Friedrich Eisenbrand

Linear Optimization

These are notes of my course Discrete Optimization. They are constantly updated.

September 8, 2015

Contents

Chapter 1
Linear programming

We start by giving some examples of linear programs and how they are used in practice.

1.1 Softdrink production

Imagine that you own a company that produces the two softdrinks, *Spring* and *Nebsi*. These softdrinks are a mixture of *water*, an ingredient A and ingredient B. The recipes for Spring and Nebsi are different. Also, the profit for the two drinks is not the same. Those are as follows.

The use of ingredients A and B and the profit per $100l$ are as follows.

	A	B	Profit
Spring	$3l$	$8l$	100 CHF
Nebsi	$6l$	$4l$	125 CHF

While the supply of water is unlimited, your company has only $30l$ of ingredient A and $44l$ of ingredient B.

At the end of the production day, the local wholesaler picks up the drinks in two barrels. The capacity of the barrel for Spring is $500l$ while the barrel for Nebsi has a capacity of $400l$.

As the manager of your small company, your goal is to come up with a *production plan* that maximizes your profit. A production plan is a two-dimensional vector $(x_1, x_2) \in \mathbb{R}^2$ which means that you will produce $x_1 \cdot 100l$ of Spring and $x_2 \cdot 100l$ of Nebsi. A production plan is *feasible* if the produced drinks fit into the respective barrels and not more of A and B is used than what is on stock. Clearly, $(5, 4)$ is not a feasible production plan, as this would require $39l$ of ingredient A which exceeds the capacity.

A feasible production plan that maximizes your profit can be found with the help of a *linear program*, a central object of study in this course.

$$
\begin{aligned}
\text{max. } & 100 \cdot x_1 + 125 \cdot x_2 \\
\text{s.t.: } & \quad 3 \cdot x_1 + 6 \cdot x_2 \leqslant 30 \\
& \quad 8 \cdot x_1 + 4 \cdot x_2 \leqslant 44 \\
& \quad x_1 \leqslant 5 \\
& \quad x_2 \leqslant 4 \\
& \quad x_1 \geqslant 0 \\
& \quad x_2 \geqslant 0
\end{aligned}
\tag{1.1}
$$

One has to maximize a linear *objective function*, in this case $f(x_1, x_2) = 100 \cdot x_1 + 125 \cdot x_2$ where (x_1, x_2) satisfies linear inequalities. The linear inequalities $x_1 \geqslant 0$ and $x_2 \geqslant 0$ reflect the fact that only positive amounts can be produced, while $x_1 \leqslant 5$ reflects the barrel capacity for Spring. The linear inequality $3 \cdot x_1 + 6 \cdot x_2 \leqslant 30$ reflects the amount of $30l$ of ingredient A that is on stock.

We can now make a drawing of all feasible production plans.

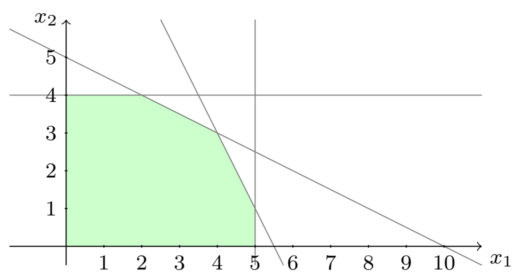


Fig. 1.1: The feasible production plans are the green area.

The set of points (x_1, x_2) that have objective function β is the line

$$\{(x_1, x_2) \in \mathbb{R}^2 \colon 100 \cdot x_1 + 125 \cdot x_2 = \beta\}$$

and our task is now to find the largest value for β such that the corresponding line still intersects the set of feasible production plans.

Figure 1.2 reveals that $(4, 3)$ is an optimal production plan and that the maximum profit that the manager can achieve is 775.

1.2 Proving optimality

How can the manager be convinced that $(4, 3)$ is an optimal production plan? Maybe he has made a mistake in his drawing or with his calculations and $(4, 3)$ is not optimal. What we will see now is a very important principle of linear programming. There is a simple way to prove optimality of solutions that we will explore later on.

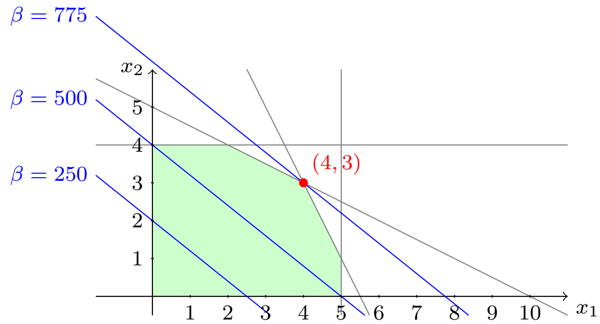


Fig. 1.2: The optimal production plan is $(4, 3)$.

Inspecting the drawing, one can see that there are two inequalities that $(4, 3)$ satisfies with equality, namely the inequalities

$$3 \cdot x_1 + 6 \cdot x_2 \leqslant 30 \tag{1.2}$$

$$8 \cdot x_1 + 4 \cdot x_2 \leqslant 44. \tag{1.3}$$

Clearly all feasible production plans satisfy these inequalities and inspecting Figure 1.2 it seems clear that $(4, 3)$ is an optimal solution of the optimization problem (1.1) where each linear inequality but the inequalities (1.2) and (1.3) have been removed.

What now follows is a very important technique that we will apply later on again in greater generality. Since each feasible production plan satisfies the inequalities (1.2) and (1.3) it satisfies also these inequalities, after they have been multiplied by $50/3$ and $25/4$ respectively. In fact the inequalities (1.2) and (1.3) are equivalent to the following two inequalities

$$50 \cdot x_1 + 100 \cdot x_2 \leqslant 500 \tag{1.4}$$

$$50 \cdot x_1 + 25 \cdot x_2 \leqslant 275. \tag{1.5}$$

By adding up the inequalities (1.4) and (1.5) we obtain the inequality

$$100 \cdot x_1 + 125 \cdot x_2 \leqslant 775 \tag{1.6}$$

which in turn is also satisfied by each feasible production plan. The left-hand-side of inequality (1.6) is the objective function and 775 is the value of the objective function evaluated at $(4, 3)$. Thus each feasible production plan yields a profit of at most 775 which is the profit yielded by $(4, 3)$. This shows that $(4, 3)$ is optimal.

1.3 Linear Programs

We use the following notation. For a matrix $A \in \mathbb{R}^{m\times n}$, $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,n\}$ we denote the i-th row of A by a_i and the j-th column of A by a^j. With A_{ij} we denote the element of A which is in the i-th row and j-th column of A. For a vector $v \in \mathbb{R}^m$ and $i \in \{1,\dots,m\}$ we denote the i-th element of v by v_i.

Definition 1.1. Let $A \in \mathbb{R}^{m\times n}$ be a matrix, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ be vectors and $I_\geqslant, I_\leqslant, I_= \subseteq \{1,\dots,m\}$ and $J_\geqslant, J_\leqslant \subseteq \{1,\dots,n\}$ be index sets. A *linear program (LP)* consists of

i) a linear *objective function*

$$\begin{gathered}\max c^T x\\ \text{or } \min c^T x\end{gathered}$$

ii) linear *constraints*

$$\begin{gathered}a_i^T x \geqslant b_i, i \in I_\geqslant\\ a_j^T x \leqslant b_j, j \in I_\leqslant\\ a_k^T x = b_k, k \in I_=\end{gathered}$$

iii) and *bounds on the variables*

$$\begin{gathered}x_j \geqslant 0,\ j \in J_\geqslant\\ x_j \leqslant 0,\ j \in J_\leqslant.\end{gathered}$$

Notice that we can re-write the objective function $\min c^T x$ as $\max -c^T x$. Similarly, the constraints $a_i^T x \geqslant b_i, i \in I_\geqslant$ are equivalent to the constrains $-a_i^T x \leqslant -b_i, i \in I_\geqslant$. Also the constraints $a_k^T x = b_k, k \in I_=$ can be replaced by the constraints $a_k^T x \leqslant b_k$, $-a_k^T x \leqslant -b_k$, $k \in I_=$. A lower bound $x_j \geqslant 0$ can be written as $-e_j^T x \leqslant 0$, where e_j is the j-th unit vector which has zeroes in every component, except for the j-th component, which is 1. Similarly an upper bound $x_j \leqslant 0$ can be written as $e_j^T x \leqslant 0$.

All-together, a linear program as in Definition 1.1 can always be written as

$$\max\{c^T x \colon \widetilde{A}x \leqslant \widetilde{b}, x \in \mathbb{R}^n\}$$

with a suitable matrix $\widetilde{A} \in \mathbb{R}^{m\times n}$ and a suitable vector $\widetilde{b} \in \mathbb{R}^m$. This representation has a name.

Definition 1.2. A linear program is in *inequality standard form*, if it is of the form

$$\max\{c^T x \colon Ax \leqslant b,\ x \in \mathbb{R}^n\}$$

for some matrix $A \in \mathbb{R}^{m\times n}$ and some vector $b \in \mathbb{R}^m$.

Definition 1.3. A point $x^* \in \mathbb{R}^n$ is called *feasible*, if x^* satisfies all constraints and bounds on the variables. If there are feasible solutions of a linear program, then the linear program is called *feasible* itself. A linear program is *bounded* if there exists a constant $M \in \mathbb{R}$ such for all feasible $x^* \in \mathbb{R}^n$ $c^T x^* \leqslant M$, if the linear program is a maximization problem and $c^T x^* \geqslant M$, if the linear program is a minimization problem. A feasible solution x^* is an optimal solution if $c^T x^* \geqslant c^T y^*$ for all feasible y^* if the linear program is a maximization problem and $c^T x^* \leqslant c^T y^*$ if the linear program is a minimization problem.

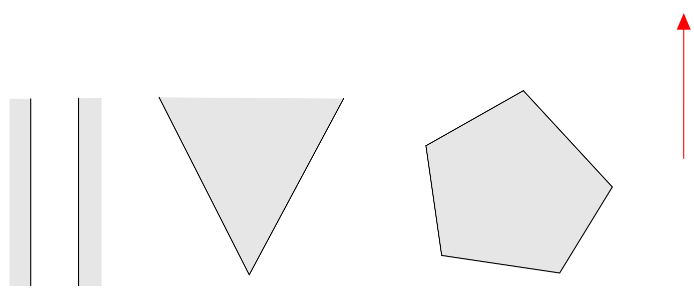


Fig. 1.3: With the objective function being to find the highest point, we have from left-to-right an infeasible linear program, an unbounded linear program and a bounded linear program.

We will see later that a feasible and bounded linear program has an optimal solution.

1.4 Fitting a line

The following is an example which is well known in statistics. Suppose that you measure points $(x_i, y_i) \in \mathbb{R}^2$ $i = 1, \ldots, n$ and you are interested in a linear function $y = a \cdot x + b$ that reflects the sample. One way to do that is by minimizing the expression

$$\sum_{i=1}^{n} (ax_i + b - y_i)^2, \tag{1.7}$$

where $a, b \in \mathbb{R}$ are the parameters of the line that we are looking for. The number $(ax_i + b - y_i)^2$ is the square of the vertical distance of the point (x_i, y_i) from the line $y = a\,x + b$.

Instead of using the method of least-squares, we could also minimize the following function, see also [12, Chapter 2.4],

$$\sum_{i=1}^{n} |ax_i + b - y_i|. \tag{1.8}$$

This objective has the advantage to be slightly more robust towards outliers. How can we model this as a linear program. The trick is to use an extra variable h_i which models the absolute value of $ax_i + b - y_i$.

$$\begin{array}{lcr} \min \sum_{i=1}^{n} h_i & & \\ h_i & \geqslant & ax_i + b - y_i,\ i = 1, \ldots, n \\ h_i & \geqslant & -(ax_i + b - y_i),\ i = 1, \ldots, n \end{array} \tag{1.9}$$

The variables of this linear program are h_i, $i = 1, \ldots, n$, a and b. For a fixed $a \in \mathbb{R}$ and $b \in \mathbb{R}$ the optimal h_i's will be $h_i = |ax_i + b - y_i|$ since the objective minimizes the sum of the h_i's. If one of the h_i's was strictly larger than $|ax_i + b - y_i|$, then the objective could be improved by making it smaller.

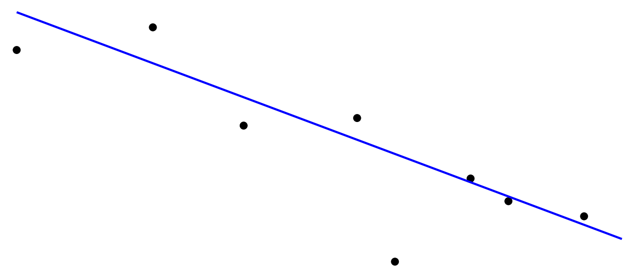


Fig. 1.4: A line that minimizes the sum of the vertical distances.

1.5 Linear Programming solvers and modeling languages

We will demonstrate now how to use a modeling language for linear programming and a linear programming solver to find a fitting line, as described in Section 1.4 for the points

$$(1, 3), (2.8, 3.3), (4, 2), (5.5, 2.1), (6, 0.2), (7, 1.3), (7.5, 1), (8.5, 0.8)$$

There are two popular formats for linear programming problems which are widely used by linear programming solvers, the *lp-format* and the *mps-format*. Both are not easy to read. To facilitate the modeling of a linear program, so-called modeling languages are used. We demonstrate the use of the popular open source modeling software called zimpl [7]. Below you see a way to model our fitting line linear program with zimpl:

```
set I := {1 to 8};
```

```
param X[I] :=  <1> 1, <2> 2.8, <3> 4  , <4> 5.5,
               <5> 6, <6> 7  , <7> 7.5, <8> 8.5 ;
param Y[I] :=  <1> 3  , <2> 3.3, <3> 2, <4> 2.1,
               <5> 0.2, <6> 1.3, <7> 1, <8> 0.8 ;
var h[I] >= -infinity <= infinity;
var a    >= -infinity <= infinity ;
var b    >= -infinity <= infinity ;

minimize cost: sum <i> in I: h[i];

subto c1: forall <i> in I:      h[i] >=   ( a * X[i] + b -Y[i]);
subto c2: forall <i> in I:      h[i] >= - ( a * X[i] + b -Y[i]);
```

Zimpl creates a linear program which is readable by linear programming solvers like QSopt or SoPlex.

1.6 Linear programming for longer OLED-lifetime

Organic Light Emitting Diodes (OLEDs) are considered as the display technology of the future and more and more commercial products are equipped with such displays as shown in Fig. 1.5. However, the cheapest OLED technology suffers from short lifetimes. We will show in this section how linear programming can be used to increase the lifetime of such displays.

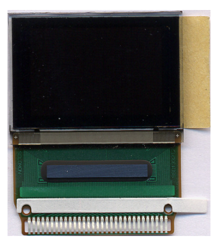


Fig. 1.5: Sample of a commercial OLED device with integrated driver chip

A (passive matrix) OLED display has a matrix structure consisting of n rows and m columns. At any crossover between a row and a column there is a vertical diode which works as a pixel. The image itself is given as an integral non-negative $n \times m$ matrix $(r_{ij}) \in [0, \ldots, \varrho]^{n \times m}$ representing its RGB values. Consider the contacts for the rows and columns as switches. For the time the switch of row i and column j is closed, an electrical current flows through the diode of pixel (i, j) and it shines. Hence, we can control the intensity of a pixel by the two quantities *electrical current* and *time*. The value r_{ij} determines the amount of time within the time frame in which the switches i and j have to be simultaneously closed. At a sufficient high frame rate e.g.

50 Hz, the perception by the eye is the average value of the light emitted by the pixel and one sees the image.

The traditional addressing scheme is row-by-row. This means that the switch for the first row is closed for a certain time while the switches for the columns are closed for the necessary amount of time dictated by the entries r_{1j}, $j = 1, \ldots, m$. Consequently the first row can be displayed in time $\max\{r_{1j} : j = 1, \ldots, m\}$. Then the second row is displayed and so on. With this addressing scheme, the pixels are idle most of the time and then have to shine with very high intensity. This puts the diodes under stress and is a major cause of the short lifetime of the displays.

How can this lifetime problem be dealt with? The main idea is to save time, or equivalently to lower the maximum intensity, by displaying several rows at once.

Consider the schematic image on the left of Fig. 1.6. Let us compute the amount of time which is necessary to display the image with this addressing scheme. The maximum value of the entries in the first row is 238. This is the amount of time which is necessary to display the first row. After that the second row is displayed in time 237. In total the time which is required to display the image is $238 + 237 + 234 + 232 + 229 = 1170$ time units.

$$\begin{matrix} 109 & 238 & 28 \\ 112 & 237 & 28 \\ 150 & 234 & 25 \\ 189 & 232 & 22 \\ 227 & 229 & 19 \end{matrix} = \begin{matrix} 0 & 82 & 25 \\ 0 & 82 & 25 \\ 0 & 41 & 22 \\ 0 & 41 & 22 \\ 0 & 0 & 0 \end{matrix} + \begin{matrix} 0 & 0 & 0 \\ 112 & 155 & 3 \\ 112 & 155 & 3 \\ 189 & 191 & 0 \\ 189 & 191 & 0 \end{matrix} + \begin{matrix} 109 & 156 & 3 \\ 0 & 0 & 0 \\ 38 & 38 & 0 \\ 0 & 0 & 0 \\ 38 & 38 & 19 \end{matrix}$$

Fig. 1.6: An example decomposition

Now consider the decomposition of the image as the sum of the three images on the right of Fig. 1.6. In the first image, each odd row is equal to its even successor. This means that we can close the switches for rows 1 and 2 simultaneously, and these two equal rows are displayed in 82 time units. Rows 3 and 4 can also be displayed simultaneously which shows that the first image on the right can be displayed in $82 + 41$ time units. The second image on the right can be displayed in $155 + 191$ time units while the third image has to be displayed traditionally. In total all three images, and thus the original image on the left via this decomposition, can be displayed in $82 + 41 + 155 + 191 + 156 + 38 + 38 = 701$ time units. This means that we could reduce the necessary time via this decomposition by roughly 40%. We could equally display the image in the original 1170 time units but reduce the peak intensity, or equally the maximum electrical current through a diode by roughly 40%.

We now show how to model the time-optimal decomposition of an image as a linear program. To decompose R we need to find matrices $F^{(1)} = (f_{ij}^{(1)})$ and $F^{(2)} = (f_{ij}^{(2)})$ where $F^{(1)}$ represents the singleline part and $F^{(2)}$ the two doubleline parts. More precisely, the i-th row of matrix $F^{(2)}$ represents the doubleline covering rows i and $i+1$. Since the overlay (addition) of the subframes must be equal to the original image to get a valid decomposition of R, the matrices $F^{(1)}$ and $F^{(2)}$ must fulfill the constraint $f_{ij}^{(1)} + f_{i-1,j}^{(2)} + f_{ij}^{(2)} = r_{ij}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, where we now and in the following use the convention to simply omit terms with indices running out of bounds. Since we cannot produce "negative" light we require also non-negativity of the variables $f_{ij}^{(\alpha)} \geqslant 0$. The goal is to find an integral decomposition that minimizes

$$\sum_{i=1}^{n} \max\{f_{ij}^{(1)} : 1 \leqslant j \leqslant m\} + \sum_{i=1}^{n-1} \max\{f_{ij}^{(2)} : 1 \leqslant j \leqslant m\} \ .$$

This problem can be formulated as a linear program by replacing the objective by $\sum_{i=1}^{n} u_i^{(1)} + \sum_{i=1}^{n-1} u_i^{(2)}$ and by adding the constraints $f_{ij}^{(\alpha)} \leqslant u_i^{(\alpha)}$. This yields

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} u_i^{(1)} + \sum_{i=1}^{n-1} u_i^{(2)} & & \\
\text{s.t.} \quad & f_{ij}^{(1)} + f_{i-1,j}^{(2)} + f_{ij}^{(2)} = r_{ij} & \quad \text{for all } i, j & \qquad (1.10)\\
& f_{ij}^{(\alpha)} \leqslant u_i^{(\alpha)} & \quad \text{for all } i, j, \alpha & \qquad (1.11)\\
& f_{ij}^{(\alpha)} \in \mathbb{R}_{\geqslant 0} & \quad \text{for all } i, j, \alpha &
\end{aligned}$$

Note that the objective does not contain the f-variables. By decomposing images like this, the average lifetime of an OLED display can be increased by roughly 100%, see [3].

Exercises

1) A company produces and sells two different products. Our goal is to determine the number of units of each product they should produce during one month, assuming that there is an unlimited demand for the products, but there are some constraints on production capacity and budget.
   There are 20000 hours of machine time in the month. Producing one unit takes 3 hours of machine time for the first product and 4 hours for the second product. Material and other costs for producing one unit of the first product amount to 3CHF, while producing one unit of the second

product costs 2CHF. The products are sold for 6CHF and 5CHF per unit, respectively. The available budget for production is 4000CHF initially. 25% of the income from selling the first product can be used immediately as additional budget for production, and so can 28% of the income from selling the second product.

a. Formulate a linear program to maximize the profit subject to the described constraints.
b. Solve the linear program graphically by drawing its set of feasible solutions and determining an optimal solution from the drawing.
c. Suppose the company could modernize their production line to get an additional 2000 machine hours for the cost of 400CHF. Would this investment pay off?

2) A factory produces two different products. To create one unit of product 1, it needs one unit of raw material A and one unit of raw material B. To create one unit of product 2, it needs one units of raw material B and two units of raw material C. Raw material B needs preprocessing before it can be used, which takes one minute per unit. At most 20 hours of time is available per day for the preprocessing. Raw materials of capacity at most 1200 can be delivered to the factory per day. One unit of raw material A, B and C has size 4, 3 and 2 respectively.
At most 130 units of the first and 100 units of the second product can be sold per day. The first product sells for 6 CHF per unit and the second one for 9 CHF per unit.
Formulate the problem of maximizing turnover as a linear program in two variables and solve it.
3) Prove the following statement or give a counterexample: The set of optimal solutions of a linear program is always finite.
4) Let (1.12) be a linear program in inequality standard form, i.e.

$$\max\{c^T x \mid Ax \leqslant b, x \in \mathbb{R}^n\} \tag{1.12}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$.
Prove that there is an equivalent linear program (1.13) of the form

$$\max\{\tilde{c}^T x \mid \tilde{A}x = \tilde{b}, x \geqslant 0, x \in \mathbb{R}^{\tilde{n}}\} \tag{1.13}$$

where $\tilde{A} \in \mathbb{R}^{\tilde{m} \times \tilde{n}}$, $\tilde{b} \in \mathbb{R}^{\tilde{m}}$, and $\tilde{c} \in \mathbb{R}^{\tilde{n}}$ are such that every feasible point of (1.12) corresponds to a feasible point of (1.13) with the same objective function value and vice versa.
Linear programs of the form in (1.13) are said to be in *equality standard form*.

5) Model the linear program (1.10) to decompose the EPFL logo with Zimpl. An incomplete model containing the encoding of the grayscale values of the logo can be found here[1].
Use an LP solver library of your choice to compute an optmal solution.

[1] http://disopt.epfl.ch/webdav/site/disopt/users/190205/public/logo_dec.zmpl

Chapter 2
Polyhedra and convex sets

Definition 2.1. A polyhedron $P \subseteq \mathbb{R}^n$ is a set of the form $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ for some $A \in \mathbb{R}^{m \times n}$ and some $b \in \mathbb{R}^m$.

We are interested in polyhedra, since the set of feasible solutions of a linear program $\max\{c^T x \colon Ax \leqslant b\}$ is a polyhedron.

Example 2.1. Consider again the soft-drink production problem from chapter 1.1. The corresponding set of feasible solutions is the polyhedron $P = \{x \in \mathbb{R}^2 \colon Ax \leqslant b\}$ with

$$A = \begin{pmatrix} 3 & 6 \\ 8 & 4 \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad \text{and} \quad b = \begin{pmatrix} 30 \\ 44 \\ 5 \\ 4 \\ 0 \\ 0 \end{pmatrix}.$$

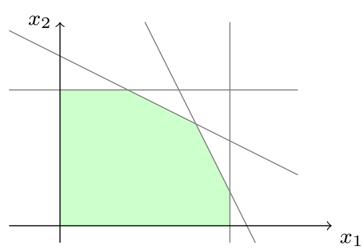


Fig. 2.1: The polyhedron of feasible solutions of the linear program (1.1).

Definition 2.2. A set $K \subseteq \mathbb{R}^n$ is *convex* if for each $u, v \in K$ and $\lambda \in [0, 1]$ the point $\lambda u + (1 - \lambda)v$ is also contained in K.

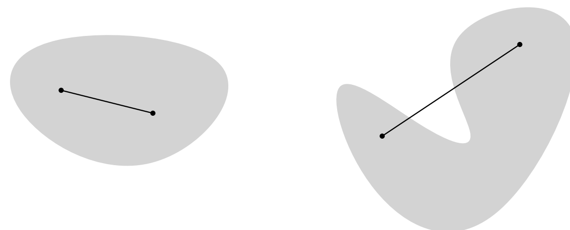


Fig. 2.2: The set on the left is convex, the set on the right is non-convex.

A *halfspace* is a set of solutions of one inequality $a^T x \leqslant \beta$ where $a \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$, i.e., a set of the form

$$\{x \in \mathbb{R}^n \colon a^T x \leqslant \beta\}.$$

A *hyperplane* a set of the form

$$\{x \in \mathbb{R}^n \colon a^T x = \beta\}.$$

It is easy to see that a halfspace is convex. Convexity is also maintained if convex sets are intersected.

Proposition 2.1. *Let I be an index set atd $C_i \subseteq \mathbb{R}^n$ be convex sets for each $i \in I$, then $\cap_{i \in I} C_i$ is a convex set.*

Consequently, the set of feasible solutions of a linear program $\{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ is a convex set. This is our motivation to study properties of convex sets.

2.1 Extreme points and vertices

Definition 2.3. An inequality $a^T x \leqslant \beta$ is *valid* for a set $K \subseteq \mathbb{R}^n$ if each $x^* \in K$ satisfies $a^T x^* \leqslant \beta$.

Definition 2.4. Let $K \subseteq \mathbb{R}^n$ be a convex set. A point $x^* \in K$ is an *extreme point* or *vertex* of K if there exists a valid inequality $a^T x \leqslant \beta$ of K such that

$$\{x^*\} = K \cap \{x \in \mathbb{R}^n \colon a^T x = \beta\}.$$

In other words, if x^* is the only point of K that satisfies the valid inequality with equality.

We can now characterize the extreme points of polyhedra. In fact, there are only finitely many of them.

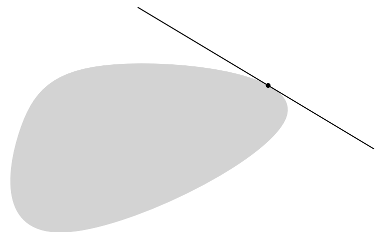


Fig. 2.3: An extreme point of a convex set.

Theorem 2.1. *Let $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ be a polyhedron. A feasible point x^* is an extreme point of P if and only if there is a sub-system $A'x \leqslant b'$ of $Ax \leqslant b$ such that*

i) x^* satisfies all inequalities of $A'x \leqslant b'$ with equality.
ii) A' has n rows and A' is non-singular.

Proof. Let $A'x \leqslant b'$ be such a sub-system and consider the valid inequality $\mathbf{1}^T A'x \leqslant \mathbf{1}^T b'$. Clearly x^* satisfies this inequality with equality. Any $y^* \in P$ that satisfies this inequality with equality must satisfy $A'x = b'$. Since A' is non-singular, x^* is the unique solution of $A'x = b'$ which means that x^* is the unique point of P that satisfies $\mathbf{1}^T A'x \leqslant \mathbf{1}^T b'$ with equality.

Assume now that there does not exist a sub-system $A'x \leqslant b'$ of $Ax \leqslant b$ with properties i) and ii). Denote the sub-system of inequalities that are satisfied by x^* with equality by $\widetilde{A}x \leqslant \widetilde{b}$. Then $\text{rank}(\widetilde{A}) < n$ and there exists a $d \neq 0 \in \mathbb{R}^n$ with $\widetilde{A}d = 0$. Consequently there exists an $\varepsilon > 0$ such that $x^* \pm \varepsilon \cdot d \in P$.

Clearly, any inequality that is satisfied by x^* with equality and that is satisfied by $x^* \pm \varepsilon d$ is satisfied by $x^* \pm \varepsilon d$ with equality as well. This implies that x^* is not an extreme point.

The relevance of vertices for linear programming is reflected in the following theorem.

Theorem 2.2. *If a linear program $\max\{c^T x : x \in \mathbb{R}^n, Ax \leqslant b\}$ is feasible and bounded and if $\text{rank}(A) = n$, then the linear program has an optimal solution that is an extreme point.*

Proof. We use the following notation. If x^* is a feasible solution then $A_{x^*}x \leqslant b_{x^*}$ is the subsystem of $Ax \leqslant b$ that is satisfied by x^* with equality. The rank of x^*, $\text{rank}(x^*)$ is the rank of A_{x^*}. The following claim implies the assertion.

> If x^* is feasible and $\text{rank}(x^*) < n$, then there exists a y^* with $c^T y^* \geqslant c^T x^*$ and $\text{rank}(y^*) > \text{rank}(x^*)$.

To prove this, let $d \neq 0 \in \mathbb{R}^n$ be a vector with $A_{x^*}d = 0$. We can assume $c^T d \geqslant 0$ by switching to $-d$ otherwise.

If $c^T d > 0$, then consider the points $x^* + \lambda d$ with $\lambda \geqslant 0$ and let λ_{max} be maximal with the corresponding point feasible. Clearly $y^* = x^* + \lambda_{max} d$ satisfies the condition of the claim.

Suppose now that $c^T d = 0$. Then $Ad \neq 0$ since $\operatorname{rank}(A) = n$. Let λ_{max} be the maximum of the set $\{\lambda \geqslant 0 \colon A(x^* \pm \lambda d) \leqslant b\}$. Then $y^* = x^* + \lambda_{max} d$ or $y^* = x^* - \lambda_{max} d$ satisfies the condition of the claim.

2.2 Linear, affine, conic and convex hulls

We now describe convex sets that are generated by a set $X \subseteq \mathbb{R}^n$ of n-dimensional vectors. The *linear hull*, *affine hull*, *conic hull* and *convex hull* of X are defined as follows.

$$\begin{aligned}
\text{lin.hull}(X) = \{&\lambda_1 x_1 + \cdots + \lambda_t x_t \mid t \geqslant 0, \\
&x_1, \dots, x_t \in X,\ \lambda_1, \dots, \lambda_t \in \mathbb{R}\} && (2.1) \\
\text{affine.hull}(X) = \{&\lambda_1 x_1 + \cdots + \lambda_t x_t \mid t \geqslant 1, \\
&x_1, \dots, x_t \in X,\ \sum_{i=1}^t \lambda_i = 1,\ \lambda_1, \dots, \lambda_t \in \mathbb{R}\} && (2.2) \\
\text{cone}(X) = \{&\lambda_1 x_1 + \cdots + \lambda_t x_t \mid t \geqslant 0, \\
&x_1, \dots, x_t \in X,\ \lambda_1, \dots, \lambda_t \in \mathbb{R}_{\geqslant 0}\} && (2.3) \\
\text{conv}(X) = \{&\lambda_1 x_1 + \cdots + \lambda_t x_t \mid t \geqslant 1, \\
&x_1, \dots, x_t \in X,\ \sum_{i=1}^t \lambda_i = 1,\ \lambda_1, \dots, \lambda_t \in \mathbb{R}_{\geqslant 0}\} && (2.4)
\end{aligned}$$

Proposition 2.2. *Let $X \subseteq \mathbb{R}^n$ and $x_0 \in X$. One has*

$$\text{affine.hull}(X) = x_0 + \text{lin.hull}(X - x_0),$$

where for $u \in \mathbb{R}^n$ and $V \subseteq \mathbb{R}^n$, $u + V$ denotes the set $u + V = \{u + v \mid v \in V\}$.

Proof. We first show that each $x \in \text{affine.hull}(X)$ is also an element of the set $x_0 + \text{lin.hull}(X - x_0)$ and then we show that each point $x \in x_0 + \text{lin.hull}(X - x_0)$ is also an element of $\text{affine.hull}(X)$.

Let $x \in \text{affine.hull}(X)$,i.e., there exists a natural number $t \geqslant 1$ and $\lambda_1, \dots, \lambda_t \in \mathbb{R}$, with $x = \lambda_1 x_1 + \cdots + \lambda_t x_t$ and $\sum_{i=1}^t \lambda_i = 1$. Now

$$\begin{aligned}
x &= x_0 - x_0 + \lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_t x_t \\
&= x_0 - \lambda_1 x_0 - \cdots - \lambda_t x_0 + \lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_t x_t \\
&= x_0 + \lambda_1 (x_1 - x_0) + \cdots + \lambda_t (x_t - x_0),
\end{aligned}$$

which shows that $x \in x_0 + \text{lin.hull}(X - x_0)$.

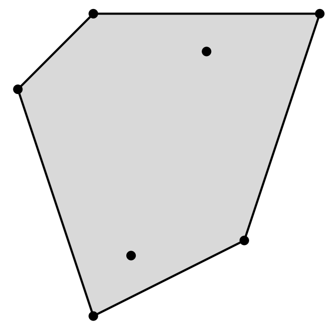


Fig. 2.4: The convex hull of 7 points in $\mathbb{R}^2$.

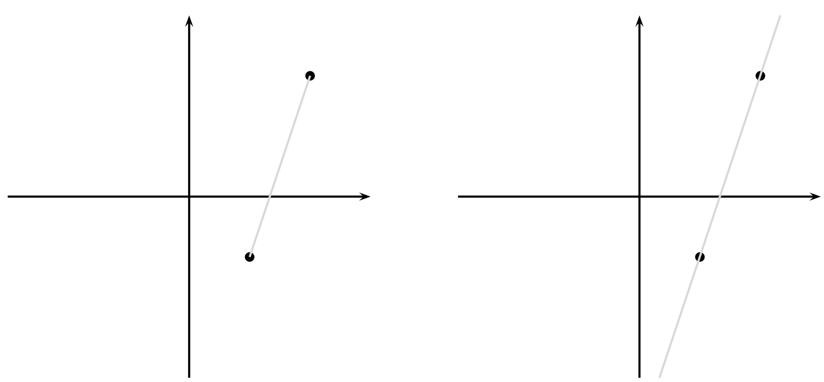


Fig. 2.5: Two points with their convex hull on the left and their affine hull on the right.

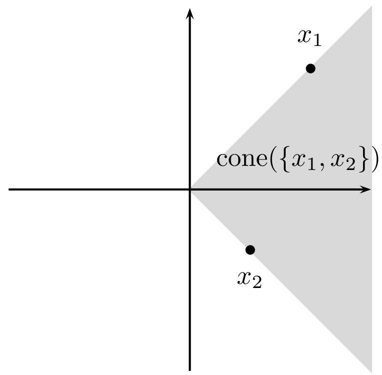


Fig. 2.6: Two points with their conic hull

Suppose now that $x \in x_0 + \text{lin.hull}(X - x_0)$. Then there exist $\lambda_1, \ldots, \lambda_t \in \mathbb{R}$ with $x = x_0 + \lambda_1(x_1 - x_0) + \cdots + \lambda_t(x_t - x_0)$. With $\lambda_0 = 1 - \sum_{i=1}^t \lambda_i$ one has $\sum_{i=0}^t \lambda_i = 1$ and

$$\begin{aligned} x &= x_0 + \lambda_1(x_1 - x_0) + \cdots + \lambda_t(x_t - x_0) \\ &= \lambda_0 x_0 + \cdots + \lambda_t x_t \end{aligned}$$

and thus that $x \in \text{affine.hull}(X)$. □

Theorem 2.3. *Let $X \subseteq \mathbb{R}^n$ be a set of points. The convex hull,* conv(X)*, of X is convex.*

Proof. Let u and v be points in conv(X). This means that there exists a natural number $t \geqslant 1$, real numbers $\alpha_i, \beta_i \geqslant 0$, and points $x_i \in X$, $i = 1, \ldots, t$ with $\sum_{i=1}^t \alpha_i = \sum_{i=1}^t \beta_i = 1$ with $u = \sum_{i=1}^t \alpha_i x_i$ and $v = \sum_{i=1}^t \beta_i x_i$. For $\lambda \in [0, 1]$ one has $\lambda\alpha_i + (1-\lambda)\beta_i \geqslant 0$ for $i = 1, \ldots, t$ and $\sum_{i=1}^t (\lambda\alpha_i + (1-\lambda)\beta_i) = 1$. This shows that

$$\lambda u + (1 - \lambda)v = \sum (\lambda_i \alpha_i + (1 - \lambda_i)\beta_i)\, x_i \in \text{conv}(X),$$

and therefore that conv(X) is convex. □

Theorem 2.4. *Let $X \subseteq \mathbb{R}^n$ be a set of points. Each convex set K containing X also contains* conv(X)*.*

Proof. Let K be a convex set containing X, and let $x_1, \ldots, x_t \in X$ and $\lambda_i \in \mathbb{R}$ with $\lambda_i \geqslant 0$, $i = 1, \ldots, t$ and $\sum_{i=1}^t \lambda_i = 1$. We need to show that $u = \sum_{i=1}^t \lambda_i x_i$ is contained in K. This is true for $t \leqslant 2$ by the definition of convex sets.

We argue by induction. Suppose that $t \geqslant 3$. If one of the λ_i is equal to 0, then one can represent u as a convex combination of $t-1$ points in X and, by induction, $u \in K$. If $t \geqslant 3$, each $\lambda_i > 0$ and $\sum_{i=1}^t \lambda_i = 1$, then one has $0 < \lambda_i < 1$ for $i = 1, \dots, t$ and thus we can write

$$u = \lambda_1 x_1 + (1 - \lambda_1) \sum_{i=2}^{t} \frac{\lambda_i}{1 - \lambda_1} x_i.$$

One has $\lambda_i/(1-\lambda_1) > 0$ and

$$\sum_{i=2}^{t} \frac{\lambda_i}{1 - \lambda_1} = 1,$$

which means that the point $\sum_{i=2}^t \frac{\lambda_i}{1-\lambda_1} x_i$ is in K by induction. Again, by the definition of convex sets, we conclude that u lies in K. □

Theorem 2.4 implies that $\operatorname{conv}(X)$ is the intersection of all convex sets containing X, i.e.,

$$\operatorname{conv}(X) = \bigcap_{\substack{K \supseteq X \\ K \text{ convex}}} K.$$

Definition 2.5. A set $C \subseteq \mathbb{R}^n$ is a *cone*, if it is convex and for each $c \in C$ and each $\lambda \in \mathbb{R}_{\geqslant 0}$ one has $\lambda \cdot c \in C$.

Similarly to Theorem 2.3 and Theorem 2.4 one proves the following.

Theorem 2.5. *For any $X \subseteq \mathbb{R}^n$, the set* $\operatorname{cone}(X)$ *is a cone.*

Theorem 2.6. *Let $X \subseteq \mathbb{R}^n$ be a set of points. Each cone containing X also contains* $\operatorname{cone}(X)$.

These theorems imply that $\operatorname{cone}(X)$ is the intersection of all cones containing X, i.e.,

$$\operatorname{cone}(X) = \bigcap_{\substack{C \supseteq X \\ C \text{ is a cone}}} C.$$

2.3 Radon's lemma and Carathéodory's theorem

Theorem 2.7 (Radon's lemma). *Let $A \subseteq \mathbb{R}^n$ be a set of $n+2$ points. There exist disjoint subsets $A_1, A_2 \subseteq A$ with*

$$\operatorname{conv}(A_1) \cap \operatorname{conv}(A_2) \neq \emptyset.$$

Proof. Let $A = \{a_1, \dots, a_{n+2}\}$. We embed these points into $\mathbb{R}^{n+1}$ by appending a 1 in the $n+1$-st component, i.e., we construct

$$A' = \left\{ \binom{a_1}{1}, \dots, \binom{a_{n+2}}{1} \right\} \subseteq \mathbb{R}^{n+1}.$$

The set A' consists of $n+2$ vectors in $\mathbb{R}^{n+1}$. Those vectors are linearly dependent. Let

$$0 = \sum_{i=1}^{n+2} \lambda_i \binom{a_i}{1} \tag{2.5}$$

be a nontrivial linear representation of 0, i.e., not all λ_i are 0. Furthermore, let $P = \{i \colon \lambda_i \geqslant 0,\, i = 1, \dots, n+2\}$ and $N = \{i \colon \lambda_i < 0,\, i = 1, \dots, n+2\}$. We claim that

$$\operatorname{conv}(\{a_i \colon i \in P\}) \cap \operatorname{conv}(\{a_i \colon i \in N\}) \neq \emptyset.$$

It follows from (2.5) and the fact that the $n+1$-st component of the vectors is 1 that $\sum_{i \in P} \lambda_i = -\sum_{i \in N} \lambda_i = s > 0$. It follows also from (2.5) that

$$\sum_{i \in P} \lambda_i a_i = \sum_{i \in N} -\lambda_i a_i.$$

The point $u = \sum_{i \in P} (\lambda_i / s) \cdot a_i = \sum_{i \in N} (-\lambda_i / s) a_i$ is contained in $\operatorname{conv}(\{a_i \colon i \in P\}) \cap \operatorname{conv}(\{a_i \colon i \in N\})$, implying the claim. □

Theorem 2.8 (Carathéodory's theorem). *Let $X \subseteq \mathbb{R}^n$, then for each $x \in \operatorname{cone}(X)$ there exists a set $\widetilde{X} \subseteq X$ of cardinality at most n such that $x \in \operatorname{cone}(\widetilde{X})$. The vectors in $\widetilde{X}$ are linearly independent.*

Proof. Let $x \in \operatorname{cone}(X)$, then there exist $t \in \mathbb{N}_+$, $x_i \in X$ and $\lambda_i \geqslant 0$, $i = 1, \dots, t$, with $x = \sum_{i=1}^t \lambda_i x_i$. Suppose that $t \in \mathbb{N}_+$ is minimal such that x can be represented as above. We claim that $t \leqslant n$. If $t \geqslant n+1$, then the x_i are linearly dependent. This means that there are $\mu_i \in \mathbb{R}$, not all equal to 0 with

$$\sum_{i=1}^t \mu_i x_i = 0. \tag{2.6}$$

By multiplying each μ_i in (2.6) with -1 if necessary, we can assume that at least one of the μ_i is strictly larger than 0. One has for each $\varepsilon \in \mathbb{R}$

$$x = \sum_{i=1}^t (\lambda_i - \varepsilon \cdot \mu_i) x_i. \tag{2.7}$$

What is the largest $\varepsilon^* > 0$ that we can pick for ε such that (2.7) is still a conic combination? We need to have

$$\lambda_i - \varepsilon \cdot \mu_i \geqslant 0, \ \text{ for each } i \in \{1, \dots, t\}. \tag{2.8}$$

Let J be the set of indices $J = \{j : j \in \{1, \dots, t\},\, \mu_j > 0\}$. We observed that we can assume $J \neq \emptyset$. We have (2.8) as long as

$$\varepsilon \leqslant \lambda_j/\mu_j \text{ for each } j \in J. \tag{2.9}$$

This means that $\varepsilon^* = \min\{\lambda_j/\mu_j : j \in J\}$. Let $j^* \in J$ be an index where this minimum is attained. Since $\lambda_i - \varepsilon^* \cdot \mu_i \geqslant 0$ for all $i = 1, \dots, t$ and since $\lambda_{j^*} - \varepsilon^* \cdot \mu_{j^*} = 0$, we have $x \in \operatorname{cone}(\{x_1, \dots, x_t\} \setminus \{x_{j^*}\})$, which is a contradiction to the minimality of t. □

Corollary 2.1 (Carathéodory's theorem for convex hulls). *Let $X \subseteq \mathbb{R}^n$, then for each $x \in \operatorname{conv}(X)$ there exists a set $\widetilde{X} \subseteq X$ of cardinality at most $n+1$ such that $x \in \operatorname{conv}(\widetilde{X})$.*

2.4 Separation theorem and Farkas' lemma

We recall a basic fact from analysis, see, e.g. [11, Theorem 4.4.1].

Theorem 2.9. *Let $X \subseteq \mathbb{R}^n$ be compact and $f : X \to \mathbb{R}$ be continuous. Then f is bounded and there exist points $x_1, x_2 \in X$ with $f(x_1) = \sup\{f(x) : x \in X\}$ and $f(x_2) = \inf\{f(x) : x \in X\}$.*

Theorem 2.10. *Let $K \subseteq \mathbb{R}^n$ be a closed convex set and $x^* \in \mathbb{R}^n \setminus K$, then there exists an inequality $a^T x \geqslant \beta$ such that $a^T y > \beta$ holds for all $y \in K$ and $a^T x^* < \beta$.*

Proof. Since the mapping $f(x) = \|x^* - x\|$ is continuous and since for any $k \in K$, $K \cap \{x \in K : \|x^* - x\| \leqslant \|x^* - k\|\}$ is compact, there exists a point $k^* \in K$ with minimal distance to x^*. Consider the midpoint $m = 1/2(k^* + x^*)$ on the line-segment $\overline{k^* x^*}$ and the hyperplane $a^T x = \beta$ with $\beta = a^T m$ and $a = (k^* - x^*)$. Clearly, $a^T x^* = \beta - 1/2\|k^* - x^*\|^2$ and $a^T k^* = \beta + 1/2\|k^* - x^*\|^2$. Suppose that there exists a $k' \in K$ with $a^T k' \leqslant \beta$. The points $\lambda k^* + (1 - \lambda)k'$, $\lambda \in [0, 1]$ are in K by the convexity of K, thus we can also assume that k' lies on the hyperplane, i.e., $a^T k' = \beta$. This means that there exists a vector x' which is orthogonal to a and $k' = m + x'$. The distance squared of a point $\lambda k^* + (1 - \lambda)k'$ with $\lambda \in [0, 1]$ to m is, by Pythagoras equal to

$$\lambda^2 \|\frac{1}{2} a\|^2 + (1 - \lambda)^2 \|x'\|^2.$$

As a function of λ, this is increasing at $\lambda = 1$. Thus there exists a point on the line-segment $\lambda x^* + (1 - \lambda)k'$ which is closer to m than k^*. This point is also closer to x^* than k^*, which is a contradiction. Therefore $a^T k > \beta$ for each $k \in K$. □

Theorem 2.11 (Farkas' lemma). *Let $A \in \mathbb{R}^{m\times n}$ be a matrix and $b \in \mathbb{R}^m$ be a vector. The system $Ax = b$, $x \geqslant 0$ has a solution if and only if for all $\lambda \in \mathbb{R}^m$ with $\lambda^T A \geqslant 0$ one has $\lambda^T b \geqslant 0$.*

Proof. Suppose that $x^* \in \mathbb{R}^n_{\geqslant 0}$ satisfies $Ax^* = b$ and let $\lambda \in \mathbb{R}^m$ with $\lambda^T A \geqslant 0$. Then $\lambda^T b = \lambda^T A x^* \geqslant 0$, since $\lambda^T A \geqslant 0$ and $x^* \geqslant 0$.

Now suppose that $Ax = b$, $x \geqslant 0$ does not have a solution. Then, with $X \subseteq \mathbb{R}^n$ being the set of column vectors of A, b is not in $\mathrm{cone}(X)$. The set $\mathrm{cone}(X)$ is convex and closed, see exercise 5. Theorem 2.10 implies that there is an inequality $\lambda^T x \geqslant \beta$ such that $\lambda^T y > \beta$ for each $y \in \mathrm{cone}(X)$ and $\lambda^T b < \beta$. Since for each $a \in X$ and each $\mu \geqslant 0$ one has $\mu \cdot a \in \mathrm{cone}(X)$ and thus $\lambda^T(\mu \cdot a) > \beta$, it follows that $\lambda^T a \geqslant 0$ for each $a \in X$. Furthermore, since $0 \in \mathrm{cone}(X)$ it follows that $0 \geqslant \beta$ and thus that $\lambda^T b < 0$.

2.5 Decomposition theorem for polyhedra

In the following we use the notation $P(A,b) = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ for the polyhedron that is defined by $Ax \leqslant b$. We prove the Minkowski-Weyl theorem in this section that shows that polyhedra can be decomposed into the Minkowski sum of a polytope and a cone.

Definition 2.6. An inequality $a^T x \leqslant \beta$ is called an *implicit equality* of $Ax \leqslant b$ if each $x^* \in P(A,b)$ satisfies $a^T x^* = \beta$. We denote the subsystem consisting of implicit equalities of $Ax \leqslant b$ by $A^= x \leqslant b^=$ and the subsystem consisting of the other inequalities by $A^\leqslant x \leqslant b^\leqslant$. A constraint is *redundant* if its removal from $Ax \leqslant b$ does not change the set of feasible solution of $Ax \leqslant b$.

In the following, a vector x satisfies $Ax < b$ if and only if $a_i^T x < b_i$ for all $1 \leqslant i \leqslant m$, where $a_1,\ldots,a_m$ are the rows of A.

Lemma 2.1. *Let $P(A,b)$ be a non-empty polyhedron. Then there exists an $x \in P(A,b)$ with $A^\leqslant x < b^\leqslant$.*

Proof. Suppose that the inequalities in $A^\leqslant x \leqslant b^\leqslant$ are $a_1^T x \leqslant \beta_1, \ldots, a_k^T x \leqslant \beta_k$. For each $1 \leqslant i \leqslant k$ there exists an $x_i \in P$ with $a_i^T x_i < \beta_i$. Thus the point $x = 1/k(x_1 + \cdots + x_k)$ is a point of $P(A,b)$ satisfying $A^\leqslant x < b^\leqslant$.

Lemma 2.2. *Let $Ax \leqslant b$ be a system of inequalities. One has*

$$\mathrm{affine.hull}(P(A,b)) = \{x \in \mathbb{R}^n \mid A^= x = b^=\} = \{x \in \mathbb{R}^n \mid A^= x \leqslant b^=\}.$$

Proof. Let $x_1, \ldots, x_t \in P(A,b)$ and suppose that $a^T x \leqslant \beta$ is an implicit equality. Then since $a^T x_i = \beta$ one has $a^T(\sum_{j=1}^t \lambda_i x_i) = \beta$. Therefore the inclusions $\subseteq$ follow.

Suppose now that x_0 satisfies $A^= x \leqslant b^=$. Let $x_1 \in P(A,b)$ with $A^\leqslant x_1 < b^\leqslant$. If $x_0 = x_1$ then $x_0 \in P(A,b) \subseteq \mathrm{affine.hull}(P(A,b))$. Otherwise the line segment between x_0 and x_1 contains more than one point in P and thus $x_0 \in$ affine.hull(P).

A nonempty set $C \subseteq \mathbb{R}^n$ is a *cone* if $\lambda x + \mu y \in C$ for each $x, y \in C$ and $\lambda, \mu \in \mathbb{R}_{\geqslant 0}$. A cone C is *polyhedral* if $C = \{x \in \mathbb{R}^n \mid Ax \leqslant 0\}$. A cone *generated by* vectors $x_1, \dots, x_m \in \mathbb{R}^n$ is a set of the form $C = \{\sum_{i=1}^m \lambda_i x_i \mid \lambda_i \in \mathbb{R}_{\geqslant 0},\ i = 1, \dots, m\}$. A point $x = \sum_{i=1}^m \lambda_i x_i$ with $\lambda_i \in \mathbb{R}_{\geqslant 0}$, $i = 1, \dots, m$ is called a *conic combination* of the $x_1, \dots, x_m$. The set of conic combinations of X is denoted by cone(X).

Theorem 2.12 (Farkas-Minkowsi-Weyl theorem). *A convex cone is polyhedral if and only if it is finitely generated.*

Proof. Suppose that $a_1, \dots, a_m$ span $\mathbb{R}^n$ and consider the cone $C = \{\sum_{i=1}^m \lambda_i a_i \mid \lambda_i \geqslant 0,\ i = 1, \dots, m\}$. Let $b \notin C$. Then the system $A\lambda = b$, $\lambda \geqslant 0$ has no solution. By Theorem 2.11 (Farkas' lemma), this implies that there exists a $y \in \mathbb{R}^n$ such that $A^T y \leqslant 0$ and $b^T y > 0$.

Suppose that the columns of A which correspond to inequalities in $A^T y \leqslant 0$ that are satisfied by y with equality have rank $< n-1$. Denote these columns by $a_{i_1}, \dots, a_{i_k}$. Then there exists a $v \neq 0$ which is orthogonal to each of these columns and to b, i.e., $a_{i_j}^T v = 0$ for each $j = 1, \dots, k$ and $b^T v = 0$. There also exists a column a^* of A which is not in the set $\{a_{i_1}, \dots, a_{i_k}\}$ such that $(a^*)^T v > 0$ since the columns of A span $\mathbb{R}^n$. Therefore there exists an $\epsilon > 0$ such that

i) $A^T(y + \epsilon \cdot v) \leqslant 0$
ii) The subspace generated by the columns of A which correspond to inequalities of $A^T x \leqslant 0$ which are satisfied by $y + \epsilon \cdot v$ with equality strictly contains $\langle a_{i_1}, \dots, a_{i_k} \rangle$.

Notice that we have $b^T y = b^T(y + \epsilon \cdot v) > 0$.

Continuing this way, we obtain a solution of the form $y+u$ of $A^T x \leqslant 0$ such that one has $n-1$ linearly independent columns of A whose corresponding inequality in $A^T x \leqslant 0$ are satisfied with equality. Thus we see that each b which does not belong to C can be separated from C with an inequality of the form $c^T x \leqslant 0$ which is uniquely defined by $n-1$ linearly independent vectors from the set $a_1, \dots, a_m$. This shows that C is polyhedral.

Suppose now that $a_1, \dots, a_m$ do not span $\mathbb{R}^n$. Then there exist linearly independent vectors $d_1, \dots, d_k$ such that each d_i is orthogonal to each of the $a_1, \dots, a_m$ and $a_1, \dots, a_m, d_1, \dots, d_k$ spans $\mathbb{R}^n$. The cone generated by $a_1, \dots, a_m, d_1, \dots, d_k$ is polyhedral and thus of the form $Ax \leqslant 0$ with some matrix $A \in \mathbb{R}^{m \times n}$. Suppose that $\langle a_1, \dots, a_m \rangle = \{x \in \mathbb{R}^n \mid Ux = 0\}$. Now $C = \{x \in \mathbb{R}^n \mid Ax \leqslant 0,\ Ux = 0\}$ and C is polyhedral.

Now suppose that $C = \{x \in \mathbb{R}^n \mid a_1^T x \leqslant 0, \dots, a_m^T x \leqslant 0\}$. The cone

$$C' := \text{cone}(a_1, \dots, a_m) = \{\sum_{i=1}^m \lambda_i a_i \mid \lambda_i \geqslant 0,\ i = 1, \dots, m\}$$

is polyhedral and thus of the form $C' = \{x \in \mathbb{R}^n \mid b_1^T x \leqslant 0, \dots, b_k^T x \leqslant 0\}$. Clearly, $\mathrm{cone}(b_1, \dots, b_k) \subseteq C$ since $b_i^T a_j \leqslant 0$. Suppose now that $y \in C \backslash \mathrm{cone}(b_1, \dots, b_k)$. Then, since $\mathrm{cone}(b_1, \dots, b_k)$ is polyhedral, there exists a $w \in \mathbb{R}^n$ with $w^T y > 0$ and $w^T b_i \leqslant 0$ for each $i = 1, \dots, k$. From the latter we conclude that $w \in C'$. From $y \in C$ and $w \in C'$ we conclude $w^T y \leqslant 0$, which is a contradiction.

A set of vectors $Q = \mathrm{conv}(X)$, where $X \subseteq \mathbb{R}^n$ is finite is called a *polytope*.

Theorem 2.13 (Decomposition theorem for polyhedra). *A set $P \subseteq \mathbb{R}^n$ is a polyhedron if and only if $P = Q + C$ for some polytope Q and a polyhedral cone C.*

Proof. Suppose $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ is a polyhedron. Consider the polyhedral cone

$$\left\{ \begin{pmatrix} x \\ \lambda \end{pmatrix} \mid x \in \mathbb{R}^n,\ \lambda \in \mathbb{R}_{\geqslant 0};\ Ax - \lambda b \leqslant 0 \right\} \tag{2.10}$$

is generated by finitely many vectors $\begin{pmatrix} x_i \\ \lambda_i \end{pmatrix}$, $i = 1, \dots, m$. By scaling with a positive number we may assume that each $\lambda_i \in \{0, 1\}$. Let Q be the convex hull of the x_i with $\lambda_i = 1$ and let C be the cone generated by the x_i with $\lambda_i = 0$. A point $x \in \mathbb{R}^n$ is in P if and only if $\begin{pmatrix} x \\ 1 \end{pmatrix}$ belongs to (2.10) and thus if and only if

$$\begin{pmatrix} x \\ 1 \end{pmatrix} \in \mathrm{cone} \left\{ \begin{pmatrix} x_1 \\ \lambda_1 \end{pmatrix}, \dots, \begin{pmatrix} x_m \\ \lambda_m \end{pmatrix} \right\}.$$

Therefore $P = Q + C$.

Suppose now that $P = Q + C$ for some polytope Q and a polyhedral cone C with $Q = \mathrm{conv}(x_1, \dots, x_m)$ and $C = \mathrm{cone}(y_1, \dots, y_t)$. A vector x_0 is in P if and only if

$$\begin{pmatrix} x_0 \\ 1 \end{pmatrix} \in \mathrm{cone} \left\{ \begin{pmatrix} x_1 \\ 1 \end{pmatrix}, \dots, \begin{pmatrix} x_m \\ 1 \end{pmatrix}, \begin{pmatrix} y_1 \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} y_t \\ 0 \end{pmatrix} \right\} \tag{2.11}$$

By Theorem 2.12 (2.11) is equal to

$$\left\{ \begin{pmatrix} x \\ \lambda \end{pmatrix} \mid Ax - \lambda b \leqslant 0 \right\} \tag{2.12}$$

for some matrix A and vector b. Thus $x_0 \in P$ if and only if $Ax_0 \leqslant b$ and thus P is a polyhedron.

Let $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$. The *characteristic cone* is $\mathrm{char.cone}(P) = \{y \mid y + x \in P \text{ for all } x \in P\} = \{y \mid Ay \leqslant 0\}$. One has

i) $y \in \mathrm{char.cone}(P)$ if and only if there exists an $x \in P$ such that $x + \lambda y \in P$ for all $\lambda \geqslant 0$

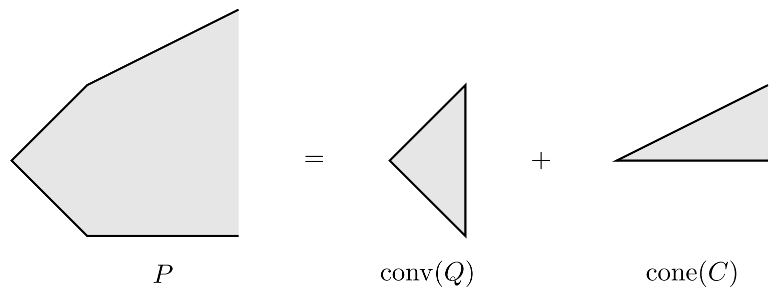


Fig. 2.7: A polyhedron and its decomposition into Q and C

ii) $P + \text{char.cone}(P) = P$
iii) P is bounded if and only if $\text{char.cone}(P) = \{0\}$.
iv) If the decomposition of P is $P = Q + C$, then $C = \text{char.cone}(P)$.

The *lineality space* of P is defined as $\text{char.cone}(P) \cap -\text{char.cone}(P)$. A polyhedron is *pointed*, if its lineality space is $\{0\}$.

2.5.1 Faces

An inequality $c^T x \leqslant \delta$ is called *valid* for P if each $x \in P$ satisfies $c^T x \leqslant \delta$. If in addition $(c^T x = \delta) \cap P \neq \emptyset$, then $c^T x \leqslant \delta$ is a *supporting inequality* and $c^T x = \delta$ is a supporting hyperplane.

A set $F \subseteq \mathbb{R}^n$ is called a *face* of P if there exists a valid inequality $c^T x \leqslant \delta$ for P with $F = P \cap (c^T x = \delta)$.

Lemma 2.3. *A set $\emptyset \neq F \subseteq \mathbb{R}^n$ is a face of P if and only if $F = \{x \in P \mid A'x = b'\}$ for a subset $A'x \leqslant b'$ of $Ax \leqslant b$.*

Proof. Suppose that $F = \{x \in P \mid A'x = b'\}$. Consider the vector $c = 1^T A'$ and $\delta = 1^T b'$. The inequality $c^T x \leqslant \delta$ is valid for P. It is satisfied with equality by each $x \in F$. If $x' \in P \backslash F$, then there exists an inequality $a^T x \leqslant \beta$ of $A'x \leqslant b'$ such that $a^T x' < \beta$ and consequently $c^T x' < \delta$.

On the other hand, if $c^T x \leqslant \delta$ defines the face F, then by the linear programming duality (see chapter 5)

$$\max\{c^T x \mid Ax \leqslant b\} = \min\{b^T \lambda \mid A^T \lambda = c, \lambda \geqslant 0\}$$

there exists a $\lambda \in \mathbb{R}^m_{\geqslant 0}$ such that $c = \lambda^T A$ and $\delta = \lambda^T b$. Let $A'x \leqslant b'$ be the subsystem of $Ax \leqslant b$ which corresponds to strictly positive entries in $Ax \leqslant b$. One has $F = \{x \in P \mid A'x = b'\}$.

A *facet* of P is an inclusion-wise maximal face F of P with $F \neq P$. An inequality $a^T x \leqslant \beta$ of $Ax \leqslant b$ is called *redundant* if $P(A,b) = P(A',b')$, where $A'x \leqslant b'$ is the system stemming from $Ax \leqslant b$ by deleting $a^T x \leqslant \beta$. A system $Ax \leqslant b$ is irredundant if $Ax \leqslant b$ does not contain a redundant inequality.

Lemma 2.4. *Let $Ax \leqslant b$ be an irredundant system. Then a set $F \subseteq P$ is a facet if and only if it is of the form $F = \{x \in P \mid a^T x = \beta\}$ for an inequality $a^T x \leqslant \beta$ of $A^{\leqslant} x \leqslant b^{\leqslant}$.*

Proof. Let F be a facet of P. Then $F = \{x \in P \mid c^T x \leqslant \delta\}$ for a valid inequality $c^T x \leqslant \delta$ of P. There exists a $\lambda \in \mathbb{R}^m_{\geqslant 0}$ with $c = \lambda^T A$ and $\delta = \lambda^T b$. There exists an inequality $a^T x \leqslant \beta$ of $A^{\leqslant} x \leqslant b^{\leqslant}$ whose corresponding entry in λ is strictly positive. Clearly $F \subseteq \{x \in P \mid a^T x = \beta\} \subset P$. Since F is an inclusion-wise maximal face one has $F = \{x \in P \mid a^T x = \beta\}$.

Let F be of the form $F = \{x \in P \mid a^T x = \beta\}$ for an inequality $a^T x \leqslant \beta$ of $A^{\leqslant} x \leqslant b^{\leqslant}$. Clearly $F \neq \emptyset$ since the system $Ax \leqslant b$ is irredundant. If F is not a facet, then $F \subseteq F' = \{x \in P \mid a'^T x = \beta'\}$ with another inequality $a'^T x \leqslant \beta'$ of $A^{\leqslant} x \leqslant b^{\leqslant}$. Let $x^* \in \mathbb{R}^n$ be a point with $a^T x^* > \beta$ and which satisfies all other inequalities of $Ax \leqslant b$. Such an x^* exists, since $Ax \leqslant b$ is irredundant. Let $\widetilde{x} \in P$ with $A^{\leqslant} \widetilde{x} < b^{\leqslant}$. There exists a point $\overline{x}$ on the line-segment $\overline{\widetilde{x} x^*}$ with $a^T \overline{x} = \beta$. This point is then also in F' and thus $a'^T x = \beta'$ follows. This shows that $a'^T x^* > \beta'$ and thus $a^T x \leqslant \beta$ can be removed from the system. This is a contradiction to $Ax \leqslant b$ being irredundant.

Lemma 2.5. *A face F of $P(A,b)$ is inclusion-wise minimal if and only if it is of the form $F = \{x \in \mathbb{R}^n \mid A'x = b'\}$ for some subsystem $A'x \leqslant b'$ of $Ax \leqslant b$.*

Proof. Let F be a minimal face of P and let $A'x \leqslant b'$ a the subsystem of inequalities of $Ax \leqslant b$ with $F = \{x \in P \mid A'x = b'\}$. Suppose that $F \subset \{x \in \mathbb{R}^n \mid A'x = b'\}$ and let $x_1 \in \mathbb{R}^n \backslash P$ satisfy $A'x_1 = b'$ and $x_2 \in F$. There exists "a first" inequality $a^T x \leqslant \beta$ of $Ax \leqslant b$ which is "hit" by the line-segment $\overline{x_2 x_1}$. Let $x^* = \overline{x_2 x_1} \cap (a^T x = \beta)$. Then $x^* \in F$ and thus $F \cap (a^T x = \beta) \neq \emptyset$. But $F \supset F \cap (a^T x = \beta)$ since $a^T x \leqslant \beta$ is not an inequality of $A'x \leqslant b'$. This is a contradiction to the minimality of F.

Suppose that F is a face with $F = \{x \in \mathbb{R}^n \mid A'x = b'\} = \{x \in P \mid A'x = b'\}$ for a subsystem $A'x \leqslant b'$ of $Ax \leqslant b$. Suppose that there exists a face $\widetilde{F}$ of P with $\emptyset \subset \widetilde{F} \subset F$. By Lemma 2.3 $\widetilde{F} = \{x \in P \mid A'x = b', A^*x = b^*\}$, where $A^*x \leqslant b^*$ is a sub-system of $Ax \leqslant b$ which contains an inequality $a^T x \leqslant \beta$ such that there exists an $x_1, x_2 \in F$ with $a^T x_1 < \beta$ and $a^T x_2 \leqslant \beta$. The line $\ell(x_1, x_2) = \{x_1 + \lambda(x_2 - x_1) \mid \lambda \in \mathbb{R}\}$ is contained in F but is not contained in $a^T x \leqslant \beta$. This shows that F is not contained in P which is a contradiction.

Exercise 18 asks for a proof of the following corollary.

Corollary 2.2. *Let* F_1 *and* F_2 *be two inclusion-wise minimal faces of* $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$*, then* $\dim(F_1) = \dim(F_2)$*.*

We say that a polyhedron contains a line $\ell(x_1, x_2)$ with $x_1 \neq x_2 \in P$ if $\ell(x_1, x_2) = \{x_1 + \lambda(x_2 - x_1) \mid \lambda \in \mathbb{R}\} \subseteq P$. A *vertex* of P is a 0-dimensional face of P. An *edge* of P is a 1-dimensional face of P.

Example 2.2. Consider a linear program $\min\{c^T x \colon Ax = b,\, x \geqslant 0\}$. A basic feasible solution defined by the basis $B \subseteq \{1, \dots, n\}$ is a vertex of the polyhedron $P = \{x \in \mathbb{R}^n \colon Ax = b,\, x \geqslant 0\}$. This can be seen as follows. The inequality $a^T x \geqslant 0$ is valid for P, where $a_B = \mathbf{0}$ and $a_{\overline{B}} = \mathbf{1}$. The inequality is satisfied with equality by a point $x^* \in P$ if and only if $x^*_{\overline{B}} = \mathbf{0}$. Since the columns of A_B are linearly independent, as B is a basis, the unique point which satisfies $a^T x \geqslant 0$ with equality is the basic feasible solution

In exercise you are asked to show that the simplex method can be geometrically interpreted as a walk on the graph $G = (V, E)$, where V is the set of basic feasible solutions and $uv \in E$ if and only if $\operatorname{conv}\{u, v\}$ is a 1-dimensional face of the polyhedron defined by the linear program.

Exercises

1) Let $\{C_i\}_{i \in I}$ be a family of convex subsets of $\mathbb{R}^n$. Show that the intersection $\bigcap_{i \in I} C_i$ is convex.
2) Show that the set of feasible solutions of a linear program is convex.
3) Prove Carathéodory's Theorem for convex hulls, Corollary 2.1.
4) Let $A \in \mathbb{R}^{n \times n}$ be a non-singular matrix and let $a_1, \dots, a_n \in \mathbb{R}^n$ be the columns of A. Show that $\operatorname{cone}(\{a_1, \dots, a_n\})$ is the polyhedron $P = \{y \in \mathbb{R}^n \colon A^{-1} y \geqslant 0\}$. Show that $\operatorname{cone}(\{a_1, \dots, a_k\})$ for $k \leqslant n$ is the set $P_k = \{y \in \mathbb{R}^n \colon a_i^{-1} x \geqslant 0, i = 1, \dots, k,\, a_i^{-1} x = 0, i = k+1, \dots, n\}$, where a_i^{-1} denotes the i-th row of A^{-1}.
5) Prove that for a finite set $X \subseteq \mathbb{R}^n$ the conic hull $\operatorname{cone}(X)$ is closed and convex.
   Hint: Use Carathéodory's theorem and exercise 4.
6) Find a countably infinite set $X \subset \mathbb{R}^2$ such that $\operatorname{cone}(X)$ is not closed. Are there any cones that are open?
7) Prove Theorem 2.5.
8) Prove Theorem 2.6.
9) Let $f : \mathbb{R}^n \to \mathbb{R}^d$ be a linear map.
   a) Show that $f(K) = \{f(x) \colon x \in K\}$ is convex if K is convex. Is the reverse also true?
   b) For $X \subseteq \mathbb{R}^n$ arbitrary, prove that $\operatorname{conv}(f(X)) = f(\operatorname{conv}(X))$.

10) Using Theorem 2.11, prove the following variant of Farkas' lemma: Let $A \in \mathbb{R}^{m\times n}$ be a matrix and $b \in \mathbb{R}^m$ be a vector. The system $Ax \leqslant b$, $x \in \mathbb{R}^n$ has a solution if and only if for all $\lambda \in \mathbb{R}^m_{\geqslant 0}$ with $\lambda^T A = 0$ one has $\lambda^T b \geqslant 0$.
11) Provide an example of a convex and closed set $K \subseteq \mathbb{R}^2$ and a linear objective function $c^T x$ such that $\inf\{c^T x \colon x \in K\} > -\infty$ but there does not exist an $x^* \in K$ with $c^T x^* \leqslant c^T x$ for all $x \in K$.
12) Consider the vectors

$$x_1 = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, x_2 = \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix}, x_3 = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}, x_4 = \begin{pmatrix} 2 \\ 4 \\ 3 \end{pmatrix}, x_5 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Let $A = \{x_1, \dots, x_5\}$. Find two disjoint subsets $A_1,\ A_2 \subseteq A$ such that

$$\operatorname{conv}(A_1) \cap \operatorname{conv}(A_2) \neq \emptyset.$$

Hint: Recall the proof of Radon's lemma
13) Consider the vectors

$$x_1 = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, x_2 = \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix}, x_3 = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}, x_4 = \begin{pmatrix} 2 \\ 4 \\ 3 \end{pmatrix}, x_5 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

The vector

$$v = x_1 + 3x_2 + 2x_3 + x_4 + 3x_5 = \begin{pmatrix} 15 \\ 14 \\ 25 \end{pmatrix}$$

is a conic combination of the x_i.
Write v as a conic combination using only three vectors of the x_i.
Hint: Recall the proof of Carathéodory's theorem
14) Prove that each nonempty polyhedron $P \subseteq \mathbb{R}^n$ can be represented as $P = L + Q$, where $L \subseteq \mathbb{R}^n$ is a linear space and $Q \subseteq \mathbb{R}^n$ is a pointed polyhedron.
15) Let $P \subset \mathbb{R}^n$ be a polytope and $f : \mathbb{R}^n \to \mathbb{R}^m$ a linear map.
    i) Show that $f(P)$ is a polytope.
    ii) Let $y \in \mathbb{R}^m$ be a vertex of $f(P)$. Show that there is a vertex $x \in \mathbb{R}^n$ of P such that $f(x) = y$.
16) Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ and consider the polyhedron $P = P(A, b)$. Show that $\dim(P) = n - \operatorname{rank}(A^=)$.
17) i) Show that the dimension of each minimal face of a polyhedron P is equal to $n - \operatorname{rank}(A)$.
    ii) Show that a polyhedron has a vertex if and only if the polyhedron does not contain a line.

18) Show that the affine dimension of the minimal faces of a polyhedron $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ is invariant.
19) In this exercise you can assume that a linear program $\max\{c^T x \mid Ax \leqslant b\}$ can be solved in polynomial time. Suppose that $P(A, b)$ has vertices and that the linear program is bounded. Show how to compute an optimal *vertex* solution of the linear program in polynomial time.
20) Let $P = \{x \in \mathbb{R}^n \colon Ax = b,\, x \geqslant 0\}$ be a polyhedron, where $A \in \mathbb{R}^{m \times n}$ has full row-rank. Let B_1, B_2 be two bases such that $|B_1 \cap B_2| = m - 1$ and suppose that the associated basic solutions x_1^* and x_2^* are feasible. Show that, if $x_1 \neq x_2$, then $\mathrm{conv}\{x_1^*, x_2^*\}$ is a 1-dimensional face of P.

Chapter 3
The simplex method

In this chapter we describe the simplex method. The task is to solve a linear program

$$\max\{c^T x \colon x \in \mathbb{R}^n,\ Ax \leqslant b\}. \tag{3.1}$$

We make the following assumption.

> The matrix $A \in \mathbb{R}^{m \times n}$ is of full column-rank. In other words, the columns of A are linearly independent.

This assumption is not a restriction, since we can solve the following equivalent linear program instead, where each x_i is represented as the difference of two positive values $x_i = x_i^+ - x_i^-$.

$$\max\{c^T x^+ - c^T x^- \colon x^+, x^- \in \mathbb{R}^n,\ Ax^+ - Ax^- \leqslant b, x^+ \geqslant 0,\ x^- \geqslant 0\}. \tag{3.2}$$

The constraint matrix of the linear program (3.2) in inequality standard form is

$$\begin{pmatrix} A & -A \\ -I_n & \mathbf{0} \\ \mathbf{0} & -I_n \end{pmatrix},$$

where I_n is the $n \times n$ identity matrix and $\mathbf{0}$ is the $n \times n$ all-zero matrix. Clearly this matrix has linearly independent columns.

3.1 Adjacent vertices

Let $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ be the polyhedron of feasible solutions of (3.1).

Definition 3.1. Two extreme points $x_1 \neq x_2$ of P are *adjacent*, if there exists a valid inequality $d^T x \leqslant \delta$ of P such that

$$P \cap \{x \in \mathbb{R}^n \colon d^T x = \delta\} = \{\lambda x_1 + (1-\lambda) x_2 \colon \lambda \in [0,1]\}.$$

In other words, $x_1 \neq x_2$ are adjacent if there exists a valid inequality for P such that the points of P that satisfy this inequality with equality are exactly the line-segment spanned by x_1 and x_2.

Similar to the characterization of extreme points in Theorem 2.1, we can state and prove the following theorem.

Theorem 3.1. *Two distinct vertices x_1 and x_2 of P are adjacent if and only if there exists a sub-system $A'x \leqslant b'$ of $Ax \leqslant b$ such that*

i) $A' \in \mathbb{R}^{(n-1)\times n}$ and the rows of A' are linearly independent.
ii) x_1 and x_2 satisfy $A'x \leqslant b'$ with equality.

Proof. Suppose that $x_1 \neq x_2$ are adjacent and suppose that $d^T x \leqslant \delta$ is a valid inequality that asserts this fact. Consider the sub-system $\widetilde{A}x \leqslant \widetilde{b}$ of inequalities of $Ax \leqslant b$ that are satisfied by $1/2(x_1 + x_2)$ with equality. These are the inequalities that are satisfied by all points on the line-segment with equality. Since $\widetilde{A}(x_1 - x_2) = 0$, one has $\operatorname{rank}(\widetilde{A}) \leqslant n-1$. If $\operatorname{rank}(\widetilde{A}) < n-1$, then there exists a $v \in \mathbb{R}^n$ that is linearly independent from $(x_1 - x_2)$ that satisfies $\widetilde{A}v = 0$. Consequently there exist a $\varepsilon > 0$ such that

$$\{\frac{1}{2}(x_1 + x_2) + \mu_1(x_1 - x_2) + \mu_2 v \colon -\varepsilon \leqslant \mu_1, \mu_2 \leqslant \varepsilon\} \subseteq P. \tag{3.3}$$

All points of the set (3.3) satisfy $d^T x \leqslant \delta$ with equality and they are not a subset of the line-segment spanned by x_1 and x_2. From this we conclude that $\operatorname{rank}(\widetilde{A}) = n-1$ which implies that there exists a sub-system $A'x \leqslant b'$ satisfying i) and ii).

Suppose on the other hand that there exists a sub-system $A'x \leqslant b'$ that satisfies i) and ii). The line spanned by x_1 and x_2 is the set of points of $\mathbb{R}^n$ that satisfies $A'x = b'$ and the intersection of this line with P is, since x_1 and x_2 are vertices, the line-segment spanned by these two points.

The inequality $\mathbf{1}^T A'x \leqslant \mathbf{1}^T b'$ is valid for P and is satisfied by the line-segment spanned by x_1 and x_2 with equality. Let $y^* \in P$ be a point that does not lie on the line segment. Then one of the inequalities of $A'x \leqslant b'$ is satisfied by y^* with strict inequality and thus y^* does not satisfy $d^T x \leqslant \delta$ with equality.

3.2 Bases, feasible bases and vertices

We will frequently use the following notation. Let $B \subseteq \{1, \ldots, m\}$ then $A_B \in \mathbb{R}^{|B|\times n}$ is the matrix consisting of the rows of A that are indexed by B and $b_B \in \mathbb{R}^{|B|}$ is the vector whose components are the ones of b indexed by B. For example, for $A = \begin{pmatrix} 3 & 2 \\ 7 & 1 \\ 8 & 4 \end{pmatrix}$, $b = \begin{pmatrix} 3 \\ 2 \\ 6 \end{pmatrix}$ and $B = \{2,3\}$, one has $A_B = \begin{pmatrix} 7 & 1 \\ 8 & 4 \end{pmatrix}$, $b_B = \begin{pmatrix} 2 \\ 6 \end{pmatrix}$.

Definition 3.2. An index set $B \subseteq \{1, \ldots, m\}$ is a *basis* if $|B| = n$ and A_B is non-singular. If in addition $x^* = A_B^{-1} b_B$ is feasible, then B is called a *feasible basis*.

Theorem 2.1 implies that every vertex x^* of $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ is *represented* by a basis B, i.e. x^* is the unique solution of $A_B x^* = b_B$. This representation however must not be unique, see Figure 3.1. We say that a linear program is *degenerate*, if there exists a basic solution $x^* \in \mathbb{R}^n$ that satisfies $n+1$ inequalities with equality. Otherwise the linear program is called *non-degenerate*. If the linear program is non-degenerate, then each vertex is represented by *exactly one* basis.

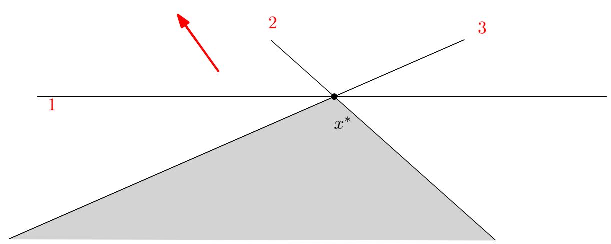


Fig. 3.1: The vertex x^* is represented by each choice of two of the three tight constraints. The linear program is degenerate. The red vector is the objective function vector and the red labels are the indices of the constraints.

Definition 3.3. A basis B is called *optimal* if it is feasible and the unique $\lambda \in \mathbb{R}^m$ with

$$\lambda^T A = c^T \text{ and } \lambda_i = 0, \, i \notin B \tag{3.4}$$

satisfies $\lambda \geqslant 0$.

The basis $\{1,2\}$ in Figure 3.1 is not optimal whereas the bases $\{2,3\}$ and $\{1,3\}$ are optimal bases.

Theorem 3.2. *If B is an optimal basis, then $x^* = A_B^{-1} b_B$ is an optimal solution of the linear program* (3.1).

Proof. The inequality $\lambda^T Ax \leqslant \lambda^T b$ is valid for $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$. But $\lambda^T A = c^T$ and $\lambda^T b = \lambda^T A x^* = c^T x^*$. Consequently x^* is an optimal solution of the linear program (3.1).

3.3 Moving to an improving vertex

Suppose now that B is a feasible but not optimal basis. Then the unique λ satisfying (3.4) has a negative component $\lambda_i < 0$ for some $i \in B$. The idea is now to move from $x_B^* = A_B^{-1} b_B$ by remaining tight at all constraints indexed by B except for i.

There is only one way this can be achieved. Namely by moving in the unique direction d with

$$a_j^T d = \begin{cases} 0 & \text{for } j \in B \setminus \{i\} \\ -1 & \text{if } j = i. \end{cases}$$

When we do this, we follow the ray $x_B^* + \varepsilon \cdot d$ with $\varepsilon \geqslant 0$. What happens to the objective function, as ε grows? Since $c^T d = \lambda^T A d = -\lambda_i > 0$, the objective function strictly grows with growing ε. There are now two cases.

At some point, we hit the boundary of a constraint and further increase of ε results in an infeasible point. Let $K \subseteq \{1, \ldots, m\}$ be the set of indices

$$K = \{k \colon 1 \leqslant k \leqslant m,\, a_k^T d > 0\}. \tag{3.5}$$

Those are the indices of constraints that, at some point, will be violated. We can increase ε until

$$\varepsilon^* = \min_{k \in K} \{(b_k - a_k^T x^*)/a_k^T d\}. \tag{3.6}$$

Now pick any $k \in K$ for which this minimum is achieved and set $B' = B \setminus \{i\} \cup \{k\}$. This is a feasible basis, d is orthogonal to all rows indexed by $B \setminus \{i\}$ but not to a_k.

In the case where there is no constraint that puts an upper bound on ε, then the linear program is *unbounded*. We have described one iteration of the simplex algorithm. We iterate this procedure until an optimal solution is found.

Algorithm 3.1 (Simplex algorithm).

Start with feasible basis B

`while` B is not optimal

  Let $i \in B$ be index with $\lambda_i < 0$
  Compute $d \in \mathbb{R}^n$ with $a_j^T d = 0$, $j \in B \setminus \{i\}$ and $a_i^T d = -1$
  Determine $K = \{k \colon 1 \leqslant k \leqslant m,\, a_k^T d > 0\}$

  `if` $K = \emptyset$
    assert LP unbounded
  `else`
    Let $k \in K$ index where $\min_{k \in K} (b_k - a_k^T x^*)/a_k^T d$ is attained
    update $B := B \setminus \{i\} \cup \{k\}$

Theorem 3.3. *If the linear program* (3.1) *is non-degenerate, then the simplex algorithm terminates.*

Proof. In the non-degenerate case, $\varepsilon^* > 0$ and the simplex algorithm makes progress, i.e., the objective function value strictly increases after each iteration. Since there is only a finite number of vertices, the algorithm terminates.

3.4 Termination in the degenerate case

In the case where the linear program (3.1) is degenerate, we cannot argue that the objective function value increases each iteration and that the simplex algorithm terminates. However, the simplex algorithm leaves us some choice. Namely, there can be several indices $i \in B$ such that $\lambda_i < 0$. Also, there could be several indices $k \in K$ attaining the minimum in (3.6). If one adheres to the *smallest index rule*, then one can prove termination of the simplex algorithm also in the degenerate case. One iteration of the simplex algorithm is now as follows.

Algorithm 3.2 (Simplex algorithm with the smallest index rule).

Start with feasible basis B

`while` B is not optimal

Let $i^* \in B$ be the *smallest index* with $\lambda_i < 0$
Compute $d \in \mathbb{R}^n$ with $a_j^T d = 0$, $j \in B \setminus \{i\}$ and $a_i^T d = -1$
Determine $K = \{k \colon 1 \leqslant k \leqslant m,\, a_k^T d > 0\}$
`if` $K = \emptyset$
assert LP unbounded
`else`
Let $k^* \in K$ the *smallest index* where $\min\limits_{k \in K} (b_k - a_k^T x^*)/a_k^T d$ is attained
update $B := B \setminus \{i^*\} \cup \{k^*\}$

Theorem 3.4. *The simplex algorithm with the smallest index rule terminates.*

Proof. We suppose that the simplex algorithm does not terminate. This means that the simplex algorithm iterates through a sequence of bases

$$B_0, B_1, \ldots, B_k$$

with $B_k = B_0$. Inspecting two succeeding bases B_ℓ and $B_{\ell+1}$ for $0 \leqslant \ell \leqslant k-1$, one has $B_{\ell+1} = B_\ell \setminus \{i\} \cup \{j\}$, i.e., i *leaves* and j *enters* B_ℓ. Now let j be *the largest index* that leaves on that sequence. Since $B_0 = B_k$, j also enters again at some point. Let p and q be the indices of bases, $0 \leqslant p, q < k$ where j leaves and enters respectively.

Let $\lambda^{(p)}$ and $d^{(q)}$ be the corresponding λ and d vectors from the iteration p and q of the simplex algorithm respectively. Since ${\lambda^{(p)}}^T A = c^T$ and $c^T d^{(q)} > 0$ we conclude

$$ {\lambda^{(p)}}^T A d^{(q)} > 0. \tag{3.7} $$

Let $i \in B_p$ be an index with

$$ \lambda_i^{(p)} a_i d^{(q)} > 0 \tag{3.8} $$

where a_i denotes the i-th row of A.

We now distinguish three cases. Let us suppose that $i > j$. Then, since j is the largest index that ever leaves or enters one has that i is also an element of B_q implying that $a_i d^{(q)}$ is 0 or -1. It cannot be -1, since this would mean that i leaves B_q contradictory to the choice of j.

Suppose then that $i < j$. Then $\lambda_i^{(p)} > 0$ since j is the smallest index that can leave the basis B_p. But $a_i d^{(q)} > 0$ is not possible, otherwise index i is an index where the minimum ($\varepsilon^* = 0$) in (3.6) is attained and $j > i$ is the smallest index where this minimum is attained. Thus this case can also be ruled out.

Finally, if $i = j$, then $\lambda_i^{(p)} < 0$ and $a_i d^{(q)} > 0$ which also contradicts (3.8).

3.5 Finding an initial basic feasible solution

The simplex algorithm starts with a feasible basis. How can such a feasible basis be determined? In fact, this can be done with an auxiliary linear program.

Any linear program has an equivalent form

$$ \max\{c^T x \colon Ax \leqslant b,\, x \geqslant 0\}. \tag{3.9} $$

We want to find an initial feasible basis for this linear program. Suppose first that we re-write the constraints $Ax \leqslant b$ as $A_1 x \leqslant b_1$ and $A_2 x \leqslant b_2$ with $b_1 \geqslant 0$ and $b_2 < 0$. Now consider the following linear program

$$ \min\{\mathbf{1}^T y \colon A_1 x \leqslant b_1,\, A_2 x \leqslant b_2 + y,\, x, y \geqslant 0, y \leqslant -b_2\}. \tag{3.10} $$

An initial basic feasible solution is $x = 0$ and $y = -b_2$ with the basis corresponding to the inequalities $x \geqslant 0$ and $y \leqslant -b_2$. The simplex algorithm applied to this linear program terminates. It finds an optimal solution with objective value 0 if and only if the original linear program is feasible. Let B be the optimal basis in this case. Then B without the indices corresponding to the constraints $y \geqslant 0$ is a feasible basis for the linear program (3.9).

3.6 Removing degeneracy by perturbation

In the following, we derive an alternative pivoting rule that also ensures termination of the simplex algorithm. We begin by *perturbing* the right-hand-sides of our constraints

$$Ax \leqslant b \tag{3.11}$$

by adding a vector p_ε

$$Ax \leqslant b + p_\varepsilon \tag{3.12}$$

where p_ε is the vector

$$p_\varepsilon = \begin{pmatrix} \varepsilon \\ \varepsilon^2 \\ \vdots \\ \varepsilon^m \end{pmatrix}.$$

Lemma 3.1. *If $Ax \leqslant b$ is feasible, then $Ax \leqslant b + p_\varepsilon$ is feasible for each $\varepsilon > 0$. If B is an infeasible basis of $Ax \leqslant b$, then B is an infeasible basis of $Ax \leqslant b + p_\varepsilon$ for $\varepsilon > 0$ sufficiently small.*

Proof. Clearly, the feasible solutions of (3.12) contain the feasible solutions of (3.11). Suppose now that B is infeasible, then there exists an index i and some $\delta > 0$ such that

$$a_i A_B^{-1} b_B \geqslant b_i + \delta,$$

where a_i and b_i are the i-th row of A and the i-th component of b respectively. Now

$$a_i A_B^{-1}(b_B + p_{\varepsilon B}) \geqslant b_i + \delta + A_B^{-1} p_{\varepsilon B} > b_i$$

if $\varepsilon > 0$ is sufficiently small.

Furthermore, we can show that the constraint system (3.12) is non-degenerate for $\varepsilon > 0$ small enough.

Lemma 3.2. *If $\varepsilon > 0$ is small enough, then* (3.12) *is non-degenerate.*

Proof. If the set of inequalities (3.12) is degenerate, then there exists a basis B and an index $i \notin B$ such that

$$a_i x^*_{B,\varepsilon} = b_i + \varepsilon^i \tag{3.13}$$

where $x^*_{B,\varepsilon} = A_B^{-1}(b_B + (p_\varepsilon)_B)$. Notice that

$$a_i x^*_{B,\varepsilon} - b_i - \varepsilon^i$$

is a nonzero polynomial in ε. It is non-zero, since the coefficient of ε^i is -1 as ε^i is not a component of $(p_\varepsilon)_B$. A nonzero polynomial has only a finite number of roots. Thus, if $\varepsilon > 0$ is small enough, no equation of the form (3.13) can hold.

Suppose now that we want to solve the linear program

$$\max\{c^T x\colon x \in \mathbb{R}^n, Ax \leqslant b\} \tag{3.14}$$

The idea is run the simplex algorithm on the perturbed linear program

$$\max\{c^T x\colon x \in \mathbb{R}^n, Ax \leqslant b + p_\varepsilon\} \tag{3.15}$$

where $\varepsilon > 0$ is sufficiently small. This linear program is non-degenerate. What is nice is that this perturbation does not have to be computed explicitly. We can formulate a pivot rule for the non-perturbed linear program (3.14) that is conform with the pivoting that is performed on the perturbed linear program (3.13). For this, assume that B is a feasible basis of the perturbed linear program (3.13). By Lemma 3.1, B is also feasible for the unperturbed linear program. Let us now consider the iteration of the simplex algorithm at the basis B. As before, we choose $i \in B$ with $\lambda_i < 0$ arbitrary and compute $d \in \mathbb{R}^n$. Now we have to determine the unique index of the inequality of (3.15) that is hit first, when moving in the direction of d. This is the largest index k among those indices of K where the minimum $\min_{k\in K}(b_k - a_k^T x^*)/a_k^T d$ is attained. Thus the following variant of the simplex algorithm ensures termination.

Algorithm 3.3 (Simplex with largest index leaving rule).

Start with feasible basis B of the *perturbed* linear program (3.15)

`while` B is not optimal

Let $i \in B$ be index with $\lambda_i < 0$
Compute $d \in \mathbb{R}^n$ with $a_j^T d = 0$, $j \in B \setminus \{i\}$ and $a_i^T d = -1$
Determine $K = \{k\colon 1 \leqslant k \leqslant m,\ a_k^T d > 0\}$

`if` $K = \emptyset$
assert LP unbounded
`else`
Let $k \in K$ be the *largest* index where $\min\limits_{k\in K}(b_k - a_k^T x^*)/a_k^T d$ is attained
update $B := B \setminus \{i\} \cup \{k\}$

Exercise 5 explains how to convert a feasible basis of (3.14) into a feasible basis of (3.15).

Theorem 3.5. *The variant of the simplex method described in Algorithm 3.3 terminates.*

Exercises

1. For each of the following assertion, provide a proof or a counterexample.

i) An index that has just left the basis B in the simplex algorithm cannot enter in the very next iteration.
ii) An index that has just entered the basis B in the simplex algorithm cannot leave again in the very next iteration.

2. Consider the auxiliary linear program to find an initial feasible basis (3.10). The constraint matrix of this linear program is of the form

$$\begin{pmatrix} A & 0 \\ -I_n & 0 \\ 0 & -I_{m_2} \\ 0 & I_{m_2} \end{pmatrix},$$

where m_2 is the number of rows of A_2. This matrix has $m+n+2\cdot m_2$ rows. Describe an initial feasible basis that corresponds to the basic feasible solution $x = 0$ and $y = 0$.
Suppose that the optimal value of the auxiliary linear program is 0 and let B' be an optimal basis found by the simplex algorithm. Prove that $B' \setminus \{m+n+1, \ldots, m+n+m_2\}$ is a feasible *basis* of the linear program (3.9).
3. Suppose that the linear program $\max\{c^T x \colon x \in \mathbb{R}^n,\ Ax \leqslant b\}$ is non-degenerate and B is an optimal basis. Show that the linear program has a unique optimal solution if and only if $\lambda_B > 0$.
4. A polyhedron $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ *contains a line*, if there exists a nonzero $v \in \mathbb{R}^n$ and an $x^* \in R^n$ such that for all $\lambda \in \mathbb{R}$, the point $x^* + \lambda \cdot v \in P$. Show that a nonempty polyhedron P contains a line if and only if A does not have full column-rank.
5. Let x^* be a basic feasible solution of the non-perturbed linear program (3.14) and let $C \subseteq \{1, \ldots, m\}$ be the indices of inequalities that are tight at x^*. Prove that the following *greedy algorithm* produces a feasible basis $B \subseteq C$ of the perturbed linear program.

Initialize $B = \emptyset$
`while` B is not a basis
    Let $i \in C$ be the largest index such that the rows
    indexed by $B \cup \{i\}$ are linearly independent
    Update $B = B \cup \{i\}$.

Chapter 4

Running time analysis of the simplex algorithm

4.1 Asymptotic analysis

An algorithm, like the simplex algorithm, executes set of *instructions* used in common programming languages like arithmetic operations, comparisons or read/write instructions. The sequence of these instructions is controlled by *loops and conditionals* like `if, while, for` etc.

Each of these instructions requires time. The *running time* of an algorithm is the *number of instructions* that the algorithm performs. This number depends on the *input* of the algorithm.

Example 4.1. Consider the following *Python* code for *matrix multiplication* using datastructures for matrices from the *Sympy* library.

```
def multiply(A,B):
    m = A.rows
    l = A.cols
    n = B.cols
    C = Matrix.zeros(m,n)

    for i in range(m):
        for j in range(n):
            s = 0
            for k in range(l):
                s = s + A[i,k]*B[k,j]
            C[i,j] = s
    return C
```

How many basic instructions does the algorithm perform on input $A \in \mathbb{R}^{m\times l}$ and $B \in \mathbb{R}^{l\times n}$? Let us first ignore the first 4 instructions, which are 4 read and write instructions. The number of additions is $l\cdot m\cdot n$. The number of multiplications is $l\cdot m\cdot n$. The number of store-instructions is $m\cdot n\cdot(1+l+1)$. The number of read-instructions is of similar magnitude.

The above example shows that an *exact counting* is sometimes tedious. Looking at the algorithm however, you quickly agree that there exists *some constant* $c \in \mathbb{R}_{>0}$ such that the algorithm performs at most $c \cdot m \cdot n \cdot l$ instructions.

In the analysis of algorithms, one does usually not care so much about the constant c above in the beginning. There are sub-fields of algorithms where this constant however matters. Especially for algorithms on large data sets, where access to external data is costly. However, this is another story that does not concern us here. When we analyze algorithms, we are interested in the *asymptotic running time*.

Definition 4.1 (O-notation).
Let $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ be functions. We say

- $T(n) = O(f(n))$, if there exist positive constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with
$$T(n) \leqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$
- $T(n) = \Omega(f(n))$, if there exist constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with
$$T(n) \geqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$
- $T(n) = \Theta(f(n))$ if
$$T(n) = O(f(n)) \text{ and } T(n) = \Omega(f(n)).$$

Example 4.2. The function $T(n) = 2n^2+3n+1$ is in $O(n^2)$, since for all $n \geqslant 1$ one has $2n^2 + 3n + 1 \leqslant 6n^2$. Here $n_0 = 1$ and $c = 6$. Similarly $T(n) = \Omega(n^2)$, since for each $n \geqslant 1$ one has $2n^2 + 3n + 1 \geqslant n^2$. Thus $T(n)$ is in $\Theta(n^2)$.

We measure the running time of algorithms in terms of the *length of the input*. The matrices $A \in \mathbb{R}^{m\times l}$ and $B \in \mathbb{R}^{l\times n}$ that are the input of the matrix-multiplication algorithm of Example 4.1 consist of $m \times l$ and $l \times n$ numbers each. The algorithm runs in time $O(m \cdot n \cdot l)$. If $m = n = l$, then one can say that the algorithm runs in time $O(n^3)$.

What does it mean for an algorithm to be efficient? For us, this will mean that it runs in polynomial time. As a first definition of polynomial time algorithm, we say that an algorithm runs in *polynomial time*, if there exists a constant k such that the algorithm runs in time $O(n^k)$, where n is the length of the input of the algorithm. The next example will make us revise this definition immediately.

Example 4.3.

```
def listexp(L):
    s = 2
    for i in L:
        s = s**2
    return s
```

The input of this algorithm is a list. Let us say that the list has n elements. Then clearly, the algorithm carries out a polynomial, even linear, number of operations. However, the algorithm squares 2 repeatedly, n times to be precise. Thus, at the end, the variable s holds the number 2^{2^n}. The number of *bits* that are needed to represent s in binary notation is 2^n which is exponential in the input length.

The example shows that, even if we perform a polynomial number of operations, the *binary representation* of the numbers that are used by the algorithm grows exponentially. This should not happen in a polynomial time algorithm. We thus make the following, now final, definition.

Definition 4.2. An algorithm runs in *polynomial time*, if there exists a constant k such that the algorithm performs $O(n^k)$ operations on rational numbers whose size is bounded by $O(n^k)$. Here, the *size* of an integer x is $\mathrm{size}(x) = \lceil \log(|x|+1) \rceil$ and for $x \in \mathbb{Q}$, $\mathrm{size}(x) = \mathrm{size}(p) + \mathrm{size}(q)$, where $x = p/q$ with $p, q \in \mathbb{Z}$, $q \geqslant 1$ and $\gcd(p, q) = 1$. Thus the size is asymptotically equal to the number of bits in the reduced representation of a rational number. The parameter n is the *input length*, where the binary encoding of the input numbers is also taken into account.

We now use this definition to analyze the famous *Euclidean* algorithm that computes the *greatest common divisor* of two integers.

For $a, b \in \mathbb{Z}$, $b \neq 0$ we say b *divides* a if there exists an $x \in \mathbb{Z}$ such that $a = b \cdot x$. We write $b \mid a$. For $a, b, c \in \mathbb{Z}$, if $c \mid a$ and $c \mid b$, then c is a *common divisor* of a and b. If at least one of the two integers a and b is non-zero, then there exists a *greatest common divisor* of a and b. It is denoted by $\gcd(a, b)$.

How do we compute the greatest common divisor efficiently? The following is called *division with remainder*. For $a, b \in \mathbb{Z}$ with $b > 0$ there exist unique integers $q, r \in \mathbb{Z}$ with

$$a = q \cdot b + r, \text{ and } 0 \leqslant r < b.$$

Now clearly, for $a, b \in \mathbb{Z}$ with $b > 0$ and $q, r \in \mathbb{Z}$ as above one has $\gcd(a, b) = \gcd(b, r)$. This gives rise to the following algorithm.

Algorithm 4.1.

```
# Condition a>=b>=0, not both equal to 0
def Euclid(a,b):
    if b == 0:
        return a
    else:
        r = a%b
        return Euclid(b,r)
```

Theorem 4.1. *The Euclidean algorithm runs in time $O(n)$.*

Proof. Suppose that a and b have at most n bits each. Clearly, the numbers in the course of the algorithm have at most n bits. Furthermore, if $a \geqslant b$, then $r \leqslant a/2$, where r is the remainder of the division of a by b. Thus each second iteration, the first parameter of the input has one bit less. Thus the number of operations is bounded by $O(n)$.

4.2 One iteration of the simplex algorithm

In the following we will analyze the complexity of one iteration of the simplex algorithm. We suppose that the input data $A \in \mathbb{Q}^{m\times n}$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$ is rational.

Now if A_B^{-1} has been computed, then $\lambda_B^T = c^T \cdot A_B^{-1}$ and d, which is the negative of a column of A_B^{-1}, can be computed with $O(n^2)$ operations. To compute K we have to compute $A \cdot d$. This can be done with $O(m \cdot n)$ operations. The index of element entering the basis can be determined by computing $x^* = A_B^{-1} b_B$, $b - Ax^*$ and Ad. Thus, if A_B^{-1} is known, this amounts to a total of

$$O(m \cdot n)$$

arithmetic operations.

In order to argue that one iteration of the simplex algorithm runs in polynomial time, we have to show that each of the numbers of A_B^{-1} has size that is polynomial in the size of the input and that A_B^{-1} can be quickly computed.

Let us first see how large the size of the numbers in A_B^{-1} can be. Suppose that

$$A = \begin{pmatrix} p_{11}/q_{11} & \cdots & p_{1n}/q_{1n} \\ & \cdots & \\ p_{n1}/q_{n1} & \cdots & p_{nn}/q_{nn} \end{pmatrix} \in \mathbb{Q}^{n\times n}.$$

is invertible. The size of the product of denominators $\prod_{i=1,j=1}^n q_{ij}$ is clearly linear in the size of the input. Now write $A = 1/Q \cdot A'$ where Q is product of denominators and $A' \in \mathbb{Z}^{n\times n}$ Since $A^{-1} = Q \cdot (A')^{-1}$ we only have to answer this question for A' instead of A. In other words, we can assume that A is integral.

For $A \in \mathbb{R}^{m\times n}$ and $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$, A_{ij} denotes the matrix obtained from A by deleting the i-th row and j-th column. The following matrix inversion formula is known as Cramer's rule.

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} \det(A_{11}) & -\det(A_{21}) & \det(A_{31}) & \dots \\ -\det(A_{12}) & \det(A_{22}) & -\det(A_{32}) & \dots \\ \det(A_{13}) & -\det(A_{23}) & \det(A_{33}) & \dots \\ \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \dots \end{pmatrix}$$

Theorem 4.2 (Hadamard bound). *Let $A \in \mathbb{R}^{n\times n}$ be non-singular. Then*

$$|\det(A)| \leqslant \prod_{i=1}^{n} \|a_i\|_2 \leqslant n^{n/2} \cdot B^n,$$

where B is upper bound on absolute values of entries of A.

Proof. The Gram-Schmidt orthogonalization of A yields a factorization

$$A = Q \cdot R,$$

where R is an upper triangular matrix with ones on the diagonal. The matrix Q has orthogonal columns, where the length of the i-th column $q^{(i)}$ is upper bounded by the length of the i-th column of A. The assertion follows from

$$\det(A)^2 = \det(Q)^2 = \det(Q^T)\det(Q) = \prod_i \|q^{(i)}\|^2.$$

Corollary 4.1. *If $A \in \mathbb{Z}^{n\times n}$ is integral and each entry in absolute value is bounded by B, then* $\operatorname{size}(\det(A)) = O(n \log n + n \cdot \operatorname{size}(B))$.

Corollary 4.2. *Let $A \in \mathbb{Q}^{n\times n}$ be an invertible matrix. The size of A^{-1} is polynomial in the size of A.*

Now we known that the size of A_B^{-1} polynomial in the size of the input (A, b, c)? Now, how expensive is it to compute A_B^{-1}? Suppose basis B is preceded by B' with

$$\begin{aligned} B' &= \{b_1, \ldots, b_{k-1}, b'_k, b_{k+1}, \ldots, b_n\} \\ B &= \{b_1, \ldots, b_{k-1}, b_k, b_{k+1}, \ldots, b_n\} \end{aligned}$$

Then each row of $A_B \cdot A_{B'}^{-1}$, except for row k, is the corresponding row of the $n \times n$ identity matrix except. Let the k-th row be $(v_1, v_2, \ldots, v_n)$. We now only have to perform the elementary column operations that turn this row into the k-th unit vector on $A_{B'}^{-1}$ to obtain A_B^{-1}. In other words, the following algorithm computes A_B^{-1} given $A_{B'}^{-1}$.

- Compute $a_{b_k}^T \cdot A_{B'}^{-1} = (v_1, \ldots, v_k, \ldots, v_n)$
- For each column $i \neq k$: Subtract v_i/v_k times column k from column i
- Divide column k by v_k

This amounts to a total number of $O(n^2)$ arithmetic operations for the update. We can conclude with the following theorem.

Theorem 4.3. *One iteration of the simplex algorithm requires a total number of $O(m \cdot n)$ operations on rational numbers whose size is polynomial in the input size.*

Chapter 5
Duality

Via the termination argument for the simplex algorithm, we can now prove the duality theorem. We are given a linear program

$$\max\{c^T x \colon x \in \mathbb{R}^n,\, Ax \leqslant b\}, \tag{5.1}$$

called the *primal* and its *dual*

$$\min\{b^T y \colon y \in \mathbb{R}^m,\, A^T y = c,\, y \geqslant 0\}. \tag{5.2}$$

We again formulate the theorem of weak duality in this setting.

Theorem 5.1 (Weak duality). *If x^* and y^* are primal and dual feasible solutions respectively, then $c^T x^* \leqslant b^T y^*$.*

Proof. We have $c^T x^* = {y^*}^T A x^* \leqslant {y^*}^T b$. □

The strong duality theorem tells us that if there exist feasible primal and dual solutions, then there exist feasible primal and dual solutions which have the same objective value. We can prove it with the simplex algorithm.

Theorem 5.2. *If the primal linear program is feasible and bounded, then so is the dual linear program. Furthermore in this case, both linear programs have an optimal solution and the optimal values coincide.*

Proof. Suppose first that A has full column rank. The simplex method finds an optimal basis B of (5.1) with x_B^* being an optimal feasible solution. At the same time, we have a $\lambda \in \mathbb{R}^m_{\geqslant 0}$ with $\lambda_i = 0$ if $i \notin B$ and $\lambda^T A = c^T$. Notice that λ is a feasible solution of the dual linear program (5.2). One has

$$c^T x_B^* = \lambda^T A x_B^* = \lambda^T b,$$

which shows the theorem in this case.

If A does not have full column rank, then we re-write the linear program (5.1) as

$$\max\{c^T(x_1 - x_2)\colon A(x_1 - x_2) \leqslant b, x_1 \geqslant 0, x_2 \geqslant 0\}. \tag{5.3}$$

There is a dual solution that we partition into three parts $\lambda_1, \lambda_2, \lambda_3 \geqslant 0$. Its dual objective function value is $\lambda_1^T b$. Furthermore

$$\lambda_1^T A - \lambda_2 = c^T,\ -\lambda_1^T A - \lambda_3 = -c^T,$$

which together with $\lambda_2, \lambda_3 \geqslant 0$ implies $\lambda_1^T A = c^T$ and $\lambda_2 = \lambda_3 = 0$. This means that λ_1 is an optimal dual solution.

We can formulate dual linear programs also if the linear program is not in inequaliy standard form. The procedure above can be described as follows. We transform a linear program into a linear program in inequality standard form and construct its dual linear program. This dual is then transformed into an equivalent linear program again which is conveniently described.

Let us perform such operations on the dual linear program

$$\min\{b^T y\colon y \in \mathbb{R}^m,\ A^T y = c,\ y \geqslant 0\}$$

of the primal $\max\{c^T x\colon x \in \mathbb{R}^n,\ Ax \leqslant b\}$. We transform it into inequality standard form

$$\begin{aligned} \max -b^T y& \\ A^T y &\leqslant c \\ -A^T y &\leqslant -c \\ -Iy &\leqslant 0. \end{aligned}$$

The dual linear program of this is

$$\begin{aligned} \min c^T x_1 - c^T x_2& \\ Ax_1 - Ax_2 - x_3 &= -b \\ x_1, x_2, x_3 &\geqslant 0 \end{aligned}$$

This is equivalent to

$$\begin{aligned} \max c^T(x_2 - x_1)& \\ A(x_2 - x_1) + x_3 &= b \\ x_1, x_2, x_3 &\geqslant 0 \end{aligned}$$

which is equivalent to the primal linear program

$$\begin{aligned} \max c^T x& \\ Ax &\leqslant b. \end{aligned}$$

Loosely formulated one could say that “The dual of the dual is the primal”. But this, of course, is not to be understood as a mathematical statement. In any case we can state the following corollary.

Corollary 5.1. *If the dual linear program has an optimal solution, then so does the primal linear program and the objective values coincide.*

We present another example of duality that we will need later on. Consider a linear program

$$\begin{array}{rcl} \max c^T x & & \\ Bx & = & b \\ Cx & \leqslant & d. \end{array} \tag{5.4}$$

After re-formulation, we obtain

$$\begin{array}{rcl} \max c^T x & & \\ Bx & \leqslant & b \\ -Bx & \leqslant & -b \\ Cx & \leqslant & d \end{array}$$

We can form the dual of the latter problem and obtain

$$\begin{array}{r} \min b^T y_1 - b^T y_2 + d^T y_3 \\ B^T y_1 - B^T y_2 + C^T y_3 = c \\ y_1, y_2, y_3 \geqslant 0. \end{array}$$

But this linear program is equivalent to the linear program

$$\begin{array}{r} \min b^T y_1 + d^T y_2 \\ B^T y_1 + C^T y_2 = c \\ y_2 \geqslant 0. \end{array} \tag{5.5}$$

This justifies to say that (5.5) is the dual of (5.4).

5.1 Zero sum games

Consider the following two-player game defined by a matrix $A \in \mathbb{R}^{m \times n}$. The *row-player* chooses a row $i \in \{1, \dots, m\}$ and the column-player chooses a column $j \in \{1, \dots, n\}$. Both players make this choice at the same time. The *payoff* for the row-player is then the matrix-element $A(i,j)$ whereas $A(i,j)$ also determines the *loss* of the column player. In other words, the column player pays $A(i,j)$ to the row-player. If this number is negative, then the row-player actually pays the absolute value of $A(i,j)$ to the column player.

Consider for example the matrix

$$A = \begin{pmatrix} 5 & 1 & 3 \\ 3 & 2 & 4 \\ -3 & 0 & 1 \end{pmatrix}. \tag{5.6}$$

If the row-player chooses the second row and the column player chooses the second-column, then the payoff for the row-player is 2, whereas this is the loss of the column player.

The row-player is now interested in finding a strategy that maximizes his *guaranteed* payoff. For example, if he chooses row 1, then the best choice of the column player would be column 2, since the second element of the first row is the smallest element of that row. Thus the strategy that maximizes the minimal possible payoff would be to choose row 2. In other words

$$\max_i \min_j A(i,j) = 2.$$

What would be the column-player's best hedging strategy? He wants to choose a column such that the largest element in this column is minimized. This column would be the second one. In other words

$$\min_j \max_i A(i,j) = 2.$$

Is it always the case that $\max_i \min_j A(i,j) = \min_j \max_i A(i,j)$? The next example shows that the answer is no:

$$\begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{5.7}$$

Here we have $\max_i \min_j A(i,j) = -1$ and $\min_j \max_i A(i,j) = 1$. This can be interpreted as follows. If the column player knows beforehand, the row to be chosen by the row-player, then he would choose a column that results in a gain for him. Similarly, if the row-player knows beforehand the column to be chosen by the column-player, then he can guarantee him a gain of one.

The idea is thus not to stick with a *pure* strategy, but to play with a *random* or *mixed* strategy. If the row-player chooses each of the two rows above uniformly at random, then his expected payoff is zero. Similarly, if the column player chooses each of his two columns with probability 1/2, then his expected payoff is zero as well.

Definition 5.1 (Mixed strategy). Let $A \in \mathbb{R}^{m\times n}$ define a two-player matrix game. A mixed strategy for the row-player is a vector $x \in \mathbb{R}^m_{\geqslant 0}$ with $\sum_{i=1}^m x_i = 1$. A mixed strategy for the column player is a vector $y \in \mathbb{R}^n_{\geqslant 0}$ with $\sum_{j=1}^n y_i = 1$.

Such mixed strategies define a probability distribution on the row and column indices respectively. If the row-player and column-player choose a row and column according to this distribution respectively, then the *expected payoff* for the row-player is

$$E[\text{Payoff}] = x^T A y. \tag{5.8}$$

For the game defined by (5.7) and $x^T = (1/2, 1/2)$ and $y^T = (1/2, 1/2)$ the expected payoff is 0.

Lemma 5.1. *Let $A \in \mathbb{R}^{m\times n}$, then*

$$\max_{x \in X} \min_{y \in Y} x^T A y \leqslant \min_{y \in Y} \max_{x \in X} x^T A y,$$

where X and Y denote the set of mixed row and column-strategies respectively.

Proof. Let x' and y' be some fixed mixed strategies of the row and column-player respectively. Clearly

$$\min_y {x'}^T A y \leqslant {x'}^T A y' \leqslant \max_x x^T A y',$$

which implies the assertion. □

The next theorem is one of the best-known results in the field of *game theory*. It states that there are mixed strategies x' and y' from above such that equality holds. It is proved with the theorem of strong duality.

Theorem 5.3 (Minimax-Theorem).

$$\max_{x \in X} \min_{y \in Y} x^T A y = \min_{y \in Y} \max_{x \in X} x^T A y,$$

where X and Y denote the set of mixed row and column-strategies respectively.

Proof. Let us inspect the value $\max_{x \in X} \min_{y \in Y} x^T A y$. This can be understood as to maximize the function

$$f(x) = \min\{(x^T A) \cdot y \colon \sum_{j=1}^{n} y_j = 1,\, y \geqslant 0\}.$$

Thus the value $f(x)$ is the optimal solution of a bounded and feasible linear program. The dual of this linear program (for fixed x) has only one variable x_0 and reads

$$\max\{x_0 \colon x_0 \in \mathbb{R},\, \mathbf{1} x_0 \leqslant A^T x\}.$$

But this shows that the maximum value of $f(x)$, where x ranges over all mixed row-strategies is the linear program

$$\begin{aligned} \max x_0 \\ \mathbf{1} x_0 - A^T x &\leqslant 0 \\ \textstyle\sum_{i=1}^m x_i &= 1 \\ x &\geqslant 0. \end{aligned} \tag{5.9}$$

Let us now inspect the value $\min_{y \in Y} \max_{x \in X} x^T A y$. Again, by applying duality this can be computed with the linear program

$$\begin{aligned} \min y_0 \\ \mathbf{1} y_0 - A y &\geqslant 0 \\ \textstyle\sum_{j=1}^n y_j &= 1 \\ y &\geqslant 0. \end{aligned} \tag{5.10}$$

It follows from the duality of (5.5) and (5.4) that the linear programs (5.9) and (5.10) are duals of each other. This proves the Minimax-Theorem. □

5.2 A proof of the duality theorem via Farkas' lemma

Remember Farkas' lemma (Theorem 2.11) which states that $Ax = b, x \geqslant 0$ has a solution if and only if for all $\lambda \in \mathbb{R}^m$ with $\lambda^T A \geqslant 0$ one also has $\lambda^T b \geqslant 0$. In fact the duality theorem follows from this. First, we derive another variant of Farkas' lemma.

Theorem 5.4 (Second variant of Farkas' lemma). *Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. The system $Ax \leqslant b$ has a solution if and only if for all $\lambda \geqslant 0$ with $\lambda^T A = 0$ one has $\lambda^T b \geqslant 0$.*

Proof. Necessity is clear: If x^* is a feasible solution, $\lambda \geqslant 0$ and $\lambda^T A = 0$, then $\lambda^T A x^* \leqslant \lambda^T b$ implies $0 \leqslant \lambda^T b$.

On the other hand, $Ax \leqslant b$ has a solution if and only if

$$Ax^+ - Ax^- + z = b,\ x^+, x^-, z \geqslant 0 \tag{5.11}$$

has a solution. So, if $Ax \leqslant b$ does not have a solution, then also (5.11) does not have a solution. By Farkas' lemma, there exists a $\lambda \in \mathbb{R}^m$ with $\lambda^T \begin{pmatrix} A & -A & I_m \end{pmatrix} \geqslant 0$ and $\lambda^T b < 0$. For this λ one also has $\lambda^T A = 0$ and $\lambda \geqslant 0$. □

We are now ready to prove the theorem of strong duality via the second variant of Farkas' lemma.

Proof (of strong duality via Farkas' lemma). Let δ be the objective function value of an optimal solution of the dual $\max\{b^T y \colon y \in \mathbb{R}^m,\ A^T y \leqslant c\}$. For all $\varepsilon > 0$, the system $A^T y \leqslant c, -b^T y \leqslant -\delta - \varepsilon$ does not have a solution. By the second variant of Farkas' lemma, there exists a $\lambda \geqslant 0$ with $\lambda^T \begin{pmatrix} -b^T \\ A^T \end{pmatrix} = 0$ and $\lambda^T \begin{pmatrix} -\delta-\varepsilon \\ c \end{pmatrix} < 0$. Write λ as $\lambda = \begin{pmatrix} \lambda_1 \\ \lambda' \end{pmatrix}$ with $\lambda' \in \mathbb{R}^n$. If λ_1 were zero, we could apply the second variant of Farkas' lemma to the system $A^T y \leqslant c$ and λ', since we know that $A^T y \leqslant c$ has a solution. Therefore, we can conclude $\lambda_1 > 0$. Furthermore, by scaling, we can assume $\lambda_1 = 1$. One has $\lambda'^T A^T = b^T$ and $\lambda'^T c < \delta + \varepsilon$. The first equation implies that λ' is a feasible solution of the primal (recall $\lambda' \geqslant 0$). The second equation shows that the objective function value of λ' is less than $\delta + \varepsilon$. This means that the optimum value of the primal linear program is also δ, since the primal has an optimal solution and ε can be chosen arbitrarily small. □

Exercises

1. Formulate the dual linear program of

$$\begin{aligned} \max\ 2x_1 + 3x_2 - 7x_3 & \\ x_1 + 3x_2 + 2x_3 &= 4 \\ x_1 + x_2 &\leqslant 8 \\ x_1 - x_3 &\geqslant -15 \\ x_1, x_2 &\geqslant 0 \end{aligned}$$

2. Consider the following linear program

$$\begin{aligned} \max\ x_1 + x_2 & \\ 2x_1 + x_2 &\leqslant 6 \\ x_1 + 2x_2 &\leqslant 8 \\ 3x_1 + 4x_2 &\leqslant 22 \\ x_1 + 5x_2 &\leqslant 23 \end{aligned}$$

   Show that $(4/3, 10/3)$ is an optimal solution by providing a suitable feasible dual solution.
3. Show that for $A \in \mathbb{R}^{m\times n}$, one has

$$\max_i \min_j A(i,j) \leqslant \min_j \max_i A(i,j).$$

4. In the lecture you have seen the simplex algorithm for linear programs of the form

$$\max\{c^T x \ : \ Ax \leqslant b\}.$$

   We will now derive a simplex algorithm for linear programs of the form

$$\min\{c^T x \ : \ Ax = b,\ x \geqslant 0\} \tag{5.12}$$

   with $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Throughout the exercise we assume that (5.12) is feasible and bounded, and that A has full row rank.
   For $i \in \{1,\dots,n\}$ we define A_i as the i-th column of A. Moreover, for some subset $B \subseteq \{1,\dots,n\}$, A_B is the matrix A restricted to the columns corresponding to elements of B.
   A subset $B \subseteq \{1,\dots,n\}$ with $|B| = m$ such that A_B has full rank is called a *basis*. The vector $x \in \mathbb{R}^n$ defined as $x_i := 0$ for all $i \notin B$ and $x_B := A_B^{-1} b$ is called the *basic solution* associated to B. Note that x is a feasible solution to (5.12) if and only if $x \geqslant 0$.
   Given a basis B and let $j \in \{1,\dots,n\}$, $j \notin B$. The vector $d \in \mathbb{R}^n$ defined as $d_j = 1$, $d_i = 0$ for all $i \notin B$ and $d_B := -A_B^{-1} A_j$ is called the *j-th basic direction*.
   Assume that the solution x associated to B is feasible. Moreover assume that $x_B > 0$.

a. Show that there is a $\theta > 0$ such that $x + \theta d$ is a feasible solution. Give a formula to compute the largest θ such that $x + \theta d$ is feasible.
b. Let θ^* be maximal. Show that there is a basis B' such that $x + \theta^* d$ is the basic solution associated to B'.
c. Let $x' = x + \theta d$. Show that the objective value of x' changes by $\theta \left(c_j - c_B^T A_B^{-1} A_j\right)$.
d. Consider a basis B with basic feasible solution x. Show that if $c - c_B^T A_B^{-1} A \geqslant 0$, then x is an optimal solution to (5.12).

This suggests the following algorithm: Start with some basis B whose associated basic solution is feasible. Compute $\bar{c} := c - c_B^T A_B^{-1} A$. If $\bar{c} \geqslant 0$, we have an optimal solution (see 4d). Otherwise, let j be such that $\bar{c}_j < 0$. Part 4b and 4c show that if we change the basis, we find a feasible solution with an improved objective value. We repeat these steps until the vector $\bar{c}$ is nonnegative.
This is the way the simplex algorithm usually is introduced in the literature. This algorithm is exactly the same as the one you learned in the lecture. To get an intuition why this is true, show the following:

a. Given a basis B, show that its associated basic solution is feasible if and only if B is a *basis* of the LP dual to (5.12).
b. Consider a basis B and its associated feasible basic solution x. As seen before, B is also a basis in the dual LP. Let y be the vertex of that basis.
   Show that for any $j \in \{1, \ldots, n\}$ we have $\bar{c}_j < 0$ if and only if $A_j^T y > c_j$.

Chapter 6
Integer Programming

An *integer program* is a problem of the form

$$\begin{aligned} \max c^T x \\ Ax \leqslant b \\ x \in \mathbb{Z}^n, \end{aligned}$$

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

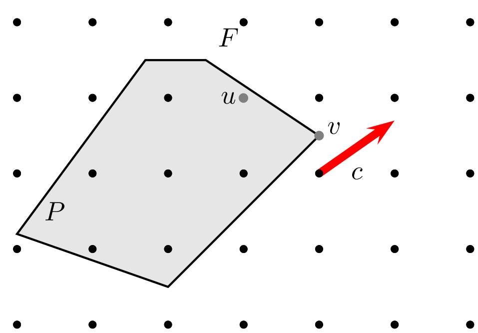


Fig. 6.1: This picture illustrates a polyhedron P, an objective function vector c and optimal points u, v of the integer program and the relaxation respectively.

The difference to linear programming is the *integrality constraint* $x \in \mathbb{Z}^n$. This powerful constraint allows to model discrete choices but, at the same time, makes an integer program much more difficult to solve than a linear program. In fact one can show that integer programming is NP-hard, which means that it is *in theory* computationally intractable. However, integer programming has nowadays become an important tool to solve difficult industrial optimization problems efficiently. In this chapter, we characterize some integer programs which are easy to solve, since the *linear programming relaxation* $\max\{c^T x \colon Ax \leqslant b\}$ yields already an optimal integer solution. The following observation is crucial.

Theorem 6.1. *Suppose that x^* is an integral optimum solution of the linear programming relaxation* $\max\{c^T x \colon Ax \leqslant b\}$, *i.e., $x^* \in \mathbb{Z}^n$, then x^* is also an optimal solution of the integer programming problem* $\max\{c^T x \colon Ax \leqslant b,\ x \in \mathbb{Z}^n\}$

Before we present an example for the power of integer programming we recall the definition of an undirected graph.

Definition 6.1 (Undirected graph, matching). An *undirected graph* is a tuple $G = (V, E)$ where V is a finite set of elements, called the *vertices* or the *nodes*, and $E \subseteq \binom{V}{2}$ is the set of *edges* of G. A *matching* of G is a subset $M \subseteq E$ such that for all $e_1 \neq e_2 \in M$ one has $e_1 \cap e_2 = \emptyset$.

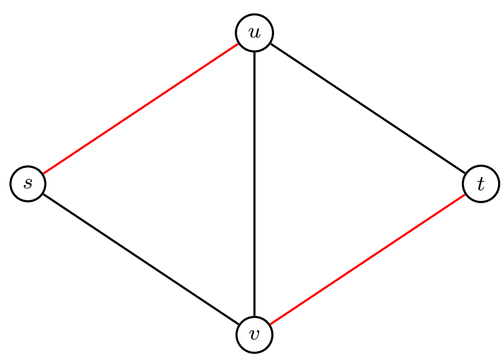


Fig. 6.2: A graph with 4 nodes $V = \{s, u, v, t\}$ and 5 edges $E = \{\{s, u\}, \{s, v\}, \{u, v\}, \{u, t\}, \{v, t\}\}$. The red edges are a matching of the graph

We are interested in the solution of the following problem, which is called *maximum weight matching* problem. Given a graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$, compute a matching with maximum weight $w(M) = \sum_{e \in M} w(e)$.

For a vertex $v \in V$, the set $\delta(v) = \{e \in E \colon v \in e\}$ denotes the *incident* edges to v. The maximum weight matching problem can now be modeled as an integer program as follows.

$$\begin{array}{c} \max \sum_{e \in E} w(e) x(e) \\ v \in V : \sum_{e \in \delta(v)} x(e) \leqslant 1 \\ e \in E : x(e) \geqslant 0 \\ x \in \mathbb{Z}^{|E|}. \end{array} \tag{6.1}$$

Clearly, if an integer vector $x \in \mathbb{Z}^n$ satisfies the constraints above, then this vector is the *incidence vector* of a matching of G. In other words, the integral solutions to the constraints above are the vectors $\{\chi^M : M \text{ matching of } G\}$, where $\chi^M(e) = 1$ if $e \in M$ and $\chi^M(e) = 0$ otherwise.

6.1 Integral Polyhedra

In this section we derive sufficient conditions on an integer program to be solved easily by an algorithm for linear programming. A central notion is the one of an integral polyhedron.

Definition 6.2 (Valid inequality, face, vertex). Let $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ be a polyhedron. An inequality $c^T x \leqslant \beta$ is *valid* for P if $c^T x^* \leqslant \beta$ for all $x^* \in P$. A *face* of P is a set of the form $P \cap \{x \in \mathbb{R}^n : c^T x = \beta\}$ for a valid inequality $c^T x \leqslant \beta$ of P. If a face consist of one point, then it is called a *vertex* of P.

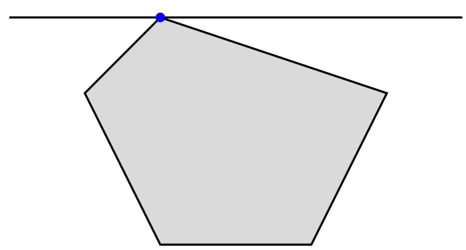


Fig. 6.3: A polyhedron with a valid inequality defining a vertex.

Definition 6.3. A rational polyhedron is called *integral* if each nonempty face of P contains an integer vector.

Lemma 6.1. *Let $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ be an integral polyhedron with $A \in \mathbb{R}^{m \times n}$ full-column rank. If the linear program*

$$\max\{c^T x : x \in \mathbb{R}^n,\, Ax \leqslant b\} \tag{6.2}$$

is feasible and bounded, then the simplex method computes an optimal integral solution to the linear program.

Proof. Recall that, if the linear program (6.2) is bounded, the simplex method finds an optimal basis $B \subseteq \{1, \ldots, m\}$ of (6.2) and the vertex of the basis

x_B^* is an optimal solution to (6.2). We have to show that x_B^* is integral. This will follow from the fact that $\{x_B^*\}$ is a face of P.

Theorem 2.1 implies that x_B^* is the unique optimum solution of the linear program $\max\{\widetilde{c}^T x \colon x \in R^n,\, a_i^T x \leqslant b(i),\, i \in B\}$, where $\widetilde{c} = \sum_{i \in B} a_i$. Consequently x_B^* is the unique solution of the linear program

$$\max\{\widetilde{c}^T x \colon x \in P\}$$

which implies that $\{x_B^*\}$ is a face defined by the valid inequality $\widetilde{c}^T x \leqslant \widetilde{c}^T x_B^*$. □

Lemma 6.2. *Let $A \in \mathbb{Z}^{n \times n}$ be an integral and invertible matrix. One has $A^{-1} b \in \mathbb{Z}^n$ for each $b \in \mathbb{Z}^n$ if and only if $\det(A) = \pm 1$.*

Proof. Recall Cramer's rule which says $A^{-1} = \widetilde{A}/\det(A)$, where $\widetilde{A}$ is the adjoint matrix of A. Clearly $\widetilde{A}$ is integral. If $\det(A) = \pm 1$, then A^{-1} is an integer matrix.

If $A^{-1} b$ is integral for each $b \in \mathbb{Z}^n$, then A^{-1} is an integer matrix. We have $1 = \det(A \cdot A^{-1}) = \det(A) \cdot \det(A^{-1})$. Since A and A^{-1} are integral it follows that $\det(A)$ and $\det(A^{-1})$ are integers. The only divisors of one in the integers are ± 1. □

Definition 6.4 (Total unimodularity). An integral matrix $A \in \{0, \pm 1\}^{m \times n}$ is called *totally unimodular* if each of its square sub-matrices has determinant $0, \pm 1$.

Theorem 6.2 (Hoffman-Kruskal Theorem). *Let $A \in \mathbb{Z}^{m \times n}$ be an integral matrix. The polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b,\, x \geqslant 0\}$ is integral for each integral $b \in \mathbb{Z}^m$ if and only if A is totally unimodular.*

Proof. Let $A \in \mathbb{Z}^{m \times n}$ be totally unimodular and $b \in \mathbb{Z}^m$. Let x^* be vertex of P and suppose that this vertex is defined by the valid inequality $c^T x \leqslant \delta$. Notice that the matrix $\binom{A}{-I}$ has full column-rank. If one applies the simplex algorithm to the problem

$$\max\{c^T x \colon x \in \mathbb{R}^n,\, \tbinom{A}{-I} x \leqslant \tbinom{b}{0}\},$$

it finds an optimal basis $B \subseteq \{1, \ldots, m+n\}$ with $x_B^* = x^*$. If A_B denotes the matrix whose rows are those rows of $\binom{A}{-I}$ indexed by B and if b_B denotes the vector whose components are those of $\binom{b}{0}$ indexed by B, then $x^* = A_B^{-1} b_B$. We are done, once we conclude that $\det(A_B) = \pm 1$, since then A_B^{-1} is an integer matrix and since b_B is an integer vector $x^* = A_B^{-1} b$ is integral as well. We can permute the columns of A_B in such a way that one obtains a matrix of the form

$$\begin{pmatrix} \overline{A} & \widetilde{A} \\ 0 & -I_k \end{pmatrix}$$

where $\overline{A}$ is a $(n-k)\times(n-k)$ sub-matrix of A and I_k is the $k\times k$ identity matrix. Here $k = |B \cap \{m+1,\dots,m+n\}|$. Clearly $0 \neq \det(A_B) = \pm\det(\overline{A}) = \pm 1$.

For the converse, suppose that A is not totally unimodular. Then there exists an index set $B \subseteq \{1,\dots,m+n\}$ with $|B| = n$ such that the matrix A_B defined as above satisfies $|\det(A_B)| \geqslant 2$. We can suppose w.l.o.g. that $B = \{1,\dots,n\}$. By Lemma 6.2 there exists choices for the components of b_B making $A_B^{-1}b_B$ non-integral. In fact, if we split B into components $L \subseteq B$ corresponding to lines of A and C corresponding to lines of $-I$ we can choose those components of b_B corresponding to L being equal to zero. Now let v be the vector with $v_i = 1$ for all $i \in C$ and $v_i = 0$ for all $i \in L$. By choosing $\gamma \in \mathbb{N}$ large enough the point $x_B^* = A_B^{-1}(b_B + \gamma A_B v)$ is non-integral and positive. Notice that starting from now we will consider a new vector $\tilde{b}$ instead of b, where $\tilde{b}_B = b_B$. In the next lines we will say $\tilde{b}_{\{1,\dots,m+n\}\setminus B}$ has to be to finish the proof. The set B is a basis of the linear program

$$\max\{\overline{c}^T x \colon x \in \mathbb{R}^n,\ \left(\begin{smallmatrix} A \\ -I \end{smallmatrix}\right)x \leqslant \left(\begin{smallmatrix} \tilde{b} \\ 0 \end{smallmatrix}\right)\},$$

where $\overline{c} = \sum_{i\in B} a_i$ and a_i denotes the i-th row of $\left(\begin{smallmatrix} A \\ -I \end{smallmatrix}\right)$. If we define for $j \in \{1,\dots,m\} \setminus B$, $\tilde{b}_j = \lceil a_j^T x_B^* \rceil$, then x_B^* is feasible and thus a vertex of P that is non-integral. □

6.2 Applications of total unimodularity

6.2.1 Bipartite matching

A graph is *bipartite*, if V has a partition into sets A and B such that each edge uv satisfies $u \in A$ and $v \in B$. Recall that $\delta(v)$ is the set of edges incident to the vertex $v \in V$, that is $\delta(v) = \{e \in E \mid v \in e\}$.

The *node-edge* incidence matrix of a graph $G = (V,E)$ is the $A \in \{0,1\}^{|V|\times|E|}$ with

$$A(v,e) = \begin{cases} 1, & \text{if } v \in e, \\ 0 & \text{otherwise.} \end{cases}$$

The integer program (6.1) can thus be formulated as

$$\max\{w^T x \colon Ax \leqslant 1,\ x \geqslant 0,\ x \in \mathbb{Z}^E\}. \tag{6.3}$$

The next lemma implies that the simplex algorithm can be used to compute a maximum-weight matching of a bipartite graph.

Lemma 6.3. *If G is bipartite, the node-edge incidence matrix of G is totally unimodular.*

Proof (Proof of Lemma 6.3). Let $G = (V, E)$ be a bipartite graph with bipartition $V = V_1 \cup V_2$.

Let A' be a $k \times k$ sub-matrix of A. We are interested in the determinant of A. Clearly, we can assume that A does not contain a column which contains no 1 or only one 1, since we simply consider the sub-matrix A'' of A', which emerges from developing the determinant of A' along this column. The determinant of A' would be zero or $\pm 1 \cdot \det(A'')$.

Thus we can assume that each column contains exactly two ones. Now we can order the rows of A' such that the first rows correspond to vertices of V_1 and then follow the rows corresponding to vertices in V_2. This re-ordering only affects the sign of the determinant. By summing up the rows of A' in V_1 we obtain exactly the same row-vector as we get by summing up the rows of A' corresponding to V_2. This shows that $\det(A') = 0$. □

6.2.2 Bipartite vertex cover

A *vertex cover* of a graph $G = (V, E)$ is a subset $C \subseteq V$ of the nodes such $e \cap C \neq \emptyset$ for each $e \in E$. Let us formulate an integer program for the *minimum-weight vertex-cover* problem. Here, one is given a graph $G = (V, E)$ and weights $w \in \mathbb{R}^V$. The goal is to find a vertex cover C with minimum weight $w(C) = \sum_{v \in V} w(v)$.

$$\begin{array}{c} \min \sum_{v \in V} w(v) x(v) \\ uv \in E : x(u) + x(v) \geqslant 1 \\ v \in V : \, x(v) \geqslant 0 \\ x \in \mathbb{Z}^V . \end{array} \tag{6.4}$$

Clearly, this is the integer program

$$\min\{w^T x \colon A^T x \geqslant 1, \, x \geqslant 0, \, x \in \mathbb{Z}^V\}, \tag{6.5}$$

where A is the node-edge incidence matrix of G. A matrix A is totally unimodular if and only of A^T is totally unimodular. Thus the simplex algorithm can be used to compute a minimum-weight vertex-cover of a bipartite graph. Furthermore we have the following theorem.

Theorem 6.3 (König's theorem). *In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover.*

Proof. Let A be the node-edge incidence-matrix of the bipartite graph $G = (V, E)$. The linear programs $\max\{1^T x \colon Ax \leqslant 1, \, x \geqslant 0\}$ and $\min\{1^T x \colon Ax \geqslant 1, \, x \geqslant 0\}$ are duals of each other. Since A is totally unimodular, the value of the linear programs are the cardinality of a maximum matching and minimum vertex-cover respectively. Thus the theorem follows from strong duality. □

6.2.3 Flows

Let $G = (V, A)$ be a directed graph. The *node-edge incidence matrix of a directed graph* is a matrix $A \in \{0, \pm 1\}^{V \times E}$ with

$$A(v,a) = \begin{cases} 1 & \text{if } v \text{ is the starting-node of } a, \\ -1 & \text{if } v \text{ is the end-node of } a, \\ 0 & \text{otherwise.} \end{cases} \tag{6.6}$$

A *feasible flow* f of G with capacities u and in-out-flow b is then a solution $f \in \mathbb{R}^A$ to the system $A\,f = b$, $0 \leqslant f \leqslant u$.

Lemma 6.4. *The node-edge incidence matrix A of a directed graph is totally unimodular.*

Proof. Let A' be a $k \times k$ sub-matrix of A. Again, we can assume that in each column we have exactly one 1 and one -1. Otherwise, we develop the determinant along a column which does not have this property. But then, the A' is singular, since adding up all rows of A' yields the 0-vector.

A consequence is that, if the b-vector and the capacities u are integral and an optimal flow exists, then there exists an integer optimal flow.

6.2.4 Doubly stochastic matrices

A matrix $A \in \mathbb{R}^{n \times n}$ is *doubly stochastic* if it satisfies the following linear constraints

$$\begin{aligned} \textstyle\sum_{i=1}^n A(i,j) &= 1, \ \forall j = 1, \dots, n \\ \textstyle\sum_{j=1}^n A(i,j) &= 1, \ \forall i = 1, \dots, n \\ A(i,j) &\geqslant 0, \ \forall 1 \leqslant i, j \leqslant n. \end{aligned} \tag{6.7}$$

A permutation matrix is a matrix which contains exactly one 1 per row and column, where the other entries are all 0.

Theorem 6.4. *A matrix $A \in \mathbb{R}^{n \times n}$ is doubly stochastic if and only if A is a convex combination of permutation matrices.*

Proof. Since a permutation matrix satisfies the constraints (6.7), then so does a convex combination of these constraints.

On the other hand it is enough to show that each vertex of the polytope defined by the system (6.7) is integral and thus a permutation matrix. However, the matrix defining the system (6.7) is the node-edge incidence matrix of the complete bipartite graph having $2n$ vertices. Since such a matrix is totally unimodular, the theorem follows.

6.3 The matching polytope

We now come to a deeper theorem concerning the convex hull of matchings. We mentioned several times in the course that the maximum weight matching problem can be solved in polynomial time. We are now going to show a theorem of Edmonds [1] which provides a complete description of the matching polytope and present the proof by Lovász [10].

Before we proceed let us inspect the symmetric difference $M_1 \Delta M_2$ of two matchings of a graph G. If a vertex is adjacent to two edges of $M_1 \cup M_2$, then one of the two edges belongs to M_1 and one belongs to M_2. Also, a vertex can never be adjacent to three edges in $M_1 \cup M_2$. Edges which are both in M_1 and M_2 do not appear in the symmetric difference. We therefore have the following lemma.

Lemma 6.5. *The symmetric difference $M_1 \Delta M_2$ of two matchings decomposes into node-disjoint paths and cycles, where the edges on these paths and cycles alternate between M_1 and M_2.*

The *Matching polytope* $P(G)$ of an undirected graph $G = (V, E)$ is the convex hull of incidence vectors χ^M of matchings M of G.

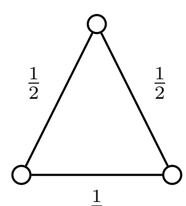


Fig. 6.4: Triangle

The incidence vectors of matchings are exactly the 0/1-vectors that satisfy the following system of equations.

$$\begin{aligned} \textstyle\sum_{e \in \delta(v)} x(e) &\leqslant 1 \; \forall v \in V \\ x(e) &\geqslant 0 \; \forall e \in E. \end{aligned} \tag{6.8}$$

However the triangle (Figure 6.4) shows that the corresponding polytope is not integral. The objective function $\max 1^T x$ has value 1.5. However, one can show that a maximum weight matching of an undirected graph can be computed in polynomial time which is a result of Edmonds [2].

The following (Figure 6.5) is an illustration of an Edmonds inequality. Suppose that U is an odd subset of the nodes V of G and let M be a matching of G. The number of edges of M with both endpoints in U is bounded from above by $\lfloor |U|/2 \rfloor$.

Thus the following inequality is valid for the integer points of the polyhedron defined by (6.8).

$$\sum_{e \in E(U)} x(e) \leqslant \lfloor |U|/2 \rfloor, \qquad \text{for each } U \subseteq V, \quad |U| \equiv 1 \pmod 2. \tag{6.9}$$

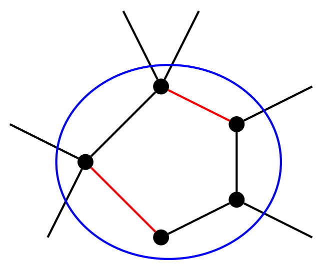


Fig. 6.5: Edmonds inequality.

The goal of this lecture is a proof of the following theorem.

Theorem 6.5 (Edmonds 65). *The matching polytope is described by the following inequalities:*

i) $x(e) \geqslant 0$ *for each* $e \in E$,
ii) $\sum_{e \in \delta(v)} x(e) \leqslant 1$ *for each* $v \in V$,
iii) $\sum_{e \in E(U)} x(e) \leqslant \lfloor |U|/2 \rfloor$ *for each* $U \subseteq V$

Lemma 6.6. *Let* $G = (V, E)$ *be connected and let* $w : E \longrightarrow \mathbb{R}_{>0}$ *be a weight-function. Denote the set of maximum weight matchings of* G *w.r.t.* w *by* $\mathscr{M}(w)$. *Then one of the following statements must be true:*

i) $\exists\, v \in V$ *such that* $\delta(v) \cap M \neq \emptyset$ *for each* $M \in \mathscr{M}(w)$
ii) $|M| = \lfloor |V|/2 \rfloor$ *for each* $M \in \mathscr{M}(w)$ *and* $|V|$ *is odd.*

Proof. Suppose both *i*) and *ii*) do not hold. Then there exists $M \in \mathscr{M}(w)$ leaving two exposed nodes u and v. Choose M such that the minimum distance between two exposed nodes u, v is minimized.

Now let t be on shortest path from u to v. The vertex t cannot be exposed.

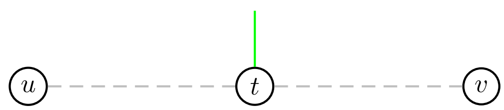


Fig. 6.6: Shortest path between u and v.

Let $M' \in \mathscr{M}(w)$ leave t exposed. Both u and v are covered by M' because the distance to u or v from t is smaller than the distance of u to v.

Consider the symmetric difference $M \triangle M'$ which decomposes into node disjoint paths and cycles. The nodes u, v and t have degree one in $M \triangle M'$. Let P be a path with endpoint t in $M \triangle M'$

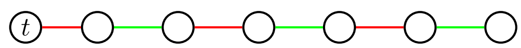


Fig. 6.7: Swapping colors.

If we swap colors on P, see Figure 6.7, we obtain matchings $\widetilde{M}$ and $\widetilde{M'}$ with $w(M) + w(M') = w(\widetilde{M}) + w(\widetilde{M'})$ and thus $\widetilde{M} \in \mathscr{M}(w)$.

The node t is exposed in $\widetilde{M}$ and u or v is exposed in $\widetilde{M}$. This is a contradiction to u and v being shortest distance exposed vertices

Proof (Proof of Theorem 6.5).

Let $w^T x \leqslant \beta$ be a *facet* of $P(G)$, we need to show that this facet it is of the form

i) $x(e) \geqslant 0$ for some $e \in E$
ii) $\sum_{e \in \delta(v)} x(e) \leqslant 1$ for some $v \in V$
iii) $\sum_{e \in E(U)} x(e) \leqslant \lfloor |U|/2 \rfloor$ for some $U \in P_{odd}$

To do so, we use the following method: One of the inequalities i), ii), iii) is satisfied with equality by each χ^M, $M \in \mathscr{M}(w)$. This establishes the claim since the matching polytope is full-dimensional and a facet is a maximal face.

If $w(e) < 0$ for some $e \in E$, then each $M \in \mathscr{M}(w)$ satisfies $e \notin M$ and thus satisfies $x(e) \geqslant 0$ with equality.

Thus we can assume that $w \geqslant 0$.

Let $G^* = (V^*, E^*)$ be the graph induced by edges e with $w(e) > 0$. Each $M \in \mathscr{M}(w)$ contains maximum weight matching $M^* = M \cap E^*$ of G^* w.r.t. w^*.

If G^* is not *connected* , suppose that $V^* = V_1 \cup V_2$, where $V_1 \cap V_2 = \emptyset$ and $V_1, V_2 \neq \emptyset$ and there is no edge connecting V_1 and V_2, then $w^T x \leqslant \beta$ can be written as the sum of $w_1^T x \leqslant \beta_1$ and $w_2^T x \leqslant \beta_2$, where β_i is the maximum weight of a matching in V_i w.r.t. w_i, $i = 1, 2$, see Figure 6.8. This would also contradict the fact that $w^T x \leqslant \beta$ is a facet, since it would follow from the previous inequalities and thus would be a redundant inequality.

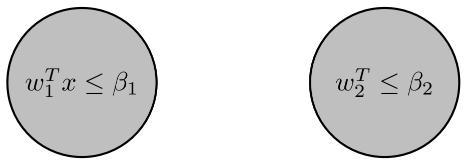


Fig. 6.8: G^* is connected.

Now we can use Lemma 6.6 for G^*.

i) $\exists v$ such that $\delta(v) \cap M = \emptyset$ for each $M \in \mathscr{M}(w)$. This means that each M in $\mathscr{M}(w)$ satisfies

$$\sum_{e \in \delta(v)} x(e) \leqslant 1 \quad \text{with equality}$$

ii) $|M \cap E^*| = \lfloor |V^*|/2 \rfloor$ for each $M \in \mathscr{M}(w)$ and $|V^*|$ is odd. This means that each M in $\mathscr{M}(w)$ satisfies

$$\sum_{e \in E(V^*)} x(e) \leqslant \lfloor |V^*|/2 \rfloor \quad \text{with equality}$$

Exercises

1. Let $M \in \mathbb{Z}^{n \times m}$ be totally unimodular. Prove that the following matrices are totally unimodular as well:
   i) M^T
   ii) $(M \quad I_n)$
   iii) $(M \quad -M)$

iv) $M \cdot (I_n - 2e_j e_j^T)$ for some j

I_n is the $n \times n$ identity matrix, and e_j is the vector having a 1 in the j^{th} component, and 0 in the other components.

2. A family $\mathcal{F}$ of subsets of a finite groundset E is *laminar*, if for all $C, D \in \mathcal{F}$, one of the following holds:

$$(i)\ C \cap D = \emptyset, \quad (ii)\ C \subseteq D, \quad (iii)\ D \subseteq C.$$

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two laminar families of the same groundset E and consider its union $\mathcal{F}_1 \cup \mathcal{F}_2$. Define the $|\mathcal{F}_1 \cup \mathcal{F}_2| \times |E|$ adjacency matrix A as follows: For $F \in \mathcal{F}_1 \cup \mathcal{F}_2$ and $e \in E$ we have $A_{F,e} = 1$, if $e \in F$ and $A_{F,e} = 0$ otherwise.
Show that A is totally unimodular.

3. Consider the following scheduling problem: Given n tasks with periods $p_1, \ldots, p_n \in \mathbb{N}$, we want to find offsets $x_i \in \mathbb{N}_0$, such that every task i can be executed periodically at times $x_i + p_i \cdot k$ for all $k \in \mathbb{N}_0$. In other words, for all pairs i, j of tasks we require $x_i + k \cdot p_i \neq x_j + l \cdot p_j$ for all $k, l \in \mathbb{N}_0$.
Formulate the problem of finding these offsets as an integer program (with zero objective function).

4. Let $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ be a polyhedron. Show that the following are equivalent for a feasible x^*:

   i) x^* is a vertex of P.
   ii) There exists a set $B \subseteq \{1, \ldots, m\}$ such that $|B| = n$, A_B is invertible and $A_B x^* = b_B$. Here the matrix A_B and the vector b_B consists of the rows of A indexed by B and the components of b indexed by B respectively.
   iii) For every feasible $x_1, x_2 \neq x^* \in P$ one has $x^* \notin \mathrm{conv}\{x_1, x_2\}$.

5. Show the following: A polyhedron $P \subseteq \mathbb{R}^n$ with vertices is integral, if and only if each vertex is integral.
6. Consider the polyhedron $P = \{x \in \mathbb{R}^3 \colon x_1 + 2\,x_2 + 4\,x_3 \leqslant 4,\ x \geqslant 0\}$. Show that this polyhedron is integral.
7. Which of these matrices is totally unimodular? Justify your answer.

$$\begin{pmatrix} 1&1&0&1&0\\ 0&0&1&0&1\\ 0&1&1&1&0\\ 1&1&0&0&0\\ 0&1&0&0&1 \end{pmatrix} \qquad \begin{pmatrix} 1&1&1&1&1\\ 1&1&0&0&0\\ 1&1&1&0&0\\ 1&1&1&1&0 \end{pmatrix}$$

8. Consider the complete graph G_n with 3 vertices, i.e., $G = (\{1,2,3\}, \binom{3}{2})$. Is the polyhedron of the linear programming relaxation of the vertex-cover integer program integral?

Chapter 7
Paths, cycles and flows in graphs

Suppose you want to find a shortest path from a given starting point to a given destination. This is a common scenario in driver assistance systems (GPS) and can be modeled as one of the most basic combinatorial optimization problems, the *shortest path problem*. In this chapter, we introduce directed graphs, shortest paths and flows in networks. We focus in particular on the maximum-flow problem, which is a linear program that we solve with direct methods, versus the simplex method, and analyze the running time of these direct methods.

7.1 Graphs

Definition 7.1. A *directed graph* is a tuple $G = (V, A)$, where V is a finite set of elements, called the *vertices* of G and $A \subseteq (V \times V)$ is the set of *arcs* of G. We denote an arc by its two defining nodes $(u, v) \in A$. The nodes u and v are called *tail* and *head* of the arc (u, v) respectively.

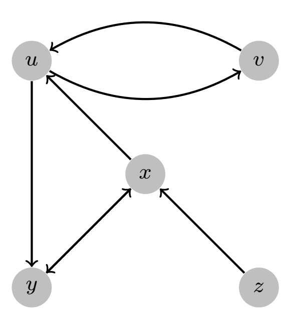


Fig. 7.1: Example of a directed graph with 5 nodes and 7 arcs.

Definition 7.2 (Walk, path, distance). A *walk* is a sequence of the form

$$P = (v_0, a_1, v_1, \ldots, v_{m-1}, a_m, v_m),$$

where $a_i = (v_{i-1}, v_i) \in A$ for $i = 1, \ldots, m$. If the nodes $v_0, \ldots, v_m$ are all different, then P is a *path*. The *length* of P is m. The *distance* of two nodes u and v is the length of a shortest path from u to v. It is denoted by $d(u, v)$.

Example 7.1. The following is a walk and a path of the graph in Figure 7.1.

$$z, (z, x), x, (x, u), u, (u, v), v, (v, u), u, (u, y), y, (y, x), x$$
$$y, (y, x), x, (x, u), u, (u, v), v$$

7.2 Representing graphs and computing the distance of two nodes

We represent a graph with n vertices $v_1, \ldots, v_n$ as an array $A[v_1, \ldots, v_n]$, where the entry $A[v_i]$ is a pointer to a linked list of vertices, the *neighbours of* v_i. $N(v_i) = \{u \in V : (v_i, u) \in A\}$.

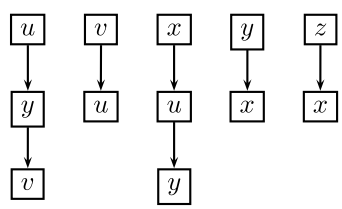


Fig. 7.2: Adjacency list representation of the graph in Figure 7.1.

7.2.1 Breadth first search

We next describe a very basic algorithm that computes the distances from a designated node $s \in V$ to all other nodes. The *distance* from s to v is denoted by $d(s, v)$. It is the smallest integer i such that there exists a path from s to v of length i. If there does not exist such a path, then s and v are *not connected* and we define $d(s, v) = \infty$. For $i \in \mathbb{N}_0$, $V_i \subseteq V$ denotes the set of vertices that have distance i from s. Notice that $V_0 = \{s\}$.

Lemma 7.1. *For $i = 1, \ldots, n-1$, the set V_i is equal to the set of vertices $v \in V \setminus (V_0 \cup \cdots \cup V_{i-1})$ such that there exists an arc $(u, v) \in A$ with $u \in V_{i-1}$.*

Proof. Suppose that $v \notin V_0 \cup \cdots \cup V_{i-1}$ and there exists an arc $uv \in A$ with $u \in V_{i-1}$. Since $u \in V_{i-1}$, there exists a path $s, a_1, v_1, a_2, v_2, \ldots, a_{i-1}, u$ of length $i-1$ from s to u. The sequence $s, a_1, v_1, a_2, v_2, \ldots, a_i, u, uv, v$ is a path of length i from s to v and thus $v \in V_i$.

If, on the other hand, $v \in V_i$, then there exists a path

$$s, a_1, v_1, \ldots, a_{i-1}, u, a_i, v$$

of length i from s to v. We need to show that $u \in V_{i-1}$ holds. Clearly, since there exists a path of length $i-1$ from s to u, one has $u \in V_j$ with $j \leqslant i-1$. If $j < i-1$, then there exists a path $s, a_1', v_1', \ldots, a_j', u$ of length j which can be extended to a path of length $j+1 < i$ from s to v

$$s, a_1', v_1', \ldots, a_j', u, a_i, v$$

which contradicts $v \in V_{i+1}$. ☐

The *breadth-first search algorithm* is an implementation of Lemma 7.1. The algorithm maintains arrays

$$D[v_1 = s, v_2, \ldots, v_n]$$
$$\pi[v_1 = s, v_2, \ldots, v_n]$$

and a queue Q that contains only s in the beginning. The array D contains at termination of the algorithm the distances from s to all other nodes and is initialized with $[0, \infty, \ldots, \infty]$. The array π contains predecessor information for shortest paths, in other words, when the algorithm terminates, $\pi[v] = u$, where uv is an arc and $D[u] + 1 = D[v]$. The array π is initialized with $[0, \ldots, 0]$.

After this initialization, the algorithm proceeds as follows.

```
while Q ≠ ∅
        u := head(Q)
        for each v ∈ N(u)
            if (D[v] = ∞)
              π[v] := u
              D[v] := D[u] + 1
              enqueue(Q, v)
        dequeue(Q)
```

Here the function *head*(Q) returns the next element in the queue and *dequeue*(Q) removes the first element of Q, while *enqueue*(Q, v) adds v to the queue as last element.

Lemma 7.2. *The breadth-first search algorithm assigns* distance labels D *correctly.*

Proof. We show the following claim by induction on $i \in \{0, \dots, n-1\}$.

For each $i \in \{1, \dots, n-1\}$ there exists a point in time where:

i) Q contains precisely the elements of V_i
ii) for each $v \in V_i$, $D[v] = d(s, v)$
iii) for each $v \in V_i$ one has $\pi[v]v$ is an arc and $\pi[v] \in V_{i-1}$.

Once this claim is shown, the lemma follows, because the labels $D[v]$ and $\pi[v]$ are only changed once, if at all, from ∞ to an integer or a vertex respectively.

Since $V_0 = \{s\}$ and since $Q = [s]$ and $D[s] = 0$ after the initialization, the claim holds for $i = 0$. Suppose $i > 0$. By the induction hypothesis, there is a point in time, where Q contains precisely V_{i-1}. By Lemma 7.1, after the last element of V_{i-1} is dequeued Q contains precisely the elements in V_i. Also, since $D[u] = d(s, u) = i - 1$ for all $u \in V_{i-1}$ we have for each $v \in V_i$ that $D[v] = d(s, v) = i$. Also $\pi[v]v$ is an arc, by virtue of the algorithm, and $\pi[v] \in V_{i-1}$. □

Definition 7.3 (Tree). A *directed tree* is a directed graph $T = (V, A)$ with $|A| = |V| - 1$ and there exists a node $r \in T$ such that there exists a path from r to all other nodes of T.

Lemma 7.3. *Consider the arrays D and π after the termination of the breadth-first-search algorithm. The graph $T = (V', A')$ with $V' = \{v \in V \colon D[v] < \infty\}$ and $A' = \{\pi(v)v \colon 1 \leqslant D[v] < \infty\}$ is a tree.*

Proof. Clearly, $|A'| = |V'| - 1$. For any $i \in \{1, \dots, n-1\}$, by backtracking the π-labels from any $v \in V_i$, we will eventually reach s.

Definition 7.4. The tree T from above is the *shortest-path-tree* of the (unweighted) directed graph $G = (V, A)$.

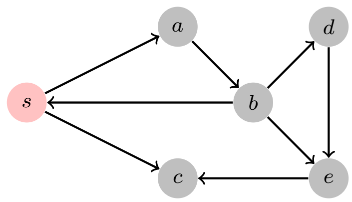
Theorem 7.1. *The breath-first-search algorithm runs in time $O(|V| + |A|)$.*

Proof. Each vertex is queued and dequeued at most once. These queuing operations take constant time each. Thus queuing and dequeuing costs $O(|V|)$ in total.

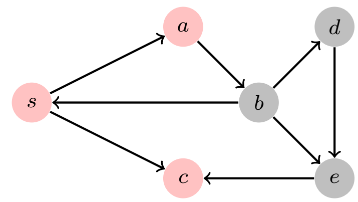
When a vertex u is dequeued, its neighbours are inspected and the operations in the **if** statement cost constant time each. Thus one has an additional cost of $O(|A|)$, since these constant-time operations are carried out for each arc $a \in A$. □

7.3 Shortest Paths

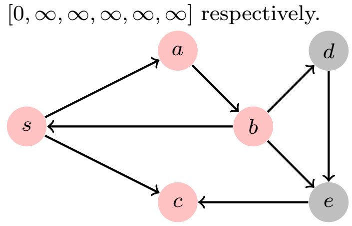
Definition 7.5 (Cycle). A walk in which starting node and end-node agree is called a *cycle*.



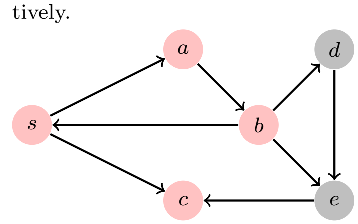
(a) The breadth-first search algorithm starts with the queue $Q = [s]$. The distance labels for $[s, a, b, c, d, e]$ are $[0, \infty, \infty, \infty, \infty, \infty]$ respectively.



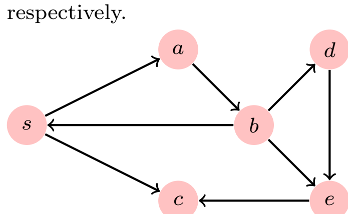
(b) After the first iteration of the **while** loop the queue is $Q = [a, c]$ and the distance labels are $[0, 1, \infty, 1, \infty, \infty]$ respectively.



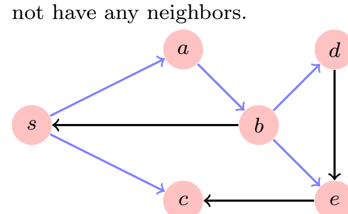
(c) After the second iteration of the **while** loop the queue is $Q = [c, b]$ and the distance labels are $[0, 1, 2, 1, \infty, \infty]$ respectively.



(d) After the third iteration of the **while** loop the queue is $Q = [b]$ and the distance labels are unchanged, since c does not have any neighbors.



(e) After the fourth iteration of the **while** loop the queue is $Q = [d, e]$ and the distance labels are $[0, 1, 2, 1, 3, 3]$ respectively.



(f) After the fifth iteration of the **while** loop the queue is empty $Q = []$ and the distance labels remain unchanged. The blue edges denote the shortest path tree.

Fig. 7.3: An example-run of breadth-first search

Suppose we are given a directed graph $D = (V, A)$ and a length function $c: A \longrightarrow \mathbb{R}$. The *length* of a walk W is defined as

$$c(W) = \sum_{\substack{a \in A \\ a \in W}} c(a).$$

We now study how to determine a shortest path in the weighted directed graph G efficiently, in case of the absence of cycles of negative length (such cycles are called *negative cycles*).

Theorem 7.2. *Suppose that each cycle in D has non-negative length and suppose there exists an $s-t$-walk in D. Then there exists a path connecting s with t which has minimum length among all walks connecting s and t.*

Proof. If there exists an $s-t$-walk, then there exists an $s-t$-path. Since the number of arcs in a path is at most $|V|-1$, there must exist a shortest *path* P connecting s and t. We claim that $c(P) \leqslant c(W)$ for all $s-t$-walks W. Suppose that there exists an $s-t$-walk W with $c(W) < c(P)$. Then let W be such a walk with a minimum number of arcs. Clearly W contains a cycle C. Since the cycle has nonnegative length, then it can be removed from W to obtain a walk whose length is at most $c(W)$ and whose number of arcs is strictly less than $|C|$. □

We use the notation $|W|, |C|, |P|$ to denote the number of arcs in a walk W a cycle C or a path P.

As a conclusion we can note here:

If there do not exist negative cycles in D, and s and t are connected, then there exists a shortest walk traversing at most $|V|-1$ arcs.

The Bellman-Ford algorithm

Let $n = |V|$. We calculate functions $f_0, f_1, \ldots, f_n : V \longrightarrow \mathbb{R} \cup \{\infty\}$ successively by the following rule.

i) $f_0(s) = 0$, $f_0(v) = \infty$ for all $v \neq s$
ii) For $k < n$ if f_k has been found, compute

$$f_{k+1}(v) = \min\{f_k(v), \min_{(u,v)\in A}\{f_k(u) + c(u,v)\}\}$$

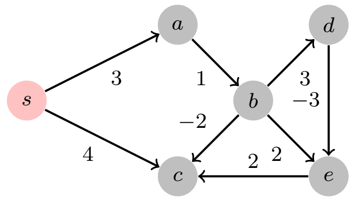
for all $v \in V$.

Theorem 7.3. *For each $k = 0, \ldots, n$ and for each $v \in V$*

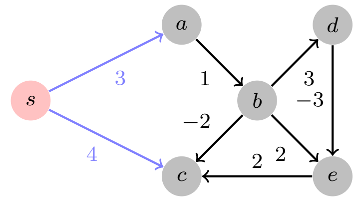
$$f_k(v) = \min\{c(P) \colon P \text{ is an } s-v\text{-walk traversing at most } k \text{ arcs}\}.$$

Corollary 7.1. *If $D = (V, A)$ does not contain negative cycles w.r.t. c, then $f_n(v)$ is equal to the length of a shortest $s-v$-path. The numbers $f_n(v)$ can be computed in time $O(|V| \cdot |A|)$.*

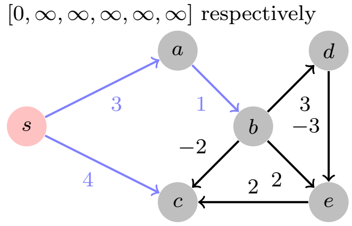
Corollary 7.2. *In time $O(|V|^2|A|)$ one can test whether $D = (V, A)$ has a negative cycle w.r.t. c and eventually return one.*



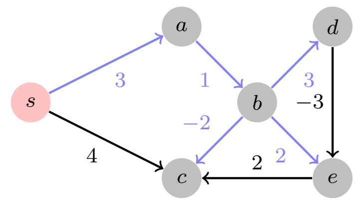
(a) The algorithm is initialized with distance labels for s, a, b, c, d, e being $[0, \infty, \infty, \infty, \infty, \infty]$ respectively



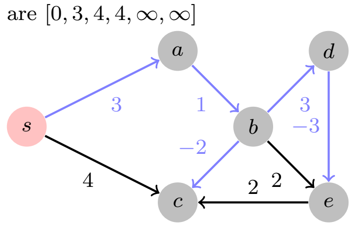
(b) After the first iteration the labels are $[0, 3, \infty, 4, \infty, \infty]$



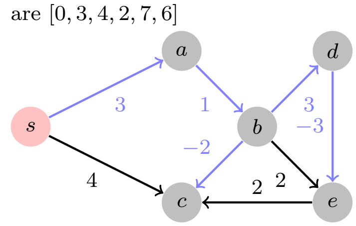
(c) After the second iteration the labels are $[0, 3, 4, 4, \infty, \infty]$



(d) After the third iteration the labels are $[0, 3, 4, 2, 7, 6]$



(e) After the fourth iteration the labels are $[0, 3, 4, 2, 7, 4]$



(f) After the fifth iteration the labels are unchanged. The shortest path distances have been computed.

Fig. 7.4: An example-run of the Bellman-Ford algorithm. The blue edges represent the tree whose paths have the corresponding lengths.

7.4 Maximum $s-t$-flows

We now turn our attention to a linear programming problem which we will solve by direct methods, motivated by the nature of the problem. We often use the following notation. If $f : A \longrightarrow B$ denotes a function and if $U \subseteq A$, then $f(U)$ is defined as $f(U) = \sum_{a \in U} f(a)$.

Definition 7.6 (Network, $s-t$-flow). A network with capacities consists of a directed simple graph $D = (V, A)$ and a *capacity function* $u : A \to \mathbb{R}_{\geqslant 0}$. A function $f : A \to \mathbb{R}_{\geqslant 0}$ is called an *$s-t$-flow*, if

$$\sum_{e \in \delta^{out}(v)} f(e) = \sum_{e \in \delta^{in}(v)} f(e), \quad \text{for all } v \in V - \{s, t\}, \tag{7.1}$$

where $s, t \in V$. The flow is *feasible*, if $f(e) \leqslant u(e)$ for all $e \in A$. The *value* of f is defined as $value(f) = \sum_{e \in \delta^{out}(s)} f(e) - \sum_{e \in \delta^{in}(s)} f(e)$. The *maximum $s-t$-flow problem* is the problem of determining a maximum feasible $s-t$-flow.

Here, for $U \subseteq V$, $\delta^{in}(U)$ denotes the arcs which are entering U and $\delta^{out}(U)$ denotes the arcs which are leaving U. Arc sets of the form $\delta^{out}(U)$ are called a *cut* of D. The *capacity of a cut* $u(\delta^{out}(U))$ is the sum of the capacities of its arcs.

Thus the maximum flow problem is a linear program of the form

$$\max \sum_{e \in \delta^{out}(s)} x(e) - \sum_{e \in \delta^{in}(s)} x(e) \tag{7.2}$$

$$\sum_{e \in \delta^{out}(v)} x(e) = \sum_{e \in \delta^{in}(v)} x(e), \text{ for all } v \in V - \{s, t\} \tag{7.3}$$

$$x(e) \leqslant u(e), \text{ for all } e \in A \tag{7.4}$$

$$x(e) \geqslant 0, \text{ for all } e \in A \tag{7.5}$$

Definition 7.7 (excess function). For any $f : A \to \mathbb{R}$, the excess function is the function $excess_f : 2^V \to \mathbb{R}$ defined by $excess_f(U) = \sum_{e \in \delta^{in}(U)} f(e) - \sum_{e \in \delta^{out}(U)} f(e)$.

Theorem 7.4. *Let $D = (V, A)$ be a digraph, let $f : A \to \mathbb{R}$ and let $U \subseteq V$, then*

$$excess_f(U) = \sum_{v \in U} excess_f(v). \tag{7.6}$$

Proof. An arc which has both endpoints in U is counted twice with different parities on the right, and thus cancels out. An arc which has its tail in U is subtracted once on the right and once on the left. An arc which has its head in U is added once on the right and once on the left. $\square$

A cut $\delta^{out}(U)$ with $s \in U$ and $t \notin U$ is called an $s - t$-cut.

Theorem 7.5 (Weak duality). *Let f be a feasible $s-t$-flow and let $\delta^{out}(U)$ be an $s - t$-cut, then $value(f) \leqslant u(\delta^{out}(U))$.*

Proof. $value(f) = -excess_f(s) = -excess_f(U) = f(\delta^{out}(U)) - f(\delta^{in}(U)) \leqslant f(\delta^{out}(U)) \leqslant u(\delta^{out}(U))$. $\square$

For an arc $a = (u, v) \in A$ the arc a^{-1} denotes the arc (v, u).

Definition 7.8 (Residual graph). Let $f : A \to \mathbb{R}$, and $u : A \to \mathbb{R}$ where $0 \leqslant f \leqslant u$. Consider the sets of arcs

$$A_f = \{a \mid a \in A, \ f(a) < u(a)\} \cup \{a^{-1} \mid a \in A, \ f(a) > 0\}. \tag{7.7}$$

The digraph $D(f) = (V, A_f)$ is called the *residual graph* of f (for capacities u).

Corollary 7.3. *Let f be a feasible $s-t$-flow and suppose that $D(f)$ has no path from s to t, then f has maximum value.*

Proof. Let U be the set of nodes which are reachable in $D(f)$ from s. Clearly $\delta^{out}(U)$ is an $s-t$-cut. Now $value(f) = f(\delta^{out}(U)) - f(\delta^{in}(U)$. Each arc leaving U is not an arc of $D(f)$ and thus $f(\delta^{out}(U)) = u(\delta^{out}(U))$. Each arc entering U does not carry any flow and thus $f(\delta^{in}(U) = 0$. It follows that $value(f) = u(\delta^{out}(U))$ and f is optimal by Theorem 7.5. □

Definition 7.9 (undirected walk). An *undirected walk* is a sequence of the form $P = (v_0, a_1, v_1, \ldots, v_{m-1}, a_m, v_m)$, where $a_i \in A$ for $i = 1, \ldots, m$ and $a_i = (v_{i-1}, v_i)$ or $a_i = (v_i, v_{i-1})$. If the nodes $v_0, \ldots, v_m$ are all different, then P is an *undirected path.*

Any directed path P in $D(f)$ yields an undirected path in D. Define for such a path P the vector $\chi^P \in \{0, \pm 1\}^A$ as

$$\chi^P(a) = \begin{cases} 1 & \text{if } P \text{ traverses } a, \\ -1 & \text{if } P \text{ traverses } a^{-1}, \\ 0 & \text{if } P \text{ traverses neither } a \text{ or } a^{-1}. \end{cases} \tag{7.8}$$

Theorem 7.6 (max-flow min-cut theorem, strong duality). *The maximum value of a feasible $s-t$-flow is equal to the minimum capacity of an $s-t$ cut.*

Proof. Let f be a maximum $s-t$-flow. Consider the residual graph $D(f)$. If this residual graph contains an $s-t$-path P, then we can route flow along this path. More precisely, there exists an $\epsilon > 0$ such that $f + \epsilon\, \chi^P$ is feasible. We have $value(f + \epsilon\, \chi^P) = value(f) + \epsilon$. This contradicts the maximality of f thus there exists no $s-t$-path in $D(f)$.

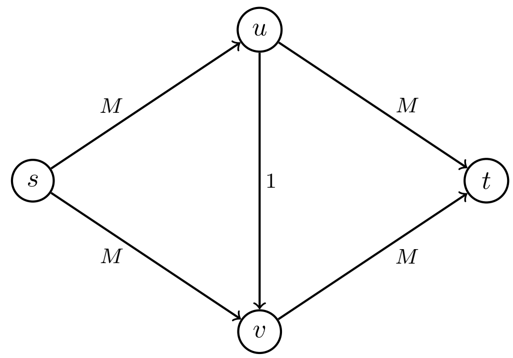
Let U be the nodes reachable from s in $D(f)$. Then $value(f) = u(\delta^{out}(U))$ and $\delta^{out}(U)$ is an $s-t$-cut of minimum capacity by the weak duality theorem.

This suggests the algorithm of Ford and Fulkerson to find a maximum flow. Start with $f = 0$. Next iteratively apply the following *flow augmentation algorithm.*

Let P be a directed $s-t$-path in $D(f)$. Set $f \leftarrow f + \epsilon\chi^P$, where ϵ is as large as possible to maintain $0 \leqslant f \leqslant u$.

Exercise 7.1. Define a *residual capacity* for $D(f)$. Then determine the maximum ϵ such that $0 \leqslant f \leqslant u$.

Theorem 7.7. *If all capacities are rational, this algorithm terminates.*



The example above shows that, if the augmenting paths are chosen in a disadvantageous way, then the Ford-Fulkerson algorithm may take $\Omega(M)$ iterations, where M is the largest capacity in the network. This happens if all augmenting paths use the arc uv or vu respectively in the residual network.

Corollary 7.4 (integrity theorem). *If $u(a) \in \mathbb{N}$ for each $a \in A$, then there exists an integer maximum flow ($f(a) \in \mathbb{N}$ for all $a \in A$).*

Proof. This follows from the fact that the residual capacities remain integral and thus the augmented flow is always integral. $\square$

Theorem 7.8. *If we choose in each iteration a shortest $s - t$-path in $D(f)$ as a flow-augmenting path, the number of iterations is at most $|V| \cdot |A|$.*

Definition 7.10. Let $D = (V, A)$ be a digraph, $s, t \in V$ and let $\mu(D)$ denote the length of a shortest path from s to t. Let $\alpha(D)$ denote the set of arcs contained in at least one shortest $s - t$ path.

Theorem 7.9. *Let $D = (V, A)$ be a digraph and $s, t \in V$. Define $D' = (V, A \cup \alpha(D)^{-1})$. Then $\mu(D) = \mu(D')$ and $\alpha(D) = \alpha(D')$.*

Proof. It suffices to show that $\mu(D)$ and $\alpha(D)$ are invariant if we add a^{-1} to D for one arc $a \in \alpha(D)$. Suppose not, then there is a directed $s - t$-path P_1 traversing a^{-1} of length at most $\mu(D)$. As $a \in \alpha(D)$ there is a path P_2 traversing a of length $\mu(D)$. If we follow P_2 until the tail of a is reached and from thereon follow P_1, we obtain another $s - t$ path P_3 in D. Similarly if we follow P_1 until the head of a is reached and then follow P_2, we obtain a fourth $s - t$ path P_4 in D. However P_3 or P_4 has length less than $\mu(D)$. This is a contradiction. $\square$

Proof (of Theorem 7.8). Let us augment flow f along a shortest $s - t$-path P in $D(f)$ obtaining flow f'. The residual graph $D_{f'}$ is a subgraph of $D' = (V, A_f \cup \alpha(D(f))^{-1})$. Hence $\mu(D_{f'}) \geqslant \mu(D') = \mu(D(f))$. If $\mu(D_{f'}) = \mu(D(f))$, then $\alpha(D_{f'}) \subseteq \alpha(D') = \alpha(D(f))$. At least one arc of P does not belong to $D_{f'}$, (the arc of minimum residual capacity!) thus the inclusion is strict. Since $\mu(D(f))$ increases at most $|V|$ times and, as long as $\mu(D(f))$ does not change, $|\alpha(D(f))|$ decreases at most $2\,|A|$ times, we have the theorem. $\square$

In the following let $m = |A|$ and $n = |V|$.

Corollary 7.5. *A maximum flow can be found in time $O(n\,m^2)$.*

7.5 Minimum cost network flows, MCNFP

In contrast to the maximum $s-t$-flow problem, the goal here is to route a flow, which comes from several sources and sinks through a network with capacities and *costs* in such a way, that the total cost is minimized.

Example 7.2. Suppose you are given a directed graph width arc weights $D = (V, A)$, $c : A \to \mathbb{R}_{\geqslant 0}$ and your task is to compute a shortest path from a particular node s to all other nodes in the graph and assume that such paths exist. Then one can model this as a MCNFP by sending a flow of value $|V| - 1$ into the source node and by letting a flow of value 1 leave each node. The costs on the arcs are defined by c. The arcs have infinite capacities. We will see later, that this minimum cost network flow problem has an integral solution which corresponds to the shortest paths from s to all other nodes.

Here is a formal definition of a minimum cost network flow problem. In this notation, vertices are indexed with the letters i, j, k and arcs are denoted by their tail and head respectively, for example (i, j) denotes the arc from i to j.

A *network* is now a directed graph $D = (V, A)$ together with a capacity function $u : A \to \mathbb{Q}_{\geqslant 0}$, a cost function $c : A \to \mathbb{Q}$ and an external flow $b : V \to \mathbb{Q}$. The value of $b(i)$ denotes the amount of flow which comes from the exterior. If $b(i) > 0$, then there is flow from the outside, entering the network through node i. If $b(i) < 0$, there is flow which leaves the network through i.

In the following we often use the notation $f(i, j)$ for the flow-value on the arc (i, j) (instead of $f((i, j))$). Similarly we write $c(i, j)$ and $u(i, j)$.

A *feasible flow* is a function $f : A \to \mathbb{Q}_{\geqslant 0}$ which satisfies the following constraints.

$$\begin{array}{ll} \sum_{e\in\delta^{out}(i)} f(e) - \sum_{j\in\delta^{in}(i)} f(e) = b_i & \text{for all } i \in V, \\ 0 \leqslant f(e) \leqslant u(e) & \text{for all } e \in A. \end{array}$$

The goal is to find a feasible flow with minimum cost:

$$\begin{array}{lll} \text{minimize} & \sum_{e\in A} c(e) f(e) & \\ \text{subject to} & \sum_{e\in\delta^{out}(i)} f(e) - \sum_{e\in\delta^{in}(i)} f(e) = b(i) & \text{for all } i \in V, \\ & 0 \leqslant f(e) \leqslant u(e) & \text{for all } (e) \in A \end{array}$$

Example 7.3. Imagine you are a pilot and fly a passenger airplane in hops from airport 1 to airport 2 to airport 3 and so on, until airport n. At airport

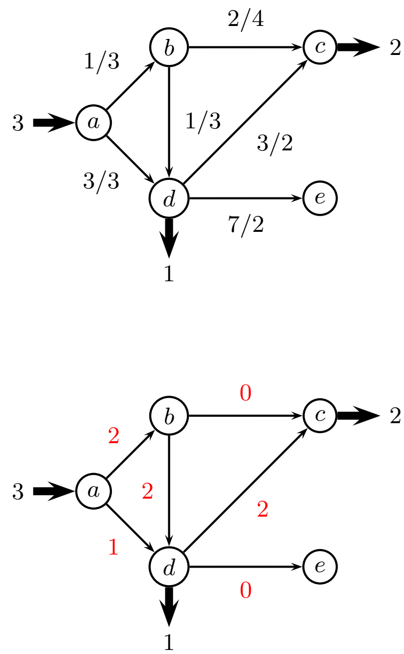


Fig. 7.5: A Network with in/out-flow, costs and capacities and a feasible flow of cost 13.

i there are b_{ij} passengers that want to travel to airport j, where $j > i$. You may decide how many of the b_{ij} passengers you will take on board. Each of the passengers will pay c_{ij} dollars for the trip. The airplane can accommodate p people.

You are a greedy pilot and think of a plan to pick up and deliver passengers on your hop from 1 to n which maximizes your revenue.

Finding this plan can be modelled as a minimum cost network flow problem. Your network has nodes $1, \ldots, n$ and arcs $(i, i+1), i = 1, \ldots, n-1$ with capacities p and without costs. These nodes do not have in/out-flow from the outside. You furthermore have nodes $i \to j$ for $i < j$ and $i, j \in \{1, \ldots, n\}$ which are excess nodes with in-flow b_{ij} from the outside. Each node $i \to j$ is connected to i and to j with a directed arc. The capacities on these arcs are infinite. The cost of the arc $(i \to j, i)$ is $-c_{ij}$. The cost of the arc $(i \to j, j)$ is zero. The outflow on the node j is the total number of passengers that want to fly to node j. An integral optimal flow to this problem is an optimal plan for you.

Throughout this chapter we make the following assumptions.

1. All data (cost, supply, demand and capacity) are integral.
2. The network contains an incapacitated directed path between every pair of nodes.
3. The supplies/demands at the nodes satisfy the condition $\sum_{i \in V} b(i) = 0$ and the MCNFP has a feasible solution.
4. All arc costs are nonnegative.
5. The graph does not contain a pair of reverse arcs.

Exercise 7.2. Show how to transform a MCNFP on a digraph with pairs of reverse arcs into a MCNFP on a digraph with no pairs of reverse arcs. The number of arcs and nodes should asymptotically remain the same.

An *arc-flow* of D is a flow vector, that satisfies the nonnegativity and capacity constraints.

$$\begin{aligned} \sum_{e \in \delta^{in}(i)} f(e) - \sum_{e \in \delta^{out}(i)} f(e) = e(i) & \quad \text{for all } i \in V, \\ 0 \leqslant f(e) \leqslant u(e) & \quad \text{for all } e \in A. \end{aligned}$$

- If $e(i) > 0$, then i is an *excess node* (more inflow than outflow).
- If $e(i) < 0$, then i is a *deficit node* (more outflow than inflow).
- If $e(i) = 0$ then i is called *balanced*.

Exercise 7.3. Prove that $\sum_{i \in V} e(i) = 0$ holds and thus that a feasible flow only exists if the sum of the $b(i)$ is equal to zero.

Let $\mathscr{P}$ be the collection of directed paths of D and let $\mathcal{C}$ be the collection of directed cycles of D. A path-flow is a function $\beta : \mathscr{P} \cup \mathcal{C} \to \mathbb{R}_{\geqslant 0}$ which assigns flow values to paths and cycles.

For $(i, j) \in A$ and $P \in \mathscr{P}$ let $\delta_{(i,j)}(P)$ be 1 if $(i, j) \in P$ and 0 otherwise. For $C \in \mathcal{C}$ let $\delta_{(i,j)}(C)$ be 1 if $(i, j) \in C$ and 0 otherwise.

A path-flow β determines a unique arc-flow

$$f(i, j) = \sum_{P \in \mathscr{P}} \delta_{(i,j)}(P)\beta(P) + \sum_{C \in \mathcal{C}} \delta_{(i,j)}(C)\beta(C).$$

Theorem 7.10. *Every path and cycle flow has a unique representation as a nonnegative arc-flow. Conversely, every nonnegative arc flow f can be represented as a path and cycle flow with the following properties:*

1. *Every directed path with positive flow connects a deficit node with an excess node.*
2. *At most $n + m$ paths and cycles have nonzero flow and at most m cycles have nonzero flow.*

If the arc flow f is integral, then so are the path and cycle flows into which it decomposes.

Proof. "$\Rightarrow$" See discussion above.
"$\Leftarrow$"

Let f be an arc flow. Suppose i_0 is a deficit node. Then there exists an incident arc (i_0, i_1) which carries a positive flow. If i_1 is an excess node, we have found a path from deficit to excess node. Otherwise, the flow balance constraint at i_1 implies that there exists an arc (i_1, i_2) with positive flow. Repeating this procedure, we finally must arrive at an excess node or revisit a node. This means that we either have constructed a directed path P from deficit node to excess node or a directed cycle C, both involving only arcs with strictly positive flow.

In the first case, let $P = i_0, \dots, i_k$ be the directed path from deficit node i_0 to excess node i_k. We set $\beta(P) = \min\{-e_{i_0}, e_{i_k}, \min\{f(i,j) \mid (i,j) \in P\}\}$ and $f(i,j) = f(i,j) - \beta(P)$, $(i,j) \in P$. In the second case, set $\beta(C) = \min\{f(i,j) \mid (i,j) \in C$ and $f(i,j) = f(i,j) - \beta(C)$, $(i,j) \in C$. Repeat this procedure until all node imbalances are zero.

Now find an arc with positive flow and construct a cycle C by following only positive arcs from there. Set $\beta(C) = \min\{f(i,j) \mid (i,j) \in C\}$ and $f(i,j) = f(i,j) - \beta(C)$, $(i,j) \in C\}$. Repeat this process until there are no positive flow-arcs left.

Each time a path or a cycle is identified, the excess/deficit of some node is set to zero or some arc is set to zero. This implies that we decompose into at most $n + m$ paths and cycles. Since cycle detection sets an arc to zero we have at most m cycles. $\square$

An arc flow f with $e(i) = 0$ for each $i \in V$ is called a *circulation*.

Corollary 7.6. *A circulation can be decomposed into at most m cycle flows.*

Let $D = (V, A)$ be a network with capacities $u(i,j)$, $(i,j) \in A$ and costs $c(i,j)$, $(i,j) \in A$ and let f be a feasible flow of the network. The *residual network* $D(f)$ is defined as follows.

- We replace each arc $(i,j) \in A$ with two arcs (i,j) and (j,i).
- The arc (i,j) has cost $c(i,j)$ and *residual capacity* $r(i,j) = u(i,j) - f(i,j)$.
- The arc (j,i) has cost $-c(i,j)$ and residual capacity $r(j,i) = f(i,j)$.
- Delete all arcs which do not have strictly positive residual capacity.

A directed cycle in $D(f)$ is called an *augmenting cycle* of f.

Lemma 7.4. *Suppose that f and f° are feasible flows, then $f - f^\circ$ is a circulation in $D(f^\circ)$. Here $f - f^\circ$ is the flow*

$$(f - f^\circ)(e) = \begin{cases} \max\{0, f(e) - f^\circ(e)\}, & \text{if } e \in A(D) \\ \max\{0, f^\circ(e) - f(e)\}, & \text{if } e^{-1} \in A(D) \\ 0, & \text{otherwise.} \end{cases}$$

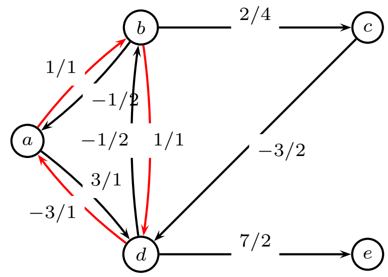


Fig. 7.6: The residual network of the flow in Figure 7.5 and a negative cycle marked by the red edges.

Proof. It is very easy to see that the flow $f - f^\circ$ satisfies the capacity constraints. One also has for each $v \in V$

$$\sum e \in \delta^{out}(v)(f(e) - f^\circ(e)) - \sum_{e \in \delta^{in}(v)} (f(e) - f^\circ(e)) = 0.$$

If a term $(f(e) - f^\circ(e))$ is negative, it is replaced by its absolute value and charged as flow on the arc e^{-1} in $D(f^\circ)$ which leaves its contribution to the sum above invariant. □

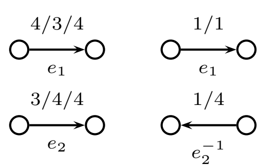


Fig. 7.7: Two arcs $e_1, e_2 \in A$ labeled with $f(e)/f^\circ(e)/u(e)$ and the corresponding flow on these arcs (or their reverse) in $D(f^\circ)$. Arcs in $D(f^\circ)$ are labeled with flow and capacity values respectively.

Theorem 7.11 (Augmenting Cycle Theorem). *Let f and f° be any two feasible flows of a network flow problem. Then f equals f° plus the flow of at most m directed cycles in $D(f^\circ)$. Furthermore the cost of f equals the cost of f° plus the cost of flow on these augmenting cycles.*

Proof. This can be seen by applying flow decomposition on the flow $f - f^\circ$ in $D(f^\circ)$. □

Theorem 7.12 (Negative Cycle Optimality Conditions). *A feasible flow f^* is an optimal solution of the minimum cost network flow problem, if and only if it satisfies the negative cycle optimality conditions: The residual network $D(f^*)$ contains no directed cycle of negative cost.*

Proof. "$\Rightarrow$" Suppose that f is a feasible flow and that $D(f)$ contains a negative directed cycle. Then f cannot be optimal, since we can augment positive flow along the corresponding cycle in the network. Therefore, if f^* is an optimal flow, then $D(f^*)$ cannot contain a negative directed cycle.

"$\Leftarrow$" Suppose now that f^* is a feasible flow and suppose that $D(f^*)$ does not contain a negative cycle. Let f° be an optimal flow with $f^\circ \neq f^*$. The vector $f^\circ - f^*$ is a circulation in $D(f^\circ)$ with non-positive cost $c^T(f^\circ - f^*) \leqslant 0$. It follows from Theorem 7.11 that the cost of f° equals the cost of f^* plus the cost of directed cycles in the residual network $D(f^*)$. The cost of these cycles is nonnegative, and therefore $c(f^\circ) \geqslant c(f^*)$ which implies that f^* is optimal. □

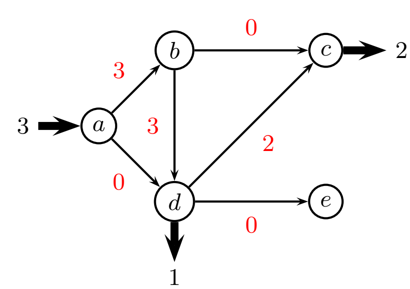


Fig. 7.8: The result of augmenting a flow of one along the negative cycle in Figure 7.6. This flow has cost 12 but is not optimal, since the residual network still contains a negative cycle.

Algorithm 7.1 (Cycle Canceling Algorithm).

1. establish a feasible flow f in the network
2. `WHILE` $D(f)$ contains a negative cycle
   a. detect a negative cycle C in $D(f)$
   b. $\delta = \min\{r(i,j) \mid (i,j) \in C\}$
   c. augment δ units of flow along the cycle C
   d. update $D(f)$
3. `RETURN` f

Theorem 7.13. *The cycle canceling algorithm terminates after a finite number of steps if the MCNFP has an optimal solution.*

Proof. The cycle canceling algorithm reduces the cost in each iteration. We have assumed that the input data is integral. Thus the cost decreases by at least one unit each iteration. Therefore the number of iterations is finite. □

Corollary 7.7. *If the capacities are integral and if the MCNFP has a optimal flow, then it has an optimal flow with integer values only.*

Let $\pi : V \to \mathbb{R}$ be a function (*node potential*). The *reduced cost* of an arc (i,j) w.r.t. π is $c_\pi((i,j)) = c((i,j)) + \pi(i) - \pi(j)$. The potential π is called *feasible* if $c_\pi((i,j)) \geqslant 0$ for all arcs $(i,j) \in A$.

Lemma 7.5. *Let $D = (V,A)$ be a digraph with arc weights $c : A \to \mathbb{R}$. Then D does not have a negative cycle if and only if there exists a feasible node potential π of D with arc weights c.*

Proof. Consider a directed path $P = i_0, i_1, \ldots, i_k$. The cost of this path is

$$c(P) = \sum_{j=1}^{k} c((i_{j-1}, i_j)).$$

The reduced cost of this path is equal to

$$c_\pi(P) = \sum_{j=1}^{k} c((i_{j-1}, i_j)) + \pi(i_0) - \pi(i_k).$$

If P is a cycle, then i_0 and i_k are equal, which means that its cost and reduced cost coincide. Thus, if there exists a feasible node potential, then there does not exist a negative cycle.

On the other hand, suppose that D, c does not contain a negative cycle. Add a vertex s to D and the arcs (s,i) for all $i \in V$. The weights (costs) of all these new arcs is 0. Notice that in this way, no new cycles are created, thus still there does not exist a negative cycle. This means we can compute the shortest paths from s to all other nodes $i \in V$. Let π be the function which assigns these shortest paths lengths. Clearly $c_\pi((i,j)) = \pi(i) - \pi(j) + c((i,j)) \geqslant 0$, since the shortest-path length to j is at most the shortest-path length to $i + c((i,j))$. □

This means that we have again a nice way to prove that a flow is optimal. Simply equip the residual network with a feasible node potential.

Corollary 7.8 (Reduced Cost Optimality Condition). *A feasible flow f^* is optimal if and only if there exists a node potential π such that the reduced costs $c_\pi(i,j)$ of each arch (i,j) of $D(f)$ are nonnegative.*

The cycle canceling algorithm is only pseudopolynomial. If we could always chose a minimum cycle (cycle with best improvement) as an augmenting cycle, we would have a polynomial number of iterations. Finding minimum cycles is NP-hard. Instead we augment along *minimum mean cycles*. One can find minimum mean cycles in polynomial time.

The *mean cost* of a cycle $C \in \mathcal{C}$ is the cost of C divided by the number of arcs in C:

$$\Big(\sum_{(i,j)\in C} c(i,j)\Big)/|C|.$$

Algorithm 7.2 (Minimum Mean Cycle Canceling, MMCC).

1. establish a feasible flow f in the network
2. `WHILE` $D(f)$ contains a negative cycle
   a. detect a minimum mean cycle C in $D(f)$
   b. $\delta = \min\{r(i,j) \mid (i,j) \in C\}$
   c. augment δ units of flow along the cycle C
   d. update $D(f)$
3. `RETURN` f

We now analyze the MMCC-algorithm. Let $\mu(f)$ denote the minimum mean-weight of a cycle in $D(f)$.

Lemma 7.6 (See Korte & Vygen [8]). *Let $f_1, f_2, \ldots$ be a sequence of feasible flows such that f_{i+1} results from f_i by augmenting flow along C_i, where C_i is a minimum mean cycle of $D(f_i)$, then*

1. *$\mu(f_k) \leqslant \mu(f_{k+1})$ for all k.*
2. *$\mu(f_k) \leqslant \frac{n}{n-1}\mu(f_l)$, where $k < l$ and $C_k \cup C_l$ contains a pair of reversed arcs.*

Proof. 1): Suppose f_k and f_{k+1} are two subsequent flows in this sequence. Consider the multi-graph H which results from C_k and C_{k+1} by deleting pairs of opposing arcs. The arcs of H are a subset of the arcs of $D(f_k)$, since an arc of C_{k+1} which is not in $D(f_k)$ must be a reverse arc of C_k.

Each node in H has even degree. Thus H can be decomposed into cycles, each of mean weight at least $\mu(f_k)$. Thus we have $c(A(H)) \geqslant \mu(f_k)|A(H)|$.

Since the total weight of each reverse pair of arcs is zero we have

$$c(A(H)) = c(C_k) + c(C_{k+1}) = \mu(f_k)|C_k| + \mu(f_{k+1})|C_{k+1}|.$$

Since $|A(H)| \leqslant |C_k| + |C_{k+1}|$ we conclude

$$\begin{aligned}\mu(f_k)(|C_k| + |C_{k+1}|) &\leqslant \mu(f_k)|A(H)|\\ &\leqslant c(A(H))\\ &= \mu(f_k)|C_k| + \mu(f_{k+1})|C_{k+1}|.\end{aligned}$$

Thus $\mu(f_k) \leqslant \mu(f_{k+1})$.

2): By the first part of the theorem, it is enough to prove the statement for k, l such that $C_i \cup C_l$ does not contain a pair of reverse arcs for each i, $k < i < l$.

Again, consider the graph H resulting from C_k and C_l by deleting pairs of opposing arcs. H is a subgraph of $D(f_k)$, since any arc of C_l which does not belong to $D(f_k)$ must be a reverse arc of $C_k, C_{k+1}, \ldots, C_{l-1}$. But only C_k contains a reverse arc of C_l. So as above we have

$$c(A(H)) = c(C_k) + c(C_l) = \mu(f_k)|C_k| + \mu(f_l)|C_{k+1}|.$$

Since $|A(H)| \leqslant |C_k| + |C_l| - 2$ we have $|A(H)| \leqslant \frac{n-1}{n}(|C_k| + |C_l|)$. Thus we get

$$\begin{aligned}\mu(f_k)\frac{n-1}{n}(|C_k| + |C_l|) &\leqslant \mu(f_k)|A(H)|\\ &\leqslant c(A(H))\\ &= \mu(f_k)|C_k| + \mu(f_l)|C_l|\\ &\leqslant \mu(f_l)(|C_k| + |C_l|)\end{aligned}$$

This implies that $\mu(f_k) \leqslant \frac{n}{n-1}\mu(f_l)$. □

Corollary 7.9. *During the execution of the MMCC-algorithm, $|\mu(f)|$ decreases by a factor of $1/2$ every $n \cdot m$ iterations.*

Proof. Let $C_1, C_2, \ldots$ be the sequence of augmenting cycles. Every m-th iteration, there must be an arc of the cycle, which is reverse to one of the succeeding $m-1$ cycles, because every iteration, one arc of the residual network will be deleted. Thus after $n\,m$ iterations, the absolute value of μ has dropped by $\left(\frac{n-1}{n}\right)^n \leqslant e^{-1} \leqslant 1/2$. □

Corollary 7.10. *If all data are integral, then the MMCC-algorithm runs in polynomial time.*

Proof.
- A lower bound on μ is the smallest cost c_{min}
- $|\mu|$ drops by $1/2$ every $m\,n$ iterations.
- After $mn \log n|c_{min}|$ iterations, absolute value of minimum mean weight cycle drops below $1/n$, thus is zero.

- **We need to prove that a minimum mean cycle can be found in polynomial time**

□

This is a so-called *weakly polynomial* bound, since the binary encoding length of the numbers in the input (here the costs) influences the running time. We now prove that the MMCC-algorithm is *strongly polynomial.*

Theorem 7.14 (See Korte & Vygen [8]). *The MMCC-algorithm requires $O(m^2\, n \log n)$ iterations (mean weight cycle cancellations).*

Proof. One shows that every $m\, n(\lceil \log n \rceil + 1)$ iterations, at least one arc is *fixed*, which means that the flow through this arc does not change anymore.

Let f_1 be some flow at some iteration and let f_2 be the flow $m\, n(\lceil \log n \rceil + 1)$ iterations later. It follows from Corollary 7.9 that

$$\mu(f_1) \leqslant 2\, n\, \mu(f_2) \tag{7.9}$$

holds.

Define the costs $c'(e) = c(e) - \mu(f_2)$ for the residual network $D(f_2)$. There exists no negative cycle in $D(f_2)$ w.r.t. this cost c'. (A cycle C has weight $c'(C) = \sum_{e \in C} c(e) - |C|\mu(f_2)$ and thus $c'(C)/|C| = \sum_{e \in C} c(e)/|C| - \mu(f_2) \geqslant 0$). By Lemma 7.5 there exists a feasible node potential π for these weights. One has $0 \leqslant c'_\pi(e) = c_\pi(e) - \mu(f_2)$ and thus

$$c_\pi(e) \geqslant \mu(f_2), \text{ for all } e \in A(D(f_2)). \tag{7.10}$$

Let C be a minimum mean cycle of $D(f_1)$. One has

$$c_\pi(C) = c(C) = \mu(f_1)\, |C| \leqslant 2\, n\, \mu(f_2)|C|. \tag{7.11}$$

It follows that there exists an arc e_0 of C such that

$$c_\pi(e_0) \leqslant 2\, n\, \mu(f_2) \tag{7.12}$$

holds. The inequalities (7.10) imply that $e_0 \notin A(D(f_2))$

We now make the following claim:

Let f' be a feasible flow such that $e_0 \in D(f')$, then $\mu(f') \leqslant \mu(f_2)$.

If we have shown this claim, then it follows from Lemma 7.6 that e_0 cannot be anymore in the residual network of a flow after f_2. Thus the flow along the arc e_0 (or e_0^{-1}) is fixed.

Let f' be a flow such that $e_0 \in A(D(f'))$. Recall that $f' - f_2$ is a circulation in $D(f_2)$ where $e_0 \notin D(f_2)$, $e_0^{-1} \in D(f_2)$ and this circulation sends flow over e_0^{-1}. This circulation can be decomposed into cycles and one of these cycles C contains e_0^{-1}. One has $c_\pi(e_0^{-1}) = -c_\pi(e_0) \geqslant -2\, n\, \mu(f_2)$ (eq. (7.12)). Using (7.10) one obtains

$$
\begin{aligned}
c(C) &= \sum_{e \in C} c_\pi(e) && (7.13)\\
&\geqslant -2\,n\,\mu(f_2) + (n-1)\mu(f_2) && (7.14)\\
&= -(n+1)\,\mu(f_2) && (7.15)\\
&> -n\,\mu(f_2). && (7.16)
\end{aligned}
$$

The reverse of C is an augmenting cycle for f' with total weight at most $n\,\mu(f_2)$ and thus with mean weight at most $\mu(f_2)$. Thus $\mu(f') \leqslant \mu(f_2)$. □

7.6 Computing a minimum cost-to-profit ratio cycle

Given a digraph $D = (V, A)$ with costs $c : A \to \mathbb{Z}$ and profit $p : A \to \mathbb{N}_{>0}$, the task is to compute a cycle $C \in \mathcal{C}$ with minimum ratio

$$\frac{c(C)}{p(C)}. \tag{7.17}$$

Notice that this is the largest number $\beta \in \mathbb{Q}$ which satisfies

$$\beta \leqslant \frac{c(C)}{p(C)}, \text{ for all } C \in \mathcal{C}. \tag{7.18}$$

By rewriting this inequality, we understand this to be the largest number $\beta \in \mathbb{Q}$ such that

$$c(C) - \beta\, p(C) \geqslant 0 \text{ for all } C \in \mathcal{C}. \tag{7.19}$$

In other words, we search the largest number $\beta \in \mathbb{Q}$ such that the digraph $D = (V, A)$ with costs $c_\beta : A \to \mathbb{Q}$, where $c_\beta(e) = c(e) - \beta\, p(e)$.

We need a routine to check whether D has a negative cycle for a given weight function c. For this we assume w.l.o.g. that each vertex is reachable from the vertex s, if necessary by introducing a new vertex s from which there is an arc with cost and profit 0 to all other nodes. The minimum cost-to-profit ration cycle w.r.t. this new graph is then the minimum cost to profit ratio cycle w.r.t. the original graph, since s is not a vertex of any cycle.

Recall the following single-source shortest-path algorithm of Bellman-Ford which we now apply with weights c_β:

Let $n = |V|$. We calculate functions $f_0, f_1, \ldots, f_n : V \longrightarrow \mathbb{R} \cup \{\infty\}$ successively by the following rule.

i) $f_0(s) = 0$, $f_0(v) = \infty$ for all $v \neq s$
ii) For $k < n$ if f_k has been found, compute

$$f_{k+1}(v) = \min\{f_k(v), \min_{(u,v) \in A}\{f_k(u) + c_\beta(u, v)\}$$

for all $v \in V$.

There exists a negative cycle w.r.t. c_β if and only if $f_n(v) < f_k(v)$ for some $v \in V$ and $1 \leqslant k < n$. Thus we can test in $O(m \cdot n)$ steps whether D, c_β contains a negative cycle.

We now apply the following idea to search for the correct value of β. We keep an interval $I = [L, U]$ with the invariant that the value β that we are searching lies in this interval I. As starting values, we can chose $L = c_{min}$ and $U = c_{max}$, where c_{min} and c_{max} are the smallest and largest cost respectively. In one iteration we compute $M = (L+U)/2$. We then check whether D, together with c_M contains a negative cycle. If yes, we know that β is at least M and we set $L \leftarrow M$. If not, then β is at most M and we update the upper bound $U \leftarrow M$.

When can we stop this procedure? We can stop it, if we can assure that only one valid cost-to-profit ratio cycle lies in $[L, U]$. Suppose that C_1 and C_2 have different cost-to-profit ratios. Then

$$|c(C_1)/p(C_1) - c(C_2)/p(C_2)| = \left| \frac{c(C_1)\,p(C_2) - c(C_2)p(C_1)}{(p(C_1)\,p(C_2))} \right| \tag{7.20}$$

$$\geqslant 1/(n^2 p_{max}^2). \tag{7.21}$$

Thus we can stop our process, if $U - L < 1/(n^2 p_{max}^2)$, since we know then that there can be only one cycle $c \in \mathcal{C}$ with $c(C)/p(C) \in [L, U]$.

Suppose that $[L, U]$ is the final interval. We know then that

$$L \leqslant c(C)/p(C) \text{ for all } C \in \mathcal{C}$$

and

$$U > c(C)/p(C) \text{ holds for some } C \in \mathcal{C}.$$

Let C be a minimum weight cycle w.r.t. the arc costs c_L. Clearly $U > c(C)/p(C) \geqslant L$ holds and thus C is the minimum cost-to-profit cycle we have been looking for.

Let us analyze the number of required iterations. We need to halve the starting interval-length $2\,c$, where c is the largest absolute value of a cost, until the length is at most $1/(n^2 p_{max}^2)$. We search the minimal $i \in \mathbb{N}$ such that

$$(1/2)^i c \leqslant 1/(n^2 p_{max}^2). \tag{7.22}$$

This shows us that we need $O(\log(c\,p_{max}^2 n^2))$ iterations which is $O(\log n \log K)$, where K is the largest absolute value of a cost or a profit.

Theorem 7.15 (Lawler [9]). *Let D be a digraph with costs $c : A \to \mathbb{Z}$ and profit $p : A \to \mathbb{N}_{>0}$ an let $K \in \mathbb{N}$ such that $|c(e)| + |p(e)| \leqslant K$ for all $e \in \mathbb{N}$. A minimum cost-to-profit ratio cycle of G can be computed in time $O(m\,n \log n \log K)$.*

But we knew a weakly polynomial algorithm for MCNFP from the exercises. So you surely ask: Can we do better for minimum cost-to-profit cycle computation? The answer is “Yes”!

7.6.1 Parametric search

Let us first roughly describe the idea on how to obtain a strongly polynomial algorithm, see [13]. The Bellman-Ford algorithm tells us whether our current β is too large or too small, depending on whether D with weights c_β contains a negative cycle or not. Recall that the B-F algorithm computes labels $f_i(v)$ for $v \in V$ and $1 \leqslant i \leqslant n$. If these labels are computed with costs c_β, then they are *piecewise linear* functions in β and we denote them by $f_i(v)[\beta]$.

Denote the optimal β that we look for by β^* and suppose that we know an interval I with such that $\beta^* \in I$ and each function $f_i(v)[\beta]$ is linear if it is restricted to this domain I. Then we can determine β^* as follows.

Let $I = [L, U]$ be the interval and remember that we are searching for the largest value of $\beta \in I$ such that $f_n(v)[\beta] = f_{n-1}(v)[\beta]$ holds for each $v \in V$. Clearly this holds for $\beta = L$. Thus we only need to check whether $\beta = U$ by computing the values $f_n(v)[U]$ and $f_{n-1}(v)[U]$ for each $v \in V$ and check whether one of these pairs consists of different numbers.

The idea is now to compute such an interval $I = [L, U]$ in strongly polynomial time.

Consider the function $f_1(v)[\beta]$. Clearly one has

$$f_1(v)[\beta] = \begin{cases} c(s,v) - \beta \cdot p(s,v) & \text{if } (s,v) \in A, \\ \infty & \text{otherwise.} \end{cases}$$

This shows that $f_1(v)[\beta]$ is a linear function in β for each $v \in V$.

Now suppose that $i \geqslant 1$ and that we have computed an interval $I = [L, U]$ with $\beta^* \in I$ and each function $f_i(v)[\beta]$ is a linear function if β is restricted to I.

Now consider the function $f_{i+1}(v)[\beta]$ for a particular $v \in V$. Recall the formula

$$f_{i+1}(v)[\beta] = \min\{f_i(v)[\beta], \min_{(u,v) \in A} \{f_i(u)[\beta] + c(u,v) - \beta \cdot p(u,v)\}\}. \quad (7.23)$$

Each of the functions $f_i(v)[\beta]$ and $f_i(u)[\beta] + c(u,v) - \beta \cdot p(u,v)$ are linear on I. The function $f_i(v)[\beta]$ can be retrieved by computing a shortest path $P_i(v)$ from s to v with arc weights c_β for some β in (L, U) which uses at most i arcs. If β is then allowed to vary, the line which is defined by $f_i(v)[\beta]$ on I is then the length of this path P with parameter β. Similarly we can retrieve the functions (lines) $f_i(u)[\beta] + c(u,v) - \beta \cdot p(u,v)$ for each $(u,v) \in A$. With the Bellman-Ford algorithm, this amounts to a running time of $O(m \cdot n)$.

We now have n lines and can now compute the lower envelope of these lines in time $O(n \log n)$ alternatively we can also compute all intersection points of these lines and sort them w.r.t. increasing β-coordinate. This would amount to $O(n^2 \log n)$. Let $\beta_1, \dots, \beta_k$ be the sorted list of these β-coordinates. Now $\beta_{trial} := \beta_{\lfloor k/2 \rfloor}$ and check whether $\beta^* > \beta_{trial}$. If yes, we can replace L by

β_{trial} and we can delete the numbers $\beta_1, \ldots, \beta_{\lfloor k/2 \rfloor -1}$. Otherwise, we replace U by β_{trial} and delete $\beta_{\lfloor k/2 \rfloor +1}, \ldots, b_k$. In any case, we halved the number of possible β-coordinates and continue in this way. Such a check requires a negative cycle test in the graph D with arc weights β_{trial} and costs $O(m \cdot n)$. In the end we have two consecutive β-coordinates and have an interval $[L, U]$ on which $f_{i+1}(v)[\beta]$ is linear. To find an interval I such that $f_{i+1}(v)[\beta]$ is linear on I and $\beta^* \in I$ costs thus $O(m \cdot n \log n)$ steps.

We now continue to tighten *this interval* such that all functions $f_{i+1}(v)[\beta], v \in V$ are linear on $[L, U]$. Thus in step $i+1$ this amounts to a running time of

$$O\left(n \cdot (m \cdot n \log n)\right).$$

The total running time is thus

$$O(n^3 \cdot m \cdot \log n).$$

Theorem 7.16. *Let $D = (V, A)$ be a directed graph and let $c : A \longrightarrow \mathbb{R}$ and $p : A \longrightarrow \mathbb{R}_{>0}$ be functions. One can compute a cycle C of D minimizing $c(C)/p(C)$ in time $O(n^3 \cdot m \cdot \log n)$.*

7.6.1.1 Exercises

1) Show that there are no two different paths from r to another node in a directed tree $T = (V, A)$.
2) Prove Lemma 7.3.
3) Why can we assume without loss of generality that a minimum cost network has a path from i to j for all $i \neq j \in V$ which is incapacitated?
4) Provide an example of a MCNFP for which the simple cycle-canceling algorithm from above can require an exponential number of cancels, if the cycles are chosen in a disadvantageous way.
5) Provide a proof of Theorem 7.7.
6) Let $Q =< u_1, \ldots, u_k >$ be the queue before an iteration of the **while** loop of the breadth-first-search algorithm. Show that $D[u_i]$ is monotonously increasing and that $D[u_1] + 1 \geqslant D[u_k]$. Conclude that the sequence of assigned labels (over time) is a monotonously increasing sequence.

Chapter 8
The ellipsoid method

It is not known whether the simplex algorithm is an algorithm that runs in polynomial time. For many pivoting rules it was even proved to require an exponential number of iterations [6]. It was long open, whether there exists a polynomial time algorithm for linear programming until Khachiyan [5] showed that the ellipsoid method[16, 14] can solve linear programs in polynomial time. The remarkable fact is that the algorithm is polynomial in the binary encoding length of the linear program. In other words, if the input consists of the problem $\max\{c^T x \colon x \in \mathbb{R}^n,\, Ax \leqslant b\}$, where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, then the algorithm runs in polynomial time in $m + n + s$, where s is the largest binary encoding length of a rational number appearing in A or b. The question, whether there exists an algorithm which runs in time polynomial in $m + n$ and performs arithmetic operations on numbers, whose binary encoding length remains polynomial in $m+n+s$ is one of the most prominent open problems in theoretical computer science and discrete optimization.

Initially, the ellipsoid method can be used to solve the following problem.

> Given a matrix $A \in \mathbb{Z}^{m \times n}$ and a vector $b \in \mathbb{Z}^m$, determine a feasible point x^* in the polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ or assert that P is *not full-dimensional* or P is unbounded.

After we understand how the ellipsoid method solves this problem in polynomial time, we discuss why linear programming can be solved in polynomial time.

Clearly, we can assume that A has full column rank. Otherwise, we can find with Gaussian elimination an invertible matrix $U \in \mathbb{R}^{n \times n}$ with $A \cdot U = \begin{pmatrix} A' & 0 \end{pmatrix}$ where A' has full column rank. The system $A'x \leqslant b$ is then feasible if and only if $Ax \leqslant b$ is feasible.

Exercise 8.1. Let x' be a feasible solution of $A'x \leqslant b$ and suppose that U from above is given. Show how to compute a feasible solution $\tilde{x}$ of $Ax \leqslant b$. Also vice versa, show how to compute x', if $\tilde{x}$ is given.

The *unit ball* is the set $B = \{x \in \mathbb{R}^n \mid \|x\| \leqslant 1\}$ and an *ellipsoid* $E(A, b)$ is the image of the unit ball under a linear map $t : \mathbb{R}^n \to \mathbb{R}^n$ with $t(x) = Ax+b$, where $A \in \mathbb{R}^{n\times n}$ is an invertible matrix and $b \in \mathbb{R}^n$ is a vector. Clearly

$$E(A,b) = \{x \in \mathbb{R}^n \mid \|A^{-1}x - A^{-1}b\| \leqslant 1\}. \tag{8.1}$$

Exercise 8.2. Consider the mapping $t(x) = \left(\begin{smallmatrix}1 & 3\\ 2 & 5\end{smallmatrix}\right)\left(\begin{smallmatrix}x(1)\\ x(2)\end{smallmatrix}\right)$. Draw the ellipsoid which is defined by t. What are the axes of the ellipsoid?

The *volume* of the unit ball is denoted by V_n, where $V_n \sim \frac{1}{\pi\, n}\left(\frac{2\,e\,\pi}{n}\right)^{n/2}$. It follows that the volume of the ellipsoid $E(A,b)$ is equal to $|\det(A)| \cdot V_n$. The next lemma is the key to the development of the ellipsoid method.

Lemma 8.1 (Half-Ball Lemma). *The half-ball* $H = \{x \in \mathbb{R}^n \mid \|x\| \leqslant 1, x_1 \geqslant 0\}$ *is contained in the ellipsoid*

$$E = \left\{x \in \mathbb{R}^n \mid \left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2}\sum_{i=2}^n x_i^2 \leqslant 1\right\} \tag{8.2}$$

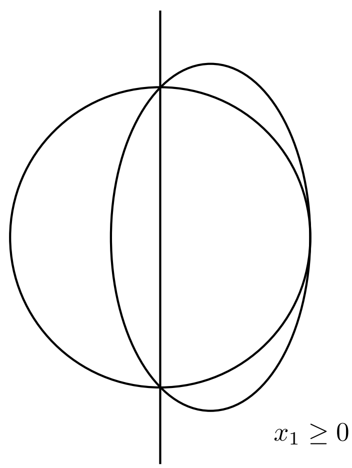


Fig. 8.1: Half-ball lemma.

Proof. Let x be contained in the unit ball, i.e., $\|x\| \leqslant 1$ and suppose further that $0 \leqslant x_1$ holds. We need to show that

$$\left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2}\sum_{i=2}^n x_i^2 \leqslant 1 \tag{8.3}$$

holds. Since $\sum_{i=2}^n x_i^2 \leqslant 1 - x_1^2$ holds we have

$$\left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2}\sum_{i=2}^{n} x_i^2$$
$$\leqslant \left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2}(1-x_1^2) \tag{8.4}$$

This shows that (8.3) holds if x is contained in the half-ball and $x_1 = 0$ or $x_1 = 1$. Now consider the right-hand-side of (8.4) as a function of x_1, i.e., consider

$$f(x_1) = \left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2}(1-x_1^2). \tag{8.5}$$

The first derivative is

$$f'(x_1) = 2\cdot\left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right) - 2\cdot\frac{n^2-1}{n^2}x_1. \tag{8.6}$$

We have $f'(0) < 0$ and since both $f(0) = 1$ and $f(1) = 1$, we have $f(x_1) \leqslant 1$ for all $0 \leqslant x_1 \leqslant 1$ and the assertion follows.

In terms of a matrix A and a vector b, the ellipsoid E is described as $E = \{x \in \mathbb{R}^n \mid \|A^{-1}x - A^{-1}b\|\}$, where A is the diagonal matrix with diagonal entries

$$\frac{n}{n+1}, \sqrt{\frac{n^2}{n^2-1}}, \ldots, \sqrt{\frac{n^2}{n^2-1}}$$

and b is the vector $b = (1/(n+1), 0, \ldots, 0)$. Our ellipsoid E is thus the image of the unit sphere under the linear transformation $t(x) = Ax + b$. The determinant of A is thus $\frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-1)/2}$ which is bounded by

$$e^{-1/(n+1)}e^{(n-1)/(2\cdot(n^2-1))} = e^{-\frac{1}{2(n+1)}}. \tag{8.7}$$

We can conclude the following theorem.

Theorem 8.1. *The half-ball $\{x \in \mathbb{R}^n \mid x_1 \geqslant 0, \|x\| \leqslant 1\}$ is contained in an ellipsoid E, whose volume is bounded by $e^{-\frac{1}{2(n+1)}} \cdot V_n$.*

Recall the following notion from linear algebra. A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is called positive definite if all its eigenvalues are positive. Recall the following theorem.

Theorem 8.2. *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. The following are equivalent.*

i) A is positive definite.
ii) $A = L^T L$, where $L \in \mathbb{R}^{n\times n}$ is a uniquely determined upper triangular matrix.

iii) $x^T A x > 0$ *for each* $x \in \mathbb{R}^n \setminus \{0\}$.
iv) $A = Q^T \mathit{diag}(\lambda_1, \dots, \lambda_n) Q$, *where* $Q \in \mathbb{R}^{n \times n}$ *is an orthogonal matrix and* $\lambda_i \in \mathbb{R}_{>0}$ *for* $i = 1, \dots, n$.

It is now convenient to switch to a different representation of an ellipsoid. An ellipsoid $\mathscr{E}(A, a)$ is the set $\mathscr{E}(A, a) = \{x \in \mathbb{R}^n \mid (x-a)^T A^{-1}(x-a) \leqslant 1\}$, where $A \in \mathbb{R}^{n \times n}$ is a symmetric positive definite matrix and $a \in \mathbb{R}^n$ is a vector. Consider the half-ellipsoid $\mathscr{E}(A, a) \cap (c^T x \leqslant c^T a)$.

Our goal is a similar lemma as the half-ball-lemma for ellipsoids. Geometrically it is clear that each half-ellipsoid $\mathscr{E}(A, a) \cap (c^T x \leqslant c^T a)$ must be contained in another ellipsoid $\mathscr{E}(A', b')$ with $\mathrm{vol}(\mathscr{E}(A', a'))/\mathrm{vol}(\mathscr{E}(A, a)) \leqslant e^{-1/(2n)}$. More precisely this follows from the fact that the half-ellipsoid is the image of the half-ball under a linear transformation. Therefore the image of the ellipsoid E under the same transformation contains the half-ellipsoid. Also, the volume-ratio of the two ellipsoids is invariant under a linear transformation.

We now record the formula for the ellipsoid $\mathscr{E}'(A', a')$. It is defined by

$$a' = a - \frac{1}{n+1} b \tag{8.8}$$

$$A' = \frac{n^2}{n^2 - 1}\left(A - \frac{2}{n+1} b\, b^T\right), \tag{8.9}$$

where b is the vector $b = A\,c/\sqrt{c^T A\, c}$. The proof of the correctness of this formula can be found in [4].

Lemma 8.2 (Half-Ellipsoid-Theorem). *The half-ellipsoid* $\mathscr{E}(A, b) \cap (c^T x \leqslant c^T a)$ *is contained in the ellipsoid* $\mathscr{E}'(A', a')$ *and one has* $\mathrm{vol}(\mathscr{E}')/\mathrm{vol}(\mathscr{E}) \leqslant e^{-1/(2n)}$.

8.1 The method

Suppose we know the following things of our polyhedron P.

I) We have a number L such that $\mathrm{vol}(P) \geqslant L$ if P is full-dimensional.
II) We have an ellipsoid $\mathscr{E}_{init}$ which contains P if P is bounded.

The ellipsoid method is now easily described.

Algorithm 8.1 (Ellipsoid method exact version).

a) (Initialize): Set $\mathscr{E}(A, a) := \mathscr{E}_{init}$
b) If $a \in P$, then assert $P \neq \emptyset$ and stop
c) If $\mathrm{vol}(\mathscr{E}) < L$, then assert that P is unbounded or P is not full-dimensional

d) Otherwise, compute an inequality $c^T x \leqslant \beta$ which is valid for P and satisfies $c^T a > \beta$ and replace $\mathscr{E}(A, a)$ by $\mathscr{E}(A', a)$ computed with formula (8.8) and goto step b).

Theorem 8.3. *The ellipsoid method computes a point in the polyhedron P or asserts that P is unbounded or not full-dimensional. The number of iterations is bounded by $2 \cdot n \ln(\mathrm{vol}(\mathscr{E}_{init})/L)$.*

Proof. Unless P is unbounded, we start with an ellipsoid which contains P. This then holds for all the subsequently computed ellipsoids. After i iterations one has

$$\mathrm{vol}(\mathscr{E})/\mathrm{vol}(\mathscr{E}_{init}) \leqslant e^{-\frac{i}{2n}}. \tag{8.10}$$

Since we stop when $\mathrm{vol}(\mathscr{E}) < L$, we stop at least after $2 \cdot n \ln(\mathrm{vol}(\mathscr{E}_{init})/L)$ iterations. This shows the claim.

8.1.1 The separation problem

At this point we can already notice a very important fact. Inspect step d of the algorithm. What is required here ? An inequality which is valid for P but not for the center a of $\mathscr{E}(A, a)$. Such an inequality is readily at hand if we have the complete inequality description of P in terms of a system $Cx \leqslant d$. Just pick an inequality which is violated by a. Sometimes however, it is not possible to describe the polyhedron of a combinatorial optimization problem with an inequality system efficiently, simply because the number of inequalities is too large. An example of such a polyhedron is the matching polytope, see Theorem 6.5.

The great power of the ellipsoid method lies in the fact that we do not have to *write down* the polyhedron entirely. We only have to solve the so-called separation problem for the polyhedron, which is defined as follows.

> Separation Problem
>
> Given a point $a \in \mathbb{R}^n$ determine, whether $a \in P$ and if not, compute an inequality $c^T x \leqslant \beta$ which is valid for P with $c^T a > \beta$.

Exercise 8.3. We are given an undirected graph $G = (V, E)$. A *spanning tree* T is a subset $T \subseteq E$ of the edges such that T does not contain a cycle and T connects all the vertices V. Consider the following *spanning tree polytope* P_{span}

$$\sum_{e \in E} x(e) = n - 1 \tag{8.11}$$

$$\sum_{e \in \delta(U)} x(e) \geqslant 1 \qquad \forall \emptyset \subset U \subset V \tag{8.12}$$

$$x(e) \leqslant 1 \qquad \forall e \in E \tag{8.13}$$

$$x(e) \geqslant 0 \qquad \forall e \in E. \tag{8.14}$$

Let x be an integral solution of P_{span} and define $T = \{e \in E \mid x(e) = 1\}$. The inequality (8.11) ensures that exactly $n-1$ edges are picked. The inequalities (8.12) ensure that T connects the vertices of G. Thus T must be a spanning tree. Clearly, there are exponentially many inequalities of type (8.12). Nevertheless, a fractional solution of this polytope can be computed using the ellipsoid method.

Show that the separation problem for P_{span} can be solved in polynomial time.

Hint: To verify whether a vector $x \in \mathbb{R}^{|E|}_{\geqslant 0}$ fulfills inequalities of type (8.12), it is a good idea to recall the MinCut or MaxFlow problem.

Via binary search even an optimal solution can be computed in polynomial time (in the input length) if we introduce edge costs (you don't have to show that). In the next semester you will see that any optimal basis solution is integral and hence defines an optimal spanning tree w.r.t. the edge costs.

Exercise 8.4. Consider the triangle defined by

$$\begin{aligned} -x_1 - x_2 &\leqslant -2 \\ 3x_1 &\leqslant 4 \\ -2x_1 + 2x_2 &\leqslant 3. \end{aligned}$$

Draw the triangle and simulate the ellipsoid method with starting ellipsoid being the ball of radius 6 around 0. Draw each of the computed ellipsoids with your favorite program (pstrics, maple ...). How many iterations does the ellipsoid method take?

Ignore the occurring rounding errors!

8.2 How to start and when to stop

In our description of the ellipsoid method, we did not explain yet what the initial ellipsoid is and when we can stop with asserting that P is either not full-dimensional or unbounded.

Suppose therefore that $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ is full-dimensional and bounded with $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$. Let B be the largest absolute value of a component of A and b. In this section we will show the following things.

i) The vertices of P are in the box $\{x \in \mathbb{R}^n \mid -n^{n/2}B^n \leqslant x \leqslant n^{n/2}B^n\}$. Thus P is contained in the ball around 0 with radius n^nB^n . Observe that the encoding length of this radius is $\mathrm{size}(n^nB^n) = O(n\log n + n\,\mathrm{size}(B))$ which is polynomial in the dimension n and the largest encoding length of a coefficient of A and b.
ii) The volume of P is bounded from below by $1/(n \cdot B)^{3n^2}$.

The following lemma is proved in any linear algebra course.

Lemma 8.3 (Inverse formula and Cramer's rule). *Let $C \in \mathbb{R}^{n\times n}$ be a nonsingular matrix. Then*

$$C^{-1}(j,i) = (-1)^{i+j}\det(C_{ij})/\det(C),$$

where C_{ij} is the matrix arising from C by the deletion of the i-th row and j-th column. If $d \in \mathbb{R}^n$ is a vector then the j-th component of $C^{-1}d$ is given by $\det(\widetilde{C})/\det(C)$, where $\widetilde{C}$ arises from C be replacing the j-th column with d.

We now define the size of a rational number $r = p/q$ with p and q relatively prime integers, a vector $c \in \mathbb{Q}^n$ and a matrix $A \in \mathbb{Q}^{m\times n}$:

- $\mathrm{size}(r) = 1 + \lceil\log(|p|+1)\rceil + \lceil\log(|q|+1)\rceil$
- $\mathrm{size}(c) = n + \sum_{i=1}^n \mathrm{size}(c(i))$
- $\mathrm{size}(A) = m\cdot n + \sum_{i=1}^n\sum_{j=1}^m \mathrm{size}(A(i,j))$

We recall the Hadamard inequality which states that for $A \in \mathbb{R}^{n\times n}$ one has

$$|\det(A)| \leqslant \prod_{i=1}^n \|a_i\|, \tag{8.15}$$

where a_i denotes the i-th column of A. In particular, if B is the largest absolute value of an entry in A, then

$$|\det(A)| \leqslant n^{n/2}B^n. \tag{8.16}$$

Now let us inspect the vertices of a polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$, where A and b are integral and the largest absolute value of any entry in A and b is bounded by B. A vertex is determined as the unique solution of a linear system $A'x = b'$, where $A'x \leqslant b'$ is a subsystem of $Ax \leqslant b$ and A' is invertible. Using Cramer's rule and our observation (8.16) we see that the vertices of P lie in the box $\{x \in \mathbb{R}^n \mid -n^{n/2}B^n \leqslant x \leqslant n^{n/2}B^n\}$. This shows i).

Now let us consider a lower bound on the volume of P. Since P is full-dimensional, there exist $n+1$ affinely independent vertices $v_0,\ldots,v_n$ of P which span a *simplex* in $\mathbb{R}^n$. The volume of this simplex is determined by the formula

$$\frac{1}{n!}\cdot\left|\det\begin{pmatrix}1 & \cdots & 1\\ v_0 & \ldots & v_n\end{pmatrix}\right|. \tag{8.17}$$

By Cramer's rule and the Hadamard inequality, the common denominator of each component of v_i can be bounded by $n^{n/2}B^n$. Thus (8.17) is bounded by

$$1/\left(n^n(n^{\frac{n}{2}} \cdot B^n)^{n+1}\right) \geqslant 1/\left(n^{3n^2}B^{2n^2}\right) \geqslant 1/(n \cdot B)^{3 \cdot n^2}, \tag{8.18}$$

which shows ii).

Now we plug these values into our analysis in Theorem 8.3. Our initial volume $\mathrm{vol}(\mathscr{E}_{init})$ is bounded by the volume of the box with side-lengths $2(n \cdot B)^n$. Thus

$$\mathrm{vol}(\mathscr{E}_{init}) \leqslant (2 \cdot n \cdot B)^{n^2}. \tag{8.19}$$

Above we have shown that

$$L \geqslant 1/(n \cdot B)^{3n^2}. \tag{8.20}$$

Clearly

$$\mathrm{vol}(\mathscr{E}_{init})/L \leqslant (n \cdot B)^{4 \cdot n^2}. \tag{8.21}$$

By Theorem 8.3 the ellipsoid method performs

$$O\left(2 \cdot n \cdot \ln\left((n \cdot B)^{4 \cdot n^2}\right)\right) \tag{8.22}$$

iterations. This is bounded by

$$O(n^3 \cdot \ln(n \cdot B)). \tag{8.23}$$

Now recall that $\log B$ is the number of *bits* which are needed to encode the coefficient with the largest absolute value of the constraint system $Ax \leqslant b$ and that n is the number of variables of this system. Therefore the expression (8.23) is polynomial in the binary input encoding of the system $Ax \leqslant b$. We conclude the following theorem.

Theorem 8.4. *The ellipsoid method (exact version) performs a polynomial number of iterations.*

8.3 The boundedness and full-dimensionality condition

In this section we want to show how the ellipsoid method can be used to solve the following problem.

> Given a matrix $A \in \mathbb{Z}^{m \times n}$ and a vector $b \in \mathbb{Z}^m$, determine a feasible point x^* in the polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ or assert that $P = \emptyset$.

8.3.1 Boundedness

We have argued that the matrix $A \in \mathbb{Z}^{m\times n}$ can be assumed to have full column rank. So, if P is not empty, then P does have at least one vertex. The vertices are contained in the box $\{x \in \mathbb{R}^n \mid -n^{n/2}B^n \leqslant x \leqslant n^{n/2}B^n\}$. Therefore, we can append the inequalities $-n^{n/2}B^n \leqslant x \leqslant n^{n/2}B^n$ to $Ax \leqslant b$ without changing the status of $P \neq \emptyset$ or $P = \emptyset$. Notice that the *binary encoding length* of the new inequalities is polynomial in the binary encoding length of the old inequalities.

8.3.2 Full-dimensionality

Exercise 8.5. Let $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ be a polyhedron and $\varepsilon > 0$ be a real number. Show that $P_\varepsilon = \{x \in \mathbb{R}^n \mid Ax \leqslant b + \varepsilon \cdot \mathbf{1}\}$ is full-dimensional if $P \neq \emptyset$.

The above exercise raises the following question. Is there an $\varepsilon > 0$ such that $P_\varepsilon = \emptyset$ if and only if $P = \emptyset$ and furthermore is the binary encoding length of this ε polynomial in the binary encoding length of A and b ?

Recall Farkas' Lemma (Theorem 2.11 and Exercise 10 of chapter 2).

Theorem 8.5. *The system $Ax \leqslant b$ does not have a solution if and only if there exists a nonnegative vector $\lambda \in \mathbb{R}^m_{\geqslant 0}$ such that $\lambda^T A = 0$ and $\lambda^T b = -1$.*

Let $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$ and let B be the largest absolute value of a coefficient of A and b. If $Ax \leqslant b$ is not feasible, then there exists a $\lambda \geqslant 0$ such that $\lambda^T(A|b) = (\mathbf{0}| - 1)$. We want to estimate the largest absolute value of a coefficient of λ with Cramer's rule and the Hadamard inequality. We can choose λ such that the nonzero coefficients of λ are the unique solution of a system of equations $Cx = d$, where each coefficient has absolute value at most B. By Cramer's rule and the Hadamard inequality we can thus choose λ such that $|\lambda(i)| \leqslant (n \cdot B)^n$. Now let $\varepsilon = 1/\left((n+1)\cdot(n\cdot B)^n\right)$. Then $|\lambda^T \mathbf{1} \cdot \varepsilon| < 1$ and thus

$$\lambda^T(b + \varepsilon \cdot \mathbf{1}) < 0. \tag{8.24}$$

Consequently the system $Ax \leqslant b + \varepsilon\mathbf{1}$ is infeasible if and only of $Ax \leqslant b$ is infeasible. Notice again that the encoding length of ε is polynomial in the encoding length of $Ax \leqslant b$ and we conclude with the main theorem of this section.

Theorem 8.6. *The ellipsoid method can be used to decide whether a system of inequalities $Ax \leqslant b$ contains a feasible point, where $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$. The number of iterations is bounded by a polynomial in n and $\log B$, where B is the largest absolute value of a coefficient of A and b.*

8.4 The ellipsoid method for optimization

Suppose that you want to solve a linear program

$$\max\{c^T x \mid x \in \mathbb{R}^n,\, Ax \leqslant b\} \tag{8.25}$$

and recall that if (8.25) is bounded and feasible, then so is its dual and the two objective values are equal. Thus, we can use the ellipsoid method to find a point (x, y) with $c^T x = b^T y$, $Ax \leqslant b$ and $A^T y = c, y \geqslant 0$.

However, we mentioned that the strength of the ellipsoid method lies in the fact that we do not need to write the system $Ax \leqslant b$ down explicitly. The only thing which has to be solvable is the *separation problem*. This is to be exploited in the next exercise.

Exercise 8.6. Show how to solve the optimization problem $\max\{c^T x \mid Ax \leqslant b\}$ with a polynomial number of calls to an algorithm which solves the separation problem for $Ax \leqslant b$. You may assume that A has full column rank and the polynomial bound on the number of calls to the algorithm to solve the separation problem can depend on n and the largest size of a component of A, b and c.

8.5 Numerical issues

We did not discuss the numerical details on how to implement the ellipsoid method such that it runs in polynomial time. One issue is crucial.

> !
>
> We only want to compute with a precision which is polynomial in the input encoding!

In the formula (8.8) the vector b is defined by taking a square root. The question thus rises on how to round the numbers in the intermediate ellipsoids such that they can be handled on a machine. Also one has to analyze the growth of the numbers in the course of the algorithm. All these issues can be overcome but we do not discuss them in this course. I would like to refer you to the book of Alexander Schrijver [15] for further details. They are not difficult, but a little technical.

Chapter 9
Primal-Dual algorithm

In this chapter we talk about matching problems, a very important topic in combinatorial optimization.

9.1 Graphs and Matchings

Recall the following definitions, which will be very useful in this chapter.

Definition 9.1 (Undirected Graph). An *undirected graph* $G = (V, E)$ consists of a set V of *vertices* (also called *nodes*) and a set E of *edges*. An edge $e = u, v$ is a 2-elements subset of V, where u and v are called the *endpoints* of e.

Definition 9.2 (Directed Graph). A *directed graph* $G = (V, A)$ consists of a set V of *vertices* (also called *nodes*) and a set A of *arcs*. An arc a is an ordered pair $(u, v) \in V \times V$, where u and v are called the *tail* and the *head* of a respectively. Moreover u and v are also called the *endpoints* of a.

Definition 9.3. A graph $G = (V, E)$ or $G = (V, A)$ is called *bipartite* if we can split V in two disjoint subsets V_1 and V_2 such that for every $e \in E$ (or $a \in A$) one endpoint in V_1 and the other is in V_2.

We recall an important result about bipartite graphs.

Lemma 9.1. *A graph is bipartite if and only if it does not contain an odd cycle (that is, a cycle of odd length).*

Definition 9.4 (Matching). Let $G = (V, E)$,or $G = (V, A)$, be a graph. A *matching* in G is a subset $M \subseteq E$,or $M \subseteq A$, of pairwise disjoint edges, or arcs (that is, edges, or arcs, which do not share any vertex).

A vertex v that is an endpoint of an edge (or arc) in M is called a *matched node*, otherwise it is called an *exposed node*.

A matching is called *perfect* if there are no exposed nodes.

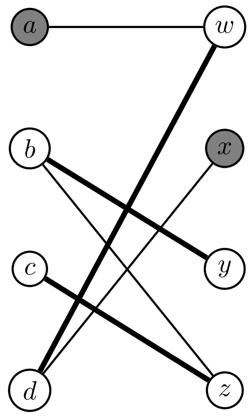


Fig. 9.1: This picture shows an example of an undirected bipartite graph $G = (V, E)$, where $V = a, b, c, d, w, x, y, z$, $E = a, w, b, y, b, z, c, z, d, w, d, x$, $V_1 = a, b, c, d$ and $V_2 = w, x, y, z$. The thicker edges form a matching, vertices a and x (marked in grey) are exposed nodes and a, w, d, x is an alternating and augmenting path.

Definition 9.5. Let $G = (V, E)$ be a graph and $M \in E$ be a matching in G, an *alternating path* is a path that alternates between edges in M and edges in $E \backslash M$. An alternating path that starts and ends at exposed nodes is called an *augmenting path*.

Definition 9.6 (Vertex cover). Let $G = (V, E)$ be a graph. A *vertex cover* is a set $C \in V$ such that every $e \in E$ has at least one endpoint in C.

9.2 Matching problems

Now that we know the definition of a matching, we can concentrate on two of the more important problems about matchings:

1. **Maximum cardinality matching problem**: Find a matching M of maximum size.
2. **Maximum (or minimum) weight matching problem**: Given a graph $G = (V, E)$ and a weight function $w : \ E \longrightarrow \mathbb{R}$, find a matching M with maximum (or minimum) weight, where the weight of a matching is
$$w(M) = \sum_{e \in M} w(e).$$

We will focus in the case where we have an undirected bipartite graph $G = (V, E)$.

9.2.1 *The maximum cardinality matching problem*

Before giving a method to find a maximum cardinality matching in a graph, we want to show how we can prove the optimality. For this purpose we recall theorem 6.3, that gives us an upper bound on the size of any matching in a given bipartite graph, and we prove another theoreme, which gives us a method to see if a matching is of maximum size or not.

Theorem 9.1 (König's theorem). *In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover.*

Theorem 9.2. *A matching M of a graph $G = (V, E)$ is of maximum cardinality if and only if there are no augmenting paths with respect to M.*

Proof. ($\Rightarrow$) Suppose there exists an augmenting path in G, call it P. Consider the set $M' = M \triangle P$. By definition of augmenting path, we have that M' has exactly one edge more than M. Since M is a matching and P starts and ends in an exposed node, the only edges of M with an endpoint in P are edges of P. This implies that M' is a matching.

($\Leftarrow$) Let M be a maximum cardinality matching and let $\tilde{M}$ be a strictly smaller matching. Consider $X = M \triangle \tilde{M}$. X is formed by cycles and alternating paths (with respect to M and $\tilde{M}$). Since $|M| > |\tilde{M}|$, X contains at least one alternating path P with more edges from M than from $\tilde{M}$. By definition of X, we see that P is an augmenting path. $\square$

We are now ready to show an algorithm that allow us to find a maximum cardinality matching in any bipartite graph.

Let $G = (V, E)$ be a bipartite graph with bipartition $V = A \sqcup B$ and M a matching. Now turn G into a directed graph $D = (V, A)$ by directing matching edges from A to B and non-matching edges from B to A. We are interested in a method to find augmenting paths or to assert that there aren't any (and thus, that M is of maximum size). For this purpose we illustrate the following claim and theorem.

Claim. Let G and D be as above. A path in D between two exposed nodes that starts with an exposed node in B (resp. in A), ends in an exposed node in A (resp. in B).

Theorem 9.3. *Let G and D be as above and M be a matching. There exists an augmenting path in G if and only if there exists a path from an exposed node in B to an exposed node in A in the directed graph D.*

Proof. ($\Rightarrow$) This is a direct consequence of the choice of the direction of the arcs in D and of the previous claim.

($\Leftarrow$) Trivial.

Let G and D be as illustrated before. How can we use the previous theorem for our purpose? If we add a vertex s to our directed graph D and connect it to all exposed nodes in B by an arc whose tail is s. Now we have that finding a directed path, in D, from an exposed node u in B to an exposed node v in A is equivalent to find a directed path from s to v passing by u. Thus, we deduce that there is an augmenting path in G if and only if there is at least one exposed node in A reachable from s. To find if there are such nodes reachable from s and the corresponding augmenting paths, we can use the *Breadth-First search* algorithm (see chapter 7.2.1) applied to vertex s in D. If there is any exposed node u in A with finite distance from s, than we can obtain an augmenting path by taking the shortest path $s, a_1, a_2, \cdots, u$ from s to u (which is given by Bread-First search) without s (i.e., the augmenting path would be $a_1, a_2, \cdots, u$).

To resume, we obtain the following algorithm.

Algorithm 9.1.
Initialise $M = \emptyset$
while *There exists M-augmenting path*
  Update M
return M

Finally, we are interested in the running time of our algorithm.

Theorem 9.4. *A maximum cardinality matching in a bipartite graph $G = (V, E)$ can be computed in time $O(|V| \cdot (|V| + |E|))$. If we assume that G does not have any isolated vertex, then we can consider the running time to be $O(|V| \cdot |E|)$.*

Proof. The *while loop* runs at most $|V|/2$ times its execution requires $O(|V| + |E|)$ $(= O(|E|)$ if we have the assumption) operations.

9.2.2 The maximum weight matching problem

In this section we consider a graph $G = (V, E)$ and a weight function $w: E \longrightarrow \mathbb{R}$. First of all, notice that, by changing the sign of the weights of all edges, we have that finding a maximum weight matching or a minimum weight matching are equivalent problems.

We can also prove that the problem of finding a minimum weight matching can always be replaced by the problem of finding a minimum weight *perfect* matching. To see that, it is sufficient to create a copy G' of our graph G and

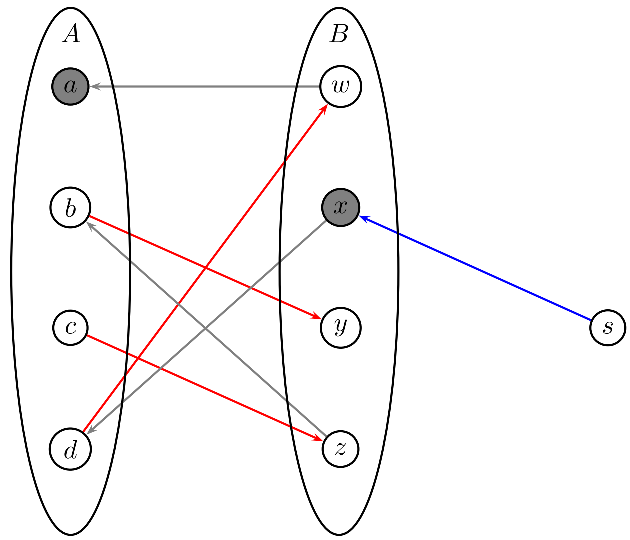


Fig. 9.2: This picture illustrate how transform a bipartite graph to apply the *maximum cardinality matching algorithm*, using the graph of figure 9.1 as an example.

add an edge of weight 0 connecting all vertices v in G to their copy v' in G'. This will give us a new graph $\tilde{G}$. Notice that $\tilde{G}$ has at least a perfect matching, thus there is one, call it $\tilde{M}$, of minimum weight. If we consider only the edges of $\tilde{M}$ in G, we obtain a minimum weight matching in G.

Moreover, we have an important relation between the difficulty of this two problems, given by the following theorem.

Theorem 9.5. *If there exists a polynomial time algorithm for minimum weight perfect matching, then there exists a polynomial time algorithm for minimum weight matching.*

Proof. Let $G = (V, E)$ be a graph. Using the process described above, we create the graph $\tilde{G} = (\tilde{V}, \tilde{E})$. Notice that $|\tilde{V}| = 2|V|$ and $|\tilde{E}| = 2|E| + |V|$. By hypothesis, we can find a minimum weight perfect matching in $\tilde{G}$ (and thus a minimum weight matching in G) in time $O((\tilde{V} + \tilde{E})^k) = O((2|V| + 2|E| + |V|)^k) = O((|V| + |E|)^{k'})$, for some $k, k' \in \mathbb{N}$ (that is, a polynomial time with respect to the size of G).

Since we concentrate our attention to bipartite graph, we show how to reduce the problem of finding a minimum weight *bipartite* matching to the problem of finding a minimum weight *bipartite* perfect matching. Let $G =$

(V, E) be a bipartite graph, with bipartition $V = A \sqcup B$. We can suppose w.l.o.g. that $|A| \geqslant |B|$. If $|A| > |B|$, add a set C of cardinality $|A| - |B|$ and add edges of weight zero connecting all nodes of A to all nodes of $B \cup C$. Call G' the bipartite graph obtained by this process. Obviously, G' admits at least one bipartite perfect matching, and thus also one of minimum weight, call it M'. By taking out all edges of weight zero from M', we obtain a bipartite minimum weight graph in G.

Also in this case, we have an important relation between the difficulty of these two matching problems.

Theorem 9.6. *If there exists a polynomial time algorithm for minimum weight bipartite perfect matching, then there exists a polynomial time algorithm for minimum weight bipartite matching.*

Proof. Let $G = (V, E)$ be a bipartite graph with bipartition $V = A \sqcup B$. Construct $G' = (V', E')$ as shown above. Notice that $|V'| = 2|A| \leqslant 2|V|$ and $|E'| = |E| + |A| \cdot (|A| - 1) \leqslant |E| + |V|^2$. Hence, by hypothesis, finding a minimum weight bipartite graph in G (by finding a minimum weight bipartite perfect matching in G') can be done in time $O((2|V|+|E|+|V|^2)^k) = O((|V|+|E|)^{k'})$, for some $k, k' \in \mathbb{N}$ (that is, a polynomial time with respect to the size of G).

Let $G = (V, E)$ be a bipartite graph. We have showed that the problem of finding a maximum weight (bipartite) matching can be reduced to the problem of finding a minimum weight (bipartite) perfect matching. Moreover, if there exists a polynomial time algorithm to solve the first problem, then there exists a polynomial time algorithm to solve the second one.

An IP formulation

Let $G = (V, E)$ be a graph. Recall the IP formulation of the minimum weight perfect matching problem, seen in chapter 6.

$$\begin{gathered} \min \textstyle\sum_{e \in E} w(e) \cdot x_e \\ v \in V : \textstyle\sum_{e \in \delta(v)} x_e = 1 \\ x \geqslant 0 \\ x \in \mathbb{Z}^{|E|}, \end{gathered} \tag{9.1}$$

where x is the characteristic vector of matchings.

We also recall his dual.

$$\begin{gathered} \min w^T \cdot x \\ A_G * x = \mathbf{1} \\ x \geqslant 0 \\ x \in \mathbb{Z}^{|E|}, \end{gathered} \tag{9.2}$$

where A_G is the node-incidence matrix of G and $w \in$

References

1. J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69:125–130, 1965.
2. J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
3. F. Eisenbrand, A. Karrenbauer, and C. Xu. Algorithms for longer oled lifetime. In *6th International Workshop on Experimental Algorithms, (WEA 07)*, pages 338–351, 2007.
4. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.
5. L. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1097, 1979.
6. V. Klee and G. J. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. Academic Press, New York, 1972.
7. T. Koch. *Rapid Mathematical Programming*. PhD thesis, Technische Universität Berlin, 2004. ZIB-Report 04-58.
8. B. Korte and J. Vygen. *Combinatorial optimization*, volume 21 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition, 2002. Theory and algorithms.
9. E. L. Lawler. *Combinatorial optimization: networks and matroids*. Holt, Rinehart and Winston, New York, 1976.
10. L. Lovász. Graph theory and integer programming. *Annals of Discrete Mathematics*, 4:141–158, 1979.
11. J. E. Marsden and M. J. Hoffman. *Elementary Classical Analysis*. Freeman, 2 edition, 1993.
12. J. Matoušek and B. Gärtner. *Understanding and Using Linear Programming (Universitext)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
13. N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979.
14. A. S. Nemirovskiy and D. B. Yudin. Informational complexity of mathematical programming. *Izvestiya Akademii Nauk SSSR. Tekhnicheskaya Kibernetika*, (1):88–117, 1983.
15. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley, 1986.
16. N. Z. Shor. Cut-off method with space extension in convex programming problems. *Cybernetics and systems analysis*, 13(1):94–96, 1977.