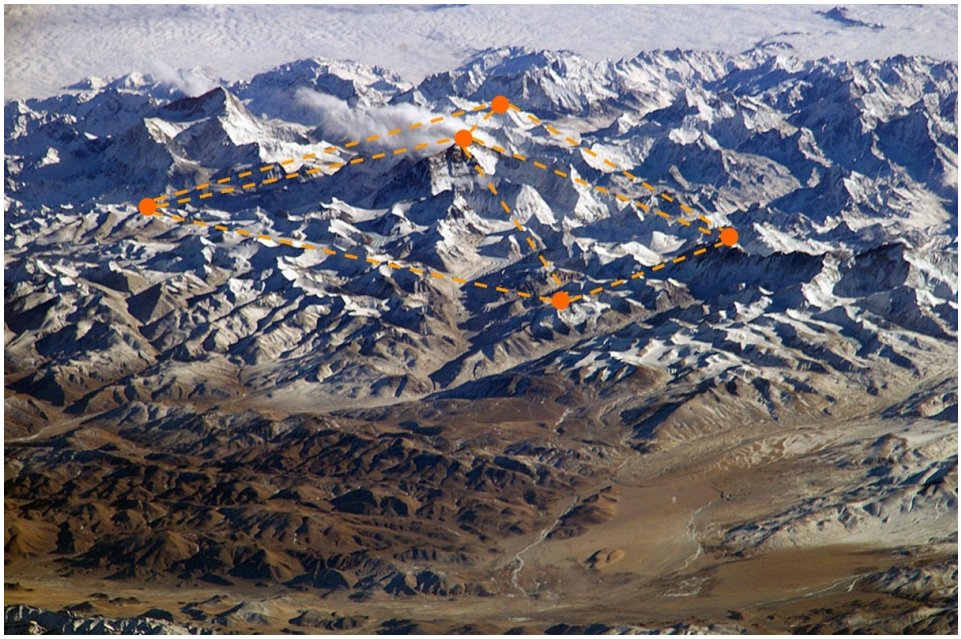
MSE | MASTER OF SCIENCE IN ENGINEERING

HS22, MSE
Module FTP_Optimiz_A/B

OPTIMIZATION

Part 1:
Linear and Integer Linear Optimization

Prof. Dr. Andreas Klinkert
with
Dr. Peter Fusek
Dr. Stephan Bütikofer



Institute of Data Analysis and Process Design (IDP)
Zurich University of Applied Sciences (ZHAW)
Rosenstrasse 3, P.O.Box, CH-8401 Winterthur
andreas.klinkert@zhaw.ch, peter.fusek@zhaw.ch, stephan.buetikofer@zhaw.ch
www.zhaw.ch/idp

Version: 16.09.2022, 11:39

"The true sign of intelligence
is not knowledge but imagination."

A. Einstein

Cover page photo: The topological maximum of this world. The Himalaya mountains and the Tibetian plateau. Photographed from the International Space Station ISS from an altitude of 120 miles.

Contents

Definitions and Notation

0.1 Sets

A *set* is a collection of (finitely many or infinitely many) different objects, called *elements* of the set. $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{Z}$ denote the sets of *real*, *rational* and *integer* numbers. $\mathbb{R}_0$, $\mathbb{Q}_0$ and $\mathbb{Z}_0$ denote these sets constrained to *non-negative* elements, for example $\mathbb{Z}_0 = \{x \in \mathbb{Z} : x \geq 0\}$.

A set S can be stated by the enumeration of its elements, for instance $S = \{1, 3, 5, 7, \ldots\}$, or by specification of a property which selects exactly the elements of S out of a certain base set. In this way $S = \{x \in \mathbb{Z} : x = 2k + 1, k = 0, 1, \ldots, \}$ describes the set of all odd positive numbers.

The *empty set* is denoted by $\emptyset$. The *cardinality* of a set is defined to be the number of elements of the set and is denoted by $|S|$. For instance, with $S = \{x \in \mathbb{Z} : 1 \leq x \leq 10,$ x is a prime number$\}$ we have $|S| = 4$. A set of sets is called a *collection.* The *power set* $\mathcal{P}(S)$ of a set S is defined as the collection of all subsets of S, i.e. $\mathcal{P}(S) = \{S' : S' \subseteq S\}$. For $|S| = n$ it holds $|\mathcal{P}(S)| = \sum_{k=0}^{n} \binom{n}{k} = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} = 2^n$.

Let S be a set. The relation $x \in S$ means that x is an *element* of S, and $x \notin S$ means that x is not an element of S.

We consider following operations for two sets S, T: The *union* $S \cup T = \{x : x \in S$ or $x \in T\}$, the *intersection* $S \cap T = \{x : x \in S$ and $x \in T\}$, and the *difference* $S - T = \{x : x \in S$ and $x \notin T\}$. Notice that $S - T \neq T - S$ if $S \neq T$.

We consider following relations for two sets S, T: $S \subseteq T$ means that S is a *subset* of T, i.e. for all $x \in S$ we have $x \in T$, and $S \subset T$ means that S is a *proper subset* of T, i.e. $S \subseteq T$ and $S \neq T$. The relation $S \nsubseteq T$ means that S is no subset of T, i.e. there is an element $x \in S$ with $x \notin T$.

Two sets S and T are called *disjoint* if $S \cap T = \emptyset$; a collection of sets $\{S_1, S_2, \ldots, S_k\}$ is called (pairwise) disjoint if $S_i \cap S_j = \emptyset$ for all $1 \leq i < j \leq k$. The sets $S_1, S_2, \ldots, S_k$ produce a *partition* of the set S if they are pairwise disjoint and $\bigcup_{i=1,\ldots,k} S_i = S$.

Let $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ be a collection of sets. A set $S \in \mathcal{S}$ is called *inclusionwise maximal in* $\mathcal{S}$ if there is no $S' \in \mathcal{S}$ with $S \subset S'$. A set $S \in \mathcal{S}$ is called *cardinality-wise maximal in* $\mathcal{S}$ if there is no $S' \in \mathcal{S}$ with $|S| < |S'|$.

0.2 Vectors, Matrices, and Linearity

For this section, let S be an arbitrary non-empty set, I and J arbitrary non-empty finite sets, and m and n positive integers.

Vectors

The notation S^n denotes the *set of n-dimensional vectors* with values in the set S, i.e. a vector $\boldsymbol{x} \in S^n$ has the form

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$$

with $x_i \in S$ for $i = 1, \dots, n$. Vectors will be always regarded as *column vectors*. To denote a vector $\boldsymbol{x} \in S^n$, we sometimes write $\boldsymbol{x} = (x_i \in S : i = 1, \dots, n)$. For a column vector $\boldsymbol{x} \in S^n$, the notation $\boldsymbol{x}^\top$ (transpose operator) describes the corresponding *row vector*, i.e. $\boldsymbol{x}^\top = (x_1, x_2, \dots, x_n)$. Hence, for a row vector $\boldsymbol{x}^\top$, the notation $(\boldsymbol{x}^\top)^\top$ describes the corresponding column vector. In the following, we sometimes omit the transpose operator if it is obvious from the context which kind of vector is meant.

The n-dimensional 0-vector $(0, 0, \dots, 0)^\top \in \mathbb{R}^n$ is denoted by $\mathbf{0}^{(n)}$, and the n-dimensional 1-vector $(1, 1, \dots, 1)^\top \in \mathbb{R}^n$ by $\mathbf{1}^{(n)}$. Instead of $\mathbf{0}^{(n)}$ and $\mathbf{1}^{(n)}$, we often write $\mathbf{0}$ and $\mathbf{1}$ if the dimensions are clear from the context.

For vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, we define the relations $=, \leq, \geq, <, >, \dots$ (known from real numbers) to hold component-wise. For instance, $\boldsymbol{x} \leq \boldsymbol{y}$ means that $x_i \leq y_i$ for all $i \in 1, \dots, n$. The relation $\boldsymbol{x} \leq_{lex} \boldsymbol{y}$ means that either $\boldsymbol{x} = \boldsymbol{y}$ or there exists $k \in \{1, \dots, n\}$ such that $x_i = y_i$ for $i = 1, \dots, k-1$ and $x_k < y_k$. We say that $\boldsymbol{x}$ *is lexicografically less or equal to* $\boldsymbol{y}$.

Matrices

The notation $S^{m \times n}$ denotes the set of $m \times n$-*dimensional matrices* with values in the set S, i.e. a matrix $\boldsymbol{A} \in S^{m \times n}$ has form

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ a_{m1} & \dots & \dots & a_{mn} \end{pmatrix}$$

with $a_{ij} \in S$ for $i = 1, \dots, m, j = 1, \dots, n$. Here, m is the number of rows and n is the number of columns of $\boldsymbol{A}$. To denote a matrix $\boldsymbol{A} \in S^{m \times n}$, we sometimes write $\boldsymbol{A} = (a_{ij} \in S : i = 1, ..., m, j = 1, \dots, n)$. Further, $\boldsymbol{A}_j$ denotes the j-*th column* of $\boldsymbol{A}$, i.e.

$\boldsymbol{A}_j = (a_{1j}, a_{2j}, \ldots, a_{mj})^\top$, and $\boldsymbol{a}^i$ denotes the *i-th row* of $\boldsymbol{A}$, i.e. $\boldsymbol{a}^i = (a_{i1}, a_{i2}, \ldots, a_{in})$. Note that $\boldsymbol{a}^i$ is always assumed to be a *row* vector.

The $m \times n$-dimensional 0-*matrix* $\mathbf{0}^{(n,m)}$ is given by $\mathbf{0}^{(n,m)} = \{0\}^{m \times n}$. The *n-dimensional identity matrix* $\boldsymbol{I}^{(n)} \in \{0,1\}^{n \times n}$ is given by

$$\boldsymbol{I}^{(n)} = (a_{ij} \in \{0,1\} : a_{ii} = 1 \text{ for } 1 \leq i \leq n \text{ and } a_{ij} = 0 \text{ for } 1 \leq i, j \leq n, i \neq j).$$

Instead of $\mathbf{0}^{(m \times n)}$ and $\boldsymbol{I}^{(n)}$, we often write $\mathbf{0}$ and $\boldsymbol{I}$ if the dimensions are clear from the context.

Multiplication of Vectors and Matrices

The *multiplication of a row vector* $\boldsymbol{a}^\top$ $(\boldsymbol{a} \in \mathbb{R}^n)$ *by a column vector* $\boldsymbol{x} \in \mathbb{R}^n$ (also called *scalar product*) results in a scalar (a number) given by

$$\boldsymbol{a}^\top \boldsymbol{x} = (a_1, \ldots, a_n) \begin{pmatrix} x_1 \\ \ldots \\ x_n \end{pmatrix} = \sum_{j=1}^{n} a_j x_j$$

The *multiplication of a matrix* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ *by a column vector* $\boldsymbol{x} \in \mathbb{R}^n$ ("right multiplication") results in a m-dimensional column vector and is defined by

$$\boldsymbol{A}\boldsymbol{x} = \begin{pmatrix} \boldsymbol{a}^1 \boldsymbol{x} \\ \ldots \\ \boldsymbol{a}^m \boldsymbol{x} \end{pmatrix} = \begin{pmatrix} (a_{11}, \ldots, a_{1n}) \begin{pmatrix} x_1 \\ \ldots \\ x_n \end{pmatrix} \\ \ldots \\ (a_{m1}, \ldots, a_{mn}) \begin{pmatrix} x_1 \\ \ldots \\ x_n \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^n a_{1j} x_j \\ \ldots \\ \sum_{j=1}^n a_{mj} x_j \end{pmatrix}$$

Notice that "right multiplication" can also be regarded as a linear combination of the columns of $\boldsymbol{A}$:

$$\boldsymbol{A}\boldsymbol{x} = \sum_{j=1}^{n} \boldsymbol{A}_j x_j = \begin{pmatrix} a_{11} \\ \ldots \\ a_{m1} \end{pmatrix} x_1 + \ldots + \begin{pmatrix} a_{1n} \\ \ldots \\ a_{mn} \end{pmatrix} x_n = \begin{pmatrix} \sum_{j=1}^n a_{1j} x_j \\ \ldots \\ \sum_{j=1}^n a_{mj} x_j \end{pmatrix}$$

The *multiplication of a row vector* $\boldsymbol{y}^\top$ $(\boldsymbol{y} \in \mathbb{R}^m)$ *by a matrix* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ ("left multiplication") results in a n-dimensional row vector and is defined by

$$\begin{aligned} \boldsymbol{y}^\top \boldsymbol{A} &= (\boldsymbol{y}^\top \boldsymbol{A}_1, \ldots, \boldsymbol{y}^\top \boldsymbol{A}_n) = \left((y_1, \ldots, y_m) \begin{pmatrix} a_{11} \\ \ldots \\ a_{m1} \end{pmatrix}, \ldots, (y_1, \ldots, y_m) \begin{pmatrix} a_{1n} \\ \ldots \\ a_{mn} \end{pmatrix} \right) \\ &= \left(\sum_{i=1}^{m} y_i a_{i1}, \ldots, \sum_{i=1}^{m} y_i a_{in} \right). \end{aligned}$$

Notice that "left multiplication" can also be regarded as a linear combination of the rows of $\boldsymbol{A}$:

$$\boldsymbol{y}^\top \boldsymbol{A} = \sum_{i=1}^{m} y_i \boldsymbol{a}^i = y_1 \left(a_{11}, \ldots, a_{1n}\right) + \ldots + y_m (a_{m1} \ldots, a_{mn}) = \left(\sum_{i=1}^{m} y_i a_{i1}, \ldots, \sum_{i=1}^{m} y_i a_{in} \right).$$

The multiplication of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times p}$ by a matrix $\boldsymbol{B} \in \mathbb{R}^{p \times n}$ results in an $m \times n$-dimensional matrix and is defined by

$$\begin{aligned} \boldsymbol{AB} &= (\boldsymbol{AB}_1, \ldots, \boldsymbol{AB}_n) = \begin{pmatrix} \boldsymbol{a}^1 \boldsymbol{B} \\ \ldots \\ \boldsymbol{a}^m \boldsymbol{B} \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{a}^1 \boldsymbol{B}_1 & \ldots & \boldsymbol{a}^1 \boldsymbol{B}_n \\ \ldots & \ldots & \ldots \\ \boldsymbol{a}^m \boldsymbol{B}_1 & \ldots & \boldsymbol{a}^m \boldsymbol{B}_n \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{p} a_{1k} b_{k1} & \ldots & \sum_{k=1}^{p} a_{1k} b_{kn} \\ \ldots & \ldots & \ldots \\ \sum_{k=1}^{p} a_{mk} b_{k1} & \ldots & \sum_{k=1}^{p} a_{mk} b_{kn} \end{pmatrix} \\ &= \left(\sum_{k=1}^{p} a_{ik} b_{kj} : i = 1, \ldots, m, j = 1, \ldots, n \right). \end{aligned}$$

Generalized Notation for Vectors and Matrices

As a generalization of the usual vector terminology, we use the notation S^I to denote the set of $|I|$-*dimensional vectors* with values in S, where the vector components are indexed by the elements of I. Instead of $\boldsymbol{x} \in S^I$, we also write $\boldsymbol{x} = (x_i \in S : i \in I)$.

For instance, if $I = \{a, b, c\}$, a vector $\boldsymbol{x} \in S^I$ has the form $\boldsymbol{x} = (x_a, x_b, x_c)^\top$ with $x_i \in S$ for $i \in I$. Note that the conventional notation S^n actually corresponds to $S^n = S^I$ with $I = \{1, \ldots, n\}$.

It is important to notice that with this general notation, the concept of a vector and the concept of a function are identical. Specifically, a vector $\boldsymbol{x} \in S^I$ defines a function $f : I \to S$, and vice-versa: (1) a vector $\boldsymbol{x} = (x_i \in S : i \in I) \in S^I$ defines the function $f : I \to S, i \mapsto x_i$, and (2) a function $f : I \to S, i \mapsto f(i)$ defines the vector $\boldsymbol{x} = (f(i) \in S : i \in I) \in S^I$.

Analogously, as a generalization of the usual matrix terminology, we use the notation $S^{I \times J}$ to denote the set of $|I| \times |J|$-*dimensional matrices* with values in S, where the rows are indexed by the elements of I and the columns are indexed by the elements of J. Instead of $\boldsymbol{A} \in S^{I \times J}$, we also write $\boldsymbol{A} = (a_{ij} \in S : i \in I, j \in J)$.

For instance, if $I = \{2, 5, 7, 8\}$ and $J = \{a, c, d\}$, a matrix $\boldsymbol{A} \in S^{I \times J}$ has the form

$$\boldsymbol{A} = \begin{pmatrix} a_{2a} & a_{2c} & a_{2d} \\ a_{5a} & a_{5c} & a_{5d} \\ a_{7a} & a_{7c} & a_{7d} \\ a_{8a} & a_{8c} & a_{8d} \end{pmatrix}$$

with $a_{ij} \in S$ for $i \in I$ and $j \in J$. Rows and columns are indexed accordingly, for instance, the "third" row has index 7 and is given by $\boldsymbol{a}^7 = (a_{7a}, a_{7c}, a_{7d})$, and the "second" column has index c and is given by $\boldsymbol{A}_c = (a_{2c}, a_{5c}, a_{7c}, a_{8c})^\top$. Note that the conventional notation $S^{m\times n}$ actually corresponds to $S^{m\times n} = S^{I\times J}$ with $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$.

It is important to notice that with this general notation, the concept of a matrix and the concept of a function with two-dimensional domain are identical. Specifically, a matrix $\boldsymbol{A} \in S^{I\times J}$ defines a function $f : I \times J \to S$, and vice-versa: (1) a matrix $\boldsymbol{A} = (a_{ij} \in S : i \in I, j \in J) \in S^{I\times J}$ defines the function $f : I \times J \to S$, $(i, j) \mapsto a_{ij}$, and (2) a function $f : I \times J \to S$, $(i, j) \mapsto f(i, j)$ defines the matrix $\boldsymbol{A} = (f(i, j) \in S : i \in I, j \in J) \in S^{I\times J}$.

When using this generalized notation for vectors and matrices, the explicit order of the elements of the index sets I and J, i.e. the order of the corresponding vector components, matrix rows, and matrix columns, is usually neglected. However, when doing matrix and vector operations, it is important to maintain consistency, and correctly match corresponding indices.

In particular, if a matrix $\boldsymbol{A} \in S^{I\times J}$ is multiplied by a column vector $\boldsymbol{x}$ from the right, $\boldsymbol{Ax} = \boldsymbol{b}$ implies $\boldsymbol{x} \in S^J$ and $\boldsymbol{b} \in S^I$, and the product is given by $\boldsymbol{Ax} = \sum_{j\in J} \boldsymbol{A}_j x_j$. Observe that in this summation, no assumption is made on the elements of J, nor on their order. Similarly, if matrix $\boldsymbol{A} \in S^{I\times J}$ is multiplied by a row vector $\boldsymbol{y}^\top$ from the left, $\boldsymbol{y}^T\boldsymbol{A} = \boldsymbol{c}^\top$ implies $\boldsymbol{y} \in S^I$ and $\boldsymbol{c} \in S^J$, and the product is given by $\boldsymbol{c}^\top = \sum_{i\in I} y_i \boldsymbol{a}^i$. Generally, if we consider two matrices $\boldsymbol{A} \in S^{I\times J}$ and $\boldsymbol{B} \in S^{K\times L}$, the product $\boldsymbol{AB} = \boldsymbol{C}$ implies $J = K$ and $\boldsymbol{C} \in S^{I\times L}$, and the product $\boldsymbol{BA} = \boldsymbol{C}$ implies $L = I$ and $\boldsymbol{C} \in S^{K\times J}$.

Furthermore, if $\boldsymbol{A} \in S^{I\times J}$ is a quadratic matrix, i.e. $|I| = |J|$, and $\boldsymbol{A}$ is non-singular, there exists the inverse matrix $\overline{\boldsymbol{A}} = \boldsymbol{A}^{-1}$ with $\boldsymbol{A}\overline{\boldsymbol{A}} = \overline{\boldsymbol{A}}\boldsymbol{A} = I^{(n)}$. Since$\boldsymbol{A}\overline{\boldsymbol{A}}$ implies that the rows of $\overline{\boldsymbol{A}}$ are indexed by J, and $\overline{\boldsymbol{A}}\boldsymbol{A}$ implies that the columns of $\overline{\boldsymbol{A}}$ are indexed by I, it follows $\overline{\boldsymbol{A}} \in S^{J\times I}$. Therefore, the rows of $\boldsymbol{A}$ correspond to the columns of $\overline{\boldsymbol{A}}$, and the columns of $\boldsymbol{A}$ correspond to the rows of $\overline{\boldsymbol{A}}$. Note that $\boldsymbol{A}\overline{\boldsymbol{A}} = I^{(n)}$ yields an identity matrix $I^{(n)} \in S^{I\times I}$, and $\overline{\boldsymbol{A}}\boldsymbol{A} = I^{(n)}$ yields an identity matrix $I^{(n)} \in S^{J\times J}$, hence, depending on the multiplication order, the identity matrix is indexed by I or by J.

We introduce an additional notation here, which is related to submatrices. Let $\boldsymbol{A} \in S^{m\times n}$ be a matrix with m rows and n columns, and denote the set of row and column indices by $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$, respectively. Let $B \subseteq I$ be a subset of row indices. $\boldsymbol{A}_B$ denotes the submatrix of $\boldsymbol{A}$ consisting of the rows of $\boldsymbol{A}$ selected by the index set B, which can be written as

$$\boldsymbol{A}_B = (\boldsymbol{a}^i : i \in B)$$

We assume by convention that the rows of $\boldsymbol{A}_B$ are arranged in the same order as in the original matrix $\boldsymbol{A}$, i.e. by increasing line numbers. For instance, let $\boldsymbol{A} \in \mathbb{R}^{5\times 3}$ be given as follows:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a}^1 \\ \boldsymbol{a}^2 \\ \boldsymbol{a}^3 \\ \boldsymbol{a}^4 \\ \boldsymbol{a}^5 \end{pmatrix} = \begin{pmatrix} 4 & 7 & 1 \\ 9 & 6 & 2 \\ 8 & 1 & 5 \\ 3 & 2 & 3 \\ 2 & 5 & 7 \end{pmatrix}$$

The submatrix $\boldsymbol{A}_B$ associated with the index set $B = \{5, 2, 4\}$ is then defined as:

$$\boldsymbol{A}_B = \begin{pmatrix} \boldsymbol{a}^2 \\ \boldsymbol{a}^4 \\ \boldsymbol{a}^5 \end{pmatrix} = \begin{pmatrix} 9 & 6 & 2 \\ 3 & 2 & 3 \\ 2 & 5 & 7 \end{pmatrix}$$

Note that $\boldsymbol{A}_B \in \mathbb{R}^{B \times J}$, i.e. the row index set is $B = \{2, 4, 5\}$, and the column index set is $J = \{1, 2, 3\}$. Thus, if we denote the submatrix $\boldsymbol{A}_B$ as $\boldsymbol{C}$, we have for instance $\boldsymbol{C}_2 = (6, 2, 5)^\top$ as the "second" column of $\boldsymbol{C} = \boldsymbol{A}_B$, and $\boldsymbol{c}^2 = (9, 6, 2)$ $(= \boldsymbol{a}^2)$ as the "first" (!) row of $\boldsymbol{C} = \boldsymbol{A}_B$.

The same notation is used for vectors. If $\boldsymbol{b} \in S^n$, $I = \{1, \ldots, n\}$, and $B \subseteq I$, the subvector $\boldsymbol{b}_B$ is defined as:

$$\boldsymbol{b}_B = (b_i : i \in B)$$

For instance, if $\boldsymbol{b} \in \mathbb{R}^5$ is given as $\boldsymbol{b} = (4, 1, 5, 2, 7)^\top$, the index set $B = \{5, 2, 4\}$ defines the subvector $\boldsymbol{b}_B = (1, 2, 7)^T$. We have $\boldsymbol{b}_B \in \mathbb{R}^B$, and if we denote the subvector $\boldsymbol{b}_B$ as $\boldsymbol{c}$, we have for instance $c_4 = 2$ $(= b_4)$ and $c_5 = 7$ $(= b_5)$.

Linearity, Rank and Miscellaneous

A vector $\boldsymbol{x} \in \mathbb{R}^n$ is a *linear combination* of a list of $k \geq 1$ vectors $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^k \in \mathbb{R}^n$ if there are numbers $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}$ such that $x = \sum_{\lambda=1}^k \lambda_i \boldsymbol{x}^i$. A list of $k \geq 1$ vectors $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^k \in \mathbb{R}^n$ is *linearly independent* if the equation $\sum_{i=1}^k \lambda_i \boldsymbol{x}^i = \boldsymbol{0}$ can only be fulfilled by setting $\lambda_1 = \lambda_2 = \ldots = \lambda_k = 0$. Otherwise the list of vectors is *linearly dependent*.

A *function* $f : V \to W$, where V and W are vector spaces over the same field, is called *linear* if for any vectors $\boldsymbol{x}^1, \boldsymbol{x}^2 \in V$ and any scalars λ_1, λ_2 the following condition holds:

$$f(\lambda_1 \boldsymbol{x}^1 + \lambda_2 \boldsymbol{x}^2) = \lambda_1 f(\boldsymbol{x}^1) + \lambda_2 f(\boldsymbol{x}^2)$$

In other words, a function is linear if and only if the function value of any linear combination of vectors is equal to the linear combination of the function values of the vectors.

It can be shown that a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is linear if and only if there exists a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ such that $f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ for any $\boldsymbol{x} \in \mathbb{R}^n$. In particular, a function $f : \mathbb{R}^n \to \mathbb{R}$ is linear if and only if there exists a vector $\boldsymbol{a} \in \mathbb{R}^n$ (i.e. there exists a "matrix" $\boldsymbol{a}^\top \in \mathbb{R}^{1 \times n}$) such that for any $\boldsymbol{x} \in \mathbb{R}^n$,we have

$$f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{x} = a_1 x_1 + \cdots + a_n x_n = \sum_{i=1}^n a_i x_i$$

The *row rank* of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is the maximal number of linearly independent rows of $\boldsymbol{A}$; the *column rank* is the maximal number of linearly independent columns of $\boldsymbol{A}$. It

can be shown that the row rank and the column rank are identical. The corresponding value is the *rank* of the matrix $\boldsymbol{A}$; it is denoted by $\operatorname{rank}(\boldsymbol{A})$. By a trivial argument, one gets $\operatorname{rank}(\boldsymbol{A}) \leq \min(m, n)$. If $\operatorname{rank}(\boldsymbol{A}) = \min(m, n)$ we say that $\boldsymbol{A}$ has *full rank*. If $\operatorname{rank}(\boldsymbol{A}) = m$ (or $\operatorname{rank}(\boldsymbol{A}) = n$), we say that $\boldsymbol{A}$ has *full row rank* (or *full column rank*, respectively).

Let $S \subseteq E$ be any subset of a finite set E. The *incidence vector* $\boldsymbol{x}^S$ of S (in E) is given by $\boldsymbol{x}^S \in \{0,1\}^E$ with

$$x_e^S = \begin{cases} 1, & e \in S \\ 0, & e \notin S \end{cases}$$

For instance, for $E = \{a, b, c, d\}$ and $S = \{b, d\}$ the incidence vector of S in E is given by $\boldsymbol{x}^S = (x_a, x_b, x_c, x_d)^\top = (0, 1, 0, 1)^\top$. On the other hand, a vector $\boldsymbol{x} \in \{0,1\}^E$ can be regarded as the incidence vector of the set $S^x = \{e \in E : x_e = 1\}$.

0.3 Graphs

0.3.1 Undirected Graphs

An *undirected graph* is a pair $G = (V, E)$, where V is a finite set and E is a set of *unordered pairs* of elements of V. We refer to the elements of V as *nodes or vertices* of G and to the elements of E as *edges* of G.

Hence, an edge $e \in E$ is an unordered pair $e = (v, w)$ of vertices $v, w \in V$; the term 'unordered pair' means that the notations (v, w) and (w, v) are regarded to be identical and refer to the same edge.

For an edge $e \in E$ with $e = (v, w)$, we denote v and w as the *extremities* of e, further v and w are said to be *adjacent* and w is said to be a neighbour of v (analogously, v is said to be a neighbour of w); we say that the edge e *connects* the vertices v and w and that the edge e is *incident* with v and w or that the vertices v and w are incident with e. An edge of the form $e = (v, v)$, i.e. with identical extremities, is denoted as a *loop*. A *simple graph* is a graph without loops. If nothing else is said, we assume that the considered graphs are simple.

The *adjacency matrix* of a graph $G = (V, E)$ is a (symmetric) matrix $\boldsymbol{A} \in \{0,1\}^{V \times V}$, with its elements are given by

$$a_{vw} = \begin{cases} 1, & (v, w) \in E \\ 0, & \text{else} \end{cases}, \qquad v, w \in V.$$

The *incidence matrix* is a matrix $\boldsymbol{A} \in \{0,1\}^{V \times E}$, with its elements are given by

$$a_{ve} = \begin{cases} 1, & v \text{ is incident with } e \\ 0, & \text{else} \end{cases}, \qquad v \in V, e \in E.$$

The *degree* $deg(v)$ of a vertex $v \in V$ is the number of edges incident with v. A vertex $v \in V$ is called *isolated* if $deg(v) = 0$, and *pendant* or a *leaf vertex* if $deg(v) = 1$. For a set of vertices $S \subseteq V$ we denote by $\delta(S)$ the set of *edges leading out of* S , i.e. the set of edges which have one extremity in S and one extremity out of S, i.e. $\delta(S) = \{(v, w) \in E : v \in S, w \in V - S\}$. Instead of $\delta(\{v\}), v \in V$, we write simply $\delta(v)$, and we have $deg(v) = |\delta(v)|$. Further, $\gamma(S)$ denotes the set of *edges lying in* S, i.e. the set of edges with both extremities in S, i.e. $\gamma(S) = \{(v, w) \in E : v, w \in S\}$. Instead of $\gamma(\{v\}), v \in V$, we write simply $\gamma(v)$.

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. If $V' = V$, then G' is called a *spanning subgraph* of G. We say that G' is the *subgraph* of G *induced by* the set of vertices V' if $E' = \gamma(V')$. The subgraph of G induced by V' is denoted by $G[V']$.

A *walk* in $G = (V, E)$ is an alternating sequence of vertices and edges of the form

$$P = (v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k)$$

where $k \geq 0$, $v_i \in V$ for $i = 0, \ldots, k$ and $e_i = (v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. If there are no misunderstandings possible, then a walk can be simply identified by the corresponding subgraph $G_P = (V_P, E_P)$ with $V_P = \{v_0, v_1, \ldots, v_k\}$ and $E_P = \{e_1, e_2, \ldots, e_k\}$, or just by the set of vertices V_P or the set of edges E_P. We say that P connects the vertices v_0 and v_k and that P leads from v_0 to v_k or from v_k to v_0. The number k of edges denotes the length of P.

The walk P is said to be *closed* if $v_0 = v_k$, else it is said to be *open*. P is called a *simple walk* if — except for the vertices v_0 and v_k — all other vertices are pairwise different, i.e. if $v_i \neq v_j$ for all vertex pairs $\{(i, j) : 0 \leq i < j \leq k\} - \{(v_0, v_k)\}$. A *path* is an open, simple walk, and a *circuit* is a closed, simple walk. A graph with no circuits is said to be *acyclic*.

Notice that a graph containing an open walk P from v_0 to v_k always contains a path P' from v_0 to v_k with $V_{P'} \subseteq V_P$ and $E_{P'} \subseteq E_P$. Analogously, the statement is true for closed walks and circuits.

A graph $G = (V, E)$ is called *connected* if every pair of vertices $v, w \in V$ is connected by a path. A *connected component* of G is a connected subgraph $G[V']$ of G induced by $V' \neq \emptyset$ which is maximal with regard to the set of vertices, i.e. $G[V']$ is connected and there exists no connected induced subgraph $G[V'']$ with $V' \subset V''$.

A *tree* is a connected, acyclic graph. A *forest* is an acyclic graph, i.e. a graph with all its connected components being trees. A *spanning tree* in a graph G is a spanning subgraph of G which is a tree (i.e. connected and acyclic). A *connector* in a graph G is a spanning subgraph of G which is connected (hence, a spanning tree is an acyclic connector). It can be shown (cf. elementary graph theory) that in a graph G with n nodes, every spanning tree has exactly $n - 1$ edges, and every connector has at least $n - 1$ edges.

A *bipartite graph* is a graph $G = (V, E)$ such that its set of vertices can be parted into two disjoint sets $S, T \subseteq V$, $S \cap T = \emptyset$ in such a way that every edge has one extremity in S and the other extremity in T, i.e. $E \subseteq \{(v, w) : v \in S, w \in T\}$.

A graph is said to be *complete* if every pair of vertices is connected by an edge, i.e. if $E = \{(i, j) : i, j \in V, i < j\}$. Hence, we have $|E| = |V|(|V| - 1)/2$.

A *clique* is a set of vertices $V' \subseteq V$ in a graph $G = (V, E)$ if $G[V']$ is complete. A *stable set* is a set of vertices $V' \subseteq V$ in a graph $G = (V, E)$ if $G[V']$ has no edges. Hence, in a clique, every pair of vertices is adjacent; in a stable set, every pair of vertices is not adjacent. A set of edges $E' \subseteq E$ is a *matching* if there are no two edges in E' with a common extremity.

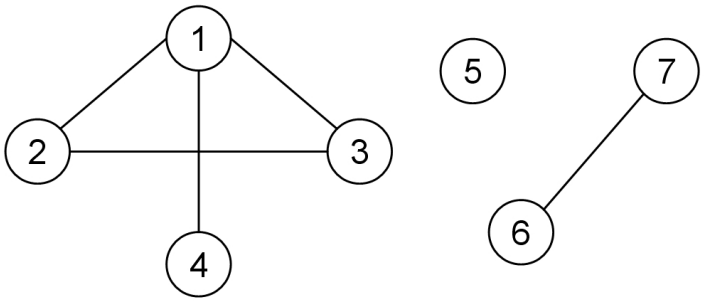


Figure 0.1: Undirected graph

Example:

The undirected graph $G = (V, E)$ depicted in Figure 0.1 has the set of vertices $V = \{1, 2, ..., 7\}$ and the set of edges $E = \{(1, 2), (1, 3), (1, 4), (2, 3), (6, 7)\}$.

The adjacency matrix of G is given by

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

and the incidence matrix is given by

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

where the edges are ordered lexicographically, i.e.

$$E = \{(1,2),(1,3),(1,4),(2,3),(6,7)\}.$$

Considering vertex degrees, we have, for example, $deg(1) = 3$, $deg(2) = 2$, $deg(6) = 1$ and $deg(5) = 0$. The set of pendant vertices of G is given by $\{4,6,7\}$, and the set of isolated vertices is given by $\{5\}$.

For the set of vertices $S = \{1,3,4\}$ we have $\delta(S) = \{(1,2),(2,3)\}$ and $\gamma(S) = \{(1,3),(1,4)\}$, for $S = \{6,7\}$ we get $\delta(S) = \emptyset$ and $\gamma(S) = \{(6,7)\}$, and for the vertex $v = 1$ it holds $\delta(v) = \{(1,2),(1,3),(1,4)\}$.

An example for a subgraph of G is given by $G' = (V', E')$ with $V' = \{1,2,3\}$ and $E' = \{(1,3),(2,3)\}$. Notice that, for example, $G' = (V', E')$ with $V' = \{1,2\}$ and $E' = \{(1,3),(2,3)\}$ is not a graph at all and hence no subgraph of G.

The subgraph $G' = (V', E')$ with $V' = \{1,2,\ldots,7\}$ and $E' = \{(1,2),(1,3),(6,7)\}$ is a spanning subgraph of G. The subgraph induced by the set of vertices $V' = \{1,2,4,5,6,7\}$ is given by $G[V'] = (V', E')$ with $E' = \{(1,2),(1,4),(6,7)\}$.

An example for a walk is

$$P_1 = (v_0, e_1, ...v_k) = (4,(4,1),1,(1,3),3,(3,2),2,(2,1),1,(1,4),4).$$

P_1 is a closed walk with length $k = 5$ which is not simple. The graph

$$P_2 = (v_0, e_1, ...v_k) = (4,(4,1),1,(1,3),3,(3,2),2)$$

is an example for a path with length $k = 3$; the graph

$$P_3 = (v_0, e_1, ...v_k) = (1,(1,3),3,(3,2),2,(2,1),1)$$

is an example for a circuit with length $k = 3$. Hence, the graph G is not acyclic.

G is not connected as, for example, vertices 5 and 6 are not connected by a path. G consists of three connected components $K_i = G[V_i], i = 1,2,3$ with $V_1 = \{1,2,3,4\}$, $V_2 = \{5\}$ and $V_3 = \{6,7\}$.

The components K_2 and K_3 are both trees; together they form a forest.The component K_1 is not a tree as it contains a circuit. The subgraph $G[V']$ induced by the set of vertices given by $V' = \{1, 2, 4\}$ is a tree. The subgraph $G' = (V', E')$ with $V' = \{1, 2, 3, 4\}$ and $E' = \{(1, 2), (1, 3), (1, 4)\}$ is a spanning tree of K_1 but no spanning tree of G.

The subgraph $G' = (V', E')$ with $V' = \{1, 2, 3, 4\}$ and $E' = \{(1, 2), (1, 3), (1, 4)\}$ is bipartite as V' can be parted in $S = \{1\}$ and $T = \{2, 3, 4\}$ such that $E' \subseteq \{(v, w) : v \in S, w \in T\}$. However, the component K_1 is not bipartite as it contains a circuit with a length being an odd number.

The component K_1 is not a complete graph but the subgraph $G[V']$ induced by $V' = \{1, 2, 3\}$ is complete. Hence, the set of vertices V' is a clique. An independent set is given by $V' = \{2, 5, 6\}$. An example for a matching in G is $E' = \{(2, 3), (1, 4), (6, 7)\}$. Notice that this matching has the maximal possible cardinality in G, i.e. there exists no matching with more than three edges in G. □

0.3.2 Directed graphs

A *directed graph* (or *digraph*) is a pair $G = (V, E)$ where V is a finite set and E is a set of *ordered pairs* of elements of V, i.e. $E \subseteq V \times V$. The elements of V are called *nodes* or *vertices* and the elements of E are called *arcs* of G.

Hence, an arc $e \in E$ is an ordered pair $e = (v, w)$ of vertices $v, w \in V$; here, 'ordered pair' means that the notations (v, w) and (w, v) are regarded to be different and denote two different arcs.

For an arc $e \in E$ with $e = (v, w)$, we denote v and w as the *extremities* of e, further $v = tail(e)$ is the *tail* and $w = head(e)$ is the *head* of e. The vertices v and w are said to be *adjacent* and w is said to be a neighbour of v (analogously, v is said to be a neighbour of w); we say that the arc e *connects* the vertices v and w and that the arc e is *incident* with v and w or that the vertices v and w are incident with e. Further, e is called to be going out of v and going into w. An arc of the form $e = (v, v)$, i.e. with identical extremities, is denoted as a *loop*. A *simple graph* is a graph without loops. If nothing else is said, we assume that the considered graphs are simple.

The *adjacency matrix* of a graph $G = (V, E)$ is a matrix $\boldsymbol{A} \in \{0, 1\}^{V \times V}$, with its elements are given by

$$a_{vw} = \begin{cases} 1, & (v, w) \in E \\ 0, & \text{else} \end{cases}, \qquad v, w \in V.$$

The *incidence matrix* is a matrix $\boldsymbol{A} \in \{-1, 0, 1\}^{V \times E}$, with its elements are given by

$$a_{ve} = \begin{cases} -1, & v = tail(e) \\ 1, & v = head(e) \\ 0, & \text{else} \end{cases}, \qquad v \in V, e \in E.$$

The *outdegree* $deg^+(v)$ of a vertex $v \in V$ is the number of arcs going out of v and the *indegree* $deg^-(v)$ is the number of arcs going into v. A vertex $v \in V$ is said to be *isolated*

if $deg^+(v) = deg^-(v) = 0$. A vertex $v \in V$ is a *source* if $deg^-(v) = 0$, and it is a *target* if $deg^+(v) = 0$. Given a set of vertices $S \subseteq V$, we denote by $\delta^+(S)$ the set of *arcs going out of* S, i.e. $\delta^+(S) = \{(v, w) \in E : v \in S, w \in V - S\}$. Analogously, we denote by $\delta^-(S)$ the set of *arcs going into* S, i.e. $\delta^-(S) = \{(v, w) \in E : v \in V - S, w \in S\}$. Further, $\gamma(S)$ denotes the set of *arcs lying in* S, i.e. the set of arcs with both extremities in S; $\gamma(S) = \{(v, w) \in E : v, w \in S\}$. Instead of $\delta^+(\{v\})$ and $\delta^-(\{v\}), v \in V$, we often write simply $\delta^+(v)$ and $\delta^-(v)$, respectively; it holds $deg^+(v) = |\delta^+(v)|$ and $deg^-(v) = |\delta^-(v)|$.

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. If $V' = V$, we say that G' is a *spanning subgraph* of G. The graph G' is the *subgraph* of G *induced by* the set of vertices V' if $E' = \gamma(V')$. The subgraph of G induced by V' is denoted by $G[V']$.

A *directed walk* in $G = (V, E)$ is an alternating sequence of vertices and arcs of the form

$$P = (v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k)$$

where $k \geq 0$, $v_i \in V$ for $i = 0, \ldots, k$ and $e_i = (v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. If there are no misunderstandings possible, then a directed walk can be simply identified by the corresponding subgraph $G_P = (V_P, E_P)$ with $V_P = \{v_0, v_1, \ldots, v_k\}$ and $E_P = \{e_1, e_2, \ldots, e_k\}$, or just by the set of vertices V_P or the set of arcs E_P. We say that P connects the vertices v_0 and v_k, and that P leads from v_0 to v_k or from v_k to v_0. The number k of arcs denotes the length of P.

The directed walk P is said to be *closed* if $v_0 = v_k$, else it is said to be *open*. P is called a *simple (directed) walk* if — except for the vertices v_0 and v_k — all other vertices are pairwise different, i.e. if $v_i \neq v_j$ for all vertex pairs $\{(i, j) : 0 \leq i < j \leq k\} - \{(v_0, v_k)\}$. A *(directed) path* is an open, simple directed walk, and a *cycle* is a closed, simple directed walk. A graph with no cycles is said to be *acyclic*.

Notice that a graph containing an open directed walk P from v_0 to v_k always contains a directed path P' from v_0 to v_k with $V_{P'} \subseteq V_P$ and $E_{P'} \subseteq E_P$. Analogously, the statement is true for closed directed walks and cycles.

A graph $G = (V, E)$ is said to be *strongly connected* if every pair of vertices $v, w \in V$ is connected by a directed path. We call G *weakly connected* if the corresponding undirected graph is connected, i.e. if every pair of vertices $v, w \in V$ is connected by a path (without considering the direction of arcs).

A *strongly connected component* of G is a strongly connected subgraph $G[V']$ of G induced by $V' \neq \emptyset$ which is maximal with regard to the set of vertices, i.e. $G[V']$ is strongly connected and there exists no strongly connected induced subgraph $G[V'']$ with $V' \subset V''$. Analogously one can define a *weakly connected component* of G.

An *arborescence* is a weakly connected, acyclic graph $G = (V, E)$ with $deg^-(v) \leq 1$ for all $v \in V$. An arborescence $G = (V, E)$ contains exactly one vertex $r \in V$ with $deg^-(r) = 0$; this vertex is called *root* of G. A *branching* is an acyclic graph $G = (V, E)$ with $deg^-(v) \leq 1$ for all $v \in V$, i. e. a graph with its weakly connected components

being arborescences. A *spanning arborescence* in a graph G is a spanning subgraph of G which is an arborescence.

A graph $G = (V, E)$ is said to be *complete* if every ordered pair of vertices is connected by an arc, i.e. $E = \{(i, j) : i, j \in V, i \neq j\}$.Hence, we have $|E| = |V|(|V| - 1)$.

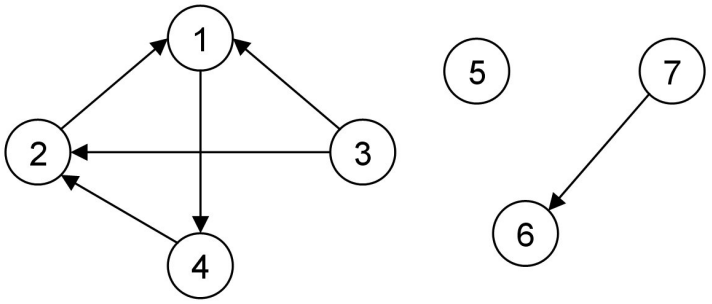


Figure 0.2: Directed graph

Example:

The directed graph $G = (V, E)$ depicted in Figure 0.2 has the set of vertices $V = \{1, 2, ..., 7\}$ and the set of arcs $E = \{(1, 4), (2, 1), (3, 1), (3, 2), (4, 2), (7, 6)\}$.

The adjacency matrix of G is given by

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

and the incidence matrix is given by

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

where the arcs are ordered lexicographically, i.e.

$$E = \{(1, 4), (2, 1), (3, 1), (3, 2), (4, 2), (7, 6)\}.$$

Considering vertex degrees in G, we have, for example, $deg^+(1) = 1$, $deg^-(1) = 2$, $deg^+(3) = 2$, $deg^-(3) = 0$ and $deg^+(5) = deg^-(5) = 0$. The sets of sources and targets of G are given by $\{3, 5, 7\}$ and $\{5, 6\}$, respectively. The set of isolated vertices is $\{5\}$.

Considering the set of vertices $S = \{1, 3, 4\}$, we have $\delta^+(S) = \{(3, 2), (4, 2)\}$, $\delta^-(S) = \{(2, 1)\}$ and $\gamma(S) = \{(1, 4), (3, 1)\}$; for $S = \{6, 7\}$ it holds $\delta^+(S) = \delta^-(S) = \emptyset$ and $\gamma(S) = \{(7, 6)\}$; for the vertex $v = 1$ we get $\delta^+(v) = \{((1, 4)\}$ and $\delta^-(v) = \{(2, 1), (3, 1)\}$.

An example for a subgraph of G is given by $G' = (V', E')$ with $V' = \{1, 2, 3\}$ and $E' = \{(3, 1), (3, 2)\}$. Notice that, for example, $G' = (V', E')$ with $V' = \{1, 2\}$ and $E' = \{(3, 1), (3, 2)\}$ is not a graph at all and, hence, it is no subgraph of G.

The subgraph $G' = (V', E')$ with $V' = \{1, 2, \ldots, 7\}$ and $E' = \{(2, 1), (3, 1), (7, 6)\}$ is a spanning subgraph of G. The subgraph induced by the set of vertices $V' = \{1, 2, 4, 5, 6, 7\}$ is given by $G[V'] = (V', E')$ with $E' = \{(1, 4), (2, 1), (4, 2), (7, 6)\}$.

An example of a directed walk in G is given by

$$P_1 = (v_0, e_1, ...v_k) = (3, (3, 2), 2, (2, 1), 1, (1, 4), 4, (4, 2), 2).$$

P_1 is an open, non-simple directed walk from $v_0 = 3$ to $v_k = 2$ with length $k = 4$. The graph

$$P_2 = (v_0, e_1, ...v_k) = (3, (3, 2), 2, (2, 1), 1, (1, 4), 4)$$

is a directed path with length $k = 3$ and

$$P_3 = (v_0, e_1, ...v_k) = (1, (1, 4), 4, (4, 2), 2, (2, 1), 1)$$

is a cycle with length $k = 3$. Hence, the graph G is not acyclic.

The graph G is neither weakly nor strong connected as, for example, vertices 5 and 6 are not connected in any way. G consists of 5 strongly connected components $K_i = G[V_i], i = 1, \ldots, 5$ with $V_1 = \{1, 2, 4\}$, $V_2 = \{3\}$, $V_3 = \{5\}$, $V_4 = \{6\}$ and $V_5 = \{7\}$. Further, G consists of 3 weakly connected components $H_i = G[V_i], i = 1, 2, 3$ with $V_1 = \{1, 2, 3, 4\}$, $V_2 = \{5\}$ and $V_3 = \{6, 7\}$.

The subgraph $G' = (V', E')$ with $V' = \{1, 2, 3, 4\}$ and $E' = \{(1, 4), (3, 1), (3, 2)\}$ is an arborescence with a root in vertex 3.

The graph G contains no complete subgraph with more than one vertex. □

1 Introduction

This course presents an introduction to the fundamentals of mathematical optimization. We will review a selection of elementary optimization techniques and discuss the application of such techniques in technical and operational environment using various examples. In particular, we will address following subjects: *Part 1:* Linear Optimization, Integer Linear Optimization, *Part 2:* Elements of Nonlinear Optimization, Optimization in Graphs, Heuristics and Metaheuristics.

The scope of the scientific field of mathematical optimization (also known as *Operations Research*) is enormous, hence this course has to be understood as an introduction presenting a survey of various problem settings and solution approaches on the one hand, and giving a fundamental mathematical understanding of various optimization methods often used in real-life problems on the other hand. For further deepened studies we refer to specialist literature. A choice of recommended textbooks is given in the list of references.

1.1 Decision Problems

In the today's business practice (for instance, in development and production of goods, in logistics or in the service sector) the decision maker is often confronted with questions and decision problems of such high complexity that they cannot be tackled just by using common sense, intuition and experience.

One can greatly benefit from quantitative (mathematical) models giving *Decision Support* in such complex decision situations. The fundamental purposes of such model are to describe the decision freedom exactly and explicitly, to estimate and indicate the consequences of various possible decisions and, finally, to give hints which decisions are *feasible* in the given the context and which decisions are *optimal* regarding the goal settings.

Different fundamental notions of the decision theory are connected with the concept of the *decision problem*. In the following we will shortly describe them.

The existence of *several* possible *action alternatives* (we call them shortly *alternatives*) the *decision maker* has to choose from is an assumption for having a *decision problem*. There is no decision problem present if there is only one alternative to choose from. (Compare the anecdotal statement made by Henry Ford at the beginning of 20th century

regarding the production of the first car "Ford Model T" manufactured in series: "Any customer can have a car painted any color that he wants so long as it is black.")

Every alternative has certain *consequences* which can be either *deterministic* (i.e. foreseeable with certainty) or *stochastic* (i.e. influenced by chance, not foreseeable with certainty).

In this course we consider decision situations with deterministic consequences only.

The decision maker determines in a decision situation a *choice of consequences* (out of all possible consequences) which seem relevant to him and which he wants to include into the decision process. Hence, alternatives shall be evaluated regarding the *considered consequences*. (Notice: If the consequences of all alternatives in a decision situation are all the same there is no decision problem to solve.)

Two principles will be applied when evaluating consequences: *Satisfaction* and *Optimization*. Satisfaction of a consequence means that the consequence has to fulfill certain *conditions (or restrictions)* in order to be *feasible*. Optimization means that the considered *consequence (objective function)* achieves the *best possible value*, i.e. it is optimal among all possible consequences.

From modeling point of view, two cases can be distinguished when considering the connection between an alternative and its consequences: (i) *analytically* computable consequences and (ii) not analytically computable consequences, i.e. consequences which have to be determined *algorithmically*.

In case (i) the dependency of the consequence on the alternative is "relatively easy", i.e. it can be expressed by an "explicitly displayable mathematical formula".

In case (ii) there is no analytical formulation of the connection because of its complexity (or the formulation is not known). In order to determine the consequence one needs to apply *algorithmic techniques* in form of a "program" (algorithm) which takes the input (alternative) and calculates the corresponding output (consequence). In fact many decision situations in real life problems contain consequences depending on alternatives in a very complex way such that the connection between alternatives and consequences cannot be expressed analytically.

Example 1.1: Strategic capacity planning in Airport Ground Handling

Let us illustrate the introduced notions on a real life problem (coming from a consulting project).

A international company acting in Airport Ground Handling services has to analyze the strategic planning of its Ground Support Equipment for a certain airport (for example push-back tractors, mobile staircases, mobile charging units, etc.). The company invests every year a double-digit million amount into new equipment.

The main decision problem in every equipment category is to determine how many units (of which types) have to be at disposal in order to cover the current or predicted

capacity demand in a cost-efficient way. The capacity demand is determined mainly by the number and type of aircraft movements.

When considering a single equipment category a *decision alternative* is specified (roughly speaking) by the number of units of every possible type to be purchased (or removed) in the considered planning period.

Two essential *consequences* to be taken into account are (i) resulting costs and (ii) the resulting capacity.

(i): The costs can be derived easily, hence, they can be determined *analytically* (i.e. by "mathematical formulas"). They are calculated regarding to a time period (for instance, a month) and they consist of fixed costs (amortization, maintenance,...) and variable (dependent on operation) costs (for example staff costs).

(ii): Typically, it is a complex task to determine the available capacity of the present equipment. The connection between the equipment stock (alternative) and the resulting capacity (consequence) rarely can be stated in an analytic way. In general, the whole operative control (which equipment, when used?) has to be analyzed in detail to obtain a capacity estimate. Further, one has to develop optimized control strategies in order to achieve the best possible utilization of the available equipment. Hence, because of this complexity the consequence "capacity" can be determined only by using *algorithmic techniques*.

When considering *satisfaction* and *optimization* of the above consequences one can say that – typically – the consequence "capacity" has to be satisfied (i.e. the alternative has to be selected in such a way that the capacity demand can be surely covered) and the consequence "costs" has to be optimized (i.e. minimized). □

1.2 Introductory Examples

In this section, we present a selection of decision or optimization problems, in form of short, practice-oriented case studies.

A business consultant or optimization specialist, in particular in the area of quantitative, model-based decision support, is typically confronted at project begin with the situation, that problems and concerns of business customers are only partially captured, and hence expressed and communicated in a very unstructured and vague form. In a first crucial project phase, a clear problem description has to be elaborated, i.e. a precisely specified formal decision problem has to be formulated. As we have just seen, this involves an exact and explicit description of the decision alternatives, the consequences to be considered, and the criteria for satisfication and optimization.

The transition from an initial, unstructured problem description to a formal decision problem with an associated quantitative model is often the most challenging phase of a business consulting project, and in many cases decisive for success or failure. This phase of *problem analysis* and *problem formulation* requires, besides methodological knowledge,

a variety of additional skills: professional experience, intuition, interest in the business specific activities and issues of the customers, communication skills, attention to detail, abstraction capability, imagination and creativity.

These capabilities and corresponding professional experience can only marginally be trained in academic lectures, unfortunately. In discrepancy with the importance of the initial problem formulation phase, due to the lack of time, lectures generally can only address small, well-structured problems that are already prepared and edited in every detail, where the phase of problem analysis and formulation is completely missing.

When composing the following examples, we tried to select practice-oriented problems as close to the real-world as possible, and to present them in such a way that, on the one hand, the tasks are fully specified, and on the other hand, there is still some remaining need for problem analysis in order to reach a formal problem description.

The problems are also selected according to methodological aspects, and we shall refer to them again during the course in order to explain or illustrate certain modeling or optimization techniques.

These examples shall give an idea on the broad variety of applications of Operations Research, and illustrate the methodological and intellectual challenges involved with the solution of optimization problems. The aim is that at the end of the course, all examples are thoroughly analyzed, modeled, implemented and solved. The experience gained shall enable the students to identify, elaborate, and solve problems of similar difficulty in their future professional work.

1.2.1 Problem 1: Frequency Assignment in Mobile Networks

This example deals – in a simplified form – with an optimization problem arising in the construction of mobile networks.

This problem was issued by the Swiss Operations Research Society SVOR/ASRO as a competition task for the Optimization Contest 2004 (see www.svor.ch). This contest takes place every year and should attract young people from secondary schools to Operations Research.

A new telecommunication provider wants to place mobile network transmitters in the 20 largest Swiss cities. Every transmitter has to be assigned a transmitting frequency. The frequencies of neighboring transmitters have to differ sufficiently in order to avoid interference. The bandwidth of the available frequencies is limited and the costs for using mobile network frequencies are very high, that is why one wants to minimize the number of frequencies needed. (Nowadays, mobile network frequencies are auctioned. An US auction of an frequency package of 22 MHz in 2008 started at a minimum price of 4.6 billions US dollar.)

Assuming that the used frequencies lie in a connected frequency band, our goal is that the bandwidth of the used frequency band (i.e. the difference between the highest and lowest used frequency) is as small as possible.

Figure 1.1: Mobile network transmitter

1	Zürich	8	Luzern	15	Chur
2	Genf	9	Biel	16	Neuchâtel
3	Basel	10	Thun	17	Vernier
4	Bern	11	Köniz	18	Uster
5	Lausanne	12	La Chaux-de-Fonds	19	Sion
6	Winterthur	13	Schaffhausen	20	Emmen
7	St. Gallen	14	Fribourg		

Table 1.1: List of 20 largest Swiss cities (sorted in descending order)

A basis for planning is a table with air distances (in km) between the 20 largest Swiss cities. Let the available frequencies be given by $f_r = f_0 + (r-1) \cdot \Delta f$, $r = 1, 2, 3, \ldots$, where f_0 is the lower end (in MHz) of the available frequency band and Δf is the distance (in MHz) between two successive frequencies.

It is assumed that the minimal necessary difference between frequencies of two transmitters falls proportionally to the distance of the transmitters: for a distance in an interval $[5k, 5(k+1)[$, the needed frequency difference is given by $(10-k) \cdot \Delta f$ for $k = 0, 1, \ldots, 9$. That means, for instance, that in the range of $0-5$ km, a minimal frequency distance of $10 \cdot \Delta f$ is needed; in the range of $45-50$ km, a minimal distance of Δf is needed; from 50 km one can neglect inferences.

1.2.2 Problem 2: Product Mixture in an Oil Refinery

An oil refinery produces four types of raw fuel: Alkylate, Catalytic-Cracked, Straight-Run and Isopentane. Two important technical properties of raw fuel are given by the performance number PN (a measure for knock resistance) and the vapor pressure RVP. Table 1.3 shows these measures together with the daily production (in barrels, 1 barrel is ca. 159 liters).

The four raw fuel types can be either sold directly at the price of 122.81 US dollar

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	224	74	96	174	19	64	40	101	98	99	133	37	123	96	129	227	14	156	39
2	224	0	186	129	50	244	282	191	133	129	126	113	252	102	270	106	3	234	94	190
3	74	186	0	69	137	85	135	78	50	89	72	76	80	90	166	80	187	88	148	75
4	96	129	69	0	78	115	156	67	26	25	4	49	123	27	160	39	131	107	80	65
5	174	50	137	78	0	193	232	141	83	81	75	67	201	50	225	57	50	184	65	140
6	19	244	85	115	193	0	50	59	119	117	118	150	23	142	95	147	246	17	175	58
7	64	282	135	156	232	50	0	91	164	152	159	196	63	183	65	192	284	50	203	92
8	40	191	78	67	141	59	91	0	81	61	70	112	76	92	96	105	193	45	116	3
9	101	133	50	26	83	119	164	81	0	51	27	32	122	38	177	29	135	114	101	79
10	98	129	89	25	81	117	152	61	51	0	24	72	130	37	146	59	132	106	62	61
11	99	126	72	4	75	118	159	70	27	24	0	49	127	24	162	38	128	110	77	68
12	133	113	76	49	67	150	196	112	32	72	49	0	152	41	207	14	114	145	105	110
13	37	252	80	123	201	23	63	76	122	130	127	152	0	150	116	151	254	40	190	75
14	123	102	90	27	50	142	183	92	38	37	24	41	150	0	182	27	104	134	66	91
15	96	270	166	160	225	95	65	96	177	146	162	207	116	182	0	199	272	83	180	99
16	129	106	80	39	57	147	192	105	29	59	38	14	151	27	199	0	107	141	91	103
17	227	3	187	131	50	246	284	193	135	132	128	114	254	104	272	107	0	236	97	192
18	14	234	88	107	184	17	50	45	114	106	110	145	40	134	83	141	236	0	162	45
19	156	94	148	80	65	175	203	116	101	62	77	105	190	66	180	91	97	162	0	117
20	39	190	75	65	140	58	92	3	79	61	68	110	75	91	99	103	192	45	117	0

Table 1.2: Air distances (km) between largest 20 Swiss cities

	PN	RVP	Production
Alkylate	107	5	3814
Catalytic-Cracked	93	8	2666
Straight-Run	87	4	4016
Isopentane	108	21	1300

Table 1.3: Properties and production (in barrel/day) of the raw fuel types

per barrel or they can be mixed in order to obtain two types of jet fuel (Avgas A and Avgas B). Jet fuel mixtures have to fulfill certain requirements regarding PN (minimum guaranteed value) and RVP (maximum guaranteed value) due to quality standards. In Table 1.4 one can find the above mentioned conditions and selling prices (in US dollar per barrel) for both jet fuel types.

	min. PN	max. RVP	Price
Avgas A	100	7	175.04
Avgas B	91	7	152.68

Table 1.4: Requirements and prices (in US dollar per barrel) for both jet fuel mixtures

The values of PN and RVP for a mixture of raw fuels are given by the weighted average of the corresponding values for the individual components.

The task is to find a production plan for this oil refinery yielding maximal possible profit, assuming that all fuels offered can be sold.

Figure 1.2: Oil refinery

1.2.3 Problem 3: Vehicle Dispatching in a Car Rental Company

A car rental company in Crete operates 10 branches located throughout the island. Customers can collect a car at a certain branch and return it at any of the branches.

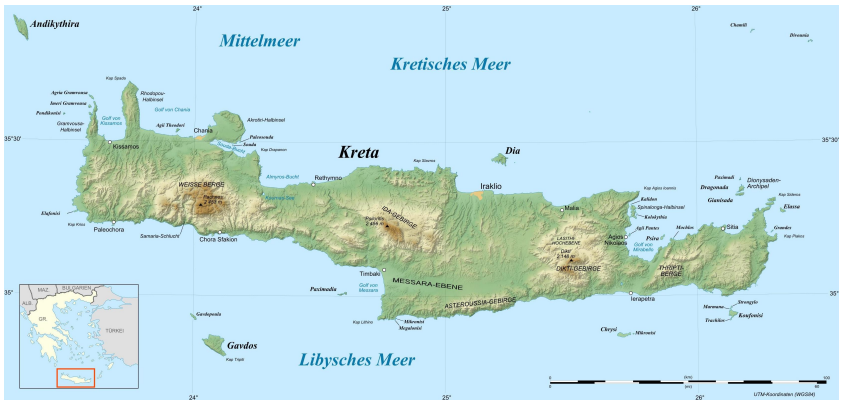


Figure 1.3: Island of Crete

The dispatcher has the task to plan every evening the overnight transfers of the vehicles between the branches in order to have the requested numbers of cars at every branch on the next morning. The transfers have to be planned in such a way that the total number of traveled kilometers is as small as possible.

The following information is available to the dispatcher: the number of cars at every branch in the evening and the number of requested cars at every branch on the next

morning. For simplicity, let us assume that all vehicles are of the same type. Further, the dispatcher has a table of distances (in km) between branches at his disposal.

Branch:	1	2	3	4	5	6	7	8	9	10
Current stock:	18	13	8	20	25	13	31	12	8	29
Requested stock:	9	22	18	30	14	8	12	15	24	21

Table 1.5: Available and requested vehicles at locations

Branch:	1	2	3	4	5	6	7	8	9	10
1	0	146	153	218	186	140	104	196	162	226
2	146	0	164	72	40	60	145	121	190	170
3	153	164	0	92	124	138	173	43	142	92
4	218	72	92	0	32	132	176	49	118	98
5	186	40	124	32	0	100	149	81	150	130
6	140	60	138	132	100	0	143	179	143	159
7	104	145	173	176	149	143	0	157	58	122
8	196	121	43	49	81	179	157	0	99	49
9	162	190	142	118	150	143	58	99	0	103
10	226	170	92	98	130	159	122	49	103	0

Table 1.6: Transfer distances (in km) between locations

1.2.4 Problem 4: Shift Planning in a Department Store

The sales staff in a department store are working in a shift system. During a business reorganization the management are planning to re-engineer the shift system. According to the new regulations, every shift has to include seven or eight working hours. For eight-hours-shifts, one or two hours of rest time have to be granted after four hours of working. For seven-hours-shifts, one hour of rest time has to be granted after three or four hours of working.

The gross costs per working hour are determined as follows: for an eight-hours-shift with one hour rest – 60 francs; for an eight-hours-shift with two hours rest – 70 francs; for a seven-hours-shift – 65 francs.

The opening hours of the department store are from 8:00 till 21:00. The demand for sales staff (number of persons per one hour interval) has been determined basing on customer numbers depending on time and can be found in Table 1.7.

Our task is to find a shift plan covering staff demand with minimal costs.

1.2.5 Problem 5: Design of a Regional Optical Fiber Network

The race for fast construction of optical fiber networks ("data highways" of the future) is running worldwide. Here, we consider the development of a small rural region consisting

Figure 1.4: Employees in a department store

Hour:	08:00	09:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00
Demand:	25	23	20	16	24	31	27	14	14	22	24	25	21

Table 1.7: Demand for sales staff per one hour intervals

of 12 locations. The network shall be constructed in such a way that every location is connected to any other location directly or indirectly (via another location).

All cables have to be underground, and no existing cable trenches can be used. Hence, the development will cause huge costs. Essentially, these costs can be seen as proportional to the length of the laid cables.

Our task is to find a network with minimal total length of laid cables. The distances (in km) between the locations are given in Table 1.8.

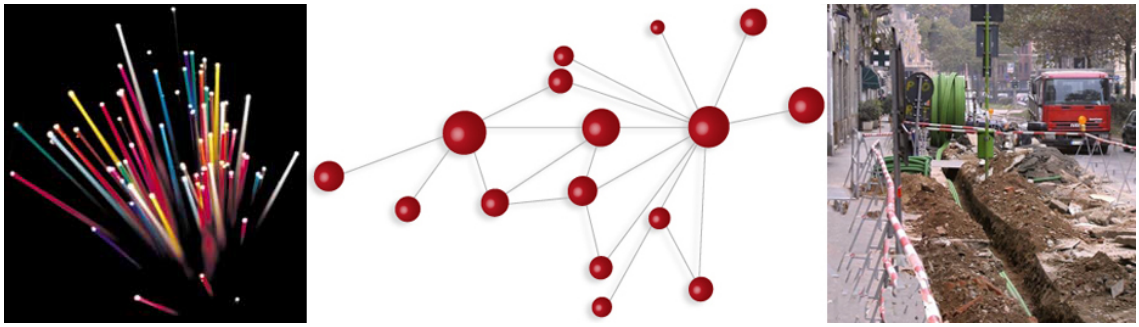


Figure 1.5: Optical fiber network

Place:	1	2	3	4	5	6	7	8	9	10	11	12
1	0	24	18	27	12	9	30	11	25	32	23	26
2	24	0	13	25	23	30	24	27	18	23	12	17
3	18	13	0	19	30	27	22	27	16	14	23	20
4	27	25	19	0	24	26	15	18	23	33	26	21
5	12	23	30	24	0	21	30	20	18	23	11	28
6	9	30	27	26	21	0	34	20	30	29	18	26
7	30	24	22	15	30	34	0	33	34	24	23	20
8	11	27	27	18	20	20	33	0	31	41	31	19
9	25	18	16	23	18	30	34	31	0	30	17	19
10	32	23	14	33	23	29	24	41	30	0	34	28
11	23	12	23	26	11	18	23	31	17	34	0	17
12	26	17	20	21	28	26	20	19	19	28	17	0

Table 1.8: Distances (in km) between locations

1.2.6 Problem 6: Sudoku

Sudoku puzzles can be solved easily when using the methods of mathematical programming. How?

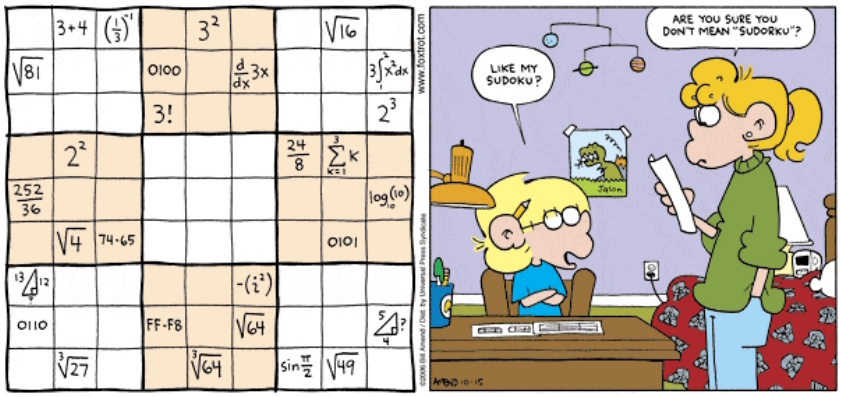


Figure 1.6: Crazy Sudoku

2 Mathematical Models

2.1 Descriptive Models and Optimization Models

In quantitative decision support, two fundamental types of mathematical models can be distinguished: (i) *Descriptive Models*, and (ii) *Optimization Models*. These two types of models and their different functionality shall briefly be discussed and illustrated in the following.

2.1.1 Descriptive Models

Descriptive models serve to determine the consequences of decision alternatives. Their functionality is such that the user specifies an alternative as the *input*, and the model calculates the corresponding consequences as the *output*. The selection of the consequences to be considered depends, as already mentioned, on the intended purpose of the model, and is part of the model development process. In short, descriptive models respond to the question "What If ?", and are also called *evaluation models* or *What-If models*.

For deterministic models, a *functional relation* between alternatives and their consequences is assumed, meaning that a certain input (a certain alternative) always yields the same consequences as output.

Mathematically, alternatives are represented by a number of *decision variables*, in short *variables*. Variables are model elements whose value can be chosen arbitrarily by the user. In contrast, there are other model elements whose value is predefined and unchangeable. They constitute the so-called data frame and are called *parameters* (with accent on the second "a", parámeter). *Consequences* are expressed mathematically as (explicitly or implicitly defined) *functions*.

Variables, parameters and consequences in a mathematical model are often associated with objects (entities) of a certain type. A collection of objects of a certain type is mathematically represented as a *set*, and association of objects to elements of the set is often expressed by means of an *index*.

In summary, the formal definition of a descriptive model comprises the following mathematical elements:

Elements of a Descriptive Model:

Sets: Sets represent the different types of objects in the model. Objects are typically represented by numbers $0, 1, 2, \ldots$. Sets of objects are usually denoted by single uppercase letters, e.g. I or J, with $I, J \in \mathbb{Z}$. The elements of a set, or their indices, are normally denoted by single lower case letters, e.g. $i \in I$ or $j \in I$.

Parameters: Parameters correspond to the predefined values in the model. In general, they are specified as one ore more sequences of indexed names, e.g. $p_1, p_2, \ldots, p_r \in \mathbb{R}$. Parameters are often combined into vectors, and then written, for instance, as $\boldsymbol{p} = (p_1, p_2, \ldots, p_r)^\top \in \mathbb{R}^r$.

Variables: Variables correspond to the modifiable values in the model. In general, they are specified as one ore more sequences of indexed names, e.g. $x_1, x_2, \ldots, x_n \in \mathbb{R}$. Variables are often combined into vectors, and then written, for instance, as $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^\top \in \mathbb{R}^n$. A decision alternative in this representation corresponds to a vector $\boldsymbol{x} \in \mathbb{R}^n$, i.e. a point in the n-dimensional real vector space $\mathbb{R}^n$.
If the possible values for a variable x_j are restricted to a certain subset $S \subseteq \mathbb{R}$, we write $x_j \in S$ and denote S as the *domain* of x_j.

Consequences: It is the task of a descriptive model to calculate the consequences to be considered, as a function of the variables $\boldsymbol{x}$ and the parameters $\boldsymbol{p}$. The *input* of the model corresponds to the variables $\boldsymbol{x}$ and the parameters $\boldsymbol{p}$, and the *output* corresponds to the calculated consequences. The consequences are defined by $k_0 = f_0(\boldsymbol{x}, \boldsymbol{p})$, $k_1 = f_1(\boldsymbol{x}, \boldsymbol{p})$, ..., $k_m = f_m(\boldsymbol{x}, \boldsymbol{p})$, where $f_i : \mathbb{R}^{n+r} \to \mathbb{R}$, $i = 0, 1, \ldots, m$, is an explicitly or implicitly defined function calculating the i-th consequence.

Figure 2.1 schematically shows the functionality of a descriptive model.

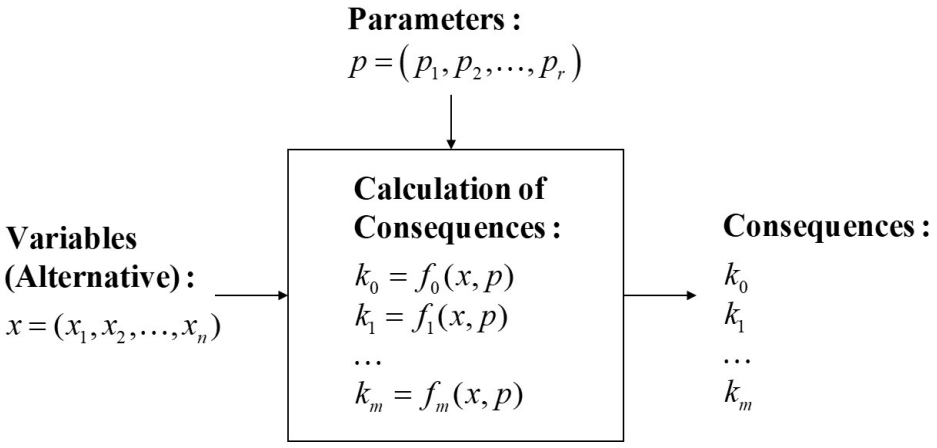


Figure 2.1: Descriptive model: Schematic functionality

Example 2.1: Formulation of a Descriptive Model

Taking the introductory example "Vehicle Dispatching in a Car Rental Company", we elaborate a formal specification of an appropriate descriptive model. A possible formulation is as follows:

Sets:

I Set of locations, $I = \{1, 2, \ldots, n\}$.

Parameters:

a_i Number of available vehicles at location i, $i \in I$.
b_j Number of requested vehicles at location j, $j \in I$.
c_{ij} Distance (km) from location i to location j, $i, j \in I$.

Variables:

x_{ij} Number of vehicles transferred from location i to location j, $i, j \in I$.

Consequences:

k_0 Total distance (km) of all transfers.
k_i^{Out} Number of vehicles transferred out of location i, $i \in I$.
k_j^{In} Number of vehicles transferred into location j, $j \in I$.

Model:

$$k_0 = \sum_{i \in I} \sum_{j \in I} c_{ij} x_{ij}$$

$$k_i^{Out} = \sum_{j \in I} x_{ij}, \qquad i \in I$$

$$k_j^{In} = \sum_{i \in I} x_{ij}, \qquad j \in I$$

Of course, names for parameters, variables, and consequences can basically be chosen as you wish. However, a deliberate and consistent naming corresponding to the usual conventions is of crucial importance for the comprehensibility of complex models. Novices in model formulation are recommended to refer to the notation in good textbooks and articles.

Finally, we would like to point out that the proposed model formulation also admits vehicle transfers between one and the same location. What does that mean? Why is the formulation still correct? How could the formulation be changed to avoid such transfers? □

Implementation

In order to become usable, a formulated descriptive model has to be implemented in an appropriate programming environment. Depending on the area of application, this can be a general-purpose programming language such as Python, Java, C#, or C++, a vector-oriented or mathematical programming language such as MatLab, R, or Mathematica, an algebraic modeling language such as GAMS, OPL, AIMMS, or LPL, or a special purpose tool such as a Discrete Event Simulation system.

In addition, an obvious and easy tool for implementing basic descriptive models are *spreadsheets*. In fact, most spreadsheet models can be viewed as descriptive models from a model-theoretic viewpoint.

A *spreadsheet model* typically comprises three types of cells: (i) *fixed cells* containing a predefined unmodifiable value, (ii) *input cells* whose value is entered by the user (cf. specification of an alternative), and (iii) *formula cells* whose value is automatically computed by a formula that depends on the fixed cells and the input cells. The analogy with descriptive models is apparent: (i) fixed cells correspond to parameters, (ii) input cells correspond to variables, and (iii) formula cells correspond to consequences.

Application

After implementation, a descriptive model can be used as follows: The user specifies a certain alternative by defining the values of the decision variables, and the model then calculates the resulting consequences.

These consequences can be used for testing whether the proposed alternative is *valid* (or *feasible*) with respect to the problem *constraints* (*"satisfication"*), and whether it is *optimal* (or at least sufficiently good) with regard to the *objective* function (*"optimization"*).

In the above example, an alternative is specified by fixing the values of the variables x_{ij}, $i, j \in I$. The model then calculates the three types of consequences defined.

Consequences k_i^{Out} and k_j^{In} are used for testing feasibility. The following conditions must be satisfied:

$$\begin{aligned} k_i^{Out} &\leq a_i, & i \in I \\ k_j^{In} &\geq b_j, & j \in I \end{aligned}$$

Explain the meaning of these inequalities.

A further important consequence to be considered is the total transfer distance k_0. According to the problem formulation, the goal is to reduce this distance as much as possible (optimization). This can be written as

$$k_0 = \min!$$

In summary, the task of the user of a descriptive model consists in finding an alternative that is on the one hand *feasible*, and on the other hand *minimizes* total transfer distance. The descriptive model supports the user by calculating the resulting consequences for the alternatives proposed.

It is important to note that both satisfication and optimization are *not part* of the descriptive model. This can also be seen in the example, from the fact that the parameters a_i and b_j do not appear in the model formulation.

Hence, a descriptive model does not make proposals for possible alternatives, and does not take into account satisfication and optimization. Therefore, it provides only limited support for constructing useful alternatives, neither with respect to feasibility nor with respect to optimality.

In the example, we can see that the number of possible alternatives may be enormous, even if the number of locations is small, so that it is almost impossible to find feasible solutions near to the optimum, only by means of arbitrarily checking out different alternatives.

2.1.2 Optimization Models

The purpose of optimization models is to find an *optimal solution* (*alternative*) for a decision problem, among the set of all *feasible solutions* (*alternatives*).

Feasibility of an alternative (cf. satisfication) is defined by formulating *constraints* (German: *Restriktionen, Nebenbedingungen*) for certain consequences that have to be satisfied. Typically, constraints are formulated as inequalities (or equalities), i.e.

$$k_i \leq b_i, \qquad i = 1, 2, \ldots, n$$

where k_i is the i-th consequence to be considered, and b_i is a parameter (a given number). The set of all feasible solutions, i.e. the set of alternatives satisfying all constraints, is called the *solution space* (German: *Lösungsraum*).

Optimality of an alternative (cf. optimization) refers to a selected consequence k_0, called the *objective function* or *objective* (German: *Zielfunktion*), whose value has to be maximized or minimized among all feasible solutions in the solution space, i.e.

$$k_0 = \min \ / \ \max !$$

In short, optimization models respond to the question "What's Best ?", and are also called *prescriptive models* (in contrast to descriptive models). Figure 2.2 schematically shows the functionality of an optimization model.

Optimization models are based on descriptive models in the sense that they include calculation of consequences, too. They comprise as additional model elements (i) *constraints* defining the solution space, and (ii) an *objective function* to be optimized.

In summary, the formal definition of an optimization model comprises the following mathematical elements:

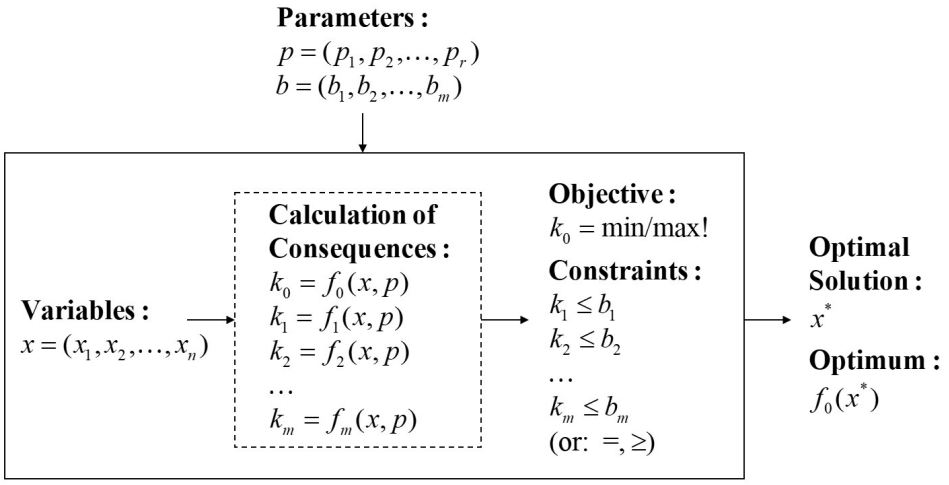


Figure 2.2: Optimization model: Schematic functionality

Elements of an Optimization Model:

Sets: Sets represent the different types of objects in the model, e.g. $I = \{1, 2, \ldots, n\}$ (see descriptive model).

Parameters: Parameters specify the predefined values in the model and can be represented as vectors, e.g. $\boldsymbol{p} = (p_1, p_2, \ldots, p_r)^\top \in \mathbb{R}^r$ (see descriptive model).

Variables: Variables specify the modifiable values in the model and can be represented as vectors, e.g. $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^\top \in \mathbb{R}^n$ (see descriptive model).

Constraints: Constraints define conditions on the consequences and are typically represented as mathematical inequalities or equalities ($\le, =, \ge$). The left-hand side (LHS) of the relation corresponds to the considered consequence $k_i = f_i(\boldsymbol{x}, \boldsymbol{p})$, $i \in I$, and the right-hand side (RHS) corresponds to the limiting value b_i, e.g.

$$f_i(\boldsymbol{x}, \boldsymbol{p}) \le b_i \qquad\qquad i \in I$$

Objective Function: The objective function corresponds to a selected consequence $k = f(x, p)$ to be minimized or maximized:

$$\min / \max \; f(x, p)$$

Example 2.2: Formulation of an Optimization Model

Taking the introductory example "Vehicle Dispatching in a Car Rental Company", we elaborate a formal specification of an appropriate optimization model. A possible formulation is as follows (cf. descriptive model):

Sets:

I Set of locations, $I = \{1, 2, \ldots, n\}$.

Parameters:

a_i Number of available vehicles at location i, $i \in I$.
b_j Number of requested vehicles at location j, $j \in I$.
c_{ij} Distance (km) from location i to location j, $i, j \in I$.

Variables:

x_{ij} Number of vehicles transferred from location i to location j, $i, j \in I$.

Constraints:

$$\begin{aligned} \sum_{j \in I} x_{ij} &\leq a_i, & i &\in I \\ \sum_{i \in I} x_{ij} &\geq b_j, & j &\in I \\ x_{ij} &\geq 0 & i, j &\in I \end{aligned}$$

Objective Function:

$$\min \sum_{i \in I} \sum_{j \in I} c_{ij} x_{ij}$$

□

2.2 General Optimization Problem

2.2.1 Problem Formulation

A general optimization problem (or optimization model) Π can be described as follows:

$$\Pi: \ \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}, \quad \text{where } S \in \mathbb{R}^n$$

The components of the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^\top \in \mathbb{R}^n$ represent the *decision variables* or *variables* (*German: Entscheidungsvariablen*). A vector $\boldsymbol{x} \in S$ is called a *(feasible) solution* (*German: (zulässige) Lösung*) of Π. S is the set of (feasible) solutions, and is called the *solution space* or *feasible region* (*German: Lösungsmenge, Lösungsraum*) of Π. The function $f : S \to \mathbb{R}$ is called the *objective function* (*German: Zielfunktion*).

The optimization problem Π corresponds to the task of finding a solution, among all feasible solutions, that maximizes the objective function, i.e. a solution $\boldsymbol{x}^* \in S$ such that

$$f(\boldsymbol{x}^*) \geq f(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in S$$

Such an $\boldsymbol{x}^*$ is called an *optimal solution* or *optimizer* (*German: Optimallösung*) of Π, and the maximum value $f(\boldsymbol{x}^*) = \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ is called the *optimum* or *optimal value* (*German: Optimum, Optimallösung*) of Π. Note that an optimization problem can have multiple optimal solutions, but only one optimum. Moreover, Π has an optimum if and only if it has an optimal solution.

It is possible that a problem Π does not have an optimum (or an optimal solution), i.e. that the expression $\max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ is not defined. To ensure the existence of a optimum (or an optimal solution), three conditions are necessary:

Feasibility: An optimization problem Π is called *feasible* (*German: zulässig*) if its solution space is not empty, i.e. $S \neq \emptyset$, otherwise Π is called *infeasible* (*German: unzulässig*). Trivially, an infeasible problem does not have an optimum. An example of an an infeasible optimization problem is the following:

$$\Pi: \ \max\{x_1 : 2x_1 + 4x_2 = 5, \boldsymbol{x} \in \mathbb{Z}^2\}$$

The objective function is $f(\boldsymbol{x}) = x_1$, and the solution space is $S = \{\boldsymbol{x} \in \mathbb{Z}^2 : 2x_1 + 4x_2 = 5\}$. Clearly $S = \emptyset$, since the sum of the two even numbers $2x_1$ and $4x_2$ can not equal the odd number 5.

Boundedness: A feasible optimization problem Π is called *bounded* (*German: beschränkt*) if there exists an upper bound $\omega \in \mathbb{R}$ such that $f(\boldsymbol{x}) \leq \omega$ for all $\boldsymbol{x} \in S$, otherwise Π is called *unbounded* (*German: unbeschränkt*). If Π is unbounded, i.e. if for all $\omega \in \mathbb{R}$ there exists $\boldsymbol{x} \in S$ such that $f(\boldsymbol{x}) > \omega$, we write $\max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\} = \infty$. An example of an unbounded optimization problem is the following:

$$\Pi: \ \max\{x_1 : 2x_1 + 4x_2 = 5, \boldsymbol{x} \in \mathbb{R}^2\}$$

The objective function is $f(\boldsymbol{x}) = x_1$, and the solution space is $S = \{\boldsymbol{x} \in \mathbb{R}^2 : 2x_1 + 4x_2 = 5\}$. We have $S \neq \emptyset$, since for instance $\boldsymbol{x} = (2.5, 0)^\top \in S$. Clearly, Π is unbounded, since the given equation yields $x_1 = \frac{5}{2} - 2x_2$, hence $x_2 \to -\infty$ implies $f(\boldsymbol{x}) = x_1 \to \infty$.

"Closedness": A feasible, bounded optimization problem Π is called *"closed"* (*German: "abgeschlossen"*) if it has an optimal solution, i.e. if there exists $\boldsymbol{x}^* \in S$ such that $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in S$, otherwise Π is called *"unclosed"* (*German: "unabgeschlossen"*). We deliberately put "closed" in quotation marks since the notion of "closedness" is actually related to elementary topological concepts not discussed here in depth (further explanations can be found in section 2.2.2). To illustrate the "pathology" of "unclosed" optimization problems, we look at the following example:

$$\Pi: \ \max\{x_1 : x_1 < 1, x_1 \in \mathbb{R}\}$$

Clearly, Π is feasible, since for instance $0.5 \in S$, and bounded, since $f(x_1) \leq 1 = \omega$ for all $x_1 \in S$. However, there is no optimal solution, since there is no solution attaining the objective value of 1, i.e. for any $x_1^* \in S$ there exists $x_1 \in S$ such that $f(x_1) > f(x_1^*)$. For instance, x_1 can be chosen as $x_1 = x_1^* + (1 - x_1^*)/2$. Then,

$1 - x_1^* > 0$ since $x_1^* \in S$, hence $(1 - x_1^*)/2 > 0$, and $x_1 > x_1^*$. Another example of an "unclosed" optimization problem is the following:

$$\Pi: \ \max\{2 - \frac{1}{x_1} : x_1 \geq 1, x_1 \in \mathbb{R}\}$$

Again, Π is feasible, since e.g. $1 \in S$, and bounded, since $f(x_1) = 2 - \frac{1}{x_1} \leq 2 = \omega$, but there is no solution attaining the objective value of 2, i.e. for any $x_1^* \in S$ there exists $x_1 \in S$ such that $f(x_1) > f(x_1^*)$. In fact, for any $x_1 > x_1^*$, we have $f(x_1) > f(x_1^*)$.
In the following, we shall only consider "closed" optimization problems.

Definition of Solution Space

The solution space S of an optimization problem $\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ can be defined mathematically in several different ways. Depending on the definition of the solution space, different requirements and difficulties may arise for the solution of Π.

In the following, we consider two possibilities for defining a solution space S. Let $G_i \subseteq \mathbb{R}$, $i = 1, 2, \ldots, n$, denote arbitrary basic sets for the domain of the variables, e.g. $G_i = \mathbb{R}, \mathbb{Q}, \mathbb{Z}, \mathbb{R}_0, \mathbb{Q}_0, \mathbb{Z}_0, \{0, 1\}, \ldots$

(i) Functional Constraints: The solution space S is defined by a set of *inequalities* of the form $g_i(\boldsymbol{x}) \leq 0$, $i = 1, 2, \ldots, p$ $(p \geq 0)$, and a set of *equalities* $h_j(\boldsymbol{x}) = 0$, $j = 1, 2, \ldots, q$ $(q \geq 0)$:

$$S = \{\boldsymbol{x} \in G_1 \times G_2 \times \cdots \times G_n : g_i(\boldsymbol{x}) \leq 0, h_j(\boldsymbol{x}) = 0, i = 1, 2, \ldots, p, j = 1, 2, \ldots, q\}$$

Here, g_i and h_j are arbitrary n-ary functions $g_i, h_j : G_1 \times G_2 \times \cdots \times G_n \to \mathbb{R}$ that are "explicitly defined in form of analytic expressions".

Example:

$$S = \{\boldsymbol{x} \in \mathbb{R} \times \mathbb{Z} \times \{0, 1\} : x_1 + x_2 + x_3 = 1, \boldsymbol{x} \geq \boldsymbol{0}\}$$

In this example, $\boldsymbol{x} = (x_1, x_2, x_3)^\top$ is a 3-dimensional vector, whose components have domains $x_1 \in \mathbb{R}$, $x_2 \in \mathbb{Z}$, and $x_3 \in \{0, 1\}$, respectively. There are four constraints in total: one equality $x_1 + x_2 + x_3 = 1$, and three inequalities $-x_1 \leq 0$, $-x_2 \leq 0$, and $-x_3 \leq 0$. Note that inequality $\boldsymbol{x} \geq \boldsymbol{0}$ is actually a vectorial inequality, expressing $(x_1, x_2, x_3)^\top \geq (0, 0, 0)^\top$.

Note that every "$\geq$" inequality can be transformed into a "$\leq$" inequality, since $h_j(\boldsymbol{x}) \geq 0 \Leftrightarrow -h_j(\boldsymbol{x}) \leq 0$. Therefore, mathematical model formulations usually include only one type of inequality (here we use "$\leq$", for instance). Furthermore, the right-hand side of all inequalities and equalities can be assumed to be zero, since any non-zero value can be moved to the left-hand side by subtraction.

Optimization models with functional constraints constitute the domain of *Mathematical Programming* (*German: Mathematische Programmierung*).

(ii) Non-Functional Constraints The definition of the solution space S may contain "non-functional" constraints, for instance logical predicates which select the feasible elements from a given set, based on certain properties:

$$S = \{\boldsymbol{x} \in G_1 \times G_2 \times \cdots \times G_n : "\boldsymbol{x} \text{ has certain properties}"\}$$

The following example shows a non-functional definition of a solution space:

$$S = \{\boldsymbol{x} \in \mathbb{Z}^n : \boldsymbol{x} \text{ is a permutation of the numbers } 1, 2, \ldots, n\}$$

As we shall discuss in more detail later on, it is theoretically possible in most cases to find, for a set defined by non-functional constraints, a corresponding formulation with functional constraints. However, finding such a formulation can be very challenging.

Maximization and Minimization

Optimization problems can be maximization or minimization problems. Every maximization problem can easily be transformed into an "equivalent" minimization problem (and vice versa), since we have

$$\max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\} = -\min\{-f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$$

and $\boldsymbol{x}^* \in S$ is an optimal solution of the maximization problem if and only if $\boldsymbol{x}^*$ is an optimal solution of the minimization problem. This connection is illustrated in Figure 2.3 using the example of a function $f : \mathbb{R} \to \mathbb{R}$.

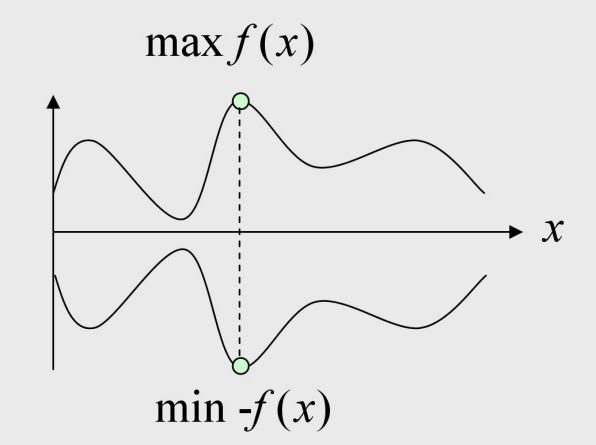


Figure 2.3: Maximization and Minimization

In the following, we shall usually consider only one type of optimization (max), and all explanations are directly applicable to the other type (min), unless indicated otherwise.

Standard Form of Optimization Problems

We sometimes refer to the *standard form of an optimization problem* (see for instance [3]) which in general is understood as a minimization problem with functional constraints as follows:

$$\begin{aligned} &\min f(\boldsymbol{x}) && \\ &g_i(\boldsymbol{x}) \leq 0, && i = 1, 2, \ldots, p \\ &h_j(\boldsymbol{x}) = 0, && j = 1, 2, \ldots, q \end{aligned}$$

As before, we have $\boldsymbol{x} \in \mathbb{R}^n$, and the functions f, g_i, and h_j are n-ary functions $\mathbb{R}^n \to \mathbb{R}$.

2.2.2 Some Definitions and Concepts

Problems and Problem Instances

Sometimes, it is appropriate to distinguish between the notion of an *optimization problem* and the notion of an *instance of an optimization problem*. Consider for instance an optimization problem of the form:

$$\max\{\sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} a_j x_j \leq b, \boldsymbol{x} \in \mathbb{R}^n\}$$

The parameters of the problem are given by $n \in \mathbb{R}_0$, $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)^\top \in \mathbb{R}^n$, $b \in \mathbb{R}$, and $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)^\top \in \mathbb{R}^n$. By fixing numerical values for the parameters $n, \boldsymbol{a}, b, \boldsymbol{c}$, we obtain a *"concrete"* optimization problem, a so-called *problem instance*. For example, with $n = 2$, $\boldsymbol{a} = (3, 5)^\top$, $b = 17$, and $\boldsymbol{c} = (4, 7)^\top$, we obtain the following instance:

$$\max\{4x_1 + 7x_2 : 3x_1 + 5x_2 \leq 17, \boldsymbol{x} \in \mathbb{R}^2\}$$

Thus, an optimization problem corresponds to a (typically infinite) collection of problem instances that all have the same "mathematical structure" but different values for the parameters. Mathematical analyses, results, methods, etc. normally refer to problems and not to individual problem instances.

Neighborhoods

Let $S \subseteq \mathbb{R}^n$ be an arbitrary set of "points" in $\mathbb{R}^n$, and $\mathcal{P}(S) = \{S' : S' \subseteq S\}$ the correspoding power set, i.e. the set of all subsets of S. A *neighborhood* (*German: Nachbarschaft*) defined on S is a (set-valued) function of the form

$$N : S \to \mathcal{P}(S)$$

that assigns to each point $\boldsymbol{x} \in S$ a set of neighboring points $N(\boldsymbol{x})$, assuming always $\boldsymbol{x} \in N(x)$, i.e. $\boldsymbol{x}$ is a neighbor of itself. $N(\boldsymbol{x})$ is called a neighborhood of $\boldsymbol{x}$.

A special neighborhood in $\mathbb{R}^n$ is the so-called *Euclidean neighborhood* or *ε-neighborhood* $N_\varepsilon(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^n$, which is defined as:

$$N_\varepsilon(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^n : \|\boldsymbol{y} - \boldsymbol{x}\| < \varepsilon\} \quad \text{where } \varepsilon > 0$$

Here, $||\boldsymbol{x}||$ denotes the Euclidean norm (the "length") of the vector $\boldsymbol{x} \in \mathbb{R}^n$ which is defined as:

$$||\boldsymbol{x}|| = \sqrt{\boldsymbol{x}^2} = \sqrt{x_1^2 + \ldots + x_n^2}$$

In $\mathbb{R}^2$ (or $\mathbb{R}^3$), the neighborhood $N_\varepsilon(\boldsymbol{x})$ corresponds to the set of all points $\boldsymbol{y}$ located inside a circle (or ball) with center $\boldsymbol{x}$ and radius $\varepsilon > 0$, with "boundary" points excluded. An illustration for $\mathbb{R}^2$ is given in Figure 2.4.

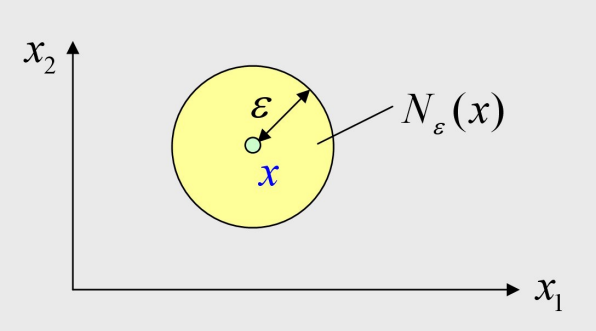


Figure 2.4: Euclidean neighborhood

A point $\boldsymbol{x} \in S$ is called an *interior point* of S if S contains an ε-neighborhood of $\boldsymbol{x}$, i.e. if there exists $\varepsilon > 0$ such that $N_\varepsilon(\boldsymbol{x}) \subseteq S$. A point $\boldsymbol{x} \in S$ is called a *boundary point* of S if every ε-neighborhood of $\boldsymbol{x}$ contains at least one point in S and one point outside S, i.e. if for all $\varepsilon > 0$, $N_\varepsilon(\boldsymbol{x}) \cap S \neq \emptyset$ and $N_\varepsilon(\boldsymbol{x}) \cap (\mathbb{R}^n - S) \neq \emptyset$. Note that a boundary point may be located inside or outside of S. The set of all boundary points is called the *boundary* of S.

A set $S \subseteq \mathbb{R}^n$ is *closed* if it contains all its boundary points, and *open* if it contains no boundary point, i.e. if all points of S are interior points. There exist sets which are neither closed nor open, for instance $S = \{x_1 \in \mathbb{R} : 0 \leq x_1 < 1\} = [0, 1[$, where 0 and 1 are boundary points with $0 \in S$ and $1 \notin S$.

A set $S \subseteq \mathbb{R}^n$ is *bounded* if there exist vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ such that $S \subseteq \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a} \leq \boldsymbol{x} \leq \boldsymbol{b}\}$, i.e. the components of the points in S do not exceed certain lower and upper bounds.

Closedness of sets is of crucial importance in optimization theory, as is shown by the following famous theorem of Weierstrass.

Theorem 2.1 (Extreme Value Theorem of Weierstrass)**.** *Let $S \subseteq \mathbb{R}^n$ be a non-empty, bounded, closed set and $f : S \to \mathbb{R}$ a continuous function. Then there exists $\boldsymbol{x}^* \in S$ with $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in S$, i.e. the optimization problem $\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ has an optimal solution (and hence a finite optimum).*

Besides the ε-neighborhood, there are other types of neighborhoods often utilized in optimization. In particular, this is the case in the development of meta-heuristics (e.g. local search) in discrete optimization. An example of a *discrete ("combinatorial") neighborhood* in $\mathbb{Z}^n$ is the following:

$$N(\boldsymbol{x}) = \{\boldsymbol{x}' \in \mathbb{Z}^n : \boldsymbol{x}' \text{ differs in at most one component from } \boldsymbol{x}\}$$

An illustration of this neighborhood in $\mathbb{R}^2$ is shown in Figure 2.5.

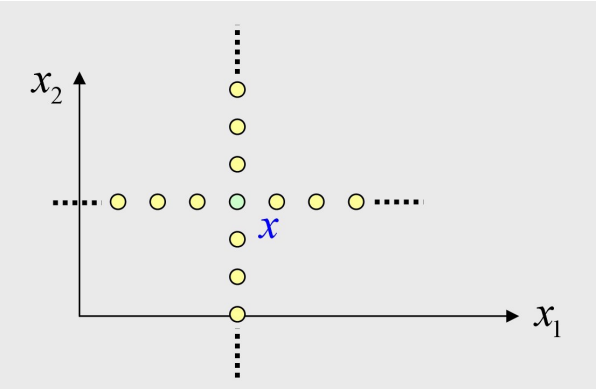


Figure 2.5: Example of a combinatorial neighborhood

Global and Local Optima

Consider an optimization problem of the form $\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ with $S \subseteq \mathbb{R}^n$. As already mentioned, $\boldsymbol{x}^* \in S$ is called an optimal solution of Π if

$$f(\boldsymbol{x}^*) \geq f(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in S$$

$\boldsymbol{x}^*$ is also called a *globally optimal solution*, and the optimum $f(\boldsymbol{x}^*)$ is called the *global optimum*.

Besides the global optimum of a problem, there may exist so-called *local optima* which also play an important role in optimization.

Definition 2.1 (Local Optimum)**.** Consider an optimization problem $\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$. A solution $\boldsymbol{x}^* \in S$ is a *locally optimal solution of* Π, and $f(\boldsymbol{x}^*)$ is a *local optimum* of Π (with respect to the neighborhood N_ε), if there exists $\varepsilon > 0$ such that

$$f(\boldsymbol{x}^*) \geq f(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in N_\varepsilon(\boldsymbol{x}^*) \cap S$$

Expressed in words, a solution is locally optimal if it is as least as good as all its neighbors in a certain neighborhood. Note that every globally optimal solution is also a locally optimal solution (but not conversely).

Figure 2.6 shows an example in $\mathbb{R}^1$. Note that in $\mathbb{R}^1$, the ε-neighborhoods to be considered for checking local optimality of a solution $x \in \mathbb{R}^1$ correspond to the open intervals $]x - \varepsilon, x + \varepsilon[$.

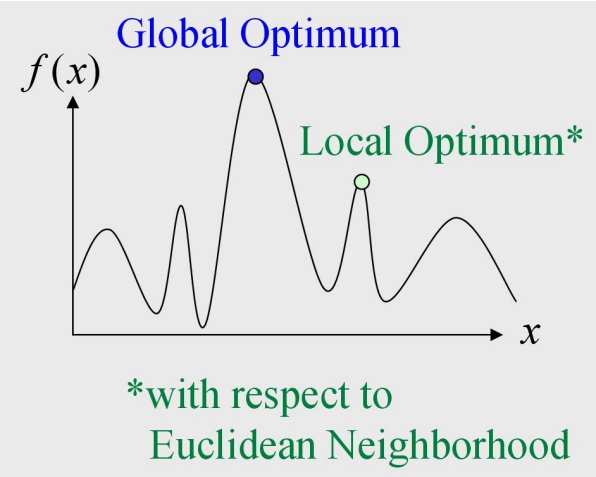


Figure 2.6: Global and local optima

Level Sets

Level sets (*German: Niveaumengen*) of functions play an important role in analysis and optimization, notably for the graphical representation of 2- and 3-dimensional functions. Recall that the graph H of a function $f : S \to \mathbb{R}$ with $S \subseteq \mathbb{R}^n$ is the $n+1$-dimensional subset $H \subseteq \mathbb{R}^{n+1}$ defined by

$$H = \{(\boldsymbol{x}, f(\boldsymbol{x})) : \boldsymbol{x} \in S\}$$

Definition 2.2 (Level Set)**.** Let $f : S \to \mathbb{R}$ with $S \subseteq \mathbb{R}^n$. The *level set* L_α of f for level $\alpha \in \mathbb{R}$ is defined as

$$L_\alpha = \{\boldsymbol{x} \in S : f(\boldsymbol{x}) = \alpha\}$$

For n-ary functions f, the level sets have "dimension" $n - 1$ (or less). In the case of $n = 2$, level sets are typically "curves" in the plane (i.e. 1-dimensional sets), and are sometimes called "level curves" or "contour lines" (German: Höhenkurven). For $n = 3$, level sets are typically "surfaces" in the 3-dimensional space (i.e. 2-dimensional sets) and are sometimes called "level surfaces" (German: Niveauflächen).

By means of level sets, graphs of 2- and 3-dimensional functions (which are 3- and 4-dimensional sets, respectively) can be represented in lower dimension as 2- and 3-dimensional images, respectively. An example is given in Figure 2.7.

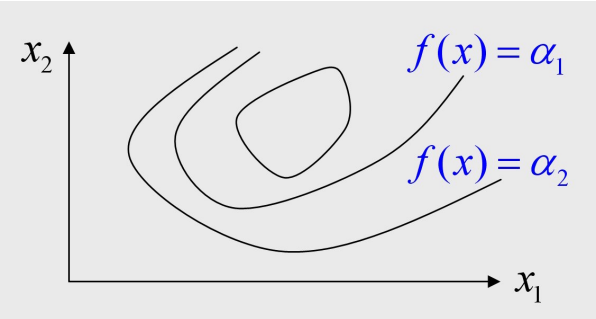


Figure 2.7: Level sets of a function $f : \mathbb{R}^2 \to \mathbb{R}$

If $f : \mathbb{R}^2 \to \mathbb{R}$ is a linear function, i.e. if $f(\boldsymbol{x}) = c_1x_1 + c_2x_2$ for some $\boldsymbol{c} \in \mathbb{R}^2$, the level sets are parallel lines in $\mathbb{R}^2$, and the vector $\boldsymbol{c}$ (the gradient of f) is orthogonal to these lines. The vector $\boldsymbol{c}$ points in the direction where the level α (i.e. the function value) increases if level lines are "shifted in parallel" in this direction. See Figure 2.8 for an example. More generally, if $f : \mathbb{R}^n \to \mathbb{R}$ is a linear function, i.e. if $f(\boldsymbol{x}) = c_1x_1 + \cdots + c_nx_n$ for some $\boldsymbol{c} \in \mathbb{R}^n$, the level sets are parallel hyperplanes (see below) in $\mathbb{R}^n$, and the vector $\boldsymbol{c}$ (the gradient of f) is orthogonal to these hyperplanes.

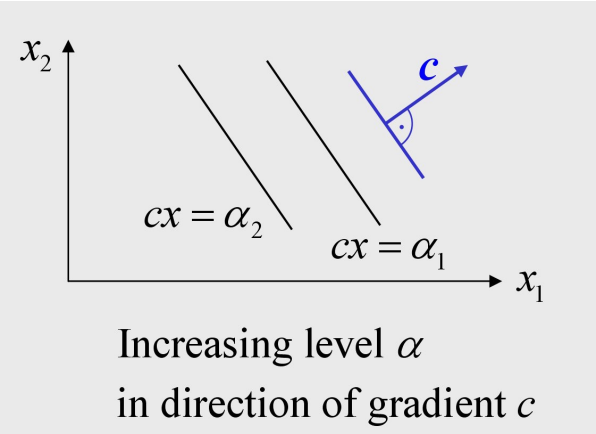


Figure 2.8: Level sets of a linear function $f : \mathbb{R}^2 \to \mathbb{R}$

2.3 Convex Optimization

Convex optimization problems constitute an important class of optimization problems. They have the special property that every locally optimal solution is a globally optimal solution (and hence, any local optimum is a global optimum) what substantially simplifies the optimization task. In the following, we shall explain the notion of convex optimization.

Definition 2.3 (Convex Combination)**.**
(i) A vector $\boldsymbol{x} \in \mathbb{R}^n$ is a *convex combination of two vectors* $\boldsymbol{x}^1, \boldsymbol{x}^2 \in \mathbb{R}^n$ if

$$\boldsymbol{x} = \lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2 \quad \text{for some } \lambda \in \mathbb{R} \text{ with } 0 \leq \lambda \leq 1$$

(ii) A vector $\boldsymbol{x} \in \mathbb{R}^n$ is a *convex combination of* k *vectors* $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^k \in \mathbb{R}^n$ if

$$\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i \quad \text{for some } \boldsymbol{\lambda} \in \mathbb{R}^k \text{ with } \mathbf{0} \leq \boldsymbol{\lambda} \leq \mathbf{1} \text{ and } \sum_{i=1}^{k} \lambda_i = 1$$

Geometrically, a convex combination $\boldsymbol{x} = \lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2$ is a point on the line segment joining the points $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$. This can easily be seen if $\boldsymbol{x}$ is written as $\boldsymbol{x} = \boldsymbol{x}^2 + \lambda(\boldsymbol{x}^1 - \boldsymbol{x}^2)$, since the vector $\boldsymbol{x}^1 - \boldsymbol{x}^2$ actually corresponds to the "direction" of the line segment from $\boldsymbol{x}^2$ to $\boldsymbol{x}^1$. In particular, $\boldsymbol{x} = \boldsymbol{x}^2$ for $\lambda = 0$, and $\boldsymbol{x} = \boldsymbol{x}^1$ for $\lambda = 1$.

Definition 2.4 (Convex Set)**.** A set $S \subseteq \mathbb{R}^n$ is *convex* if

$$\lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2 \in S \quad \text{for all } \boldsymbol{x}^1, \boldsymbol{x}^2 \in S \text{ and all } \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1$$

In other words, a set S is convex if it contains the line segments between all pairs of points $\boldsymbol{x}^1, \boldsymbol{x}^2 \in S$ (see Figure 2.9).

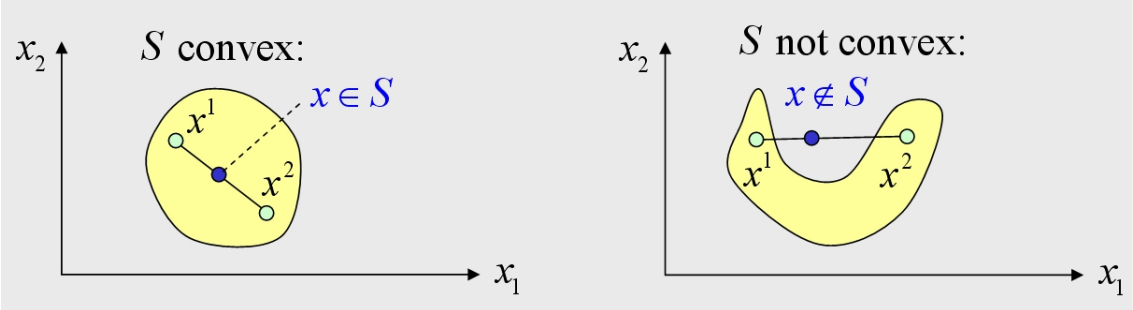


Figure 2.9: Convex set

Proposition 2.1. *The intersection of a collection of convex sets* $S_1, S_2, \ldots, S_k$, $k \geq 1$, *is convex.*

Proof. Let $\boldsymbol{x}^1, \boldsymbol{x}^2 \in \bigcap_{i=1}^{k} S_i$, and let $\boldsymbol{x} = \lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2$ for some $\lambda \in \mathbb{R}$ with $0 \leq \lambda \leq 1$. Since $\boldsymbol{x}^1, \boldsymbol{x}^2 \in S_i$ and S_i is convex, it follows that $\boldsymbol{x} \in S_i$ for all $i = 1, 2, \ldots, k$, hence $\boldsymbol{x} \in \bigcap_{i=1}^{k} S_i$. □

Definition 2.5 (Convex and Concave Function)**.**
(i) A function $f : S \to \mathbb{R}$ is *convex* in $S \subseteq \mathbb{R}^n$, if S is convex and

$$f(\lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2) \leq \lambda f(\boldsymbol{x}^1) + (1-\lambda) f(\boldsymbol{x}^2)$$
$$\text{for all } \boldsymbol{x}^1, \boldsymbol{x}^2 \in S \text{ and all } \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1$$

(ii) A function $f : S \to \mathbb{R}$ is *concave* in $S \subseteq \mathbb{R}^n$, if S is convex and

$$f(\lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2) \geq \lambda f(\boldsymbol{x}^1) + (1-\lambda) f(\boldsymbol{x}^2)$$
$$\text{for all } \boldsymbol{x}^1, \boldsymbol{x}^2 \in S \text{ and all } \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1$$

In other words, a function is convex (concave) if it is overestimated (underestimated) by linear interpolation. See Figure 2.10 for an illustration in $\mathbb{R}^1$. Note that f is convex if and only if $-f$ is concave.

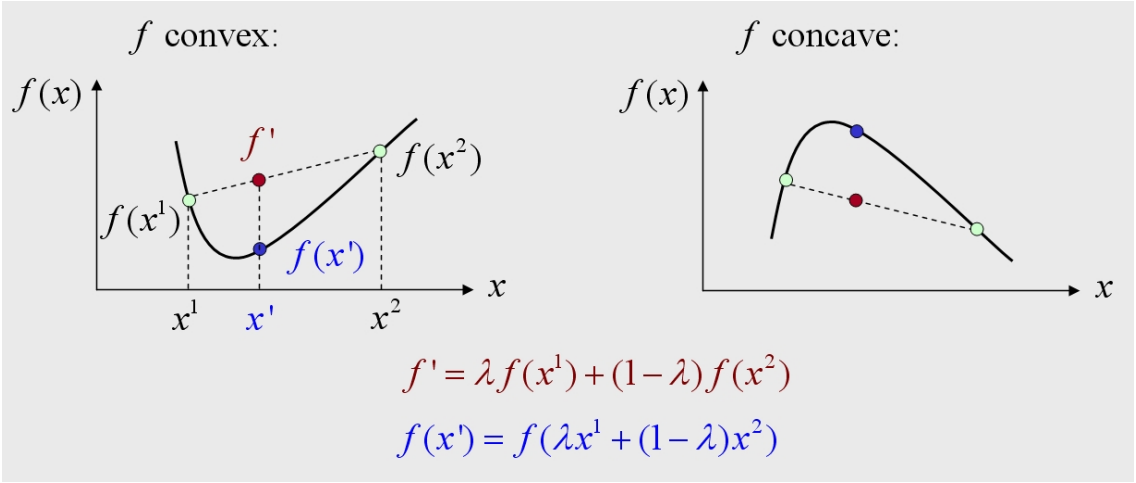


Figure 2.10: Convex and concave functions

Proposition 2.2. *A linear function $f : S \to \mathbb{R}$ on a convex set $S \subseteq \mathbb{R}^n$ is convex and concave in S.*

Proof. Let $f : S \to \mathbb{R}$ be a linear function and $S \subseteq \mathbb{R}^n$ a convex set. Let $\boldsymbol{x}^1, \boldsymbol{x}^2 \in S$ and $0 \leq \lambda \leq 1$. By definition of linearity, we have

$$f(\lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2) = \lambda f(\boldsymbol{x}^1) + (1-\lambda) f(\boldsymbol{x}^2)$$

Since this equality includes both cases, "$\leq$" and "$\geq$", f is both convex and concave in S. □

Definition 2.6 (Convex Optimization Problem)**.** A *convex optimization problem* is an optimization problem of the form

$$\min\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$$

where $S \in \mathbb{R}^n$ is a convex set and $f : S \to \mathbb{R}$ is a convex function on S.

Often, *maximization* of a *concave* function over a convex set S is also called a convex optimization problem (or a concave optimization problem). Clearly, such a problem can be transformed into another equivalent convex optimization problem: We have $\max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\} = -\min\{-f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$, and since f is concave, $-f$ is convex, i.e. the minimization problem is convex.

A fundamental result in convex optimization is the following theorem.

Theorem 2.2. *In a convex optimization problem, every locally optimal solution is a globally optimal solution.*

Proof. Consider the convex optimization problem $\Pi : \min\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ with f convex in S. Let $\boldsymbol{x}^*$ be a locally optimal solution of Π. Then there exists $\varepsilon > 0$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in N_\varepsilon(x^*) \cap S$. Suppose that $\boldsymbol{x}^*$ is not globally optimal, i.e. there exists $\boldsymbol{y}^* \in S$ such that $f(\boldsymbol{y}^*) < f(\boldsymbol{x}^*)$. Clearly $\boldsymbol{y}^* \notin N_\varepsilon(\boldsymbol{x}^*)$, i.e. $||\boldsymbol{y}^* - \boldsymbol{x}^*|| \geq \varepsilon$, since otherwise $f(\boldsymbol{x}^*) \leq f(\boldsymbol{y}^*)$. Consider now the point $\boldsymbol{z}$ given by

$$\boldsymbol{z} = (1-\lambda)\boldsymbol{x}^* + \lambda \boldsymbol{y}^* \quad \text{with} \quad \lambda = \frac{\varepsilon}{2||\boldsymbol{y}^* - \boldsymbol{x}^*||}$$

Note that $0 < \lambda < 1$ since $\varepsilon > 0$ and $||\boldsymbol{y}^* - \boldsymbol{x}^*|| \geq \varepsilon$. It follows that $||\boldsymbol{z} - \boldsymbol{x}^*|| = \frac{\varepsilon}{2}$ since

$$(\boldsymbol{z} - \boldsymbol{x}^*)^2 = ((1-\lambda)\boldsymbol{x}^* + \lambda \boldsymbol{y}^* - \boldsymbol{x}^*)^2 = (\lambda(\boldsymbol{y}^* - \boldsymbol{x}^*))^2 = \frac{\varepsilon^2}{4(\boldsymbol{y}^* - \boldsymbol{x}^*)^2}(\boldsymbol{y}^* - \boldsymbol{x}^*)^2 = \frac{\varepsilon^2}{4}$$

Consequently, $||\boldsymbol{z} - \boldsymbol{x}^*|| = \frac{\varepsilon}{2} < \varepsilon$ and hence $\boldsymbol{z} \in N_\varepsilon(\boldsymbol{x}^*)$. Because of the convexity of S, $\boldsymbol{x}^*, \boldsymbol{y}^* \in S$ implies $\boldsymbol{z} \in S$, since $\boldsymbol{z}$ is a convex combination of $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$. From the convexity of f, and since $f(\boldsymbol{y}^*) < f(\boldsymbol{x}^*)$ and $\lambda > 0$, it then follows

$$f(\boldsymbol{z}) = f((1-\lambda)\boldsymbol{x}^* + \lambda \boldsymbol{y}^*) \leq (1-\lambda)f(\boldsymbol{x}^*) + \lambda f(\boldsymbol{y}^*) < (1-\lambda)f(\boldsymbol{x}^*) + \lambda f(\boldsymbol{x}^*) = f(\boldsymbol{x}^*)$$

Thus $f(\boldsymbol{z}) < f(\boldsymbol{x}^*)$. But since $\boldsymbol{z} \in N_\varepsilon(\boldsymbol{x}^*) \cap S$, we obtain a contradiction to the local optimality of $\boldsymbol{x}^*$, which says that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in N_\varepsilon(\boldsymbol{x}^*) \cap S$, and in particular, $f(\boldsymbol{x}^*) \leq f(\boldsymbol{z})$. Therefore, $\boldsymbol{x}^*$ must be globally optimal. □

Another interesting optimization property arises with the *maximization* of a *convex* function (or the *minimization* of a *concave* function) f over a convex set $S \subseteq \mathbb{R}^n$. In this case, the following theorem holds (without proof).

Theorem 2.3. *Let $S \subseteq \mathbb{R}^n$ be a convex, closed, and bounded set. Consider an optimization problem of the form $\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ (resp. $\Pi : \min\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$) with f convex (resp. concave) in S, and f not constant in S. Then any optimal solution is a boundary point of S.*

Figure 2.11 shows an illustration of these two theorems. Figure 2.12 demonstrates by means of an example, that in a maximization problem with a concave objective function and a non-convex solution space, it is possible to have local maxima which are not globally optimal.

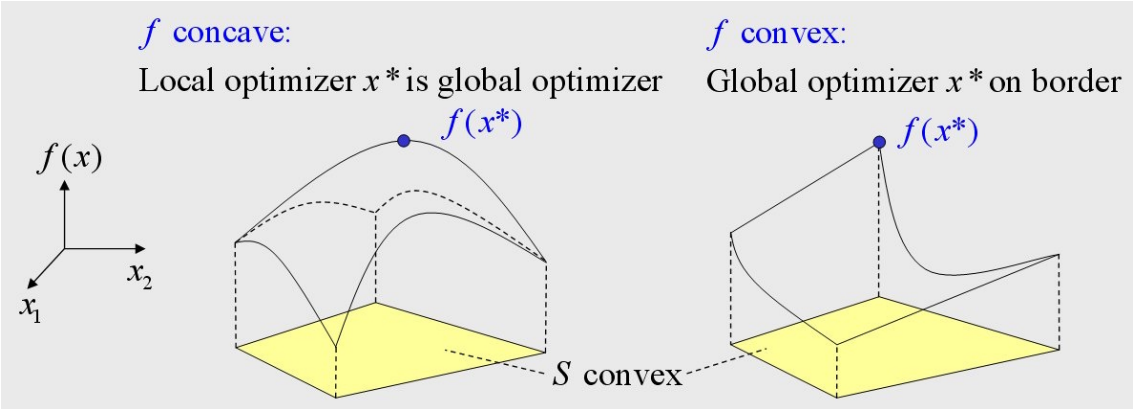


Figure 2.11: Maximization of a convex or concave function on a convex solution space

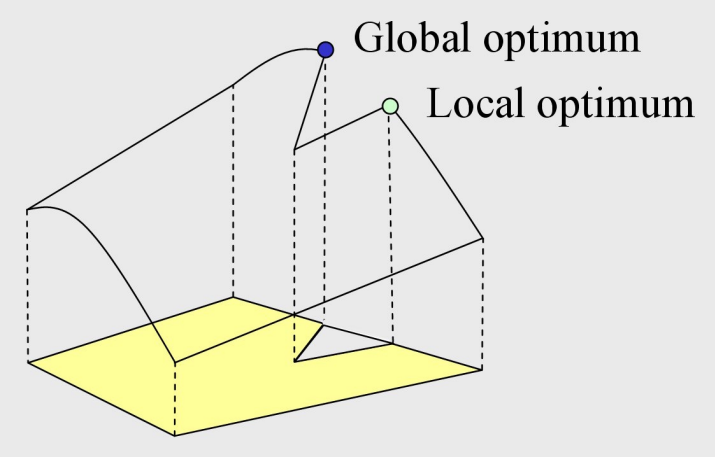


Figure 2.12: Local optimum in the maximization of a concave function over a non-convex solution space

Finally, note that in the case of optimization (maximization or minimization) of a *linear function* f over a convex, closed, and bounded set S, both of the above theorems are valid, since f is convex as well as concave. Therefore, in this case, any locally optimal solution is globally optimal, and is located on the boundary of S. To determine a globally

optimal solution, it hence suffices to find a *locally optimal solution on the boundary* of S.

In convex optimization, we often consider problems with functional constraints of the following form:

$$\min\{f(\boldsymbol{x}) : g_i(\boldsymbol{x}) \leq b_i, i \in I, \boldsymbol{x} \in \mathbb{R}^n\}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ as well as all constraint defining functions $g_i : \mathbb{R}^n \to \mathbb{R}, i \in I$, are convex on $\mathbb{R}^n$. This type of optimization problem is called *Convex Programming*. The following proposition shows that the solution space of such problems is indeed convex.

Proposition 2.3. *A set of the form* $S = \{\boldsymbol{x} \in \mathbb{R}^n : g_i(\boldsymbol{x}) \leq b_i, i \in I\}$ *with convex functions* $g_i : \mathbb{R}^n \to \mathbb{R}$ *and* $b_i \in \mathbb{R}$ *for* $i \in I$ *is convex.*

Proof. First show that the so-called sublevel sets $U = \{\boldsymbol{x} \in T : g(\boldsymbol{x}) \leq b\}$, $b \in \mathbb{R}$, of a function $g : T \to \mathbb{R}$ are convex if g is convex on $T \subseteq \mathbb{R}^n$. Let $\boldsymbol{x}^1, \boldsymbol{x}^2 \in U$ and $\boldsymbol{z} = \lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2$ with $0 \leq \lambda \leq 1$. The convexity of T implies $\boldsymbol{z} \in T$. Since g is convex on T and $\boldsymbol{x}^1, \boldsymbol{x}^2 \in U$, we further have

$$g(\boldsymbol{z}) = g(\lambda \boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2) \leq \lambda g(\boldsymbol{x}^1) + (1-\lambda) g(\boldsymbol{x}^2) \leq \lambda b + (1-\lambda) b = b$$

Hence $g(\boldsymbol{z}) \leq b$ and $\boldsymbol{z} \in T$, i.e. $\boldsymbol{z} \in U$. Therefore, any convex combination $\boldsymbol{z}$ of two points $\boldsymbol{x}^1, \boldsymbol{x}^2 \in U$ is in U, i.e. U is convex.

Clearly, S is the intersection of the sublevel sets $S_i = \{\boldsymbol{x} \in \mathbb{R}^n : g_i(\boldsymbol{x}) \leq b_i\}$, i.e. $S = \bigcap_{i \in I} S_i$. Since the functions g_i are convex, the sets S_i are also convex, and since, by Proposition 2.1, the intersection of convex sets is convex, we have shown that S is convex. □

Finally, we point out that most optimization methods in convex optimization are notably based on the property that, due to the special mathematical structure, global optimality of a solution can be determined on the basis of "local" criteria. An example for this are the so-called Karush-Kuhn-Tucker conditions in the case of continuously differentiable functions.

2.4 Types of Optimization Models and Methods

The methodological requirements for solving mathematical optimization problems are mainly determined by the mathematical structure of the objective function and the solution space. Depending on their properties, different difficulties have to be faced and different solution methods have to be applied. In the following, some elementary criteria for categorizing optimization models are listed, and a short overview of basic types of optimization methods is given.

2.4.1 Types of Optimization Models

We consider again general optimization models (optimization problems) of the form

$$\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$$

where $S \subseteq \mathbb{R}^n$, and in particular, functionally specified optimization models where the solution space is defined by a finite number of equations and inqualities of the form

$$S = \{\boldsymbol{x} \in G_1 \times G_2 \times \ldots \times G_n : g_i(\boldsymbol{x}) = 0, h_j(\boldsymbol{x}) \leq 0, i = 1, \ldots, p, j = 1, \ldots, q\}$$

Constrained vs. Unconstrained Optimization *(German: Eingeschränkte vs. Uneingeschränkte Optimierung)* A problem is called constrained if $S \subset \mathbb{R}^n$, and unconstrained if $S = \mathbb{R}^n$. Constrained problems comprise *constraints* (feasibility conditions) restricting the set of feasible solutions S to a proper subset of $\mathbb{R}^n$. These contraints can be of functional or non-functional type. In the latter case, solving the problem typically involves the additional task to find a suitable functional description of the constraints (this is not always necessary, however). Generally spoken, constrained problems are typically more difficult to solve than unconstrained problems.

Global vs. Local Optimization Global optimization corresponds to the task of finding a globally optimal solution for an optimization problem (or to establish that no such solution exists). Local optimization aims at finding locally optimal solutions. Trivially, global optimization is at least as difficult as local optimization. As we have seen in the case of convex optimization, there are classes of optimization problems for which both tasks are identical. Often the methodological approaches for finding global or local optima are of completely different nature. Many domains of optimization theory beyond convex optimization are mainly focused on local optimization, including the domain of general non-linear optimization.

Differentiable vs. Non-Differentiable Optimization For differentiable optimization models we generally assume that a functional specification is available and all involved functions f, g_i, h_j are twice continuously differentiable. If this is not the case, one speaks of non-differentiable models. Most optimization methods for non-linear, differentiable models are based in some way on function approximation by means of Taylor series expansion.

Discrete vs. Continuous Optimization Discrete optimization models have a discrete solution space S, i.e. the number of elements (solutions) in S is finite or countably infinite. Thus, we have

$$S \subseteq G_1 \times G_2 \times \ldots G_n \qquad \text{where } G_i \subset \mathbb{R} \text{ is discrete for } i = 1, \ldots, n$$

Often we consider discrete optimization models where all decision variables have to be integer, i.e. $S \in \mathbb{Z}^n$. Also of importance are models where only a part of

the variables are required to be discrete and the other variables are continuous, i.e. $S \subseteq G_1 \times G_2 \times \ldots G_n$ where G_i is discrete for at least one $i \in \{1, \ldots, n\}$. This kind of model is sometimes also called discrete. Continuous optimization refers to the case where no variables are restricted to discrete values.
Discrete optimization is of great importance in the practice of Operations Research since many real-world problems from business and industry, in particular in the area of Operations Management, can be formulated and solved as discrete optimization problems.
The methodological approaches of discrete optimization are typically quite different from that of continuous optimization so that these two categories constitute two more or less independent research areas in Operations Research.

Convex vs. Non-Convex Optimization Convex optimization, as discussed, deals with minimizing (or maximizing) a convex (or concave) objective function f over a convex solution space S. It plays a prominent role in optimization theory since the search for a global optimum reduces to the search of a local optimum.

Linear vs. Non-Linear Optimization Linear optimization (also called Linear Programming) deals with optimizing a linear objective function subject to a finite number of linear constraints (inqualities or equalities). Linear models lie at the roots of Operations Research and feature profound mathematical structure that nowadays allows to solve efficiently even very large-scale problem instances with millions of variables and constraints.
Of central importance in real-world applications are linear models with integer requirements for all or some of the variables (Integer Linear Programming, Mixed Integer Linear Programming). Linear and Integer Linear Programming will be discussed in more detail in the following.

2.4.2 Types of Optimization Methods

There exist a variety of optimization approaches and methods (algorithms) that can be categorized and described according to diverse criteria. We will give a rough classification of optimization methods based on some basic conceptual characteristics.

Exact vs. Heuristic Methods Exact optimization methods guarantee to provide an optimal solution, if existent, and otherwise certify that there is no optimal solution. In contrast, heuristic methods or heuristics (from the ancient Greek word "heuriskein": to find, to discover) are algorithms that seek to find as good a solution as possible, without any garantee for optimality. Basically, every method that searches for some feasible solution for an optimization problem can be viewed as a heuristic (of better or worse quality). Heuristics are typically employed if there are no exact methods available, or exact methods are not applicable due to exceeding computation time.

General vs. Problem-Specific Methods General optimization methods are procedures applicable to a large class of optimization problems. In contrast, problem-specific

methods are only applicable to a narrowly limited problem type and are often explicitly developed for a specific case.

Heuristic methods can be further classified into:

Constructive Heuristics *(German: Konstruktive Heuristiken, Eröffnungsverfahren)* Constructive heuristics gradually build up a solution step by step, typically starting with an "empty solution" and adding in every step a "solution element" (e.g. the value of a decision variable) to the current "partial solution" until a complete solution is achieved.

Improvement Heuristics *(German: Verbesserungsverfahren)* Improvement heuristics (also called iterative improvement methods) start with one or several feasible solutions (e.g. provided by a constructive heuristic) and try to gradually improve the current solution(s) by applying small ("marginal") changes or combining elements of different solutions. This type of heuristics is often summarized under the generic term *metaheuristics*. We distinguish two main types of metaheuristics: (1) *trajectory-based* methods, and (2) *population-based* methods.

Trajectory-base methods start with a single initial solution and iteratively try to improve this solution through marginal changes. The transition from one solution to the next "improved" solution is called a move. A move typically comprises the evaluation of a number of "neighbor solutions" in the "neighborhood" of the current solution, and the selection of an appropriate successive solution among them. The process of moving from one solution to a "neighbor" solution creates a series of subseqent solutions than can be viewed as a "trajectory" through the solution space. Trajectory-based methods are also called *local search* methods. They stop when no better solution can be found in the neighborhood of the current solution, and the resulting solution is called locally optimal, with respect to the neighborhood used.

Population-based methods, in particular genetic algorithms, start with an initial population of candidate solutions, and iteratively try to improve this solution pool by combining solutions ("crossover"), changing individual solutions ("mutation"), and retaining only the best solutions in the pool ("selection").

Metaheuristics are of central importance in the practice of Operations Research since many real-world optimization problems are too big or too complex to be solved by exact methods. A variety of successful optimization applications, e.g. in production, logistics, and services, are based on metaheuristic approaches.

Metaheuristics play also an important role, in combination with sophisticated nonlinear "local" optimization methods, in finding globally optimal solutions (or nearly optimal solutions) in non-convex optimization.

3 Linear Programming

3.1 Problem Formulation

In Linear Programming, our goal is to optimize (i.e. maximize or minimize) a *linear objective function* subject to a finite number of *linear constraints* (i.e. linear inequality or equality constraints).

Remember that a *linear function* $f : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{x} \mapsto f(\boldsymbol{x})$ has the form

$$f(\boldsymbol{x}) = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = \sum_{j=1}^{n} a_j x_j \qquad \text{or} \qquad f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{x}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$ denotes the *vector of variables* and $\boldsymbol{a} = (a_1, \ldots, a_n)^\top \in \mathbb{R}^n$ the *vector of coefficients*.

A *linear inequality* is a constraint of the form

$$\sum_{j=1}^{n} a_j x_j \leq b \quad \text{or} \quad \boldsymbol{a}^\top \boldsymbol{x} \leq b$$

with $b \in \mathbb{R}$ (a *linear equation* being defined analogously with "=").

A finite set of linear inequalities can be written in *matrix form* as

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$. The individual inequalities are then given in *vector form* by

$$\boldsymbol{a}^i \boldsymbol{x} \leq b_i, \qquad i \in I$$

where $\boldsymbol{a}^i$ denotes i-th row of $\boldsymbol{A}$ and $I = \{1, \ldots, m\}$. *Written out explicitly*, the inequalities read as

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i \in I$$

When considering linear equality constraints, alle the above mentioned applies analogously. Depending on the situation, we write linear constraints in matrix form, in vector form, or in explicit notation.

If nothing else is said, we assume from now on that $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c} \in \mathbb{R}^n$, and $I = \{1, \ldots, m\}$, $J = \{1, \ldots, n\}$.

A Linear Program (LP) can be written in different forms, and for each form, we distinguish between maximization and minimization problems:

- in general form
- in canonical form
- in standard form
- in inequality form

These forms are defined in the following, where we assume that the index sets I_1, I_2, I_3 form a partition of $I = \{1, \ldots, m\}$, and the index sets J_1, J_2, J_3 form a partition of $J = \{1, \ldots, n\}$.

- **LP in General Form:**

$$\begin{aligned} \max/\min\ & \boldsymbol{c}^\top \boldsymbol{x} & \\ \boldsymbol{a}^i \boldsymbol{x} &\leq b_i, & i \in I_1 \\ \boldsymbol{a}^i \boldsymbol{x} &= b_i, & i \in I_2 \\ \boldsymbol{a}^i \boldsymbol{x} &\geq b_i, & i \in I_3 \\ x_j &\geq 0, & j \in J_1 \\ x_j &\text{ free}, & j \in J_2 \\ x_j &\leq 0, & j \in J_3 \end{aligned}$$

- **LP in Canonical Form:**

$$\begin{aligned} \max\ & \boldsymbol{c}^\top \boldsymbol{x} & \\ \boldsymbol{a}^i \boldsymbol{x} &\leq b_i, & i \in I \\ x_j &\geq 0, & j \in J \end{aligned}$$

$$\begin{aligned} \min\ & \boldsymbol{c}^\top \boldsymbol{x} & \\ \boldsymbol{a}^i \boldsymbol{x} &\geq b_i, & i \in I \\ x_j &\geq 0, & j \in J \end{aligned}$$

- **LP in Standard Form:**

$$\begin{aligned} \max/\min\ & \boldsymbol{c}^\top \boldsymbol{x} & \\ \boldsymbol{a}^i \boldsymbol{x} &= b_i, & i \in I \\ x_j &\geq 0, & j \in J \end{aligned}$$

- **LP in Inequality Form:**

$$\begin{aligned} &\max \boldsymbol{c}^\top \boldsymbol{x} \\ &\boldsymbol{a}^i \boldsymbol{x} \leq b_i, \qquad i \in I \end{aligned}$$

$$\begin{aligned} &\min \boldsymbol{c}^\top \boldsymbol{x} \\ &\boldsymbol{a}^i \boldsymbol{x} \geq b_i, \qquad i \in I \end{aligned}$$

It is easily possible to transform a LP from any form into any other form. The following transformation rules can be used:

- A "$\leq$" inequality can be replaced by a "$\geq$" inequality (and vice versa):

$$\begin{aligned} \boldsymbol{a}^i \boldsymbol{x} \leq b_i \quad &\rightsquigarrow \quad -\boldsymbol{a}^i \boldsymbol{x} \geq -b_i \\ \boldsymbol{a}^i \boldsymbol{x} \geq b_i \quad &\rightsquigarrow \quad -\boldsymbol{a}^i \boldsymbol{x} \leq -b_i \end{aligned}$$

- An equation can be replaced by two inequalities:

$$\boldsymbol{a}^i \boldsymbol{x} = b_i \quad \rightsquigarrow \quad \boldsymbol{a}^i \boldsymbol{x} \leq b_i,\ \boldsymbol{a}^i \boldsymbol{x} \geq b_i$$

- An inequality can be replaced by an equation (using an additional *slack variable*) and a non-negativity condition:

$$\begin{aligned} \boldsymbol{a}^i \boldsymbol{x} \leq b_i \quad &\rightsquigarrow \quad \boldsymbol{a}^i \boldsymbol{x} + x_i^s = b_i,\ x_i^s \geq 0 \\ \boldsymbol{a}^i \boldsymbol{x} \geq b_i \quad &\rightsquigarrow \quad \boldsymbol{a}^i \boldsymbol{x} - x_i^s = b_i,\ x_i^s \geq 0 \end{aligned}$$

- A non-positive variable can be replaced by a non-negative variable:

$$x_j \leq 0 \quad \rightsquigarrow \quad x_j := -\overline{x}_j,\ \overline{x}_j \geq 0$$

- A free variable can be replaced by the difference of two non-negative variables:

$$x_j \text{ free} \quad \rightsquigarrow \quad x_j := x_j^+ - x_j^-,\ x_j^+, x_j^- \geq 0$$

Consider for instance the following transformation:

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{x} \leq 0 \quad \rightsquigarrow \quad \boldsymbol{A}'\boldsymbol{x}' = \boldsymbol{b},\ \boldsymbol{x}' \geq 0$$

We obtain $\boldsymbol{A}' = (-\boldsymbol{A}, \boldsymbol{I})$ and $\boldsymbol{x}' = \binom{\overline{\boldsymbol{x}}}{\boldsymbol{x}^s}$ with $\overline{\boldsymbol{x}} \in \mathbb{R}^n$ and $\boldsymbol{x}^s \in \mathbb{R}^m$, where $\boldsymbol{I} \in \{0,1\}^{m \times m}$ denotes the identity matrix of size m.

3.2 Geometric Aspects

Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. We call $\boldsymbol{Ax} \leq \boldsymbol{b}$ a *system of linear inequalities* and $\boldsymbol{Ax} = \boldsymbol{b}$ a *system of linear equations* for the variables $\boldsymbol{x} \in \mathbb{R}^n$.

A set of linear inequalities $\boldsymbol{a}^i \boldsymbol{x} \leq b_i$, $i \in I$, is said to be *linearly independent* if the vectors of coefficients $\boldsymbol{a}^i$, $i \in I$, are linearly independent (analogous for equations).

Let us introduce the following notation: Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $B \subseteq \{1, \ldots, m\}$. $\boldsymbol{A}_B$ denotes the submatrix of $\boldsymbol{A}$ consisting of the rows of $\boldsymbol{A}$ selected by the index set B, and $\boldsymbol{b}_B$ denotes the subvector of $\boldsymbol{b}$ consisting of the components of $\boldsymbol{b}$ selected by the index set B. We assume that the rows of $\boldsymbol{A}_B$ and $\boldsymbol{b}_B$ are arranged in the same order as in the original matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$, i.e. by increasing line numbers. We write

$$\boldsymbol{A}_B = (\boldsymbol{a}^i : i \in B), \quad \text{and} \quad \boldsymbol{b}_B = (b_i : i \in B)$$

Definition 3.1. Let $\boldsymbol{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The set $H = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}^\top \boldsymbol{x} \leq b\}$ is called a *halfspace* and the set $H' = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}^\top \boldsymbol{x} = b\}$ a *hyperplane* in $\mathbb{R}^n$. H' is called the *defining hyperplane* of H.

For a hyperplane $H' = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}^\top \boldsymbol{x} = b\}$, the vector $\boldsymbol{a}$ is called a *normal vector* of H'. Indeed, it is easy to see that $\boldsymbol{a}$ is orthogonal to all vectors parallel to H'. The set of vectors parallel to H' is given by $\{\boldsymbol{x} - \boldsymbol{x}_0 : \boldsymbol{x} \in H'\}$ for some arbitrary $\boldsymbol{x}_0 \in H'$. We obtain $\boldsymbol{a}^\top (\boldsymbol{x} - \boldsymbol{x}_0) = \boldsymbol{a}^\top \boldsymbol{x} - \boldsymbol{a}^\top \boldsymbol{x}_0 = b - b = 0$ for all $\boldsymbol{x} \in H'$, i.e. $\boldsymbol{a}$ is orthogonal to all vectors parallel to H'.

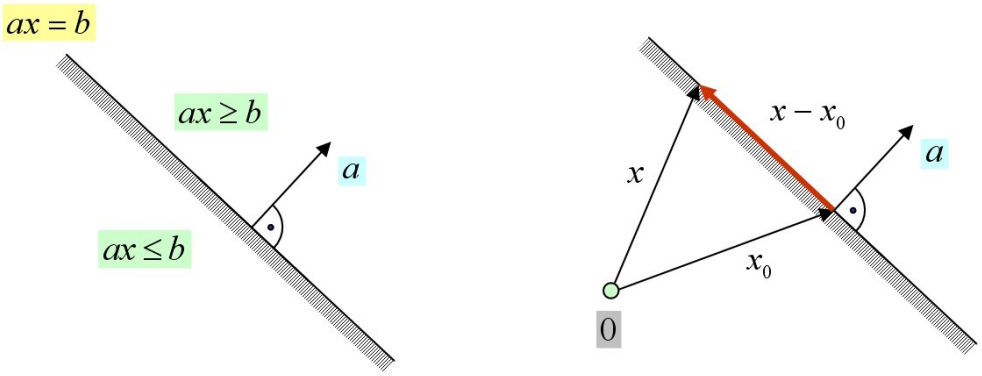


Figure 3.1: Halfspaces and hyperplanes

Definition 3.2. A *polyhedron* P in $\mathbb{R}^n$ is the intersection of a finite number of halfspaces in $\mathbb{R}^n$, i.e.

$$P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{Ax} \leq \boldsymbol{b}\}$$

for some matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and some vector $\boldsymbol{b} \in \mathbb{R}^m$. The sets $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}^i \boldsymbol{x} = b_i\}, i = 1, \ldots, m$, are called the *defining hyperplanes* of P.

Hence, a polyhedron is the solution set of a linear inequality system. Notice that the solution set of a linear equality system is a polyhedron as well, because every system of linear equations can be equivalently written as a system of inequalities:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \qquad \Longleftrightarrow \qquad \begin{pmatrix} \boldsymbol{A} \\ -\boldsymbol{A} \end{pmatrix} \boldsymbol{x} \leq \begin{pmatrix} \boldsymbol{b} \\ -\boldsymbol{b} \end{pmatrix}$$

In fact, the solution set of any linear program is a polyhedron, independently of the form in which the LP is given. If a polyhedron is specified by both equality and inequality constraints (general form), then the defining hyperplanes comprise the given equations and the equations associated to the given inequalities.

Definition 3.3. A *polytope* in $\mathbb{R}^n$ is a bounded polyhedron, i.e. a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ such that there exist $\boldsymbol{\ell}, \boldsymbol{u} \in \mathbb{R}^n$ with

$$P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{\ell} \leq \boldsymbol{x} \leq \boldsymbol{u}\}$$

Figure 3.2 shows a polytope in $\mathbb{R}^2$ with five defining hyperplanes.

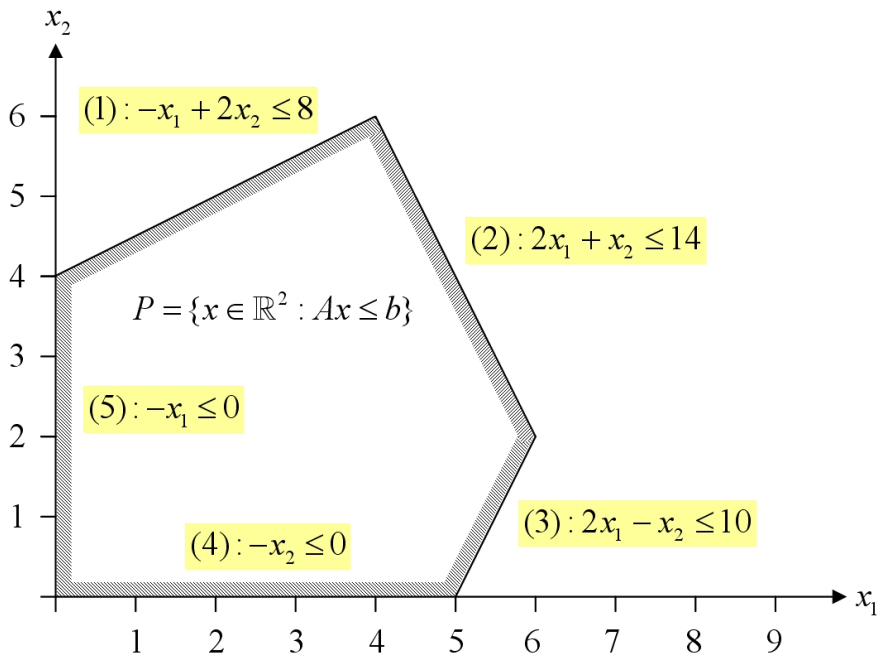


Figure 3.2: Example of a polytope in $\mathbb{R}^2$

Proposition 3.1. *A polyhedron is a convex set.*

Proof. By Theorem 2.2, a linear function is convex. Therefore, by Theorem 2.3, a halfspace of the form $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}^\top \boldsymbol{x} \leq b\}$ is a convex set. Finallly, the intersection of a set of halfspaces is convex by Theorem 2.1. □

Definition 3.4. A point $\boldsymbol{x} \in \mathbb{R}^n$ is a *vertex* of a polyhedron P if the following holds: (i) $\boldsymbol{x} \in P$, and (ii) $\boldsymbol{x}$ cannot be constructed as a strict convex combination of two points in P, i.e. there exist no $\boldsymbol{x}^1, \boldsymbol{x}^2 \in P, \boldsymbol{x}^1 \neq \boldsymbol{x}^2$, such that $\boldsymbol{x} = \lambda \boldsymbol{x}^1 + (1 - \lambda)\boldsymbol{x}^2$ for some $0 < \lambda < 1$.

Proposition 3.2. *A point $\boldsymbol{x}$ is a vertex of a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ with $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ if and only if the following holds: (i) $\boldsymbol{x} \in P$, and (ii) $\boldsymbol{x}$ lies on n linearly independent, defining hyperplanes of P.*

Corollary 3.1. *A point $\boldsymbol{x}$ is a vertex of a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ with $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ if and only if the following holds: (i) $\boldsymbol{x} \in P$, and (ii) there exists a selection $B \subseteq \{1, \ldots, m\}$ of $|B| = n$ linearly independent rows of $\boldsymbol{A}$ such that $\boldsymbol{x} = \boldsymbol{A}_B^{-1}\boldsymbol{b}_B$.*

Note that $\boldsymbol{A}_B \subseteq \mathbb{R}^{n \times n}$ has full row (and column) rank and hence, it is non-singular. Consequently, the inverse $\boldsymbol{A}_B^{-1}$ exists, and the system of equations $\boldsymbol{A}_B\boldsymbol{x} = \boldsymbol{b}_B$ possesses exactly one solution given by $\boldsymbol{x} = \boldsymbol{A}_B^{-1}\boldsymbol{b}_B$.

Corollary 3.1 is an immediate consequence of Theorem 3.2: The condition that $\boldsymbol{x}$ is lying on n linearly independent, defining hyperplanes is equivalent to the condition that $\boldsymbol{x}$ is a solution of the equation system $\boldsymbol{A}_B\boldsymbol{x} = \boldsymbol{b}_B$ where B consists of the indices of the n hyperplanes. The matrix $\boldsymbol{A}_B$ is non-singular as these hyperplanes are linearly independent, hence $\boldsymbol{x} = \boldsymbol{A}_B^{-1}\boldsymbol{b}_B$ is the only solution of this system. Geometrically, this corresponds to the fact that in $\mathbb{R}^2$, $n = 2$ linearly independent lines intersect in exactly one point, and analogously in $\mathbb{R}^3$, $n = 3$ linearly independent planes intersect in exactly one point, and in $\mathbb{R}^n$,n linearly independent planes intersect in exactly one point.

Corollary 3.1 describes the vertices of a polyhedron in an "algebraic" way. In general, the following terminology will be used:

Definition 3.5. Let $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ with $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. A selection $B \subseteq \{1, \ldots, m\}$ of $|B| = n$ linearly independent rows of $\boldsymbol{A}$ is called a *basic selection*, and the corresponding matrix $\boldsymbol{A}_B$ is called a *basis* of $\boldsymbol{A}$. The vector $\boldsymbol{x} = \boldsymbol{A}_B^{-1}\boldsymbol{b}_B$ is called a basic solution of $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ corresponding to the basis $\boldsymbol{A}_B$. B, $\boldsymbol{A}_B$ and $\boldsymbol{x} = \boldsymbol{A}_B^{-1}\boldsymbol{b}_B$ are said to be *feasible* if $\boldsymbol{x} \in P$.

Hence, we have that every feasible basis $\boldsymbol{A}_B$ defines exactly one vertex $\boldsymbol{x} \in P$, and for every vertex $\boldsymbol{x} \in P$ there exists at least one feasible basis $\boldsymbol{A}_B$ such that $\boldsymbol{x}$ is represented by the corresponding basic solution, i.e. such that $\boldsymbol{x} = \boldsymbol{A}_B^{-1}\boldsymbol{b}_B$. There exist polyhedra with vertices $\boldsymbol{x}$ represented by several different feasible basis. This is the case when more than n defining hyperplanes intersect in $\boldsymbol{x}$. Such points are called *degenerated vertices*.

Obviously, there exist polyhedra without vertices. Theorem 3.2 implies that a polyhedron in $\mathbb{R}^n$ can have vertices only if it has at least n linearly independent, defining hyperplanes. Halfspaces and hyperplanes in $\mathbb{R}^n$, $n \geq 2$, are examples of polyhedra without vertices, since their description consists of exactly one defining hyperplane. In

general, a polyhedron of the form $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ possesses vertices if and only if the matrix $\boldsymbol{A}$ has full column rank n, i.e. if $\boldsymbol{A}$ has at least n rows and contains a selection of n linearly independent rows. Polyhedra with vertices are called *pointed*.

Notice that a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ with $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ has at most $\binom{m}{n}$ vertices as this is the maximal number of bases $\boldsymbol{A}$ can have.

Notice further that a basic solution of the form $\boldsymbol{x} = \boldsymbol{A}_B^{-1}\boldsymbol{b}_B$ is not always feasible, i.e. it is possible that $\boldsymbol{x} \notin P$ depending on the situation. Geometrically, this means that $\boldsymbol{x}$ is the intersection of n defining hyperplanes which are intersecting outside of P. In Figure 3.4, for instance, the intersection of hyperplanes (1) and (3) is a basic solution that is not feasible.

Finally, we mention a central theorem, sometimes called the Fundamental Theorem of Linear Programming.

Theorem 3.1. *A LP with solution space P has always an optimal solution at a vertex if (i) P has any vertices, and (ii) the optimum is finite.*

Remember that in the section about convex programming, we stated the following: when maximizing a concave function over a convex solution space, a locally optimal solution is globally optimal. Further, when maximizing a convex function over a convex solution space, all local optimal solutions are located on the boundary of the solution space. A linear function is both convex and concave, and both mentioned properties apply. Hence, when looking for a global optimum a LP, we can restrict ourselves to the boundary of the solution space and stop the search as soon as a local optimum has been found. Theorem 3.1 states even more: when looking for a globally optimal solution it is enough to check all vertices of the solution space. The number of vertices of a polyhedron is finite, thus Linear Programming can be interpreted as a discrete optimization problem with a finite solution space (consisting of all the vertices).

Geometric Solution of Linear Programs in $\mathbb{R}^2$ and $\mathbb{R}^3$

In $\mathbb{R}^2$ (and $\mathbb{R}^3$), Linear Programs can be solved "by hand" using elementary geometric instruments (with corresponding approximate accuracy). The idea is that, in $\mathbb{R}^2$, the level sets $L_\gamma = \{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{c}^\top \boldsymbol{x} = \gamma\}$ of the objective function $f(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x}$ are represented by parallel lines, and the objective value γ grows when shifting these lines in the direction of the normal vector $\boldsymbol{c}$.

Solving a LP in maximization form means to shift the level line L_γ in a parallel way in direction of the vector $\boldsymbol{c}$ as far as possible such that it touches the solution space P at the last possible point, i.e. $L_\gamma \cap P \neq \emptyset$ and $L_{\gamma'} \cap P = \emptyset$ for all $\gamma' > \gamma$. This method is illustrated with an example in Figure 3.3.

In $\mathbb{R}^3$, the method corresponds to a parallel shift of level planes.

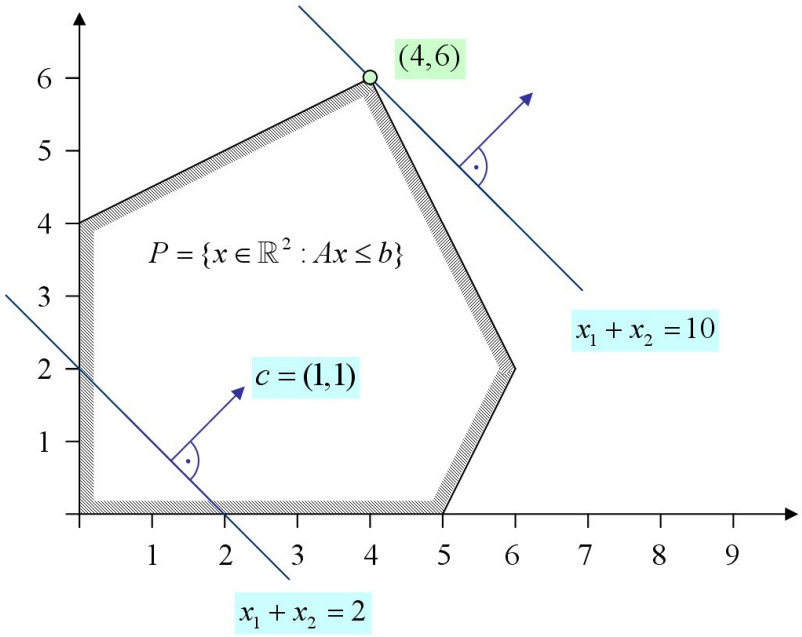


Figure 3.3: Geometrisc solution of a LP in $\mathbb{R}^2$

3.3 Simplex Algorithm

The Simplex Algorithm was developed by George B. Dantzig in 1947–1951, motivated by the goal to solve, among others, certain linear programs arising from planning tasks in the U.S. Air Force. Nowadays, the Simplex Method is one of the crucial technologies in mathematical programming.

In general, the Simplex Algorithm is described for linear programs in standard form, i.e. $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \ge \boldsymbol{0}\}$. In this course however, we shall, for didactic purposes, explain the method for LPs in *inequality form*, i.e. $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}\}$. The advantage of this approach is that geometrical aspects can be illustrated and understood much easier. When discussing the method in standard form, direct geometric visualizations are impossible, since non-trivial examples of polyhedra in standard form with at least two variables and two constraints result in four dimensional problems, after introducing slack variables. A comprehensive presentation of the Simplex Algorithm in inequality form can be found for instance in Grötschel et al. [11].

Consider a linear program of the form

$$\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}\}$$

where we assume that $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ has full column rank. In this case the polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}\}$ has at least one vertex, according to the above discussion.

The Simplex Algorithm starts at a vertex $\boldsymbol{v} \in P$, and performs a test whether there exists an edge starting in $\boldsymbol{v}$, such that going along that edge results in an increase of

the objective function value. If there is no such edge, then the current vertex v is an optimal solution. Otherwise one can – starting at $\boldsymbol{v}$ – follow such an edge, and in this way, obtain higher objective function values. Either we end up in a neighbouring vertex $\boldsymbol{v}'$, and start over again. Or it is possible to follow this edge forever without leaving the polyhedron P, i.e. the problem is unbounded.

For a formal presentation of the Simplex Algorithm we have to consider the following mathematical questions: How can vertices and edges be described algebraically? How does the objective function value change when moving along an edge? How far can we move along an edge?

The following presentation of the Simplex Algorithm gives answers to these questions and defines the method in a formal way.

Simplex Algorithm in Inequality Form:

Input: An instance of a linear program $\Pi : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P\}$ where $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and $\operatorname{rank}(\boldsymbol{A}) = n$. Further, a feasible basic selection $B \subseteq \{1, \ldots, m\}$, $|B| = n$.

Output: An optimal solution at a vertex $\boldsymbol{v} \in P$, if the problem Π is bounded. Otherwise, a direction $\boldsymbol{d} \in \mathbb{R}^n$ along which the objective function grows infinitely.

1. Determine the inverse $\overline{\boldsymbol{A}} := \boldsymbol{A}_B^{-1}$ and the feasible basic solution $\boldsymbol{v} = \overline{\boldsymbol{A}}\boldsymbol{b}_B$.
2. Calculate the "reduced costs" $\boldsymbol{u}^\top = \boldsymbol{c}^\top \overline{\boldsymbol{A}}$.
3. If $\boldsymbol{u}^\top \geq \boldsymbol{0}^\top$, then stop. Vertex $\boldsymbol{v}$ is an optimal solution since

$$\boldsymbol{c}^\top \boldsymbol{v} = \boldsymbol{u}^\top \boldsymbol{A}_B \boldsymbol{v} = \boldsymbol{u}^\top \boldsymbol{b}_B \geq \boldsymbol{u}^\top \boldsymbol{A}_B \boldsymbol{x} = \boldsymbol{c}^\top \boldsymbol{x}, \quad \text{for all } \boldsymbol{x} \in P$$

4. Otherwise, we have $\boldsymbol{u}^\top \not\geq \boldsymbol{0}^\top$. Choose $j \in B$ with $u_j < 0$. Define direction $\boldsymbol{d}$ by

$$\boldsymbol{d} = -\overline{\boldsymbol{A}}_j$$

5. Determine λ^* as the greatest number $\lambda \in \mathbb{R}_0$ fulfilling

$$\boldsymbol{A}(\boldsymbol{v} + \lambda \boldsymbol{d}) \leq \boldsymbol{b}$$

   Notice that the individual inequalities of this system are given by $\boldsymbol{a}^i \boldsymbol{v} + \lambda \boldsymbol{a}^i \boldsymbol{d} \leq b_i$.

6. If $\lambda^* = \infty$, then stop. This is the case if and only if $\boldsymbol{A}\boldsymbol{d} \leq \boldsymbol{0}$. We get

$$\boldsymbol{c}^\top(\boldsymbol{v} + \lambda \boldsymbol{d}) = \boldsymbol{c}^\top \boldsymbol{v} + \lambda \boldsymbol{c}^\top \boldsymbol{d} = \boldsymbol{c}^\top \boldsymbol{v} + \lambda(-\boldsymbol{c}^\top \overline{\boldsymbol{A}}_j) = \boldsymbol{c}^\top \boldsymbol{v} + \lambda(-u_j)$$

   Hence, for $\lambda \to \infty$, it follows $\boldsymbol{c}^\top(\boldsymbol{v} + \lambda \boldsymbol{d}) \to \infty$, i.e. the objective function grows infinitely along direction $\boldsymbol{d}$.

7. Otherwise, we get a finite value $\lambda^* \geq 0$. This is the case if and only if $\boldsymbol{A}\boldsymbol{d} \not\leqslant \boldsymbol{0}$. Clearly, λ^* can be calculated as follows:

$$\lambda^* = \min \left\{ \frac{b_i - \boldsymbol{a}^i \boldsymbol{v}}{\boldsymbol{a}^i \boldsymbol{d}} : i \in \{1, \dots m\},\ \boldsymbol{a}^i \boldsymbol{d} > 0 \right\}$$

   Assume that this minimum is attained at index k. Notice that $k \notin B$, since

$$\boldsymbol{a}^i \boldsymbol{d} = \boldsymbol{a}^i(-\overline{\boldsymbol{A}}_j) = \begin{cases} 0, & i \in B - \{j\} \\ -1, & i = j \end{cases}$$

8. Form the new basic selection B' (perform a "change of basis") by

$$B' = B - \{j\} \cup \{k\}$$

   Notice that B' is a feasible basic selection, and $\boldsymbol{v}' = \boldsymbol{v} + \lambda^* \boldsymbol{d}$ is the corresponding basic solution. Set $B := B'$, and go to Step 1.

□

Example for Simplex Algorithm:

We are supposed to solve the following LP $\Pi : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P\}$ with $P = \{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$:

$$\max 5x_1 + 8x_2 \tag{3.1}$$
$$-3x_1 + 2x_2 \leq 2 \tag{3.2}$$
$$-2x_1 + 5x_2 \leq 16 \tag{3.3}$$
$$6x_1 + 5x_2 \leq 72 \tag{3.4}$$
$$-x_1 \leq 0 \tag{3.5}$$
$$-x_2 \leq 0 \tag{3.6}$$

The input data in matrix notation is given by

$$\boldsymbol{A} = \begin{pmatrix} -3 & 2 \\ -2 & 5 \\ 6 & 5 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 2 \\ 16 \\ 72 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{c} = (5, 8)^\top$$

Consider the feasible basic selection $B = \{1, 2\}$ with assoicated vertex $(2, 4)^\top$. Figure 3.4 shows the solution space P, and the vector $\boldsymbol{c}$ defining the objective function of the LP Π. In the following, we present the iteration of the Simplex Algorithm corresponding to the move from vertex $(2, 4)^\top$ to the neighboring optimal vertex $(7, 6)^\top$.

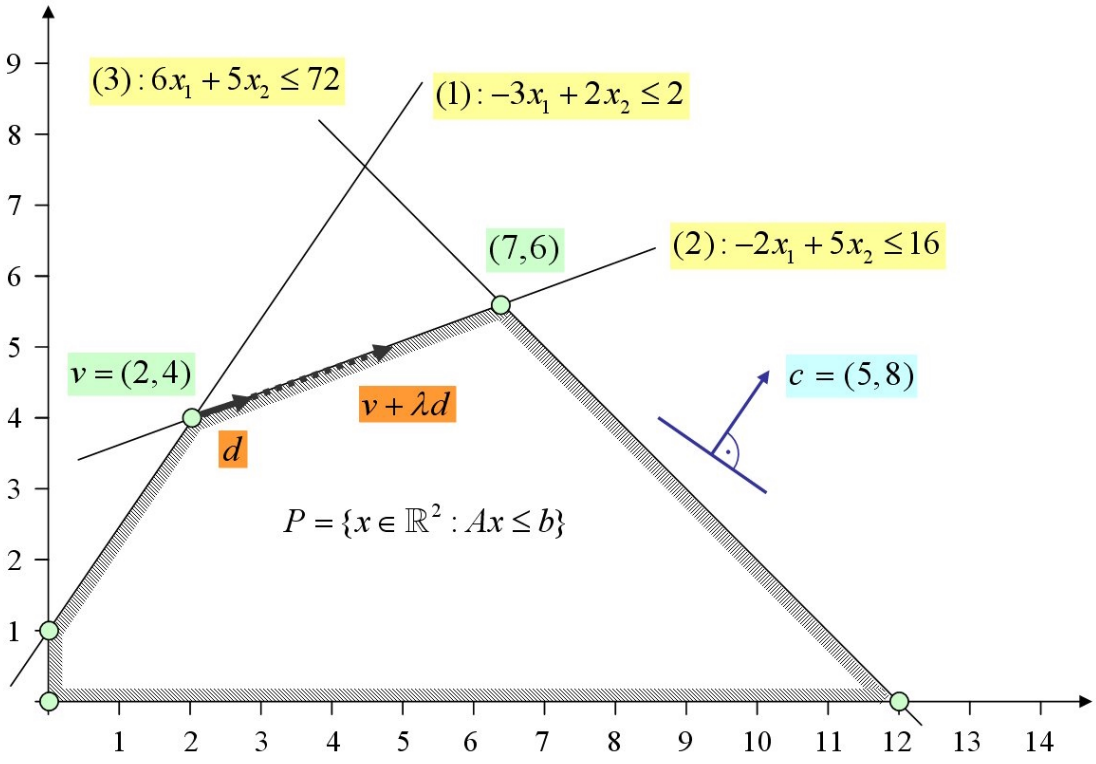


Figure 3.4: Example for Simplex Algorithm

Iteration 1:

1. Basis $\boldsymbol{A}_B$ and corresponding right hand side $\boldsymbol{b}_B$ are given by

$$\boldsymbol{A}_B = \begin{pmatrix} -3 & 2 \\ -2 & 5 \end{pmatrix}, \quad \boldsymbol{b}_B = \begin{pmatrix} 2 \\ 16 \end{pmatrix}$$

   Calculation of the basis inverse $\overline{\boldsymbol{A}} = \boldsymbol{A}_B^{-1}$ and the corresponding feasible basic solution $\boldsymbol{v} = \overline{\boldsymbol{A}}\boldsymbol{b}_B$ yields

$$\overline{\boldsymbol{A}} = \boldsymbol{A}_B^{-1} = \begin{pmatrix} -\frac{5}{11} & \frac{2}{11} \\ -\frac{2}{11} & \frac{3}{11} \end{pmatrix}, \quad \boldsymbol{v} = \overline{\boldsymbol{A}}\boldsymbol{b}_B = \begin{pmatrix} -\frac{5}{11} & \frac{2}{11} \\ -\frac{2}{11} & \frac{3}{11} \end{pmatrix} \begin{pmatrix} 2 \\ 16 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

2. The vector $\boldsymbol{u}^\top = \boldsymbol{c}^\top \overline{\boldsymbol{A}}$ of the "reduced costs" is calculated as

$$\boldsymbol{u}^\top = \boldsymbol{c}^\top \overline{\boldsymbol{A}} = (5, 8) \begin{pmatrix} -\frac{5}{11} & \frac{2}{11} \\ -\frac{2}{11} & \frac{3}{11} \end{pmatrix} = \left(-\frac{41}{11}, \frac{34}{11} \right)$$

3. If $\boldsymbol{u}^\top \geq \boldsymbol{0}^\top$, then stop. In our case, this is not true.

4. We have $\boldsymbol{u}^\top \ngeq \boldsymbol{0}^\top$. Choose $j \in B$ such that $u_j < 0$. The first component of $\boldsymbol{u}^\top$ (corresponding to the first column of $\overline{\boldsymbol{A}} = \boldsymbol{A}_B^{-1}$ and the first row of A_B) is the only possible choice in this case, hence $j = 1$. Define the corresponding direction

$$\boldsymbol{d} = -\overline{\boldsymbol{A}}_1 = \begin{pmatrix} \frac{5}{11} \\ \frac{2}{11} \end{pmatrix}$$

Notice the meaning of this vector in Figure 3.4.

5. Determine λ^* defined as the greatest number $\lambda \in \mathbb{R}_0$ such that $\boldsymbol{A}(\boldsymbol{v}+\lambda\boldsymbol{d}) \leq \boldsymbol{b}$, i.e.

$$\boldsymbol{A}(\boldsymbol{v}+\lambda\boldsymbol{d}) = \boldsymbol{A}\boldsymbol{v} + \lambda\boldsymbol{A}\boldsymbol{d} = \begin{pmatrix} -3 & 2 \\ -2 & 5 \\ 6 & 5 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \end{pmatrix} + \lambda \begin{pmatrix} -3 & 2 \\ -2 & 5 \\ 6 & 5 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \frac{5}{11} \\ \frac{2}{11} \end{pmatrix} =$$

$$= \begin{pmatrix} 2 \\ 16 \\ 32 \\ -2 \\ -4 \end{pmatrix} + \lambda \begin{pmatrix} -1 \\ 0 \\ \frac{40}{11} \\ -\frac{5}{11} \\ -\frac{2}{11} \end{pmatrix} \leq \begin{pmatrix} 2 \\ 16 \\ 72 \\ 0 \\ 0 \end{pmatrix} = \boldsymbol{b}$$

6. If $\lambda^* = \infty$, then stop. This is the case if and only if $\boldsymbol{A}\boldsymbol{d} \leq \boldsymbol{0}$. As we see this is not true in our case.

7. Thus, we get a finite value for $\lambda^* \geq 0$. This value can be determined in the following way:

$$\lambda^* = \min\left\{ \frac{b_i - \boldsymbol{a}^i\boldsymbol{v}}{\boldsymbol{a}^i\boldsymbol{d}} : i \in \{1,\ldots,m\}, \boldsymbol{a}^i\boldsymbol{d} > 0 \right\} =$$
$$= \min\left\{ \frac{b_3 - \boldsymbol{a}^3\boldsymbol{v}}{\boldsymbol{a}^3\boldsymbol{d}} \right\} = \min\left\{ \frac{72-32}{\frac{40}{11}} \right\} = 11$$

The minimum is attained for $k = 3$. Notice that $3 \notin B$.

8. Form the new feasible basic selection B' as follows:

$$B' = B - \{j\} \cup \{k\} = \{1,2\} - \{1\} \cup \{3\} = \{2,3\}$$

The corresponding basic solution $\boldsymbol{v}'$ is given by

$$\boldsymbol{v}' = \boldsymbol{v} + \lambda^*\boldsymbol{d} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} + 11 \begin{pmatrix} \frac{5}{11} \\ \frac{2}{11} \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \end{pmatrix}$$

Set $B := B' = \{2,3\}$, and go to Step 1.

Iteration 2:

1. Basis $\boldsymbol{A}_B$ and corresponding right hand side $\boldsymbol{b}_B$ are given by

$$\boldsymbol{A}_B = \begin{pmatrix} -2 & 5 \\ 6 & 5 \end{pmatrix}, \quad \boldsymbol{b}_B = \begin{pmatrix} 16 \\ 72 \end{pmatrix}$$

   Calculation of the basis inverse $\overline{\boldsymbol{A}} = \boldsymbol{A}_B^{-1}$ and the corresponding feasible basic solution $\boldsymbol{v} = \overline{\boldsymbol{A}}\boldsymbol{b}_B$ yields (compare Step 8 of Iteration 1):

$$\overline{\boldsymbol{A}} = \boldsymbol{A}_B^{-1} = \begin{pmatrix} -\frac{1}{8} & \frac{1}{8} \\ \frac{3}{20} & \frac{1}{20} \end{pmatrix}, \quad \boldsymbol{v} = \overline{\boldsymbol{A}}\boldsymbol{b}_B = \begin{pmatrix} -\frac{1}{8} & \frac{1}{8} \\ \frac{3}{20} & \frac{1}{20} \end{pmatrix} \begin{pmatrix} 16 \\ 72 \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \end{pmatrix}$$

2. The vector $\boldsymbol{u}^\top = \boldsymbol{c}^\top \overline{\boldsymbol{A}}$ of the "reduced costs" is calculated as

$$\boldsymbol{u}^\top = \boldsymbol{c}^\top \overline{\boldsymbol{A}} = (5, 8) \begin{pmatrix} -\frac{1}{8} & \frac{1}{8} \\ \frac{3}{20} & \frac{1}{20} \end{pmatrix} = \left(\frac{23}{40}, \frac{41}{40} \right)$$

3. If $\boldsymbol{u}^\top \geq \boldsymbol{0}^\top$, then stop. This is true in our case. Hence, we have found the following optimal solution $\boldsymbol{v}$ with corresponding optimal objective value $f(\boldsymbol{v})$:

$$\boldsymbol{v} = \begin{pmatrix} 7 \\ 6 \end{pmatrix}, \quad f(\boldsymbol{v}) = \boldsymbol{c}^\top \boldsymbol{v} = (5, 8) \begin{pmatrix} 7 \\ 6 \end{pmatrix} = 83$$

□

4 Integer Linear Programming

In the following we assume that $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^n$, if not stated otherwise.

4.1 Definitions and Concepts

4.1.1 Problem Formulation

Integer Linear Programming — similarly to Linear Programming — considers the problem to optimize a *linear objective function* subject to a finite number of *linear constraints* (i.e. linear inequalities or linear equations), now with the additional condition that the values of some or all variables have to be *integer numbers*.

According to Chapter 3, the solution space of a Linear Program is a polyhedron. A general description of a LP is given by

$$\text{LP: } \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P\} \quad \text{with } P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$$

Hence, the solution space of an *Integer Linear Program (ILP)* is defined as the set of all integer vectors of a polyhedron, and can be written as

$$\text{ILP: } \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\} \qquad \text{with } P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$$

Integer Linear Programming is often referred to, in short, as *Integer Programming (IP)*. Figure 4.1 shows an ILP in $\mathbb{R}^2$.

Binary (Linear) Programming (BLP, 0-1 (Linear) Programming) is a special type of Integer Programming where all variables are allowed to attain the values 0 or 1 only. A Binary Linear Program can be written as follows:

$$\text{BLP: } \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \{0,1\}^n\} \quad \text{with } P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$$

We say that we are dealing with *Mixed Integer (Linear) Programming (MILP or MIP)* in the case where just a part of the variables is supposed to have integer values. A Mixed Integer Program can be written in the following form:

$$\text{MIP: } \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}_K^n\} \quad \text{with } P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$$

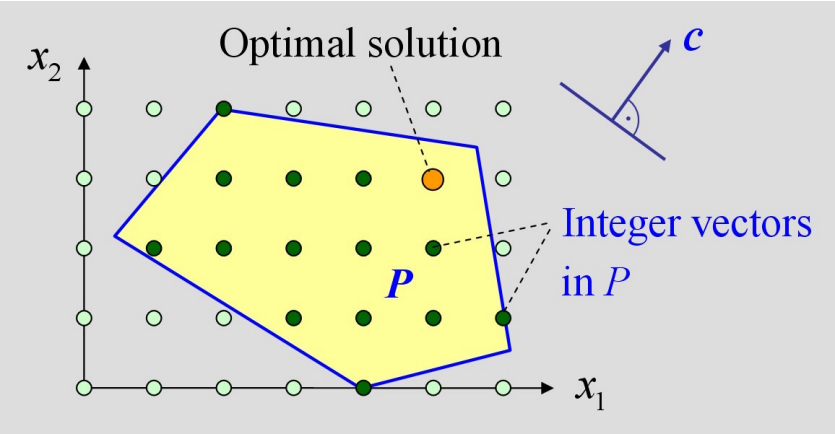


Figure 4.1: Integer Programming (ILP) in $\mathbb{R}^2$

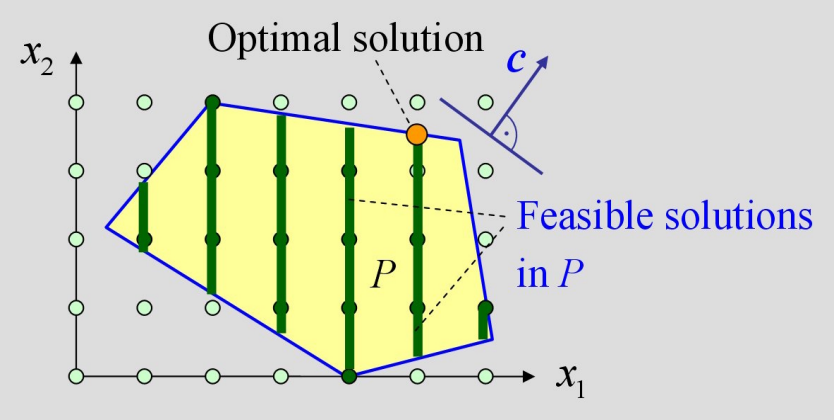


Figure 4.2: Mixed Integer Programming (MIP) in $\mathbb{R}^2$

Here, $K \subseteq \{1, \ldots, n\}$ denotes the index set of the integer variables and $\mathbb{Z}^n_K = \{\boldsymbol{x} \in \mathbb{R}^n : x_j \in Z \text{ for } j \in K\}$. Sometimes, Mixed Integer Programming is simply referred to as Integer Programming, too. Figure 4.2 shows a MIP in $\mathbb{R}^2$ where $x_1 \in \mathbb{Z}$ is assumed to be an integer variable and $x_2 \in \mathbb{R}$ is assumed to be a real variable.

In the following, we focus entirely on (pure) Integer Programming (ILP). However, large parte of the theoretical foundations and algorithmic approaches apply both to ILP and MIP, and many mathematical results can easily be transferred from ILP to MIP.

4.1.2 Application and Methodology

Integer Programming plays a crucial role when modelling complex, practice-oriented applications of Operations Research. A big portion of problem settings originating in Operations Management (for example, production, logistics, services) can be modelled with help of ILP. In other words, ILP achieved the meaning of a 'universal' modelling language for many mathematical optimization problems.

The modelling with binary 0-1-variables plays a central role as binary variables can be used to model yes/no decisions appearing very often in practice.

It should be pointed out that solving an Integer Linear Program is much more difficult than solving a Linear Program. The Integrality condition changes the mathematical structure of the optimization problem dramatically; hence, completely different solution approaches are needed when compared to the continuous case.

With regard to complexity, Integer Programs belong to a completely different class than Linear Programs. Linear Programs belong to the class of polynomially solvable problems whereas Integer Programs belong to the class of NP-complete problems (for which no polynomial algorithms are known). Here, we say that an algorithm is polynomial if the number of necessary calculation steps is bounded by a polynomial of the size of the problem instance, i.e. size of the necessary input. For further details regarding complexity theory we refer to specialist literature.

Despite the difficulties when solving Integer Programs, nowadays there are very high-performing ILP solvers. Such solvers are—thanks to deep and profound mathematical techniques—able to solve very big ILPs with tens of thousands of variables and constraints almost to optimality in a few minutes. These days, GUROBI (www.gurobi.com) and CPLEX (www.ibm.com) belong to the most successful and efficient ILP solvers. When looking for non-commercial solvers one can use the GLPK solver or LPSOLVE (licensed under GNU) or the SCIP solver from Konrad-Zuse-Zentrum in Berlin.

To a layperson it is often incomprehensible that solving Integer Programs is much more difficult than solving Linear Programs. There is a widespread view that an almost optimal solution of an ILP can be found easily just by solving the corresponding LP (without the integrality condition) and rounding up/down the found optimal solution. This argumentation ignores the problem that when using rounding it may not be possible to find a feasible solution of the ILP at all.

Figure 4.3 shows an illustration of this effect. The depicted example shows clearly that we can't obtain a feasible solution by rounding. At the same time, the distance between the optimal solution of the ILP and the optimal solution of the LP can be arbitrarily large.

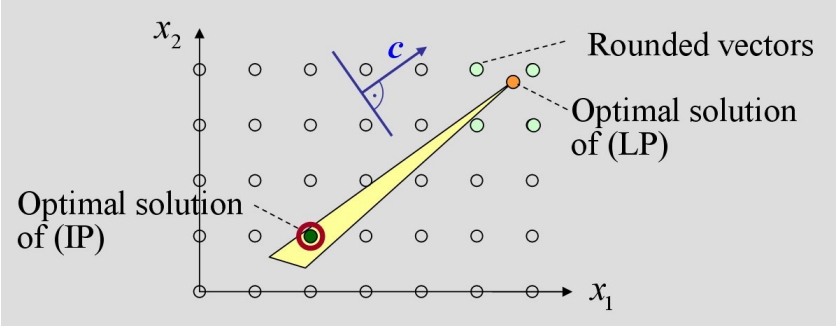


Figure 4.3: Naive rounding is not a solution approach for ILP

In case of 0-1 variables, the rounding problem becomes even more pronounced: by rounding up/down, every possible solution can be generated. Hence, the optimal solution of the LP found in this way has no direct value for the construction of the optimal solution of the ILP.

Two underlying methodological approaches can be used for solving of integer programs: (i) *Branch-and-Bound* and (ii) *Cutting Planes* . We are going to discuss in detail both of them. At first, we need to introduce further elementary notions and concepts.

4.1.3 Relaxations

So-called *relaxations* play often an important role when solving optimization problems.

Definition 4.1. Consider the optimization problem $\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$. A problem of the form $\Pi' : \max\{f'(\boldsymbol{x}) : \boldsymbol{x} \in S'\}$ is *a relaxation* of Π if $S \subseteq S'$ and $f'(\boldsymbol{x}) \geq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in S$.

Often, relaxations with $f'(\boldsymbol{x}) = f(\boldsymbol{x})$ are considered. In such a case, the relaxation means just that the solution space becomes bigger. This is often achieved by disregarding certain constraints of the original problem.

In particular, relaxations can be used to determine *bounds* for optimal values of optimization problems. One is interested in *upper bounds* when dealing with a maximization problem, and in *lower bounds* for minimization problems. In the following discussion, we restrict ourselves to maximization problems. Minimization problems can be treated analogously.

An *(upper) bound* for a maximization problem $\max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ is a value $z^{UB} \in \mathbb{R}$ such that

$$z^{UB} \geq \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}.$$

Hence, an upper bound z^{UB} is a result of an "optimistic" estimate for the maximum, i.e. the maximum cannot exceed the value z^{UB}.

Clearly, an optimum of a relaxation gives a bound for the original problem as is shown in the next theorem.

Proposition 4.1. *Let* $\Pi' : \max\{f'(\boldsymbol{x}) : \boldsymbol{x} \in S'\}$ *be a relaxation of* $\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$. *Then*

$$\max\{f'(\boldsymbol{x}) : \boldsymbol{x} \in S'\} \geq \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$$

Proof. Let $\boldsymbol{x}'$ be an optimal solution of Π' and let $\boldsymbol{x}^*$ be an optimal solution of Π. As $S \subseteq S'$ we get $\boldsymbol{x}^* \in S'$, hence $f'(\boldsymbol{x}') \geq f'(\boldsymbol{x}^*) \geq f(\boldsymbol{x}^*)$. □ □

When applying relaxations, the main idea is to try to find, for an optimization problem Π, which is "difficult" to solve, a relaxation Π', which is "easy" to solve. By means of this relaxation, one would like to estimate the maximum of Π by a good upper bound. Furthermore, an upper bound z^{UB} can possibly be used to show the optimality of a known feasible solution $\boldsymbol{x}^*$. Namely, $\boldsymbol{x}^*$ is optimal if $f(\boldsymbol{x}^*) = z^{UB}$ (however, the opposite implication is not true in general).

The quality of a bound coming from a relaxation is determined by the quality of the relaxation itself, i.e. one is interested in relaxations Π' where the optimum of Π' is as near as possible to the optimum of Π. For the case of maximization it means, the lower the resulting upper bound, the better the corresponding relaxation.

If considering relaxations with an objective function f' being "identical" with the original objective function f, i.e. $f'(\boldsymbol{x}) = f(\boldsymbol{x})$ for $\boldsymbol{x} \in S$, we can say that relaxation Π'' is *better* than relaxation Π' if $S'' \subset S'$, because we have

$$\max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S''\} \leq \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S'\} \quad \text{if } S'' \subset S'$$

The so-called *LP-relaxation* is important when dealing with integer programs. This relaxation is given by the Linear Program obtained from the original Integer Program by removing the integrality condition.

Definition 4.2. The *LP relaxation* of an integer program $\Pi : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\}$ is given by

$$\Pi^{LP} : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P\}$$

The LP relaxation plays a central role for most solution methods in Integer Programming. In particular, this is the case for the elementary methods (i) *Branch-and-Bound*, and (ii) *Cutting Planes*, which will be discussed later. Figure 4.4 shows an ILP and its LP relaxation in $\mathbb{R}^2$ along with the corresponding optimal solutions.

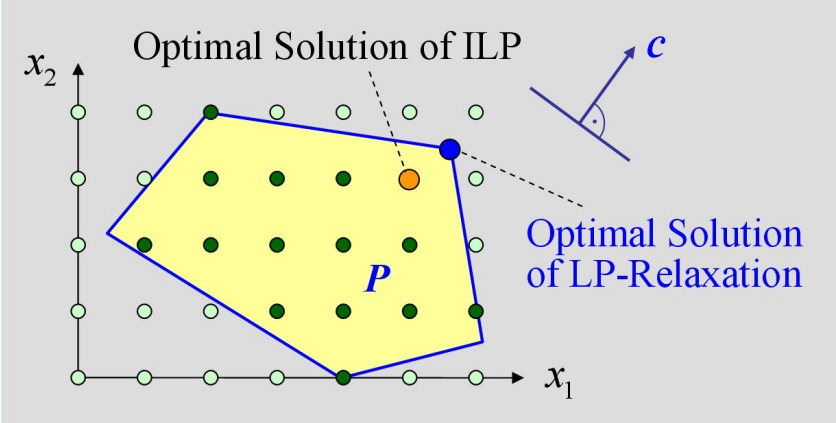


Figure 4.4: ILP and associated LP relaxation

4.2 Branch-and-Bound Method

4.2.1 General Branch-and-Bound Method

Branch-and-Bound (B&B) is a general approach for the exact solution of optimization problems which, in principle, can be applied to any type of problem. However, in general, the termination of a B&B algorithm after a finite number of steps is only ensured for optimization problems with a finite solution space. In the following, the principle of Branch-and-Bound is explained for maximization problems. In case of minimization problems the arguments are analogous.

The idea of Branch-and-Bound is to split the original optimization problem step-by-step into "subproblems" and to try to solve these. If a subproblem cannot be solved, then it well again be split into subproblems. This process will be continued until every generated subproblem either (i) is solved, or (ii) can be neglected because it can be shown that it does not contain an optimal solution of the original problem (or is infeasible).

Branching

The partitioning of the original problem $\Pi^0 = \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$ into subproblems $\Pi^1, \ldots, \Pi^k$ is usually done by dividing the solution space S into subsets $S^1, \ldots, S^k$ such that $\bigcup_{q=1}^{k} S^q = S$, i.e. every solution $\boldsymbol{x} \in S$ is contained in at least one of the subsets $S^1, \ldots, S^k$. In general, these subsets are chosen such that they form a partition of S, i.e. such that $S^p \cap S^q = \emptyset$ for $p, q \in \{1, \ldots, k\}$, $p \neq q$. In this case, every solution $\boldsymbol{x} \in S$ is contained in exactly one subset.

The resulting subproblems are given by $\Pi^q : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S^q\}$, for $q = 1, \ldots, k$. The partitioning of these subproblems into further subproblems is based by the same principle.

The step-by-step partition of the solution space S and the corresponding creation of the subproblems is often represented in form of a branching tree (see Figure 4.5).

The tree nodes correspond to different subproblems; an arrow from node p to node q means that the subproblem Π^q is one of the lower-level subproblems created by partitioning the higher-level subproblem Π^p.

All the nodes without a successor node in the branching tree have not been partitioned yet. They are either (i) solved optimally or (ii) to be neglected or to be 'processed' further. Nodes to be processed further are called *open nodes*.

The process of step-by-step *partitioning* of a subproblem into lower-level subproblems is called '*Branching*' and is one of the ingredients of Branch-and-Bound.

Notice that, for a 'sensible' branching concept, the solution spaces of the subproblems become smaller and smaller. For the case of a finite solution space S, the solution spaces of subproblems which cannot be neglected consist just of a single solution after finitely

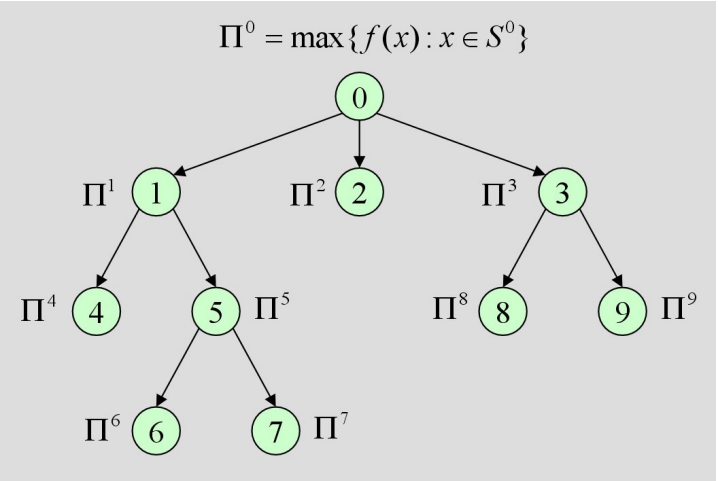


Figure 4.5: Schematic branching tree of a Branch-and-Bound method

many branching steps. In such a situation, the optimal solution of the corresponding subproblem is trivial. Hence, the B&B method terminates in the case of a finite solution space.

In the extreme case when no subproblems can be neglected according to (ii), the B&B-method corresponds to a complete enumeration of all possible solutions in S. Branch-and-Bound is often denoted as an *intelligent* or *implicit enumeration method*. The word intelligent is used to emphasize that—typically—not all solutions are enumerated; the idea is to find as many subproblems as possible which can be neglected (such that we do not need to investigate the corresponding solutions).

Notice that, for discrete optimization problems of a size relevant for real world applications, an optimization method based on complete enumeration rarely represents a viable approach. For example, the number of solutions of a Binary Linear Program with 1000 variables is of order 2^{1000}; this number is greater than the estimated number of atoms in the universe (ca. 10^{70}).

Bounding

The second ingredient (and the core idea) of Branch-and-Bound is the so-called Bounding described in the following. When processing a subproblem Π^r, one (i) calculates an *upper bound* z_r^{UB} for Π^r and at the same time (ii) one tries to find a good solution $\boldsymbol{x}_r^{LB}$ heuristically. Notice that $z_r^{LB} := f(\boldsymbol{x}_r^{LB})$ gives a lower bound for Π^r, i.e. it holds

$$z_r^{UB} \geq \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S^r\} \geq z_r^{LB}$$

During the entire B&B process, the best solution found so far is saved in a "global" variable $\boldsymbol{x}^{LB}$ (called "best incumbent solution"), together with the corresponding best value of the objective function $z^{LB} := f(\boldsymbol{x}^{LB})$. If a better solution $\boldsymbol{x}_r^{LB}$ for the subproblem Π^r (with the value z_r^{LB}) can be found, then these global variables have to be adjusted.

The crucial step of the method is the following one: If for a subproblem Π^r, the calculated upper bound z_r^{UB} fulfils

$$z_r^{UB} \leq z^{LB}$$

then the subproblem Π^r can be neglected. In fact, because z_r^{UB} is an upper bound for Π^r, we get

$$\max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S^r\} \leq z_r^{UB} \leq z^{LB} = f(\boldsymbol{x}^{LB})$$

i.e., all solutions of Π^r are worse than (or at most as good as) the best solution $\boldsymbol{x}^{LB}$ found so far. Hence, it is not necessary to process this subproblem further, and Π^r can be neglected. This "cutting off" of the node Π^r in the branching tree is called *Pruning* or, more exactly, *Pruning by Dominance* (as the bound of the subproblem is dominated by the best solution found so far).

There exist other reasons for *Pruning* of a node. A subproblem Π^r can also be neglected if (i) it is infeasible, i.e. the solution space is empty, or if (ii) it was possible to solve it optimally, with optimum z_r^{LB}; typically, optimality is proven by $z_r^{LB} = z_r^{UB}$.

The difference between a "non-intelligent" complete enumeration and an "intelligent" Branch-and-Bound method is mainly given by the fact that, in B&B, certain subproblems can be neglected thanks to Pruning; hence, usually, just a very small part of possible solutions have to be evaluated directly.

Summing up, we can say that three elements are needed for a Branch-and-Bound method:

- A method for calculation of an upper bound for any subproblem Π^r.
- A method for heuristic determination of a good feasible solution for any subproblem Π^r.
- A mechanism for partitioning a subproblem Π^r into lower-level subproblems with "the same structure". Here, asking for the same structure means that the methods for calculation a bound and for finding a feasible solution can be applied to any subproblem.

When considering the development difficulty of these elements one can say that typically, the calculation of good bounds is the mathematically most challenging part. A deep investigation of mathematical problem structures and the development of sophisticated algorithms is often needed to achieve this. Very often the principle of relaxations is used to calculate the bounds. Another approach is to use principles from duality theory.

The heuristic search for good feasible solutions in the subproblems can be very challenging as well. There are a number of general and problem specific approaches which can help to cope with this task.

Usually, the development of branching mechanisms for partitioning of the problems into subproblems is mathematically less demanding.

A formal specification of a general Branch-and-Bound method is given by the following pseudo code.

A General Branch-and-Bound Algorithm:

Input: An instance of an optimization problem of the form $\Pi : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$. We assume that Π is bounded.

Output: If $z^{LB} > -\infty$, then z^{LB} is the optimum and $\boldsymbol{x}^{LB}$ is an optimal solution of Π, otherwise Π is infeasible.

1. *Initialisation:* Set $\widehat{r} := 0$, $S^0 := S$, $\Pi^0 := \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S^0\}$ (i.e. $\Pi^0 = \Pi$), $R := \{0\}$ and $z^{LB} := -\infty$.
2. *Termination:* If $R = \emptyset$, then stop.
3. *Node selection :* Select a node $r \in R$ and set $R := R - \{r\}$.
4. *Bound calculation and heuristic solution:*
   (i) Calculate an upper bound z_r^{UB} for the problem Π^r. If Π^r is infeasible, then set $z_r^{UB} := \infty$ and go to 6.
   (ii) Look heuristically for a feasible solution $\boldsymbol{x}_r^{LB}$ of Π^r. If a solution is found, then set $z_r^{LB} := f(\boldsymbol{x}_r^{LB})$, otherwise $z_r^{LB} := -\infty$.
5. *Global improvement:* If $z_r^{LB} > z^{LB}$, then set $z^{LB} := z_r^{LB}$ and $\boldsymbol{x}^{LB} := \boldsymbol{x}_r^{LB}$.
6. *Pruning:* If one of the following cases occurs, then go to 2:
   Case 1 (Infeasibility): $z_r^{UB} = \infty$.
   Case 2 (Optimality): $z_r^{LB} = z_r^{UB}$.
   Case 3 (Dominance): $z_r^{UB} \leq z^{LB}$.
7. *Branching:* Divide S^r into $k \geq 2$ subsets $S^{\widehat{r}+1}, S^{\widehat{r}+2}, \ldots, S^{\widehat{r}+k}$ such that $S^q \subseteq S^r$ for $q \in R^r := \{\widehat{r}+1, \ldots, \widehat{r}+k\}$ and $\bigcup_{q \in R^r} S^q = S^r$. Save the new subproblems $\Pi^q : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S^q\}, q \in R^r$. Set $R := R \cup R^r$ and $\widehat{r} := \widehat{r} + k$. Go to 2.

□

4.2.2 Branch-and-Bound Method for Integer Linear Programming

In most cases integer linear programs are solved using a Branch-and-Bound approach. Usually, one makes use of the following three elements:

- LP relaxations are used to determine bounds. These can be solved with an LP method efficiently.
- In order to find a feasible solution one can use the fact that an optimal solution of a LP relaxation is a feasible solution of the ILP if it is integer. In this case the corresponding subproblem is solved optimally, hence we don't need to process it further. Additionally to this approach, efficient B&B methods usually use further heuristic methods in order to find good solutions of the subproblems.
- Typically, the branching in a subproblem works as follows. A branching step is only necessary if the found optimal solution $\boldsymbol{x}^{LP}$ of the LP relaxation is not integer. In

such a case on has to choose a non-integer component x_j^{LP} of this solution and to produce two subproblems by adding one of the following two constraints, respectively:

$$x_j \leq \lfloor x_j^{LP} \rfloor$$
$$x_j \geq \lfloor x_j^{LP} \rfloor + 1$$

Here, for $y \in \mathbb{R}$, we denote by $\lfloor y \rfloor$ the greatest integer number which is less or equal to y (floor of y). Apparently, the union of the solution spaces of the newly added problems contains all integer solutions of the higher-level problem; the non-integer vector x^{LP} is eliminated.

The following pseudo code describes a formal specification of an elementary Branch-and-Bound Algorithm.

Elementary Branch-and-Bound Algorithm for ILP:

Input: An instance of an ILP in the form $\Pi : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\}$ with $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$. It is assumed that Π is bounded.

Output: If $z^{LB} > -\infty$, then z^{LB} is the optimum and $\boldsymbol{x}^{LB}$ is an optimal solution of Π, otherwise Π is infeasible.

1. *Initialisation:* Set $\widehat{r} := 0$, $P^0 := P$, $S^0 := P \cap \mathbb{Z}^n$, $\Pi^0 := \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in S^0\}$ (i.e. $\Pi^0 = \Pi$), $R := \{0\}$ and $z^{LB} := -\infty$.
2. *Termination:* If $R = \emptyset$, then stop.
3. *Node selection:* Choose a node $r \in R$ and set $R := R - \{r\}$.
4. *Bound calculation and heuristic solution:*
   (i) Calculate an upper bound z_r^{UB} for the problem Π^r:
   Solve the LP relaxation $\Pi_{LP}^r : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P^r\}$ with the resulting optimum z^{LP} and an optimal solution $\boldsymbol{x}^{LP}$. If Π_{LP}^r is infeasible, then set $z_r^{UB} := \infty$ and go to 6. Otherwise set $z_r^{UB} := z^{LP}$.
   (ii) Look for a feasible solution $\boldsymbol{x}_r^{LB}$ of Π^r heuristically:
   If $\boldsymbol{x}^{LP} \in \mathbb{Z}^n$, then $\boldsymbol{x}_r^{LB} := \boldsymbol{x}^{LP}$ is an optimal solution of the ILP Π^r. In this case, set $z_r^{LB} := f(\boldsymbol{x}^{LP})$ such that $z_r^{LB} = z^{LP} = z_r^{UB}$, otherwise $z_r^{LB} := -\infty$.
5. *Global improvement:* If $z_r^{LB} > z^{LB}$ the set $z^{LB} := z_r^{LB}$ and $\boldsymbol{x}^{LB} := \boldsymbol{x}_r^{LB}$.
6. *Pruning:* In one of the following cases go to 2:
   Case 1 (Infeasibility): $z_r^{UB} = \infty$.
   Case 2 (Optimality): $z_r^{LB} = z_r^{UB}$.
   Case 3 (Dominance): $z_r^{UB} \leq z^{LB}$.
7. *Branching:* Divide the set S^r into $k = 2$ subsets $S^{\widehat{r}+1}$ and $S^{\widehat{r}+2}$:
   As $\boldsymbol{x}^{LP} \notin \mathbb{Z}^n$ (because of $z_r^{LB} < z_r^{UB}$), there exists an index j such that $x_j^{LP} \notin \mathbb{Z}$. Set:

$$P^{\widehat{r}+1} = \{\boldsymbol{x} \in P^{\widehat{r}} : x_j \leq \lfloor x_j^{LP} \rfloor\},$$
$$P^{\widehat{r}+2} = \{\boldsymbol{x} \in P^{\widehat{r}} : x_j \geq \lfloor x_j^{LP} \rfloor + 1\}.$$

Set $S^q := P^q \cap \mathbb{Z}^n$ for $q \in R^r := \{\widehat{r}+1, \widehat{r}+2\}$ and save the new subproblems $\Pi^q : \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in S^q\}, q \in R^r$. Set $R := R \cup R^r$ and $\widehat{r} := \widehat{r} + k$. Go to 2.

□

4.2.3 Example: 0-1 Knapsack Problem

The Branch-and-Bound Method for Integer Linear Programming shall be illustrated on the example of the so-called *Knapsack Problem.*

The Knapsack Problem (more precisely 0-1 Knapsack Problem) corresponds to the following task:

A finite number of objects $j \in J$ are given. Every object $j \in J$ has a certain volume $a_j \geq 0$ and a certain benefit $c_j \geq 0$. The task is to fill the knapsack with capacity b by a selection of the given objects such that the total volume of the selected objects doesn't exceed the capacity of the knapsack and at the same time the total benefit becomes maximal.

The 0-1 Knapsack Problem can be formulated as an ILP very easily. For every object $j \in J$ we introduce a binary variable x_j with value 1 if the corresponding object is put into the knapsack and with value 0 otherwise. The ILP formulation reads as:

$$\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{a}^\top \boldsymbol{x} \leq b, \boldsymbol{x} \in \{0,1\}^J\}$$

where $\boldsymbol{a} = (a_j : j \in J)$ and $\boldsymbol{c} = (c_j : j \in J)$. Actually, the 0-1 Knapsack Problem is the simplest possible binary ILP, composed only of an objective function and a single constraint (assuming that all parameters are non-negative).

In contrast to the 0-1 Knapsack Problem, the general Knapsack Problem allows to put objects more than once into the knapsack, i.e. in the above ILP formulation the condition $\boldsymbol{x} \in \{0,1\}^J$ is replaced by $\boldsymbol{x} \in \mathbb{Z}_0^J$.

Example: Renovation of Branch Offices

Let us consider the following example as an application of the 0-1 Knapsack Problem.

A retail business company is planning the renovation of its five branch offices. The allowed renovation budget is 3.7 million francs. The costs (in 100'000 francs) for renovating the branches are given in Table 4.1. The benefit resulting from the renovation of a branch is measured by the number of additional customer visits per year expected for the renovated branch. These expected numbers (in thousand persons per year) are also given in Table 4.1. It must be noted that renovations have to be carried out completely, i.e. no partial renovations are allowed.

Branch:	1	2	3	4	5
Costs (in 100'000 Fr.):	17	16	21	8	12
Benefit (in 1000 persons/year):	11	14	16	8	7

Table 4.1: Costs and benefit of renovations

The task can be formulated as a Knapsack Problem as follows:

Sets:

J Set of the branches, $J = \{1, \ldots, n\}$.

Parameters:

a_j Costs (in 100'000 Fr.) for renovation of branch j, $j \in J$.
b Total budget (in 100'000 Fr.).
c_j Expected additional customer visits (in 1000 persons per year) in a renovated branch j, $j \in J$.

Variables:

x_j Binary indicator with value 1 if and only if branch j is going to be renovated, $j \in J$.

Objective function and constraints:

$$\begin{aligned} &\max \sum_{j \in J} c_j x_j \\ &\sum_{j \in J} a_j x_j \leq b \\ &x_j \in \{0, 1\}, \qquad\qquad j \in J \end{aligned}$$

The Knapsack Problem corresponding to the example reads as follows (if explicitly formulated):

$$\begin{aligned} \max 11x_1 + 14x_2 + 16x_3 + 8x_4 + 7x_5 \\ 17x_1 + 16x_2 + 21x_3 + 8x_4 + 12x_5 \leq 37 \\ x_1, \ldots, x_5 \in \{0, 1\} \end{aligned}$$

In the following, we are going to develop a simple, general B&B method for the 0-1 Knapsack Problem, and to illustrate it on the example. The basis for the method is the elementary B&B algorithm for ILP explained above.

When developing a B&B method, we have to address the three mentioned elements: (1) calculation of bounds, (2) heuristic solution search, and (3) definition of subproblems (branching principle). We discuss these elements in detail.

1. Calculation of bounds: In order to calculate an upper bound z_r^{UB} for node r, we use (as usually is done) the LP relaxation, i.e. $z_r^{UB} = z^{LP}$. Specially for the Knapsack Problem, the LP relaxation can be solved very easily "by hand", as shown in the following.

Old objects j'	4	2	3	1	5
New objects j	1	2	3	4	5
$\frac{c_j}{a_j}$	$\frac{8}{8}$	$\frac{14}{16}$	$\frac{16}{21}$	$\frac{11}{17}$	$\frac{7}{12}$
c_j	8	14	16	11	7
a_j	8	16	21	17	12

Table 4.2: Objects sorted according to decreasing benefit coefficient

First, for every object $j \in J$, the *benefit coefficient* (i.e. benefit pro volume unit) is determined. We sort the objects in decreasing order according to the benefit coefficient. One can easily see that an optimal solution of the LP relaxation (i.e. an optimal "fractional packing" of the knapsack) can be found by putting the objects into the knapsack one after another, starting at the object with the greatest benefit coefficient, and stopping at the last object which fits into the knapsack completely. From the following object, one has to take a fraction such that the knapsack becomes full.

Formally, this algorithm can be described as follows. Number the objects in a new fashion,in decreasing order of the benefit coefficients, i.e. we have for the newly numbered objects $J = \{1, \ldots, n\}$:

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \ldots \geq \frac{c_n}{a_n}$$

Now, determine the index k as the index of the last object fitting completely into the knapsack, i.e.

$$k = \max\{k' : \sum_{j=1}^{k'} a_j \leq b\}$$

It is assumed that not all objects fit into the knapsack, i.e. $\sum_{j=1}^{n} a_j > b$ (otherwise the Knapsack Problem is trivial). This assumption implies that $k < n$.

The optimal solution x^{LP} of the LP relaxation is now given by

$$x_j^{LP} = \begin{cases} 1, & j = 1, \ldots, k, \\ (b - \sum_{j'=1}^{k} a_{j'})/a_j, & j = k+1 \\ 0, & j = k+2, \ldots, n \end{cases}$$

Note that this optimal solution contains at most one fractional variable.

In our example, the objects have been renumbered in decreasing order of the benefit coefficient as shown in Table 4.2. This means that the new variable vector $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5)^\top$ corresponds to the old variable vector $(x_4, x_2, x_3, x_1, x_5)^\top$.

Hence, the optimal solution of the LP relaxation of the Knapsack Problem (in node $r = 0$) is given by:

$$\boldsymbol{x}^{LP} = (1, 1, \tfrac{13}{21}, 0, 0) \quad \text{and} \quad z^{LP} = 8 + 14 + \tfrac{13}{21}16 = 31\tfrac{19}{21}$$

2. Heuristic solution search: Starting at the optimal solution $\boldsymbol{x}^{LP}$ of the LP relaxation, one can easily construct a feasible (integer) solution $\boldsymbol{x}^{LB}$ for the Knapsack Problem by setting the (only) fractional variable (if present) to zero:

$$x_j^{LB} = \begin{cases} 1, & j = 1, \ldots, k, \\ 0, & j = k+1, \ldots, n \end{cases}$$

In the example, we obtain as a heuristic solution (in node $r = 0$):

$$\boldsymbol{x}^{LB} = (1, 1, 0, 0, 0) \quad \text{and} \quad z^{LB} = 8 + 14 = 22$$

3. Definition of subproblems: The subproblems are constructed by rounding down and rounding up the (single) fractional variable, respectively, i.e. the fractional variable will be set to 0 or 1.

A subproblem corresponding to node r can be described formally as follows: Two index sets $J_r^0, J_r^1 \subseteq J$ indicate which variables are set to 0 or 1. The variables not fixed yet (i.e. the actual decision variables at node r) are given by the set $J_r = J - J_r^0 - J_r^1$. At node r, we have:

$$x_j = \begin{cases} 0, & j \in J_r^0, \\ 1, & j \in J_r^1, \\ \text{not fixed}, & j \in J_r \end{cases}$$

Hence, subproblem Π^r corresponds to the following Knapsack Problem:

$$\Pi^r : \max\{\sum_{j \in J_r} c_j x_j : \sum_{j \in J_r} a_j x_j \leq b - \sum_{j \in J_r^1} a_j x_j\}$$

Note that the capacity of the knapsack is now reduced by the total volume $\sum_{j \in J_r^1} a_j x_j$ of the objects already packed (i.e. corresponding to the variables set to 1 according to J_r^1).

Node selection strategy: In order to specify a B&B method completely, one needs to define a strategy for the selection of the next node to be processed, after the processing of a node is finished. There are numerous approaches in the literature which we do not discuss here.

For our example, we use a *Depth-First-Search Strategy* which chooses the next node to be processed as the node resulting from *rounding up* the fractional variable. In the above B&B pseudo code, this corresponds to the node $\widehat{r}+2$, and in the following B&B decision

tree, it corresponds to the node on the right-hand side. If there are no successor nodes, one goes back to the last generated nodes (*backtracking*, see example).

The whole procedure of the Branch-and-Bound Method applied to our example is depicted in Table 4.3. The first column contains an entry of the form "[3] $r = 4(2)$" meaning that in the third iteration ([3]), the node $r = 4$ was processed. The notation 4(2) means that node 4 is a direct follower of node 2. All other entries correspond to the values calculated according the B&B pseudo code.

A graphical representation of the B&B decision tree together with all relevant information is given in Figure 4.5.

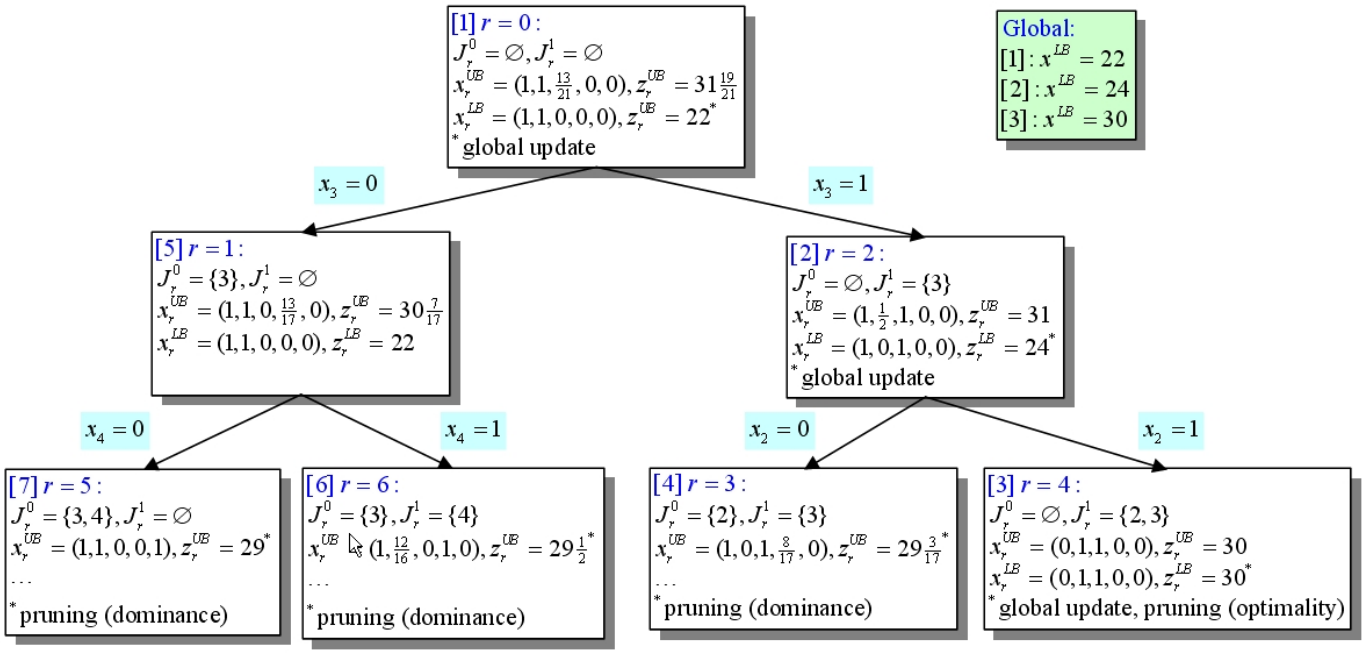


Figure 4.6: B&B branching tree for the Knapsack example

[1] $r = 0(.)$	$J_r^0 = \emptyset, J_r^1 = \emptyset$ $x_r^{UB} = x^{LP} = (1, 1, \frac{13}{21}, 0, 0)$ $x_r^{LB} = (1, 1, 0, 0, 0)$ $x^{LB} = x_r^{LB} = (1, 1, 0, 0, 0)$ $R = \{1, 2\}$	$J_r = \{1, 2, 3, 4, 5\}$ $z_r^{UB} = z^{LP} = 8 + 14 + \frac{13}{21}16 = 31\frac{19}{21}$ $z_r^{LB} = 8 + 14 = 22$ $z^{LB} = x_r^{LB} = 22$	global update
[2] $r = 2(0)$	$J_r^0 = \emptyset, J_r^1 = \{3\}$ $x_r^{UB} = x^{LP} = (1, \frac{1}{2}, 1, 0, 0)$ $x_r^{LB} = (1, 0, 1, 0, 0)$ $x^{LB} = x_r^{LB} = (1, 0, 1, 0, 0)$ $R = \{1, 3, 4\}$	$J_r = \{1, 2, 4, 5\}$ $z_r^{UB} = z^{LP} = 8 + \frac{1}{2}14 + 16 = 31$ $z_r^{LB} = 8 + 16 = 24$ $z^{LB} = z_r^{LB} = 24$	global update
[3] $r = 4(2)$	$J_r^0 = \emptyset, J_r^1 = \{2, 3\}$ $x_r^{UB} = x^{LP} = (0, 1, 1, 0, 0)$ $x_r^{LB} = (0, 1, 1, 0, 0)$ $x^{LB} = x_r^{LB} = (0, 1, 1, 0, 0)$ $R = \{1, 3\}$	$J_r = \{1, 4, 5\}$ $z_r^{UB} = z^{LP} = 14 + 16 = 30$ $z_r^{LB} = 14 + 16 = 30$ $z^{LB} = z_r^{LB} = 30$	pruning (Π^r opt.) global update
[4] $r = 3(2)$	$J_r^0 = \{2\}, J_r^1 = \{3\}$ $x_r^{UB} = x^{LP} = (1, 0, 1, \frac{8}{17}, 0)$ $R = \{1\}$	$J_r = \{1, 4, 5\}$ $z_r^{UB} = z^{LP} = 8 + 16 + \frac{8}{17}11 = 29\frac{3}{17}$	pruning (≤ 30)
[5] $r = 1(0)$	$J_r^0 = \{3\}, J_r^1 = \emptyset$ $x_r^{UB} = x^{LP} = (1, 1, 0, \frac{13}{17}, 0)$ $x_r^{LB} = (1, 1, 0, 0, 0)$ $R = \{5, 6\}$	$J_r = \{1, 2, 4, 5\}$ $z_r^{UB} = z^{LP} = 8 + 14 + \frac{13}{17}11 = 30\frac{7}{17}$ $z_r^{LB} = 8 + 14 = 22$	
[6] $r = 6(1)$	$J_r^0 = \{3\}, J_r^1 = \{4\}$ $x_r^{UB} = x^{LP} = (1, \frac{12}{16}, 0, 1, 0)$ $R = \{5\}$	$J_r = \{1, 2, 5\}$ $z_r^{UB} = z^{LP} = 8 + \frac{12}{16}14 + 11 = 29\frac{1}{2}$	pruning (≤ 30)
[7] $r = 5(1)$	$J_r^0 = \{3, 4\}, J_r^1 = \emptyset$ $x_r^{UB} = x^{LP} = (1, 1, 0, 0, 1)$ $R = \emptyset$	$J_r = \{1, 2, 5\}$ $z_r^{UB} = z^{LP} = 8 + 14 + 7 = 29$	pruning (≤ 30)

Table 4.3: B&B method for the Knapsack example

4.3 Cutting Plane Method

4.3.1 ILP Formulations and Convex Hull

As we have seen in the preceding section, LP relaxations play a central role in solving ILPs by means of Branch-and-Bound, if they are used, as it is usually the case, for calcu-

lating bounds. The quality of the bounds is of crucial importance for the efficiency of any Branch-and-Bound method, since it essentially determines the number of subproblems (nodes) to be processed.

In this section, we will learn how ILPs can be formulated in different ways, and how certain formulations are preferable to others, since their LP relaxations provide stronger bounds. Furthermore, the notion of *convex hull* will be introduced, and it will be shown that the best formulation for an ILP is provided by the convex hull of its feasible (integer) solutions.

We consider an ILP of the form

$$\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\} \quad \text{with } P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$$

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

Definition 4.3. Let $S = P \cap \mathbb{Z}^n$ be the solution space of some ILP Π. A polyhedron $P' \subseteq \mathbb{R}^n$ is called an *(ILP) formulation* for Π (or for S or P) if and only if $P' \cap \mathbb{Z}^n = S$.

Every polyhedron whose integer vectors correspond exactly to the solution space S of some ILP Π, is therefore a formulation for this problem Π. Obviously, there exist infinitely many formulations for an ILP. Figure 4.7 shows two possible formulations for a given ILP.

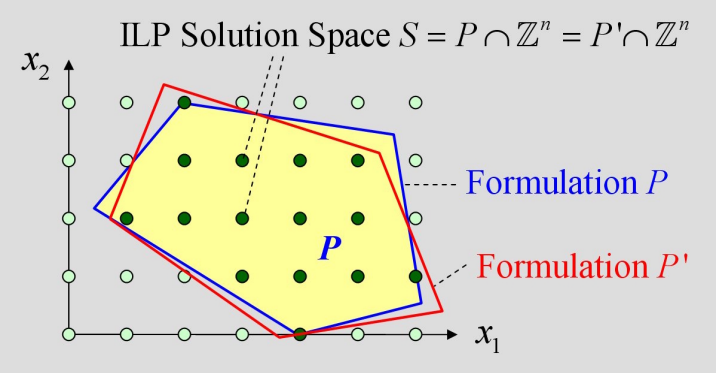


Figure 4.7: Two possible formulations for an ILP

Consider now two formulations P and P' for some ILP. We say that formulation P' is *better* (or *stronger*) than P if $P' \subset P$. Clearly, the LP relaxation associated with P' provides an upper bound at least as good as the LP relaxation associated with P, since $P' \subset P$ implies:

$$\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P'\} \leq \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P\}$$

Figure 4.8 schematically shows an improved formulation for an ILP. The optimal solution of the ILP is indicated as orange point, and the optimal solutions of the LP relaxations

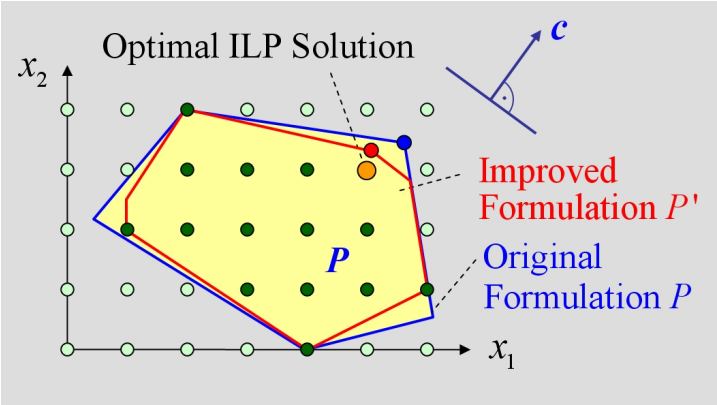


Figure 4.8: Improved formulation for an ILP

associated with P and P' as blue and red point, respectively. Clearly, P' provides a better (i.e. smaller) upper bound for the ILP than P.

In order to answer the question regarding the best possible formulation for an ILP, we need the concept of *convex hull*.

Definition 4.4. For any set $S \subseteq \mathbb{R}^n$, the *convex hull* $\text{conv}(S)$ is defined as the set of all vectors in $\mathbb{R}^n$ that can be represented as convex combination of finitely many vectors from S, i.e.

$$\text{conv}(S) = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i \text{ with } \boldsymbol{x}^i \in S, \lambda_i \geq 0, i = 1, \ldots, k, \sum_{i=1}^{k} \lambda_i = 1\}$$

It can be shown that the convex hull of a set S is the smallest convex set containing S, i.e.

(i) $S \subseteq \text{conv}(S)$
(ii) There is no convex set S' with $S \subseteq S'$ and $S' \subset \text{conv}(S)$

In order to get a geometric idea of the notion of convex hull, consider a set $S \subseteq \mathbb{R}^2$ of points in the plane. For $S = \{\boldsymbol{x}^1\}$, $\text{conv}(S)$ corresponds to the set $\{\boldsymbol{x}^1\}$, for $S = \{\boldsymbol{x}^1, \boldsymbol{x}^2\}$, $\text{conv}(S)$ corresponds to the line segment connecting $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$, and for $S = \{\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3\}$, $\text{conv}(S)$ corresponds to the triangle formed by the three points (provided that the points are not arranged on a line).

In optimization problems with linear objective function, the notion of convex hull plays an important role, as shown in the following proposition.

Proposition 4.2. *Consider an optimization problem* Π *with linear objective function and arbitrary solution space* $S \subseteq \mathbb{R}^n$ *, i.e. written as maximization problem, we have* $\Pi : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in S\}$*. If* Π *has a finite optimum, the following holds:*

$$\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in S\} = \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \text{conv}(S)\}$$

Proof. Let $\boldsymbol{x}^*$ be an optimal solution of $\Pi' = \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \text{conv}(S)\}$. Since $\boldsymbol{x}^* \in \text{conv}(S)$, it can be represented as:

$$\boldsymbol{x}^* = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i \text{ with } \lambda_i \geq 0, i = 1, \ldots, k, \sum_{i=1}^{k} \lambda_i = 1$$

where $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k \in S$ are finitely many vectors from S. It follows:

$$\boldsymbol{c}^\top \boldsymbol{x}^* = \sum_{i=1}^{k} \lambda_i \boldsymbol{c}^\top \boldsymbol{x}^i \leq \sum_{i=1}^{k} \lambda_i \max\{\boldsymbol{c}^\top \boldsymbol{x}^i : i = 1, \ldots, k\} = \max\{\boldsymbol{c}^\top \boldsymbol{x}^i : i = 1, \ldots, k\}$$

We have thus shown that for any optimal solution $\boldsymbol{x}^* \in \text{conv}(S)$ of Π', there exists a solution $\boldsymbol{x}^i \in S$ which is at least as good as $\boldsymbol{x}^*$, i.e. $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in S\} \geq \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \text{conv}(S)\}$. On the other hand, since $S \subseteq \text{conv}(S)$, we have $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in S\} \leq \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \text{conv}(S)\}$. Therefore, $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in S\} = \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \text{conv}(S)\}$. □

Expressed in words, the above Theorem says that for arbitrary optimization problems with *linear objective functio*n, replacing the solution space S by its convex hull $\text{conv}(S)$ does not change the optimum. At first glance, this insight seems of little benefit for a solution approach, since the original solution space S is replaced by a new solution space $\text{conv}(S)$, which in general is much larger than the original space.

However, the crucial point is that convex sets have favorable geometric properties which in general can be leveraged for the design of powerful optimization algorithms. In particular, this is also the case in Integer Programming, as we will see in the following.

Definition 4.5 (Integer Hull)**.** The *integer hull* of a polyhedron $P \subseteq \mathbb{R}^n$ is given by $P_\mathbb{Z} = \text{conv}(P \cap \mathbb{Z}^n)$.

Hence, $P_\mathbb{Z}$ corresponds to the convex hull of all integer vectors contained in the polyhedron P.

Definition 4.6. A polyhedron $P \subseteq \mathbb{R}^n$ is called *rational* if there is a rational matrix $\boldsymbol{A} \subseteq \mathbb{Q}^{m \times n}$ and a rational vector $\boldsymbol{b} \subseteq \mathbb{Q}^m$ such that $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$.

In the following, we will restrict our discussion to rational polyhedra. The restriction to rational coefficients does normally not imply any practical limitation, since real-world applications can almost always be modelled with rational numbers. Furthermore, numerical calculations with digital computers are always based on rational numbers since memory capacity for representing numbers is finite.

Proposition 4.3. *Let $P \subseteq \mathbb{R}^n$ be a rational polyhedron. Then $P_\mathbb{Z} = \text{conv}(P \cap \mathbb{Z}^n)$ is a rational polyhedron, too. Further, all vertices of $P_\mathbb{Z}$ are integer.*

Note that the integer hull of a non-rational polyhedron is not necessarily a polyhedron, since its description may require an infinite number of linear inequalites.

The above Proposition has the following interpretation. We consider an ILP Π with rational formulation P and solution space $S = P \cap \mathbb{Z}^n$. The convex hull of the solution space (i.e. the integer hull of P) is given by $P_\mathbb{Z} = \text{conv}(S) = \text{conv}(P \cap \mathbb{Z})$, and corresponds again to a (rational) polyhedron. Since $P_\mathbb{Z} = \text{conv}(S)$ is the smallest convex set containing S, there exists no convex set S' with $S \subseteq S'$ and $S' \subset P_\mathbb{Z}$. Consequently, as every polyhedron is a convex set, there exists no polyhedron P' with $S \subseteq P'$ and $P' \subset P_\mathbb{Z}$. Therefore, the integer hull $P_\mathbb{Z}$ is the best possible formulation for Π (or P).

Corollary 4.1 (Best ILP Formulation)**.** *If $P \subseteq \mathbb{R}^n$ is a rational polyhedron, then $P_\mathbb{Z} = \text{conv}(P \cap \mathbb{Z}^n)$ is the best possible formulation for P.*

Figure 4.9 shows a (unbounded) polyhdron P and its integer hull.

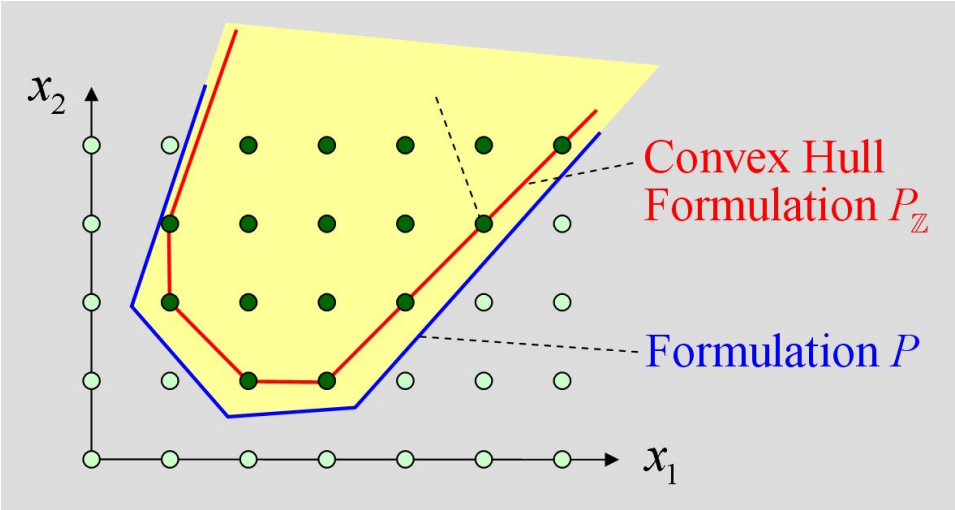


Figure 4.9: A polyhedron P and its integer hull $P_\mathbb{Z}$

As a central result of this section, the following Proposition states that, in principle, any *Integer Linear* Program with (rational) formulation P can solved by solving the associated *Linear* Program with solution space $P_\mathbb{Z}$. Therefore, and maybe somewhat surprisingly, we have theoretically transformed a "difficult" optimization problem of type "ILP" into an "easy" problem of type "LP".

Proposition 4.4. *Let $P \subseteq \mathbb{R}^n$ be a rational polyhedron and assume that the ILP $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \subseteq P \cap \mathbb{Z}^n\}$ has a finite optimum. Then it holds:*

$$\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\} = \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P_\mathbb{Z}\}$$

Proof. The result follows directly from Proposition 4.2, since $P_{\mathbb{Z}} = \text{conv}(P \cap \mathbb{Z}^n)$. □

Figure 4.10 shows an exemplary ILP with formulation P, and the best possible formulation $P_{\mathbb{Z}}$. Note that the optimum of the LP $\max\{cx : x \in P_{\mathbb{Z}}\}$ corresponds to the optimum of the ILP. Observe also that all vertices of $P_{\mathbb{Z}}$ are elements of $P \cap \mathbb{Z}^n$, and hence integer vectors.

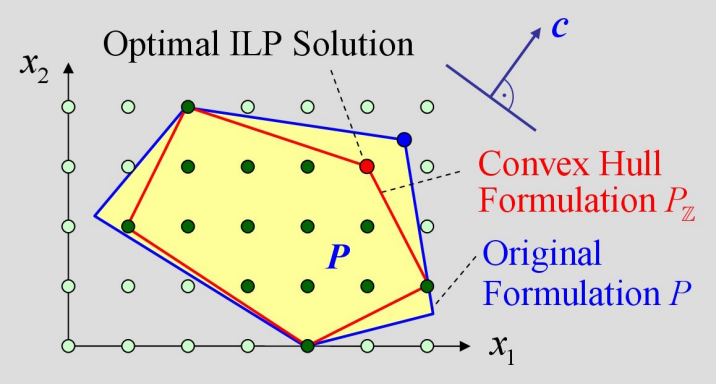


Figure 4.10: Integer hull is the best formulation for an ILP

In summary, the following important insights have been elaborated (where rationality is assumed throughout):

- The best possible formulation for an ILP of the form $\Pi : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\}$ is given by the integer hull $P_{\mathbb{Z}} = \text{conv}(P \cap \mathbb{Z}^n)$, and $P_{\mathbb{Z}}$ is an *integer polyhedron*, i.e. all vertices are integer vectors.
- In principle, the optimum of ILP Π can be determined by solving the LP $\Pi' : \max\{cx : x \in P_{\mathbb{Z}}\}$.
- An (integer) optimal solution of ILP Π can be found by solving the LP Π' by means of a "vertex method", i.e. an LP solution method such as the Simplex Algorithm, which guarantees to find an optimal solution that is a vertex (if there exists an optimal solution at all). Since all vertices of $P_{\mathbb{Z}}$ are integer, a "vertex method" for LP Π' produces an optimal solution (if existent) which is a vertex and hence integer, and therefore also an optimal solution of ILP Π.

However, the theoretical result that every ILP can basically be solved as an LP, is in general not directly applicable to practical problems due to the following two fundamental issues.

- In most cases, an explicit polyhedral description of $P_{\mathbb{Z}}$, i.e. a representation of the form $P_{\mathbb{Z}} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ with finitely many inequalities, is not available (i.e. unknown). It can even be shown that the problem of finding an "appropriate" polyhedral description of $P_{\mathbb{Z}}$ is in some sence "computationally equally difficult" as solving the ILP itself.

- Even if a polyhedral description of $P_{\mathbb{Z}}$ is known, it is a matter of fact that for most ILPs, this description involves a huge number of linear inequalities (i.e. number of rows of matrix A). Tyically, this number is *exponential* with respect to the number of variables, i.e. if $P_{\mathbb{Z}} \subseteq \mathbb{R}^n$, the number of constraints is of order 10^n. Directly solving an LP with such a huge number of constraints is already for small values of n (e.g. $n = 100$) absolutely impossible.

In view of these two fundamental difficulties, we will now discuss the following two directly related issues:

- Given an ILP $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\}$ with formulation $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, how can we construct an improved ("tighter") formulation $P' = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}'\boldsymbol{x} \leq \boldsymbol{b}'\}$, or even the best possible formulation $P_{\mathbb{Z}} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}''\boldsymbol{x} \leq \boldsymbol{b}''\}$?
- How can we solve LPs with a huge ("exponential") number of contraints?

Both questions lead us to the concept of *valid inequalities (German: gültige Ungleichungen)* and *cutting planes (German: Schnittebenen)*, as well as the related algorithmic approach called *Cutting Plane Method (German: Schnittebenen-Verfahren)*.

But first, we will consider the introductory example "Design of a Regional Optical Fiber Network", and discuss different possible ILP formulations for this problem. In particular, we will see that an "intellectually simpler" formulation does not necessarily represent a better formulation from the polyhedral point of view.

4.3.2 Example: Minimum Spanning Trees in Graphs

The introductory example "Design of a Regional Optical Fiber Network" can be formulated as optimization problem in an undirect graph, the so-called *Minimum Weight Spanning Tree* problem (see also Appendix).

For an undirected graph $G = (V, E)$ with edge weights c_{vw} for $(v, w) \in E$, we introduce the following notions (see also the introductory definitions in Section 0.3). A *connector* in G is a set of edges $T \subseteq E$ such that the subgraph (V, T) is connected. A *spanning tree* in G is a set of edges $T \subseteq E$ such that the subgraph (V, T) is connected and acyclic (i.e. a spanning tree is an acyclic connector). It can be shown that every spanning tree in a graph with n vertices comprises exactly $n - 1$ edges, and every connector at least $n - 1$ edges. The *weight* $c(T)$ of a set of edges $T \subseteq E$ is defined as the sum of the weights of all edges in T, i.e. $c(T) = \sum_{(v,w)\in T} c_{vw}$.

Clearly, the introductory example "Design of a Regional Optical Fiber Network" can be formulated as the problem of finding a *Minimum Weight Connector* in a graph $G = (V, E)$, i.e. a connector $T \in E$ with minimum weight $c(T)$. The nodes $v \in V$ correspond to the locations, the edges $(v, w) \in E$ to the possible connections (cable trenches) between the locations, and the edge weights $c_{v,w}$ to the connection lengths (distances).

Suppose now that $T \in E$ is a minimum weight connector in $G = (V, E)$, and all edge weights are positive, i.e. $c_{vw} > 0$ for $(v, w) \in E$. It can easily be seen that T does not

contain any cycle, i.e. T is actually a spanning tree. Indeed, if T would have a cycle, we could remove an abitrary edge from this cycle, and the resulting subset T' would still be a connector (why?). But since all arc weights are positive, $c(T') < c(T)$, a contradiction to the minimality of T.

Therefore, the considered example "Design of a Regional Optical Fiber Network" can also be formulated as a *Minimum Weight Spanning Tree* problem. However, for didactical purposes, the following discussion will focus on the *Minimum Weight Connector* formulation.

In order to establish an ILP formulation for the *Minimum Weight Connector* problem in a graph $G = (V, E)$, we consider an arbitrary subset of edges $T \subseteq E$, and introduce for each edge $(v, w) \in E$ a binary variable x_{vw} with the following interpretation:

$$x_{vw} = \begin{cases} 1, & \text{if } (v,w) \in T \\ 0, & \text{else} \end{cases}$$

We have therefore a vector of $|E|$ variables $\boldsymbol{x} \in \{0,1\}^E$, which defines a subset of edges $T \subseteq E$ given by:

$$T = \{(v,w) \in E : x_{vw} = 1\}$$

Our task is now to formulate an ILP for the Minimum Weight Connector problem, i.e. to find appropriate linear constraints and an objective function for the variables $\boldsymbol{x}$, such that the associated optimal solutions correspond exactly to the connectors of minimum weight in G. We will present two different ILP formulations: Version 1 is conceptually easier to understand, but it is not the best possible formulation, since the associated polyhedron P^1 is not integer and therefore $P^1 \supset P_{\mathbb{Z}}$. Version 2 corresponds to the best possible formulation, hence the associated polyhedron P^2 is integer and $P^2 = P_{\mathbb{Z}}$. P^2 is called the *Connector Polytope* and has been stated by Edmonds and Fulkerson (1970), see e.g. [24] on page 863.

Version 1 (ILP Formulation: Non-Integer Polytope):

Let $G = (V, E)$ be an undirected graph with positive arc weights $c_{vw} > 0$ for $(v, w) \in E$. The condition that an edge subset $T \subseteq E$ defines a connected subgraph (V, T) can be formulated as follows: We fix an arbitrary node $v_0 \in V$ and require that any non-empty subset S of nodes not containing v_0, i.e. any $\emptyset \subset S \subseteq V - \{v_0\}$, has at least one outgoing edge, i.e. an edge $(v, w) \in E$ with $v \in S$ and $w \in V - S$. We write $\delta(S)$ to denote the set of outgoing edges of an arbitrary node set $S \subseteq V$, i.e. $\delta(S) = \{(v,w) \in E : v \in S, w \in V - S\}$ (cf. the introductory definitions 0.3).

Based on this idea, we obtain the following ILP formulation for the Minimum Weight Connector problem:

$$\min \sum_{(v,w)\in E} c_{vw}x_{vw} \tag{4.1}$$

$$\sum_{(v,w)\in\delta(S)} x_{vw} \geq 1, \qquad \emptyset \subset S \subseteq V - \{v_0\} \tag{4.2}$$

$$x_{vw} \in \{0,1\}, \qquad (v,w) \in E \tag{4.3}$$

It can be shown that $\boldsymbol{x}$ is a solution of this ILP if and only if $\boldsymbol{x}$ is the incidence vector of a connector. Therefore, since all edge weights are positive, an optimal solution corresponds to a minimum weight connector.

Writing this ILP in the usual notation as $\min\{\boldsymbol{c}^\top\boldsymbol{x} : \boldsymbol{x} \in P^1 \cap \mathbb{Z}^n\}$, the polyhedron P^1 is defined by the following constraints:

$$\sum_{(v,w)\in\delta(S)} x_{vw} \geq 1, \qquad \emptyset \subset S \subseteq V - \{v_0\}$$

$$0 \leq x_{vw} \leq 1, \qquad (v,w) \in E$$

Note that this formulation comprises an *exponential* number of inequalities, namely $2^{|V|-1} - 1 + 2\,|E|$. In fact, there is a "δ-inequality" associated to every S, and the number of node subsets S with $\emptyset \subset S \subseteq V - \{v_0\}$ is given by $2^{|V|-1} - 1$ (recall that a set with n elements has 2^n subsets). In addition, the number of 0-1 bounds is $2|E|$.

Unfortunately, this polyhedron P^1 has non-integer vertices, i.e. we have $P^1 \supset P_{\mathbb{Z}}$, and P^1 is not the best possible (integer hull) formulation. Below, this will be geometrically illustrated with a small example in $\mathbb{R}^3$.

Version 2 (ILP Formulation: Integer Polytope, Connector Polytope):

An alternative formulation (actually the best possible) can be established as follows. Let Π be the set of all "non-trivial" partitions of the node set V, i.e.

$$\Pi = \{\{S_1,\ldots,S_k\} : 2 \leq k \leq n, \emptyset \subset S_p \subseteq V, S_p \cap S_q = \emptyset, p,q \in \{1,\ldots,k\}, p \neq q, \bigcup_{p\in\{1,\ldots,k\}} S_p = V\}$$

Remember that, expressed in words, a "non-trivial" partition of a set V is a collection of $k \geq 2$ non-empty subsets $\{S_1,\ldots,S_k\}$ of V, such that the subsets are pairwise disjunct, and their union yields V. For any partition $\mathcal{P} \in \Pi$, we define $\delta(\mathcal{P})$ as the set of all edges joining two different sets of $\mathcal{P}$, i.e. for partition $\mathcal{P} = \{S_1,\ldots,S_k\}$ we have:

$$\delta(\mathcal{P}) = \{(v,w) \in E : v \in S_p, w \in S_q, p,q \in \{1,\ldots,k\}, p \neq q\}$$

The central idea is now, that for any partition $\mathcal{P} = \{S_1, \ldots, S_k\}$, a connector must connect all subsets $S_1, \ldots, S_k$, otherwise there would be an isolated subset. In order to connect $|\mathcal{P}| = k$ subsets, we need at least $k - 1$ edges (remember that analogously, a connector with n nodes has at least $n - 1$ edges). This consideration leads to the following ILP formulation for the Minimum Weight Connector problem:

$$\min \sum_{(v,w)\in E} c_{vw} x_{vw} \tag{4.4}$$

$$\sum_{(v,w)\in\delta(\mathcal{P})} x_{vw} \geq |\mathcal{P}| - 1, \qquad \mathcal{P} \in \Pi \tag{4.5}$$

$$x_{vw} \in \{0, 1\}, \qquad (v, w) \in E \tag{4.6}$$

Writing this ILP in the usual notation as $\min\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P^2 \cap \mathbb{Z}^n\}$, the polyhedron P^2 is defined by the following constraints:

$$\sum_{(v,w)\in\delta(\mathcal{P})} x_{vw} \geq |\mathcal{P}| - 1, \qquad \mathcal{P} \in \Pi$$

$$0 \leq x_{vw} \leq 1, \qquad (v, w) \in E$$

P^2 is called the *Connector Polytope*. Indeed, it can be shown that P^2 is the convex hull of the incidence vectors of all connectors of G. Thus, we have $P^2 = P_{\mathbb{Z}}$, and all vertices of P^2 are integer, and therefore 0-1 vectors (because of $0 \leq x_{vw} \leq 1$). Furthermore, $\boldsymbol{x}$ is an integer (0-1) vector in P^2 if and only if $\boldsymbol{x}$ is the incidence vector of a connector in G.

In summary, the polytope P^2 is integer and represents the best possible ILP formulation for the connector problem. The integrality constraints can therefore be omitted, and the Minimum Weight Connector problem can theoretically be solved by solving the following LP (without integrality constraints):

$$\min\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P^2\}$$

However, directly solving this LP by means of an LP solver (e.g. based on the Simplex method) would fail in practice, since the number of constraints is too large (exponential in the number of nodes) even for small instances.

Illustration with an Example in $\mathbb{R}^3$:

We consider a complete graph $G = (V, E)$ with only three nodes, i.e. $|V| = 3$, and give a geometric illustration of the two formulations (polytopes) discussed above. Graph G is shown in Figure 4.11.

The edge set of G is given by $E = \{(1, 2), (1, 3), (2, 3)\}$. (Remember that for undirected graphs, the notations (v, w) and (w, v) denote the same edge.) We have three associated binary variables x_{12}, x_{13}, and x_{23}, hence $\boldsymbol{x} = (x_{12}, x_{13}, x_{x23})^\top \in \mathbb{R}^E$ is 3-dimensional.

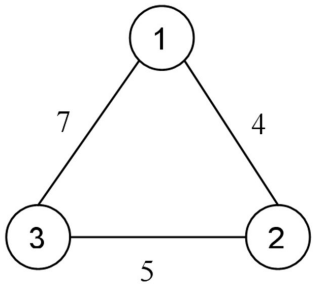


Figure 4.11: Complete graph $G = (V, E)$ with $|V|$ =3

(1) For the first formulation P^1, we choose $v^0 = 1$. This leads to the following constraints for P^1:

$$\begin{aligned} x_{12} + x_{23} &\geq 1, \quad \text{for node set } S = \{2\} \text{ mit } \delta(S) = \{(1,2),(2,3)\} \\ x_{13} + x_{23} &\geq 1, \quad \text{for node set } S = \{3\} \text{ mit } \delta(S) = \{(1,3),(2,3)\} \\ x_{12} + x_{13} &\geq 1, \quad \text{for node set } S = \{2,3\} \text{ mit } \delta(S) = \{(1,2),(1,3)\} \\ 0 \leq x_{12}, x_{13}, x_{23} &\leq 1 \end{aligned}$$

Polytope P^1 is depicted in Figure 4.12.

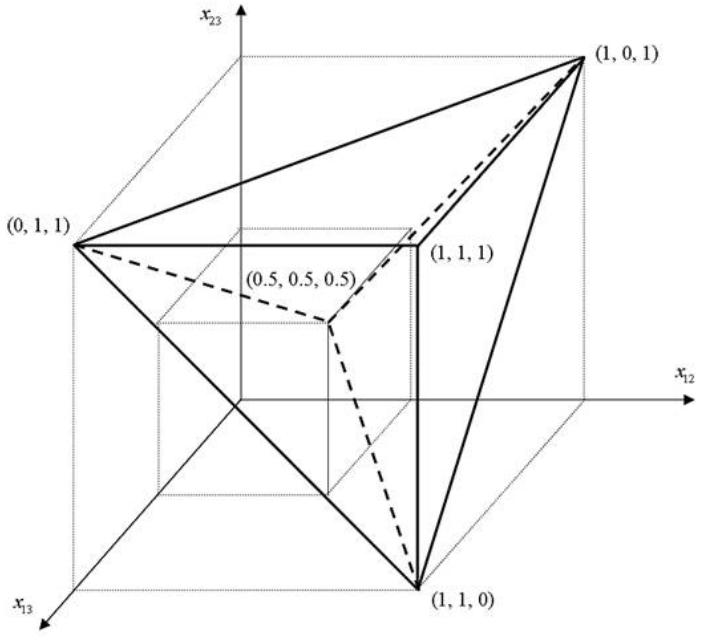


Figure 4.12: Graphical representation of formulation P^1 for $|V| = 3$

As we can see, P^1 has 5 vertices with coordinates $(1,1,0)$, $(1,0,1)$, $(0,1,1)$, $(1,1,1)$, and $(\frac{1}{2},\frac{1}{2},\frac{1}{2})$. The last vertex is not integer, which shows that $P^1 \neq P_{\mathbb{Z}}$.

(2) For the second formulation (polytope) P^2, we have the following constraints:

$$
\begin{aligned}
x_{12} + x_{23} &\geq 1, \quad \text{for partition } \mathcal{P} = \{\{2\}, \{1,3\}\} \text{ with } \delta(\mathcal{P}) = \{(1,2),(2,3)\} \\
x_{13} + x_{23} &\geq 1, \quad \text{for partition } \mathcal{P} = \{\{3\}, \{1,2\}\} \text{ with } \delta(\mathcal{P}) = \{(1,3),(2,3)\} \\
x_{12} + x_{13} &\geq 1, \quad \text{for partition } \mathcal{P} = \{\{1\}, \{2,3\}\} \text{ with } \delta(\mathcal{P}) = \{(1,2),(1,3)\} \\
x_{12} + x_{13} + x_{23} &\geq 2, \quad \text{for partition } \mathcal{P} = \{\{1\}, \{2\}, \{3\}\} \text{ with } \delta(\mathcal{P}) = \{(1,2),(1,3),(2,3)\} \\
0 \leq x_{12}, x_{13}, x_{23} &\leq 1
\end{aligned}
$$

The associated polytope, the *Connector Polytope*, is depicted in Figure 4.13.

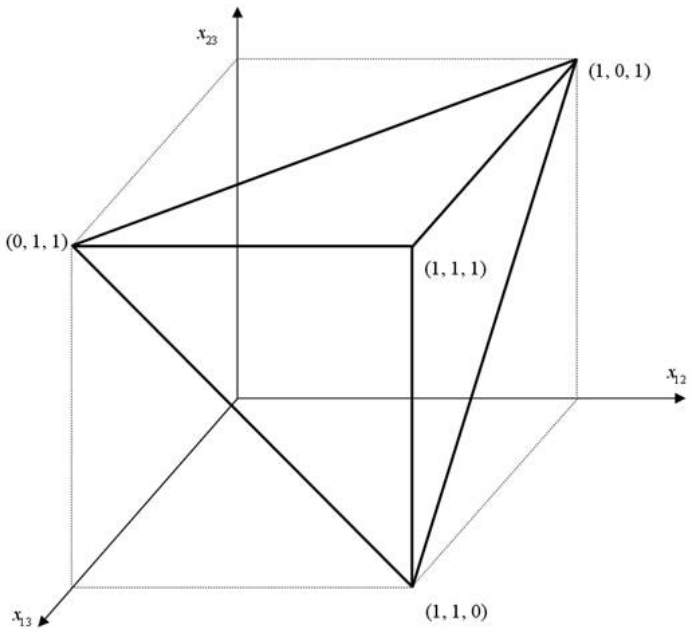


Figure 4.13: Graphical representation of formulation P^2 for $|V| = 3$

As we can see, the additional constraint $x_{12} + x_{13} + x_{23} \geq 2$ in P^2 "cuts away" the the non-integer vertex $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ of P^1. P^2 has only integer vertices and hence $P^2 = P_{\mathbb{Z}}$.

Finally, we remark that the problem of finding a Minimum Weight Spanning Tree in a graph belongs to the easiest problems of Combinorial Optimization, and can be solved by means of a so-called Greedy Algorithm (for more information, see Theory of Matroids). The algorithm works as follows:

Greedy Algorithm for Minimum Weight Spanning Tree: Sort all edges according to non-decreasing weights (i.e. smallest weights first). Process the edges in this order, and include an edge in the current edge set T (i.e. "partial tree"), if it does not form a cycle with the edges already in T.

4.3.3 Cutting Plane Method

Definition 4.7 (Valid Inequality)**.** Let $P \subseteq \mathbb{R}^n$ be a polyhedron. An inequality of the form $\boldsymbol{\alpha}^\top \boldsymbol{x} \leq \beta$ with $\boldsymbol{\alpha} \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ is called a *valid inequality* for P if $\boldsymbol{\alpha}^\top \boldsymbol{x} \leq \beta$ holds for all $\boldsymbol{x} \in P$.

Definition 4.8 (Cutting Plane)**.** Let $P \subseteq \mathbb{R}^n$ be a polyhedron. A valid inequality for the integer hull $P_\mathbb{Z}$ is called a *cutting plane* or *cut* for P.

Given a cutting plane $\boldsymbol{\alpha}^\top \boldsymbol{x} \leq \beta$, we also denote the set $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \beta\}$ as a cutting plane, i.e. we do usually not distinguish between the inequality defining a cutting plane, and the associated set of vectors fulfilling this inequality.

Given a polyhedron P, we say that a cutting plane for P of the form $\boldsymbol{\alpha}^\top \boldsymbol{x} \leq \beta$ *cuts away* a point $\boldsymbol{x}^* \in \mathbb{R}^n - P_\mathbb{Z}$ (i.e. a point outside of the integer hull), if $\boldsymbol{\alpha}^\top \boldsymbol{x}^* > \beta$. Note that $\boldsymbol{\alpha}^\top \boldsymbol{x}^* \leq \beta$ for all $\boldsymbol{x} \in P_\mathbb{Z}$. Typically, we are interested in cutting planes which cut away points contained in P, but not in $P_\mathbb{Z}$ (i.e. points in $P - P_\mathbb{Z}$).

General Cutting Plane Method for ILP:

Input: An ILP instance of the form $\Pi : \max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\}$ with rational formulation $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$. It is assumed that Π has a finite optimum.

Output: An optimal solution $\boldsymbol{x}^*$ of Π (if the algorithm terminates).

1. Initialization: $P' := P$.
2. Solve the linear relaxation $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in P'\}$ using a vertex method. Let $\boldsymbol{x}^*$ be the optimal solution (vertex) found.
3. If $\boldsymbol{x}^*$ is integer, then stop.
4. Find a cutting plane for P' which cuts away $\boldsymbol{x}^*$, i.e. an inequality $\boldsymbol{\alpha}^\top \boldsymbol{x} \leq \beta$ which is valid for all $\boldsymbol{x} \in P'_\mathbb{Z}$, but $\boldsymbol{\alpha}^\top \boldsymbol{x}^* > \beta$.
5. Add this cutting plane to the polyhedron P' to obtain the new polyhedron $P'' := \{x \in P' : \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \beta\}$.
6. Set $P' := P''$ and go to Step 2.

□

Note that there is no guarantee that this algorithm terminates, i.e. the solution process has possibly to be terminated "artificially" after a certain time. In this case, x^* is a (non-integer) upper bound for the optimum of Π.

Figure 4.14 schematically illustrates the cutting plane procedure.

The crucial point in a cutting plane method is apparently to find an appropriate cutting plane in step 4. In practice, one typically does not look for "arbirtrary" cutting planes, but one tries to elaborate, on a problem-specific base, certain suitable classes (types) of

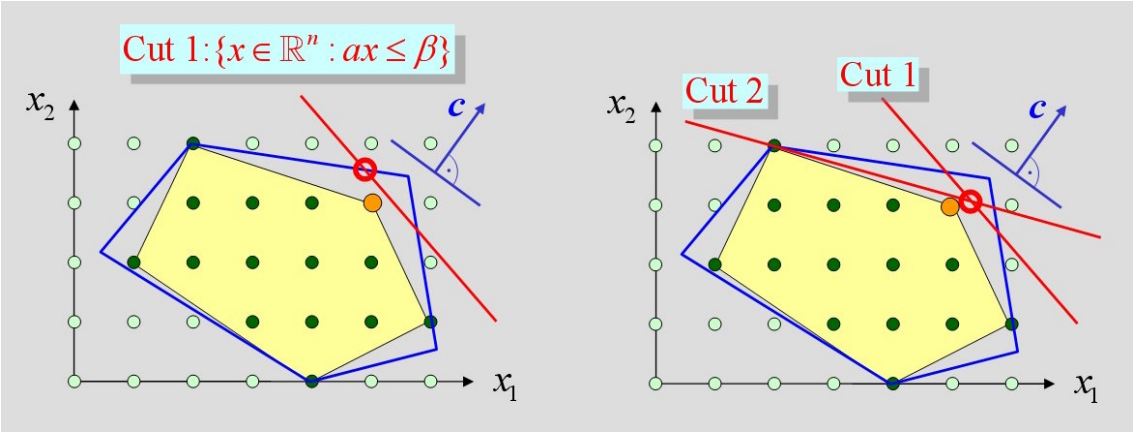


Figure 4.14: Schematic illustration of Cutting Plane Method

cutting planes, that are valid and useful for the problem considered. In step 4, one then focuses on these classes to find a cutting plane.

The development of a specific cutting plane algorithm involves two crucial and challenging tasks:

- The search for "suitable" classes of cutting planes for the problem considered. Typically, one tries to find cutting planes that are necessary for the description of $P_{\mathbb{Z}}$ (so-called facet-defining inequalities describing the highest-dimensional faces of $P_{\mathbb{Z}}$). Identifying such cutting planes is difficult in general, and at the same time, it is crucial for the efficieny of a cutting plane algorithm.
- The solution of the so-called *separation problem*: Given a polyhedron P and a fractional point $x^* \in P - P_{\mathbb{Z}}$, find (within the considered class) a cutting plane for P that cuts away x^*, if existing. As a deep result in integer programming and complexity theory, it can be shown that the computational complexity of the separation problem is basically the same as that for solving the associated ILP, i.e. if one of these problems is efficiently solvable, the other problem is efficiently solvable, too.

4.3.4 General Cutting Planes

There exist various approaches to derive, in a general way, cutting planes for a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$. One important approach is given by the so-called *Gomory-Chvátal cutting planes (G-C cuts)*.

Definition 4.9. Let $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ be a rational polyhedron. A *Gomory-Chvátal cutting plane (G-C cutting plane)* for P is an inequality of the form $\boldsymbol{\alpha}^\top \boldsymbol{x} \leq \lfloor \beta \rfloor$ where $\boldsymbol{\alpha}^\top = \boldsymbol{u}^\top \boldsymbol{A} \in \mathbb{Z}^n$ (integer!) and $\beta = \boldsymbol{u}^\top \boldsymbol{b}$ for a vector $\boldsymbol{u} \in \mathbb{Q}^m$ with $\boldsymbol{u} \geq \boldsymbol{0}$.

Here, the notation $\lfloor \beta \rfloor$ denotes the largest integer number lower or equal than β (also called "floor"):

$$\lfloor \beta \rfloor = \max\{e \in \mathbb{Z} : e \leq \beta\} \qquad \text{for } \beta \in \mathbb{R}$$

Proposition 4.5. *A* Gomory-Chvátal cutting plane *for P is a valid inequality for $P_\mathbb{Z}$ (i.e. a cutting plane for P).*

Apparently, there exist infinitely many G-C cuts for a polyhedron P. However, it can be shown that a *finite* subset of them implies the rest.

Consider for a rational polyhedron $P \subseteq \mathbb{R}^n$ all possible G-C cutting planes and define:

$$\mathcal{GC}(P) = \{(\boldsymbol{\alpha}, \beta) : \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \lfloor \beta \rfloor \text{ is a G-C cutting plane for } P\}$$

One can ask the question whether it is possible to construct the integer hull $P_\mathbb{Z}$ using only G-C cutting planes. Let:

$$P^{\mathcal{GC}} = \{\boldsymbol{x} \in P : \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \lfloor \beta \rfloor \text{ for } (\boldsymbol{\alpha}, \beta) \in \mathcal{GC}(P)\}$$

The following arguments apply to rational polytopes (bounded polyhedra). It is possible to generalize these results to unbounded polyhedra.

Proposition 4.6. *If P is a rational polytope, then $P^{\mathcal{GC}}$ is also a rational polytope.*

We have $P_\mathbb{Z} \subseteq P^{\mathcal{GC}} \subseteq P$, but in general $P_\mathbb{Z} \neq P^{\mathcal{GC}}$.

However, it is possible to repeat this approach in the sense that all G-C cuts of $P^{\mathcal{GC}}$ are again added to $P^{\mathcal{GC}}$, etc. Denote $P^0 = P$ and $P^i = (P^{i-1})^{\mathcal{GC}}$ for $i = 1, 2, \ldots$, and consider the following sequence of polytopes:

$$P = P^0 \supseteq P^1 \supseteq \ldots \supseteq P_\mathbb{Z}$$

Proposition 4.7. *If P is a rational polytope, then there exists a finite number k, such that $P^k = P_\mathbb{Z}$. The smallest number k with this property is called* Chvátal Rank *of P.*

In other words, we have shown that, in principle, the integeger hull $P_\mathbb{Z}$ of a polyhedron P can be constructed in a finite number of steps by "successively adding" G-C cuts.

G-C cuts provide a theoretical framework for finding polyhedral descriptions of integer hulls. Furthermore, for certain problems, they are also useful to derive specific classes of cutting planes which then can be used in cutting plane algorithms.

From a computational point of view, direct application of G-C cuts in cutting plane methods has been of minor significance for a long time. Ralph E. Gomory developed in 1960, as a poineering achievement, a generally applicable cutting plane method for solving ILPs, where a special type of G-C cuts is utilized, the so-called *Gomory cutting planes*. This method generates, based on a fractional basic solution of the Simplex Algorithm (in Standard Form), in a simple way a Gomory cut, which cuts away this basis solution. The method terminates in a finite number of steps, but is computationally not efficient, and did not achieve practical significance. However, in recent times, G-C cuts are of increasing importance in view of new mathematical insights and technologies in the development of ILP solvers.

Finally, we show by means of an example how G-C cuts can heuristically be used to construct the integer hull of a polyhedron.

Example: Gomory-Chvátal Cutting Planes

Consider the (rational) polytope $P = \{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{x} \text{ fulfils } (1) \dots (3)\}$:

$$-2x_1 + x_2 \leq 0 \quad (1)$$
$$2x_1 + x_2 \leq 6 \quad (2)$$
$$-x_2 \leq -1 \quad (3)$$

A graphical representation of P is given in Figure 4.15.

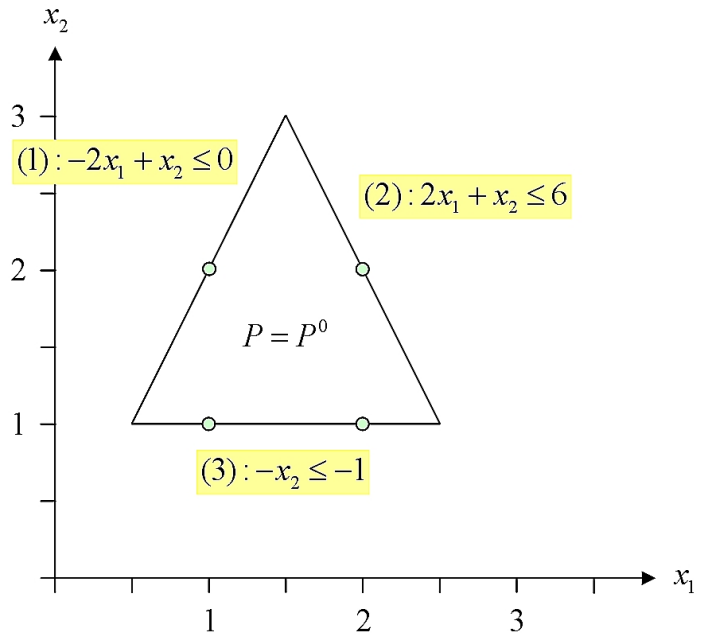


Figure 4.15: Polytope P

The four integer points of P are given by:

$$P \cap \mathbb{Z} = \{(1,1), (1,2), (2,1), (2,2)\}$$

In matrix notation, we have $P = \{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, where:

$$\boldsymbol{A} = \begin{pmatrix} -2 & 1 \\ 2 & 1 \\ 0 & -1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{pmatrix} 0 \\ 6 \\ -1 \end{pmatrix}.$$

1. Construction of P^1

Starting with $P = P^0$, one can find (by trying) the following four G-C cutting planes relevant for the description of $P^1 = P^{\mathcal{GC}}$:

$$\boldsymbol{u}^\top = (\tfrac{1}{2}, 0, \tfrac{1}{2}), \boldsymbol{\alpha}^\top = \boldsymbol{u}^\top \boldsymbol{A} = (-1, 0), \beta = \boldsymbol{u}^\top \boldsymbol{b} = -\tfrac{1}{2}, \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \lfloor \beta \rfloor : \quad -x_1 \leq -1 \tag{4}$$

$$\boldsymbol{u}^\top = (0, \tfrac{1}{2}, \tfrac{1}{2}), \boldsymbol{\alpha}^\top = \boldsymbol{u}^\top \boldsymbol{A} = (1, 0), \beta = \boldsymbol{u}^\top \boldsymbol{b} = 2\tfrac{1}{2}, \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \lfloor \beta \rfloor : \quad x_1 \leq 2 \tag{5}$$

$$\boldsymbol{u}^\top = (\tfrac{3}{4}, \tfrac{1}{4}, 0), \boldsymbol{\alpha}^\top = \boldsymbol{u}^\top \boldsymbol{A} = (-1, 1), \beta = \boldsymbol{u}^\top \boldsymbol{b} = 1\tfrac{1}{2}, \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \lfloor \beta \rfloor : \quad -x_1 + x_2 \leq 1 \tag{6}$$

$$\boldsymbol{u}^\top = (\tfrac{1}{4}, \tfrac{3}{4}, 0), \boldsymbol{\alpha}^\top = \boldsymbol{u}^\top \boldsymbol{A} = (1, 1), \beta = \boldsymbol{u}^\top \boldsymbol{b} = 4\tfrac{1}{2}, \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \lfloor \beta \rfloor : \quad x_1 + x_2 \leq 4 \tag{7}$$

The resulting polytope is given by:

$$P^1 = \{\boldsymbol{x} \in P : \boldsymbol{x} \text{ fulfils } (4) \ldots (7)\} = \{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{x} \text{ fulfils } (1) \ldots (7)\}$$

In matrix notation we have $P^1 = \{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{A}^1 \boldsymbol{x} \leq \boldsymbol{b}^1\}$ where:

$$\boldsymbol{A}^1 = \begin{pmatrix} -2 & 1 \\ 2 & 1 \\ 0 & -1 \\ -1 & 0 \\ 1 & 0 \\ -1 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{b}^1 = \begin{pmatrix} 0 \\ 6 \\ -1 \\ -1 \\ 2 \\ 1 \\ 4 \end{pmatrix}$$

Note that constraints (1) and (2) are redundant for the description of P^1, and hence, can be discarded. A graphical representation of P^1 is given in Figure 4.16.

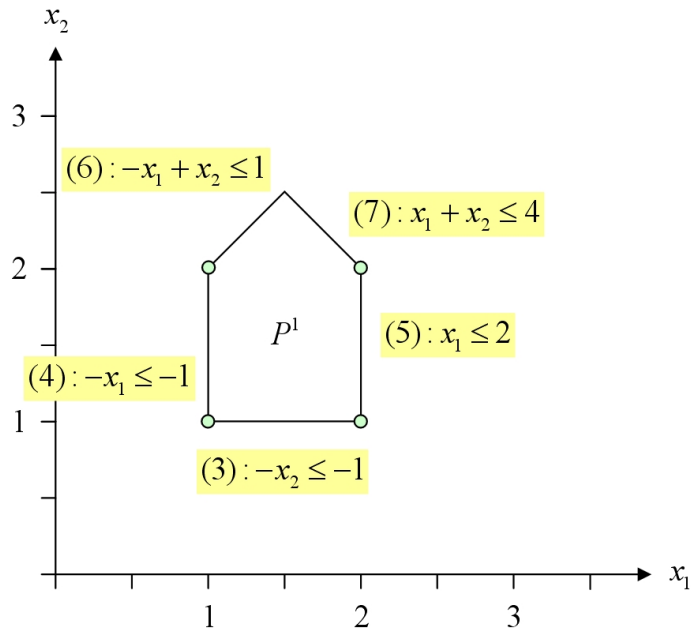


Figure 4.16: Polytope P^1

2. Construction of P^2

Starting with P^1, one can find (by trying) the following G-C cutting plane which is relevant for the description of $P^2 = (P^1)^{\mathcal{GC}}$:

$$\boldsymbol{u}^\top = (0,0,0,0,0,\tfrac{1}{2},\tfrac{1}{2}), \boldsymbol{\alpha}^\top = \boldsymbol{u}^\top \boldsymbol{A} = (0,1), \beta = \boldsymbol{u}^\top \boldsymbol{b} = 2\tfrac{1}{2}, \boldsymbol{\alpha}^\top \boldsymbol{x} \leq \lfloor \beta \rfloor : \qquad x_2 \leq 2 \tag{8}$$

The resulting polytope is given by $P^2 = \{\boldsymbol{x} \in P^1 : \boldsymbol{x} \text{ fulfils } (8)\} = \{\boldsymbol{x} \in \mathbb{R}^2 : x \text{ fulfils } (1) \ldots (8)\}$. After eliminating redundant constraints we obtain:

$$P^2 = \{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{x} \text{ fulfils } (3), (4), (5), (8)\}$$

Apparently, P^2 coincides with the integer hull of P, i.e. $P^2 = P_{\mathbb{Z}}$. Hence, polytope P has Chvátal Rank 2. A graphical representation of P^2 is given in Figure 4.17.

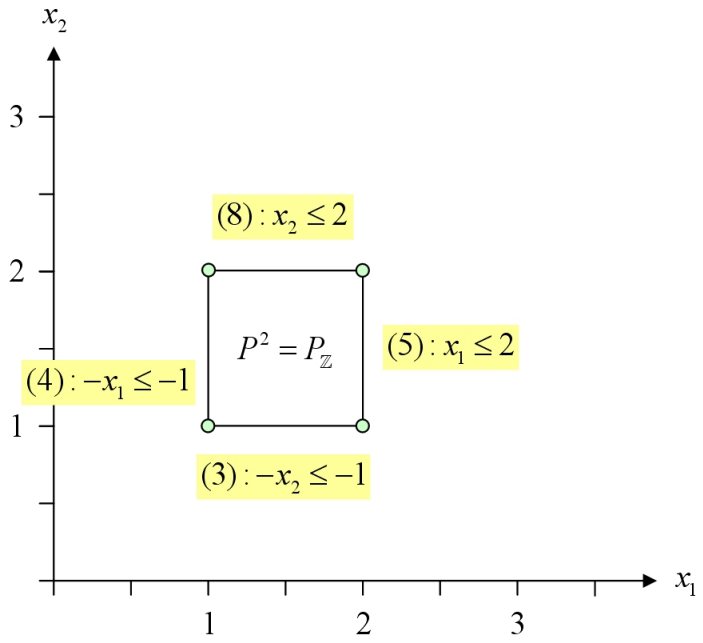


Figure 4.17: Polytope $P^2 = P_{\mathbb{Z}}$

Note that (as a hint for the Exercises), when deriving G-C cutting planes, the matrices $A, A^1, \ldots$ do not have to be written down explicitly. It is enough to specify which inequalities and which coefficients are used to produce the associated linear combination. For example, the derivation of constraint (7) can be written as:

$$\tfrac{1}{4}(1) + \tfrac{3}{4}(2) : x_1 + x_2 \leq 4\tfrac{1}{2} \qquad \Rightarrow \qquad x_1 + x_2 \leq 4$$

Bibliography

[1] Bazaraa, M.S., Jarvis, J.J., Hanif, D.S., Linear Programming and Network Flows, 3rd Edition, Wiley, New Jersey, 2004.

[2] Bertsimas, D., Tsitsiklis, J.N., Introduction to Linear Optimization, Athena Scientific, Belmont, 1997.

[3] Boyd, St., Vandenberghe, L., Convex Optimization, 6th Printing, Cambridge University Press, Cambridge, 2008. 37

[4] Chong, E.K.P, Zak, S.H., An Introduction to Optimization, Wiley, New York, 1996.

[5] Chvatal, V., Linear Programming, Freeman, New York, 1983.

[6] Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A., Combinatorial Optimization, Wiley, New York, 1998.

[7] Dantzig, G.B., Linear Programming and Extensions, Princeton, New Jersey, 1963.

[8] Domschke, W., Drexl, A., Einführung in Operations Research, 7th Edition, Springer, Berlin, 2007.

[9] Floudas, Ch.A., Pardalos, P.M., Encyclopedia of Optimization, 2nd Edition, Springer, New York, 2009.

[10] Griva, I., Nash, St.G., Sofer, A., Linear and Nonlinear Optimization, 2nd Edition, SIAM, Philadelphia, 2009.

[11] Grötschel, M., Lovasz, L., Schrijver, A., Geometric Algorithms and Combinatorial Optimiziation, Springer, Berlin, 1988. 58

[12] Hillier, F.S., Lieberman, G.J., Introduction to Operations Research, McGraw-Hill, 2009.

[13] Kallrath, J., Gemischt-ganzzahlige Optimierung: Modelle und Anwendungen: Mit Fallstudien aus Chemie, Energiewirtschaft, Metallgewerbe, Produktion und Logistik, Vieweg, 2002.

[14] Koop, A., Moock, H., Lineare Optimierung: Eine anwendungsorientierte Einführung in Operations Research, Spektrum Akademischer Verlag, 2007.

[15] Korte, B., Vygen, J., Combinatorial Optimization: Theory and Algorithms, 4th Edition, Springer, Berlin, 2008.

[16] Lee, J., A First Course in Combinatorial Optimization, Cambridge University Press, Edinburgh, 2004.

[17] Luenberger, D.G., Ye, Y., Linear and Nonlinear Programming, 3rd Edition, Springer, New York, 2008.

[18] Matousek, J., Gaertner, B., Understanding and Using Linear Programming, Springer, Berlin, 2007.

[19] Nemhauser, G.L., Wolsey, L.A., Integer and Combinatorial Optimization, Wiley, New York, 1988.

[20] Nemhauser, G.L., Rinnooy Kan, A.H.G., Todd, M.J. (Eds.), Handbooks in Operations Research and Management Science: Volume 1: Optimization, Elsevier, Amsterdam, 1989.

[21] Papadimitriou, Ch.H., Steiglitz, K., Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, New Jersey, 1982.

[22] Pochet, Y., Wolsey, L.A., Production Planning by Mixed Integer Programming, Springer, New York, 2006.

[23] Schrijver, A., Theory of Linear and Integer Programming, Wiley, Chichester, 1986.

[24] Schrijver, A., Combinatorial Optimization: Polyhedra and Efficiency, Springer, Berlin, 2003. 87

[25] Shapiro, J.F., Mathematical Programming: Structures and Algorithms, Wiley, 1979.

[26] Shapiro, R.D., Optimization Models for Planning and Allocation: Text and Cases in Mathematical Programming, Wiley, 1984.

[27] Vanderbei, R.J., Linear Programming, Foundations and Extensions, 3rd Edition, Springer, New York, 2008.

[28] Williams, H.P., Model Building in Mathematical Programming, 5th Edition, Wiley, West Sussex, 2013.

[29] Wolsey, L.A., Integer Programming, Wiley, New York, 1998.

[30] Zimmermann, H.-J., Operations Research: Methoden und Modelle, 2nd Edition, Vieweg, 2007.