } y \in P) \\
&= \underbrace{y(E[S_1])}_{=|S_1|-1} + \underbrace{y(E[S_2])}_{=|S_2|-1} + y(E(S_1 \setminus S_2, S_2 \setminus S_1)) && \text{(by (5.15))} \\
&= |S_1| + |S_2| - 2 + \underbrace{y(E(S_1 \setminus S_2, S_2 \setminus S_1))}_{\geq 0} && \text{(because } S_1, S_2 \in \mathcal{F}) \\
&\geq |S_1| + |S_2| - 2 \quad .
\end{aligned}$$

Hence, all inequalities above must be satisfied with equality, thus implying $S_1 \cup S_2, S_1 \cap S_2 \in \mathcal{F}$ and $y(E(S_1 \setminus S_2, S_2 \setminus S_1)) = 0$. Finally, since $y > 0$, we have that $y(E(S_1 \setminus S_2, S_2 \setminus S_1)) = 0$ implies $E(S_1 \setminus S_2, S_2 \setminus S_1) = \emptyset$, as desired. □

Notice that for Lemma 5.24 we need $\mathcal{F}$ to contain singleton sets to cover the case $S_1, S_2 \in \mathcal{F}$ with $|S_1 \cap S_2| = 1$.

Definition 5.25: Intersecting sets

Two sets $S_1, S_2 \subseteq V$ are called *intersecting* if $S_1 \cap S_2 \neq \emptyset$, $S_1 \setminus S_2 \neq \emptyset$, and $S_2 \setminus S_1 \neq \emptyset$.

Notice that a family of sets is laminar if and only if no two of its sets are intersecting.

Lemma 5.26

The statement (5.14) holds and y is thus uniquely defined by the TU system (5.13). Therefore, y is integral.

Proof. Assume for the sake of contradiction that (5.14) does not hold. Let

$$Q = \mathrm{span}(\{\chi^{E[H]} : H \in \mathcal{H}\}) \subseteq \mathbb{R}^E \quad .$$

Assuming (5.14) not to hold is thus equivalent to the existence of $S \in \mathcal{F}$ with $\chi^{E[S]} \notin Q$. Among all tight spanning tree constraints that violate (5.14), let $S \in \mathcal{F}$ be one such that

$$\mathcal{H}_S = \{H \in \mathcal{H} : S \text{ and } H \text{ are intersecting}\}$$

has smallest size. Notice that $\mathcal{H}_S \neq \emptyset$, for otherwise we could have added S to $\mathcal{H}$ without destroying laminarity of $\mathcal{H}$, thus contradicting maximality of $\mathcal{H}$. Let $H \in \mathcal{H}_S$. By Lemma 5.24 we have $S \cap H, S \cup H \in \mathcal{F}$ and

$$\chi^{E[S]} + \chi^{E[H]} = \chi^{E[S\cap H]} + \chi^{E[S\cup H]} \quad .$$

Notice that $\chi^{E[S]} \notin Q$ and $\chi^{E[H]} \in Q$. Hence, at least one of $\chi^{E[S\cap H]}, \chi^{E[S\cup H]}$ is not in Q. However, we have

$$\begin{aligned} |\mathcal{H}_{S\cap H}| &< |\mathcal{H}_S| \quad \text{and} \\ |\mathcal{H}_{S\cup H}| &< |\mathcal{H}_S| \quad , \end{aligned}$$

because there is no set in $\mathcal{H}$ that is intersecting with any of the two sets $S \cap H$ or $S \cup H$ but not S. Furthermore, the set H is intersecting with S but not with $S \cap H$ or $S \cup H$. (See problem sets.) This contradicts the choice of S. □

5.8.2 Integrality of dominant of r-arborescence polytope

We recall the description of the dominant of the r-arborescence polytope:

$$P = \left\{x \in \mathbb{R}^A_{\geq 0} : x(\delta^-(S)) \geq 1 \; \forall S \subseteq V \setminus \{r\}, S \neq \emptyset\right\} \quad .$$

To show its integrality, we provide an analogous proof to the case of the spanning tree polytope.

Proof of integrality of P Let $y \in P$ be a vertex of P. As for the spanning tree case, we can assume that the underlying graph $G = (V, A)$ has no arcs $a \in A$ with $y(a) = 0$, since those arcs can be removed and the restriction of y to its support is still a vertex in the polyhedron P that corresponds to the reduced graph. We denote by $\mathcal{F} \subseteq 2^V$ the set of all y-tight constraints:

$$\mathcal{F} = \{S \subseteq V \setminus \{r\} : y(\delta^-(S)) = 1\} \ .$$

Hence, y is the unique solution to the linear system

$$x(\delta^-(S)) = 1 \quad \forall S \in \mathcal{F} \ . \tag{5.16}$$

Let $\mathcal{H} \subseteq \mathcal{F}$ be a maximal laminar subfamily of $\mathcal{F}$. As in the spanning tree case, we show that the following system implies all constraints of (5.16):

$$x(\delta^-(S)) = 1 \quad \forall S \in \mathcal{H} \ . \tag{5.17}$$

Again, the linear system (5.17) is TU, which can be seen by applying the Ghouila-Houri criterion with respect to the rows as follows. Consider any subset of the rows, which can be represented by a laminar subfamily $\mathcal{F}' \subseteq \mathcal{H}$. We partition this subfamily $\mathcal{F}'$ into a '$+$' and '$-$' group as follows. The topmost sets of $\mathcal{F}'$, i.e., the ones not contained in any other set of $\mathcal{F}'$ are in the '$+$' group, their children are in the '$-$' group, and we continue alternating that way (see Figure 5.7). One can easily verify that this partition leads to a vector with entries $\{-1, 0, 1\}$ as desired. TU-ness of (5.17) also follows from results shown in the problem sets, where we considered a system $x(\delta^+(S))$ for S being part of an arbitrary laminar family. The system (5.17) easily reduces to this case by reversing the directions of the arcs.

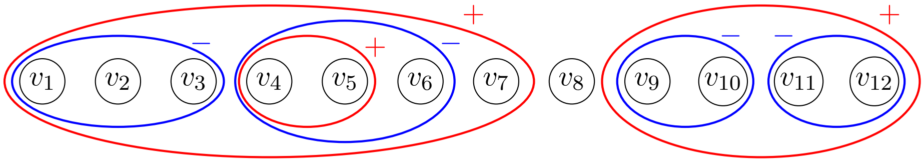


Figure 5.7: Illustration of how we apply the Ghouila-Houri criterion to the subfamily $\mathcal{F}'$, which is the laminar family depicted in the figure.

Hence, it remains to show that every constraint of (5.16) is implied by (5.17). Similar to the spanning tree case, we need to get a better understanding of directed cuts to deduce that $\mathcal{H}$ is a family rich enough to ensure that (5.17) implies (5.16). For this, we start with a basic property on directed cuts.

Lemma 5.27

For any two sets $S_1, S_2 \subseteq V$, we have

$$\chi^{\delta^-(S_1)} + \chi^{\delta^-(S_2)} = \chi^{\delta^-(S_1 \cap S_2)} + \chi^{\delta^-(S_1 \cup S_2)} + \chi^{A(S_1 \setminus S_2, S_2 \setminus S_1)} + \chi^{A(S_2 \setminus S_1, S_1 \setminus S_2)} \ ,$$

which implies in particular

$$\chi^{\delta^-(S_1)} + \chi^{\delta^-(S_2)} \geq \chi^{\delta^-(S_1 \cap S_2)} + \chi^{\delta^-(S_1 \cup S_2)} \ .$$

Proof. This proof can be done analogously to the proof of Lemma 5.23, by considering all different arc types. □

From Lemma 5.27, we can derive properties on tight cuts.

Lemma 5.28

If $S_1, S_2 \in \mathcal{F}$ with $S_1 \cap S_2 \neq \emptyset$, then $S_1 \cup S_2, S_1 \cap S_2 \in \mathcal{F}$ and $A(S_1 \setminus S_2, S_2 \setminus S_1) = \emptyset$, $A(S_2 \setminus S_1, S_1 \setminus S_2) = \emptyset$. In particular, this implies by Lemma 5.27

$$\chi^{\delta^-(S_1)} + \chi^{\delta^-(S_2)} = \chi^{\delta^-(S_1 \cup S_2)} + \chi^{\delta^-(S_1 \cap S_2)} \ .$$

Proof. By Lemma 5.27 and non-negativity of y we have

$$\begin{aligned} &y(\delta^-(S_1)) + y(\delta^-(S_2)) = \\ &\quad y(\delta^-(S_1 \cup S_2)) + y(\delta^-(S_1 \cap S_2)) + y(A(S_1 \setminus S_2, S_2 \setminus S_1)) + y(A(S_2 \setminus S_1, S_1 \setminus S_2)) \ . \end{aligned} \tag{5.18}$$

Using (5.18), we obtain

$$\begin{aligned} 2 &= y(\delta^-(S_1)) + y(\delta^-(S_2)) \\ &= y(\delta^-(S_1 \cup S_2)) + y(\delta^-(S_1 \cap S_2)) && \text{(by (5.18))} \\ &\qquad + \underbrace{y(A(S_1 \setminus S_2, S_2 \setminus S_1))}_{\geq 0} + \underbrace{y(A(S_2 \setminus S_1, S_1 \setminus S_2))}_{\geq 0} \\ &\geq \underbrace{y(\delta^-(S_1 \cup S_2))}_{\geq 1} + \underbrace{y(\delta^-(S_1 \cap S_2))}_{\geq 1} \\ &\geq 2 \ . && \text{(because } y \in P) \end{aligned}$$

Hence, all inequalities above must be satisfied with equality, thus implying $S_1 \cup S_2, S_1 \cap S_2 \in \mathcal{F}$ and $y(A(S_1 \setminus S_2, S_2 \setminus S_1)) = y(A(S_2 \setminus S_1, S_1 \setminus S_2)) = 0$. Finally, because $y > 0$, we have $A(S_1 \setminus S_2, S_2 \setminus S_1) = A(S_2 \setminus S_1, S_1 \setminus S_2) = \emptyset$, as desired. □

We are now ready to show that every constraint of (5.16) is implied by (5.17). Again, for sake of contradiction assume that this is not the case. Let $S \in \mathcal{F}$ be a set such that the constraint

$x(\delta^-(S)) = 1$ is not implied by (5.17), and among all such sets we choose one such that the number of sets in $\mathcal{H}$ that are intersecting with S is minimum. We define

$$Q = \operatorname{span}(\{\chi^{\delta^-(H)} : H \in \mathcal{H}\}) \ .$$

Hence, $S \in \mathcal{F}$ satisfies $\chi^{\delta^-(S)} \notin Q$. Let $H \in \mathcal{H}$ be a set such that S and H are intersecting. By Lemma 5.28 we have

$$\underbrace{\chi^{\delta^-(S)}}_{\notin Q} + \underbrace{\chi^{\delta^-(H)}}_{\in Q} = \chi^{\delta^-(S\cup H)} + \chi^{\delta^-(S\cap H)} \ .$$

Hence, at least one of $\chi^{\delta^-(S\cup H)}$ and $\chi^{\delta^-(S\cap H)}$ is not in Q. Since both $S \cup H$ and $S \cap H$ are intersecting with a strictly smaller number of sets in $\mathcal{H}$ than S, this contradicts the choice of S and finishes the proof, implying that each constraint of (5.16) is implied by (5.17).

5.8.3 Upper bound on number of edges of minimally k-edge-connected graphs

So far, we used combinatorial uncrossing to prove that certain polyhedra are integral. More precisely, we had inequality descriptions of the polyhedra where constraints where defined by sets, and we considered a maximal laminar family of sets that corresponded to tight constraints. Laminarity of the set system allowed us to show that the corresponding constraint matrix is TU, and combinatorial uncrossing arguments were employed to argue that they are a full-rank system. We now consider a quite different setting, not immediately related to polyhedra, where combinatorial uncrossing can be applied. This highlights nicely the versatility of combinatorial uncrossing, which goes far beyond the study of polyhedra.

Let $G = (V, E)$ be an undirected graph on n vertices. We recall that G is k-edge-connected for $k \in \mathbb{Z}_{\geq 0}$ if there are k edge-disjoint paths between any pair of vertices. By Menger's Theorem—a special case of the max-flow min-cut theorem—this is equivalent to the property that every cut $S \subsetneq V, S \neq \emptyset$ has size at least k, i.e., $|\delta(S)| \geq k$. A k-edge-connected graph G is called *minimally* k-edge-connected if removing any edge from G leads to a graph that is not k-edge-connected anymore. Hence, a graph is minimally k-edge-connected if and only if each edge is in a cut of size k, which is the size of a minimum cut because G is k-edge-connected. We are interested in determining the maximum number of edges that a minimally k-edge-connected graph on n vertices can have.

Figure 5.8 shows an example of a minimally k-edge-connected graph with $k \cdot (n-1)$ edges. We will show that this number is tight using combinatorial uncrossing. The notion of *crossing sets* is key in the proof.

Definition 5.29: Crossing sets

Two sets $S_1, S_2 \subseteq V$ are called *crossing* if none of the following sets is empty: $S_1 \cap S_2$, $S_1 \setminus S_2$, $S_2 \setminus S_1$, and $V \setminus (S_1 \cup S_2)$.

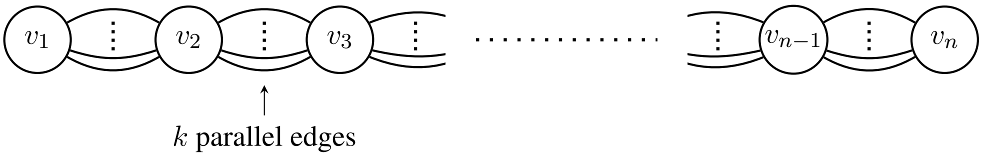


Figure 5.8: Example of a minimally k-edge-connected graph with n vertices and $k\cdot(n-1)$ edges.

In other words, two sets $S_1, S_2 \subseteq V$ are crossing if they are intersecting and $S_1 \cup S_2 \neq V$. One key property of crossing sets is that their intersection and union can be interpreted as cuts. (We recall that $S \subseteq V$ is a cut if $\emptyset \neq S \neq V$.)

Using combinatorial uncrossing, we now show a tight upper bound on the number of edges of a minimally k-edge-connected graph.

Theorem 5.30

Let $G = (V, E)$ be a minimally k-edge-connected graph. Then $|E| \leq k \cdot (|V| - 1)$.

Proof. We call a family $\mathcal{H} \subseteq 2^V$ of minimum cuts in G a *certifying family* if each edge is in at least one of them, i.e., for every $e \in E$ there is a cut $S \in \mathcal{H}$ such that $e \in \delta(S)$ and for each $S \in \mathcal{H}$ we have $|\delta(S)| = k$. We choose the name certifying family because $\mathcal{H}$ certifies that the graph is minimally k-edge-connected. Our goal is to construct a small certifying family $\mathcal{H}$ of size $|\mathcal{H}| \leq n - 1$, where $n := |V|$. Notice that the existence of such a small certifying family proves the lemma since $E \subseteq \bigcup_{S\in\mathcal{H}} \delta(S)$, and thus $|E| \leq (n-1)\cdot k$, because all cuts in $\mathcal{H}$ have size k since they are minimum.

Notice that for each cut $S \subseteq V$, also the set $V \setminus S$ is a cut containing the same edges. We can therefore choose an arbitrary vertex $r \in V$ and focus only on minimum cuts not containing r. An advantage of doing so is that whenever two cuts not containing r are intersecting, then they are also crossing.

To obtain $\mathcal{H}$, we start with an arbitrary certifying family $\mathcal{F} \subseteq 2^V$, which can be obtained as follows. For each $e \in E$, let

$$\mathcal{C}_e = \{S \subseteq V \setminus \{r\} : e \in \delta(S), |\delta(S)| = k\} \ .$$

By minimality of G we have $\mathcal{C}_e \neq \emptyset$ for every $e \in E$; for otherwise, e could be removed without destroying k-edge-connectivity. For each $e \in E$, let $S_e \in \mathcal{C}_e$ be an arbitrary cut in $\mathcal{C}_e$. We define

$$\mathcal{F} = \{S_e : e \in E\} \ .$$

Hence, $\mathcal{F}$ may contain up to $|E|$ sets, but possibly fewer, since we may have $S_e = S_{e'}$ for two different edges $e, e' \in E$.

In the same way we proved Lemma 5.28 one can show that for any two minimum cuts S_1, S_2 in G that are crossing, also $S_1 \cup S_2$ and $S_1 \cap S_2$ are minimum cuts, and furthermore $E(S_1 \setminus S_2, S_2 \setminus S_1) = \emptyset$. Since all cuts in $\mathcal{F}$ are minimum, we can apply this reasoning to any two cuts in $\mathcal{F}$. We first want to turn $\mathcal{F}$ into a laminar certifying family. Assume that $\mathcal{F}$ is not laminar. Hence, there are two intersecting cuts $S_1, S_2 \in \mathcal{F}$, which are also crossing because $r \notin S_1 \cup S_2$.

Let

$$\mathcal{F}' = (\mathcal{F} \setminus \{S_1, S_2\}) \cup \{S_1 \cup S_2, S_1 \cap S_2\} \ .$$

Thus, $\mathcal{F}'$ is still a family of minimum cuts. Furthermore, since $E(S_1 \setminus S_2, S_2 \setminus S_1) = \emptyset$, we have

$$\delta(S_1) \cup \delta(S_2) = \delta(S_1 \cup S_2) \cup \delta(S_1 \cap S_2) \ .$$

Hence, $\mathcal{F}'$ remains a certifying family. We claim that if we repeat this uncrossing procedure as long as there are at least two intersecting cuts in our certifying family, we will eventually end up with a laminar family of cuts. To see that this uncrossing procedure will stop, we introduce a potential function ϕ for certifying families:

$$\phi(\mathcal{F}) = \sum_{S \in \mathcal{F}} |S| \cdot |\overline{S}| \ ,$$

where $\overline{S} = V \setminus S$. Notice that for any two intersecting sets $S_1, S_2 \in \mathcal{F}$, we have

$$|S_1||\overline{S_1}| + |S_2||\overline{S_2}| > |S_1 \cup S_2||\overline{S_1 \cup S_2}| + |S_1 \cap S_2||\overline{S_1 \cap S_2}| \ . \tag{5.19}$$

Hence, whenever we do uncrossing to go from a certifying family $\mathcal{F}$ to another one $\mathcal{F}'$, we have $\phi(\mathcal{F}) > \phi(\mathcal{F}')$. Notice that the potential may even decrease more than by the slack in the inequality (5.19), because the uncrossing may lead to sets that already exist in the family, in which case also the size of the family decreases. In any case, the potential strictly decreases after each uncrossing step. Since the potential is by definition non-negative for any certifying family and decreases by at least one unit at each step, the uncrossing procedure must stop. Hence, there are no two intersecting sets in the final certifying family $\mathcal{L}$, and thus, $\mathcal{L}$ is laminar. Recall from the problem sets that a laminar family (not containing the empty set) on a ground set of size n may still have up to $2n-1$ sets. Notice, however, that since $\mathcal{L}$ is a laminar family over the set $V \setminus \{r\}$, which has size $n-1$, we actually have $|\mathcal{L}| \leq 2n-3$. We further purge the family $\mathcal{L}$ to obtain $\mathcal{H}$ as follows. As long as there is a set $L \in \mathcal{L}$ such that the children $L_1, \dots, L_q$ of L in the family $\mathcal{L}$ form a partition of L, then we remove the set L from $\mathcal{L}$. We call this constellation of sets L, $L_1, \dots, L_q$ an *obstruction*. The resulting family will still be a certifying family since any edge contained in $\delta(L)$ is also contained in precisely one of the cuts $L_1, \dots, L_q$, i.e., $\delta(L) \subseteq \bigcup_{i=1}^{q} \delta(L_i)$. Let $\mathcal{H}$ be the resulting family, which does not contain any obstructions anymore. Notice that $\mathcal{H}$ is a laminar family over $V \setminus \{r\}$. Every laminar family $\mathcal{H}'$ on $n-1$ elements without obstructions and not containing the empty set satisfies $|\mathcal{H}'| \leq n-1$, since for each set $H' \in \mathcal{H}'$, there is an element of the ground set such that H' is the smallest set containing this element. Thus, $\mathcal{H}$ can have at most as many sets as there are elements in $V \setminus \{r\}$, which implies $|\mathcal{H}| \leq n-1$ and completes the proof. □

6 Ellipsoid Method

The Ellipsoid Method is a procedure that can be used to solve in particular linear and, more generally, convex optimization problems. We will focus on linear optimization problems with a bounded feasible region, i.e., the feasible region is a polytope P, as this covers the most important use cases of the Ellipsoid Method in the context of Combinatorial Optimization.

A crucial advantage of the Ellipsoid Method compared to other approaches for linear programming, like the Simplex Method or interior-point methods, is that one does not need to process an explicit inequality description of P to solve the linear program. More precisely, the way the Ellipsoid Method gains information about P is through a method that solves an arguably simpler sub-problem, known as the *separation problem*.

6.1 Separation problem

The separation problem for a polyhedron $P \subseteq \mathbb{R}^n$ is defined as follows.

Definition 6.1: Separation problem & separation oracle

Given a point $y \in \mathbb{R}^n$:

- Decide whether $y \in P$, and if this is not the case,
- find $c \in \mathbb{R}^n$ such that $P \subseteq \{x \in \mathbb{R}^n \colon c^\top x < c^\top y\}$.

A procedure that solves the separation problem (for P) is often called a *separation oracle* (for P).

The second point in the above definition, where a vector $c \in \mathbb{R}^n$ has to be returned, is often called *separating y from p.*

In the context of the Ellipsoid Method, we call a hyperplane $H = \{x \in \mathbb{R}^n \colon c^\top x = \alpha\}$ a *separating hyperplane*, or more precisely a *y-separating hyperplane*, if the following holds: Let

$$\begin{aligned} H_{\leq} &:= \{x \in \mathbb{R}^n : c^\top x \leq \alpha\}, \text{ and} \\ H_{\geq} &:= \{x \in \mathbb{R}^n : c^\top x \geq \alpha\} \end{aligned}$$

be the two closed half-spaces defined by H; then H is *separating* if P is contained in one of the above half-spaces, y in the other one, and H does not simultaneously contain y and is touching P, i.e., either $y \notin H$ or $P \cap H = \emptyset$. Hence, if $H = \{x \in \mathbb{R}^n : c^\top x = \alpha\}$ is a separating hyperplane, then c is a vector that solves the separation problem for y. Vice-versa, if $c \in \mathbb{R}^n$ is a vector solving the separation problem for y, then there exists a separating hyperplane with normal vector c, for example, $H = \{x \in \mathbb{R}^n : c^\top x \leq c^T y\}$.

The fact that the Ellipsoid Method is based on a separation oracle and does not need to process an explicit inequality description of P, allows it to solve in polynomial time many linear programs with an exponential number of constraints. For example, one can use the Ellipsoid Method to optimize any linear function over the matching polytope, and thus, in particular, one can efficiently compute a maximum weight matching via the Ellipsoid Method. The reason for this is that even though the matching polytope has an exponential number of constraints, one can construct a polynomial time separation oracle for it.

6.2 Optimization results based on Ellipsoid Method

Before formally introducing the Ellipsoid Method, we first highlight what kind of results can be obtained through it. The Ellipsoid Method is a very general and versatile technique, and a thorough coverage of its implications is beyond the scope of this course. We therefore restrict ourselves to a statement about optimizing over $\{0,1\}$-polytopes, which is of particular interest in the context of Combinatorial Optimization, where we often deal with $\{0,1\}$-polytopes.

Theorem 6.2

Let $P \subseteq \mathbb{R}^n$ be a $\{0,1\}$-polytope for which we are given a separation oracle. Furthermore, let $w \in \mathbb{Z}^n$. Then the Ellipsoid Method allows for finding an optimal vertex solution to the linear program $\max\{w^\top x \colon x \in P\}$ using a polynomial number (in n) of elementary operations and calls to the separation oracle for P.

Hence, Theorem 6.2 implies that an efficient separation oracle for a $\{0,1\}$-polytope P is all we need to efficiently solve any linear program over P.

6.3 Example applications

We showcase the construction of a separation oracle on the example of the dominant of the r-arborescence polytope, which has an exponential number of facets. Hence, the Simplex Method, as we have discussed it, is not suitable for such a problem, because even just reading in the constraints to construct the simplex tableau would take exponential time, before we even start to do any computations.

In the problem sets, we will see further applications of how the Ellipsoid Method can be used to solve non-trivial problems through the construction of an appropriate separation oracle.

Minimum weight r-arborescence

Consider the problem of finding a minimum weight r-arborescence in a directed graph $G = (V, A)$ with non-negative arc weights $w \colon A \to \mathbb{Z}_{\geq 0}$. Clearly, this problem is equivalent to minimizing the linear function w over the dominant P of the r-arborescence polytope which, by Theorem 5.20, is described by

$$P = \left\{x \in \mathbb{R}^A_{\geq 0} \colon x(\delta^-(S)) \geq 1 \quad \forall S \subseteq V \setminus \{r\}, S \neq \emptyset \ \right\} \ .$$

Strictly seen, we cannot apply Theorem 6.2 to P, because P is not a $\{0,1\}$-polytope. More precisely, although P is integral, it is unbounded. This can easily be remedied by intersecting P with the hypercube to obtain

$$Q = P \cap [0,1]^A \enspace .$$

In the problem sets we show that the intersection of the dominant of any $\{0,1\}$-polytope with the unit hypercube results in a $\{0,1\}$-polytope. Hence, Q is a $\{0,1\}$-polytope. Moreover, because P is the dominant of a $\{0,1\}$-polytope and we are minimizing a non-negative linear function over it, there is an optimal solution x^* to $\min\{w^\top x \colon x \in P\}$ with $x^* \in \{0,1\}^n$. Hence, $x^* \in Q$. Thus, a minimum r-arborescence can be found by finding a vertex solution to

$$\min\{w^\top x \colon x \in Q\} \enspace . \tag{6.1}$$

By Theorem 6.2, it suffices to find a polynomial-time separation oracle for Q to find an optimal vertex solution to (6.1). It therefore remains to design a polynomial-time separation oracle for Q. Here we can exploit the simple description of P provided by Theorem 5.20.

So, let $y \in \mathbb{R}^A$. We first check whether $y \in [0,1]^A$. This is easy to check since the unit hypercube is defined by $2|A|$ constraints. If one of these constraints is violated, then it immediately leads to a separating hyperplane. Hence, assume $y \in [0,1]^A$. It remains to check whether one of the constraints $x(\delta^-(S)) \geq 1$ for some $S \subseteq V \setminus \{r\}, S \neq \emptyset$ is violated. This can be checked via minimum s-t cut computations. For each $v \in V \setminus \{r\}$, we compute a minimum r-v cut $S_v \subseteq V$, where we use y as the capacities on the arcs. If one of these cuts S_v has a value strictly less than 1, then the constraint corresponding to $V \setminus S_v$ is violated and leads to a separating hyperplane. Otherwise, if $y(\delta^+(S_v)) \geq 1$ for all $v \in V$, then y satisfies all constraints of P and therefore $y \in Q$, because of the following. Assume that there was a violated constraint, i.e., $y(\delta^-(S)) < 1$ for some $S \subseteq V \setminus \{r\}, S \neq \emptyset$. Then, for any $v \in S$ we have

$$y(\delta^+(S_v)) \leq y(\delta^+(V \setminus S)) = y(\delta^-(S)) < 1 \enspace ,$$

where the first inequality follows from the fact that $V \setminus S$ is an r-v cut, and its cut value is thus at least as large as the value of the cut S_v, which is by definition a minimum r-v cut.

This shows that we can solve the separation problem over Q, and thus, by Theorem 6.2, we can find an optimal vertex solution to (6.1) in polynomial time through the Ellipsoid Method. Furthermore, as discussed above, such a vertex solution corresponds to a minimum weight r-arborescence.

6.4 Ellipsoid Method for finding point in full-dimensional $\{0,1\}$-polytope

Our final goal is to show how to do linear optimization via the Ellipsoid Method. However, to approach this goal, we start with the following simpler problem, which will lead us to linear optimization later on.

Given a full-dimensional polytope $P \subseteq \mathbb{R}^n$, find a point $x \in P$. (6.2)

As we will discuss later, a linear programming problem over some (full-dimensional) polytope Q, i.e.,

$$\begin{array}{rl} \max & w^\top x \\ & x \in Q \ , \end{array} \tag{6.3}$$

can be reduced to finding points in $P_b = Q \cap \{w^\top x \geq b\}$ for varying right-hand sides $b \in \mathbb{R}$. For the time being, we do not assume that P is a $\{0,1\}$-polytope. We will add further conditions later on when we need them.

As the name suggests, the Ellipsoid Method works with ellipsoids. To be precise, additional to a separation oracle for P, it needs an ellipsoid E_0 that contains P, which is often called the *starting ellipsoid*. (In turns out that in case of a $\{0,1\}$-polytope P, one can choose a trivial ellipsoid, namely the smallest ball containing $[0,1]^n$. This is the reason why Theorem 6.2 does not require a starting ellipsoid to be given.)

We recall that an ellipsoid is the image of the unit ball under an affine bijection, which can be defined as follows.

Definition 6.3: Ellipsoid

An ellipsoid in $\mathbb{R}^n$ is a set

$$E(a, A) := \{x \in \mathbb{R}^n \colon (x-a)^\top A^{-1}(x-a) \leq 1\} \ ,$$

where $a \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ is a positive definite matrix. The point a is called the *center* of the ellipsoid $E(a, A)$.[a]

[a] We use the common convention that a positive definite matrix is by definition symmetric.

In particular, an ellipsoid is always full-dimensional. Notice that the above definition of ellipsoid indeed corresponds to the image of the unit ball under an affine bijection. This can be seen as follows. A matrix $A \in \mathbb{R}^{n\times n}$ is positive definite if and only if there is a full-rank matrix $Q \in \mathbb{R}^{n \times n}$ such that $A = QQ^\top$. Hence, $A^{-1} = (Q^\top)^{-1}(Q^{-1}) = (Q^{-1})^\top Q^{-1}$, and therefore

$$\begin{aligned} E(a, A) &= \{x \in \mathbb{R}^n \colon \|Q^{-1}(x-a)\|_2 \leq 1\} \\ &= \{y + a \colon y \in \mathbb{R}^n, \|Q^{-1}y\|_2 \leq 1\} && \text{(Substitution with } y = x - a.\text{)} \\ &= \{Qz + a \colon z \in \mathbb{R}^n, \|z\|_2 \leq 1\} \ . && \text{(Substitution with } z = Q^{-1}y.\text{)} \end{aligned}$$

Figure 6.1 shows an example of an ellipsoid for $n = 2$.

The separation oracle for P and the ellipsoid $E_0 \supseteq P$ is all that is needed to run the Ellipsoid Method and obtain a point $x \in P$. However, to make sure that the Ellipsoid Method runs in polynomial, time we need a further condition. One sufficient condition for the Ellipsoid Method to run in polynomial time is that $\log\left({}^{\mathrm{vol}(E_0)}\!/_{\mathrm{vol}(P)}\right)$ is polynomially bounded by the input. We will see later that this holds for most cases we are interested in.

6.4.1 Description of Ellipsoid Method

The Ellipsoid Method to solve (6.2) is described in Algorithm 8.

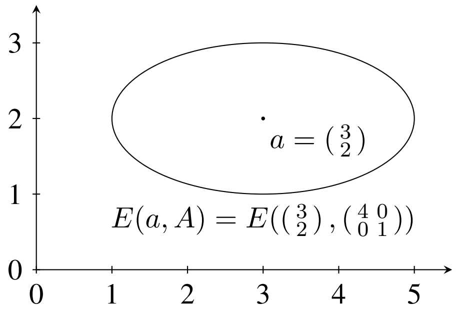

$a = \binom{3}{2}$
$E(a,A) = E(\binom{3}{2}, \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix})$
0
1
2
3
0
1
2
3
4
5

Figure 6.1: An example of an axis-parallel ellipsoid $E(a,A)$ in two dimensions. Notice that the eigenvectors of A correspond to the axes of the ellipsoid, and the square roots of the eigenvalues correspond to the radii of the corresponding axes.

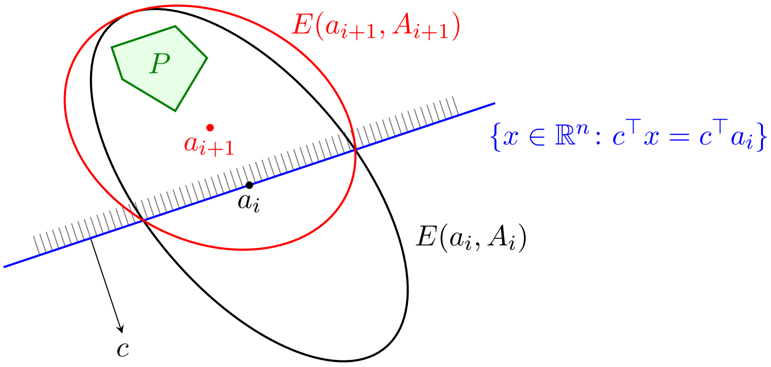


Figure 6.2: Illustration of a single iteration of the Ellipsoid Method. The polytope P is inside each ellipsoid considered by the Ellipsoid Method.

Algorithm 8: Ellipsoid Method

```
Input  : Separation oracle for a polytope $P \subseteq \mathbb{R}^n$ with $\dim(P) = n$, and an ellipsoid
         $E_0 = E(a_0, A_0)$ with $P \subseteq E_0$.
Output: A point $y \in P$.
$i = 0$.
while $a_i \notin P$ (checked with separation oracle) do
    Get $c \in \mathbb{R}^n$ such that $P \subseteq \{x \in \mathbb{R}^n : c^\top x < c^\top a_i\}$, using separation oracle.
    Find min. volume ellipsoid $E_{i+1} = E(a_{i+1}, A_{i+1})$ containing $E_i \cap \{x \in \mathbb{R}^n : c^\top x \leq c^\top a_i\}$.
    $i = i + 1$.
return $a_i$.
```

We will soon give some more details on how to compute E_{i+1}. It turns out that there is a relatively simple way to describe E_{i+1} in terms of E_i and c. Unfortunately, as we will see soon, this description of E_{i+1} involves taking a square root, which is an operation that we cannot perform up to arbitrary precision under the typical computational assumptions. Hence, in practice, one only computes an ellipsoid E'_{i+1} that approximates E_{i+1} and has a polynomial-size description. To simplify the exposition we will not go into these numerical details and assume that we use an exact description of E_{i+1}.

Figure 6.2 illustrates one iteration of the Ellipsoid Method. Notice that we have $P \subseteq E_i$ for each ellipsoid considered in the Ellipsoid Method. This can easily be verified by induction. The first ellipsoid E_0 contains P by assumption. Furthermore, for each iteration i, $P \subseteq \{x \in \mathbb{R}^n \colon c^\top x \leq c^\top a_i\}$, and hence $P \subseteq E_i \cap \{x \in \mathbb{R}^n \colon c^\top x \leq c^\top a_i\}$, as $P \subseteq E_i$. We thus obtain $P \subseteq E_{i+1}$ because $E_i \cap \{x \in \mathbb{R}^n \colon c^\top x \leq c^\top a_i\} \subseteq E_{i+1}$ by definition of E_{i+1}.

6.4.2 Getting a bound on the number of iterations

The key property of the constructed ellipsoids, besides the fact that they contain P, is that they shrink in terms of volume. More precisely, we have the following.

Lemma 6.4

$$\frac{\mathrm{vol}(E_{i+1})}{\mathrm{vol}(E_i)} < e^{-\frac{1}{2(n+1)}} \quad .$$

Before proving Lemma 6.4, we observe that it immediately implies an upper bound on the number of iterations that the Ellipsoid Method performs.

Lemma 6.5

The Ellipsoid Method will stop after at most $2(n+1)\ln\left(\frac{\mathrm{vol}(E_0)}{\mathrm{vol}(P)}\right)$ iterations.

Proof. Let $L \in \mathbb{Z}_{\geq 0}$ be the last iteration of the Ellipsoid Method, i.e., the value of the variable i in Algorithm 8 when it terminates. Since E_L contains P, we must have $\mathrm{vol}(P) \leq \mathrm{vol}(E_L)$, which, combined with Lemma 6.4, leads to

$$\mathrm{vol}(P) \leq \mathrm{vol}(E_L) \leq \mathrm{vol}(E_0) e^{-\frac{L}{2(n+1)}} \quad ,$$

and thus

$$L \leq 2(n+1)\ln\left(\frac{\mathrm{vol}(E_0)}{\mathrm{vol}(P)}\right) \quad . \qquad \square$$

In what follows, we will prove Lemma 6.4 and give an explicit description of the ellipsoid E_{i+1} based on c and $E_i = E(a_i, A_i)$. Later, we will show how the bound on the number of iterations given by Lemma 6.5 can be used to prove that the Ellipsoid Method runs in polynomial time for any full-dimensional $\{0,1\}$-polytope. To this end, we will first make the link between checking feasibility of a polytope and optimizing an LP more explicit.

Proof of Lemma 6.4 and explicit description for E_{i+1}

A key simplification for proving Lemma 6.4 is to observe that it suffices to consider the special case where $E_i = E(0, I)$ is the unit ball and $H_i = \{x \in \mathbb{R}^n \colon x_1 \geq 0\}$. As we show next, one can reduce to this case by an appropriate affine transformation.

Lemma 6.6

Let $E_i = E(a_i, A_i)$ be an ellipsoid and $H_i = \{x \in \mathbb{R}^n \colon c^\top x \leq c^\top a_i\}$ with $c^\top A_i c = 1$ (this property of c can be achieved without loss of generality by scaling c). Let $H_B = \{x \in \mathbb{R}^n \colon x_1 \geq 0\}$, and let E_B be a minimum volume ellipsoid containing $E(0,I) \cap H_B$. Let $\rho \colon \mathbb{R}^n \to \mathbb{R}^n$ be an affine bijection defined as follows:

$$\rho(x) = Q_i R x + a_i \enspace ,$$

where $Q_i \in \mathbb{R}^{n\times n}$ is any matrix such that $A_i = Q_i Q_i^\top$, and R is any orthogonal matrix satisfying $R^\top Q_i^\top c = -e_1$, where $e_1 = (1,0,\dots,0) \in \{0,1\}^n$ is the vector with a single one in the first coordinate and zeros everywhere else.
Then, $\rho(E(0,I)) = E_i$, $\rho(H_B) = H_i$, and $\rho(E_B)$ is a minimum volume ellipsoid containing $E_i \cap H_i$.

Hence, Lemma 6.6 shows that to prove Lemma 6.4, it suffices to consider the unit ball case cut by the hyperplane H_B. Indeed, the ratio between a minimum volume ellipsoid E_{i+1} containing $E_i \cap H_i$, and $\mathrm{vol}(E_i) = \mathrm{vol}(\rho(E(0,I)))$ is equal to

$$\frac{\mathrm{vol}(E_{i+1})}{\mathrm{vol}(E_i)} = \frac{\mathrm{vol}(\rho(E_B))}{\mathrm{vol}(\rho(E(0,I)))} = \frac{\mathrm{vol}(E_B)}{\mathrm{vol}(E(0,I))} \enspace , \tag{6.4}$$

where the second equality follows by the fact that ρ scales the volumes of all measurable sets by the same factor.

Also, the lemma provides an explicit transformation ρ that allows for transforming a minimum volume ellipsoid for this special case into a minimum volume ellipsoid for the general case. Later, we will use this transformation to give an explicit description of the center a_{i+1} and defining matrix A_{i+1} for the next ellipsoid $E_{i+1} = E(a_{i+1}, A_{i+1})$ in the Ellipsoid Method.

Notice that $c^\top A_i c = 1$ implies $\|Q_i^\top c\|_2 = 1$, and hence, there is an orthogonal matrix R that maps $Q_i^\top c$ to $-e_1$. (Because an orthogonal transformation does not change lengths, $\|Q_i^\top c\|_2 = 1$ is necessary for R to exist.)

Proof of Lemma 6.6. The proof plan is as follows. We start by observing that ρ is a bijection, $\rho(E(0,I)) = E_i$, and $\rho(H_i) = H_B$. Furthermore, applying ρ to any measurable set changes its volume by a factor that only depends on ρ (a property that is well-known for affine transformations). From these properties we can interpret the special case with ball $E(0,I)$ and half-space H_B as the preimage with respect to ρ of the original problem. Because all volumes are scaled by the same factor, a minimum volume ellipsoid in the special case thus corresponds to a minimum volume ellipsoid for the original problem.

The transformation ρ is indeed a bijection since R is full-rank and Q_i is full-rank because

$A_i = Q_i Q_i^\top$ and A_i is full-rank. Furthermore,

$$\begin{aligned}
\rho(E(0,I)) &= \{\rho(x)\colon x \in \mathbb{R}^n, x^\top x \le 1\}\\
&= \{Q_i R x + a_i \colon x \in \mathbb{R}^n, x^\top x \le 1\}\\
&= \Big\{y \in \mathbb{R}^n \colon (y-a_i)^\top (Q_i^{-1})^\top R R^{-1} Q_i^{-1} (y-a_i) \le 1\Big\} && (y = Q_i R x + a_i)\\
&= \Big\{y \in \mathbb{R}^n \colon (y-a_i)^\top A_i^{-1} (y-a_i) \le 1\Big\} && (Q_i Q_i^\top = A_i)\\
&= E_i \enspace .
\end{aligned}$$

Similarly, the hyperplane H_B gets mapped to H_i:

$$\begin{aligned}
\rho(H_B) &= \{\rho(x)\colon x \in \mathbb{R}^n, x_1 \ge 0\}\\
&= \{Q_i R x + a_i \colon x \in \mathbb{R}^n, x_1 \ge 0\}\\
&= \{y \in \mathbb{R}^n \colon (R^{-1} Q_i^{-1}(y-a_i))^\top e_1 \ge 0\} && (y = Q_i R x + a_i)\\
&= \{y \in \mathbb{R}^n \colon (y-a_i)^\top (Q_i^{-1})^\top R e_1 \ge 0\} && (R^{-1} = R^\top \text{ by orthogonality of } R)\\
&= \{y \in \mathbb{R}^n \colon -(y-a_i)^\top c \ge 0\} && (R^\top Q_i^\top c = -e_1)\\
&= \{y \in \mathbb{R}^n \colon c^\top y \le c^\top a_i\}\\
&= H_i \enspace .
\end{aligned}$$

Notice that the above relations combined with $E_B \supseteq E(0,I) \cap H_B$ imply that $\rho(E_B)$ is indeed an ellipsoid that contains $\rho(E(0,I)) \cap \rho(H_B) = E_i \cap H_i$. It remains to show that $\rho(E_B)$ has minimum volume among all ellipsoid containing $E_i \cap H_i$. To prove this, let E_{i+1} be a minimum volume ellipsoid containing $E_i \cap H_i$ and we will show $\mathrm{vol}(E_{i+1}) \ge \mathrm{vol}(\rho(E_B))$ to finish the proof.

We recall a well-known property of affine functions $x \mapsto Ax + v$, namely that they scale any volume by the same factor $|\det A|$. For ρ this implies that for any measurable set $U \subseteq \mathbb{R}^n$ we have

$$\mathrm{vol}(\rho(U)) = \mathrm{vol}(U) \cdot |\det(Q_i R)| = \mathrm{vol}(U) \cdot |\det Q_i||\det R| = \mathrm{vol}(U) \cdot |\det Q_i| \enspace . \qquad (6.5)$$

Notice that $\rho^{-1}(E_{i+1})$ is an ellipsoid containing $\rho^{-1}(E_i) \cap \rho^{-1}(H_i) = E(0,I) \cap H_B$, and hence

$$\mathrm{vol}(\rho^{-1}(E_{i+1})) \ge \mathrm{vol}(E_B) \enspace , \qquad (6.6)$$

because E_B has minimum volume among all ellipsoid containing $E(0,I) \cap H_B$. We thus obtain

$$\begin{aligned}
\mathrm{vol}(E_{i+1}) &= \mathrm{vol}(\rho^{-1}(E_{i+1})) \cdot |\det Q_i| && \text{(by 6.5)}\\
&\ge \mathrm{vol}(E_B) \cdot |\det Q_i| && \text{(by 6.6)}\\
&= \mathrm{vol}(\rho(E_B)) \enspace , && \text{(by 6.5)}
\end{aligned}$$

thus completing the proof. □

Lemma 6.7

Let $H_B = \{x \in \mathbb{R}^n \colon x_1 \geq 0\}$. Then the ellipsoid

$$E_B = \left\{ x \in \mathbb{R}^n \;\middle|\; \left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2}\sum_{j=2}^{n} x_j^2 \leq 1 \right\} \tag{6.7}$$

contains $E(0,I) \cap H_B$.

The ellipsoid E_B is actually even the minimum volume ellipsoid containing $E(0,I) \cap H_B$. We do not show this fact in these notes. (We will prove this in one of the problem sets.) However, we will show that the ratio ${}^{\mathrm{vol}(E_B)}/_{\mathrm{vol}(E(0,I))}$ satisfies the inequality of Lemma 6.4. Of course, even without accepting that E_B is the smallest ellipsoid containing $E(0,I) \cap H_B$, this shows Lemma 6.4. Furthermore, also in the analysis that follows, we never need to prove that E_B is the smallest ellipsoid containing $E(0,I) \cap H_B$, because we can simply assume that, in the Ellipsoid Method, we work with the description of E_B given by (6.7) without assuming that it has minimum volume. We will later generalize the description of E_B given in (6.7) to the general case when E_i is not necessarily the unit ball and H_B is replaced by a general half-space going through the center of E_i.

Proof of Lemma 6.7. Let $x \in E(0,I) \cap H_B$. We have

$$\begin{aligned}
&\left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2}\sum_{j=2}^{n} x_j^2 \\
&\quad = \frac{n^2+2n+1}{n^2} x_1^2 - \left(\frac{n+1}{n}\right)^2 \frac{2x_1}{n+1} + \frac{1}{n^2} + \frac{n^2-1}{n^2}\sum_{j=2}^{n} x_j^2 \\
&\quad = \frac{2n+2}{n^2} x_1^2 - \frac{2n+2}{n^2} x_1 + \frac{1}{n^2} + \frac{n^2-1}{n^2}\underbrace{\sum_{j=1}^{n} x_j^2}_{\leq 1} && (x \in E(0,I)) \\
&\quad \leq \frac{2n+2}{n^2} \underbrace{x_1(x_1-1)}_{\leq 0} + 1 && (0 \leq x_1 \leq 1) \\
&\quad \leq 1 \ ,
\end{aligned}$$

and thus $x \in E_B$. □

Proof of Lemma 6.4. As discussed, we can assume due to Lemma 6.6 that $E_i = E(0,I)$ is the unit ball, and the separating hyperplane used in the Ellipsoid Method at iteration i, which goes through the center of E_i, is given by $H_B = \{x \in \mathbb{R}^n \colon x_1 \geq 0\}$. From (6.7) we can read off the matrix A_{i+1} and the center point a_{i+1}, which define the ellipsoid E_{i+1}, i.e.,

$$E_{i+1} = \{x \in \mathbb{R}^n \colon (x - a_{i+1})^\top A_{i+1}^{-1} (x - a_{i+1}) \leq 1\} \ .$$

We have

$$a_{i+1} = \left(\frac{1}{n+1}, 0, 0, \dots, 0\right)^\top \quad ,$$

$$A_{i+1}^{-1} = \begin{pmatrix} (\frac{n+1}{n})^2 & 0 & 0 & \dots & 0 \\ 0 & \frac{n^2-1}{n^2} & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & \ddots & 0 \\ 0 & \dots & & 0 & \frac{n^2-1}{n^2} \end{pmatrix} \quad ,$$

and thus

$$A_{i+1} = \begin{pmatrix} (\frac{n}{n+1})^2 & 0 & 0 & \dots & 0 \\ 0 & \frac{n^2}{n^2-1} & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & \ddots & 0 \\ 0 & \dots & & 0 & \frac{n^2}{n^2-1} \end{pmatrix} \quad .$$

As discussed, an ellipsoid $E(a, A)$ is the image of the unit ball with respect to the affine transformation $\phi(x) = Qx + a$, where $Q \in \mathbb{R}^{n\times n}$ is such that $A = QQ^\top$. Thus

$$\mathrm{vol}(E(a, A)) = \mathrm{vol}(E(0, I)) \cdot |\det(Q)| = \mathrm{vol}(E(0, I)) \cdot \sqrt{\det(A)} \quad .$$

Hence,

$$\begin{aligned} \frac{\mathrm{vol}(E_{i+1})}{\mathrm{vol}(E_i)} &= \frac{\sqrt{\det(A_{i+1})}}{\sqrt{\det(I)}} = \sqrt{\det(A_{i+1})} \\ &= \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{\frac{n-1}{2}} \\ &= \left(1 - \frac{1}{n+1}\right)\left(1 + \frac{1}{n^2-1}\right)^{\frac{n-1}{2}} \\ &< e^{-\frac{1}{n+1}} e^{\frac{n-1}{2(n^2-1)}} = e^{-\frac{1}{n+1}} e^{\frac{1}{2(n+1)}} = e^{-\frac{1}{2(n+1)}} \quad , \quad (1 + x < e^x \ \ \forall x \in \mathbb{R} \setminus \{0\}) \end{aligned}$$

as desired.

□

6.4.3 From the unit ball to the general case

In this section we discuss briefly how to derive an explicit description of the ellipsoid E_{i+1} in the general case, where E_i is not necessarily the unit ball and the half-space cutting E_i is a general half-space going through the origin a_i of E_i. Having such a description of E_{i+1} allows us to make the step of computing E_{i+1} in Algorithm 8 explicit.

We perform the generalization in two steps. First, we still assume that E_i is the unit ball. However, we consider an arbitrary half-space cutting it. Then, we extend this case to the general case.

An alternative way to get a description of E_{i+1} and a_{i+1} is to derive an explicit affine transformation that directly transforms the unit ball and half-space $\{x \in \mathbb{R}^n \colon x_1 \geq 0\}$ to a general ellipsoid and general half-space going through its center.

General half-space cutting $E(0, I)$

Assume that $E_i = E(0, I)$ is still the unit ball, and consider a general half-space $H_i = \{x \in \mathbb{R}^n \colon c^\top x \leq 0\}$, where $\|c\|_2 = 1$, which can easily be achieved by scaling c. One can use an orthogonal transformation to transform this case to the one where the half-space is given by $\{x \in \mathbb{R}^n \colon x_1 \geq 0\}$. One can verify (and we do this in the problem sets) that this leads to an ellipsoid

$$\begin{aligned} E_{i+1} &= E(a_{i+1}, A_{i+1}) \ \text{, where} \\ a_{i+1} &= -\frac{1}{n+1} c \ \text{, and} \\ A_{i+1} &= \frac{n^2}{n^2-1}\left(I - \frac{2}{n+1} cc^\top\right) \ . \end{aligned}$$

General case

Now let $E_i = E(a_i, A_i)$ be a general ellipsoid, and let $H_i = \{x \in \mathbb{R}^n \colon c^\top x \leq c^\top a_i\}$ be a general hyperplane going through the center a_i of E_i. Since A_i is positive definite, there is a matrix $Q_i \in \mathbb{R}^{n \times n}$ such that $A_i = Q_i Q_i^\top$. Consider the affine bijective transformation $\phi(x) = Q_i x + a_i$ and its inverse $\phi^{-1}(x) = Q_i^{-1}(x - a_i)$. The function ϕ transforms $E(0, I)$ to $E_i = E(a_i, A_i)$. Hence, we can first apply ϕ^{-1} to E_i and H_i to obtain $E_i' = E(0, I)$ and H_i', then we can use our description of E_{i+1} for the unit ball case, and finally, we transform the found ellipsoid back using ϕ.

We start by describing the image of H_i under ϕ^{-1} to obtain H_i'.

$$\begin{aligned} H_i' = \phi^{-1}(H_i) &= \{Q_i^{-1}(x - a_i) \colon x \in \mathbb{R}^n, c^\top x \leq c^\top a_i\} \\ &= \{y \in \mathbb{R}^n \colon c^\top (Q_i y + a_i) \leq c^\top a_i\} && (y = Q_i^{-1}(x - a_i)) \\ &= \{y \in \mathbb{R}^n \colon c^\top Q_i y \leq 0\} \ . \end{aligned}$$

Hence, we have

$$\begin{aligned} H_i' &= \{x \in \mathbb{R}^n \colon d^\top x \le 0\} \ , \text{ where} \\ d &= \frac{Q_i^\top c}{\sqrt{c^\top Q_i Q_i^\top c}} = \frac{Q_i^\top c}{\sqrt{c^\top A_i c}} \ . \end{aligned}$$

By the unit ball case discussed previously, the minimum volume ellipsoid E'_{i+1} that contains $E(0,I) \cap H_i'$ is given by

$$\begin{aligned} E'_{i+1} &= E(a'_{i+1}, A'_{i+1}) \ , \text{ where} \\ a'_{i+1} &= -\frac{1}{n+1} d \ , \text{ and} \\ A'_{i+1} &= \frac{n^2}{n^2-1}\left(I - \frac{2}{n+1} dd^\top\right) \ . \end{aligned}$$

Hence, the ellipsoid E_{i+1} is given by

$$\begin{aligned} E_{i+1} = \phi(E'_{i+1}) &= \{\phi(x) \colon x \in \mathbb{R}^n, (x - a'_{i+1})^\top A'^{-1}_{i+1}(x - a'_{i+1}) \le 1\} \\ &= \{y \in \mathbb{R}^n \colon (Q_i^{-1}(y - a_i) - a'_{i+1})^\top A'^{-1}_{i+1}(Q_i^{-1}(y - a_i) - a'_{i+1}) \le 1\} \\ &= \{y \in \mathbb{R}^n \colon (y - a_i - Q_i a'_{i+1})^\top (Q_i^{-1})^\top A'^{-1}_{i+1} Q_i^{-1}(y - a_i - Q_i a'_{i+1}) \le 1\} \ . \end{aligned}$$

where the second equality follows by using the substitution $y = \phi(x) = Q_i x + a_i$.

From this description we can derive a description of the center a_{i+1} and defining positive definite matrix A_{i+1} of $E_{i+1} = E(a_{i+1}, A_{i+1})$. For simplicity of notation we define

$$b = \frac{A_i c}{\sqrt{c^\top A_i c}} = Q_i d \ .$$

We obtain,

$$a_{i+1} = a_i + Q_i a'_{i+1} = a_i - \frac{1}{n+1} Q_i d = a_i - \frac{1}{n+1} b \ ,$$

and

$$A^{-1}_{i+1} = (Q_i^{-1})^\top A'^{-1}_{i+1} Q_i^{-1} \ ,$$

which implies

$$A_{i+1} = Q_i A'_{i+1} Q_i^\top = \frac{n^2}{n^2-1}\left(A_i - \frac{2}{n+1} bb^\top\right) \ .$$

We can now restate the Ellipsoid Method as described in Algorithm 8 in a more explicit form, by giving explicit formulas for how to compute E_{i+1}. Notice that there is no need to compute a Cholesky factorization $A_i = Q_i Q_i^\top$ of A_i.

Algorithm 9: Ellipsoid Method

```
Input  : Separation oracle for a polytope $P \subseteq \mathbb{R}^n$ with $\dim(P) = n$, and an ellipsoid
         $E_0 = E(a_0, A_0)$ with $P \subseteq E_0$.
Output: A point $y \in P$.
$i = 0$.
while $a_i \notin P$ (checked with separation oracle) do
    Get $c \in \mathbb{R}^n$ such that $P \subseteq \{x \in \mathbb{R}^n : c^\top x < c^\top a_i\}$, using separation oracle.
    Let $b = \frac{A_i c}{\sqrt{c^\top A_i c}}$.
    Let $a_{i+1} = a_i - \frac{1}{n+1} b$.
    Let $A_{i+1} = \frac{n^2}{n^2-1}(A_i - \frac{2}{n+1} bb^\top)$.
    $i = i + 1$.
return $a_i$.
```

We recall that the square root that has to be taken for the calculation of b is an operation that we cannot perform exactly under the usual computational assumptions. Therefore, to obtain a working polynomial time version of the Ellipsoid Method, one typically only computes an approximate version of E_{i+1} that still contains $E_i \cap H_i$ but is slightly bigger than the minimum volume ellipsoid containing $E_i \cap H_i$.

6.4.4 From checking feasibility to optimization over $\{0,1\}$-polytopes

Even though the Ellipsoid Method works in a much more general context, we focus from now on on the problem of maximizing a linear function w over a $\{0,1\}$-polytope P. This case is highly relevant in combinatorial optimization, and allows us to show nicely how one can derive that the Ellipsoid Method runs in polynomial time on an interesting class of LPs.

Without loss of generality we can assume $w \in \mathbb{Z}^n$, since any $w \in \mathbb{Q}^n$ can be transformed into an integral vector through scaling. We start by discussing how to obtain the optimal value $\nu^* = \max\{w^\top x \colon x \in P\}$ of the LP in polynomial time, before we provide details of how this procedure can be used to also obtain an optimal vertex solution $x^* \in P$.

Again, we assume that P is full-dimensional.

Getting the optimal LP value ν^*

Let $w_{\max} = \max\{|w_k| \colon k \in [n]\}$. Since $P \subseteq \mathbb{R}^n$ is a $\{0,1\}$-polytope, there is an optimal solution $x^* \in \{0,1\}^n$, and since $\nu^* = w^\top x^*$, we have

$$\nu^* \in [-nw_{\max}, nw_{\max}] \cap \mathbb{Z} \ .$$

To determine ν^*, we check the non-emptiness of a series of polytopes of the form

$$P(\nu) = P \cap \left\{x \in \mathbb{R}^n \colon w^\top x \geq \nu - \frac{1}{2}\right\} \ ,$$

where $\nu \in [-nw_{\max}, nw_{\max}] \cap \mathbb{Z}$. Notice that ν^* is the largest $\nu \in [-nw_{\max}, nw_{\max}] \cap \mathbb{Z}$ for which $P(\nu)$ is non-empty. Hence, we can find ν^* by performing a binary search over $\nu \in$

$[-nw_{\max}, nw_{\max}] \cap \mathbb{Z}$ and checking non-emptiness of $P(\nu)$. This takes $O(\log(nw_{\max})) = O(\log n + \log w_{\max})$ iterations, which is polynomial in the input.

Notice that in the definition of $P(\nu)$, we are looking for solutions of value at least $\nu - \frac{1}{2}$, instead of simply looking for solutions of value ν. Adding the $-\frac{1}{2}$ term guarantees that $P(\nu)$ is either empty or full-dimensional. If we drop the $-\frac{1}{2}$ term, then the polytope $P(\nu^*)$ is not full-dimensional anymore.

To show that ν^* can be computed in polynomial time via the Ellipsoid Method we still need to prove that we can check non-emptiness of $P(\nu)$ in polynomial time for any $\nu \in [-nw_{\max}, nw_{\max}] \cap \mathbb{Z}$. For this we have to specify what starting ellipsoid E_0 we choose, and we need to show that the Ellipsoid Method finds a point in $P(\nu)$ in a polynomial number of iterations if $P(\nu) \neq \emptyset$.

Starting ellipsoid E_0 As the starting ellipsoid E_0, we choose the ball centered at $(\frac{1}{2}, \dots, \frac{1}{2})$ of radius $\frac{1}{2}\sqrt{n}$, which is the smallest ball that contains $[0,1]^n$. Hence, $P(\nu) \subseteq [0,1]^n \subseteq E_0$. The volume of E_0 is

$$\mathrm{vol}(E_0) = \frac{1}{2^n}(\sqrt{n})^n \, \mathrm{vol}(E(0,I)) \enspace .$$

We recall that the volume of the unit ball $E(0,I)$ is given by

$$\mathrm{vol}(E(0,I)) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} \leq \pi^{n/2} \leq 2^n \enspace ,$$

where Γ is the gamma function, which extends factorials to non-integers. It is given by $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$. However, the crude upper bound $\mathrm{vol}(E(0,I)) \leq 2^n$ also immediately follows by observing that $E(0,I) \subseteq [-1,1]^n$, and hence $\mathrm{vol}(E(0,I)) \leq \mathrm{vol}([-1,1]^n) = 2^n$. Thus,

$$\log(\mathrm{vol}(E_0)) = O(n \log n) \enspace .$$

Bounding the number of iterations To bound the number of iterations of the Ellipsoid Method when applied to $P(\nu)$ for $\nu \in \mathbb{Z}$ we leverage Lemma 6.5, for which we need to show that if $P(\nu) \neq \emptyset$, then its volume is not too small. Hence, assume $P(\nu) \neq \emptyset$, which implies $\exists y_0 \in P(\nu) \cap \{0,1\}^n$. (Actually, any optimal vertex solution to $\max\{w^\top x \colon x \in P\}$ can be chosen as y_0.) Since P is assumed to be a full-dimensional $\{0,1\}$-polytope, there are furthermore points $y_1, \dots, y_n \in P \cap \{0,1\}^n$ such that the simplex $\Delta = \mathrm{conv}(\{y_0, \dots, y_n\})$ is full-dimensional. The points $y_1, \dots, y_n$ are not necessarily contained in $P(\nu)$. Therefore we shrink the simplex Δ towards the vertex y_0, to obtain a scaled-down simplex Δ' which is fully contained in $P(\nu)$. We will then use $\mathrm{vol}(\Delta')$ as a lower bound for $\mathrm{vol}(P(\nu))$. More precisely, we show that it suffices to shrink Δ by a factor of

$$\alpha = \frac{1}{2nw_{\max}} \enspace .$$

The simplex Δ' has as vertices y_0, and the following points $z_1, \dots, z_n$:

$$z_i = y_0 + \alpha(y_i - y_0) \quad \forall i \in [n] \enspace .$$

To show $\Delta' = \operatorname{conv}(\{y_0, z_1, \dots, z_n\}) \subseteq P(\nu)$, we have to show that $z_i \in P(\nu)$ for $i \in [n]$, which reduces to checking $w^\top z_i \geq \nu - \frac{1}{2}$ because $z_i \in P$. This indeed holds:

$$\begin{aligned} w^\top z_i = \underbrace{w^\top y_0}_{\geq \nu} + \alpha w^\top (y_i - y_0) &\geq \nu + \alpha w^\top (y_i - y_0) \quad (\text{since } w^\top y_0 \geq \nu - 1/2 \text{ and } w^\top y_0 \in \mathbb{Z}) \\ &\geq \nu - \alpha n w_{\max} = \nu - \frac{1}{2} \ . \end{aligned}$$

Hence, $\Delta' \subseteq P(\nu)$. Furthermore, since Δ' is a scaled-down version of Δ, with scaling factor α, we obtain

$$\operatorname{vol}(\Delta') = \alpha^n \operatorname{vol}(\Delta) \ .$$

Recall that the volume of a simplex in $\mathbb{R}^n$ is $\frac{1}{n!}$ times the volume of the corresponding parallelepiped. In particular, we obtain

$$\operatorname{vol}(\Delta) = \frac{1}{n!} \cdot |\det(y_1 - y_0, \dots, y_n - y_0)| \geq \frac{1}{n!} \ ,$$

since the matrix with columns $y_i - y_0$ is integral and non-singular because Δ is full-dimensional. Hence,

$$\operatorname{vol}(\Delta') = \alpha^n \operatorname{vol}(\Delta) \geq \left(\frac{1}{2n w_{\max}}\right)^n \frac{1}{n!} \ ,$$

and therefore

$$-\log(\operatorname{vol}(P(\nu))) \leq -\log(\operatorname{vol}(\Delta')) = O(n \log n + n \log w_{\max}) \ .$$

Thus, by Lemma 6.5 the number of iterations needed by the Ellipsoid Method to find a point in $P(\nu)$ is at most

$$\begin{aligned} 2(n+1) \ln \left(\frac{\operatorname{vol}(E_0)}{\operatorname{vol}(P(\nu))} \right) &= O\big(n \cdot [\log(\operatorname{vol}(E_0)) - \log(\operatorname{vol}(P(\nu))]\big) \\ &= O(n \cdot (n \log n + n \log w_{\max})) \\ &= O(n^2 \cdot (\log n + \log w_{\max})) \ , \end{aligned}$$

which is polynomial.

Consequently, to check non-emptiness of $P(\nu)$ for an arbitrary $\nu \in \mathbb{Z}$, it suffices to run the Ellipsoid Method for $O(n^2 \cdot (\log n + \log w_{\max}))$ iterations. It will either find a point in $P(\nu)$, which obviously shows non-emptiness of $P(\nu)$, or if no point in $P(\nu)$ is found over this many iterations, then $P(\nu) = \emptyset$.

Determining an optimal $\{0,1\}$-solution x^*

The above discussion shows that we can find with the Ellipsoid Method the optimal value ν^* of any LP over a full-dimensional $\{0,1\}$-polytope P. Furthermore, we also get a point $y \in P$ with $w^\top y \geq \nu^* - \frac{1}{2}$. However, especially in Combinatorial Optimization settings we are interested in getting an optimal vertex solution $x^* \in P \cap \{0,1\}^n$, which typically corresponds to some underlying discrete structure we are interested in. There are several ways how this can be done.

One procedure that often works very efficiently is to start with y and do local operations to obtain a vertex x with $w^\top x \geq w^\top y$, which implies that x is an optimal vertex solution. We do not go into details of this procedure, and present a conceptually simpler, though often slower, approach that works in our setting and is efficient.

The idea of this approach is to reduce the problem of finding an optimal vertex solution to a sequence of problems that ask to find only the optimal value of an LP. Let $S \subseteq [n]$. Consider a modified objective function w^S defined by

$$w_i^S = \begin{cases} w_i + 1 & \text{if } i \in S \ , \\ w_i & \text{if } i \in [n] \setminus S \ . \end{cases}$$

Optimizing with respect to w^S allows for obtaining insights into an optimal solution using the following result.

Lemma 6.8

Let $S \subseteq [n]$. The following two statements are equivalent.

(i) There is an optimal solution x^* to $\max\{w^\top x \colon x \in P\}$ with $x_i^* = 1$ for $i \in S$.
(ii) $\nu^* + |S| = \max\{(w^S)^\top x \colon x \in P\}$.

Proof. By definition of w^S, we have for any point $y \in \{0,1\}^n$,

$$(w^S)^\top y = w^\top y + |\{i \in S \colon y_i = 1\}| \ .$$

Clearly, for any vertex $y \in P$, we have $w^\top y \leq \nu^*$ with equality if and only if y^* is a maximizer of $\max\{w^\top x \colon x \in P\}$. Furthermore, we also have $|\{i \in S \colon y_i = 1\}| \leq |S|$. This shows that $\max\{(w^S)^\top x \colon x \in P\} \leq \nu^* + |S|$, with equality if and only if there is a vertex $y \in P$ that maximizes $\max\{w^\top x \colon x \in P\}$ and satisfies $y_i = 1$ for $i \in S$, thus proving the lemma. □

Based on Lemma 6.8, we can construct an optimal vertex solution coordinate by coordinate. More precisely, we will determine among all optimal vertex solutions of $\max\{w^\top x \colon x \in P\}$ the *lexicographically largest* one $x^* = (x_1^*, \dots, x_n^*)$, i.e., for any other optimal vertex solution y, there exists $k = k(y) \in \{0, \dots, n-1\}$ such that $y_i = x_i^*$ for $i \in [k]$, and $x_{k+1}^* = 1$ whereas $y_{k+1} = 0$.

We construct x^* iteratively. In the ith iteration we determine the value of x_i^*. Assume that we know the values of $x_1^*, \dots, x_{i-1}^*$ for some $i \in [n]$. Let $U = \{j \in [i-1] \colon x_j^* = 1\}$, and define $S = U \cup \{i\}$. Using Lemma 6.8 we can decide whether there is an optimal solution z^* to $\max\{w^\top x \colon x \in P\}$ with $z_j^* = 1$ for $j \in S$. If this is the case, then $x_i^* = 1$, otherwise, $x_i^* = 0$. Hence, after n iterations we obtain x^*.

In summary, we obtain the following theorem.

Theorem 6.9

Let $P \subseteq \mathbb{R}^n$ be a full-dimensional $\{0,1\}$-polytope for which we are given a separation oracle. Furthermore, let $w \in \mathbb{Z}^n$. Then the Ellipsoid Method allows for finding an optimal vertex solution to the linear program $\max\{w^\top x \colon x \in P\}$ using a polynomial number of elementary operations and calls to the separation oracle for P.

6.5 Comments on the non-full-dimensional case

As stated in Theorem 6.2 we can also apply the Ellipsoid Method to optimize a linear function over a $\{0,1\}$-polytope P that is not full-dimensional. We only give a rough outline how this generalization can be achieved, without providing full details.

To simplify the explanation, assume that P is $n-1$ dimensional. Hence, there is a unique hyperplane $H = \{x \in \mathbb{R}^n \colon g^\top x = b\}$ such that $P \subseteq H$.

The ellipsoids constructed during the Ellipsoid Method become more and more flat over the iterations, because all of them contain P and their volumes shrink. After a well-chosen number of iterations $k = O(\mathrm{poly}(n))$ of the Ellipsoid Method, we consider the shortest axis of the current ellipsoid $E_k = E(a_k, A_k)$, which we denote by $d \in \mathbb{R}^n$. Hence, d is the eigenvector of A_k that corresponds to the smallest eigenvalue. Consider the hyperplane $F = \{x \in \mathbb{R}^n \colon d^\top x = d^\top a_k\}$. Intuitively, we expect F to be "close" to H. Indeed, if k is chosen well, one can round the coefficients of the hyperplane F to obtain the hyperplane H. However, this rounding procedure is not straightforward, and is based on an approximation algorithm to solve Simultaneous Diophantine Approximations. Once H is obtained, one can reduce the problem to the affine subspace defined by H by eliminating one of the coordinates using the equation provided by H. We therefore consider P to be a polytope in the affine subspace defined by H. Since P is $(n-1)$-dimensional in its original description in $\mathbb{R}^n$, we thereby obtain a full-dimensional polytope when removing one variable of P. A similar argument as above, applied repeatedly, works when P has dimension that is smaller than $n-1$.

Formalizing these arguments leads to a proof of Theorem 6.2.

7 Equivalence Between Optimization and Separation

The Ellipsoid Method provides a way to do (linear) optimization using a separation oracle, i.e., we can do optimization through separation. Interestingly, there are also approaches to solve the separation problem given an oracle for the optimization problem. In this section we highlight some main ideas that underlie this relation. As in the Ellipsoid Method, there are some technical requirements that need to be fulfilled for this approach to work. Still, this very strong relation between optimization and separation is often simply referred to as the "equivalence between optimization and separation".

Assume we want to solve the separation problem over a polytope $P \subseteq \mathbb{R}^n$, and the only way how we can access P is via an optimization oracle, which, for any $c \in \mathbb{R}^n$, returns an optimal solution to $\max\{c^\top x \colon x \in P\}$. To do separation through optimization, we define the so-called *polar* P° of P, which is a polyhedron over which we can separate if we can optimize over P. More formally, for any set $X \subseteq \mathbb{R}^n$, we define its polar to be

$$X^\circ = \{y \in \mathbb{R}^n \colon x^\top y \leq 1 \; \forall x \in X\} \enspace .$$

Figure 7.1 shows the polar of a polytope $P \subseteq \mathbb{R}^2$.

Given an optimization oracle for P, the following is a separation oracle for its polar P°. Let $y \in \mathbb{R}^n$ be the input of the separation problem over P°. We obtain a maximizer $z \in \operatorname{argmax}\{y^\top x \colon x \in P\}$ using the optimization oracle for P. If $y^\top z \leq 1$, then $y \in P^\circ$, because for any $x \in P$ we have $x^\top y \leq z^\top y \leq 1$. Otherwise, $\{x \in \mathbb{R}^n \colon z^\top x \leq 1\}$ is a half-space separating y from P°.

This simple observation together with some additional properties of the polar can be used to separate also over P, in many natural settings. For this we first state some properties of the polar.

Lemma 7.1

Let $X \subseteq \mathbb{R}^n$ be a compact (i.e., closed and bounded) convex set, containing the origin in its interior. Then

(a) X° is a compact convex set with the origin in its interior.
(b) $(X^\circ)^\circ = X$.

Proof. (a) We start by showing that X° is convex and compact. X° is clearly convex and closed since it is the intersection of half-spaces. To show boundedness of X°, notice that since the origin is in the interior of X, there exists $\delta > 0$ such that $B(0, \delta) \subseteq X$, where $B(0, \delta)$ is the closed ball of radius δ that is centered at the origin. We show boundedness

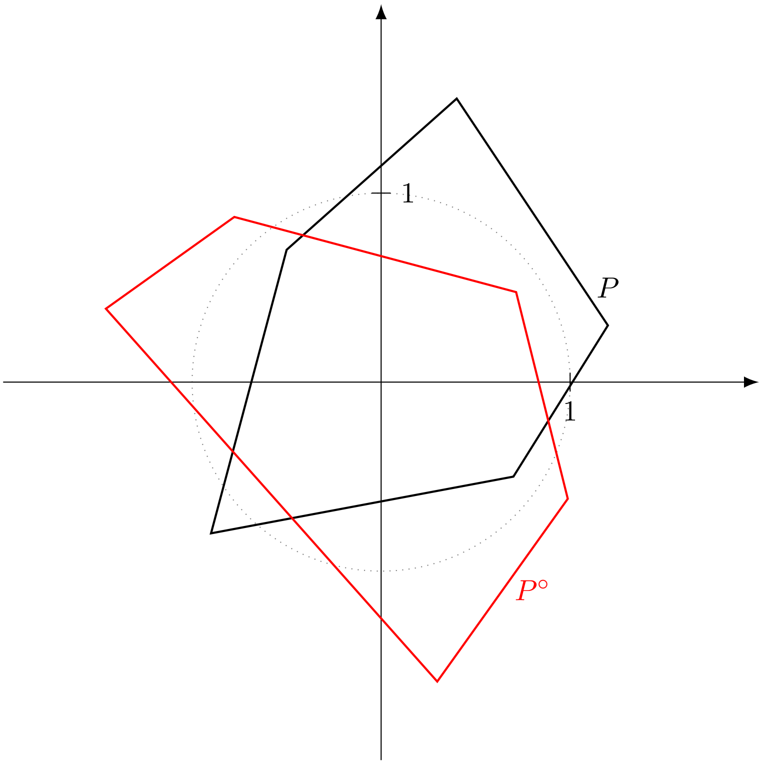


Figure 7.1: A polytope P and its polar P°.

of X° by showing that $X^\circ \subseteq B(0, 1/\delta)$. Indeed, for any $y \in \mathbb{R}^n$ with $\|y\|_2 > \frac{1}{\delta}$, we obtain $x^\top y > 1$ where $x = \delta \cdot \frac{y}{\|y\|_2} \in B(0, \delta) \subseteq X$; hence, $y \notin X^\circ$.
It remains to show that the origin is in the interior of X°. Since X is bounded, there exists some $M > 0$ such that $X \subseteq B(0, M)$. We complete this part of the proof by showing $B(0, \frac{1}{M}) \subseteq X^\circ$. Let $y \in B(0, \frac{1}{M})$. We will show that y does not violate any constraint in the description of X°. Indeed, for any $x \in X \subseteq B(0, M)$, we obtain

$$y^\top x \leq \underbrace{\|y\|_2}_{\leq 1/M} \underbrace{\|x\|_2}_{\leq M} \leq 1 \ ,$$

and hence $y \in X^\circ$.

(b) We clearly have $X \subseteq (X^\circ)^\circ$, because for any $x \in X$ and $y \in X^\circ$ we have $x^\top y \leq 1$. Thus, x fulfills all the constrains of $(X^\circ)^\circ$.
To show $(X^\circ)^\circ \subseteq X$, assume by contradiction that $\exists z \in (X^\circ)^\circ \setminus X$. Because X is convex and closed, there is a hyperplane separating z from X, i.e., there is $c \in \mathbb{R}^n \setminus \{0\}$ and $b \in \mathbb{R}$ such that $c^\top x \leq b \; \forall x \in X$ and $c^\top z > b$. Notice that $b > 0$ since the origin is in the interior of X. Let $y = \frac{1}{b} c$. We have $y \in X^\circ$, because for any $x \in X$,

$$x^\top y = \frac{1}{b} x^\top c \leq 1 \ .$$

However, $y \in X^\circ$ implies $z \notin (X^\circ)^\circ$ because

$$y^\top z = \frac{1}{b} c^\top z > 1 \ ,$$

which contradicts the assumption $z \in (X^\circ)^\circ$.

□

Lemma 7.2

Let $P \subseteq \mathbb{R}^n$ be a polytope containing the origin in its interior. Then P° is a polytope. Moreover, for any $x \in \mathbb{R}^n$, we have

$$x \text{ is a vertex of } P \quad \Leftrightarrow \quad \{y \in \mathbb{R}^n \colon x^\top y \leq 1\} \text{ is facet-defining for } P^\circ.$$

Proof. Let

$$Q = \{y \in \mathbb{R}^n \colon x^\top y \leq 1 \ \forall x \in \text{vertices}(P)\} \ . \tag{7.1}$$

We first show $P^\circ = Q$. Notice that this implies that P° is a polytope (boundedness follows from Lemma 7.1).

We clearly have $P^\circ \subseteq Q$. To show $Q \subseteq P^\circ$, consider a point $y \in Q$. We will show $y \in P^\circ$ by showing that $x^\top y \leq 1$ for all $x \in P$. Hence, let $x \in P$. Since P is a polytope, x can be written as a convex combination of its vertices, i.e., $x = \sum_{i=1}^k \lambda_i x_i$, where x_i is a vertex of P for $i \in [k]$, $\sum_{i=1}^k \lambda_i = 1$, and $\lambda_i \geq 0 \ \forall i \in [k]$. Thus,

$$x^\top y = \sum_{i=1}^k \lambda_i x_i^\top y \leq \sum_{i=1}^k \lambda_i = 1 \ ,$$

as desired, where the inequality follows from $x_i^\top y \leq 1 \ \forall i \in [k]$ since $y \in Q$.

$\Leftarrow$) Notice that $P^\circ = Q$ also implies the '$\Leftarrow$' implication of the statement, because the description (7.1) of $Q = P^\circ$ contains facet-defining inequalities for each facet of P°. Hence, if for some $x \in \mathbb{R}^n$ we have that $\{y \in \mathbb{R}^n \colon x^\top y \leq 1\}$ is facet-defining, then it has to be a possibly scaled version of one of the constraints in (7.1). However, because all right-hand sides of the considered constraints are one, this implies that the constraint $x^\top y \leq 1$ must be one of the constraints in the description (7.1), thus implying $x \in \text{vertices}(P)$.

$\Rightarrow$) Let $x_1, \dots, x_k$ be the vertices of P. We will show that $\{y \in \mathbb{R}^n \colon x_k^\top y \leq 1\}$ is facet-defining for P°. As shown in the first part, we have $P^\circ = Q := \{y \in \mathbb{R}^n \colon x_i^\top y \leq 1 \ \forall i \in [k]\}$. Hence, we have to show that $\{y \in \mathbb{R}^n \colon x_k^\top y \leq 1\}$ is facet-defining for Q.

Assume for the sake of deriving a contradiction that $\{y \in \mathbb{R}^n \colon x_k^\top y \leq 1\}$ is not facet-defining for Q. Notice that $Q = P^\circ$ is full-dimensional because it contains the origin in its interior due to Lemma 7.1. Together with Lemma 1.20, this implies that the non-facet-defining constraint $x_k^\top y \leq 1$ in (7.1) is redundant. Hence, the constraint $x_k^\top y \leq 1$ is implied by the constraints $x_i^\top y \leq 1$ for $i \in [k-1]$, i.e., a conic combination of the constraints $x_i^\top y \leq 1$ for $i \in [k-1]$ leads to the constraint $x_k^\top y \leq \alpha$ for some $\alpha \leq 1$. Formally, this means that there are multipliers

$\lambda_i \geq 0$ for $i \in [k-1]$ such that

$$\sum_{i=1}^{k-1} \lambda_i x_i = x_k \ \text{, and}$$

$$\sum_{i=1}^{k-1} \lambda_i \leq 1 \ .$$

However, this implies that x_k is a convex combination of $x_1, \dots, x_{k-1}$ and the origin. Moreover, notice that x_k is different from $x_1, \dots, x_{k-1}$, and x_k is not the origin, because x_k is a vertex of P and the origin is in the interior of P and thus cannot be a vertex of P. In other words, x_k is a non-trivial convex combination of other points in P, namely $x_1, \dots, x_{k-1}$ and the origin. This contradicts the fact of x_k being a vertex of P. □

Based on the above results, there is a natural strategy to construct a separation oracle for a polytope $P \subseteq \mathbb{R}^n$ that contains the origin in its interior. Notice that if P is full-dimensional, there is always a way to translate P such that it contains the origin in its interior. As discussed, an optimization oracle for P implies a separation oracle for P°. Now, if the technical requirements to apply the Ellipsoid Method to P° are fulfilled, we can use this separation oracle to optimize over P°. Finally, an optimization oracle for P° implies a separation oracle for $(P^\circ)^\circ = P$. Furthermore, Lemma 7.2 implies that if the optimization oracle always returns a vertex solution, then the separation oracle constructed with the above approach does not just return any separating hyperplane, but even one that is facet-defining.

Figure 7.2 summarizes this interplay between optimization and separation through the polar.

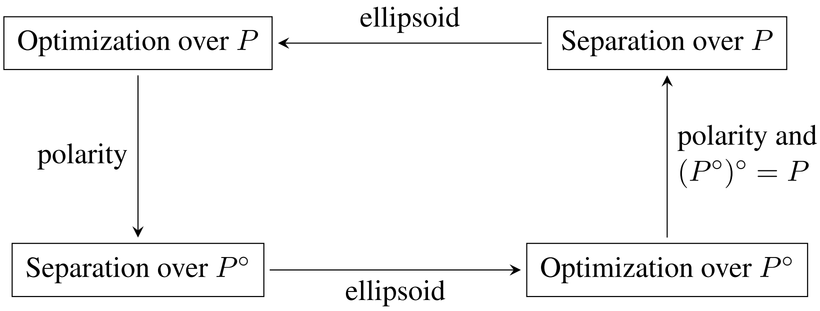


Figure 7.2: A graphical representation of the relation between optimization and separation via the Ellipsoid Method and polar polytopes.

8 Integer Programming

In short, integer programming is linear programming with the additional requirement that all variables need to take integral values. Integer programs (IPs) allow for modeling a very large class of discrete mathematical optimization problems. In particular, most classical combinatorial optimization problems can easily be modeled as IPs. However, this comes at a price. IPs are an $\mathcal{NP}$-hard problem class and, depending on the problem type at hand, finding optimal (or even feasible) solutions may take excessive time. Nevertheless, strong methods have been developed to deal with many IPs that we face in real-world applications. In this chapter, we provide a brief introduction to some of the key techniques used to solve IPs in practice. This exposition is less focussed on specific theoretical properties, but rather provides a glimpse into a broadly used and highly influential toolbox to solve IPs.

8.1 Introduction to integer programming

As mentioned, integer programs are defined analogously to linear programs with the only difference that all variables are required to take integer values, for example,

$$\begin{array}{rrcl} \max & c^\top x & & \\ & Ax & \leq & b \\ & x & \in & \mathbb{Z}^n \ , \end{array}$$

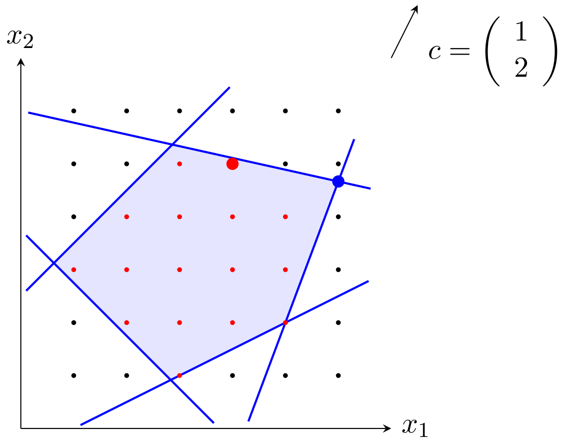
where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$. As with linear programming, the above form is just one way to write integer programs. More generally, one can also use equalities or '$\geq$' constraints.

Figure 8.1 shows an example integer program in two dimensions. When forgetting about the integrality constraints, one obtains a linear program, which is called the *linear relaxation* of the IP. Unfortunately, the linear relaxation of an IP often does not shed much light on the integer program. Still, many methods for solving IPs repeatedly use linear relaxations of certain well-defined sub-problems. In particular branch & bound procedures and also branch & cut procedures heavily rely on linear relaxations, because they can be solved very quickly and can be used as upper bounds on underlying IPs.

We start with a recreational application of IPs, by showing how they can be used to solve the eight queens puzzle.

IP example: the eight queens puzzle

The eight queens puzzle is a famous problem linked to chess. The goal of the puzzle is to place eight chess queens on a standard 8×8 chessboard so that no two queens threaten each other, i.e., there is no more than one queen in each row, column, and diagonal. Figure 8.2 shows the squares



$$\begin{array}{lrcrcr} \max & x_1 & + & 2x_2 & & \\ \text{s.t.} & 2x_1 & + & 9x_2 & \leq & 54 \\ & -2x_1 & + & 2x_2 & \leq & 5 \\ & 4x_1 & + & 4x_2 & \geq & 15 \\ & x_1 & - & 2x_2 & \leq & 1 \\ & 8x_1 & - & 3x_2 & \leq & 34 \end{array}$$

Figure 8.1: Example of an integer program in two dimensions. The blue area shows the linear relaxation of the IP, i.e., the feasible solutions if one disregards the integrality constraints. The IP only optimizes over the integer points inside the relaxation, which are highlighted in red. The blue point shows the unique optimal solution of the linear relaxation, and the large red one shows the unique optimal solution of the IP.

threatened by a queen and Figure 8.3 shows an example placement of seven queens where there are no unthreatened squares left for an eighth queen.

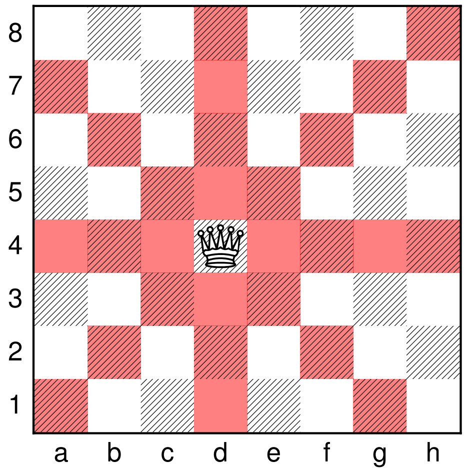


Figure 8.2: The squares in red are those where the shown queen can move to. Any other queen on one of those squares is said to be *threatened* by the shown queen.

To reformulate this combinatorial problem in terms of an integer program, we introduce a binary variable $x_{ij} \in \{0,1\}$ for each square of the board, which is indexed by a pair $(i,j) \in [8] \times [8]$. Hence, each such binary variable will simply be an integer variable with linear constraints

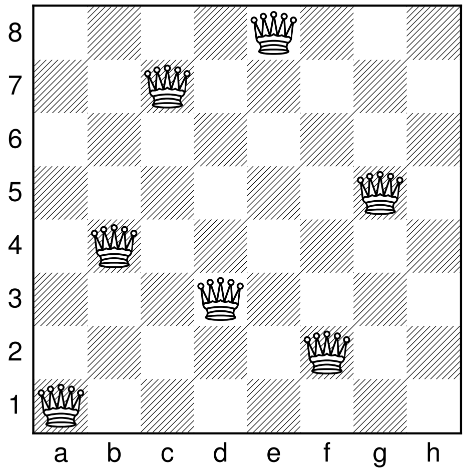


Figure 8.3: Example of a placement of seven queens on the board. Note that there is a queen in each of the files a-g and every square in the h file is threatened, so there is no valid square to place the eighth queen.

$0 \leq x_{ij} \leq 1$. Moreover, we think of $x_{ij} = 1$ as placing a queen on square (i,j), and of $x_{ij} = 0$ as not doing so.

The constraints now follow in a straightforward way from the statement of the puzzle: For each row, column, and diagonal of the chessboard, the sum of the corresponding variables is at most 1. The objective of the optimization problem is to maximize the sum of all components of the vector x, i.e., the number of queens placed on the board. Thus, we arrive at the following IP:

$$\begin{array}{rrcll} \max & \displaystyle\sum_{i=1}^{8}\sum_{j=1}^{8} x_{ij} & & & \\ & \displaystyle\sum_{j=1}^{8} x_{ij} & \leq & 1 & \forall i \in [8] \\ & \displaystyle\sum_{i=1}^{8} x_{ij} & \leq & 1 & \forall j \in [8] \\ & \displaystyle\sum_{\substack{i,j\in[8]:\\ i+j=k}} x_{ij} & \leq & 1 & \forall k \in \{2,3,\dots,16\} \\ & \displaystyle\sum_{\substack{i,j\in[8]:\\ i-j=k}} x_{ij} & \leq & 1 & \forall k \in \{-7,-6,\dots,7\} \\ & x & \in & \{0,1\}^{8\times 8} & . \end{array} \tag{8.1}$$

Let us check that the optimal solutions of the IP (8.1) correspond one-to-one to the solutions of the eight queens puzzle. On the one hand, the optimal value does not exceed 8 because a chessboard has 8 rows and there can be at most one queen per row. On the other hand, every solution of the puzzle corresponds to a feasible solution of the integer program attaining the value 8, which is the upper bound. Therefore, the optimal solutions of the integer program are precisely the solutions of the eight queens puzzle.

Figure 8.4 shows an optimal solution to the problem.

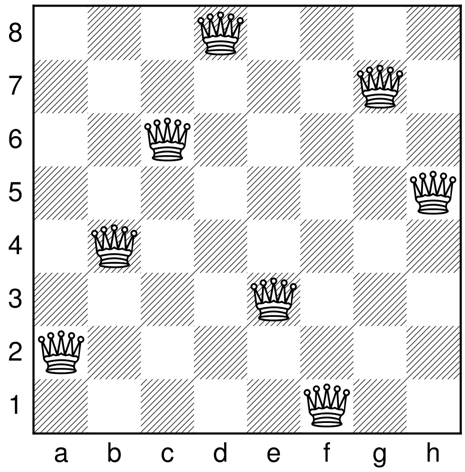


Figure 8.4: A solution to the eight queens puzzle.

It turns out that the eight queens puzzle has 12 distinct solutions that are not related by rotations or reflections, and 92 solutions in total.

Exercise 8.1

Construct an IP to find a solution to the eight queens puzzle that cannot be obtained from the one shown in Figure 8.4 through rotations and reflections.

8.2 Branch & bound

Branch & bound, and various variations thereof, is one of the most common solution approaches for integer programs. The principal idea is to intelligently explore the solution space and to try to cut off large parts of the solution space as soon as one can be sure that there is no optimal solution in them.

To illustrate the approach we consider the following IP with binary variables. The approach can easily be extended to integer variables with larger ranges.

$$\begin{array}{rrcrcrcrcl}
\max & 75x_1 & + & 6x_2 & + & 3x_3 & + & 33x_4 & & \\
 & 774x_1 & + & 76x_2 & + & 22x_3 & + & 42x_4 & \leq & 875 \\
 & 67x_1 & + & 27x_2 & + & 794x_3 & + & 53x_4 & \leq & 875 \\
 & & & & & & & x & \in & \{0,1\}^4
\end{array}$$

We start by replacing the integrality constraints $x \in \{0,1\}^4$ by $x \in [0,1]^4$ and solve the resulting linear relaxation. This leads to the solution shown in box 1 of Figure 8.5 (the number of the box is highlighted with a gray background), i.e.,

$$(x_1, x_2, x_3, x_4) = (1, 0.506, 0.934, 1) \ ,$$

with objective value $z = 113.837$. If this LP solution had been integral, we could have stopped right-away and return it as optimal solution. Indeed, the LP we solved optimized over a bigger solution set, obtained from the original IP by ignoring the integrality requirement on the variables. However, the values of x_2 and x_3 are not integral. We choose one of these variables, say x_2, and *branch* on this variable. More precisely, we create two sub-problems, one in which we force x_2 to be equal to 0 (box 9 in Figure 8.5) and one with $x_2 = 1$ (box 2). (If x_2 had been an integer variable with a larger range instead of a binary one, then we could have branched into two problems where one requires $x_2 \leq 0$ and the other one $x_2 \geq 1$.)

The optimal solution to the original problem is the better of the optimal solutions to the two newly created problems through branching. Branch & bound now recurses on the sub-problems, leading to a so-called branch & bound tree as shown in Figure 8.5. However, this tree is typically constructed in a depth-first order. Actually, the numbering of the boxes in Figure 8.5 highlights a depth-first order in which the sub-problems may be constructed and explored in a branch & bound procedure.

Let us start with the first 5 boxes. Here we successively branch on a fractional variable. In this case only the 5th box, where all variables are set to integral values, leads to a feasible integral solution. If it had happened that already an earlier sub-problem, say the one corresponding to box 4, had been integral, then we would not have branched further. Box 5 is the first feasible solution we found and we save it together with its value. We backtrack to the previous problem, the one corresponding to box 4, and go into the branch $x_3 = 1$. This leads to an infeasible linear program, which we ignore. We backtrack to the closest box above box 6 for which one of the two branches has not been explored yet. This is box 3, and we explore the other branch, i.e., $x_4 = 1$. In this case, the resulting LP is infeasible, and we can therefore stop exploring this branch further. Clearly, if the LP is infeasible, then the IP, which has integrality constraints on top of the LP, is infeasible, too.

A new important step happens at the next box we explore after backtracking. This is box 8, for which the optimal LP value is 42. Because this value is strictly smaller than the value of the best solution we found so far, which was found in box 5 and has value 81, we can stop exploring this box any further. Notice that coincidentally, the optimal solution to the LP of box 8 happens to be integral. However, this is irrelevant for this reasoning. Indeed, if even the linear relaxation cannot improve on the currently best solution, then there is no reason to branch further to obtain an integral solution. This step is called the *bounding* step of the branch & bound procedure, and explains the second part of its name.

We proceed as explained, and find the problem in box 10, whose optimal LP solution happens to be integral and improves on the best solution found previously. The last sub-problem we consider is the one in box 11, where we have an optimal LP value of only 86.72, which is less than the value of 108 of our best solution found so far, coming from box 10. Hence, no further exploration of problems stemming from box 11 is necessary. At this point, the branch & bound tree has been fully explored, and we return the best solution found, which is the one from box 10.

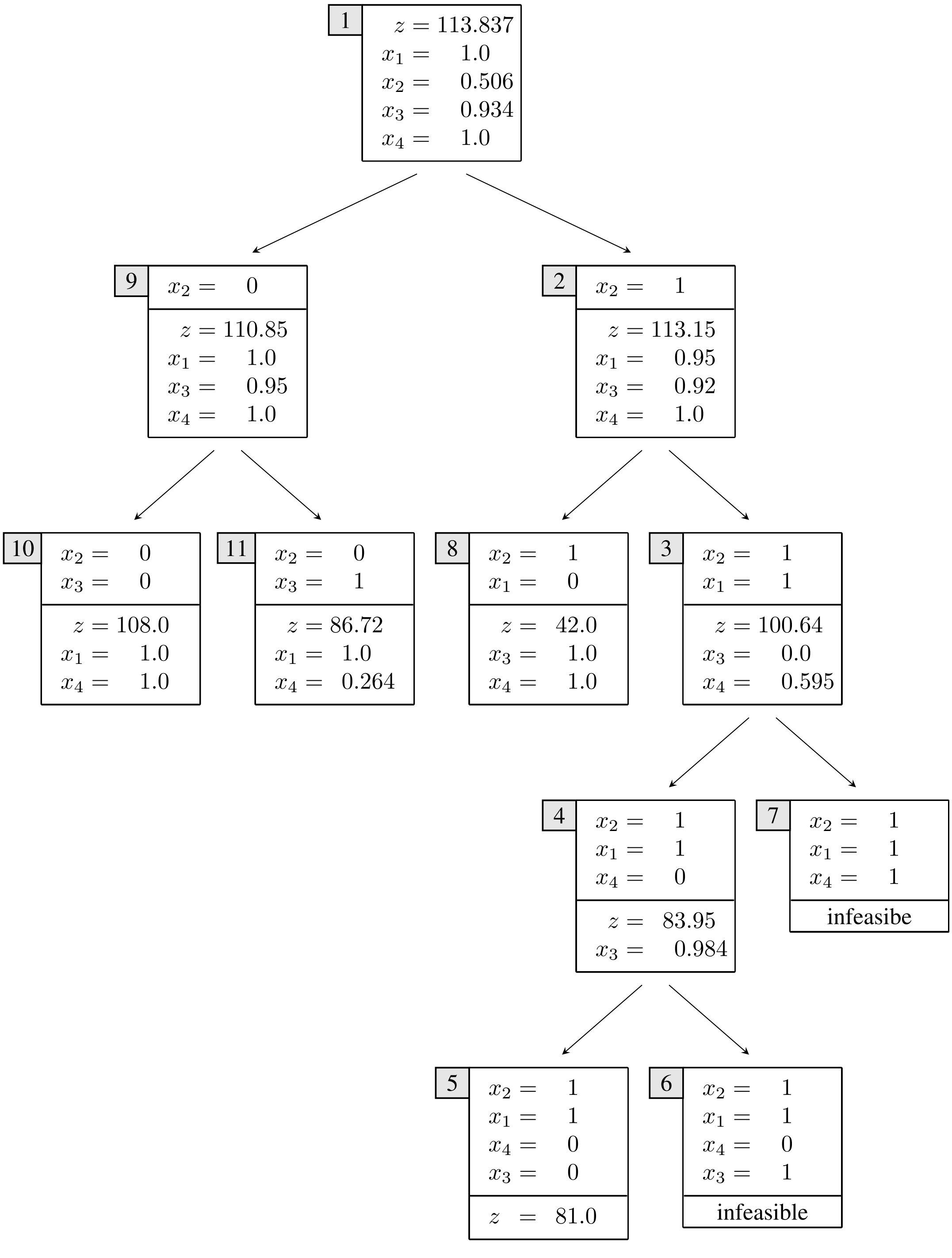


Figure 8.5: A branch & bound tree. Box 1 shows the optimal solution to the linear relaxation of the original problem. This solution is fractional, and we branch on the variable x_2 to obtain two new LP solutions shown in boxes 9 and 2, and so on. The lower part of each box shows the optimal LP solution to the problem when fixing variables as shown in the upper part of the box.

Remark 8.2: Incumbent

The best solution found so far is often called the *incumbent*.

Remark 8.3: Non-uniqueness of branch & bound tree

The tree constructed in this way is not unique, because there are various orders in which the branchings can be performed. First, one can choose the variable on which to branch, and second one can choose on which of the two children to continue first when doing a branching.

Remark 8.4: Speed-ups via strong incumbents

For a branch & bound procedure to be fast, it is crucial that a strong feasible solution is found quickly, because this can be used for the bounding step in the future, i.e., to discard certain sub-problems because even their linear relaxation has a worse objective than the value of the incumbent. This explains why we applied depth-first search to explore the branch & bound tree, with the hope to quickly find a feasible solution. Moreover, branch & bound procedures are typically combined with various heuristics to quickly obtain strong feasible solutions, which then helps to speed up the exploration of the solution space through branch & bound.

Remark 8.5: Speed-ups via strong relaxations

As we discussed, the reason why branch & bound procedures do not need to enumerate over all possible solutions, but can sometimes discard large parts of the solution space, is due to bounding. The bounding step is based on comparing solutions to LP relaxations with the value of the incumbent. For this to be effective, we do not only need strong incumbents, but it is also paramount to use a strong linear relaxation to start with. Our discussions on polyhedral relaxations of problems provides important ingredients and intuition on how to build such relaxations for various problems. The focus in Chapter 5 on polyhedral approaches for combinatorial optimization problems was on efficiently solvable problems, where we were able to describe the corresponding combinatorial polytope. For hard problems, there is little hope that we can get a good inequality description of their combinatorial polytope, because it is unlikely that they can be solved efficiently. Nevertheless, similar techniques and reasonings allow for obtaining descriptions that are good approximations of the convex hull of the solutions of the IP we are interested in.

8.3 Branch & cut

Branch & cut is an enhancement of branch & bound procedures through so-called cutting planes. More precisely, consider a sub-problem in the branch & bound procedure where we would normally branch, i.e., the sub-problem

(i) is not infeasible,
(ii) the value of its LP relaxation is strictly better than the value of the incumbent,
(iii) its LP solution is not integral.

Hence, for the example considered in the previous section, this could be a sub-problem corresponding to any of the boxes 1, 2, 3, 4, or 9 in Figure 8.5. Instead of branching at such a node, a branch & cut procedure may first add so-called *cutting planes*. These are linear inequalities that cut off the current optimal LP solution, but do not cut off any integral solution. Figure 8.6 shows again the feasible solutions to our introductory IP for branch & bound together with a cutting plane (in green) that cuts off the current optimal LP solution. By adding a cutting plane, the LP relaxation is strengthened. One then solves the strengthened LP relaxation, and continues with that one.

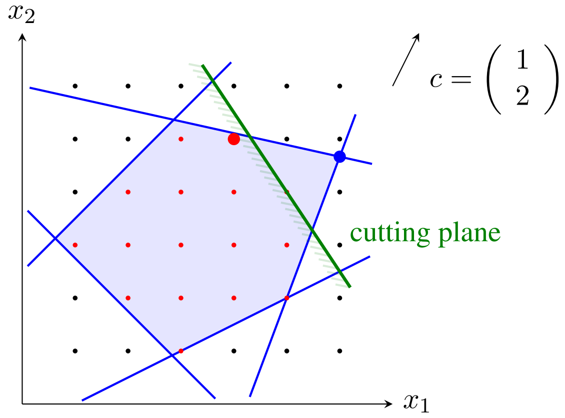


Figure 8.6: Example of a cutting plane. The cutting plane should cut off the current optimal solution to the linear relaxation but must not cut off any integral solutions.

The stronger LP, obtained after adding a cutting plane, leads to a better upper bound and makes it easier to use bounding, because the optimal value of the LP may have decreased after adding a cutting plane. Also, it is possible that the strengthened LP has an LP relaxation with an integral optimal solution, or one that is infeasible; in both cases, no further branching is necessary. In short, the goal of adding cutting planes is to speed up the procedure by reducing the total number of branchings, and hence sub-problems to be solved.

The study of cutting planes is a research area in its own. A detailed discussion of cutting planes is beyond the scope of this script. Still, to provide a glimpse into how cutting planes can be generated, we show how so-called Chvátal-Gomory cuts can be found through the simplex tableau.

8.3.1 Generating a Chvátal-Gomory cut through the simplex tableau

Chvátal-Gomory cuts are obtained by rounding the coefficients of linear inequalities that are valid for the linear relaxation. To exemplify how they can be constructed, consider again our introductory IP example for branch & bound, which we repeat below for convenience.

$$\begin{array}{lrcrcrcrcl}
\max & 75x_1 & + & 6x_2 & + & 3x_3 & + & 33x_4 & & \\
 & 774x_1 & + & 76x_2 & + & 22x_3 & + & 42x_4 & \leq & 875 \\
 & 67x_1 & + & 27x_2 & + & 794x_3 & + & 53x_4 & \leq & 875 \\
 & & & & & & & x & \in & \{0,1\}^4
\end{array}$$

We show how to add a cutting plane to the linear relaxation of this problem, i.e., this corresponds to box 1 in Figure 8.5. For this we solve the linear relaxation of the above problem with the Simplex Method. We first write the relaxation in canonical form:

$$\begin{array}{lrcrcrcrcl}
\max & 75x_1 & + & 6x_2 & + & 3x_3 & + & 33x_4 & & \\
 & 774x_1 & + & 76x_2 & + & 22x_3 & + & 42x_4 & \leq & 875 \\
 & 67x_1 & + & 27x_2 & + & 794x_3 & + & 53x_4 & \leq & 875 \\
 & x_1 & & & & & & & \leq & 1 \\
 & & & x_2 & & & & & \leq & 1 \\
 & & & & & x_3 & & & \leq & 1 \\
 & & & & & & & x_4 & \leq & 1 \\
 & & & & & & & x & \in & \mathbb{R}^4_{\geq 0} \ .
\end{array}$$

By adding slack variables $y_1, \dots, y_6$ to the above inequalities, where the i-th inequality from top down gets the slack variable y_i, we obtain the problem in standard form, with the following corresponding tableau.

	x_1	x_2	x_3	x_4	1
z	-75	-6	-3	-33	0
y_1	774	76	22	42	875
y_2	67	27	794	53	875
y_3	1	0	0	0	1
y_4	0	1	0	0	1
y_5	0	0	1	0	1
y_6	0	0	0	1	1

By applying phase II of the Simplex Method, we obtained the following optimal tableau.

	y_3	y_1	y_2	y_6	1
z	14.2289	0.0784	0.0016	29.623	113.8373
x_2	-10.2608	0.0133	-0.0004	-0.5386	0.506
x_3	0.2645	-0.0005	0.0013	-0.0484	0.9337
x_1	1	0	0	0	1
y_4	10.2608	-0.0133	0.0004	0.5386	0.494
y_5	-0.2645	0.0005	-0.0013	0.0484	0.0663
x_4	0	0	0	1	1

Hence, an optimal LP solution is

$$(x_1, x_2, x_3, x_4) = (1, 0.506, 0.9337, 1) \ ,$$

which corresponds to the solution that is shown in box 1 of Figure 8.5. This solution is fractional because x_2 and x_3 have fractional values. In particular, this implies that these two variables are basic; for otherwise, they would have value zero in the basic solution that corresponds to the tableau. Consider one of the two variables, say x_2. Its corresponding constraint in the optimal tableau reads

$$x_2 - 10.2608y_3 + 0.0133y_1 - 0.0004y_2 - 0.5386y_6 = 0.506 \ .$$

By rounding down the coefficients of the left-hand side to the next integer, we obtain the following weaker constraint with integral coefficients:

$$x_2 + \lfloor -10.2608 \rfloor y_3 + \lfloor 0.0133 \rfloor y_1 + \lfloor -0.0004 \rfloor y_2 + \lfloor -0.5386 \rfloor y_6 \leq 0.506 \ ,$$

i.e.,

$$x_2 - 11y_3 - y_2 - y_6 \leq 0.506 \ .$$

We now observe that in an integral solution, not just the variables x_1, x_2, x_3, x_4 need to be integral, but also the slack variables $y_1, \ldots, y_6$, because all the coefficients and right-hand sides of our problem are integral. (Note that this can be achieved for any rational LP/IP by scaling the constraints.) Because the above inequality has integral coefficients on the left-hand side, any integral solution will lead to an integral left-hand side value. Hence, we can round down the right-hand side coefficient to the next integer and obtain an inequality that is valid for any feasible *integral* solution to the original problem. The resulting inequality is

$$x_2 - 11y_3 - y_2 - y_6 \leq \lfloor 0.506 \rfloor = 0 \ . \tag{8.2}$$

Moreover, the basic optimal solution we computed must violate this constraint, because it sets the non-basic variables to 0. Hence, even after rounding its coefficients, the constraint remains tight for this solution. Moreover, we strictly decrease the right-hand side through the rounding step, because we assumed it to be fractional. Consequently, the constraint must be violated by the basic optimal solution corresponding to the tableau. Thus, the cutting plane we found, given by (8.2), indeed cuts off the computed basic optimal LP solution, and we can add it to the LP.