Manuscript

Operations Research and Decision Analysis

Winter Semester 2025/2026

© Prof. Dr. Rainer Kolisch
Technical University of Munich
School of Management
Arcisstr. 21, 80333 München
http://www.ot.mgt.tum.de/om

This manuscript covers the material of the modules MGT001374 and MGT001464 'Operations Research & Decision Analysis' in the Bachelor program 'Management and Technology' and the Master program 'Master in Management'.

Manuscript

Operations Research and Decision Analysis

Winter Semester 2025/2026

© Prof. Dr. Rainer Kolisch
Technical University of Munich
School of Management
Arcisstr. 21, 80333 München
http://www.ot.mgt.tum.de/om

This manuscript covers the material of the modules MGT001374 and MGT001464 'Operations Research & Decision Analysis' in the Bachelor program 'Management and Technology' and the Master program 'Master in Management'.

Contents

Chapter 1

Decision Analysis

Literature:

- Eisenführ et al. [5].
- Hillier and Lieberman [7] Chapter 15 "Decision Analysis".
- Ravindran [9] Chapter 6 "Decision Analysis".
- Winston [11] Chapter 13 "Decision making under uncertainty".

1.1 Decision Situations

We can characterize decision situations according to the level of uncertainty into

- Decision under certainty: The future is known and described by a single scenario.
- Decision under uncertainty: There are $n > 1$ different scenarios for the future. There are no probabilities known for the scenarios.
- Decision under risk: There are n different scenarios for the future. Each scenario $j = 1, \ldots, n$ has a probability $0 < p_j < 1$ with $\sum_{j=1}^{n} p_j = 1$.

According to the number of objectives (goals, criteria) we distinguish between

- Single criterion decision problem
- Multi–criteria decision problem

According to the number of decision makers we differentiate between

- Single person decision
- Group decision

Regarding the time line we distinguish between

- Static decision: A decision is made for the future. The capital budgeting problem treated in Section 6.3 is a static decision.
- Dynamic decision: A number of decision for different times in the future such as periods $1, 2, ..., T$ are made. The dynamic ordering problem treated in Section 6.2 is a dynamic problem.

Regarding the behavior of the opponent we distinguish between

- Randomly acting opponent (game against nature)
- Strategically acting opponent (game theory)

1.2 Decision Making under Uncertainty

1.2.1 Decision Matrix and Efficient Alternatives

The decision situation is characterized as follows:

- Uncertainty
- One criterion
- Single decision maker
- Static decision

Alternatives (actions):

- There are m alternatives $a_1, a_2, \ldots, a_i, \ldots, a_m$.
- The decision maker has to select one alternative.

Scenarios (states of nature):

- There are n scenarios $s_1, s_2, \ldots, s_j, \ldots, s_n$.
- Exactly one scenario will happen.

Outcomes: If the decision maker selects alternative a_i and scenario s_j materializes the outcome is $e_{i,j}$.

Sequence of events:

1. Decision maker chooses alternative a_i
2. Scenario s_j materializes
3. The outcome is $e_{i,j}$

Decision matrix

	s_1	..	s_j	..	s_n
a_1	$e_{1,1}$	..	$e_{1,j}$	..	$e_{1,n}$
..					
a_i	$e_{i,1}$	..	$e_{i,j}$	..	$e_{i,n}$
..					
a_m	$e_{m,1}$	..	$e_{m,j}$	..	$e_{m,n}$

Figure 1.1: Decision matrix for a decision under uncertainty

Decision problem: Which alternative should the decision maker choose if he wants to maximize the outcome?

Example Newsvendor problem: A newsvendor is making his living by buying newspapers in the morning and selling them throughout the day. The cost for buying a newspaper is 20 Cent and selling a newswpaper he makes a revenue of 25 Cent. The demand for newspapers is between 6 and 10. Any newspaper not sold at the end of the day is worthless.

Alternatives are the number of newspaper bought in the morning a_i: $a_1 = 1, a_2 = 2, \ldots, a_m = m$

Scenarios are the number of customers who want to buy a newspaper throughout the day s_j: $s_1 = 6, s_2 = 7, \ldots, s_5 = 10$

If the newsvendor chooses alternative a_i and thus buys i newspapers in the morning and scenario s_j materializes, i.e., $j + 5$ customers want to buy a newspaper, then the resulting profit is:

$$e_{i,j} = \begin{cases} (25-20)\cdot i, & \text{if } i \leq j+5 \\ 25\cdot(j+5) - 20\cdot i, & \text{if } i > j+5 \end{cases}$$

or

$$e_{i,j} = 25\cdot\min(i, j+5) - 20\cdot i \quad \begin{pmatrix} i = 1,2,\ldots,m \\ j = 1,2,\ldots,5 \end{pmatrix}$$

By this we obtain the decision matrix given in Figure 1.2.

	s_j				
a_i	1	2	3	4	5
1	5	5	5	5	5
2	10	10	10	10	10
3	15	15	15	15	15
4	20	20	20	20	20
5	25	25	25	25	25
6	30	30	30	30	30
7	10	35	35	35	35
8	-10	15	40	40	40
9	-30	-5	20	45	45
10	-50	-25	0	25	50
11	-70	-45	-20	5	30
12	-90	-65	-40	-15	10

Figure 1.2: Decision matrix for the newsvendor problem

Definition 1 *An alternative a_i is* **efficient** *if no other alternative a_r exists for which*

$$e_{r,j} \geq e_{i,j} \text{ for all } j = 1,\ldots,n$$

and

$$e_{r,j} > e_{i,j} \text{ for at least one } j$$

holds. If there is an alternative a_r, then a_r **dominates** *a_i.*

Observation: Alternatives $a_1,\ldots,a_5$ as well as alternatives a_{11} and a_{12} are dominated by alternative a_6.

Dominated alternatives are eliminated from the decision matrix because there will always be one non-eliminated alternative which is better than a dominated alternative. After the

elimination of dominated alternative the remaining efficient alternatives are re–numbered from 1 onwards.

In order to select one of the efficient alternatives, different decision rules have been proposed. None of them is right or wrong. Rather, different decision rules reflect different attitudes towards decision making in an uncertain situation. For all rules we assume that wee seek to maximize the outcome.

1.2.2 Maximin Rule

For each alternative a_i we determine the minimum outcome over all scenarios:

$$\min_j \{e_{i,j}\}. \tag{1.1}$$

The maximim rule chooses the alternative, which maximizes the minimum (worst case) outcome. The maximin rule assures a minimum outcome which is 30 for the Newsvendor example. A decision maker applying the maximin rule is risk averse.

		s_j					
		1	2	3	4	5	min $\{e_{ij}\}$
	1	30	30	30	30	30	30 ←
	2	10	35	35	35	35	10
a_i	3	-10	15	40	40	40	-10
	4	-30	-5	20	45	45	-30
	5	-50	-25	0	25	50	-50

Figure 1.3: Maximim rule applied to the Newsvendor problem

1.2.3 Maximax Rule

		s_j					
		1	2	3	4	5	max $\{e_{ij}\}$
	1	30	30	30	30	30	30
	2	10	35	35	35	35	35
a_i	3	-10	15	40	40	40	40
	4	-30	-5	20	45	45	45
	5	-50	-25	0	25	50	50 ←

Figure 1.4: Maximax rule applied to the Newsvendor problem

For each alternative a_i we determine the maximum outcome over all scenarios:

$$\max_j \{e_{i,j}\}. \tag{1.2}$$

The alternative which gives way to the maximum (best case) outcome is selected. The maximax rule provides the chance for the maximum outcome which is 50 for the Newsvendor example. A decision maker applying the maximax rule is risk seeking.

1.2.4 Hurwicz Rule

Leonid Hurwicz, 2007 recipient of the Nobel price in economics proposed a rule, which allows a weighting of the minimum and maximum attainable outcome. Using weighting parameter $\lambda \in [0, 1]$ for each alternative a_i we calculate

$$e_i = \lambda \cdot \max_j \{e_{i,j}\} + (1 - \lambda) \cdot \min_j \{e_{i,j}\} \tag{1.3}$$

and choose the alternative with maximum e_i.

		s_j							
		1	2	3	4	5	$\max_j \{e_{i,j}\}$	$\min_j \{e_{ij}\}$	e_i
a_i	1	30	30	30	30	30	30	30	30
	2	10	35	35	35	35	35	10	32,5
	3	-10	15	40	40	40	40	-10	35
	4	-30	-5	20	45	45	45	-30	37,5
	5	-50	-25	0	25	50	50	-50	40 ←

Figure 1.5: Hurwicz rule with $\lambda = 0.9$ applied to the Newsvendor problem

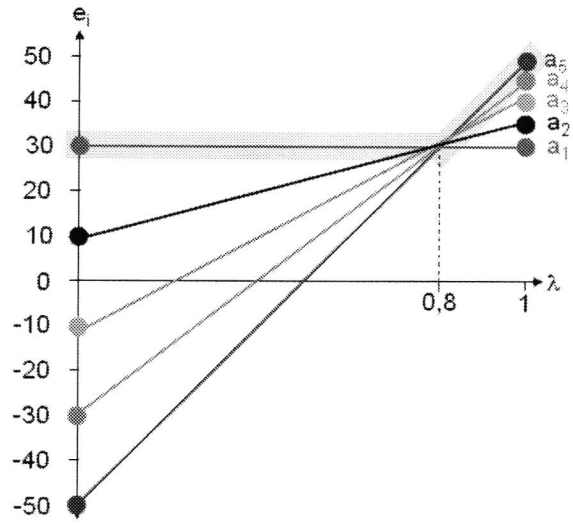


Figure 1.6: e_i–values of the alternatives depending on λ

The parameter λ is termed "optimism parameter" because for $\lambda = 1$ we obtain the maximax rule while for $\lambda = 0$ we obtain the minimax rule. Figure 1.5 gives the application of the Hurwicz rule for a parameter of $\lambda = 0.9$. Figure 1.6 provides the e_i–values for all alternatives and all values of λ. For the Newsvendor example, alternative a_1 is the choice for $0 \leq \lambda < 0.8$ while a_5 should be selected for $0.8 < \lambda \leq 1$. For $\lambda = 0.8$ each of the alternatives could be selected.

1.2.5 Minimax–Regret Rule

As the name suggests, the minimax–regret rule applies the minimax–principle to the regret value. The regret associated with an alternative and a scenario is the difference in outcome achieved vs. the outcome which could have been obtained if the alternative leading to the maximum outcome for the scenario would have been selected.

Using the minimax–regret rule is done in three steps:

1. Calculating the best outcome for each scenario

$$e_j^* = \max_i \{e_{i,j}\} \tag{1.4}$$

2. Calculating the regret matrix

$$r_{i,j} = e_j^* - e_{i,j} \tag{1.5}$$

3. Applying the minimax rule to the regret matrix

		s_j				
		1	2	3	4	5
	1	30	30	30	30	30
	2	10	35	35	35	35
a_i	3	-10	15	40	40	40
	4	-30	-5	20	45	45
	5	-50	-25	0	25	50
$\max\{e_{ij}\}$		30	35	40	45	50

Figure 1.7: Step 1 of the minimax–regret rule: Calculation of the maximum outcome for each scenario

		s_j					
		1	2	3	4	5	$\max\{r_{ij}\}$
	1	0	5	10	15	20	20 ←
	2	20	0	5	10	15	20 ←
a_i	3	40	20	0	5	10	40
	4	60	40	20	0	5	60
	5	80	60	40	20	0	80

Figure 1.8: Step 2 and 3 of the minimax–regret rule: Regret matrix and selection of the alternative

A decision maker which follows the minimax–regret rule wants to minimize the maximum regret when selecting an alternative.

1.2.6 Laplace Rule

The assumption of the Laplace rule is that each scenario has the same probability $\frac{1}{n}$. We then can calculate the expected outcome when selecting alternative a_i as

$$e_i = \frac{1}{n} \sum_j e_{i,j}. \tag{1.6}$$

The alternative with the maximum expected outcome is selected. A decision maker applying the Laplace rule is risk neutral.

		s_j 1	2	3	4	5	$1/5 \cdot \Sigma e_{i,j}$	
	1	30	30	30	30	30	30	←
	2	10	35	35	35	35	30	←
a_i	3	-10	15	40	40	40	25	
	4	-30	-5	20	45	45	15	
	5	-50	-25	0	25	50	0	

Figure 1.9: Laplace rule applied to the Newsvendor problem

1.3 Decision Making under Risk: Expected Utility Theory

Decision situation:

- Risk
- Single criterion
- Single decision maker
- Static decision

We extend the decision matrix introduced in Section 1.2 by the probability p_j for scenario s_j. It has to hold that $0 < p_j < 1$ and $\sum_j^n p_j = 1$.

	p_1 .. p_j .. p_n
	s_1 .. s_j .. s_n
a_1	$e_{1,1}$ .. $e_{1,j}$ .. $e_{1,n}$
..	
a_i	$e_{i,1}$ .. $e_{i,j}$ .. $e_{i,n}$
..	
a_m	$e_{m,1}$ .. $e_{m,j}$ .. $e_{m,n}$

Figure 1.10: Decision matrix for decision making under risk

Example: Financial investment. A decision maker can choose between

- a bond for which we will receive 10,000 at the end of the year or
- a share which with probability of 50% will be valued with 30,000 or with 50% will be valued with 0 at the end of the year.

Decision matrix for the financial investment decision:

	0.5	0.5
	s_1	s_2
a_1 (Bond)	10,000	10,000
a_2 (Share)	30,000	0

Figure 1.11: Decision matrix for the financial investment

1.3.1 Lotteries

Each alternative a_i in a decision under risk can be depicted as a lottery L_i. Lottery L_i can have n possible outcomes $e_{i,1}, \ldots, e_{i,n}$ with associated probabilities $p_1, \ldots, p_n$ where $0 < p_j < 1$ and $\sum_j p_j = 1$ has to hold (see above). The lottery is denoted as

$$L_i = (p_1, e_{i,1}; \ldots; p_j, e_{i,j}; \ldots; p_n, e_{i,n}) \tag{1.7}$$

and can be graphically depicted as given in Figure 1.12

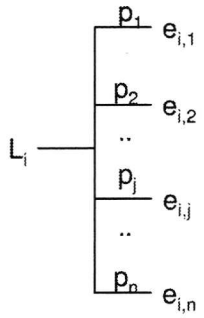


Figure 1.12: Lottery

The two financial investment are denoted as $L_1(\text{Bond}) = (1, 10,000)$ and $L_2(\text{Share}) = (0.5, 30,000; 0.5, 0)$. The graphical representation is given in Figure 1.13. Which one of the two lotteries should a decision maker pick? Obviously there is not one decision which fits for each decision maker. Rather, the decision depends on the decision maker and more precisely on his risk attitude. We thus need a method which allows the decision maker to value and thus compare and select lotteries taking into account his risk attitude.

1.3.2 St. Petersburg–Paradoxon

Using the Laplace–approach we could value a lottery by its expected value

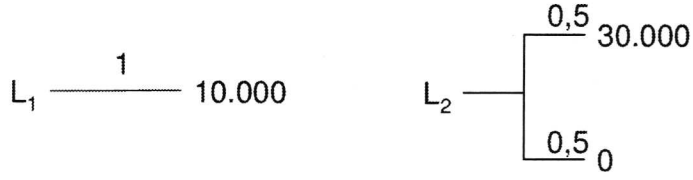


Figure 1.13: Lotteries for bond and share

$$EV(L_i) = \sum_j p_j \cdot e_{i,j}$$

The expected values for the two financial investment lotteries are $EV(L_1) = 1 \cdot 10.000 = 10.000$ and $EV(L_2) = 0.5 \cdot 30,000 + 0.5 \cdot 0 = 15,000$. Based on the expected value we have to select lottery 2 (the share).

In order to illustrate the downside of the Laplace-criterion for selecting risky alternatives, let us consider the following lottery: A fair coin is flipped until tail (i.e. number) shows up. Then, the person who has chosen the lottery (the player) receives 2^n EURO where n is the number of times the coin has been flipped. Thus, flips $1, \ldots, n-1$ showed heads and the last flip showed tail. If we limit the number of flips to 10 we obtain the lottery shown in Figure 1.14.

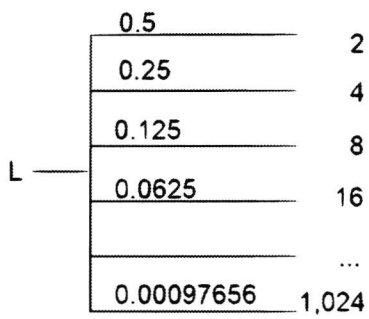


Figure 1.14: St. Petersburg lottery

The probability distribution $f(x)$ for a payment of x with a limit of 10 flips is given in Figure 1.15.

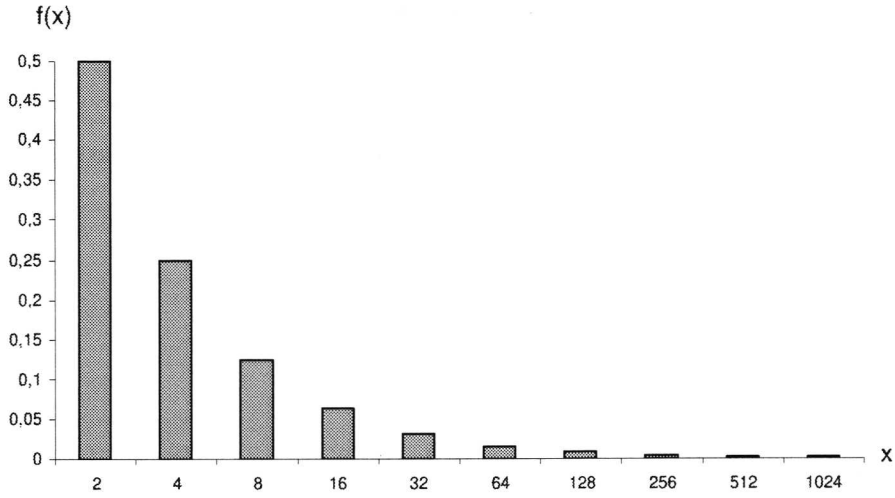


Figure 1.15: Probability distribution of the St. Petersburg lottery

The expected value of the lottery with a limit of 10 flips is

$$EV(L) = \sum_{j=1}^{10} \left(\frac{1}{2}\right)^j \cdot 2^j = \sum_{j=1}^{10} 1^j = 10$$

Without a flip limit the expected value of the lottery is

$$EV(L) = \sum_{j=1}^{\infty} \left(\frac{1}{2}\right)^j 2^j = \sum_{j=1}^{\infty} 1^j = \infty$$

Although the expected value is infinite the vast majority of decision makers are willing to exchange the lottery with a certain payment of 50 EURO. This indicates that the expected value is not (always) a suited method in order to value lotteries.

1.3.3 Utility Function and Expected Utility

The concept of utility functions has been proposed by Bernoulli in 1738 and in 1947 refined by von Neumann and Morgenstern.

Definition 2 *The* **expected utility** *of a lottery L_i is*

$$EU(L_i) = \sum_j p_j \cdot u(e_{i,j}) \tag{1.8}$$

where $u(x)$ is the utility function. The utility function transforms each risky outcome $e_{i,j}$ into a utility in the interval $[0, 1]$. Let e^+ be the maximum outcome and let e^- be the minimum outcome. Then, the following holds for the utility function $u(x)$:

1. $u(e^-) = 0$
2. $u(e^+) = 1$
3. $u(a) < u(b)$ if $e^- \leq a < b \leq e^+$

Let us consider three characteristic utility functions for a lottery with $e^- = 10$ and $e^+ = 100$.

Linear utility function

$$u(x) = -\frac{1}{9} + \frac{1}{90} \cdot x$$

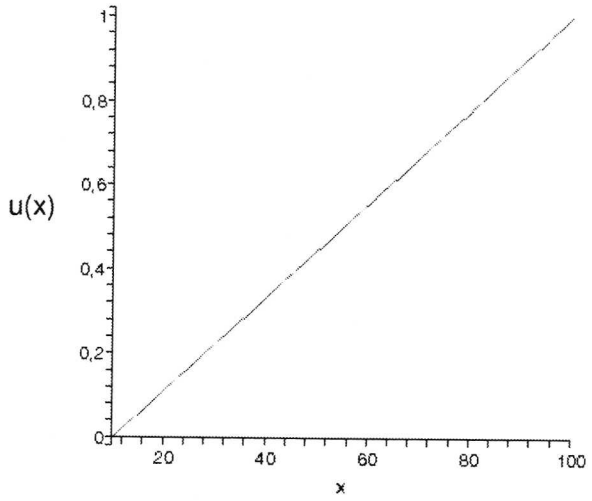


Figure 1.16: Linear utility function

Concave utility function

$$u(x) = -0,46 + 0,146x^{0,5}$$

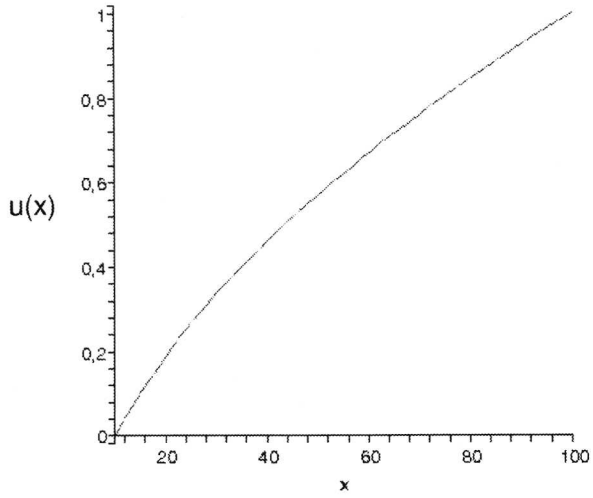


Figure 1.17: Concave utility function

Convex utility function

$$u(x) = -\frac{1}{99} + \frac{1}{9900} \cdot x^2$$

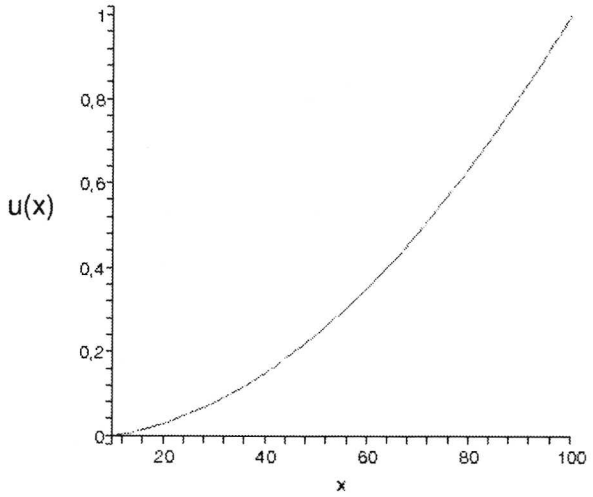


Figure 1.18: Convex utility function

If the utility function of a decision maker is known, he can compare two alternatives (lotteries) as follows:

- $a_1 \succ a_2$: Alternative a_1 is preferred over alternative a_2 if $EU(a_1) > EU(a_2)$.
- $a_1 \prec a_2$: Alternative a_2 is preferred over a_1 if $EU(a_1) < EU(a_2)$ holds.
- $a_1 \sim a_2$: The decision maker is indifferent between alternatives a_1 and a_2 if $EU(a_1) = EU(a_2)$ holds.

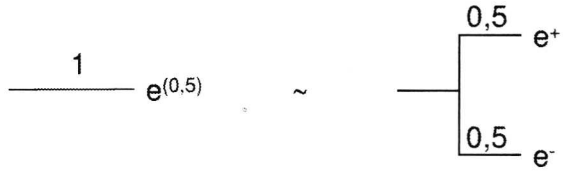
1.3.4 Generating a Utility Function

Using the St. Petersburg lottery we will now show how we can generate a decision maker specific utility function. For this we will be using the so–called "certainty equivalent method" (the name will soon become apparent). We assume that we have a maximization problem.

1. Determine
   - the maximum possible outcome e^+ and
   - the minimum possible outcome e^-.
2. The decision maker has to determine the certain outcome $e^{(0.5)}$ for which he is indifferent to a lottery with 50% probability to recieve e^+ and 50% probability to receive e^-, i.e.,

   $$L = (1, e^{(0.5)}) \sim L = (0.5, e^+; 0.5, e^-).$$

   The certain outcome for which the decision maker is indifferent to the lottery is called "certainty equivalent". Note that each lottery has it's specific certainty equivalent.

   Since $L_1 \sim L_2$ holds, $EU(L_1) = EU(L_2)$ has to to hold as well and thus $u(e^{(0.5)}) = 0.5 \cdot u(e^+) + 0.5 \cdot u(e^-) = 0.5 \cdot 1 + 0.5 \cdot 0 = 0.5$. The fact that we are obtain a utility value of 0.5 is the reason why we put "0.5" in the superscript.
3. Analogously, we determine the outcome $e^{(0.25)}$ by $L = (1, e^{(0.25)}) \sim L = (0.5, e^{(0.5)}; 0.5, e^-)$ and the outcome $e^{(0.75)}$ by $L = (1, e^{(0.75)}) \sim L = (0.5, e^+; 0.5, e^{(0.5)}$ with utility $u(e^{(0.25)}) = 0.25$ and $u(e^{(0.75)}) = 0.75$, respectively.
4. If necessary, further points $(e^{(0.125)}, 0.125)$, $(e^{(0.875)}, 0.875)$, ... can be determined.

Example: Determination of the utility function for the St. Petersburg lottery restricted to 10 coin flips.

1. $e^- = 2$, $e^+ = 1,024$
2. $L = (1, e^{(0.5)}) \sim L = (0.5, 1,024; 0.5, 2)$

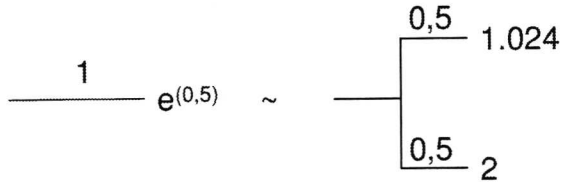


Figure 1.19: Deriving $e^{(0.5)}$

Result: $e^{(0.5)} = 400$

3. $L = (1, e^{(0.25)}) \sim L = (0.5, 400; 0.5, 2)$

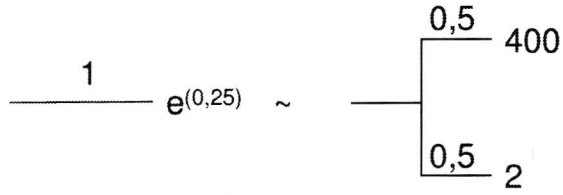


Figure 1.20: Deriving $e^{(0.25)}$

Result: $e^{(0.25)} = 150$

4. $L = (1, e^{(0.75)}) \sim L = (0.5, 1.024; 0.5, 400)$

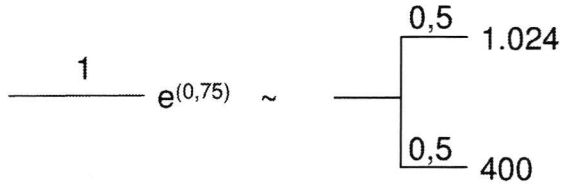


Figure 1.21: Deriving $e^{(0.75)}$

Result: $e^{(0.75)} = 600$

We thus obtain the following utility function:

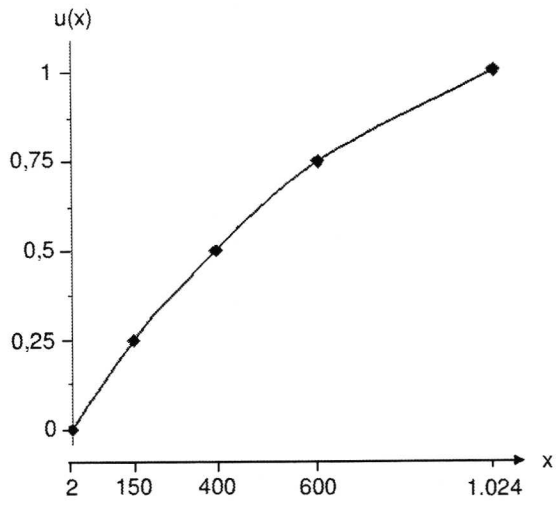


Figure 1.22: Resulting utility function

1.3.5 Risk Attitude of a Decision Maker

Definition 3 *The* **certainty equivalent** *CE of a lottery L is the outcome for which the decision maker is indifferent between the risky lottery and the certain outcome CE:*

$$(1, CE(L)) \sim L \tag{1.9}$$

Definition 4 *The* **risk premium** *(RP) is the difference between the expected value of the lottery* $(EV(L) = \sum_j p_j \cdot e_j)$ *and the certainty equivalent CE:*

$$RP(L) = EV(L) - CE(L) \tag{1.10}$$

Definition 5 *If the second derivative of the utility function* $u(x)$ *exists and the first derivative of* $u(x)$ *at* x *with* $e^- \leq x \leq e^+$ *is not zero, the risk attituce of the decision maker at* x *can be determined with the* **Arrow–Pratt measure** *(AP):*

$$AP(x) = -\frac{u''(x)}{u'(x)}$$

The third property of a utility function as given above states that the utility function is monotonically increasing. Hence, we have $u'(x) > 0$. In case the utility function is linear we have $u''(x) = 0$ and thus $AP(x) = 0$. For a utility function which is concave we have $u''(x) < 0$ (the slope and thus the increase in utility becomes smaller with increasing x) and thus $AP(x) > 0$. For a convex utility function we have $u''(x) > 0$ (the slope and thus the increase in utility becomes larger with increasing x) and thus we have $AP(x) < 0$.

We distinguish between the following three risk attitudes:

- Risk averse
- Risk seeking
- Risk neutral

Let us summarize these risk attitudes in terms of the concepts introduced:

Risk averse decision maker:

- Concave utility function.
- For all $x \in [e^-, e^+]$ we have $AP(x) > 0$.
- For a lottery L=(0.5, e^+; 0.5, e^-) the decision maker will have a certainty equivalent CE < EV. Note that this leads to the concave shape of the utility function given in Figure 1.23.
- The risk premium RP is positive.
- An example for a risk averse decision is to buy an insurance contract.

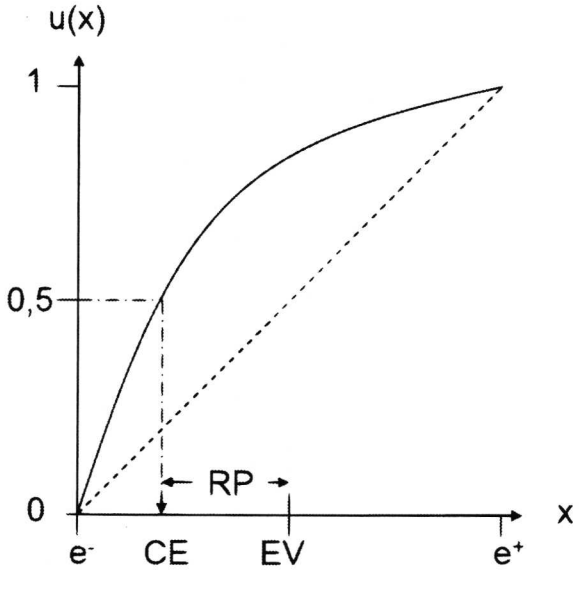


Figure 1.23: Utility function of a risk averse decision maker

Risk seeking decision maker:

- Convex utility function.
- For all $x \in [e^-, e^+]$ we have $AP(x) < 0$.

- For a lottery L=(0.5, e^+; 0.5, e^-) the decision maker will have a certainty equivalent CE > EV which leads to the convex shape of the utility function given in Figure 1.24.
- The risk premium RP is negative.
- An example for a risk seeking decision is to take part in a gamble.

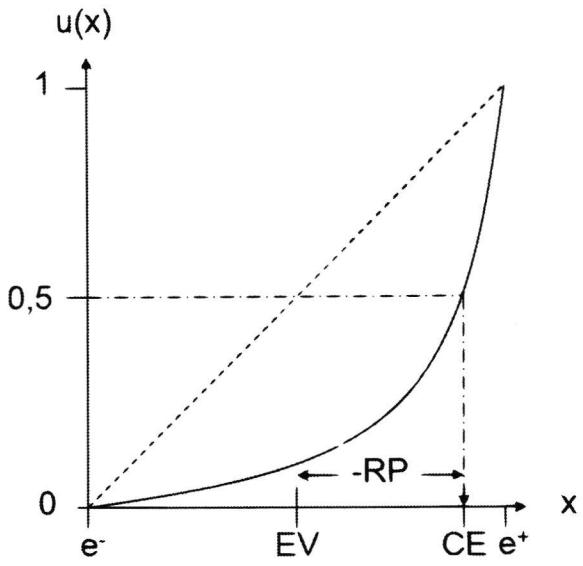


Figure 1.24: Utility function of the risk seeking decision maker

Risk neutral decision maker:

- Linear utility function.
- For all $x \in [e^-, e^+]$ we have $AP(x) = 0$.
- For a lottery L=(0.5, e^+; 0.5, e^-) the decision maker will have a certainty equivalent CE = EV which leads to the linear utility function given in Figure 1.25.
- The risk premium RP is zero.

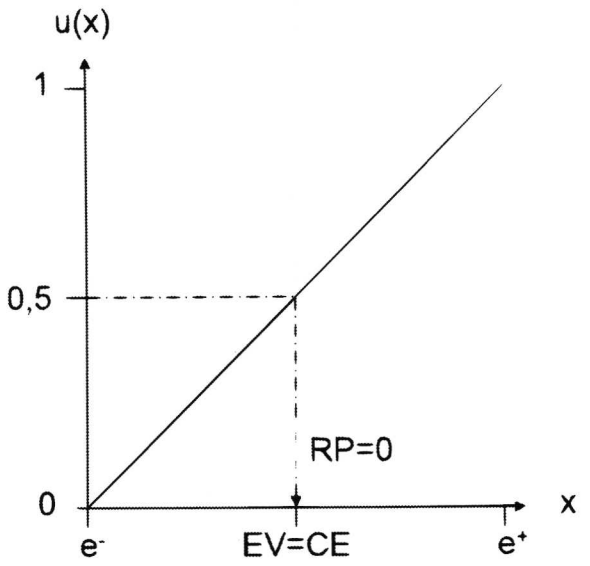


Figure 1.25: Utility function of a risk neutral decision maker

Utility Function of Friedman and Savage

Friedman and Savage [6] have empirically observed utility functions with a concave and a convex section (see Figure 1.26).

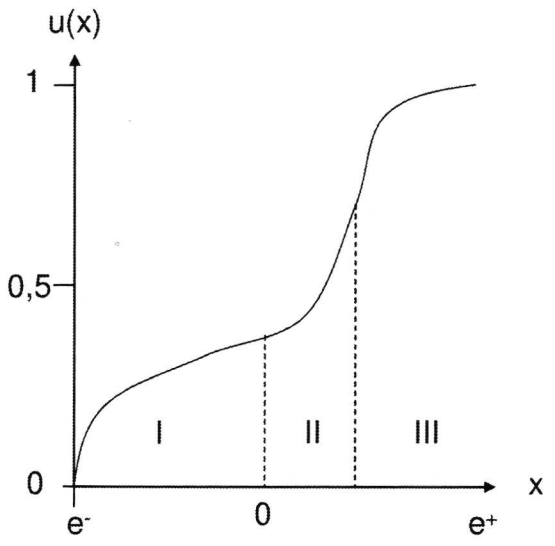


Figure 1.26: Utility function of Friedman and Savage

- In section I the decision maker is risk averse. E.g., he wants to avoid losses by signing an insurance contract.

- In section II the decision maker is risk seeking. E.g., he takes part in gambling.
- In section III the decision maker is risk averse. E.g., by investing money in bonds rather than in shares.

1.3.6 Decision Matrix and Expected Utility

Having a decision matrix we can apply the concept of expected utility by calculating for each alternative the expected utility and then selecting the alternative which maximizes the expected utility.

	p_1 .. p_j .. p_n
	s_1 .. s_j .. s_n
a_1	$e_{1,1}$ .. $e_{1,j}$ .. $e_{1,n}$
..	
a_i	$e_{i,1}$ .. $e_{i,j}$ .. $e_{i,n}$
..	
a_m	$e_{m,1}$ .. $e_{m,j}$ .. $e_{m,n}$

Figure 1.27: Decision matrix in a risk situation

For this we proceed as follows:

Applying expected utility theory to a decision matrix

1. Deriving the utility function $u(x)$ based on the outcomes $e_{i,j}$ of the decision matrix.
2. Transforming the decision matrix with outcomes $e_{i,j}$ into a decision matrix with utlities $u(e_{i,j})$.
3. Calculating for each alternative a_i the expected utility $EU(a_i)$:

$$EU(a_i) = \sum_j p_j \cdot u(e_{i,j})$$

4. Selecting the alternative with maximum expected utility.

Example: Applying expected utility to the decision matrix given in Figure 1.28 with maximum outcome $e^+ = 10$ and minimum outcome $e^- = 2$. Figure 1.29 shows the selection of optimal alternative for a risk seeking decision maker with utility $u(x) = \left(\frac{x-2}{8}\right)^2$. Figure 1.30 shows the selection of the optimal alternative for a risk averse decision maker with utility function $u(x) = \sqrt{\frac{x-2}{8}}$. Finally, Figure 1.31 shows the selection of the optimal alternative for a risk neutral decision maker with utility function $u(x) = -\frac{1}{4} + \frac{1}{8}x$.

	p_j	0,1	0,2	0,5	0,2
	s_j	1	2	3	4
a_i	1	2	2	6	10
	2	6	3	5	4
	3	4	8	4	5
	4	3	9	5	2

Figure 1.28: Example decision matrix in a risk situation

	p_j	0,1	0,2	0,5	0,2	EU(a_i)
	s_j	1	2	3	4	
a_i	1	0,00	0,00	0,25	1,00	0,33 ←
	2	0,25	0,02	0,14	0,06	0,11
	3	0,06	0,56	0,06	0,14	0,18
	4	0,02	0,77	0,14	0,00	0,23

Figure 1.29: Selecting an alternative with utility function $u(x) = \left(\frac{x-2}{8}\right)^2$

1.4 Decision Making under Risk: The μ–σ–Criterion

The μ–σ–criterion is a method common in practice. It employs for each alternative the two measures expected value μ (see Equation 1.11) and variance σ^2 (see Equation 1.12). The variance measures the extend to which an outcome may deviate from the expected value. The standard deviation is $\sigma = \sqrt{\sigma^2}$. Figure 1.32 gives the expected values and the variances of the alternatives for the example.

$$\mu(a_i) = \sum_j p_j \cdot e_{i,j} \tag{1.11}$$

$$\sigma^2(a_i) = \sum_j p_j \cdot (\mu(a_i) - e_{i,j})^2 \tag{1.12}$$

	p_j	0,1	0,2	0,5	0,2	$\mu(a_i)$	$\sigma^2(a_i)$
	s_j	1	2	3	4		
a_i	1	2	2	6	10	5,6	7,84
	2	6	3	5	4	4,5	0,85
	3	4	8	4	5	5	2,4
	4	3	9	5	2	5	5,4

Figure 1.32: Expected value and variance of the alternatives

The μ–σ–criterion calculates for each alternative a_i a preference value $\Phi(a_i)$. $\Phi(a_i)$ is a function of μ and σ. The alternative a_i with maximum preference value $\Phi(a_i)$ is selected.

	p_j	0,1	0,2	0,5	0,2	$EU(a_i)$	
	s_j	1	2	3	4		
	1	0,00	0,00	0,71	1,00	0,55	
	2	0,71	0,35	0,61	0,50	0,55	
a_i	3	0,50	0,87	0,50	0,61	0,60	←
	4	0,35	0,94	0,61	0,00	0,53	

Figure 1.30: Selecting an alternative with utility function $u(x) = \sqrt{\frac{x-2}{8}}$

	p_j	0,1	0,2	0,5	0,2	$EU(a_i)$	
	s_j	1	2	3	4		
	1	0,00	0,00	0,50	1,00	0,45	←
	2	0,50	0,13	0,38	0,25	0,31	
a_i	3	0,25	0,75	0,25	0,38	0,38	
	4	0,13	0,88	0,38	0,00	0,38	

Figure 1.31: Selecting an alternative with utiltiy function $u(x) = -\frac{1}{4} + \frac{1}{8}x$

Risk neutral decision maker A risk neutral decision maker is only taking into account the expected value. Hence, his preference function is

$$\Phi(a_i) = \mu(a_i) \tag{1.13}$$

In the example given in Figure 1.28 the risk neutral decision maker chooses alternative a_1. He is indifferent between a_3 and a_4. Figure 1.33 provides the information on the calculation of the preference function. Note that a for a risk neutral decision maker the μ–σ–criterion reduces to the μ–criterion (expected value criterion). The expected value criterion will lead to the same alternative as the expected utility criterion.

	p_j	0,1	0,2	0,5	0,2	$\mu(a_i)$	$\sigma^c(a_i)$	$\Phi(a_i)$	
	s_j	1	2	3	4				
	1	2	2	6	10	5,6	7,84	5,6	←
	2	6	3	5	4	4,5	0,85	4,5	
a_i	3	4	8	4	5	5	2,4	5	
	4	3	9	5	2	5	5,4	5	

Figure 1.33: Decision of a risk neutral decision maker with the μ–σ–criterion

Risk seeking decision maker The risk seeking decision maker is interested in the expected value and in the amount the expected value can be exceeded. He is therefore interested in a large μ and a large σ. A concrete function for $\Phi()$ is

$$\Phi(a_i) = \mu(a_i) + 0.5 \cdot \sigma^2(a_i) \tag{1.14}$$

A risk seeking decision maker with preference function (1.14) chooses alternative a_1. Regarding alternatives a_3 and a_4 the decision maker prefers alternative a_4 because it has the higher σ. Figure 1.34 provides the details of the calculations.

	p_j	0,1	0,2	0,5	0,2	$\mu(a_i)$	$\sigma^2(a_i)$	$\Phi(a_i)$	
	s_j	1	2	3	4				
a_i	1	2	2	6	10	5,6	7,84	9,52	←
	2	6	3	5	4	4,5	0,85	4,93	
	3	4	8	4	5	5	2,4	6,20	
	4	3	9	5	2	5	5,4	7,70	

Figure 1.34: Decision of a risk seeking decision maker with the μ–σ–criterion

Risk averse decision maker The risk averse decision maker is interested in a large expected outcome μ and in a small σ because he views σ as a threat that μ is decreased. A concrete function for $\Phi()$ is

$$\Phi(a_i) = \frac{3}{2} \cdot \mu(a_i) - 2 \cdot \sigma(a_i) \tag{1.15}$$

The risk averse decision maker with preference function (1.15) chooses alternative a_2. Regarding alternatives a_3 and a_4 the decision maker prefers a_3 because it has the smaller σ. Figure 1.35 provides the details of the calculations.

	p_j	0,1	0,2	0,5	0,2	$\mu(a_i)$	$\sigma^2(a_i)$	$\Phi(a_i)$	
	s_j	1	2	3	4				
a_i	1	2	2	6	10	5,6	7,84	2,80	
	2	6	3	5	4	4,5	0,85	4,91	←
	3	4	8	4	5	5	2,4	4,40	
	4	3	9	5	2	5	5,4	2,85	

Figure 1.35: Decision of a risk averse decision maker with the μ–σ–criterion

1.5 Dynamic Decision Making under Risk: Decision Tree

Characterization of the decision situation:

- Risk

- One criterion
- Single decision maker
- Dynamic decision

In a dynamic decision context, the decision maker has to make a series of decisions. Between two successive decision one scenario will take place. Let us give an example with the introduction of new product to the market. We assume that the product has been developed. Now the decision maker faces the following alternative decisions:

1. Obtain information about the chances of the product on the market by testing the product on a test market (costs of 30) and undertaking the decision about introducing the product to the market (or not) afterwards.
   - If the result of the test market is positive (60% probability) there is a 85% probability that the product will make a profit of 300 and a 15% probability that the product will make a loss of 100 when introduced to the market.
   - If the result of the test market is negative (40% probability) there is a 10% probability that the product will make a profit of 300 and a probability of 90% that the product will make a loss of of 100.
2. Introduction of the product to the market without having a market study. In this case there is a probability of 55% to make a profit of 300 and a probability of 45% to make a loss of 100.
3. No introduction of the product to the market.

What sequence of decisions should the decision maker undertake in order to maximize its profit? In what follows we will model and solve the problem with a decision tree.

1.5.1 Elements of a Decision Tree

A decision tree is an undirected graph with three node types

- decision node (represented as square),
- chance nodes (represented as circle) and
- outcome node (represented as triangle)

and edges which connect two nodes. Figure 1.36 gives the decision tree for the product introduction example.

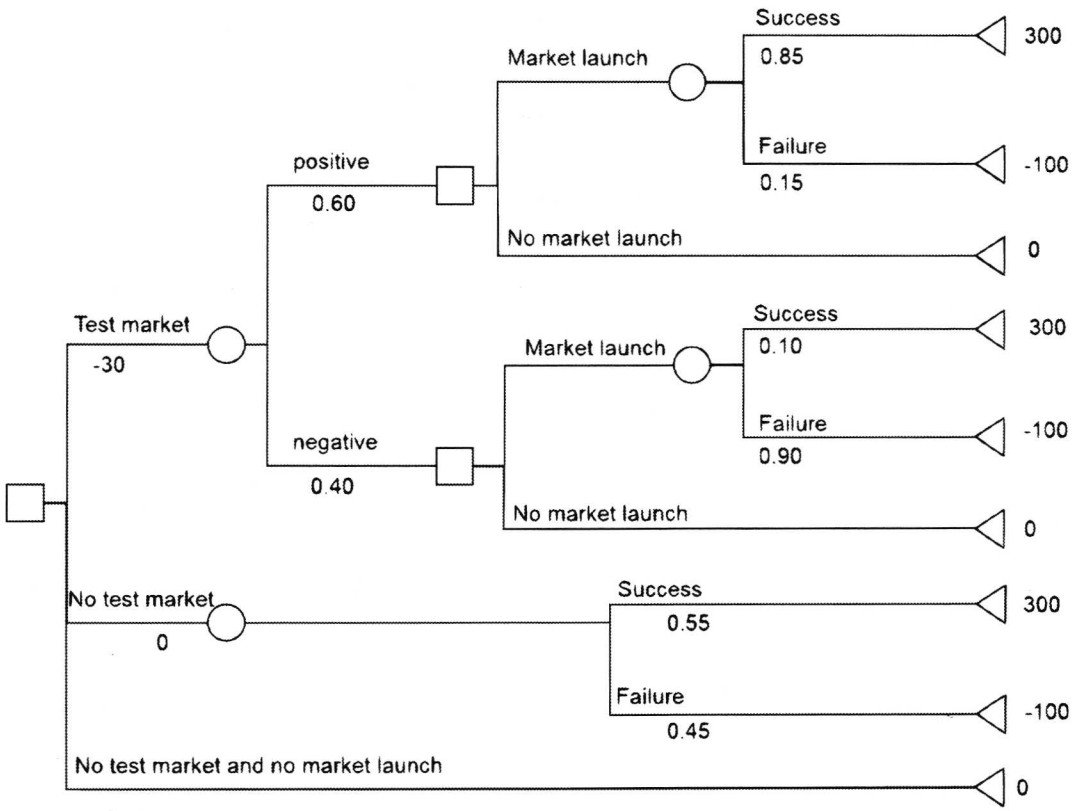


Figure 1.36: Decision tree for the new product introduction

1.5.2 Solving a Decision Tree

We assume a risk neutral decision maker and hence can apply the expected value criterion.

- The decision tree is solved backwards, that is we start with the result nodes on the right side.
- For each final chance node (i.e. there are no more chance nodes on the right side connected by edges) we calculate the associated expected value.
- For each final decision node we select the decision (that is the edge connecting the decison node on the left with a chance node or an outcome node on the right) with the maximum expected value. The arcs of non selected decision are marked with "//". The value of the decision node is set to the value of the selected decision.

Due to the backward solution of the decision tree, the method is termed "roll back procedure".

Solving the decision tree of the new product introduction

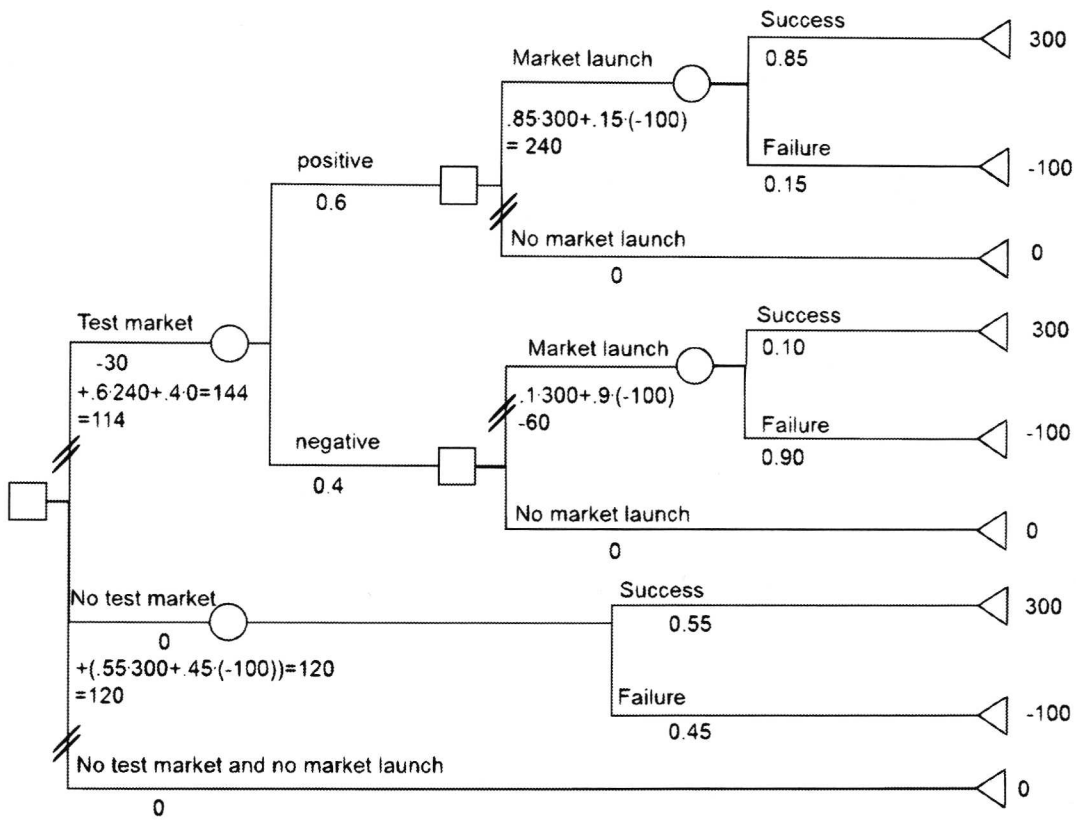


Figure 1.37: Solving the new product introduction to optimality

Remark: Solving a decision tree with the roll back procedure provides optimal decisions for each decision node. Hence, even if the realization of a chance node is unfavorable we will obtain the optimal decision for proceeding. E.g., assume that we have decided to run the test market and that the result is negative. In this case it is optimal not to introduce the product to the market. The fact that we will obtain an optimal decision for each node is termed "flexible planning".

Observation: The optimal decision for the risk neutral decision maker is to introduce the product to the market without test market. However, making this decision goes along with a 45% probability to make a loss of 100. In case of a test market the probability to make a loss of of 130 (30 costs for the test market and 100 loss for the failed market introduction afterward) is down to $0.6 \cdot 0.15 = 9\%$ (probability of a positive result from the test market and a loss on the market).

1.5.3 Decision Tree Software

There are a number of decision tree software systems on the market. One is Precision Tree from Palisade (`http://www.palisade.com/precisiontree/`), see Figure 1.38.

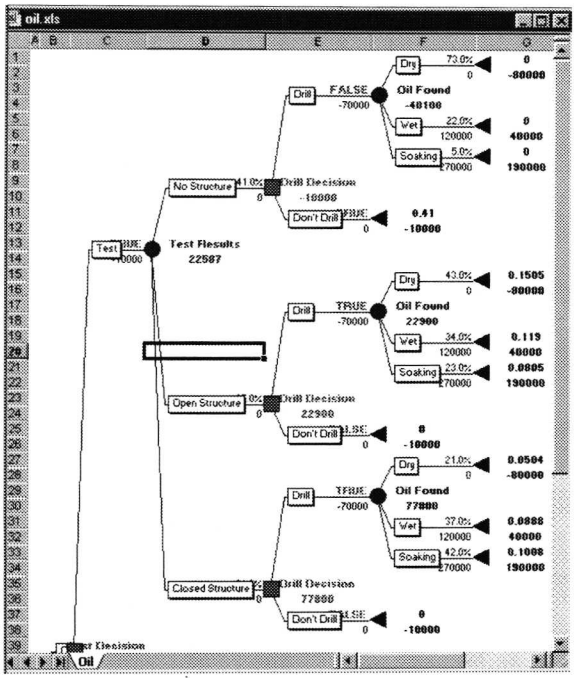


Figure 1.38: Screenshot of the "Precision Tree" software

1.5.4 Value of Information

Value of the test market

In order to determine the value of the test market study let us assume that the test market study is free instead of costing 30. Solving the decision tree with a free test market reduces the cost by 30 and thus raises the value to 144.

Now let us assume that the test market cannot be conducted. We thus delete the edge "test market" as well as all successor nodes and egdges (going from left to right) from the tree. Solving the resulting decision tree to optimality gives a value of 120. The difference $144 - 120 = 24$ is the value of the test market. This is the maximum costs which should be spend for the test market.

Value of perfect information

The test market helps to reduce the risk about knowing if we will be successful on the market. That is, a positive result increases the success probability from 55% to 85%. However, the test market does not eliminate the risk at all. The question arises about the value of perfect

information. The perfect information would mean that the test market tells us if we will be successful or not with a probability of 100%. With a perfect test market we could thus retrieve the information about the outcome of the chance node on the market success first and could then make a decision to introduce the product to the market afterwards. This sequence of chance node and decision node is given in the decision tree depicted in Figure 1.39. Solving this decision tree, we obtain a value of of 165. Hence, the value of perfect information is $165 - 120 = 45$. If there would be a firm which could provide the perfect information, the company would be willing to pay up to 45 for this.

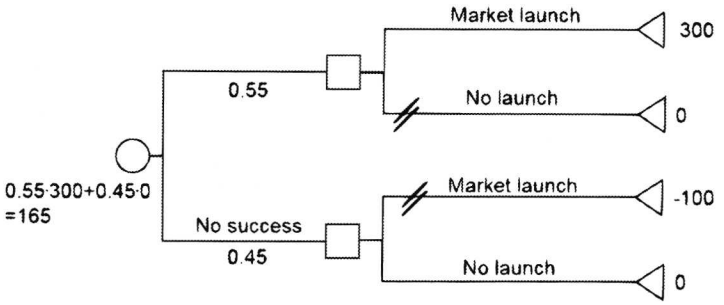


Figure 1.39: Calculating the value of the perfect information

1.5.5 Sensitivity Analysis

The expected value without test market is 120 and with test market is 114. Hence, a risk neutral decision maker would not opt for the test market. However, we can ask how high the success probability of a positive test market should be such that the the test market would be chosen. For this let us denote the success probability with p (so far p has been 0.85). If the decision maker is indifferent between the test market and introducing the product without test market the following has to hold:

$$
\begin{aligned}
& (p \cdot 300 + (1-p) \cdot (-100)) \cdot 0.6 - 30 = 0.55 \cdot 300 - 0.45 \cdot 100 \\
\Leftrightarrow\ & (p \cdot 300 + (1-p) \cdot (-100)) \cdot 0.6 - 30 = 120 \\
\Leftrightarrow\ & (p \cdot 300 + (1-p) \cdot (-100)) \cdot 0.6 = 150 \\
\Leftrightarrow\ & p \cdot 300 + (1-p) \cdot (-100) = 250 \\
\Leftrightarrow\ & p \cdot 400 = 350 \\
\Leftrightarrow\ & p = \frac{350}{400} = 0.875
\end{aligned}
$$

Hence, if the test market could bring forth a success probability of 0.875 in case of a positive result, the decision maker can also choose the test market without sacrificing expected value. For any success probability larger than 0.875 the test market should be chosen in order to maximize the expected value.

1.5.6 Decision Tree and Expected Utility

Solving the decision tree with the expected utility approach

1. All values (such as the costs of the test market) which are associated with edges emanating from decision or chances nodes are shifted to the outcome nodes. E.g., the costs of the test markets are added to the six outcome nodes (from top to bottom) emanating from the test market decision.
2. Determine the utility function $u(x)$ of the decision maker based on the best outcome e^+ and the worst outcome e^-.
3. Transform all outcomes e into utilities $u(e)$.
4. Calculate the expected utility of the decision tree using the roll back procedure.

Example

Result after step 1 see Figure 1.40.

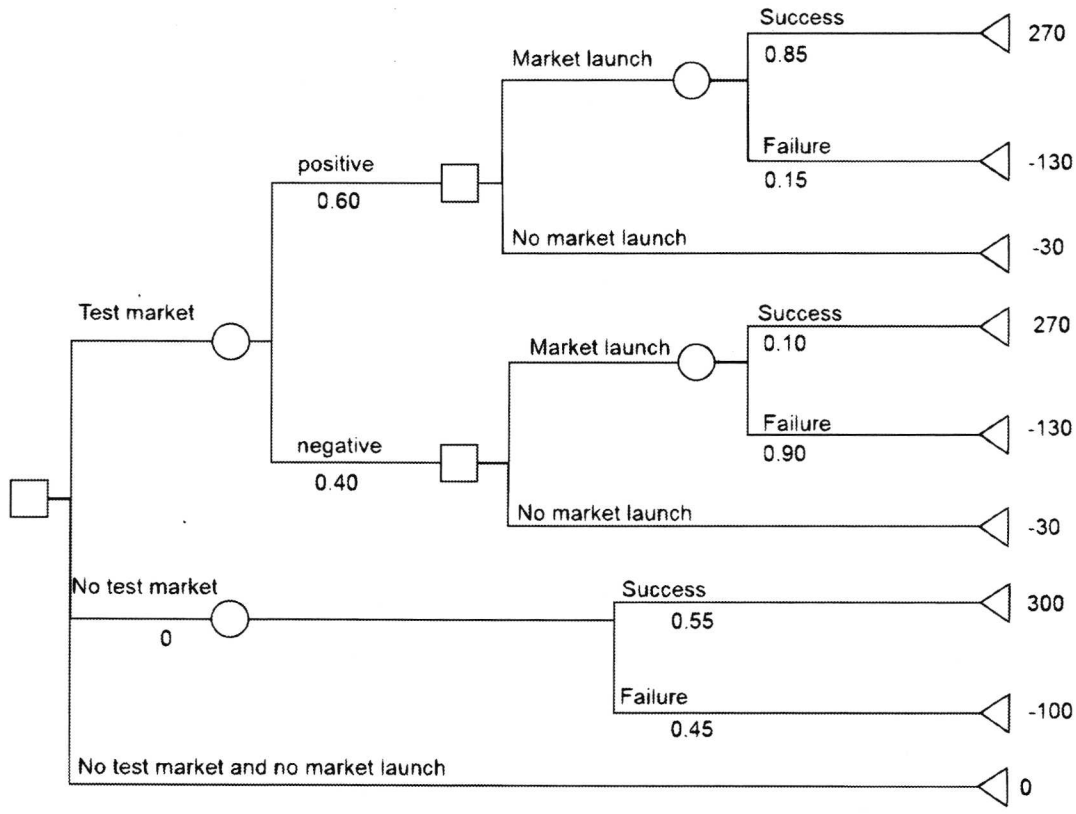


Figure 1.40: Decision tree of the product introduction with the test costs shifted to the outcome nodes

Result after step 2:

Based on $e^+ = 300$ and $e^- = -130$ a feasible utility function of a risk averse decision maker is

$$u(x) = \left(\frac{x+130}{430}\right)^{0.5}$$

Result after step 4 see Figure 1.41:

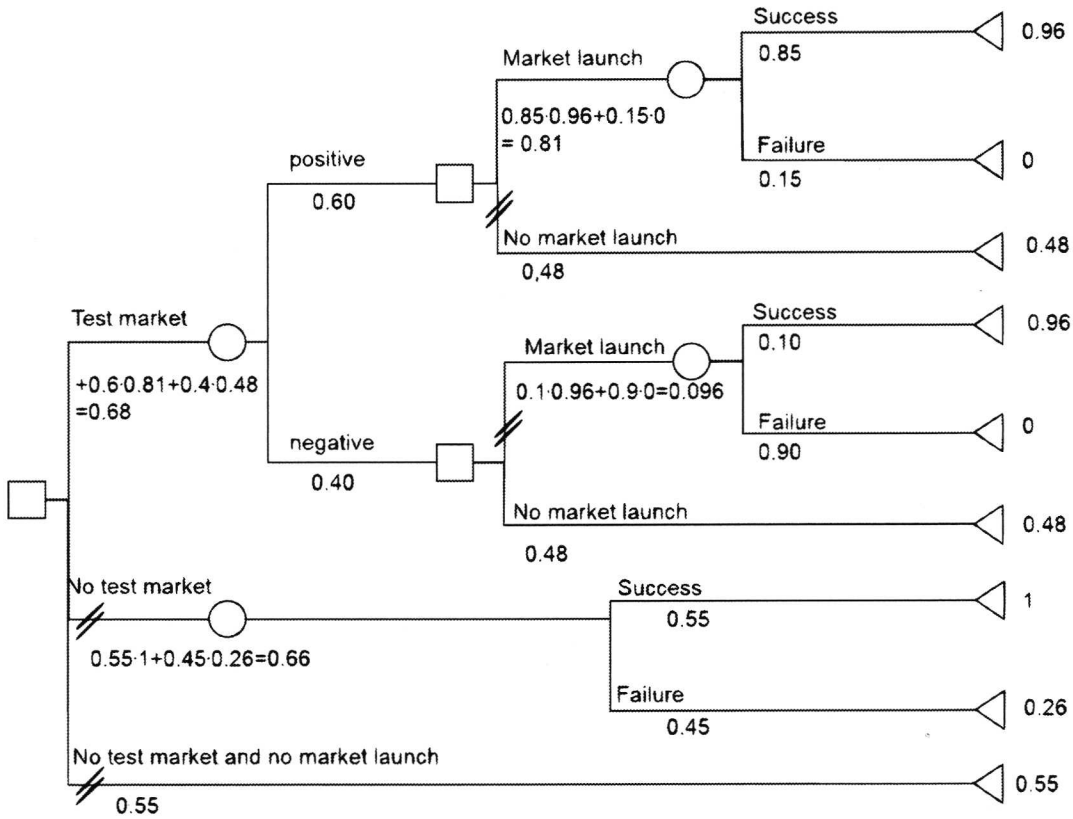


Figure 1.41: Calculation of the expected utility of the decision tree

Observation: The risk averse decision maker with the utility function given above selects the test market while the risk neutral decision maker has not selected the test market.

1.6 Multi–Criteria Decision Making: The Scoring Model

Characterization of the decision situation:

- Deterministic
- Multiple criteria
- Single decsion maker
- Static decision

Scoring Model

The scoring model (German: Nutzwertanalyse) is a simple approach to consider multiple criteria which is prevalent in practice. There are more advanced methods in the literature. E.g., the analytic hierarchy process (AHP).

Idea: The decision matrix is modified by introducing goals $z_1, \ldots, z_n$ with weighty w_j instead of scenarios $s_1, \ldots, s_n$ with probability p_j.

Alternatives:

- $a_1, a_2, \ldots, a_i, \ldots, a_m$.
- The decision maker has to choose one alternative.

Goals (criteria, objectives):

- $z_1, z_2, \ldots, z_j, \ldots, z_n$.
- Goal z_j has a weight w_j with $0 < w_j < 1$.
- $\sum_j w_j = 1$ holds.

Outcomes:

- If the decision maker chooses a_i, he will obtain outcome $e_{i,j}$ with respect to goal z_j.

Decision matrix:

	w_1	..	w_j	..	w_n
	z_1	..	z_j	..	z_n
a_1	$e_{1,1}$	..	$e_{1,j}$	..	$e_{1,n}$
..					
a_i	$e_{i,1}$	..	$e_{i,j}$	..	$e_{i,n}$
..					
a_m	$e_{m,1}$	..	$e_{m,j}$	..	$e_{m,n}$

Figure 1.42: Decision matrix for a multi–criteria decision

Which alternative should the decision maker select if he wants to maximize the outcome for the different criteria taking into account the criteria weights?

1.6.1 Applying the Scoring Model

1. Determine the goals (as a rule of thumb the number of goals should not exceed 7).
2. Determine the weight of the goals.
   - Simple approach to obtain weights: Each goal is valued on a scale between 1 and 5 with 5 for a very important goal, 4 for an important goal, 3 for a goal which has average importance, 2 for a less important goal and 1 for a goal which has only marginal importance. Let $g_j \in \{1, ..., 5\}$ be the thus determined value of goal j.
   - Normalized goal weights $0 < w_j < 1$ are now derived by calculating

$$w_j = \frac{g_j}{\sum_j g_j} \tag{1.16}$$

3. Determine the outcomes $e_{i,j}$
4. For each goal j a value function $v_j()$ is determined.
   (a) Determine the best outcome e_j^+ and the worst outcome e_j^-.
   (b) The best outcome e_j^+ has a value of $v_j(e_j^+) = 1$ and the worst outcome e_j^- has a value of $v_j(e_j^-) = 0$.
   (c) Determine the shape (linear, convex, concave) of the value function $v_j()$.
   For the case of a linear value function we obtain the following function:
   - Slope: $\Delta_j = \frac{1}{e_j^+ - e_j^-}$.
   - y–intercept: $v_j^0 = -e_j^- \cdot \Delta_j$.
   - Value function:

$$\begin{aligned} v_j(e_{i,j}) &= v_j^0 + \Delta_j \cdot e_{i,j} \\ &= -\frac{e_j^-}{e_j^+ - e_j^-} + \frac{1}{e_j^+ - e_j^-} \cdot e_{i,j} \end{aligned}$$

5. Calculating the score of each alternative:

$$S(a_i) = \sum_j w_j \cdot v_j(e_{i,j}) \tag{1.17}$$

6. Select the alternative with the highest score.

Example: Scoring Model

A company has to choose one out of three products for introducing it to the market. The goals are z_1 (low) costs and z_2 (high) success probability on the export market.

Product	Costs	Success probability on the export market
1	20	low
2	30	good
3	50	very good

Step 2 of the Scoring Model:

Costs are important, the success on the export market is less important.

$\Rightarrow g_1 = 4,\ g_2 = 2$

$\Rightarrow w_1 = \frac{4}{6} = 0.67,\ w_2 = \frac{2}{6} = 0.33.$

Step 4 of the Scoring Model:

Goal 1:

(a) $e_1^+ = 20,\ e_1^- = 50$

(c) Determination of a linear value function $v_1(e_1)$:

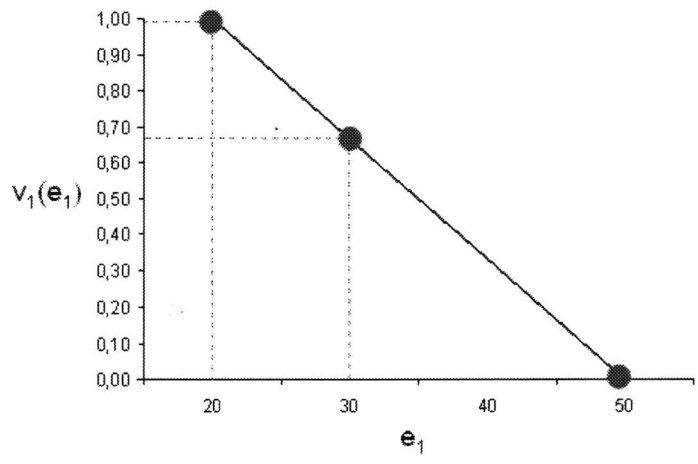


Figure 1.43: Linear value function $v_1(e_1)$ for goal 1

$\Rightarrow v_1(e_1) = 1,67 - \frac{1}{30} \cdot e_1$

$\Rightarrow v_1(20) = 1,\ v_1(30) = 0.67,\ v_1(50) = 0$

Goal 2:

Verbal (qualitative) valuation	Numerical (quantitative) valuation
very good	5
good	4
average	3
modest	2
low	1

(a) $e_2^+ = 5$ (very good), $e_2^- = 1$ (low)

(c) Determination of a linear value function $v_2(e_2)$:

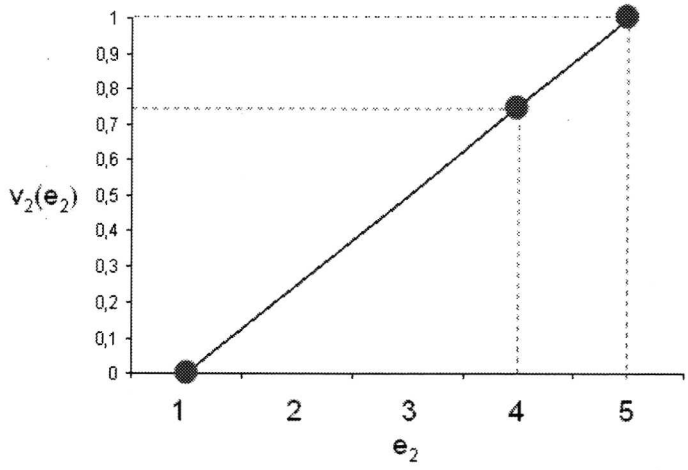


Figure 1.44: Linear value function $v_2(e_2)$ for goal 2

$\Rightarrow v_2(e_2) = -\frac{1}{4} + \frac{1}{4} \cdot e_2$

$\Rightarrow v_2(1) = 0,\ v_2(4) = 0.75,\ v_2(5) = 1$

Step 5:

w_j	0.67	0.33	
z_j	1 (Costs)	2 (Export)	S(Product)
Product 1	1	0	0.67
Product 2	0.67	0.75	0.69
Product 3	0	1	0.33

1.6.2 Sensitivity Analysis

Q: How does the score and thus the decision change if the weight w_1 of the cost goal is changed?

Let us epxress the score of $S(a_i)$ of the alternatives as a function of the weight w_1 of goal z_1:

$$\begin{aligned} S(a_1) &= w_1 \cdot 1 + (1 - w_1) \cdot 0 = w_1 \\ S(a_2) &= w_1 \cdot 0.67 + (1 - w_1) \cdot 0.75 \\ &= w_1 \cdot (0.67 - 0.75) + 0.75 = 0.75 - 0.08 \cdot w_1 \\ S(a_3) &= w_1 \cdot 0 + (1 - w_1) \cdot 1 = 1 - w_1 \end{aligned}$$

Graphical analysis:

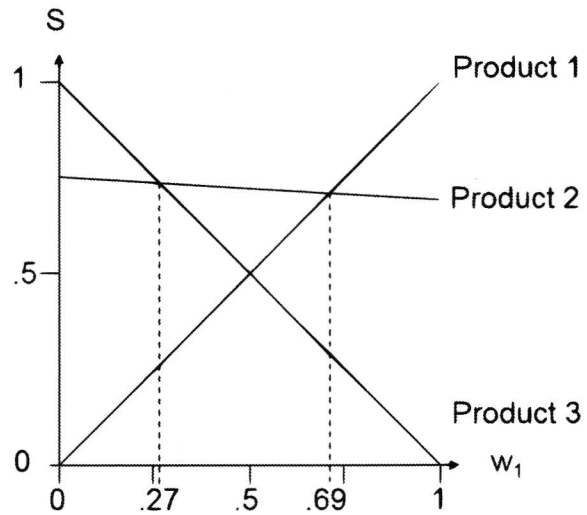


Figure 1.45: Sensitivity analysis of the Scoring model

For $w_1 < 0.27$ the decision maker should chose product 3, for $0.27 < w_1 < 0.69$ the decision maker should choose product 2 and for $w_1 > 0.69$ the decision maker should choose product 1. For $w_1 = 0.27$ the decision maker is indifferent between product 3 and 2 and for $w_1 = 0.69$ the decision maker is indifferent between product 2 and 1.

Chapter 2

Linear Algebra

Literature: Strang [10].

2.1 Matrix and Vector

A matrix $\mathbf{A}$ is defined as

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix}$$

consisting of m rows amd n columns. In total it has $m \cdot n$ elements. The element in row i and column j is denoted with a_{ij}. A matrix with $m = n$ is called a square matrix.

A column vector $\mathbf{b}$ is defined as

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

In what follows, whenever we speak of a vector we mean a column vector. Below we will show that next to column vectors there are also row vectors. Note that a matrix $\mathbf{A}$ consists of n column vectors. A matrix with a single column is a vector.

2.2 Matrix Multiplication

A matrix $\mathbf{A}$ with m rows and n columns and a matrix $\mathbf{B}$ with n rows and o columns can be multiplied, denoted as $\mathbf{A} \cdot \mathbf{B}$. The resulting matrix $\mathbf{C}$ will have m rows and o columns.

Element $c_{i,j}$ of matrix $\mathbf{C}$ is calculated as

$$c_{i,j} = \sum_{l=1}^{n} a_{i,l} \cdot b_{l,j}.$$

Note the following special cases of matrix multiplication: Multiplying a matrix with a vector will give a vector, multiplying a vector with a matrix will give a matrix, multiplying a row vector with a column vector will give a single number, called scalar.

2.3 Transpose of a Matrix

The transpose of matrix $\mathbf{A}$ is denoted as $\mathbf{A}^{\mathrm{T}}$. Element $a^{\mathrm{T}}_{i,j}$ of the transposed matrix is calculated as follows:

$$a^{\mathrm{T}}_{j,i} = a_{i,j}.$$

A matrix $\mathbf{A}$ of size m rows and n columns has a transpose $\mathbf{A}^{\mathrm{T}}$ with n rows and m columns. The transpose of a column vector $\mathbf{b}$ gives row vector $\mathbf{b}^{\mathrm{T}}$.

2.4 Standard Vector and Identity Matrix

The square matrix

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \ddots & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & \vdots \\ 0 & \vdots & \ddots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

is called identity matrix. It holds $\mathbf{I} \cdot \mathbf{A} = \mathbf{A}$ and $\mathbf{A} \cdot \mathbf{I} = \mathbf{A}$.

The vectors $(1, 0, \dots, 0)^{\mathrm{T}}, (0, 1, 0, \dots)^{\mathrm{T}}, \dots, (0, \dots, 0, 1)^{\mathrm{T}}$, which form the identity matrix are called standard vectors.

2.5 Inverse of a Matrix

For matrix $\mathbf{A}$ the inverse is denoted as $\mathbf{A}^{-1}$. It holds $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}$. An invertible matrix is also called non-singular or non-degenerate. The inverse of a matrix can be derived with the Gauß-Jordan algorithm.

2.6 Gauß-Jordan Algorithm for Matrix Inversion

We start by concatenating matrix $\mathbf{A}$ of dimension $m \times n$ and matrix $\mathbf{I}$ of dimension $m \times m$ to matrix $\mathbf{A} \mid \mathbf{I}$ of dimension $m \times (n + m)$

$$\mathbf{A} \mid \mathbf{I} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} & 1 & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & \ddots & a_{2,n} & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & 0 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & 0 \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} & \vdots & \vdots & \dots & 1 \end{pmatrix}$$

We select the first row i for which $a_{i,1} \neq 0$. If $i > 1$, we move row i to become the first row and move the rows $1, \dots, i-1$ to become rows $2, \dots, i$. If $a_{i,1} = 0$ for all i, then the we can proceed with the second column.

We now assume that $a_{1,1} \neq 0$. $a_{1,1}$ is called pivot. We divide each element in row 1 by $a_{1,1}$ such that we obtain $a_{1,1} = 1$. Next, we replace $a_{i,j}$ by $a_{i,j} - a_{i,1} \cdot a_{1,j}$ for all columns $j = 1, \dots, n$ of rows $i = 2, \dots, m$. Due to this, we obtain $a_{1,1} = 1$ and $a_{j,1} = 0$ for $j = 2, \dots, m$. As a result we have the first column vector of the transformed matrix $\mathbf{A} \mid \mathbf{I}$ as

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

We proceed with the second column in the same way as with first column. We select the first row $i > 1$ for which $a_{i,2} \neq 0$ and, if necessary, rearrange the row numbers such that we have $a_{2,2} \neq 0$. $a_{2,2}$ is the second pivot. We divide the pivot row 2 by $a_{2,2}$ to obtain the pivot $a_{2,2} = 1$. Next, we transform the non-pivot rows $i = 1, 3, 4, \dots, m$ by replacing $a_{i,j}$ by $a_{i,j} - a_{i,2} \cdot a_{2,j}$ for all columns $j = 1, \dots, n$. As result we have obtained the second column with $a_{2,2} = 1$ and $a_{i,2} = 0$ for $i = 1, 3, 4 \dots, n$, i.e. the first two columns of the matrix $\mathbf{A} \mid \mathbf{I}$ are

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

We proceed with row three until row m. After finishing the Gauß-Jordan algorithm, we obtain $\mathbf{I} \mid \mathbf{A^{-1}}$ with the inverse

$$\mathbf{A^{-1}} = \begin{pmatrix} a^{-1}_{1,1} & a^{-1}_{1,2} & .. & a^{-1}_{1,n} \\ a^{-1}_{2,1} & a^{-1}_{2,2} & .. & a^{-2}_{2,n} \\ .. & .. & .. & .. \\ .. & .. & .. & .. \\ a^{-1}_{m,1} & a^{-1}_{m,2} & .. & a^{-1}_{m,n} \end{pmatrix}$$

2.7 Solving $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$

We can solve $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ with $\mathbf{A}$ of size $m \times m$ and $\mathbf{x}$ and $\mathbf{b}$ of size $m \times 1$ either by $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b}$ or by directly applying the Gauß-Jordan algorithm. For this, we augment matrix $\mathbf{A}$ by vector $\mathbf{b}$ to obtain matrix $(\mathbf{A} \mid \mathbf{b})$ of size $m \times (m+1)$ and transform it to $(\mathbf{I} \mid \bar{\mathbf{b}})$ which gives the solution $\mathbf{x} = \bar{\mathbf{b}}$.

Chapter 3

Linear Programming

Literature: Bradley et al. [3] Chapters 1 – 4.6

3.1 Linear Program and Graphical Solution

3.1.1 Introductory Example

In the following we are going to treat linear programs (LPs) and how to solve them graphically for the special case when there are only two decision variables (in short: variables). This will lay the foundation for considering LPs with n variables, which will be solved numerically. Let us start with the following example:

A company is producing two different beer glasses: Wheat beer glasses (WG) and Lager glasses (LG). In order to produce one units WG the production facilities have to run 6 hours while for one units LG, 5 hours are required. In one week the facilities can run for 60 hours. Note that a units does not relate to one glass but a larger packing unit such as a pallet. Once produced, the glasses have to be stored before they are shipped to the retailers. The warehouse has a total capacity of 75 units. One unit of WG requires 5 storage units while one unit of LG requires 10 storage units. The contribution margin (the price per unit minus the variable costs) for WG is 5 EURO while the contribution margin for LG is 7 EURO. The maximum demand for WG is 8 units per week. The demand for LG is not limited. How many units of WG and LG shall be produced in order to maximize the total contribution margin?

3.1.2 Modeling and Linear Program

The problem verbally stated above will now be described in terms of a mathematical model. The process of doing so is called "modeling", the result is called a "mathematical program". More specifically, in our case it is a "linear program". A linear program consists of variables

and multiple linear equations. One of the equations is the objective function, the remaining equations are constraints.

We start with the variables. In the remainder of this chapter we will get to know different type of variables. The variables, which are needed in order to describe the decision to be made are named "decision variables". For our problem we have the following two decision variables:

$$\begin{aligned} x_1 &= \text{Number of produced WG units} \\ x_2 &= \text{Number of produced LG units} \end{aligned}$$

Objective function:

$$\text{Maximize the total contribution margin } z = 5x_1 + 7x_2$$

Production capacity constraint:

$$6x_1 + 5x_2 \leq 60$$

Warehouse capacity constraint:

$$5x_1 + 10x_2 \leq 75$$

Market capacity constraint for wheat beer glasses:

$$x_1 \leq 8$$

Definition of the variables (non-negativity constraints):

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Putting it all together, we obtain the following linear program:

$$\begin{aligned} &\text{Maximize } z = 5x_1 + 7x_2 && (3.1) \\ &\text{subject to the constraints} \\ &6x_1 + 5x_2 \leq 60 && (3.2) \\ &5x_1 + 10x_2 \leq 75 && (3.3) \\ &1x_1 \leq 8 && (3.4) \\ &x_1, x_2 \geq 0 && (3.5) \end{aligned}$$

3.1.3 Graphical Representation of the Linear Program

We will now solve the linear program by first representing it graphically and then using basic linear algebra techniques to derive the optimal solution. This technique is restricted to linear programs with not more than two decision variables.

Let us first depict the solution space, which is defined by the constraints. Without considering any constraints, the solution space is defined by the two variables x_1 and x_2. With these two variables, the solution space is 2-dimensional. If for the moment we do not consider the non-negativity constraints given in (3.5), i.e. if we assume $x_1 \in \mathbb{R}$ and $x_2 \in \mathbb{R}$, the solution space is $\mathbf{x} \in \mathbb{R}^2$. The "2" attached to $\mathbb{R}$ states that we are considering the 2-dimensional space with two variables, the $\mathbb{R}$ states that the values of the variables are in the set of real numbers. Hence, feasible $\mathbf{x}$ can be in all four quadrants of the coordinate system and they can be arbitrary far from the origin. E.g., possible solutions are

$$\mathbf{x} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 1 \\ 90 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} -50 \\ -10 \end{pmatrix}, \text{and } \mathbf{x} = \begin{pmatrix} -3 \\ 0.83 \end{pmatrix}.$$

Now, we are adding the constraints. We are starting with the Production capacity constraint (3.2). If we replace "$\leq$" by "$=$", we obtain the equation

$$6x_1 + 5x_2 = 60.$$

This equation is a line in the 2-dimensional space. We can draw this line either with two points or with one point and the slope. Two points: Setting $x_2 = 0$, we obtain $x_1 = 10$ and thus the first point $(10, 0)$. Setting $x_1 = 0$, we obtain $x_2 = 12$ and thus the second point $(0, 12)$. Figure (3.1) gives the line defined by these two points. Point-Slope: We undertake the following transformation in order to obtain the point-slope representation:

$$\begin{aligned} 6x_1 + 5x_2 &= 60 && | -6x_1 \\ 5x_2 &= 60 - 6x_1 && | : 5 \\ x_2 &= 12 - 1.2x_1 \end{aligned}$$

Definition 6 *In n-dimensional space with* $\mathbf{a} \in \mathbb{R}^n$ *and* $\mathbf{x} \in \mathbb{R}^n$, $\sum_{j=1}^{n} a_j x_j = b$ *is a* **hyperplane**. *Hence, in 2-dimensional space, a hyperplane is a line, in 3-dimensional space, a hyperplane is a plane.*

Now, with respect to the Production capacity constraint (3.2), all $\mathbf{x}$ on the line and below the line are feasible.

Definition 7 *In n-dimensional space we call the feasible space, which is defined by* $\sum_{j=1}^{n} a_j x_j \leq b$ *or* $\sum_{j=1}^{n} a_j x_j \geq b$ *a* **halfspace**.

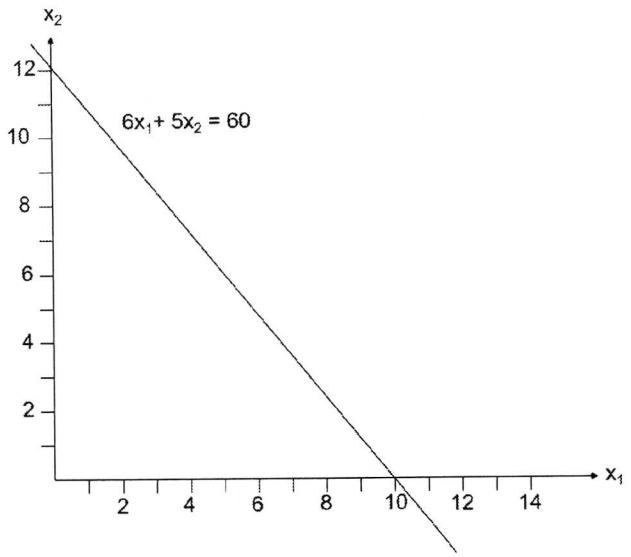


Figure 3.1: Line $6x_1 + 5x_2 = 60$

Figure 3.2 shows the halfspace of the production capacity constraint. The term halfspace clearly describes the fact that we cut the entire 2-dimensional space into two parts, the lower part of it being the considered halfspace.

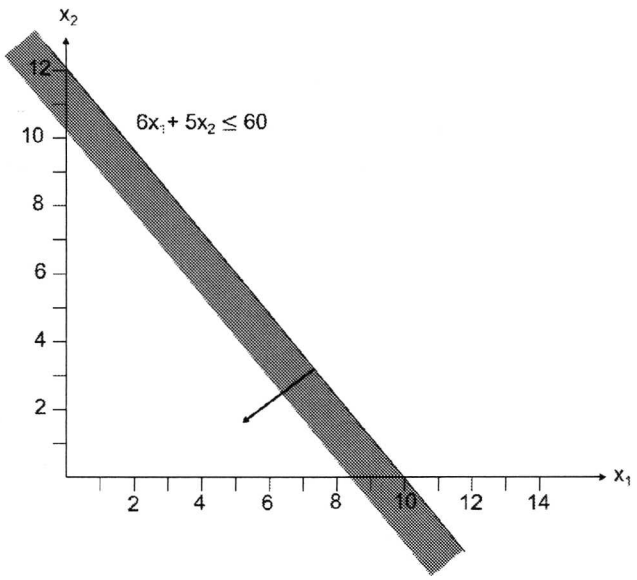


Figure 3.2: Halfspace defined by the production capacity constraint

Each of the five constraints (production capacity, warehouse capacity, market capacity, non-negativity of the two decision variables) defines an associated halfspace. The intersection of the five halfspaces is given in Figure 3.3.

Definition 8 *A* **polyhedron** P *is the intersection of a finite set of halfspaces. It can be described as* $P = \{\mathbf{x} \in \mathbb{R} \mid \mathbf{Ax} \leq \mathbf{b}\}$.

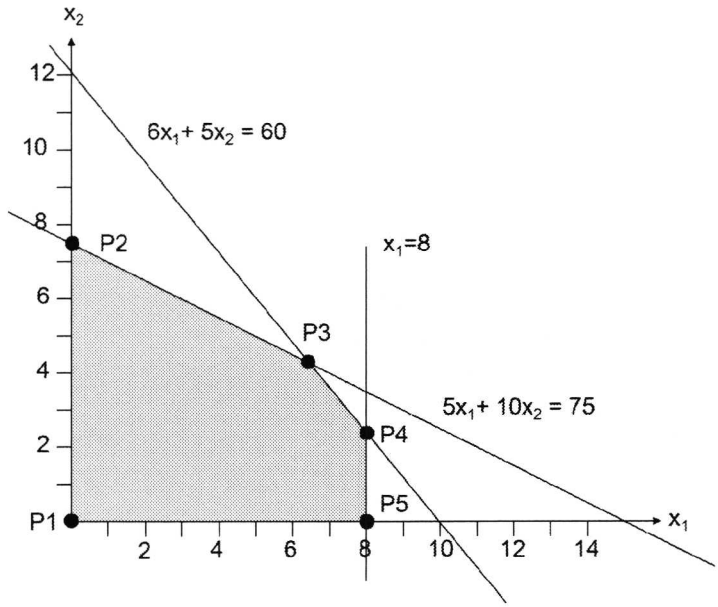


Figure 3.3: Polyhedron representing the feasible solution space of the beer glass problem

Definition 9 *Let* P *be a polyhedron in* n*-dimensional space, written as* $P \subset \mathbb{R}^n$. *A vector* $\mathbf{x} \in P$ *is an* **extreme point** *of* P *if there are not two vectors* $\mathbf{x}^1, \mathbf{x}^2 \in P$, *both different from* $\mathbf{x}$, *and a scalar* $0 < \lambda < 1$, *such that* $\mathbf{x} = \lambda\mathbf{x}^1 + (1 - \lambda)\mathbf{x}^2$.

In the polyhedron of Figure 3.3, the points P1 - P5 are extreme points. The point represented by the vector $\mathbf{x}^{\mathrm{T}} = (4, 0)$ is not an extreme point since we have $(\mathbf{x}^1)^{\mathrm{T}} = (0, 0) \in P$, $(\mathbf{x}^2)^{\mathrm{T}} = (0, 8) \in P$, and $\lambda = 0.5$. Each extreme point in Figure 3.3 is defined by the intersection of two constraints. Some of the constraints have the form $x_j = 0$. I.e., for points P2 and P5, one of the constraints is $x_1 = 0$ and $x_2 = 0$, respectively. For point P1, the two constraints are $x_1 = 0$ and $x_2 = 0$.

In order to find the solution with the largest objective function value we have to consider the Objective function (3.1), i.e.,

$$z = 5x_1 + 7x_2$$

Note that z is a variable which depends on the variables x_1 and x_2. Hence, for each feasible $\mathbf{x}$, we obtain an associated objective function z. Let us draw the objective function line with the value $z = 5 \cdot 7 = 35$. This objective function value is obtained by the following two $\mathbf{x}^{\mathrm{T}}$: $(7, 0)$ and $(0, 5)$. An objective function value of $z = 70$ is obtained by the two $\mathbf{x}^{\mathrm{T}}$: $(14, 0)$ and $(0, 10)$. Figure 3.4 shows the polyhedron and the two lines with z-values 35 and

70 as well as a line with objective function value $z = 62.14$. We observe that as further the distance of the objective function from the origin is, the larger is the objective function value z. The objective function, which is the farthest away from the origin and for which at least one point lies in the polyhedron, goes through the extreme point P3. It has the objective function value $z = 62.14$.

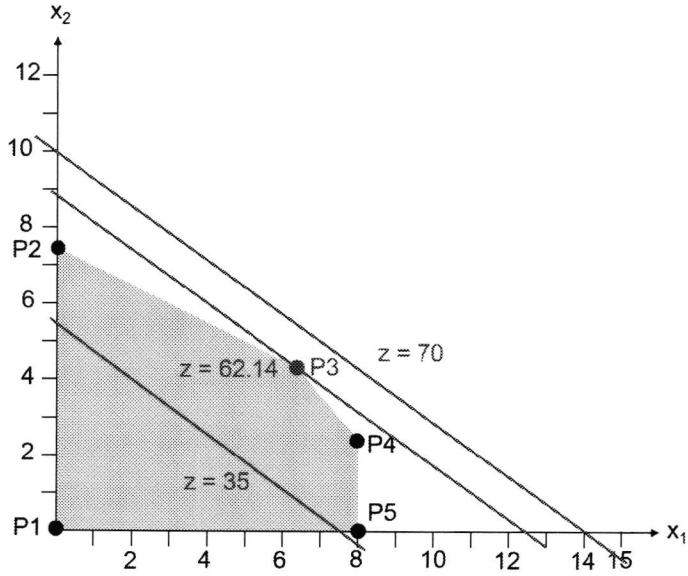


Figure 3.4: Objective function

Property 1 *An optimal solution of a linear program is always an extreme point of the polyhedron.*

3.1.4 Solving the Linear Program Graphically

The extreme point P3 is the intersection of the two lines representing the border of the halfspaces of the production constraint and the warehouse constraint. We determine P3 with the Gauß-Jordan algorithm. For this, we write the two constraints

$$\begin{aligned} 6x_1 + 5x_2 &= 60 \quad \text{(Production constraint)} \\ 5x_1 + 10x_2 &= 75 \quad \text{(Inventory constraint)} \end{aligned}$$

in matrix notation as

$$\begin{pmatrix} 6 & 5 \\ 5 & 10 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 60 \\ 75 \end{pmatrix}$$

We derive the solution with the Gauß-Jordan algorithm applied to the augmented matrix $(\mathbf{A} \mid \mathbf{b})$

$$\left(\begin{array}{cc|c} 6 & 5 & 60 \\ 5 & 10 & 75 \end{array} \right)$$

Since $a_{1,1} \neq 0$, the pivot is $a_{1,1}$. We transform row 1 by dividing it by $a_{1,1} = 6$, which gives

$$\left(\begin{array}{cc|c} 1 & 0.83 & 10 \\ 5 & 10 & 75 \end{array}\right)$$

Next, we subtract $a_{2,1} = 5$ times row 1 from row 2 to in order to obtain $a_{2,1} = 0$.

$$\left(\begin{array}{cc|c} 1 & 0.83 & 10 \\ 0 & 5.83 & 25 \end{array}\right)$$

In order to obtain $a_{2,2} = 1$, we divide row 2 by $a_{2,2} = 5.83$, which gives

$$\left(\begin{array}{cc|c} 1 & 0.83 & 10 \\ 0 & 1 & 4.29 \end{array}\right)$$

Finally, to obtain $a_{1,2} = 0$ and thus $\mathbf{A} = \mathbf{I}$, we subtract $a_{1,2} = 0.83$ times row 2 from row 1, which gives

$$\left(\begin{array}{cc|c} 1 & 0 & 6.43 \\ 0 & 1 & 4.29 \end{array}\right).$$

The solution is thus $x_1 = 6.42$ and $x_2 = 4.29$. We calculate the objective function value z associated with $\mathbf{x}$ as

$$z = 5x_1 + 7x_2 = 5 \cdot 6.42 + 7 \cdot 4.29 = 62.14.$$

Hence, the solution to the beer glass production planning problem is to produce $x_1 = 6.42$ units of wheat beer glasses and $x_2 = 4.28$ units of lager beer glasses. This gives a total contribution margin of 62.14. Since P3 is at the intersection of the production and the warehouse constraint, there will be no left-over production or inventory capacity. However, as can bee seen from Figure 3.3, the market capacity is not fully exploited. The not used market capacity is $8 - x_1 = 8 - 6.42 = 1.58$.

3.1.5 Enumerating all Solutions

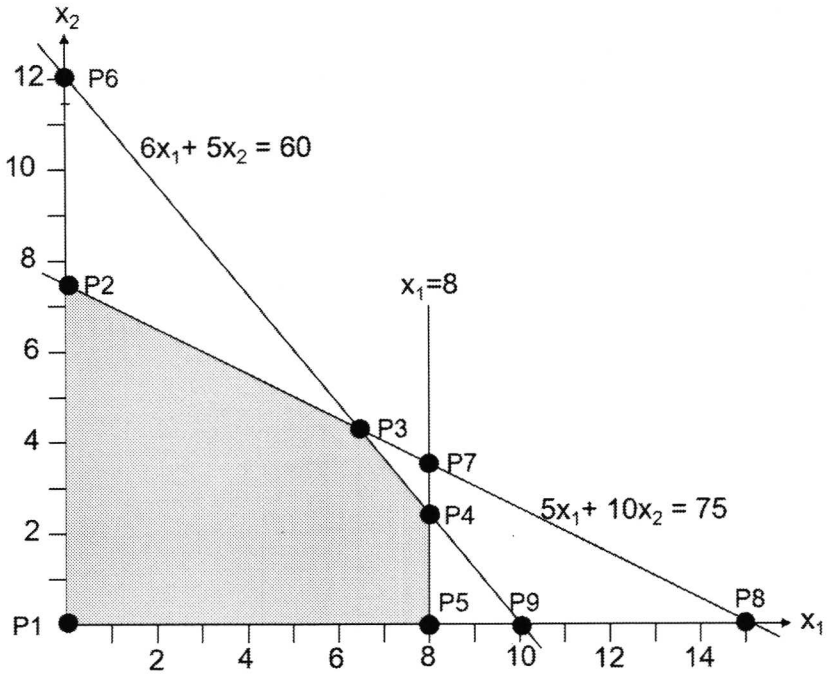


Figure 3.5: All intersections of two lines

Figure 3.5 gives the 5 points P1 - P5, which define the polyhedron of the beer glass production problem as well as the points P6 -.P9, which are - as the points P1 - P5 - intersections of two lines. If we are in the n-dimensional space we cannot draw the polyhedron any more and hence we cannot find the extreme points of the polyhedron by visual inspection. Instead, we can enumerate all possible intersections of 2 of the 5 lines given by the 3 constraints and the two non-negativity constraints of the the variables x_1 and x_2. The formula for the number of possibilities to select k elements out of set of $q \geq k$ elements is

$$\binom{q}{k} := \frac{q!}{k!(q-k)!}.$$

For $k = 2$ and $q = 5$ we obtain 10 possible combinations. 9 of them are depicted by the points P1 - P9 in Figure 3.5. Table 3.1 gives all 10 points and the associated two lines. From there we can see that point P10 does not exist, because the lines $x_1 = 0$ and $x_1 = 8$ are parallel and thus have no intersection. Each point could be checked for feasibility with respect to the constraints of the linear program and out of the feasible points, the one with the maximum objective function value could be selected in order to obtain the optimal solution. However, the effort for enumerating all points increases considerable with the size of the linear program measured in the number of decision variables and the number of constraints. E.g., for a linear program with 10 decision variables and 10 constraints, we

would have $\binom{20}{10} = 184,756$ points to consider. Real-world problem have more than tens of thousands variables and constraints. Hence, enumeration of points is not a valid approach.

Point	x_1	x_2	z	Intersection of lines		Feasible	Note
P1	0	0	0	$x_1 = 0$	$x_2 = 0$	Yes	
P1	0	7.5	52.5	$x_1 = 0$	$5x_1 + 10x_2 = 75$	Yes	
P3	6.42	4.29	62.14	$5x_1 + 10x_2 = 75$	$6x_1 + 5x_2 = 60$	Yes	Optimal
P4	8	2.4	56.8	$6x_1 + 5x_2 = 60$	$x_1 = 8$	Yes	
P5	8	0	40	$x_1 = 8$	$x_2 = 0$	Yes	
P6	0	12	84	$x_1 = 0$	$6x_1 + 5x_2 = 60$	No	
P7	8	2.4	56.8	$x_1 = 8$	$5x_1 + 10x_2 = 75$	No	
P8	15	0	75	$x_2 = 0$	$5x_1 + 10x_2 = 75$	No	
P9	10	0	50	$x_2 = 0$	$6x_1 + 5x_2 = 60$	No	
P10	-	0	-	$x_1 = 0$	$x_1 = 8$	No	Not existing

Table 3.1: Listing of points

3.2 Simplex Algorithm

Literature: Jensen and Bard [8] pp. 63.

The Simplex algorithm was invented by George B. Dantzig in 1947. The main characteristics of the Simplex are that it does only consider extreme points of the polyhedron and it does not enumerate these but moves from one extreme point to one neighbor (so-called adjacent) extreme point if the objective function value can be improved. It stops with an optimal solution, if for the current extreme point there is no adjacent extreme point with a larger objective function value.

3.2.1 Standard Form

In order to apply the Simplex algorithm we have to transform the linear program in a specific form, which we call "standard form". In the literature, this form is also called "canonical form".

Definition 10 *An LP is in* **standard form** *if:*

1. *All variables except z are constrained to be non-negative, i.e., $x_j \geq 0$.*
2. *All constraints except the non-negativity constraints of the variables are stated as equalities. The term left of the equal sign is called "left-hand side", the term right of the equal sign is called "right-hand side".*

3. *Each right-hand-side coefficient is non-negative.*

4. *For each constraint except the non-negativity constrains of the variables there is exactly one variable with a coefficient of 1 in this constraint and a coefficient of 0 in all other constraints and the objective function.*

Definition 11 *Given a linear program in standard form with m equal-constraints and $n > m$ variables, the m variables, which relate to condition 4 of the standard form are called* **basic variables.** *The $n - m$ remaining variables are called* **non-basic variables**. *The set of basic variables is called* **basis.**

For the beer glass problem we can derive the standard form by introducing for each of the Constraints (3.2) - (3.4) a so-called "slack variable", which takes the difference between the left-hand-side and the right-hand-side. We call these variables x_3, x_4, and x_5, respectively. This gives us

$$\text{Max } z = 5x_1 + 7x_2 + 0x_3 + 0x_4 + 0x_5 \tag{3.6}$$

s.t.

$$6x_1 + 5x_2 + 1x_3 + 0x_4 + 0x_5 = 60 \tag{3.7}$$

$$5x_1 + 10x_2 + 0x_3 + 1x_4 + 0x_5 = 75 \tag{3.8}$$

$$1x_1 + 0x_2 + 0x_3 + 0x_4 + 1x_5 = 8 \tag{3.9}$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0 \tag{3.10}$$

Inspecting the linear program (3.6) – (3.10), we see that is in standard form. As for condition 4, for Constraint (3.7) we have variable x_3, for Constraint (3.8) we have variable x_4, and for Constraint (3.9) it is variable x_5. These variables measure the capacity units, which are not used up by the production quantities x_1 and x_2. The variables x_3, x_4, and x_5 are basic variables, accordingly $\{x_3, x_4, x_5\}$ is the basis, and x_1 and x_2 are non-basic variables.

Remark: If we have a linear program with an objective function, which is maximized and all constraints of the type $\leq$, then the slack variables will be the basic variables of the first standard form. However, in the course of the Simplex algorithm this typically changes.

3.2.2 Tableau Notation

Next, we introduce the tableau notation. The tableau is a special way to write down a linear program. For our linear program in standard form we obtain the start tableau given in Table 3.2.

BV	Value	x_1	x_2	x_3	x_4	x_5
x_3	60	6	5	1	0	0
x_4	75	5	10	0	1	0
x_5	8	1	0	0	0	1
$-z$	0	5	7	0	0	0

Table 3.2: Start tableau of the linear program

The tableau has $m+1$ rows which represent the m constraints plus the objective function and $n+2$ columns, which represent the n variables of the standard form plus two additional columns. In contrast to the linear program, the objective function is in the last instead of the first row. The row of a constraint has in columns 3 to $(n+2)$ the coefficients of the left-hand-side. In other words, the **A**-matrix of the LP in standard form is given in rows $1-m$ and columns $3-(n+2)$. In the second column the value of the right-hand side is given. The first row gives the information regarding condition 4 of the standard form, i.e., which variable is associated with the constraint. E.g., for the production constraint in standard form given in the first row, the tableau states in the first column that the base variable is x_3 and in the second column that the right-hand side is 60. The objective function in the last row has a "$-z$" entry because we can rewrite the objective function as follows:

$$\begin{aligned} z &= 5x_1 + 7x_2 + 0x_3 + 0x_4 + 0x_5 && \mid -z \\ 0 &= 5x_1 + 7x_2 + 0x_3 + 0x_4 + 0x_5 - 1z \end{aligned}$$

This way, z is treated as a variable and the objective function is treated as an equation with $n+1$ variables ($x_1 - x_5$ and z).

The constraints of the linear program in standard form can be written in matrix notation as follows:

$$\begin{pmatrix} 6 & 5 & 1 & 0 & 0 \\ 5 & 10 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 60 \\ 75 \\ 8 \end{pmatrix}$$

This set of $m = 3$ equations with $n = 5$ variables can be solved by setting $n-m = 2$ variables to zero and deriving the values of the remaining m variables by Gaußian elimination. If we choose the $n-m$ non-basic variables to be zero, the value of the remaining m basic variables can then be read immediately without applying Gaußian elimination because the resulting **A**-matrix will be an identity matrix. For our example, the variables x_1 and x_2 are non basic and hence we set $x_1 = 0$ and $x_2 = 0$. This gives

$$\begin{pmatrix} 6 & 5 & 1 & 0 & 0 \\ 5 & 10 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 60 \\ 75 \\ 8 \end{pmatrix}.$$

Since $x_1 = 0$ and $x_2 = 0$, the first two columns of the **A**-matrix can be deleted and we obtain

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 60 \\ 75 \\ 8 \end{pmatrix}.$$

From this we can read the solution $x_3 = 60$, $x_4 = 75$, and $x_5 = 8$. Hence, the solution for our linear program is $x_1 = 0$, $x_2 = 0$, $x_3 = 60$, $x_4 = 75$, and $x_5 = 8$ with objective function value $z = 0$. This information can also be read from the Tableau in Table 3.2.

Note: Selecting the $n - m$ non-basic variables, which are set to zero, relates to selecting $n - m$ hyperplanes, which uniquely define the extreme point. In our example, selecting x_1 and x_2 to be non-basic variables means choosing the two lines x_1 and x_2, which define the extreme point (0,0).

3.2.3 Choosing the Pivot Element

Starting with the tableau representing the linear program in standard form, the Simplex checks if there is a adjacent solution, which has a larger objective function value.

Definition 12 *Given a basis with m basic variables and $n - m$ non-basic variables, an* **adjacent basis (or neighbor basis)** *is a basis where exactly one former non-basic variable becomes a basic variable and one former basic variable becomes a non-basic variable.*

The Simplex algorithm selects the non-basic variable with the largest positive coefficient in the objective function. In our example this is x_2 with coefficient $c_2 = 7$. The coefficient c_j in the objective function states the increase of the objective function if the non-basic variable x_j is changed from 0 to 1. Hence, this is a per-unit increase and not an absolute increase. In order to determine the value of the new basic variable x_2, we first look at Figure 3.5. If we increase x_2, there are two lines which limit the increase: Line $5x_1 + 10x_2 = 75$ and line $6x_1 + 5x_2 = 60$. Associated with these two lines are x_2-values of 7.5 and 12. Since the new value of x_2 has to be feasible for the linear program it has to be in the polyhedron and hence the smaller value has to hold. Thus, x_2 is increased from 0 to 7.5. This will lead to the new solution represented by point P2. Point P2 is defined by the intersection of $x_1 = 0$ and $5x_1 + 10x_2 = 75$. The latter equation represents the line of the warehouse constraint. Setting $x_2 = 7.5$ will fully use the warehouse capacity of 75. In the Constraint (3.8) of the linear program in standard form this means that the slack variable of this constraint x_4 has to be 0. This is exactly the basic variable in the start tableau, which become a non-basic variable. Let us now use the tableau in order to determine the basic variables, which becomes non-basic. For each row of the tableau representing a constraint (i.e., rows $i = 1, \ldots, m$) we calculate $\frac{b_i}{a_{i,j}}$, where j is the index of the non-basic variable, which has been selected to become a basic variable. The tableau in Table 3.3 provides the information about this fraction. The constraint i with the smallest $\frac{b_i}{a_{i,j}}$ defines the new value of the the former non-basic variable, which will become a basic variable and it defines the former basic

variable, which will become non-basic. This is the basic variable associated with the selected constraint. For our example, the selected constraint is the warehouse constraint (numbered with (2) in the tableau) and the associated basic variable is x_4. In the tableau we denote the non-basic variable, which will become basic with $\uparrow$ and the basic-variable, which will become non-basic with $\leftarrow$. The element $a_{i,j}$ in the matrix, defined by these two choices is boxed. In our example this is $a_{2,2} = 10$. This element is called pivot element.

	BV	Value	x_1	x_2	x_3	x_4	x_5		
	x_3	60	6	5	1	0	0	(1)	$\frac{60}{5} = 12$
$\leftarrow$	x_4	75	5	$\boxed{10}$	0	1	0	(2)	$\frac{75}{10} = 7.5$
	x_5	8	1	0	0	0	1	(3)	
	$-z$	0	5	7	0	0	0	(4)	
				$\uparrow$					

Table 3.3: Selection of the pivot element in the start tableau of the linear program

We summarize:

Definition 13 *The* **pivot element** (i, j) *of a tableau is determined by first selecting the non-basic variable x_j with largest positive coefficient value c_j and second by selecting the constraint i with $a_{i,j} > 0$, for which $\frac{b_i}{a_{i,j}}$ is smallest.*

3.2.4 Transformation of the Tableau

After we have selected the pivot element, we have to determine the new solution. We first do this in matrix notation

$$\begin{pmatrix} 6 & 5 & 1 & 0 & 0 \\ 5 & 10 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ x_2 \\ x_3 \\ 0 \\ x_5 \end{pmatrix} = \begin{pmatrix} 60 \\ 75 \\ 8 \end{pmatrix}$$

Since the non-basic variables $x_1 = 0$ and $x_4 = 0$, the first and the fourth column of **A**-matrix can be deleted and we obtain

$$\begin{pmatrix} 5 & 1 & 0 \\ 10 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ x_3 \\ x_5 \end{pmatrix} = \begin{pmatrix} 60 \\ 75 \\ 8 \end{pmatrix}$$

Since the **A**-matrix is not an identity matrix, we have to transform **A** into **I** using the Gauß-Jordan algorithm, which gives

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ x_3 \\ x_5 \end{pmatrix} = \begin{pmatrix} 7.5 \\ 22.5 \\ 8 \end{pmatrix}$$

Hence, the new solution is $x_1 = 0$, $x_2 = 7.5$, $x_3 = 22.5$, $x_4 = 0$, and $x_5 = 8$. The corresponding objective function value is $z = 5 \cdot 0 + 7 \cdot 7.5 + 0 \cdot 22.5 + 0 \cdot 0 + 0 \cdot 8 = 52.5$.

We now show how the transformation can be done in the tableau. There, we have to transform the vector of the pivot column into a standard vector with the 1-entry for the pivot element. Hence, row (2) in the tableau of Table 3.3 has to be divided by the pivot element $a_{2,2} = 10$. The resulting tableau is given in Table 3.4. Note that we will number the rows of the transformed matrix as (5) - (8) and hence row (2) in the start tableau is transformed to row (6) in the transformed tableau.

	BV	Value	x_1	x_2	x_3	x_4	x_5	
	x_3	60	6	5	1	0	0	(1)
←	x_4	7.5	0.5	$\boxed{1}$	0	0.1	0	(6) = (2) : 10
	x_5	8	1	0	0	0	1	(3)
	$-z$	0	5	7	0	0	0	(4)
				↑				

Table 3.4: Transformation of the pivot row

Next, for each row i except the pivot row we subtract $a_{i,j}$ times the already transformed pivot row from row i in order to obtain a 0 entry of row i in the pivot column j. The resulting tableau is given in Table 3.5. Note, that the entry of the basic variable in the pivot row has been changed from x_4 to x_2.

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_3	22.5	3.5	0	1	−0.5	0	(5) = (1) − 5 · (6)
x_2	7.5	0.5	$\boxed{1}$	0	0.1	0	(6) = (2) : 10
x_5	8	1	0	0	0	1	(7) = (3) − 0 · (6)
$-z$	−52.5	1.5	0	0	−0.7	0	(8) = (4) − 7 · (6)

Table 3.5: Transformation of the remaining rows

From the tableau given in Table 3.5 we see that an adjacent solution with larger objective function exists because the coefficient of non-basic variable x_1 in the objective function indicates a per unit increase of the objective function of $c_1 = 1.5$. For the new pivot column x_1 we determine the pivot row in the tableau given in Table 3.6.

We transform the second tableau to obtain the tableau with the third solution given in Table 3.7. Since none of the non-basic variables has a positive coefficient in the objective function row, there is no adjacent solution with a larger objective function value than the current solution and hence the Simplex algorithm stops with the optimal solution.

	BV	Value	x_1	x_2	x_3	x_4	x_5		
$\leftarrow$	x_3	22.5	$\boxed{3.5}$	0	1	−0.5	0	(5)	$\frac{22.5}{3.5} = 6.42$
	x_2	7.5	0.5	1	0	0.1	0	(6)	$\frac{7.5}{0.5} = 15$
	x_5	8	1	0	0	0	1	(7)	$\frac{8}{1} = 8$
	$-z$	−52.5	1.5	0	0	−0.7	0	(8)	
			$\uparrow$						

Table 3.6: Selection of the pivot element in the second tableau

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_1	6.42	$\boxed{1}$	0	0.28	−0.14	0	(9) = (5) : 3.5
x_2	4.28	0	1	−0.14	0.17	0	(10) = (6) − 0.5 · (9)
x_5	1.57	0	0	−0.28	0.14	1	(11) = (7) − 1 · (9)
$-z$	−62.14	0	0	−0.42	−0.48	0	(12) = (8) − 1.5 · (9)

Table 3.7: Third and optimal tableau

3.2.5 Formal Description of the Simplex Algorithm

We now provide a formal description of the Simplex algorithm using the notation given in Table 3.8. With b_i, c_j, and $a_{i,j}$ we denote the parameters of the linear program. With $\bar{b}_i$, $\bar{c}_j$, and $\bar{a}_{i,j}$ we denote the parameters in the tableaus. We use the different notation because as we have seen in Section 3.2.4 the values of the elements in the tableau change from one tableau to the next. For the start tableau we set $\bar{b}_i = b_i$, $\bar{c}_j = c_j$, and $\bar{a}_{i,j} = a_{i,j}$.

m	Number of constraints excluding non-negativity constraints
$n \geq m$	Number of variables
$n - m$	Number of non-basic variables
x_j	Variables ($j = 1, \ldots, n$)
$\bar{b}_i$	Right-hand side of constraint i ($i = 1, \ldots, m$)
$\bar{c}_j$	Objective function coefficient of variable x_j ($j = 1, \ldots, n$)
$\bar{a}_{i,j}$	Coefficient of variable x_j ($j = 1, \ldots, n$) in constraint i ($i = 1, \ldots, m$)

Table 3.8: Notation of the Simplex algorithm

Simplex Algorithm

Prerequisite: Tableau with the LP with max objective function in standard form.

1. If $\bar{c}_j \leq 0$ for $j = 1, \ldots, n$, then stop with "optimal solution".
   Continue if $\bar{c}_j > 0$ exists for at least one $j = 1, \ldots, n$.

2. Select the pivot column s:

$$\bar{c}_s = \max_j \{\bar{c}_j \mid \bar{c}_j > 0\}$$

If $\bar{a}_{i,s} \leq 0$ for $i = 1, \ldots, m$, stop; Special case: "Unbounded problem".
Continue only if $\bar{a}_{i,s} > 0$ holds for at least one $i = 1, \ldots, m$.

3. Select the pivot row r:

$$\frac{\bar{b}_r}{\bar{a}_{r,s}} = \min_i \left\{ \frac{\bar{b}_i}{\bar{a}_{i,s}} \mid \bar{a}_{i,s} > 0 \right\}$$

4. Replace basic variable in row r by variable s (change of basic variable) and generate standard form.

5. Go to 1.

Property 2 *The Simplex algorithm stops after performing a finite number of pivot steps. The result is either*

1. *an optimal solution or*
2. *an unbounded problem. In this case the polyhedron is not constrained by a halfspace in the direction of the objective function.*

For the special case that one of the basic variables has a value of 0, it is theoretically possible that the Simplex does not stop. We will discuss this case in Section 3.8.4.

3.3 Deriving the Standard Form

In order to start the Simplex algorithm we need the LP in standard form. So far we have discussed how we can obtain the standard form for constrains of the type "$\leq$". In what follows we show how we can transform all type of constraints into standard form.

3.3.1 Less-Than-Or-Equal Constraints

Consider constraint

$$40x_1 + 10x_2 + 6x_3 \leq 55.$$

We transform this constraint into standard form by introducing the new variable $x_5 \geq 0$.

$$40x_1 + 10x_2 + 6x_3 + x_5 = 55$$

The new variable x_5 takes the value of the difference between the right-hand side 55 and the former left-hand side $40x_1 + 10x_2 + 6x_3$. Since x_5 is defined $x_5 \geq 0$, the left-hand side can never be greater than the right-hand side.

Definition 14 *A variable introduced to transform a "$\leq$"-constraint into a "$=$"-constraint is called a* **slack variable**.

Definition 15 *A variable defined in the linear program before it is converted into standard form is termed a* **decision variable**. *Decision variables represent managerial decisions such as the number of beer glasses of different types we want to produce.*

3.3.2 Greater-Than-Or-Equal Constraints

Consider constraint

$$40x_1 + 10x_2 + 6x_3 \geq 32.$$

We introduce the new variable $x_4 \geq 0$ and obtain

$$40x_1 + 10x_2 + 6x_3 - 1x_4 = 32.$$

Definition 16 *A variable introduced to transform a "$\geq$"-constraint into a "$=$"-constraint is called a* **surplus variable**.

Obviously, x_4 cannot be the basic variable associated to the constraint because it has a coefficient of -1 and hence its value would be $x_4 = -32$, which is not feasible. In order to have a basic variable for the constraint, we add another variable $x_5 \geq 0$, which gives

$$40x_1 + 10x_2 + 6x_3 - 1x_4 + 1x_5 = 32.$$

Definition 17 *A variable introduced in "$=$"-constraint in order to have a basic variable for this constraint is called an* **artificial variable**.

In any feasible solution artificial variables have to be non-basic. We will address this problem in the next section. For the moment we will assign an artificial variable x_j a coefficient $c_j = -M$ in the objective function, where M is a very large number. With this, we indicate that a solution where the artificial variable has a value of zero will always be preferred to a solution where the artificial variable has a value greater than zero.

3.3.3 Constraint with Negative Right-Hand-Side

Consider constraint

$$10x_1 - 5x_2 - 6x_3 \leq -30.$$

We can obtain a positive right-hand side by multiplying this constraint with -1, which gives

$$-10x_1 + 5x_2 + 6x_3 \geq 30.$$

Now we can proceed as in the case of "$\geq$"-constraints.

3.3.4 Equality Constraints

Again, we have to introduce an artificial variable. The following example illustrates this. Given constraint

$$5x_1 + 3x_2 = 20,$$

we introduce the artificial variable $x_3 \geq 0$ and obtain

$$5x_1 + 3x_2 + 1x_3 = 20.$$

3.3.5 Non-Restricted Variables

Let us consider the following constraint

$$\begin{aligned} x - \Delta L &= 50 \\ x &\geq 0 \\ \Delta L &\in \mathbb{R} \end{aligned}$$

where the variable $x \geq 0$ denotes the produced quantity, the variable $\Delta L \in \mathbb{R}$ is the change in inventory, and 50 is the demand. If the produced quantity is less than the demand, then $\Delta L < 0$ and $|\Delta L|$ units have to be taken out of the inventory. If the produced quantity is greater than the demand, then $\Delta L > 0$ units are put into inventory. If the produced quantity equals demand, then the change in inventory is $\Delta L = 0$. E.g., for $x = 30$ we obtain $\Delta L = -20$, for $x = 80$ we obtain $\Delta L = 30$, and for $x = 50$ we have $\Delta L = 0$.

In the standard form we only allow variables ≥ 0 and hence we cannot use ΔL. Instead, we split ΔL in two variables both defined to be ≥ 0: An increase in inventory $\Delta L^+ \geq 0$ and a decrease in inventory $\Delta L^- \geq 0$. Obviously it has to hold $\Delta L = \Delta L^+ - \Delta L^-$. Using the two variables in the equation above we obtain

$$\begin{aligned} x - \Delta L^+ + \Delta L^- &= 50 \\ x, \Delta L^+, \Delta L^- &\geq 0. \end{aligned}$$

Note that for obtaining the standard form for this equation we have to add an artificial variable.

3.3.6 Example

We now apply the techniques discussed above in order to transform an LP into standard form. Let us consider the following LP:

$$\begin{aligned} \text{Min } & -1x_1 + 1x_2 && && (3.11)\\ \text{subject to} & && && \\ & 2x_1 + 1x_2 \leq 10 && && (3.12)\\ & 1x_1 + 2x_2 \geq 3 && && (3.13)\\ & 1x_1 + 1x_2 = -5 && && (3.14)\\ & x_1 \in \mathbb{R} && && (3.15)\\ & x_2 \geq 0 && && (3.16) \end{aligned}$$

Objective function. We defined linear programs to be of the type "max". In order to obtain a max objective function we multiply the min objective function given above with -1 and obtain

$$\text{Max } 1x_1 - 1x_2.$$

Less-Than-Or-Equal Constraints. To transform Constraint (3.12) into standard form we add slack variable $x_3 \geq 0$ and obtain

$$2x_1 + 1x_2 + 1x_3 = 10.$$

Greater-Than-Or-Equal Constraint. Constraint (3.13) is transformed by introducing surplus variable x_4 and artificial variable x_5, which gives

$$1x_1 + 2x_2 - 1x_4 + 1x_5 = 3.$$

Equal Constraints

Constraint (3.14) is first multiplied by -1 in order to obtain a positive right-hand side.

$$-1x_1 - 1x_2 = 5$$

Next, we add the artificial variable x_6:

$$-1x_1 - 1x_2 + 1x_6 = 5$$

Real-Valued Variables. Decision variable $x_1 \in \mathbb{R}$ is not restricted to be greater equal than zero. We substitute it by an increase variable $x_1^+ \geq 0$ and a decrease variable $x_1^- \geq 0$, each defined to be non-negative.

$$1x_1 = 1x_1^+ - 1x_1^-$$

Linear program in standard form. Having performed the steps given above we obtain

$$\begin{aligned}
&\text{Max } 1x_1^+ - 1x_1^- - 1x_2 + 0x_3 + 0x_4 - Mx_5 - Mx_6 \\
&\text{subject to} \\
&\qquad 2x_1^+ - 2x_1^- + 1x_2 + 1x_3 + 0x_4 + 0x_5 + 0x_6 = 10 \\
&\qquad 1x_1^+ - 1x_1^- + 2x_2 + 0x_3 - 1x_4 + 1x_5 + 0x_6 = 3 \\
&\qquad -1x_1^+ + 1x_1^- - 1x_2 + 0x_3 + 0x_4 + 0x_5 + 1x_6 = 5 \\
&\qquad\qquad x_1^+, x_1^-, x_2, x_3, x_4, x_5, x_6 \geq 0
\end{aligned}$$

Note that the artificial variables have an objective function coefficient of $-M$ indicating that in any feasible solution we want these variables to be set to zero. In the following sections we will discuss two methods how we can set artificial variables to zero.

3.4 Deleting Artificial Variables

3.4.1 The Two-Phase Method

If we have a linear program in standard form with artificial variables, the latter will be basic variables and thus have positive values. However, for any feasible solution of the linear program, all artificial variables have to be non-basic and thus zero. Consider the following example where two products have to be produced subject to a capacity limit (first constraint) and a second constraint, which states that the sum of all products produced has to be at least 3.

$$\begin{aligned}
&\text{Max } 2x_1 + 1x_2 \\
&\text{s.t.} \\
&\qquad 3x_1 + 2x_2 \leq 18 \\
&\qquad 1x_1 + 2x_2 \geq 3 \\
&\qquad\quad x_1, x_2 \geq 0
\end{aligned}$$

We transform this linear program into standard form by using slack variable x_3, surplus variable x_4, and artificial variable x_5.

$$\begin{aligned}
&\text{Max } w = 0x_1 + 0x_2 + 0x_3 + 0x_4 - 1x_5 \\
&\text{Max } z = 2x_1 + 1x_2 + 0x_3 + 0x_4 + 0x_5 \\
&\text{s.t.} \\
&\qquad 3x_1 + 2x_2 + 1x_3 + 0x_4 + 0x_5 = 18 \\
&\qquad 1x_1 + 2x_2 + 0x_3 - 1x_4 + 1x_5 = 3 \\
&\qquad\qquad x_i \geq 0 \ (i = 1, \ldots, 5)
\end{aligned}$$

Since the artificial variable has to be zero for the original problem, we have added objective function w, which minimizes the artificial variable (maximizes the artificial variable multiplied by -1). We now apply the Simplex to solve the linear program with objective function w.

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_3	18	3	2	1	0	0	(1)
x_5	3	1	2	0	-1	1	(2)
$-w$	0	0	0	0	0	-1	(3)

This tableau is not in standard form because the basic variable x_5 has a coefficient $c_5 = -1$ in the objective function row. We transform the tableau in standard form by undertaking the following transformation.

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_3	18	3	2	1	0	0	(4) = (1)
x_5	3	1	2	0	-1	1	(5) = (2)
$-w$	3	1	2	0	-1	0	(6) = (3) + (2)

The tableau is now in standard form. However, it is not optimal because the non-basic variables x_1 and x_2 have positive objective function coefficients. Hence, we select the pivot element, see below.

	BV	Value	x_1	x_2	x_3	x_4	x_5		
	x_3	18	3	2	1	0	0	(4)	$\frac{18}{2} = 9$
$\leftarrow$	x_5	3	1	$\boxed{2}$	0	-1	1	(5)	$\frac{3}{2} = 1.5$
	$-w$	3	1	2	0	-1	0	(6)	
				$\uparrow$					

Transforming the tableau we obtain

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_3	15	2	0	1	1	−1	$(7) = (4) - 2 \cdot (8)$
x_2	1.5	0.5	$\boxed{1}$	0	−0.5	0.5	$(8) = (5) : 2$
$-w$	0	0	0	0	0	−1	$(9) = (6) - 2 \cdot (8)$

We have determined the optimal solution for the first phase. The solution does not have an artificial variable in the basis. We now move to the second phase by replacing the objective function of the first phase by the objective function of the second phase and removing the artificial variable. The latter can be deleted because it has a value of zero. For the sake of clarity, we first replace the objective function value, which gives the tableau

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_3	15	2	0	1	1	−1	(7)
x_2	1.5	0.5	1	0	−0.5	0.5	(8)
$-z$	0	2	1	0	0	0	(9).

Now, we delete the artificial variable x_5, which gives the tableau

BV	Value	x_1	x_2	x_3	x_4	
x_3	15	2	0	1	1	(7)
x_2	1.5	0.5	1	0	−0.5	(8)
$-z$	0	2	1	0	0	(9).

Since the tableau is not in standard form (the objective function coefficient of the basic variable x_2 is $\bar{c}_2 = 1$), we transform the tableau to

BV	Value	x_1	x_2	x_3	x_4	
x_3	15	2	0	1	1	$(10) = (7)$
x_2	1.5	0.5	1	0	−0.5	$(11) = (8)$
$-z$	−1.5	1.5	0	0	0.5	$(12) = (9) - (8)$

The tableau for phase 2 is now in standard form. Since it is not optimal, we proceed with the Simplex algorithm and after two more transformations we obtain the optimal solution

BV	Value	x_1	x_2	x_3	x_4	
x_4	3	0	−1.33	0.33	1	(13)
x_1	6	1	0.66	0.33	0	(14)
$-z$	−12	0	−0.33	−0.66	0	(15).

3.4.2 The Big M Method

In the big M method we penalize all artificial variables with objective function coefficient $c = -M$ in the start tableau, where M is a very large number. In order to maximize the objective function value, the Simplex algorithms seeks to drive all artificial variables to be non-basic. Let us consider the example from the two-phase method. The initial tableau of the big M method is

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_3	18	3	2	1	0	0	(1)
x_5	3	1	2	0	-1	1	(2)
$-z$	0	2	1	0	0	$-M$	(3)

The tableau is not in standard form because basic variable x_5 has the coefficient $c_5 = -M$ in the objective function. Hence, we transform to

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_3	18	3	2	1	0	0	(4)
x_5	3	1	2	0	-1	1	(5)
$-z$	$3 \cdot M$	$2 + M$	$1 + 2 \cdot M$	0	$-M$	0	(6) $=$ (3) $+ M \cdot$ (2).

Now the tableau is in standard form but not optimal. Hence, we apply the Simplex algorithm.

	BV	Value	x_1	x_2	x_3	x_4	x_5		
	x_3	18	3	2	1	0	0	(4)	$\frac{18}{2} = 9$
$\leftarrow$	x_5	3	1	$\boxed{2}$	0	-1	1	(5)	$\frac{3}{2} = 1.5$
	$-z$	$3 \cdot M$	$2 + M$	$1 + 2 \cdot M$	0	$-M$	0	(6)	
				$\uparrow$					

BV	Value	x_1	x_2	x_3	x_4	x_5	
x_3	15	2	0	1	1	-1	(7) $=$ (4) $- 2 \cdot$ (8)
x_2	1.5	0.5	$\boxed{1}$	0	-0.5	0.5	(8) $=$ (5) $: 2$
$-z$	-1.5	1.5	0	0	0.5	$-0.5 - M$	(9) $=$ (6) $- (1 + 2M) \cdot$ (8)

In the resulting tableau x_5 is a non-basic variable since $\bar{c}_5 = -0.5 - M < 0$ and hence x_5 can be deleted, which gives the following tableau:

BV	Value	x_1	x_2	x_3	x_4	
x_3	15	2	0	1	1	(7)
x_2	1.5	0.5	1	0	-0.5	(8)
$-z$	-1.5	1.5	0	0	0.5	(9)

This is the same tableau we have reached in the second phase of the two-phase method. As in the two-phase method, we have to apply two further transformations of the tableau before optimality is reached.

3.5 Shadow Prices and Reduced Cost

3.5.1 Shadow Prices

Let us resume with the beer glass production planning problem where the objective function coefficient of decision variable x_2 has been changed to $c_2 = 4.5$. By introducing the slack variables $x_3 \geq 0$ (production capacity), $x_4 \geq 0$ (warehouse capacity) and $x_5 \geq 0$ (market capacity) we obtain the following linear program in standard form:

$$\text{Max } z = 5x_1 + 4.5x_2 + 0x_3 + 0x_4 + 0x_5 \tag{3.17}$$

subject to

$$6x_1 + 5x_2 + 1x_3 + 0x_4 + 0x_5 = 60 \tag{3.18}$$

$$5x_1 + 10x_2 + 0x_3 + 1x_4 + 0x_5 = 75 \tag{3.19}$$

$$x_1 + 0x_2 + 0x_3 + 0x_4 + 1x_5 = 8 \tag{3.20}$$

$$x_j \geq 0 \quad (j = 1, \ldots, 6) \tag{3.21}$$

The associated tableau is:

BV	Value	x_1	x_2	x_3	x_4	x_5
x_3	60	6	5	1	0	0
x_4	75	5	10	0	1	0
x_5	8	1	0	0	0	1
$-z$	0	5	4.5	0	0	0

Applying the Simplex algorithm we obtain the following optimal tableau.

BV	Value	x_1	x_2	x_3	x_4	x_5
x_2	4.28	0	1	−0.14	0.17	0
x_5	1.57	0	0	−0.28	0.14	1
x_1	6.42	1	0	0.28	−0.14	0
$-z$	−51.42	0	0	−0.78	−0.05	0

with the optimal solution $x_2 = 4.28$, $x_5 = 1.57$, $x_1 = 6.42$, and $z = 51.42$.

Let us now look at the coefficients in the objective function row of the optimal tableau. By definition we have for the basic variables $\bar{c}_1 = 0$, $\bar{c}_2 = 0$, and $\bar{c}_5 = 0$. For the non-basic

variables we have $\bar{c}_3 = -0.78$ and $\bar{c}_4 = -0.05$. If we increase the current non-basic variable x_3 from 0 to 1, the change in the objective function is -0.78. If we would set $x_3 = -1$ (which is not feasible since all variables have to be greater than or equal 0) the change in the objective function value would be 0.78. Setting $x_3 = -1$ is the same as increasing the capacity of the production from 60 to 61. Hence, the value of one additional unit of production capacity is 0.78. This value is called the shadow price of the production constraint.

Definition 18 *For a linear program the* **shadow price** *of a constraint is the change in the optimal value of the objective function per unit increase in the right-hand side, all other problem data remaining unchanged. Alternative names for shadow prices are* **dual variable** *or* **opportunity cost**.

According to this definition the shadow price of the warehouse constraint is 0.05 and the shadow price of the market constraint is 0. A shadow price of 0 for the market constraint reflects that if we increase the market capacity by 1 unit to 9 (e.g., by advertising) this would not increase the value of the objective function because already the current market demand of 8 is not fully exploited.

For the beer glass production planning problem in non-standard form we have a max objective function and $\leq$-constraints. For this problem shadow prices are either positive (production and warehouse constraints) or zero (market constraint). The question arises on the sign of the shadow prices in case of other combinations than a max objective function and $\leq$-constraints. In order to obtain an answer, let us consider the following linear program.

$$\text{Max } z = 0x_1 + 1x_2 \tag{3.22}$$

subject to

$$1x_1 + 1x_2 \leq 15 \tag{3.23}$$

$$1x_1 + 0x_2 \geq 10 \tag{3.24}$$

$$x_1, x_2 \geq 0 \tag{3.25}$$

We transfer the linear program in standard by introducing slack variable x_3 for Constraint (3.23) as well as surplus variable x_4 and artificial variable x_5 for Constraint (3.24). Solving the start tableau with the two-phase method or the big M method gives the following optimal tableau, where the artificial variable x_5 has been eliminated.

BV	Value	x_1	x_2	x_3	x_4
x_2	5	0	1	1	1
x_1	10	1	0	0	-1
$-z$	-5	0	0	-1	-1

The optimal solution is straightforward. Due to Constraint (3.24) we have to set $x_1 = 10$. With $x_1 = 10$, Constraint (3.23) limits x_2 to 5. Now, if we increase the right-hand side of

Constraint (3.23) by one unit we can increase x_2 to 6, which increases the objective function value by 1. Hence, the shadow price of Constraint (3.23) is 1. Now, if we increase the right-hand side of Constraint (3.24) with everything else unchanged, i.e., the right-hand side of Constraint (3.23) equal 15, we have to increase x_1 to 11 and consequently decrease x_2 to 4, which leads to a decrease of the objective function value by 1 and thus a shadow price of -1. Hence, for a max problem a $\geq$-constraint has a shadow price which is either zero or negative. The table below gives the sign of the shadow price for each combination of the direction of the objective function and the type of constraint. What we have not discussed is a $=$-constraint. Since a $=$-constraint can be expressed by one $\leq$ and one $\geq$-constraint it follows that the sign of the shadow price can be positive or negative. A more depth discussion on the sign of the shadow price will be given in Section 3.6, in particular see Table 3.11.

Constraint	Objective Function	
	Max	Min
$\leq$	+	−
$\geq$	−	+

Table 3.9: Sign of the shadow price

As stated in Definition 18, the shadow price is also called dual variable or opportunity cost. We will provide a discussion of duality and dual variables in Section 3.6. The term opportunity costs is typically used for production planning problems such as the beer glass production planning problem where we have an objective function maximizing profit or contribution margin subject to limited production capacity.

3.5.2 Reduced Cost

In order to introduce reduced cost let us consider the beer glass production planning problem and let us assume that the company has a third product, a special wheat beer glass (SG) with a fancy design. The table below provides all data for the beer glass production planning problem with three types of beer glasses.

	WG	LG	SG	Capacity
Production	6	5	8	60
Warehouse	5	10	5	75
Market	1	0	0	8
Contribution margin	5	4.5	6	

We employ the following decision variables

$x_1 \geq 0$ units of wheat beer glasses (WG) to be produced,
$x_2 \geq 0$ units of lager glasses (LG) to be produced, and
$x_3 \geq 0$ units of design wheat beer glasses (SG) to be produced.

The associated linear program reads:

$$\begin{aligned}
&\text{Max } z = 5x_1 + 4.5x_2 + 6x_3 && && && (3.26)\\
&\text{subject to} && && &&\\
&6x_1 + 5x_2 + 8x_3 && \leq && 60 && (3.27)\\
&5x_1 + 10x_2 + 5x_3 && \leq && 75 && (3.28)\\
&1x_1 + 0x_2 + 0x_3 && \leq && 8 && (3.29)\\
&x_1, x_2, x_3 && \geq && 0 && (3.30)
\end{aligned}$$

Transforming the linear program in standard form we obtain

$$\begin{aligned}
&\text{Max } z = 5x_1 + 4.5x_2 + 6x_3 + 0x_4 + 0x_5 + 0x_6 && && && && (3.31)\\
&\text{subject to} && && && &&\\
&6x_1 + 5x_2 + 8x_3 + 1x_4 + 0x_5 + 0x_6 && = && 60 && && (3.32)\\
&5x_1 + 10x_2 + 5x_3 + 0x_4 + 1x_5 + 0x_6 && = && 75 && && (3.33)\\
&x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 1x_6 && = && 8 && && (3.34)\\
&x_j && \geq && 0 && (j = 1, \ldots, 6) && (3.35)
\end{aligned}$$

The associated tableau is

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6
x_4	60	6	5	8	1	0	0
x_5	75	5	10	5	0	1	0
x_6	8	1	0	0	0	0	1
$-z$	0	5	4.5	6	0	0	0

Applying the Simplex algorithm yields the optimal tableau

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6
x_2	4.28	0	1	−0.28	−0.14	0.17	0
x_6	1.57	0	0	−1.57	−0.28	0.14	1
x_1	6.42	1	0	1.57	0.28	−0.14	0
$-z$	−51.42	0	0	−0.57	−0.78	−0.05	0

The optimal solution is to produce $x_1 = 6.42$ units wheat beer glasses, $x_2 = 4.28$ units lager glasses and $x_3 = 0$ units special wheat beer glasses. The market capacity for wheat beer glasses is not fully exploited with $x_6 = 1.57$ units of non-catered demand.

The decision variable x_3 has a coefficient of $\bar{c}_3 = -0.57$ in the objective function row. This indicates that the optimal objective function value of $z = 51.42$ would decrease by 0.57 if x_3 would be increased to 1. This value is called the reduced cost.

Definition 19 *For a max problem, the* **reduced cost** *(RC) of a non-basic decision variable is the coefficient in the objective function of the optimal tableau. The reduced cost gives the change of the objective function value if the value of the non-basic variable is increased from 0 to 1.*

Observation: For a max problem the reduced cost of a non-basic variable are negative and the reduced cost of a basic variable are zero. In the special case of multiple optimal solutions, some non-basic variable have zero reduced cost. We will discuss this case in Section 3.8.5

3.5.3 Pricing Out

We now show that reduced cost can be calculated by using the shadow prices. Let us denote with y_i the shadow price of constraint i $(i = 1, \ldots, m)$. The equation for the reduced cost of decision variable j $(j = 1, \ldots, n)$ is then

$$\bar{c}_j = c_j - \sum_{i=1}^{m} a_{i,j} \cdot y_i \tag{3.36}$$

Let us look at the extended beer glass example. We have the shadow prices $y_1 = 0.786$, $y_2 = 0.057$, and $y_3 = 0$ for the production capacity, the warehouse capacity, and the market capacity constraint. Note that the shadow prices have a positive sign since we are having a max problem and $\leq$-constraints. Also note that for accuracy we are using three decimal places instead of two as given in the tableau. Using the $a_{i,j}$ coefficients from the input data we obtain:

$$\begin{aligned}
\bar{c}_1 &= 5 - 6 \cdot 0.786 - 5 \cdot 0.057 - 1 \cdot 0 = 0 \\
\bar{c}_2 &= 4.5 - 5 \cdot 0.786 - 10 \cdot 0.057 - 0 \cdot 0 = 0 \\
\bar{c}_3 &= 6 - 8 \cdot 0.786 - 5 \cdot 0.057 - 0 \cdot 0 = -0.57
\end{aligned}$$

Using the relationship between shadow prices and reduced cost given in Equation (3.36) we found out that it is not profitable to produce design wheat bear glasses. This result has been obtained without formulating and solving the extended wheat bear glass problem.

Definition 20 *Calculating the reduced cost of a decision variable by using the value of the shadow prices is termed* **pricing out**.

3.6 Duality

3.6.1 Motivation

Let us now look at the linear program for a car production planning problem where we have the three products "Trend", "Comfort" and "Sport", represented by decision variables x_1,

x_2, and x_3, respectively. The products are produced subject to two constrained resources: "metal shop" (Constraint 3.38) and "wood shop" (Constraint 3.39). The linear program for this problem reads as follows:

$$\text{Max } z = 6x_1 + 14x_2 + 13x_3 \tag{3.37}$$

subject to

$$0.5x_1 + 2x_2 + 1x_3 \leq 24 \tag{3.38}$$

$$1x_1 + 2x_2 + 4x_3 \leq 60 \tag{3.39}$$

$$x_1, x_2, x_3 \geq 0 \tag{3.40}$$

The optimal tableau for the linear program (3.37) - (3.40) is

BV	Value	x_1	x_2	x_3	x_4	x_5
x_1	36	1	6	0	4	−1
x_3	6	0	−1	1	−1	0.5
$-z$	−294	0	−9	0	−11	−0.5

and the optimal solution is $x_1 = 36$ units Trend, $x_2 = 0$ units Comfort, $x_3 = 6$ units Sport, and $z = 294$ units total contribution margin. The shadow prices for the constraints are $y_1 = 11$ for the metal shop and $y_2 = 0.5$ for the wood shop. The reduced cost for the decision variables are

$$\begin{aligned} \text{Trend} : \bar{c}_1 &= 6 - (0.5 \cdot 11 + 1 \cdot 0.5) = 6 - 12 \cdot 0.5 = 0 \\ \text{Comfort} : \bar{c}_2 &= 14 - (2 \cdot 11 + 2 \cdot 0.5) = 14 - 23 = -9 \\ \text{Sport} : \bar{c}_3 &= 13 - (1 \cdot 11 + 4 \cdot 0.5) = 13 - 13 = 0 \end{aligned}$$

Each decision variable x_j relates to a product j ($j = 1, \ldots, 3$) and the reduced cost $\bar{c}_j$ can be viewed as the difference between the contribution margin c_j and the capacity demand $a_{i,j}$ of the resources i ($i = 1, \ldots, m$) valued by the shadow prices y_i. Products with positive production quantities relate to decision variables in the basis and for which it holds that the contribution margin is greater than or equal to the resource cost when valued with the shadow prices. Products with zero production quantity relate to decision variables not in the basis and for which it holds that the contribution margin is less than the resource costs.

Now, taking the shadow prices we can value the available capacity. We obtain

$$y_1 \cdot 24 + y_2 \cdot 60 = 11 \cdot 24 + 0.5 \cdot 60 = 294$$

We observe that the capacity valued with the shadow prices of the optimal solution is identical to the optimal objective function value.

Next, we want to determine the shadow prices of the resources without solving the production planning problem. Let us define decision variables $y_1 \geq 0$ (price of one capacity unit of the

metal shop) and $y_2 \geq 0$ (price of one capacity unit of the wood shop). We can state the following linear program for determining the shadow prices of the resources.

$$\text{Min } v = 24 \cdot y_1 + 60 \cdot y_2 \tag{3.41}$$

subject to

$$0.5 \cdot y_1 + 1 \cdot y_2 \geq 6 \tag{3.42}$$

$$2 \cdot y_1 + 2 \cdot y_2 \geq 14 \tag{3.43}$$

$$1 \cdot y_1 + 4 \cdot y_2 \geq 13 \tag{3.44}$$

$$y_1, y_2 \geq 0 \tag{3.45}$$

The objective function (3.41) minimizes the cost of the resources valued with the shadow prices y. Constraints (3.42), (3.43) and (3.44) state for products 1 (Trend), 2 (Comfort) and 3 (Sport) that the cost of the used resources when valued with the shadow prices is greater equal than the contribution margin. Finally, Constraint (3.45) defines the shadow prices to have non-negative values.

Solving the linear program to optimality we obtain $y_1 = 11$, $y_2 = 0.5$ and $v = 294$. We observe that we obtain the same objective function value as when solving the production planning problem to optimality. Furthermore, the optimal prices y equal the shadow prices of the capacity constraints of the production planning problem.

Problem (3.41) – (3.45) is the so-called "dual problem" of Problem (3.37) – (3.40). The original problem (3.37) – (3.40) is called "primal problem".

Property 3 *Every primal problem has an associated dual problem. The dual problem of a dual problem is the primal problem.*

Now, let us look at the shadow prices of the dual problem, which we denote with u. These are $u_1 = 36$ for Constraint (3.42), $u_2 = 0$ for Constraint (3.43), and $u_3 = 6$ for Constraint (3.44). We observe that the shadow prices of the dual problem have the same values as the optimal decision variables in the primal problem. We thus can state that by solving the dual problem we obtain the same information as by solving the primal problem.

3.6.2 Rules for Obtaining the Dual Problem

Let us first state the LP of the primal and the dual problem. The primal LP reads:

$$\text{Max } z = \sum_{j=1}^{n} c_j \cdot x_j \tag{3.46}$$

s.t.

$$\sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i \quad (i = 1, 2, \ldots, m) \tag{3.47}$$

$$x_j \geq 0 \quad (j = 1, 2, \ldots, n) \tag{3.48}$$

The dual LP reads:

$$\text{Min } v = \sum_{i=1}^{m} b_i \cdot y_i \tag{3.49}$$

s.t.

$$\sum_{i=1}^{m} a_{i,j} \cdot y_i \geq c_j \quad (j = 1, 2, \ldots, n) \tag{3.50}$$

$$y_i \geq 0 \quad (i = 1, 2, \ldots, m) \tag{3.51}$$

Table 3.10 gives the relationship between the primal and the dual LP.

Dual Variable	Primal Variable					Primal Constraint	Min v
	$x_1 \geq 0$	$x_2 \geq 0$	$x_3 \geq 0$	...	$x_n \geq 0$		
$y_1 \geq 0$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	...	$a_{1,n}$	$\leq$	b_1
$y_2 \geq 0$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	...	$a_{2,n}$	$\leq$	b_2
...	...	...	...	...	...	...	...
$y_m \geq 0$	$a_{m,1}$	$a_{m,2}$	$a_{m,3}$	...	$a_{m,n}$	$\leq$	b_m
Dual Constraint	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$		
Max z	c_1	c_2	c_3	...	c_n		

Table 3.10: Relationship between the primal and the dual LP

The rules for obtaining the dual of a primal LP are given in Table 3.11. The rules can be read in both directions. Going from the left to the right column, we start with a primal maximization problem and obtain a dual minimization problem. Alternatively, going from right to left we start with a primal minimization problem and obtain a dual maximization problem.

Max-problem	Min-problem
i–th constraint $\leq$	i–th variable ≥ 0
i–th constraint $\geq$	i–th variable ≤ 0
i–th constraint $=$	i–th variable $\in \mathbb{R}$
j–th variable ≥ 0	j–th constraint $\geq$
j–th variable ≤ 0	j–th constraint $\leq$
j–th variable $\in \mathbb{R}$	j–th constraint $=$

Table 3.11: Rules for transforming the primal to the dual LP

3.6.3 Example for Obtaining the Dual Problem

Consider the primal problem

$$\begin{aligned} \text{Max } & -1x_1 + 1x_2 - 2x_3 \\ \text{subject to} & \\ & 1x_1 + 1x_2 + 3x_3 = 7 \\ & 2x_1 - 3x_2 + 1x_3 \leq 15 \\ & 1x_1 + 2x_2 - 1x_3 \geq -3 \\ & 2x_1 + 1x_2 + 1x_3 \leq 12 \\ & x_1 \in \mathbb{R} \\ & x_2 \leq 0 \\ & x_3 \geq 0 \end{aligned}$$

Since we have a primal problem with max objective function we go from left to right in Table 3.11 and thus obtain a dual minimization problem.

Decision variables For every constraint in the primal problem we obtain a decision variable in the dual problem. The domain of the decision variable depends on the type of the primal constraint. In case of a primal max problem we have the following domains:

i–th constraint of the primal problem	i-th variable of the dual problem
$\ldots = \ldots$	$y_1 \in \mathbb{R}$
$\ldots \leq \ldots$	$y_2 \geq 0$
$\ldots \geq \ldots$	$y_3 \leq 0$
$\ldots \leq \ldots$	$y_4 \geq 0$

Objective function The coefficients of the dual objective function correspond to the right-hand sides of the primal constraints.

Right-hand side of constraint i of the primal problem	Objective function coefficient of variable i in the dual problem
$\ldots = 7$	$7 \cdot y_1$
$\ldots \leq 15$	$+15 \cdot y_2$
$\ldots \geq -3$	$-3 \cdot y_3$
$\ldots \leq 12$	$+12 \cdot y_4$

Constraints For each decision variable in the primal problem we obtain an associated constraint in the dual problem. The type of constraints in the dual problem is derived from the domain of the decision variable in the primal problem.

j–th decision variable in the primal problem	j-th constraint in the dual problem
$x_1 \in \mathbb{R}$	$\ldots = \ldots$
$x_2 \leq 0$	$\ldots \leq \ldots$
$x_3 \geq 0$	$\ldots \geq \ldots$

The right-hand side values of the dual problem correspond to the coefficients in the objective function of the primal problem.

Objective function coefficient of the j–th variable in the primal problem	Right-hand side value of the j–constraint of the dual problem
$-1 \cdot x_1$	$\ldots = -1$
$+1 \cdot x_2$	$\ldots \leq 1$
$-2 \cdot x_3$	$\ldots \geq -2$

For each constraint of the dual problem we obtain the coefficients of the variables (in the columns) from the coefficients of the associated decision variable in the constraints (rows) of the primal problem.

Coefficients of the j–th variable in the constraints of the primal problem	Coefficients of the variables in the j–th constraint of the dual problem
$1 \cdot x_1 + \ldots = 7$	$1 \cdot y_1$
$2 \cdot x_1 - \ldots \leq 15$	$+2 \cdot y_2$
$1 \cdot x_1 + \ldots \geq -3$	$+1 \cdot y_3$
$2 \cdot x_1 + \ldots \leq 12$	$+2 \cdot y_4 = -1$

The following table summarizes the primal and the dual problem.

Primal Problem	Dual Problem
Max	Min
$-1 \cdot x_1 + 1 \cdot x_2 - 2 \cdot x_3$	$7 \cdot y_1 + 15 \cdot y_2 - 3 \cdot y_3 + 12 \cdot y_4$
s.t.	s.t.
$1 \cdot x_1 + 1 \cdot x_2 + 3 \cdot x_3 = 7$	$y_1 \in \mathbb{R}$
$2 \cdot x_1 - 3 \cdot x_2 + 1 \cdot x_3 \leq 15$	$y_2 \geq 0$
$1 \cdot x_1 + 2 \cdot x_2 - 1 \cdot x_3 \geq -3$	$y_3 \leq 0$
$2 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_3 \leq 12$	$y_4 \geq 0$
$x_1 \in \mathbb{R}$	$1 \cdot y_1 + 2 \cdot y_2 + 1 \cdot y_3 + 2 \cdot y_4 = -1$
$x_2 \leq 0$	$1 \cdot y_1 - 3 \cdot y_2 + 2 \cdot y_3 + 1 \cdot y_4 \leq 1$
$x_3 \geq 0$	$3 \cdot y_1 + 1 \cdot y_2 - 1 \cdot y_3 + 1 \cdot y_4 \geq -2$

3.6.4 Duality Properties

We assume a primal maximization and thus a dual minimization problem.

Property 4 Weak duality property. *Let v be the objective function value of a feasible but not optimal solution of the dual problem and let z be the objective function value of a feasible but not optimal solution of the primal problem, then $v > z$ holds.*

Note: If we have a primal problem which has only constraints of type $\leq$ and thus the dual problem has only variables with domain $\geq$, then we can always find a feasible solution for the dual problem by assigning large values to the dual variables y.

Property 5 Strong duality property. *For an optimal solution of the primal and dual problem we have $v = z$.*

Property 6 Unboundness property. *For an unbounded primal problem we have a non-feasible dual problem.*

Property 7 Complementary slackness. *For the j-th variable in the basis of the optimal solution of the primal problem, we have the associated constraint in the dual with zero slack. That is, the slack variable is not in the basis of the optimal solution of the dual problem.*

In case, the j-th variable is not in the basis of the optimal solution of the primal problem, we have a non-zero slack in the associated constraint of the dual problem. That is, the slack variable is in the basis of the optimal solution of the dual problem.

3.7 Dual Simplex Algorithm

3.7.1 Motivation

Let us consider the following primal linear program:

$$\begin{aligned} &\text{Max } z = -1 \cdot x_1 - 1 \cdot x_2 \\ &\text{subject to} \\ &\qquad -2 \cdot x_1 - 1 \cdot x_2 \leq 4 \\ &\qquad -2 \cdot x_1 + 4 \cdot x_2 \leq -8 \\ &\qquad -1 \cdot x_1 + 3 \cdot x_2 \leq -7 \\ &\qquad x_1, x_2 \geq 0 \end{aligned}$$

Transforming the linear program into its dual we obtain:

$$
\begin{aligned}
&\text{Min } v = 4 \cdot y_1 - 8 \cdot y_2 - 7 \cdot y_3 \\
&\text{subject to} \\
&\qquad -2 \cdot y_1 - 2 \cdot y_2 - 1 \cdot y_3 \;\geq\; -1 \\
&\qquad -1 \cdot y_1 + 4 \cdot y_2 + 3 \cdot y_3 \;\geq\; -1 \\
&\qquad\qquad y_1, y_2, y_3 \;\geq\; 0
\end{aligned}
$$

We write the dual as max problem by multiplying the min objective function with -1. Furthermore, we multiply the two constraint with -1 in order to obtain non-negative right-hand sides.

$$
\begin{aligned}
&\text{Max } -v = -4 \cdot y_1 + 8 \cdot y_2 + 7 \cdot y_3 \\
&\text{subject to} \\
&\qquad 2 \cdot y_1 + 2 \cdot y_2 + 1 \cdot y_3 \;\leq\; 1 \\
&\qquad 1 \cdot y_1 - 4 \cdot y_2 - 3 \cdot y_3 \;\leq\; 1 \\
&\qquad\qquad y_1, y_2, y_3 \;\geq\; 0
\end{aligned}
$$

Below we provide the initial tableaus for the primal problem on the left and the dual problem on the right.

	BV	Value	x_1	x_2	x_3	x_4	x_5
	x_3	4	-2	-1	1		
$\xleftarrow{1}$	x_4	-8	$\boxed{-2}$	4		1	
	x_5	-7	-1	3			1
	$-z$	0	-1	-1			

$\uparrow_2$

Tableau of the primal Program

	BV	Value	y_1	y_2	y_3	y_4	y_5
$\xleftarrow{2}$	y_4	1	2	$\boxed{2}$	1	1	
	y_5	1	1	-4	-3		1
	v	0	-4	8	7		

$\uparrow_1$

Tableau of the dual program

We observe the following relationships between the primal and the dual program, see also the rules for obtaining the dual in Section 3.6.2:

- The entries in the right-hand side of the primal linear program given in the column "Value" multiplied by -1 correspond to coefficients in the objective function of the dual linear program.
- In the case of $\bar{b}_i < 0$ in the tableau of the primal LP we have **primal infeasibility**. Primal infeasibility corresponds to $\bar{c}_j > 0$ in the tableau of the dual LP. Hence, from the point of view of the Simplex algorithm, the tableau of the dual LP is not optimal.
- The coefficients in the objective function row of the tableau of the primal LP correspond to the right-hand side of the tableau of the dual LP multiplied by -1. $\bar{c}_j \leq 0$ in the

tableau of the primal LP (**primal optimality**) thus corresponds to $\bar{b}_i \geq 0$ in the tableau of the dual LP (**dual feasibility**).

- The coefficient matrix of the non-basic variables y_1, y_2, and y_3 in the tableau of the dual LP correspond to the transposed matrix of the coefficient matrix of the non-basic variables x_1 and x_2 in the tableau of the primal LP multiplied by -1.

For our example we observe that the primal LP is infeasible because of $x_4 < 0$ and $x_5 < 0$. The dual LP is due to $\bar{c}_2 > 0$ and $\bar{c}_3 > 0$ not optimal. Since the dual has standard form and is feasible we can use the Simplex algorithm in order to perform one pivot step on the dual LP. The pivot column and row as well as the pivot element are given in the table. The number next to the arrow indicates the sequence in which we select the pivot column and the pivot row. Since the primal and the dual LP correspond to each other we can also select the pivot element in the infeasible primal LP. There, we select first the pivot row and afterwards the pivot column. After having selected the pivot element we establish the new basic variable and transform the tableau into standard form.

	BV	Value	x_1	x_2	x_3	x_4	x_5
	x_3	12		-5	1	-1	
	x_1	4	1	-2		-0.5	
$\overset{1}{\leftarrow}$	x_5	-3		1		$\boxed{-0.5}$	1
	$-z$	4		-3		-0.5	
						$\uparrow_2$	

Primal programm after first pivot step

	BV	Value	y_1	y_2	y_3	y_4	y_5
$\overset{2}{\leftarrow}$	y_2	0.5	1	1	$\boxed{0.5}$	0.5	
	y_5	3	5		-1	2	1
	v	-4	-12		3	-4	
					$\uparrow_1$		

Dual Program after first pivot step

We observe that the primal LP is still infeasible and that the dual LP is still not optimal. Hence, we undertake another pivot step and obtain the tableaus below.

BV	Value	x_1	x_2	x_3	x_4	x_5
x_3	18		-7	1		-2
x_1	7	1	-3			-1
x_4	6		-2		1	-2
$-z$	7		-4			-1

Primal program after second pivot step

BV	Value	y_1	y_2	y_3	y_4	y_5
y_3	1	2	2	1	1	
y_5	4	7	2		3	1
v	-7	-18	-6		-7	

Dual program after second pivot step

Both, primal and dual LP are now feasible and optimal. The optimal solution of the primal LP is $x_3 = 18$, $x_1 = 7$, and $x_4 = 6$ with objective function value $z = -7$. The optimal solution of the dual LP is $y_3 = 1$ and $x_5 = 4$ with objective function value $v = -7$. The strong duality property holds, i.e. $z = v$. Also, the complementary slackness property holds: For the decision variables of the primal LP which are in the basis (x_1) the slack variable in the corresponding constraint of the dual LP is zero ($y_4 = 0$). For the decision variable of the primal LP which is not in the basis ($x_2 = 0$) the slack variable in the corresponding constraint of the dual LP is greater zero ($y_5 = 4$).

3.7.2 Formal Presentation of the Dual Simplex Algorithm

In the example above we have solved the primal problem given in the left part of the tables with the dual Simplex algorithm. As the primal Simplex algorithm, the dual Simplex algorithm requires specific conditions and follows specific rules. Before we give a formal presentation of the dual Simplex, let us motivate the steps undertaken and the choices made above.

Corresponding to the primal Simplex applied to the dual problem (right side of the table) selecting from the columns with $\bar{c}_j > 0$ the one with the maximum $\bar{c}_j > 0$, the dual Simplex applied to the primal problem (left side of the table) selects from the rows with $\bar{b}_i < 0$ the one with minimum $\bar{b}_i$. For selecting the pivot column, we have to consider the following: After the pivot step, the RHS-value of the selected row has to become positive. The new RHS-value is obtained by $\frac{\bar{b}_i}{\bar{a}_{i,j}}$. Since the value of the RHS before the pivot step is $\bar{b}_i < 0$, we can only select pivot columns with $\bar{a}_{i,j} < 0$. The next questions, is which of the columns, for which $\bar{a}_{i,j} < 0$ holds, has to be selected. In order to explain the selection rule for the pivot column, let us consider the following example (see Bradley et al. [3] p. 179):

BV	Value	x_1	x_2	x_3	x_4	
x_3	-1	-1	-1	1		(1)
x_4	-2	-2	-3		1	(2)
$-z$	0	-3	-1			(3)

First, the dual Simplex selects row (2). For the first and the second column $\bar{a}_{2,j} < 0$ holds and hence we need a rule in order to select the pivot column. We recall that after performing the dual pivot step the dual solution has to stay feasible, i.e.; $\bar{c}_j \leq 0$ has to hold. Let us subtract row (2) multiplied with $\alpha > 0$ from the objective function (3). We then obtain

	BV	Value	x_1	x_2	x_3	x_4
	$-z$	0	-3	-1		
$-\alpha \cdot ($		-2	-2	-3		1)

For maintaining dual feasibility all objective function coefficients have to be non-positive and hence it has to hold:

$$\begin{aligned} -3 - (-2 \cdot \alpha) &\leq 0 \\ -1 - (-3 \cdot \alpha) &\leq 0 \\ 0 - 1 \cdot \alpha &\leq 0 \end{aligned}$$

Solving each equation for α, we obtain:

$$\begin{aligned} 2 \cdot \alpha \leq 3 \Rightarrow \alpha &\leq \frac{3}{2} \\ 3 \cdot \alpha \leq 1 \Rightarrow \alpha &\leq \frac{1}{3} \\ \alpha &\geq 0 \end{aligned}$$

Obviously, $\alpha = \frac{1}{3}$ has to hold and we obtain the new objective function:

BV	Value	x_1	x_2	x_3	x_4
$-z$	$\frac{2}{3}$	$-\frac{7}{3}$			$-\frac{1}{3}$

The x_2–column has now the coefficient 0 which makes x_2 the variable entering the basis and the associated column the pivot column. Hence, the dual Simplex has selected amongst the columns with $\bar{a}_{ij} < 0$ the one with minimum $\frac{\bar{c}_j}{\bar{a}_{ij}}$. Note that since we are having dual feasibility, $\bar{c}_j \leq 0$ holds for all columns and hence $\frac{\bar{c}_j}{\bar{a}_{ij}} \geq 0$ holds for all columns.

We now can provide the formal description of the dual Simplex algorithm:

Dual Simplex Algorithm

Prerequisite: The tableau is dual feasible ($\bar{c}_j \leq 0$ for all j) and primal infeasible ($\bar{b}_i < 0$ for at least one i).

1. If $\bar{b}_i \geq 0$ for $i = 1, \ldots, m$ and $\bar{c}_j \leq 0$ for $j = 1, \ldots, n$ then stop; "Optimal solution". Continue if $\bar{b}_i < 0$ for at least one i.

2. Select the pivot row r:

$$\bar{b}_r = \min_i \{\bar{b}_i \mid \bar{b}_i < 0\}$$

   If $\bar{a}_{r,j} \geq 0$ for $j = 1, \ldots, n$, stop; Special case: "No feasible solution". Continue only if $\bar{a}_{r,j} < 0$ for at least one j.

3. Select the pivot column s:

$$\frac{\bar{c}_s}{\bar{a}_{r,s}} = \min_j \left\{ \frac{\bar{c}_j}{\bar{a}_{r,j}} \mid \bar{a}_{r,j} < 0 \right\}$$

4. Replace the basic variable in row r by variable s (change of basic variable) and generate standard form.

5. Go to 1.

Property 8 *The dual Simplex ends*

(a) with a feasible and optimal solution or

(b) by proving that no feasible solution exists.

The dual Simplex is often employed when a new constraint is added to an already existing optimal tableau. For example in the case of an optimally solved production planning problem where a constraint is added for a minimum number of units produced for one product. If the number of produced units of the product in the optimal solution is less than the minimum number, the value of the surplus variable will be negative. Let us consider the optimal tableau of the car production planning problem, which we have introduced at the beginning of this Section.

BV	Value	x_1	x_2	x_3	x_4	x_5
x_1	36	1	6	0	4	−1
x_3	6	0	−1	1	−1	0.5
$-z$	−294	0	−9	0	−11	−0.5

The optimal solution is $x_1 = 36$, $x_3 = 6$, and $z = 294$. We now want to produce at least 5 units of Comfort (x_2). Hence, we obtain the following additional constraint:

$$x_2 \geq 5.$$

Introducing surplus variable x_6, we obtain

$$x_2 - x_6 = 5.$$

Adding this constraint to the optimal tableau, we obtain:

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6
x_1	36	1	6	0	4	−1	0
x_3	6	0	−1	1	−1	0.5	0
x_6	−5	0	−1	0	0	0	1
$-z$	−294	0	−9	0	−11	−0.5	0

The resulting tableau fulfills the conditions for the dual Simplex. We perform one iteration of the dual Simplex by selecting the pivot element.

	BV	Value	x_1	x_2	x_3	x_4	x_5	x_6	
	x_1	36	1	6	0	4	−1	0	(1)
	x_3	6	0	−1	1	−1	0.5	0	(2)
$\overset{1}{\leftarrow}$	x_6	−5	0	$\boxed{-1}$	0	0	0	1	(3)
	$-z$	−294	0	−9	0	−11	−0.5	0	(4)
				$\uparrow_2$					

After one iteration of the dual Simplex we obtain the optimal tableau:

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6	
x_1	6	1	0	0	4	-1	6	$(5) = (1) + 6 \cdot (3)$
x_3	11	0	0	1	-1	$\frac{1}{2}$	-1	$(6) = (2) - (3)$
x_2	5	0	1	0	0	0	-1	$(7) = -1 \cdot (3)$
$-z$	-249	0	0	0	-11	-0.5	-9	$(8) = 4 - 9 \cdot (3)$

3.8 Special Cases

3.8.1 No Feasible Solution

Let us consider the beer glass production planning problem, for which we add the additional constraint $6x_1 + 5x_2 \geq 80$. The constraint forces the use of at least 80 capacity units. The resulting LP reads:

$$\text{Max } z = 5x_1 + 4.5x_2 \tag{3.52}$$

subject to

$$6x_1 + 5x_2 \leq 60 \tag{3.53}$$

$$6x_1 + 5x_2 \geq 80 \tag{3.54}$$

$$5x_1 + 10x_2 \leq 75 \tag{3.55}$$

$$x_1 \leq 8 \tag{3.56}$$

$$x_1, x_2 \geq 0 \tag{3.57}$$

Obviously, the first and the second constraint contradict each other and there is no feasible solution. Figure 3.6 illustrates the empty solution space.

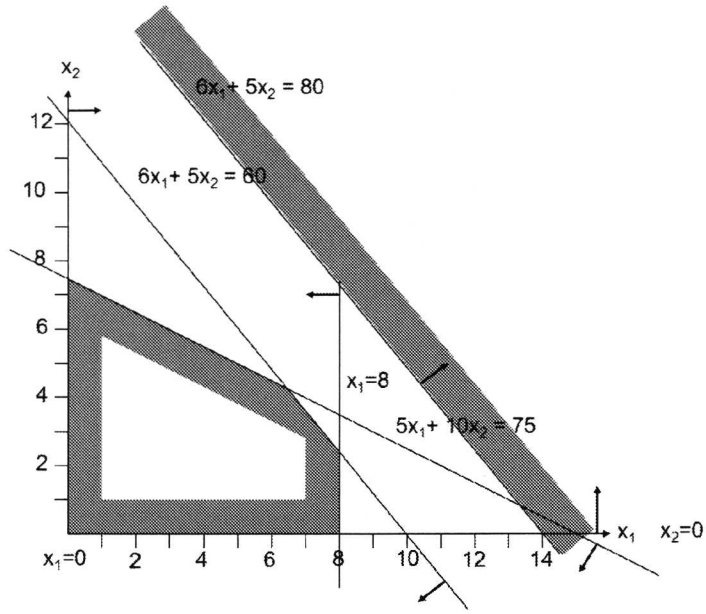


Figure 3.6: Special case: No feasible solution

In order to solve the extended beer glass production planning problem, we have to introduce an artificial variable for Constraint (3.54). We then apply the big M-method in order to obtain a basic feasible solution without artificial variable. However, the optimal solution of the big M-method is a solution where the artificial variable is still in the basis. This indicates that the problem has no feasible solution. When applying the Two-Phase method, we obtain an optimal solution for the first objective function w with artificial variables in the basis.

3.8.2 Unbounded Solution

Let us again use the beer glass production planning problem where we convert all "$\leq$"- to "$\geq$"-constraints.

$$\text{Max } z = 5x_1 + 4.5x_2 \tag{3.58}$$

subject to

$$6x_1 + 5x_2 \geq 60 \tag{3.59}$$

$$5x_1 + 10x_2 \geq 75 \tag{3.60}$$

$$x_1 \geq 8 \tag{3.61}$$

$$x_1, x_2 \geq 0 \tag{3.62}$$

Figure 3.7 shows the solution space which is unbounded because the objective function line can move in northeast direction without being stopped by a constraint.

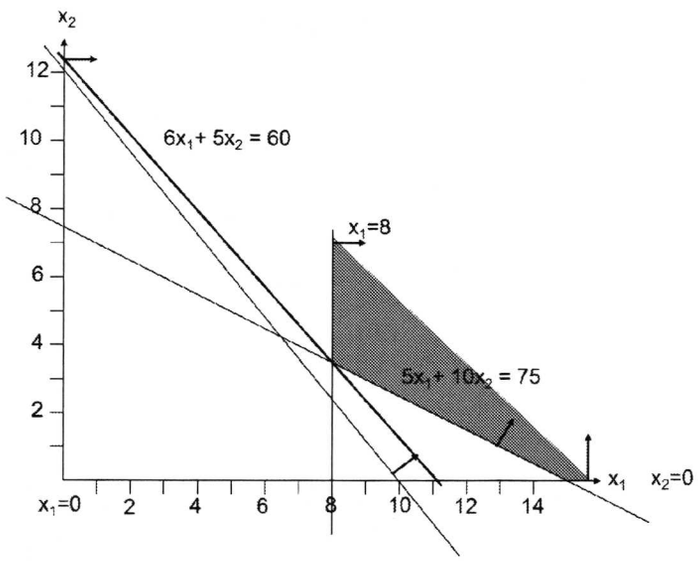


Figure 3.7: Special case: Unbounded solution space

When solving the LP with the primal Simplex algorithm, a pivot column j is selected where $\bar{a}_{ij} \leq 0$ holds for all rows $i = 1, \ldots, m$. Thus, there is no limit in increasing the value of variable x_j (see Step 2 of the primal Simplex algorithm).

3.8.3 Redundant Constraints

We employ the beer glass production planning problem and add the new constraint $10x_1 + 12x_2 \leq 120$, which yields the following LP:

$$\text{Max } z = 5x_1 + 4.5x_2 \tag{3.63}$$

subject to

$$6x_1 + 5x_2 \leq 60 \tag{3.64}$$

$$5x_1 + 10x_2 \leq 75 \tag{3.65}$$

$$x_1 \leq 8 \tag{3.66}$$

$$10x_1 + 12x_2 \leq 120 \tag{3.67}$$

$$x_1, x_2 \geq 0 \tag{3.68}$$

Figure 3.8 provides the solution space. The new constraint is depicted in bold.

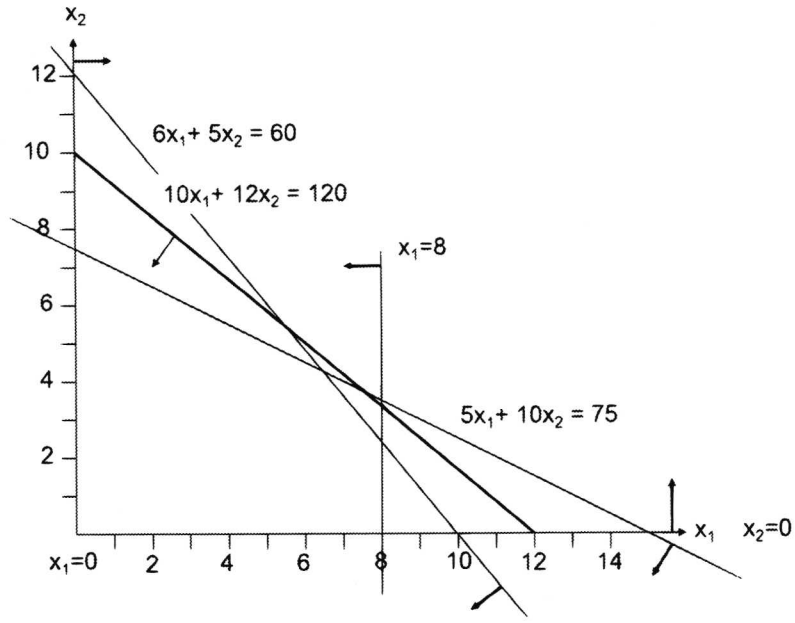


Figure 3.8: Special case: Redundant constraint

Obviously, the new constraint does not alter the solution space. We say that the new constraint is "redundant" as it adds no new information. When solving the problem with the Simplex, at the end of each iteration the slack variable associated with a redundant constraint is larger than zero. Hence, the slack variable is always in the basis.

3.8.4 Primal Degeneracy

Let us consider the following LP:

$$\text{Max } z = 0x_1 + 1x_2 + 1x_3 \tag{3.69}$$

subject to

$$1x_1 + 0x_2 + 1x_3 \leq 4 \tag{3.70}$$

$$x_1, x_2, x_3 \geq 0 \tag{3.71}$$

Figure 3.9 shows the solution space and the unique optimal solution $x_1 = 0$, $x_2 = 0$, and $x_3 = 4$ with objective function value $z = 4$. Note that the solution space is 3-dimensional and that the optimal solution is defined by three planes in this 3-dimensional space (where we describe each plane as an equation): $1x_1 + 0x_2 + 0x_3 = 0$, $0x_1 + 1x_2 + 0x_3 = 0$, and $1x_1 + 0x_2 + 1x_3 = 4$.

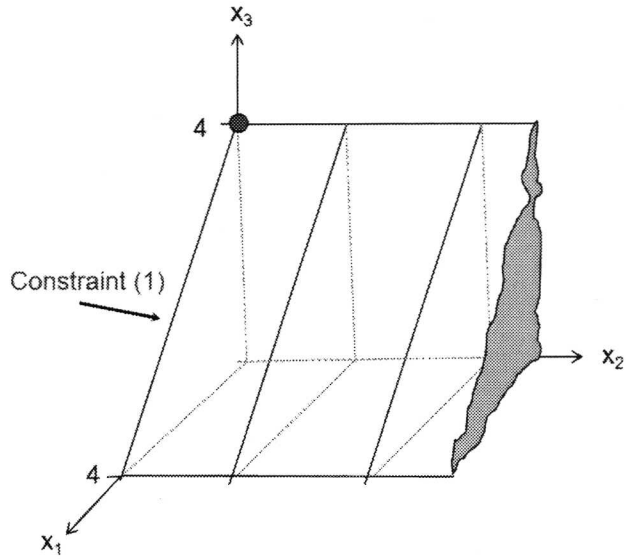


Figure 3.9: Solution space of LP (3.69) - (3.71)

Now, we add constraint $1x_2 + 1x_3 \leq 4$ and obtain the LP

$$\text{Max } z = 0x_1 + 1x_2 + 1x_3 \tag{3.72}$$

subject to

$$1x_1 + 0x_2 + 1x_3 \leq 4 \tag{3.73}$$

$$0x_1 + 1x_2 + 1x_3 \leq 4 \tag{3.74}$$

$$x_1, x_2, x_3 \geq 0 \tag{3.75}$$

Figure 3.10 shows the solution space and the optimal solution. Adding Constraint (3.74) has altered the solution space and hence this constraint is not redundant. The solution space is still 3-dimensional but the optimal solution is now defined by the intersection of the four planes
$1x_1 + 0x_2 + 0x_3 = 0$, $0x_1 + 1x_2 + 0x_3 = 0$, $1x_1 + 0x_2 + 1x_3 = 4$, and $0x_1 + 1x_2 + 1x_3 = 4$.

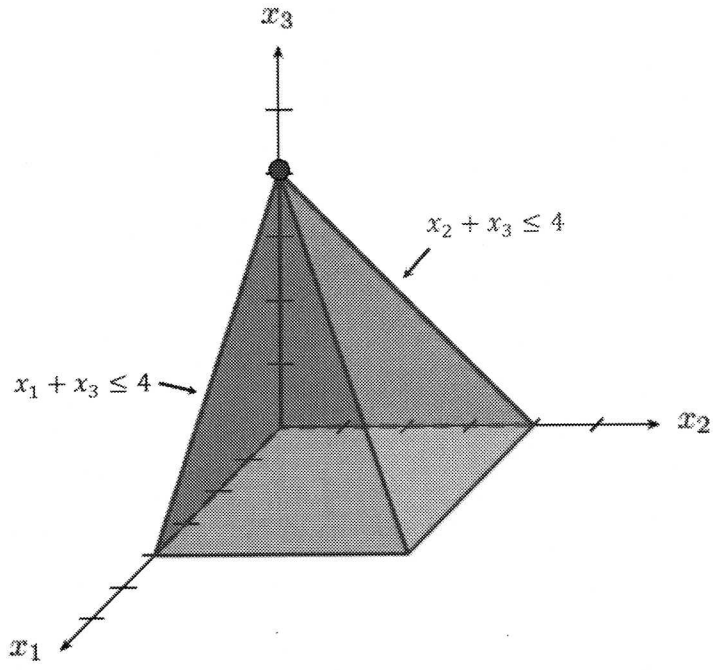


Figure 3.10: Special case: Primal degeneracy

We write down the start tableau of the LP with Constraints (3.73) and (3.74). For the first pivot step we have to choose x_3 as pivot column. For the pivot row we can either choose row 1 (where x_4 will leave the basis) or row 2 (where x_5 will leave the basis).

Option 1: x_4 leaves the basis.

BV	Value	x_1	x_2	x_3	x_4	x_5
x_4	4	1	0	$\boxed{1}$	1	0
x_5	4	0	1	1	0	1
$-z$	0	0	-1	1	0	0

Choosing option 1, we obtain the following optimal tableau with basic variables $x_3 = 4$ and $x_5 = 0$. If a basic variable has a value of 0, we call the solution to be **primal degenerated**.

BV	Value	x_1	x_2	x_3	x_4	x_5
x_3	4	1	0	1	1	0
x_5	0	-1	1	0	-1	1
$-z$	-4	-1	-1	0	-1	0

Option 2: x_5 leaves the basis.

BV	Value	x_1	x_2	x_3	x_4	x_5
x_4	4	1	0	1	1	0
x_5	4	0	1	$\boxed{1}$	0	1
$-z$	0	0	-1	1	0	0

Choosing option 2, we obtain the following optimal tableau wtih basic variables $x_3 = 4$ and $x_4 = 0$. In this case, basic variable $x_4 = 0$ and hence we have again primal degeneracy. Since non-basic variable x_1 has a coefficient of $\bar{c}_1 = 0$ in the optimal objective function, we also have so-called dual degeneracy (see Section 3.8.5).

BV	Value	x_1	x_2	x_3	x_4	x_5
x_4	0	1	-1	0	1	-1
x_3	4	0	1	1	0	1
$-z$	-4	0	-2	0	0	-1

Summary: If an extreme point is overdetermined (it is defined by one more constraint than necessary) we have primal degeneracy. We can detect primal degeneracy by a basic valued with 0. In the case of primal degeneracy, theoretically the Simplex can get stuck in such an extreme point without reaching the optimal solution. This is termed "cycling". However, this case is very rare in practice. Furthermore, cycling can be effectively avoided by specific rules for choosing the pivot element (see Jensen and Bard [8] p. 83) .

3.8.5 Dual Degeneracy

We consider the beer glass production planning problem where we set the objective function coefficient of the wheat bear glasses to 5.4, which yields the following linear program where the y_1, y_2, and y_3 denote the dual variables of Constraint (3.77), (3.78), and (3.79), respectively:

$$\text{Max } z = 5.4x_1 + 4.5x_2 \tag{3.76}$$

subject to

$$6x_1 + 5x_2 \leq 60 \quad (y_1) \tag{3.77}$$

$$5x_1 + 10x_2 \leq 75 \quad (y_2) \tag{3.78}$$

$$x_1 \leq 8 \quad (y_3) \tag{3.79}$$

$$x_1, x_2 \geq 0 \tag{3.80}$$

Note that the Objective function (3.77) is parallel to Constraint (3.77). Solving the linear program, we obtain the optimal tableau

BV	Value	x_1	x_2	x_3	x_4	x_5
x_2	2.4	0	1	0.2	0	−1.2
x_4	11	0	0	−2	0.5	7
x_1	8	1	0	0	0	1
$-z$	−54	0	0	−0.9	0	0

The non-basic variable x_5 has a coefficient of 0 in the objective function row. Hence, we can introduce x_5 to the basis without decreasing the objective function value. Performing one pivot step yields:

BV	Value	x_1	x_2	x_3	x_4	x_5
x_2	4.286	0	1	−0.143	0.171	0
x_5	1.571	0	0	−0.286	0.0143	1
x_1	6.429	1	0	0.286	−0.0143	0
$-z$	−54	0	0	−0.9	0	0

The variables x_1 and x_2 remain in the basis but have different values. The former non-basic variable x_5 has entered the basis and the former basic variable x_4 left the basis. The objective function value does not change. We thus have found a second optimal solution. Furthermore, all points which are on the line connecting the two optimal solutions are also optimal. Hence, we have multiple optimal solutions.

From the two optimal tableaus we can read the value of the dual variables as $y_1 = 90$, $y_2 = 0$, and $y_3 = 0$. In the first optimal tableau, dual variable y_3 has value zero although Constraint (3.79) defines the extreme point. In the second optimal tableau, dual variable y_2 has value zero although Constraint (3.78) defines the extreme point. Hence, in each of the two optimal solutions, one dual variable of a binding constraint is zero. Since a dual variable of a binding constraint should have a positive value, this is called **dual degeneracy**. From the primal point of view, a non-basic variable has an objective function value of zero.

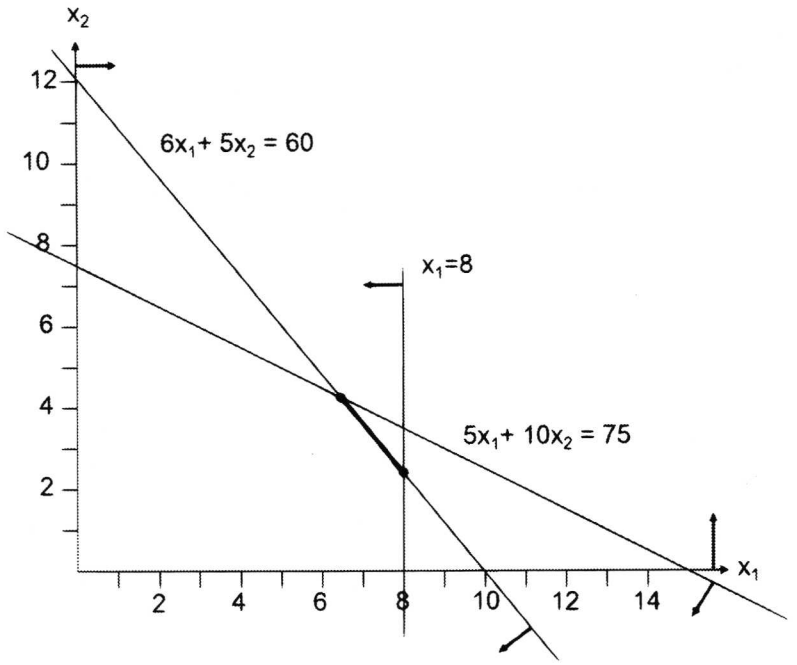


Figure 3.11: Special case: Multiple optimal solutions

3.9 Sensitivity Analysis

Until now we have assumed that all data required is known precisely. We term this as a deterministic approach where we solve a deterministic problem. However, often data is not deterministic but stochastic. That is, we don't know precisely the value of the data but rather a range of possible values. Sensitivity analysis is about determining the impact a change of one parameter of the linear program has on the optimal solution when all other parameters are kept at their original values. Only changing one parameter at a time while not changing the other parameters is termed "ceteris paribus" which is the latin saying for "all else being equal". Parameters of a linear program are the coefficients in the objective function function, the right-hand sides, and the coefficient matrix. In this manuscript, we only present the sensitivity analysis for the coefficients of the objective function of the linear program. Commercial programs for solving linear programs can undertake a sensitivity analysis. E.g., LINDO can undertake a sensitivity analysis for all coefficients in the objective function and all right-hand side values. For each decision variable j with deterministic coefficient c_j in the objective function, we want to determine the range of coefficients $[c_j^{\min}, c_j^{\max}]$ with $c_j^{\min} \leq c_j \leq c_j^{\max}$ such that for each value in the range there is no change of the optimal solution.

3.9.1 Change in the Objective Function Coefficient of a Non-Basic Variable

Let us consider the extended beer glass production planning problem given in Section 3.5.2. The reduced costs of the non-basic variable x_3 are -0.57. Reducing the objective function coefficient by some value $\Delta c_3 > 0$ leads to reduced costs of $-0.57 - \Delta c_3$ and hence leaves the variable out of the basis. Due to this, neither the optimal solution nor the optimal objective function value are changed. Increasing the objective function value coefficient of x_3 by $\Delta c_3 > 0$ increases the reduced costs. For $\Delta c_3 > 0.57$ the reduced costs become positive and thus x_3 has to be selected as a new basic variable. This would change the basis and increase the optimal objective function value. Putting the two observations together, the objective function coefficient c_3 of variable x_3 can vary in the range

$$-\infty \leq c_3 \leq 6.57$$

without changing the optimal objective function value nor the optimal basis.

We can also derive this information more formally. Assuming that there is an unknown change of c_3 by $\Delta c_3 \in \mathbb{R}$ we obtain the following initial tableau:

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6
x_4	60	6	5	8	1	0	0
x_5	75	5	10	5	0	1	0
x_6	8	1	0	0	0	0	1
$-z$	0	5	4.5	$6 + \Delta c_3$	0	0	0

Since the steps of the Simplex just add and subtract number to the objective function coefficient of variable x_3, we obtain the following optimal tableau:

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6
x_2	4.28	0	1	-0.28	-0.14	0.17	0
x_6	1.57	0	0	-1.57	-0.28	$\cdot$ 0.14	1
x_1	6.42	1	0	1.57	0.28	-0.14	0
$-z$	-51.42	0	0	$-0.57 + \Delta c_3$	-0.78	-0.05	0

For this tableau, x_3 is not in the basis as long as

$$-0.57 + \Delta c_3 \leq 0$$

holds. Re-arranging this equation, we obtain

$$\Delta c_3 \leq 0.57.$$

For the coefficient c_3 this implies that

$$-\infty \leq c_3 \leq 6.57.$$

3.9.2 Change of the Objective Function Coefficient of a Basic Variable

Let us consider basic variable x_1 in the optimal tableau of the extended beer glass production planning problem. The objective function coefficient of $c_1 = 5$ is changed by adding $\Delta c_1 \in \mathbb{R}$ which gives the initial tableau

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6
x_4	60	6	5	8	1	0	0
x_5	75	5	10	5	0	1	0
x_6	8	1	0	0	0	0	1
$-z$	0	$5 + \Delta c_1$	4.5	6	0	0	0

and thus the following optimal tableau:

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6	
x_2	4.28	0	1	−0.28	−0.14	0.17	0	(1)
x_6	1.57	0	0	−1.57	−0.28	0.14	1	(2)
x_1	6.42	1	0	1.57	0.28	−0.14	0	(3)
$-z$	−51.42	Δc_1	0	−0.57	−0.78	−0.05	0	(4)

In order to obtain the tableau in standard form with all basic variables having a coefficient of zero in the objective function row we have to perform the elementary row calculation

$$(5) \quad = \quad (4) - \Delta c_1 \cdot (3),$$

which yields

BV	Value	x_1	x_2	x_3	x_4	x_5	x_6	
x_2	4.28	0	1	−0.28	−0.14	0.17	0	(1)
x_6	1.57	0	0	−1.57	−0.28	0.14	1	(2)
x_1	6.42	1	0	1.57	0.28	−0.14	0	(3)
$-z$	$-51.42 - 6.42\Delta c_1$	0	0	$-0.57 - 1.57\Delta c_1$	$-0.78 - 0.28\Delta c_1$	$-0.05 + 0.14\Delta c_1$	0	(5)

This tableau is optimal if for all variables we have an objective function coefficient less than equal zero. That is, it has to hold:

$$\begin{aligned} \text{For column } x_3 : -0.57 - 1.57\Delta c_1 &\leq 0, \\ \text{For column } x_4 : -0.78 - 0.28\Delta c_1 &\leq 0, \text{ and} \\ \text{For column } x_5 : -0.05 + 0.14\Delta c_1 &\leq 0. \end{aligned}$$

Re-arranging the equations we obtain

$$\begin{aligned} \Delta c_1 &\geq -0.369, \\ \Delta c_1 &\geq -2,785, \text{ and} \\ \Delta c_1 &\leq 0.35, \end{aligned}$$

which yields the following feasible range for Δc_1

$$= -0.369 \leq \Delta c_1 \leq 0.35.$$

The objective function coefficient c_1 ot the basic variable can thus vary in the range

$$4.631 \leq c_1 \leq 5.35$$

without a change in the optimal basis. Note that by altering Δc_1 and thus changing c_1 the value of the optimal objective function will change. This can be immediately seen in Row (5) of the optimal tableau.

Chapter 4

Integer and Mixed-Integer Programming

Literature: Bradley et al. [3] Chapter 9.

A general mixed-integer program (MIP) reads as follows:

$$\text{Max } z = \sum_{j=1}^{n} c_j \cdot x_j \tag{4.1}$$

subject to

$$\sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i \quad i = 1, 2, \ldots, m \tag{4.2}$$

$$x_j \geq 0 \quad j = 1, 2, \ldots, n \tag{4.3}$$

$$x_j \in \mathbb{Z} \quad \text{for some } j = 1, 2, \ldots, n \tag{4.4}$$

Mixed-integer programs have continuous variables ($x \geq 0$) and integer variables ($x \in \mathbb{Z}_0^+$). A program where all variables are integer is termed an integer program (IP). Programs with integer variables, which can only take the values 0 and 1 are called binary programs.

4.1 Examples

4.1.1 Production Planning Problem

A production planning problem with integer production quantities is an example for an integer program.

$$\text{Max } z = 5x_1 + 8x_2 \tag{4.5}$$

subject to

$$x_1 + x_2 \leq 6 \tag{4.6}$$

$$5x_1 + 9x_2 \leq 45 \tag{4.7}$$

$$x_1, x_2 \in \mathbb{Z}_0^+ \tag{4.8}$$

4.1.2 Warehouse Location Problem

The warehouse location problem is an example for a mixed-integer program. The problem can be desribed as follows:

- There are m location for warehouses. Running a warehouse at location $i = 1, 2, \ldots, m$ incurs annual cost of f_i.
- There are n areas with demand. Area $j = 1, 2, \ldots, n$ has a demand of d_j.
- Delivering one unit from warehouse in location i to area j incurs transportation cost of $c_{i,j}$ per unit. The transportation costs are termed "variable costs" as they are a function of the number of units transported.
- For each warehouse it has to be decided if the warehouse should be run and for each demand area it has to be decided how many units are received from the opened warehouses such that the demand is fulfilled. The objective is to minimize the total costs which are made up by the costs for running the warehouses and the transportation cost.

We model the problem by defining the binary variable $y_i \in \{0, 1\}$ which is 1, if the warehouse in location i is operated. Furthermore, we employ the continuous variable $x_{i,j} \geq 0$ for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$ stating how many units are transported from warehouse i to demand area j. Using these variables we can calculate the following two costs:

The costs for running the warehouses are $\sum_{i=1}^{m} f_i \cdot y_i$

The costs for transporting goods from the warehouses to demand areas are $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} \cdot x_{i,j}$

The transportation from warehouse i to demand area j are $c_{i,j} \cdot x_{i,j}$. These costs depend on $x_{i,j}$ the number of units transported. If no units are transported between i and j then the costs are zero. Such costs are called "variable costs" as they are a variable of the number of units transported. By contrast, the costs for running a warehouse do not depend on the number of units transported. These costs are called "fixed costs". Once we have decided to open a warehouse, the costs for running it are fixed and do not depend on the number of units which are shipped from that warehouse to any demand area.

We now can model the warehouse location problem as the following mixed-integer program:

$$\text{Min } z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{i,j} \cdot x_{i,j} + \sum_{i=1}^{m} f_i \cdot y_i \tag{4.9}$$

subject to

$$\sum_{i=1}^{m} x_{i,j} = d_j \qquad j = 1,\ldots,n \tag{4.10}$$

$$\sum_{j=1}^{n} x_{i,j} - y_i \left(\sum_{j=1}^{n} d_j\right) \leq 0 \qquad i = 1,\ldots,m \tag{4.11}$$

$$x_{i,j} \geq 0 \qquad i = 1,\ldots,m;\ j = 1,\ldots,n \tag{4.12}$$

$$y_i = \{0,1\} \qquad i = 1,\ldots,m \tag{4.13}$$

The Objective function (4.9) minimizes total costs, i.e., the sum of variable transportation costs and fixed costs for running the opened warehouses. Constraint (4.10) enforce for each area j that the number of units transported to the are aquals the demand. Constraint (4.11) assures for each warehouse i that units can only be shipped from the warehouse ($x_{i,j} > 0$) if it is opened ($y_i = 1$). Constraints (4.12) and (4.13) define the continuous and binary variables, respectively.

4.1.3 Capital Budgeting Problem

The capital budgeting problem is an example for a binary program. The problem can be described as follows:

- There are $j = 1, 2, \ldots, n$ investment alternatives and $i = 1, 2, \ldots, m$ resources. Resource i has a capacity of b_i. Project j generates a value of c_j and requires capacity $a_{i,j}$ from resource j.
- Which investment alternatives should be selected such that the selected investment alternatives do not require more capacity of the resources than available and the total value of the selected investment alternatives is maximized.

We introduce the binary variable $x_j \in \{0, 1\}$ which is 1, if investment alternative j is selected and 0 otherwise. We can now formulate the capital budgeting problem as follows:

$$\text{Max } z = \sum_{j=1}^{n} c_j \cdot x_j \tag{4.14}$$

subject to

$$\sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i \qquad i = 1, 2, \ldots, m \tag{4.15}$$

$$x_j \in \{0, 1\} \qquad j = 1, 2, \ldots, n \tag{4.16}$$

If investment alternatives are interpreted at projects, the problem is also called project portfolio planning problem because a portfolio of projects has to be determined. Further applications of the capital budgeting problem are:

- Selection of research and development projects.
- Selection of investment alternatives.
- Selection of equipment to take on a hiking tour (knapsack problem). In this case, the two resources are weight and volume.
- Selection of freight for airline cargo companies. The value is the contribution margin, the resources are weight and volume.

4.2 Properties of Integer Programs

Let us employ the integer production planning problem introduced in Section 4.1.1.

$$\text{Max } z = 5x_1 + 8x_2 \tag{4.17}$$

subject to

$$x_1 + x_2 \leq 6 \tag{4.18}$$

$$5x_1 + 9x_2 \leq 45 \tag{4.19}$$

$$x_1, x_2 \geq \mathbb{Z}_0^+ \tag{4.20}$$

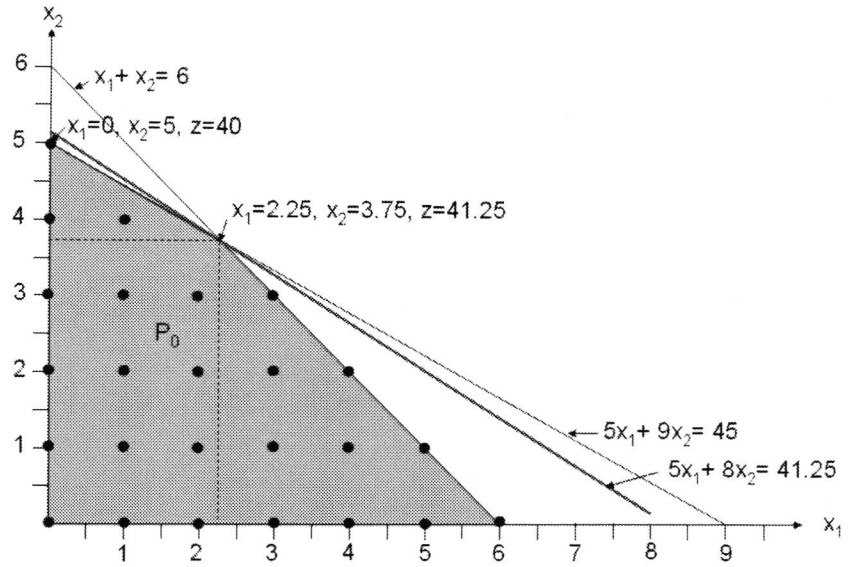


Figure 4.1: Solution space of the continuous and the integer optimization problem

Definition 21 *The* **LP-relaxation** *(LPR) of an integer or a mixed-integer program is obtained by relaxing the integrality constraints of the integer variables. The LP-relaxation can be solved with the Simplex method.*

For the integer program (4.17) – (4.20) we obtain:

	Optimum LP	Rounding up	Rounding down	Optimum IP
x_1	2.25	3	2	0
x_2	3.75	4	3	5
z	41.25	Not feasible	34	40

For this problem we can make the following observations:

- The optimal solution of the linear program (column "Optimum LP") is not integer.
- Rounding up the basic variables of the LP solution (column "Rounding up") is creating an infeasible integer solution.
- Rounding down the basic variables of the LP solution (column "Rounding down") is creating a feasible but not optimal integer solution.
- The objective function value of the optimum LP relaxation is greater than the objective function value of the optimum integer program.

In general we can state the following:

Property 9 *For a* **maximization problem**, *the optimal objective function value of the LP-relaxation is always greater than or equal the optimal objective function value of the integer program (IP). That is, $z^*(LP) \geq z^*(IP)$ holds. The objective function value of the optimal LP-solution is thus an* **upper bound** *of the objective function value of the optimal IP-solution.*

For a **minimization problem** *the optimal objective function value of the LP-relaxation is always less than or equal the optimal objective function value of the integer program (IP). That is, $z^*(LP) \leq z^*(IP)$ holds. The objective function value of the optimal LP-solution is thus a* **lower bound** *of the objective function value of the optimal IP-solution.*

4.3 Branch–and–Bound

Branch–and–Bound is a method for solving integer, mixed-integer programs and binary programs. The main idea is to divide the overall problem into sub-problems. For each sub-problem we solve a relaxation of the integer problem, which often but not necessarily is an LP relaxation. The relaxation renders the difficult integer problem into a problem which is easier to solve. Having obtained the optimal solution of the relaxed problem, we distinguish between the following four cases:

1: The solution is feasible for the integer problem. Then we are done with this sub-problem.

2: The solution is not feasible for the integer problem and does not have an objective function value which is better (larger in case of a maximization problem and smaller in the case of a minimization problem) than the one of the best integer solution we have obtained so far. We call the best feasible solution, which we have found so far, the "incumbent solution". Due to Property 9 we know that any integer solution emanating from the sub-problem at hand cannot have a better objective function value than the sub-problem. Hence, we don't have to search for an integer solution emanating from this sub-problem and we are done with the sup-problem.

3: There is no feasible solution for the relaxed problem and hence there can be no feasible solution for the integer problem. Hence, we are done with this sub-problem.

4: The solution is not feasible for the integer problem and does have an objective function value which is better than the incumbent solution. In this case, there is the chance that there is an integer solution emanating from our problem with an improved objective function value for the overall problem.

For case 4 we substitute the problem at hand with two new sub-problems such that the union of the two sub-problems has the same solution space as the problem at hand but that the non-integer solution is excluded.

Starting with the integer problem, branch–and–bound scrutinizes problems according to the four cases above. For cases 1 – 3 the problem does not have to be considered any more. For case 4, two new sub-problems are created. The set of all problems which still have to be scrutinized is managed with a candidate list K.

4.3.1 Solving the Relaxation with the Simplex Algorithm

Let us consider the integer program (4.17) – (4.20) which is the starting problem and stated as P_0. At the beginning, the candidate list consists of this problem only, i.e., $K = \langle P_0]$.

Problem P_0: We derive the optimal solution of the LP-relaxation of P_0 by solving it with the Simplex algorithm. We obtain $x_1 = 2.25$, $x_2 = 3.75$ and $z = 41.25$. This is clearly the upper bound and hence we have $\bar{z} = 41.25$. We derive a lower bound $\underline{z}$ for P_0 by rounding down all fractional variables of the LP-solution. This gives $x_1 = 2$, $x_2 = 3$ with $z = 34$ and thus we have $\underline{z} = 34$. This gives the incumbent solution. An easy (but worse) lower bound can be derived by setting all variables to 0, i.e., $x_1 = 0$ and $x_2 = 0$, which gives $\underline{z} = 0$.

We now have to decide what to do with the problem P_0 according to the four cases stated above. For the LP relaxation of P_0 we have Case 4 since $z > \underline{z}$, We thus have to substitute P_0 into two supbroblems. We do this by creating the following two sub-problems

- P_0 and constraint "$x_2 \leq 3$" $\Rightarrow$ sub-problem P_1
- P_0 and constraint "$x_2 \geq 4$" $\Rightarrow$ sub-problem P_2

Note that for the integer problems it holds $P_0 = P_1 \cup P_2$ as can be seen in Figure 4.2.

In general, we proceed as follows:

Creating of sub-problems: Having a problem P_i, for which Case 4 applies and branching on an integer variable x, for which we have obtained the non-integer optimal value x^* we generate a "left" problem P_{i+1} by adding the constraint "$x \leq \lfloor x^* \rfloor$" to problem P_i and a "right" problem P_{i+2} by adding the constraint "$x \geq \lceil x^* \rceil$" to problem P_i. In case of several integer variables with non-integer values there has to be a rule, which of the variables is selected for branching. In our example we have selected x_2. One possible rule is to branch on the fractional variable, which has the smallest distance to an integer is largest, i.e., the i for which the term $\min(\lceil x_i^* \rceil - x_i^*, x_i^* - \lfloor x_i^* \rfloor)$ is largest.

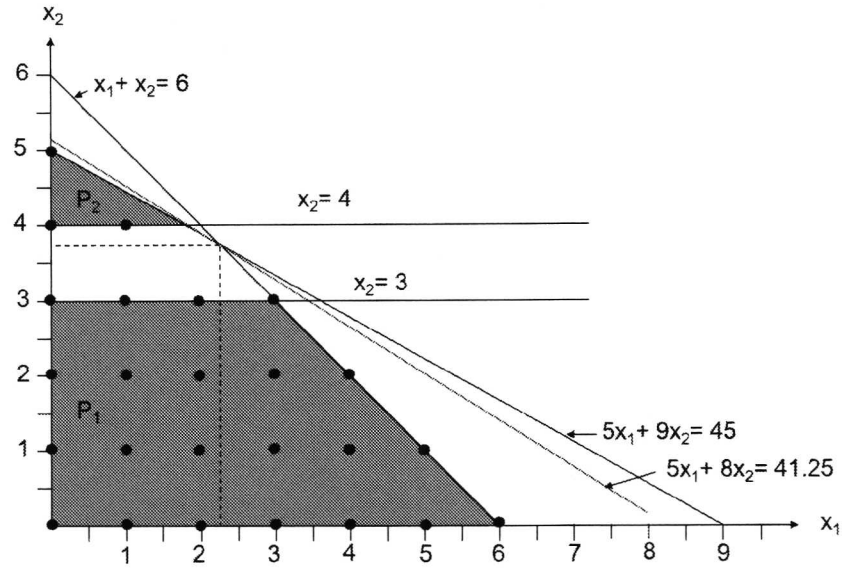


Figure 4.2: Sub-problems emanating from P_0

Due to branching and creating the two new problems P_1 and P_2 we don't have to consider problem P_0 any more. We thus delete P_0 from the candidate list and add the two new problems P_1 and P_2, i.e., the candidate list is updated as follows $K = \langle P_0] \rightarrow K = \langle P_1, P_2]$. Since the original problem P_0 has been replaced by the two problems P_1 and P_2, the upper bound of the objective function cannot be larger than $\max\{z_{LP}(P_1), z_{LP}(P_2)\} = 41$. We update $\bar{z}$ accordingly. In general, the upper bound is the maximum of the objective function of the LP-relaxations of all sub-problems in the candidate list, i.e., $\bar{z} = \max\{z_{LP}(P_i) \mid i \in K)\}$.

We proceed by selecting one problem out of the candidate list. There are different rules when selecting a problem from the candidate list:

- Last-in-first-out (LIFO): The last problem, which had been added to the candidate list is selected. In our example, the last two problems added were P_1 and P_2. In this case we need a second rule, a so-called tiebreaker. We define the second rule to select the problem with the larger number, which is in our case P_2. When applying the LIFO-rule, the branch-and-bound tree is developed vertically. Hence, applying this rule is also called "depth-first" search.

- Maximum-upper-bound (MUB): For a maximization problem we select the problem from the candidate list with the largest objective function value for the optimal solution of the relaxation. When applying the MUB-rule, the branch-and-bound tree is developed horizontally. Hence, applying this rule is also termed breadth–first search.

Let us use the LIFO-rule for solving our example problem. Hence we select problem P_2.

Problem P_2: Solving the LP relaxation we obtain $x_1 = 1.8$, $x_2 = 4$ and $z = 41$. We observe that the optimum objective function value of problem P_2 is less than the optimal objective function of problem P_0. In general, the following holds:

Property 10 *For a maximization problem the optimal objective function value of a sub-problem is always less than or equal to the optimal objective function value of the parent problem.*

Property 10 holds because we derive a sub-problem by adding a constraint to the superordinate problem. If the new constraint is binding, then the optimal objective function value of the sub-problem is smaller. If the new constraint is not binding, then the optimal objective function of the sub-problem is equal to the one of the superordinate problem. Property 10 can be clearly seen in the branch–and–bound tree provided in Figure 4.5.

Since the optimal solution of the relaxation of problem P_2 is not integer and has an objective function value greater than the lower bound $\underline{z}$ we have Case 4. Thus, we have to divide problem P_2 into two sub-problems. Since only variable x_1 is non–integer we create the two sub-problems by adding constraints regarding x_1 as follows:

- $x_1 \leq 1 \Rightarrow$ sub-problem P_3
- $x_1 \geq 2 \Rightarrow$ sub-problem P_4

We add the two new problems to the candidate list and remove problem P_2 from the candidate list which gives $(P_4, P_3, P_1]$. Note that the problems in the list are sorted according to the LIFO-rule and the tiebreaker rule (largest problem number first). Figure 4.3 gives the solution space. The upper bound is updated to $\bar{z} = \max\{-, 40.56, 39\} = 40.56$. The entry "$-$" denotes that the LP-relaxation of problem P_4 does not have a feasible solution.

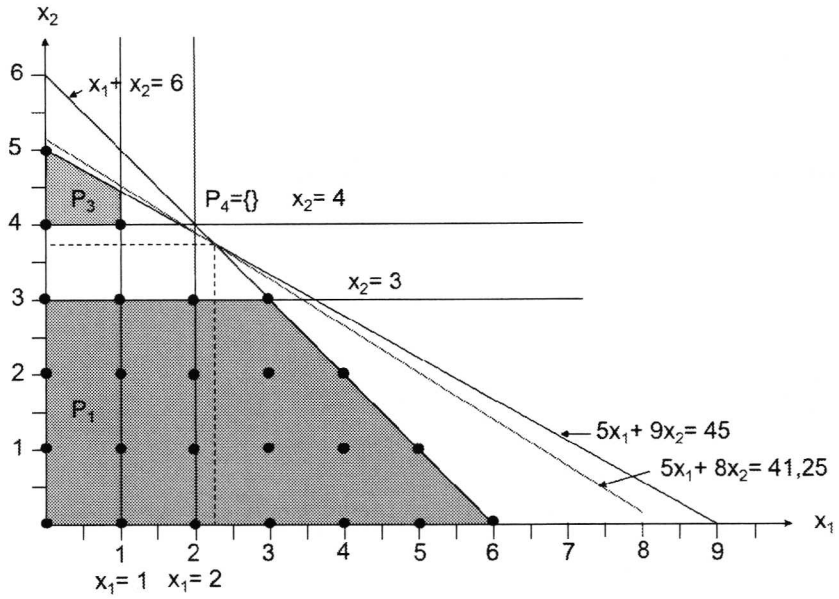


Figure 4.3: Sub-problems P_1, P_3 and P_4

Problem P_4: We select P_4 as the first problem in the candidate list. The LP-relaxation (and thus P_4) is not feasible and we can eliminate the problem by Case 3. The updated candidate list is $\langle P_3, P_1]$. Since we have not added a new sub-problem to the list, there is no update of the upper bound.

Problem P_3: We select P_3 from the candidate list. The optimal solution of the LP–relaxation is $x_1 = 1$, $x_2 = 4.45$ and $z = 40.56$. We thus have Case 4 and have to create new sub-problems:

- $x_2 \leq 4 \Rightarrow$ sub-problem P_5
- $x_2 \geq 5 \Rightarrow$ sub-problem P_6

Adding the new problems to the candidate list and removing problem P_3 from the list gives $\langle P_6, P_5, P_1]$ with the new upper bound $\bar{z} = \max\{40, 37, 39\} = 40$. Figure 4.4 shows the solution space.

Problem P_6: The next problem is P_6. The optimal LP–relaxation is $x_1 = 0$, $x_2 = 5$ and $z = 40$. This solution is integer and hence we have Case 1. Since $z = 40$ is greater than the lower bound we have found a new incumbent and consequently we update the lower bound to $\underline{z} = 40$. Finally, we remove problem P_6 from the candidate list which yields $\langle P_5, P_1]$. The best objective function we can achieve with these two sub-problems is $\bar{z} = \max\{40, 39\} = 40$. Since the incumbent solution has an objective function value of $\underline{z} = 40$, we cannot find a better solution. Hence we have found an optimal solution and can stop the branch-and-bound

algorithm. Formally, we stop the branch-and-bound algorithm for $\underline{z} = \bar{z}$. The solution found is the incumbent, which is $x_1 = 0$, $x_2 = 5$ with objective function $z = 40$.

Table 4.1 and Figure 4.5 summarize the application of the branch–and–bound procedure for the example problem. The number given in parentheses on the right side the problem number in Figure 4.5 provide the information on the sequence in which the problems are selected from the candidate list. E.g., "$P_6(5)$" states that problem P_6 is the fifth problem which is picked from the list. Figure 4.6 shows the solutions space of the LP-relaxation if of P_0 as the grey area. The integer solutions are all intersections of a horizontal and a vertical line within the grey area. Furthermore, the LP relaxations of all sub-problems are provided. Note that in total the branch–and–bound procedure has scrutinized five problems, i.e., P_0, P_2, P_3, P_4, and P_6, which is considerably less than enumerating all 25 integer solutions. In column 8 of Table 4.1, Δ denotes the solution gap, which is calculated as follows:

$$\Delta = \frac{|z(\text{best bound}) - z(\text{incumbent})|}{|z(\text{incumbent})|}.$$

The solution gap is calculated for any state of the branch-and-bound algorithm. It measures the difference of the objective function values between the best possible solution and the incumbent solution as a percentage of the objective function value of the incumbent solution. Example: After having solved problem P_0, we obtain a solution gap of $\Delta = 21.32\%$. This indicates that based on the existing incumbent with objective function 34 we can at most improve by 21.32% to an objective function of 41.25. The best bound is the best solution, which is still possible. In case of a maximization problem, the objective function of the

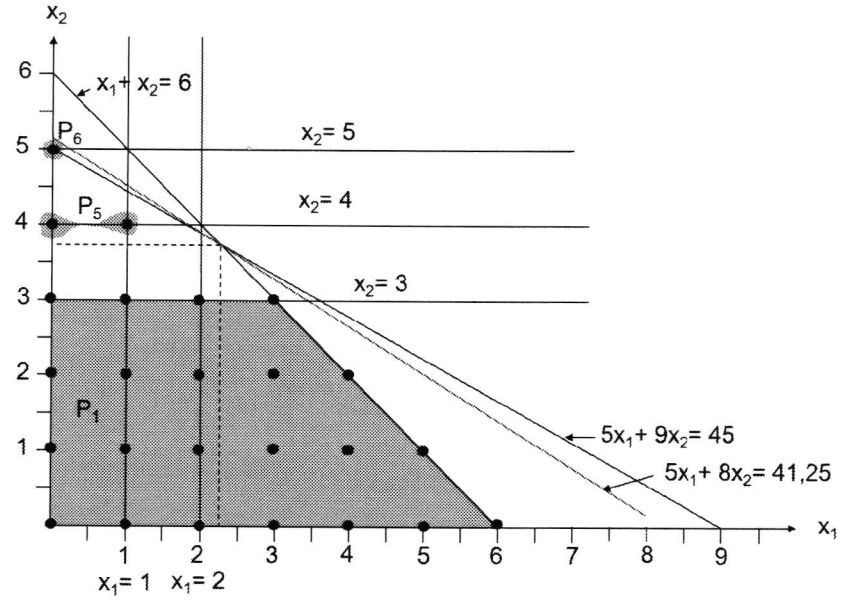


Figure 4.4: Sub-problems P_1, P_5 and P_6

incumbent solution is $\underline{z}$ and the objective function of the best bound is $\overline{z}$. In case of a minimization problem, the objective function of the incumbent solution is $\overline{z}$ and the objective function of the best bound is $\underline{z}$. When the solution gap turns zero, the optimal solution has been found.

		Sol. LP-relax.						Incumbent		
K	P_{i^*}	x_1	x_2	z	$\overline{z}$	$\underline{z}$	Δ	IP sol.	Case	Branch
$\langle P_0]$	P_0	2.25	3.75	41.25	41.25	34	21.32%	$x_1 = 2$ $x_2 = 3$	4)	$x_2 \leq 3 \rightarrow P_1$ $x_2 \geq 4 \rightarrow P_2$
$\langle P_2, P_1]$	P_2	1.8	4	41	41	34	20.59%		4)	$x_1 \leq 1 \rightarrow P_3$ $x_1 \geq 2 \rightarrow P_4$
$\langle P_4, P_3, P_1]$	P_4	–	–	–	40.56	34	19.29%		3)	
$\langle P_3, P_1]$	P_3	1	4.44	40.56	40.56	34	19.29%		4)	$x_2 \leq 4 \rightarrow P_5$ $x_2 \geq 5 \rightarrow P_6$
$\langle P_6, P_5, P_1]$	P_6	0	5	40	40	40	0%	$x_1 = 0$ $x_2 = 5$	1)	
Opt. Sol.		0	5	40						

Table 4.1: Protocol of the Branch–and–Bound: LIFO and first branching on x_2

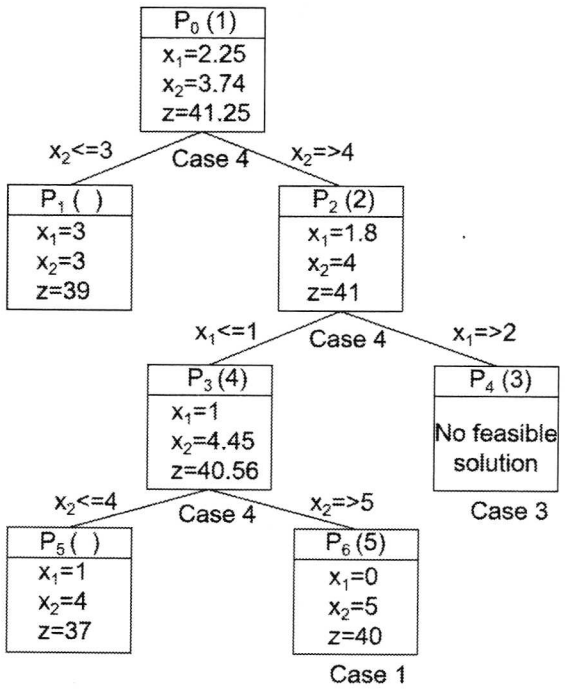


Figure 4.5: Branch–and–Bound tree when using LIFO and first branching on x_2

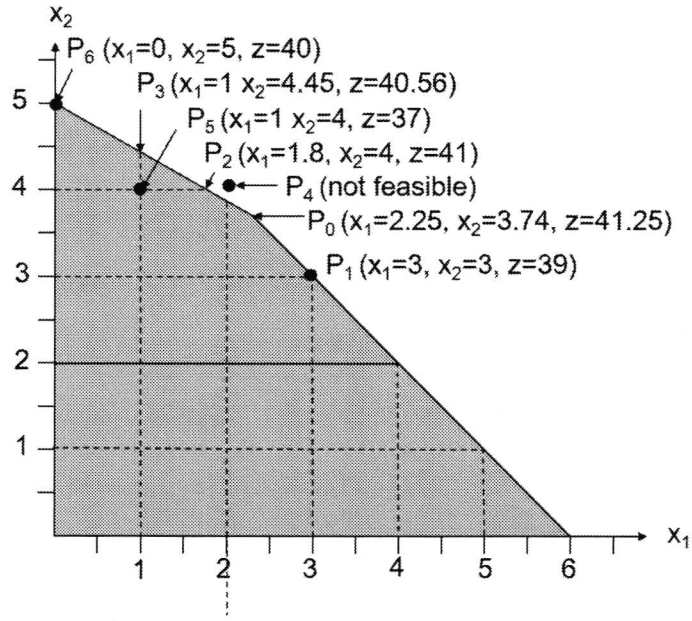


Figure 4.6: Solutions visited by the Branch-and-Bound procedure with LIFO and branching on x_2

The flowchart given in Figure 4.7 provides the details of the branch–and–bound procedure for solving a maximization problem. The notation is given in Table 4.2.

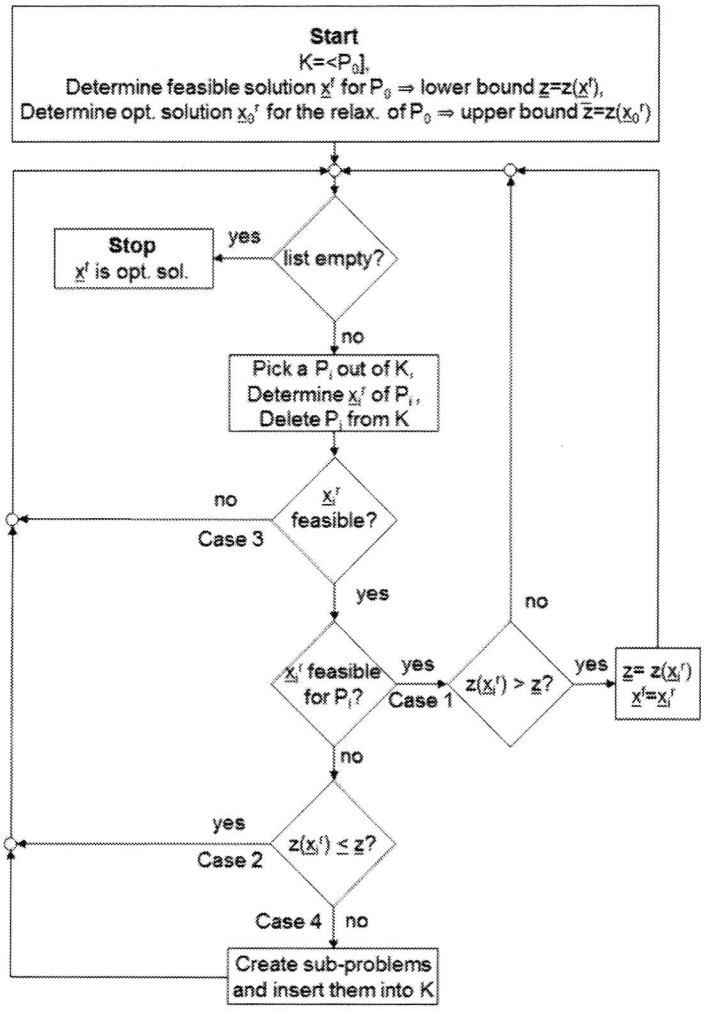


Figure 4.7: Flowchart of the branch–and–bound procedure

P_i	= (Sub-)Problem i $(i = 0, 1, \dots)$
$\langle]$	= Candidate list
$\underline{x}^f$	= Feasible solution for the integer problem P_0
$\underline{x}_0^r$	= Optimal solution of the relaxation of the starting problem P_0
$\underline{x}_i^r$	= Optimal solution of the relaxation of (sub-)problem P_i $(i = 0, 2, \dots)$
$z(\underline{x})$	= Objective function value of solution $\underline{x}$
$\underline{z}$	= Lower bound of the objective function value of the (starting) problem P_0
$\overline{z}$	= Upper bound of the objective function value of the (starting) problem P_0

Table 4.2: Notation for the Branch–and–Bound flowchart

Table 4.3 gives the protocol when solving the problem with Branch-and-Bound where sub-problems are selected with LFIO and the first branching is on x_1.

K	P_{i^*}	Sol. LP-relax. x_1	x_2	z	$\overline{z}$	$\underline{z}$	Δ	Best IP sol.	Case	Branch
$\langle P_0]$	P_0	2.25	3.75	41.25	41.25	34	21.32%	$x_1 = 2$ $x_2 = 3$	4)	$x_1 \leq 2 \rightarrow P_1$ $x_1 \geq 3 \rightarrow P_2$
$\langle P_2, P_1]$	P_2	3	3	39	41.11	39	5.41%		1)	
$\langle P_1]$	P_1	2	3.88	41.11	41.11	39	5.41%		4)	$x_2 \leq 3 \rightarrow P_3$ $x_2 \geq 4 \rightarrow P_4$
$\langle P_4, P_3]$	P_4	1.8	4	41	41	39	5.13%		4)	$x_1 \leq 1 \rightarrow P_5$ $x_1 \geq 2 \rightarrow P_6$
$\langle P_6, P_5, P_3]$	P_6	–	–	–	40.56	39	4.00%		3)	
$\langle P_5, P_3]$	P_5	1	4.4	40.56	40.56	39	4.00%		4)	$x_2 \leq 4 \rightarrow P_7$ $x_2 \geq 5 \rightarrow P_8$
$\langle P_8, P_7, P_3]$	P_8	0	5	40	40	40	0%	$x_1 = 0$ $x_2 = 5$	1)	
Opt. Sol.		0	5	40						

Table 4.3: Protocol of the Branch–and–Bound: LIFO and first branching on x_1

4.3.2 Truncated Branch–and–Bound

In the case of truncated branch–and–bound we don't stop the procedure until there are no more candidate problems in the list but as soon as we have found a solution with a solution gap, which is less than or equal to a maximal solution gap $\overline{\Delta}$. E.g, for $\overline{\Delta} = 20\%$, the truncated branch–and–bound procedure would have stopped with the incumbent solution $x_1 = 2, x_2 = 3$, and $z = 24$ once the list $K = \langle P_4, P_3, P_1]$ with upper bound $\overline{z} = 40.56$ had been generated because $\Delta = 19.29 < \overline{\Delta} = 20$.

Figure 4.8 gives the flowchart for the truncated branch–and–bound procedure.

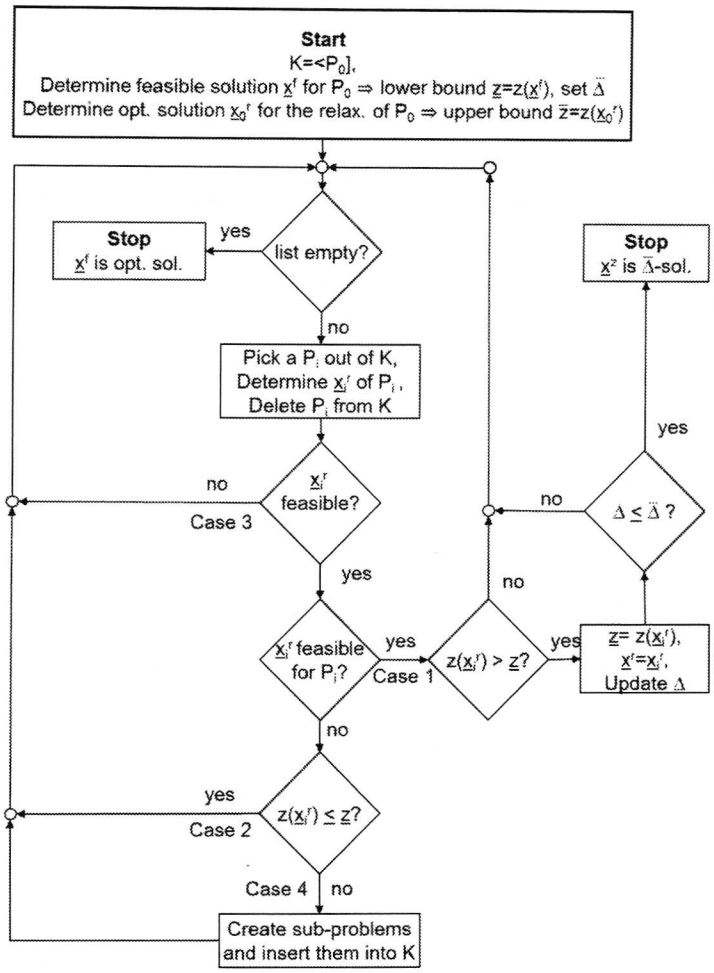


Figure 4.8: Flowchart of the truncated branch–and–bound procedure

4.3.3 Solving the Relaxation without the Simplex Algorithm

Reference: Winston [11] pp. 467 – 468 and 516 – 519.

Let us consider the following capital budgeting problem with an available budget of 14:

Project (j)	Value (c_j)	Required Budget (a_j)	Value per required budget unit ($\frac{c_j}{a_j}$)
1	16	5	3.20
2	22	7	3.14
3	12	4	3.00
4	8	3	2.67

Table 4.4: Capital budgeting example

Selecting all four projects would require a budget of $5 + 7 + 4 + 3 = 19$ which obviously is not feasible. Hence we have to select a subset of the four projects which does not require a budget of more than 14 and which maximizes the sum of the values. Modeling the problem as binary problem we obtain:

$$\text{Max } z = 16x_1 + 22x_2 + 12x_3 + 8x_4 \tag{4.21}$$

$$\text{subject to}$$

$$5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \tag{4.22}$$

$$x_j \in \{1, 0\} \quad (j = 1, 2, 3, 4) \tag{4.23}$$

Let us first determine a lower and an upper bound for the problem.

Upper Bound. For this we relax problem (4.21) – (4.23) by defining the binary variables x_j to be continuous in the interval $0 \leq x_j \leq 1$. That is we can select from each project a continuous amount between 0 and 100%. Now looking at the value per required budget unit $\frac{c_j}{a_j}$ of the projects we make a decision on the relaxed x_j such that the scarce budget is used to create the maximum value. Clearly we select first the project with the highest $\frac{c_j}{a_j}$ which is project 1. From this project we realize as much as possible which is $x_1 = 1$. By this we have created a value of 16 and the remaining budget is $14 - 5 = 9$. Now we ask ourselves which of the projects still available (projects 2, 3 and 4) makes the largest contribution per budget unit. This is project 2. We try to realize as much as possible of this project which is $x_2 = 1$. This increases the value of the selected projects to 38 and the remaining budget is decreased to $9 - 7 = 2$. The next project to be selected is project 3 and the largest amount which can be realized is $x_3 = 0.5$ since project 3 requires a budget of 4 but only 2 budget unit are left. By selecting $x_3 = 0.5$ the value is increased to $38 + 0.5 \cdot 12 = 44$ and the remaining budget is decreased to $2 - 0.5 \cdot 4 = 0$. The solution obtained is the optimal solution of the linear problem (4.21) – (4.22) with $0 \leq x_j \leq 1$ and thus we have an upper bound $\bar{z} = 44$. Note that have obtained this bound without solving the problem with the Simplex algorithm.

Lower Bound. For the lower bound we take the solution obtained and round down the fractional variables which gives $x_1 = 1$, $x_2 = 1$, $x_3 = 0$ and $x_4 = 0$ with $z = 38$ and thus $\underline{z} = 38$. Furthermore we now have $\Delta = \frac{44-38}{38} = 15.79\%$.

For building up the branch–and–bound tree we proceed as follows:

Branching. Whenever we encounter Case 4 we branch on the non–integer variable x_j. Note that when Case 4 applies to the capital budgeting problem we have exactly one non–integer variable and hence we don't have to make a decision on the non-integer variable used for branching. We create two branches where the left branch is setting the variable to 0 while the right branch is setting the variable to 1. The sub-problem created by setting $x_j = 0$ is given the smaller and the sub-problem created by setting $x_j = 1$ is given the higher number.

Selecting a problem from the candidate list. We select the problem with the maximum

upper bound (maximum upper bound rule, MUB). For this we have to calculate for each new sub-problem its upper bound.

Table 4.5 and Figure 4.9 provide the details when applying the branch–and–bound procedure to the capital budgeting problem. Due to using the MUB–rule we have the information on the upper bound of a sub-problem whenever it enters the candidate list. Thus we can in each iteration update the upper bound $\bar{z}$ by taking the maximum upper bound values of all problems in the candidate list. For example, for the candidate list $\langle P_2, P_1]$ we have the upper bounds 43.33 for P_1 and 43.71 for P_2 and thus $\bar{z} = 43.71$. The upper bound 44 of problem P_0 is not valid any more since the two sub-problems P_1 and P_2 have replaced P_0 and thus $x_3 = 0.5$ is not feasible. In the case of updating $\bar{z}$ we can stop with the optimal solution after having selected P_5. In the case of not updating $\bar{z}$ we would have to proceed. Table 4.5 and Figure 4.9 provide the information for the case of not updating.

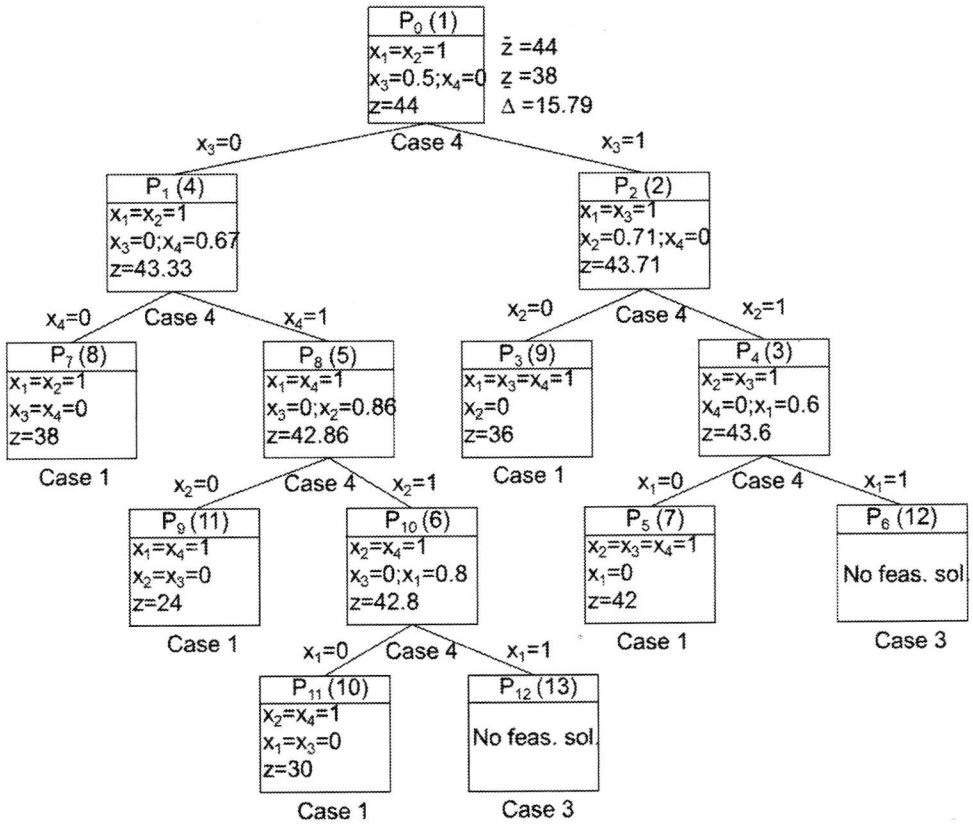


Figure 4.9: Branch–and–bound tree for the capital budgeting problem

K	P_{i^*}	Solution of the relaxation x_1	x_2	x_3	x_4	z	$\overline{z}$ / $\underline{z}$	Δ	*)	Branch	UB
$\langle P_0]$	P_0	1	1	0.5	0	44	44 38	15.79%	4)	$x_3 = 0 \rightarrow P_1$ $x_3 = 1 \rightarrow P_2$	43.33 43.71
$\langle P_2, P_1]$	P_2	1	0.71	1	0	43.71	43.71 38	15.03%	4)	$x_2 = 0 \rightarrow P_3$ $x_2 = 1 \rightarrow P_4$	36 43.6
$\langle P_4, P_1, P_3]$	P_4	0.6	1	1	0	43.6	43.6 38	14.74%	4)	$x_1 = 0 \rightarrow P_5$ $x_1 = 1 \rightarrow P_6$	42 n.feas.
$\langle P_1, P_5,$ $P_3, P_6]$	P_1	1	1	0	0.67	43.33	43.33 38	14.03%	4)	$x_4 = 0 \rightarrow P_7$ $x_4 = 1 \rightarrow P_8$	38 42.86
$\langle P_8, P_5, P_7,$ $P_3, P_6]$	P_8	1	0.86	0	1	42.86	42.86 38	12.79%	4)	$x_2 = 0 \rightarrow P_9$ $x_2 = 1 \rightarrow P_{10}$	24 42.8
$\langle P_{10}, P_5, P_7,$ $P_3, P_9, P_6]$	P_{10}	0.8	1	0	1	42.8	42.8 38	12.63%	4)	$x_1 = 0 \rightarrow P_{11}$ $x_1 = 1 \rightarrow P_{12}$	30 n.feas.
$\langle P_5, P_7, P_3, P_{11},$ $P_9, P_6, P_{12}]$	P_5	0	1	1	1	42	42 42	0%	1)		
$\langle P_7, P_3, P_{11},$ $P_9, P_6, P_{12}]$	P_7	1	1	0	0	38			1)		
$\langle P_3, P_{11}, P_9,$ $P_6, P_{12}]$	P_3	1	0	1	1	36			1)		
$\langle P_{11}, P_9,$ $P_6, P_{12}]$	P_{11}	0	1	0	1	30			1)		
$\langle P_9, P_6, P_{12}]$	P_9	1	0	0	1	24			1)		
$\langle P_6, P_{12}]$	P_6	–	–	–	–	n.feas.			3)		
$\langle P_{12}]$	P_{12}	–	–	–	–	n.feas.			3)		
$\langle]$											
Opt. solution		0	1	1	1	42					

*) = Case for eliminating the problem P_{i^*}

Table 4.5: Protocol of the Branch–and–Bound procedure when solving the capital budgeting problem

Chapter 5

Network Flow Problems

Literature:

- Bertsekas [2] Chapter 1.
- Domschke et al. [4] Chapter 3.
- Winston [11] Chapter 8.
- Jensen and Bard [8] Chapters 5 and 6.

Network flow problems can be modeled and solved as linear programs. Also, there are algorithms available, which solve specific network flow problems more efficiently than applying the Simplex to a linear program formulation.

5.1 Basics and Definitions

Definition 22 *A* **directed graph (digraph)** $G = (V, A)$ *is made of a set of nodes* V *and a set of arcs* A. *An arc* $(i, j) \in A$ *is defined by a start node* $i \in V$ *and a finish node* $j \in V$. *An arc* (i, j) *may have a weight (or cost)* $c_{i,j}$. *A directed graph with arc weights is denoted as* $G = (V, A, c)$.

Definition 23 *An* **undirected graph** $G = [V, A]$ *is made of a set of nodes* V *and a set of edges* A. *An edge* $[i, j]$ *is defined by the two nodes* i *and* j. *An undirected graph where edges have costs is denoted as* $G = [V, A, c]$.

An undirected graph can be depicted by a directed graph by substituting each edge $[i, j]$ by the arc (i, j) and the reverse arc (j, i). Figures 5.1 shows from left to right a directed graph, a directed graph with arc weights, and an undirected graph.

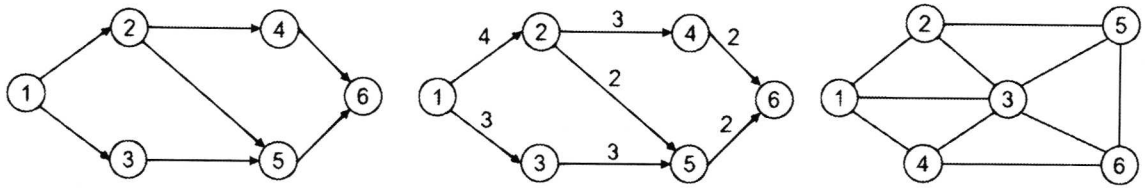


Figure 5.1: Directed graph (left), directed weighted graph (middle) and undirected graph (right)

Definition 24 *A* **path** *in a digraph $G = (V, A)$ is a sequence of $k \geq 2$ nodes $(n_1, n_2, \ldots, n_k)$ and the associated $k-1$ arcs $((n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, n_k))$.*

Example: The path from node 1 via node 2 to node 5 in the left part of Figure 5.1 is defined by the node sequence $(1, 2, 5)$ and the associated arcs $((1, 2), (2, 5))$.

5.2 Minimum Cost Flow Problems

We consider the directed graph $G = (V, A, c, \lambda, \kappa)$ with node set V, arc set A, arc costs $c_{i,j} \in \mathbb{R}$, minimum arc flows $\lambda_{i,j} \geq 0$, and maximum arc flows $\kappa_{i,j} \geq 0$. For each node $i \in V$ we have a net flow $f_i \in \mathbb{R}$. The net flow is the difference between the flow out of node i and the flow into node i, i.e., net flow = outflow − inflow. A net flow of $f_i > 0$ indicates that node i has a net supply and a net flow of $f_i < 0$ indicates that node i has a net demand. Accordingly, we call these nodes "supply nodes" and "demand nodes", respectively. Examples for supply nodes are production facilities, which produce a certain number of units per period. An example for a demand node is a retailer, which sells a certain number of units to customers. Nodes for which $f_i = 0$ holds are called "transshipment nodes". Examples for transshipment nodes are warehouses. On average, a warehouse has the same number of units entering and leaving the node.

Consider the example given in Figure 5.2 with node set $V = \{1, 2, 3, 4, 5\}$, arc set $A = \{(1, 3), (1, 4), (2, 4), (3, 5), (4, 5)\}$, net flows $f = (4, 1, -2, 0, -3)$, and costs $c_{1,3} = c_{2,4} = 3$, $c_{1,4} = c_{3,5} = c_{4,5} = 2$. Arc $(4, 5)$ has a minimum flow of $\lambda_{4,5} = 2$ and a maximum flow of $\kappa_{4,5} = 3$. The flows of all other arcs are unrestricted, i.e., $\lambda_{i,j} = 0$ and $\kappa_{i,j} = \infty$ for $(i, j) \in A \setminus \{(4, 5)\}$.

The linear program formulation of the minimum cost flow problem is given in (5.1) - (5.5).

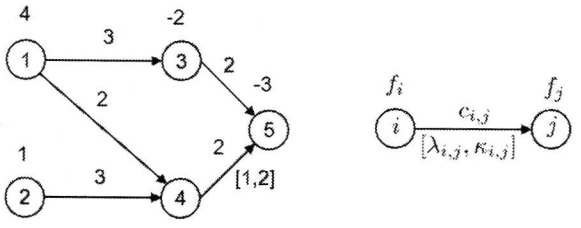


Figure 5.2: Example of the min cost flow problem

$$\text{Max } z = \sum_{(i,j)\in A} c_{i,j} \cdot x_{i,j} \tag{5.1}$$

s.t.

$$\sum_{(i,j)\in A} x_{i,j} - \sum_{(h,i)\in A} x_{h,i} = f_i \quad (i \in V) \tag{5.2}$$

$$x_{i,j} \geq \lambda_{i,j} \quad ((i,j) \in A) \tag{5.3}$$

$$x_{i,j} \leq \kappa_{i,j} \quad ((i,j) \in A) \tag{5.4}$$

$$x_{i,j} \geq 0 \quad ((i,j) \in A) \tag{5.5}$$

The decision variable $x_{i,j}$ defined in constraint (5.5) gives the flow for each arc $(i,j) \in A$. The objective function (5.1) minimizes the total cost, which is the product of the cost per unit of flow $c_{i,j}$ multiplied with the flow $x_{i,j}$ summed up over all arcs. Constraint (5.2) gives for each node i the associated flow balance constraints, which states that the sum of flows out of node i minus the sum of the flows into node i equals the net flow of node i. The constraint (5.3) defines for each arc (i,j) the minimum flow and the constraint (5.4) defines for each arc (i,j) the maximum flow. Note that for a feasible solution, $\sum_{i\in V} f_i = 0$ has to hold, i.e., the total supply has to be equal to the total demand.

The linear program of the minimum cost flow problem for the example given in Figure 5.2 is as follows:

$$\text{Max } 3x_{1,3} + 2x_{1,4} + 3x_{2,4} + 2x_{3,5} + 2x_{4,5}$$

s.t.

$$\begin{aligned}
x_{1,3} + x_{1,4} &= 4 \\
x_{2,4} &= 1 \\
x_{3,5} - x_{1,3} &= -2 \\
x_{4,5} - x_{1,4} - x_{2,4} &= 0 \\
-x_{3,5} - x_{4,5} &= -3 \\
x_{4,5} &\geq 2 \\
x_{4,5} &\leq 3 \\
x_{1,3}, x_{1,4}, x_{2,4}, x_{3,5}, x_{4,5} &\geq 0
\end{aligned}$$

Table 5.1 presents this linear program in a variant of the tableau representation introduced

in Chapter 3. The difference to the representation used in Chapter 3 is that the LP is not required to be in standard form, there is no column, which gives the base variable for each constraint, and that the value of the right-hand side is given in the rightmost column. We only provide nonzero entries in the tableau. If there is no entry, then the coefficient is 0.

	$x_{1,3}$	$x_{1,4}$	$x_{2,4}$	$x_{3,5}$	$x_{4,5}$		RHS
Objective function:	3	2	3	2	2		
Flow balance constraint node 1:	1	1				$=$	4
Flow balance constraint node 2:			1			$=$	1
Flow balance constraint node 3:	-1			1		$=$	-2
Flow balance constraint node 4:		-1	-1		1	$=$	0
Flow balance constraint node 5:				-1	-1	$=$	-3
Min flow constraint arc $(4,5)$:					1	$\geq$	2
Max flow constraint arc $(4,5)$:					1	$\leq$	3

Table 5.1: LP of the example of the minimum cost flow problem in tableau notation

The information on the graph is represented in the flow balance constraints of Tableau 5.1. Let us first look at the columns of Tableau 5.1. Each columns represents the flow variable $x_{i,j}$ of one arc (i,j). Going down the rows of the column, we have for the starting node of the arc a 1 entry and for the ending node of the arc a -1 entry. All other entries are 0. E.g., for the column of variable $x_{1,3}$ we have a 1 entry in the flow balance constraint of node 1 and a -1 entry in the flow balance constraints of node 3. Let us now look at the rows. Each row of the flow balance constraints represents one node i. Reading the row of the flow balance constraint for node i we have a -1 entry for all columns $x_{h,i}$, for which the arc (h,i) leads into node i and a 1 entry for all columns $x_{i,j}$, for which the arc (i,j) leads out of node i. E.g., for the flow balance constraint relating to node 4, we have the entries -1 for the flow variables $x_{1,4}$ and $x_{2,4}$ associated with the arcs $(1,4)$ und $(2,4)$ leading into node 4 and we have the entries 1 for the flow variable $x_{4,5}$ associated with the arc $(4,5)$ leading out of node 4

Solving the example problem gives the solution depicted in Figure 5.3 with total cost of 20. Note that the solution is integer although we have solved an LP. This is due to the special structure of the minimum cost flow problem. However, obviously it holds only for the case of integer net flows f_i.

5.2.1 The Transportation Problem

The transportation problem is a special minimum cost flow problem where there are only supply nodes and demand nodes but no transshipment nodes. There are no arcs between any pair of supply nodes and between any pair of demand nodes. Arcs are only leading from supply nodes to demand nodes. An arc from a supply node i to a demand node j represents a transportation link between the two nodes. If there exist no arc between supply node i

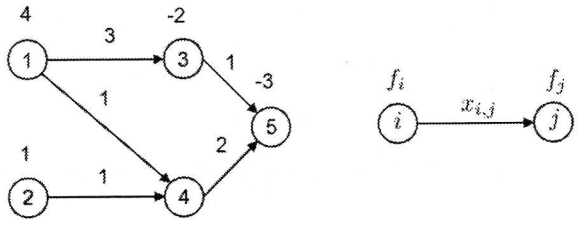


Figure 5.3: Optimal solution of the example of the the minimum cost flow problem

and demand node j, no flow can exist from node i to node j. Figure 5.4 gives an example of a transportation problem for which Table 5.2 provides the tableau of the linear program. Note that the cost matrix c given in Figure 5.4 provides only entries for arcs given in the graph.

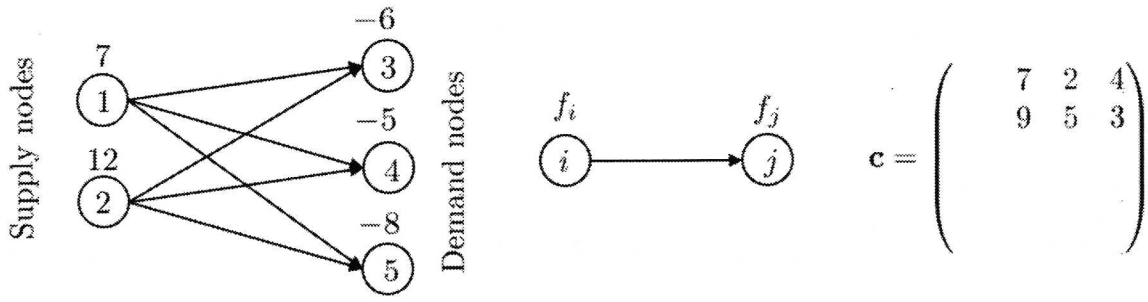


Figure 5.4: Example of the transportation problem

	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$		RHS
Objective function:	7	2	4	9	5	3		
Flow balance constraint node 1:	1	1	1				=	7
Flow balance constraint node 2:				1	1	1	=	12
Flow balance constraint node 3:	−1			−1			=	−6
Flow balance constraint node 4:		−1			−1		=	−5
Flow balance constraint node 5:			−1			−1	=	−8

Table 5.2: LP of the example of the transportation problem in tableau notation

A special application of the transportation problem is the assignment of outgoing student to international universities. The supply nodes depict a set of students, which have applied for an international semester at a partner university. Each student (and thus supply node) has a net flow of 1. Next, we have a set of demand nodes where each demand node depicts an international partner university. The net flow of the university is the number of incoming students, the university likes to welcome for one semester. The cost $c_{i,j}$ depicts the preference of student i for university j. Note that the costs have to be negative since the objective

function of the minimum cost flow problem minimizes the total cost. The larger the absolute value of $c_{i,j}$, the higher the preference of student i for university j is.

Figure 5.5 gives an example of the student assignment problem, Table 5.6 gives the associated LP in tableau notation, and Figure 5.6 gives the optimal solution.

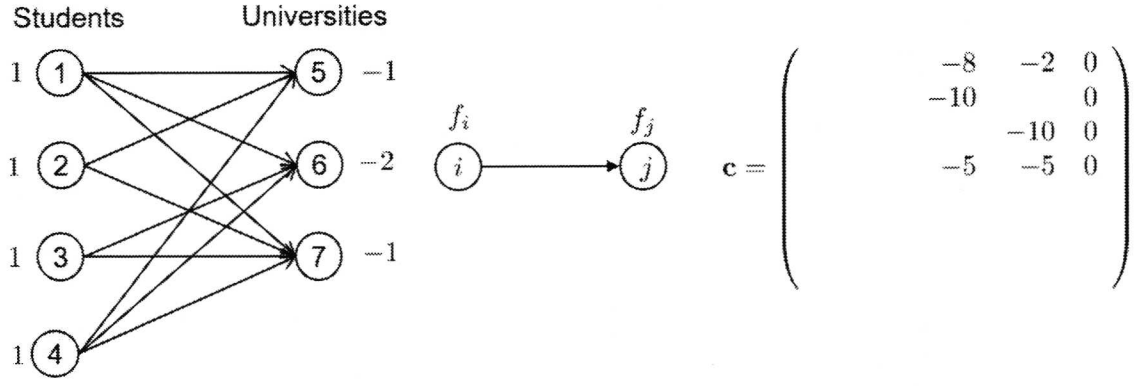


Figure 5.5: Example of the problem of assigning students to universities

	$x_{1,5}$	$x_{1,6}$	$x_{1,7}$	$x_{2,5}$	$x_{2,7}$	$x_{3,6}$	$x_{3,7}$	$x_{4,5}$	$x_{4,6}$	$x_{4,7}$		RHS
OF:	−8	−2	0	−10	0	−10	0	−5	−5	0		
FBC 1 :	1	1	1								=	1
FBC 2 :				1	1						=	1
FBC 3 :						1	1				=	1
FBC 4 :								1	1	1	=	1
FBC 5 :	−1			−1				−1			=	−1
FBC 6 :		−1				−1			−1		=	−2
FBC 7 :			−1		−1		−1			−1	=	−1

Table 5.3: LP of the student assignment problem in tableau notation

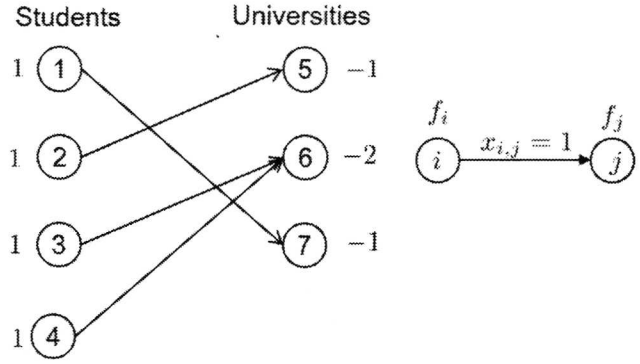


Figure 5.6: Optimal solution of the problem of assigning students to universities

5.2.2 The Assignment Problem

The assignment problem is a special transportation and thus minimum cost flow problem where the number of supply nodes is equal to the number of demand nodes, each supply node has a net flow of 1, and each demand node a net flow of -1. An arc (i, j) depicts if supply node i can be assigned to demand node j. A solution of the assignment problem connects each supply node i with a flow of 1 to exactly one demand node and vice versa. The arcs, for which $x_{i,j} = 1$ holds will form pairs between one supply node and one demand node. The optimal solution provides the set of pairs of nodes with minimal cost. An example of the assignment problem is the assignment of jobs to workers. The supply node depict the workers and the demand nodes depict the jobs. The cost $c_{i,j} > 0$ on arc (i, j) gives the time worker i needs to undertake job j. The assignment problem then assigns each worker to one job such that the total time to undertake all jobs is minimized.

5.2.3 Shortest Path Problems

Often we are interested in the shortest path from one start node $s \in V$ to a terminal node $t \in V$. The term "shortest" refers to the sum of the costs $c_{i,j}$ of the arcs we are traversing when moving from s to t. This might refer to distance, cost or travel time. We can model the shortest path problem as a special minimum cost flow problem by defining node s as single supply node with net flow $f_s = 1$ and node t as single demand node with net flow $f_t = -1$. All other nodes $i \in V \setminus \{s, t\}$ are defined as transshipment nodes with $f_i = 0$. For all arcs $(i, j) \in A$ we have $\lambda_{i,j} = 0$ and $\kappa_{i,j} = \infty$.

For the shortest path from $s = 1$ to $t = 5$ in the graph given in Figure 5.2, we obtain the LP in tableau notation given in Table 5.4.

	$x_{1,3}$	$x_{1,4}$	$x_{2,4}$	$x_{3,5}$	$x_{4,5}$		RHS
Objective function:	3	2	3	2	2		
Flow balance constraint node 1:	1	1				=	1
Flow balance constraint node 2:			1			=	0
Flow balance constraint node 3:	-1			1		=	0
Flow balance constraint node 4:		-1	-1		1	=	0
Flow balance constraint node 5:				-1	-1	=	-1

Table 5.4: LP of the example of the shortest path problem in tableau notation

If we want to determine the shortest path from a node $s \in V$ to all remaining nodes $V \setminus \{s\}$, then we have to assing a net flow of $f_s = |V| - 1$ to the node s and a net flow of $f_j = -1$ to all nodes $j \in V \setminus \{s\}$. Table 5.5 provides the LP in tableau notation for this case applied to the example given in Figure 5.2.

	$x_{1,3}$	$x_{1,4}$	$x_{2,4}$	$x_{3,5}$	$x_{4,5}$		RHS
Objective function:	3	2	3	2	2		
Flow balance constraint node 1:	1	1				=	4
Flow balance constraint node 2:			1			=	−1
Flow balance constraint node 3:	−1			1		=	−1
Flow balance constraint node 4:		−1	−1		1	=	−1
Flow balance constraint node 5:				−1	−1	=	−1

Table 5.5: Tableau notation of the LP for the example for deriving the shortest paths from node 1 to all other nodes

The Dijkstra Algorithm

The Dijkstra algorithm calculates the shortest paths from a single start node s to all other nodes $i \in V$ of the graph including s without using an LP. A prerequisite for the algorithm is a graph with non–negative arc weights $c_{i,j} \geq 0$ for all $(i,j) \in A$. The Dijkstra algorithm uses the two vectors d and p of size n where n denotes the number of nodes of the graph, i.e., $n = |V|$.

- $d[1,\ldots,n]$, where $d[i]$ gives the length of the shortest path from node s to node i.
- $p[1,\ldots,n]$, where $p[i]$ gives the immediate predecessor node of node i. $p[i]$ is the last node visited before reaching node i on the shortest path from node s to node i.

Furthermore, the Dijkstra algorithm uses a subset of nodes $M \subset V$. A node i is added to M once we have determined a first path from s to i. Node i is removed from M once we know that we have found the shortest path from s to i.

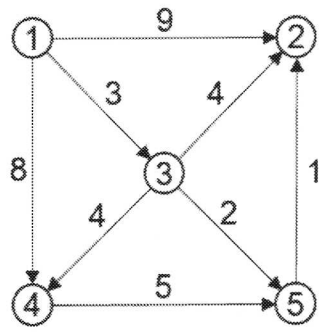


Figure 5.7: Example problem for the Dijkstra algorithm

Example: Consider the subgraph consisting of nodes $V = \{1,2,3\}$ and arcs $A = \{(1,2),(1,3),(3,2)\}$ of the directed graph given in Figure 5.7 where we have calculated the shortest distances from node $s = 1$ to all other nodes and the result is $d = (0,7,3)$ and

$p = (1, 3, 1)$. From d we can see that the shortest distance from node 1 to node 2 is $d[2] = 7$. The shortest path with this length can be determined as follows. From $p[2] = 3$ we know that the last node we have to visit before we reach node 2 is node 3. In other words, we will use arc $(3, 2)$ in order to reach node 2. Now we can look up the immediate predecessor node of node 3. This is $p[3] = 1$. That is, we will use arc $(1, 3)$ in order to reach node 3. Since the start node of this arc is node 1, we have found the start of the shortest path from node 1 to node 2. The arc sequence of the shortest path is $((1, 3), (3, 2))$ and the node sequence is $(1, 3, 2)$.

The general idea of the Dijkstra algorithm is as follows. In each iteration, the node $i \in M$ with the smallest $d[i]$-value is selected. By this we have found the shortest path from s to i. Node i is then removed from M and each direct successor j of i (i.e., each node for which an arc leads from i to j) is added to M if the distance via i, i.e., $d[i] + c_{i,j}$, is smaller than the current distance $d[j]$ and i is not already in M. At the beginning of the algorithm, for all nodes i except the start node s the distance is initialized to $d[i] = \infty$. The pseudocode of the Dijkstra algorithm is given in Figure 5.8.

Dijkstra Algorithm

Initialization
$M = \{s\}$
$d[s] = 0$, $d[i] = \infty$ for all $i \in V \setminus \{s\}$
$p[s] = s$, $p[i]$ is empty for all $i \in V \setminus \{s\}$

Iteration
if M is not empty **then**
    Choose node $h \in M$ with smallest $d[h]$
    Delete h from M
    for all nodes i with $(h, i) \in A$ **do**
        if $d[i] > d[h] + c_{h,i}$ **then**
            $d[i] = d[h] + c_{h,i}$
            $p[i] = h$
            if i is not in M **then**
                Insert i into M
            end if
        end if
    end for
end if

Figure 5.8: Pseudocode of the Dijkstra algorithm

Let us apply the Dijkstra algorithm to the weighted digraph given in Figure 5.7 with $s = 1$. The initialization gives

Initialization:

i	1	2	3	4	5
$d[i]$	0	∞	∞	∞	∞
$p[i]$	1				
$M = \{1\}$					

Iteration 1: In the first iteration, node 1 is selected ($h = 1$) and afterwards deleted from set M (lines $4-6$ of the pseudocode). The selected node is marked with a box in the table. Afterwards, we check for each node, i for which an arc leads from node $h = 1$ to i if we can reduce the current distance $d[i]$ if we use the arc (h, i) to reach node i. This is for nodes 2,3, and 4 the case and accordingly, their distance d and their immediate predecessor p are updated. Furthermore, since none of the nodes 2,3, and 4 is in M, these nodes are added to M. The state of the algorithm after the first iteration is thus as follows:

i	$\boxed{1}$	2	3	4	5
$d[i]$	0	9	3	8	∞
$p[i]$	1	1	1	1	
$M = \{2, 3, 4\}$					

Iteration 2: Since M is not empty, the node i with the smallest $d[i]$ is selected. This is node 3 and hence we have $h = 3$. We now know that we have found the shortest path from node $s = 1$ to node 3 and accordingly, we delete node 3 from M. From node 3 arcs lead to nodes 2, 4, and 5. Hence, we have to check for each of these nodes $i \in \{2, 4, 5\}$ if we can reduce the current distance $d[i]$ by choosing a path, where the last arc is (h, i). We start with node $i = 2$. The current distance is $d[2] = 9$ but when using a path via node $h = 3$ we obtain a distance of $d[3] = 3$ plus $c_{3,2} = 4$, which is 7 and thus shorter than the current distance. Hence, we update $d[2] = 7$ and $p[2] = 3$. The update of $p[2] = 3$ states that we will use the arc $(3, 2)$ as last arc on the path from node $s = 1$ to node 2. We now turn to node $i = 4$. The distance for this node can also be updated to $d[4] = 7$ and the new last arc is $(3, 4)$, which states that node 3 is the last node traversed on the path before we reach node 4. Finally, node 5 is updated from $d[5] = \infty$ to $d[5] = 5$. Since node 5 has not been in M it is added to M. The following table gives the state of the algorithm after the second iteration:

i	1	2	$\boxed{3}$	4	5
$d[i]$	0	7	3	7	5
$p[i]$	1	3	1	3	3
$M = \{2, 4, 5\}$					

Iteration 3: M is not empty and we select node 5, which has the smallest $d[i]$-value. Node 5 is removed afterwards. We check if we can reduce the distance for node 2, because arc $(5, 2)$ leads into node 2. The distance for node 2 can be further reduced and we update $d[2] = 6$ and $p[2] = 5$. Node 2 is already in M and stays there. The state of the algorithm after iteration 3 is:

i	1	2	3	4	$\boxed{5}$
$d[i]$	0	6	3	7	5
$p[i]$	1	5	1	3	3
$M = \{2, 4\}$					

Iteration 4: M is not empty and we select node 2, which has the smallest $d[i]$-value. Node 2 is removed afterwards. Node 2 has no outgoing arcs and hence we cannot check for any reductions of the distance. The state of the algorithm after iteration 4 is:

i	1	$\boxed{2}$	3	4	5
$d[i]$	0	6	3	7	5
$p[i]$	1	5	1	3	3
$M = \{4\}$					

Iteration 5: M is not empty and we select node 4, which has the smallest $d[i]$-value. Node 4 is removed afterwards. Node 4 has an outgoing arc to node 5. However, the current distance $d[5]$ cannot be reduced. The state of the algorithm after iteration 5 is:

i	1	2	3	$\boxed{4}$	5
$d[i]$	0	6	3	7	5
$p[i]$	1	5	1	3	3
$M = \{\}$					

Since M is empty, the Dijkstra algorithm stops with the optimal solution given in the state of the algorithm after iteration 5. The shortest distance for node $s = 1$ to node 2 is $d[2] = 5$ and the associated path is determined recursively as follows: The immediate predecessor node of node 2 is $p[2] = 5$. The immediate predecessor node of node 5 is $p[5] = 3$. The immediate predecessor node of node 3 is $p[3] = 1$. Hence, the optimal node sequence from node 1 to node 2 is $(1, 3, 5, 2)$ and the associated arc sequence is $((1, 3), (3, 5), (5, 2))$.

Note that the Dijstra algorithm selects and deletes exactly one node from M in each iteration and stops once M is empty. Hence, the number of iterations equals the number of nodes of the graph. If we are not interested in the shortest distance from node s to all nodes in the graph V but to a subset of nodes V', we can stop the algorithm as soon as every node of V' has been selected and removed from M. For example, for $V' = \{1, 3, 5\}$, we can stop the Dijkstra algorithm after iteration 3.

Observation: Let $v(i)$ be the node removed in iteration $i = 1, \ldots, |V|$. It holds $d[v(i)] \leq d[v(i+1)]$ for $i = 1, \ldots, |V| - 1$, i.e., the distance of the picked and removed nodes is non-decreasing in the order of the selection. For our example we have $d[1] = 0 < d[3] = 3 < d[5] = 5 < d[2] = 6 < d[4] = 7$.

The Floyd–Warshall Algorithm

The Floyd–Warshall algorithm (see Ahuja et al. [1] pp. 147) calculates shortest distances between all node pairs of the graph. In contrast to the Dijkstra algorithm there are no prerequisites for applying the algorithm, i.e., arc weights can be negative and the graph can have cycles. The Floyd–Warshall algorithm uses the following notation:

- Distance matrix d where $d[i, j]$ contains the shortest distance from node $i \in V$ to node $j \in V$.
- Predecessor matrix p where $p[i, j]$ denotes the immediate predecessor node of node $j \in V$ on the shortest path from node $i \in V$ to node $j \in V$.

Note that the matrices d and p are straight forward extension of the vectors d and p of the Dijkstra algorithm. Figure 5.9 provides the pseudocode of the Floyd–Warshall algorithm.

Floyd–Warshall Algorithm

Initialization
1: **for all** nodes $i \in V$ **do**
2: Set $d_{i,i} = 0$ and $p_{i,i} = i$
3: **for all** nodes $j \in V \setminus \{i\}$ **do**
4: **if** $(i, j) \in A$ **then**
5: Set $d_{i,j} = c_{i,j}$ and $p_{i,j} = i$
6: **else**
7: Set $d_{i,j} = \infty$ and $p_{i,j} =$ ' $-$ '
8: **end if**
9: **end for**
10: **end for**

Iteration
11: **for all** nodes $v \in V$ **do**
12: **for all** nodes $i \in V$ **do**
13: **for all** nodes $j \in V$ **do**
14: **if** $d_{i,j} > d_{i,v} + d_{v,j}$ **then**
15: Set $d_{i,j} = d_{i,v} + d_{v,j}$
16: Set $p_{i,j} = p_{v,j}$
17: **end if**
18: **end for**
19: **end for**
20: **end for**

Figure 5.9: Pseudocode of the Floyd–Warshall algorithm

We apply the Floyd–Warshall algorithm for the weighted digraph given in Figure 5.7. Table 5.6 gives the state of the Floyd-Warshall algorithm after the initialization. In row i and column j the table provides the values of $d[i,j]$ and $p[i,j]$ separated by a semicolon.

	1	2	3	4	5
1	0; 1	9; 1	3; 1	8; 1	∞; –
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	4; 3	0; 3	4; 3	2; 3
4	∞; –	∞; –	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

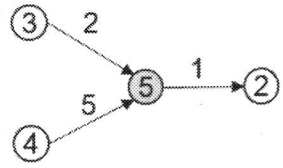
Table 5.6: Floyd–Warshall algorithm: d– und p–matrix after the initialization

Initialization: After the initialization we obtain the distance matrix d and the predecessor matrix p given in Table 5.6. We can observe three different types of entries in the matrix. First, in the diagonals, we have the entries $d[i,i] = 0$ and $p[i,i] = i$ for $i = 1, \ldots, 5$. Second, for each arc $(i,j) \in A$ we have an entry $d[i,j] = c_{i,j}$ and $p[i,j] = i$. In our example we have $|A| = 8$ arcs and thus 8 of these entries. Third, for all other entries we have $d[i,j] = \infty$ and $p[i,j] = -$. This indicates that there is no direct arc connection between node i and node j.

We now consider the iterations. The first for-loop in line 11 of the algorithm shows that we have $|V|$ iterations. In each iteration one node will be selected. The nodes can be picked in any order. For our example, we select the nodes in the order of descending numbers, i.e., 5, 4, 3, 2, and 1. Having selected a node v it is then checked if for all combinations of the other nodes a path via v can shorten the distance between the nodes. This is done in the lines 12 - 19. In the following we structure the presentation along the node iteration on the highest level given by line 11.

Iteration 1 : Node 5 is selected, i.e., $v = 5$. In Figure 5.10 we provide the following information: On the left the state of the $p-$ and d-matrix at the start of the iteration are given. Grey highlighted are the row and the column of the activity v, which has been selected in this iteration. In the middle of the figure, a graph is given, which depicts all currently existing distances $d[i,v] < \infty$ into node v and distances $d[v,j] < \infty$ out of node v. On the right of the figure, a table gives all combinations of a node i of this graph leading into v and from node v into a node j of the graph. Each combination is given in one row. In the second column the distance between i and j at the beginning of the iteration is given and in the third column the distance is given, which can be obtained by traversing through node v. In case, traversing through node v shortens the distance between i and j updating takes place. For iteration 1, the updated d- and p-matrices are given in Table 5.7 where the cells $[i,j]$ with changed d- and p-entries are highlighted in grey.

	1	2	3	4	5
1	0; 1	9; 1	3; 1	8; 1	∞; –
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	4; 3	0; 3	4; 3	2; 3
4	∞; –	∞; –	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5



(i,j)	$d[i,j]$	$d[i,5]+d[5,j]$
(3, 2)	4	3
(4, 2)	∞	6

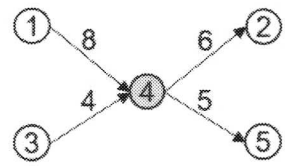
Figure 5.10: Iteration 1 of the Floyd–Warshall algorithm with $v = 5$

	1	2	3	4	5
1	0; 1	9; 1	3; 1	8; 1	∞; –
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Table 5.7: Floyd–Warshall algorithm: d– und p–matrix after iteration 1

Iteration 2: Node 4 is selected, i.e., $v = 4$. Updating of distances is done for the path from node 1 to node 5. Table 5.8 gives the d- and p-matrix after the updating. The predecessor information is updated as follows: $p[1,5] = p[4,5] = 4$.

	1	2	3	4	5
1	0; 1	9; 1	3; 1	8; 1	∞; –
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5



(i,j)	$d[i,j]$	$d[i,5]+d[5,j]$
(1, 2)	9	$8+6=14$
(1, 5)	∞	$8+5=13$
(3, 2)	3	$4+6=10$
(3, 5)	2	$4+5=9$

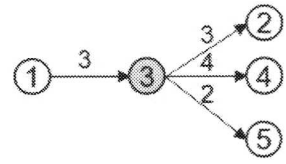
Figure 5.11: Iteration 2 of the Floyd–Warshall algorithm with $v = 4$

	1	2	3	4	5
1	0; 1	9; 1	3; 1	8; 1	13; 4
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Table 5.8: Floyd–Warshall algorithm: d– und p–matrix after iteration 2

Iteration 3: Node 3 is selected, i.e. $v = 3$. Updating of distances is done for the paths from node 1 to node 2, from node 1 to node 4, and from node 1 to node 5. Table 5.9 gives the d- and p-matrix after the updating. The predecessor information are updated as follows: $p[1,2] = p[3,2] = 5$, $p[1,4] = p[3,4] = 3$, and $p[1,5] = p[3,5] = 3$.

	1	2	3	4	5
1	0; 1	9; 1	3; 1	8; 1	13; 4
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5



(i,j)	$d[i,j]$	$d[i,5]+d[5,j]$
$(1,2)$	9	$3+3=6$
$(1,4)$	8	$3+4=7$
$(1,5)$	13	$4+2=5$

Figure 5.12: Iteration 3 of the Floyd–Warshall algorithm with $v = 3$

	1	2	3	4	5
1	0; 1	6; 5	3; 1	7; 3	5; 3
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Table 5.9: Floyd–Warshall algorithm: d– und p–matrix after iteration 3

Iteration 4: Node 2 is selected, i.e. $v = 2$. Since there is not node, which can be reached from node 2, no updating takes place. Figure 5.13 gives the associated information of the d- and p-matrix.

	1	2	3	4	5
1	0; 1	6; 5	3; 1	7; 3	5; 3
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

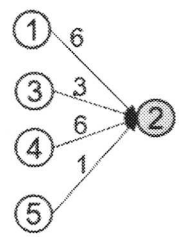


Figure 5.13: Iteration 4 of the Floyd–Warshall algorithm with $v = 2$

Iteration 5: Node 1 is selected, i.e. $v = 1$. Since there is not node which can reach node 1, no updating takes place and the final solution is obtained. It is given in the d- and p-matrix of Figure 5.14.

	1	2	3	4	5
1	0; 1	6; 5	3; 1	7; 3	5; 3
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

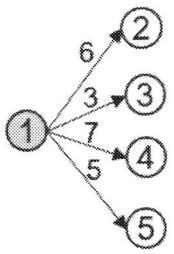


Figure 5.14: Iteration 5 of the Floyd–Warshall algorithm with $v = 1$

5.3 The Minimum Spanning Tree Problem

The minimum spanning tree problem employs an undirected graph with edge weights $G = [V, A, c]$. We want to find a subgraph of G' of G which contains all nodes V but where the set of edges is $A' \subseteq A$. A' has to be such that all nodes are directly or indirectly connected to each other, which implies that none of the nodes is isolated. The objective is to find A' and thus $G' = [V, A', c]$ such that the sum of the edge weights is minimized. We consider the following example (see Domschke et al. [4] pp. 85):

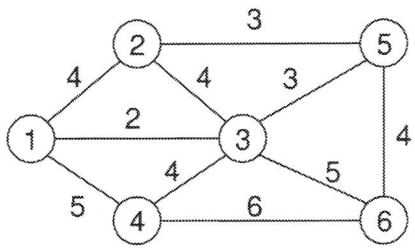


Figure 5.15: Graph for calculating a minimum spanning tree

Obviously, all nodes are directly or indirectly connected to each other and there are no isolated edges. However, we don't need all of the 10 edges with a total cost of $\sum_{[i,j]\in A} c_{i,j} = 40$. For example, we can delete edge $[2, 5]$ and thus reduce the cost by $c_{2,5} = 3$ to 37 without having a disconnected node. We will now introduce the Kruskal algorithm for obtaining the minimum spanning tree, that is a undirected graph where there are no disconnected nodes and where the sum of the cost of the edges is minimum. Figure 5.16 provides the pseudocode of the Kruskal algorithm.

In the initialization we sort the edges $[i, j] \in A$ according to non–decreasing costs $c_{i,j}$, i.e., the edge with the lowest cost first, with the second lowest second etc. In case of edges with identical cost we sort first according to i and then to j with the smallest number always first. Applying the Kruskal algorithm to the example given in Figure 5.15 we obtain the list $L = \langle [1, 3], [2, 5], [3, 5], [1, 2], [2, 3], [4, 3], [5, 6], [1, 4], [3, 6], [4, 6]]$. Afterwards, we perform the

Kruskal Algorithm

Initialization

1: Sort the edges $[i, j] \in A$ according to non–decreasing costs $c_{i,j}$
2: The sorted edges are put into list $L = \langle]$
3: Set $A' := \emptyset$

Iteration

4: **while** $L \neq \emptyset$ **do**
5: Select the next edge $[i, j]$ from the list L
6: **if** graph $G' = [V, A' \cup [i, j]]$ does not contain cycles **then**
7: Set $A' := A' \cup [i, j]$
8: **end if**
9: Delete edge $[i, j]$ from the list L
10: **end while**

Figure 5.16: Pseudocode of the Kruskal algorithm

seven iterations given in Table 5.10 and Figure 5.17 until we have a spanning tree where all nodes are connected. The remaining three iterations would always create a cycle in Step 2 and thus there are no further edges inserted.

Iteration	$[i, j]$	A'	$\sum_{[i,j] \in A'} c_{i,j}$
Initialization	–	–	0
1	[1,3]	$\{[1, 3]\}$	2
2	[2,5]	$\{[1, 3], [2, 5]\}$	5
3	[3,5]	$\{[1, 3], [2, 5], [3, 5]\}$	8
4	[1,2]	$\{[1, 3], [2, 5], [3, 5]\}$	8
5	[2,3]	$\{[1, 3], [2, 5], [3, 5]\}$	8
6	[4,3]	$\{[1, 3], [2, 5], [3, 5], [4, 3]\}$	12
7	[5,6]	$\{[1, 3], [2, 5], [3, 5], [4, 3], [5, 6]\}$	16

Table 5.10: Protocol of the Kruskal algorithm

Note that a minimum spanning tree for a graph with $n = |V|$ nodes has $n - 1$ edges. Hence, as soon as the n-th edge has been inserted to graph $G' = [V, A', c]$, the algorithm can be terminated. The Kruskal algorithm belongs to the class of "greedy algorithms" because the next edge which is evaluated for being added to the graph has always the minimum cost of all not yet considered edges. For the minimum spanning tree problem, the Kruskal algorithm yields the optimal solution. However, for many other problems such as the Traveling Salesman Problem greedy algorithms don't have an optimality guarantee.

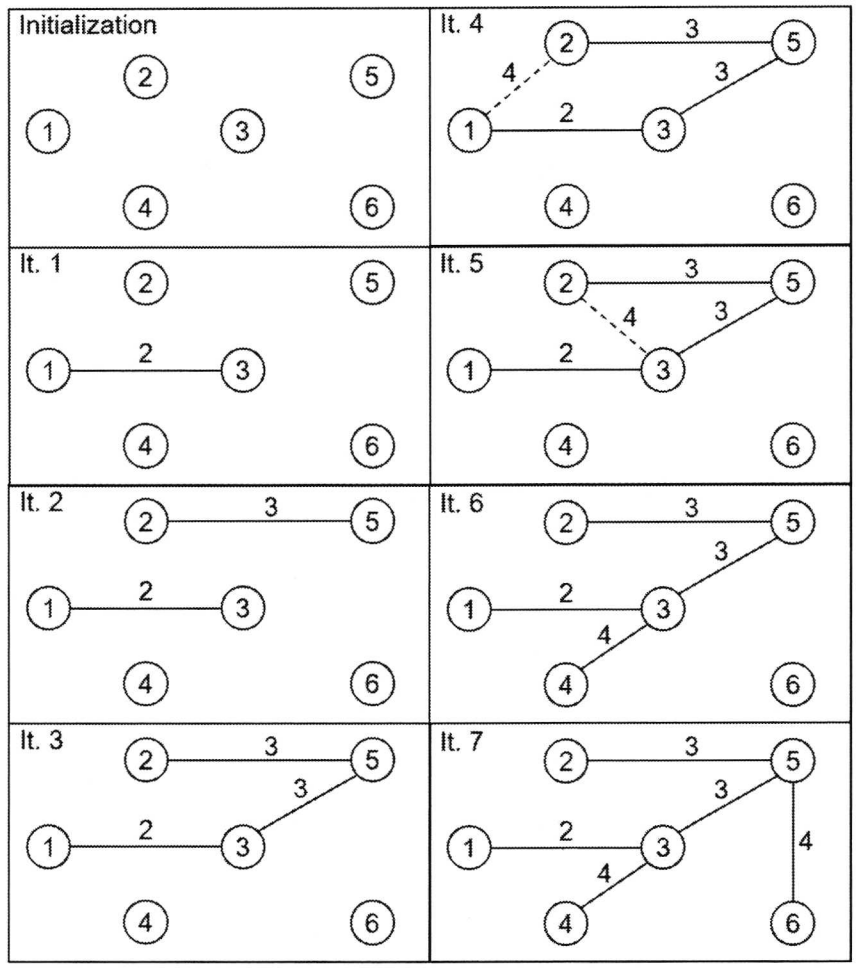


Figure 5.17: Graphical representation of the Kruskal algorithmu

5.4 The Maximum Flow Problem

Example: An oil company wants to ship the maximum amount of oil from node 1 to node 5. Figure 5.18 shows the pipeline network $G = (V, A, \kappa)$ where arcs denote pipes and the weight of the arcs κ denote the capacity of the pipe.

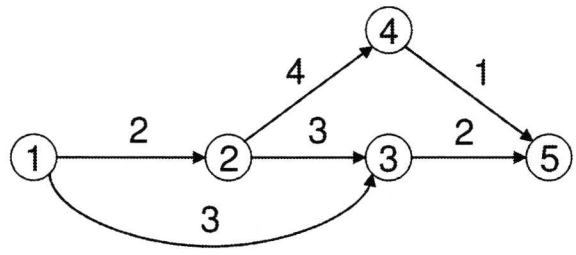


Figure 5.18: Pipeline network

The problem can be modeled as linear program by using the following variable:

$$x_{i,j} = \text{flow between node } i \text{ and node } j$$

We add an arc from node 1 to 5 with capacity $\kappa_{1,5} = \infty$ to graph $G = (V, A, \kappa)$ (see Figure 5.19).

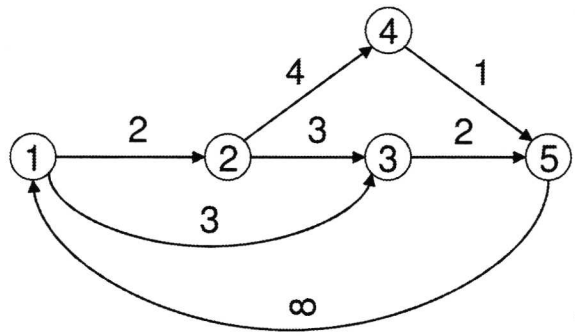


Figure 5.19: Pipeline network modeled as maximum flow problem

We now maximize the flow in arc (5,1) by taking into account the flow balance constraints for each node as well as the capacities of the arcs. The flow balance constraint of a node states that the inflow into the node has to be equal to the outflow of the node.

The resulting linear program reads as follows:

$$\begin{aligned}
\text{Max } z = x_{5,1} & \\
\text{subject to} & \\
x_{1,2} &\leq 2 \\
x_{1,3} &\leq 3 \\
x_{2,3} &\leq 3 \\
x_{2,4} &\leq 4 \\
x_{3,5} &\leq 2 \\
x_{4,5} &\leq 1 \\
x_{5,1} - x_{1,2} - x_{1,3} &= 0 \\
x_{1,2} - x_{2,3} - x_{2,4} &= 0 \\
x_{1,3} + x_{2,3} - x_{3,5} &= 0 \\
x_{2,4} - x_{4,5} &= 0 \\
x_{3,5} + x_{4,5} - x_{5,1} &= 0 \\
x_{1,2}, x_{1,3}, x_{2,3}, x_{2,4}, x_{3,5}, x_{4,5}, x_{5,1} &\geq 0
\end{aligned}$$

The first six constraints give the capacity constraint for each of the six arcs. The following five constraints are the flow balance constraint for each node. The final constraint defines the continuous decision variables. Soving this linear program to optimality yields $z = 3$, $x_{1,2} = 2$, $x_{1,3} = 1$, $x_{2,3} = 1$, $x_{2,4} = 1$, $x_{3,5} = 2$, $x_{4,5} = 1$ and $x_{5,1} = 3$. This solution is given in Figure 5.20 where for each arc the magnitude of the flow is represented by the thickness of the arc. For each arc $x_{i,j}, \kappa_{i,j}$ denotes the flow and the capacity of the arc.

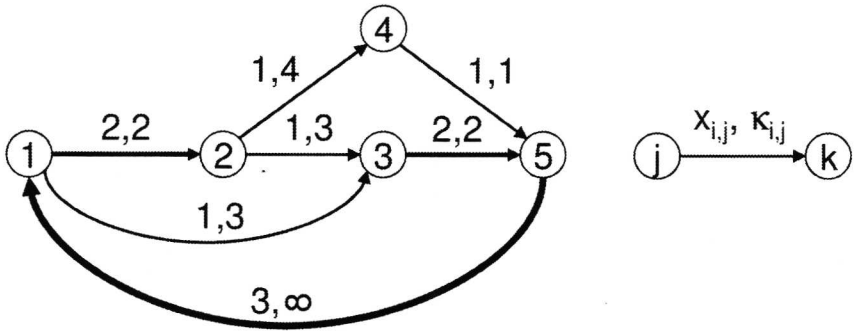


Figure 5.20: Optimal solution of the maximum flow problem

General LP–formulation. For a general LP–formulation of the maximum flow problem we denote the graph $G = (V, A, \lambda, \kappa)$ with minimum flow $\lambda_{i,j}$ and maximum flow $\kappa_{i,j}$ for each arc (i, j). The node set V contains all nodes, including one one source q and one sink s. We want to determine the maximum flow from the source q to the sink s $(q, s \in V)$. The graph is amended by arc (s, q) with minimum flow $\lambda_{s,q} = -\infty$ and maximum flow $\lambda_{s,q} = \infty$. The LP reads as follows:

$$\text{Max } z = x_{s,q} \tag{5.6}$$

subject to

$$\sum_{(h,j)\in A} x_{h,j} - \sum_{(j,k)\in A} x_{j,k} = 0 \quad \text{für alle } j \in V \tag{5.7}$$

$$x_{i,j} \geq \lambda_{i,j} \quad \text{für alle } (i,j) \in A \tag{5.8}$$

$$x_{i,j} \leq \kappa_{i,j} \quad \text{für alle } (i,j) \in A \tag{5.9}$$

The Ford–Fulkerson Algorithm

The Ford–Fulkerson algorithm solves the maximum flow problem without using the Simplex. We introduce the algorithm with the example given in Figure 5.21. The notation $[\lambda_{i,j}, \kappa_{i,j}]$ given for each arc denotes the minimum flow of the arc $\lambda_{i,j}$ and the maximum flow of the arc $\kappa_{i,j}$. Furthermore, q denotes the start node and s the final node, i.e., we want to maximize the flow from node q to node s. We also call the start node as "source" and the final node as "sink".

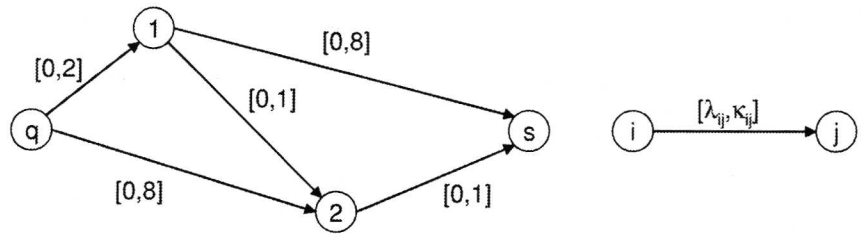


Figure 5.21: Example for the Ford–Fulkerson algorithm

Initialization The algorithm starts with a feasible solution. For the example problem we can derive a feasible solution by setting the flow x of each arc to zero. Figure 5.22 provides the flow for each arc using the notation $[\lambda_{i,j}, \kappa_{i,j}]\, x_{i,j}$. Table 5.11 provides information on the start solution. Each row of Table 5.11 contains information for one node. In the second column, for the flow received from other nodes is given. In the last but one row the increase in the flow from q to s is given. The last row contains the total flow from q to s.

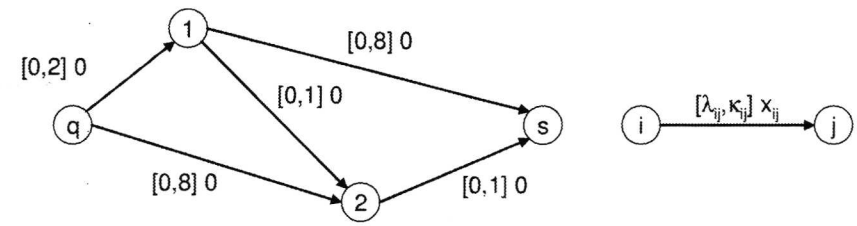


Figure 5.22: Feasible start solution

Node	Start solution
q	$(+, 0)$
1	$(q^+, 0)$
2	$(q^+, 0), (1^+, 0)$
s	$(1^+, 0), (2^+, 0)$
Δ Flow	0
$\sum$ Flows	0

Table 5.11: Feasible start solution

Iteration 1 Starting with a feasible solution, the algorithm seeks to improve the existing solution by adding a flow. This is done by searching for a path from q to s where the additional flow can be accommodated. Searching for the additional flow it is always started at the source q. The amount which is available at q is infinite which is indicated by the entry $(+, \infty)$ in the q–row of Table 5.12. Starting at q we search for an arc where we can ship an additional flow in the direction of the sink s. Let us select arc $(q, 1)$. Taking into account the current flow of zero we can add 2 more flow units to this arc. This is stated by entry $(q^+, 2)$

in the row of node 1 in Table 5.12. More precisely, the entry states that 2 additional flow units are transported from source (q) in the direction of the arc (+). From node 1 we can ship $\kappa_{1,2} - x_{1,2} = 1$ additional units to node 2. From node 2 we ship $\kappa_{2,s} - x_{2,s} = 1 - 0 = 1$ additional units to the sink. For each node on the path it has to hold that the additional flow out of the node has to be smaller than or equal to the additional flow into the node. Once we have reached the final node, the additional flow on the entire path is set to the additional flow on the arc leading into s. After the first iteration, we obtain an additional flow of 1. Table 5.12 shows the details of iteration 1 and Figure 5.23 gives the resulting flow.

Node	Start solution	Iteration 1
q	$(+, 0)$	$(+, \infty)$
1	$(q^+, 0)$	$(q^+, 2)$
2	$(q^+, 0), (1^+, 0)$	$(1^+, 1)$
s	$(1^+, 0), (2^+, 0)$	$(2^+, 1)$
Δ Flow	0	1
$\sum$ Flows	0	1

Table 5.12: Solution after Iteration 1

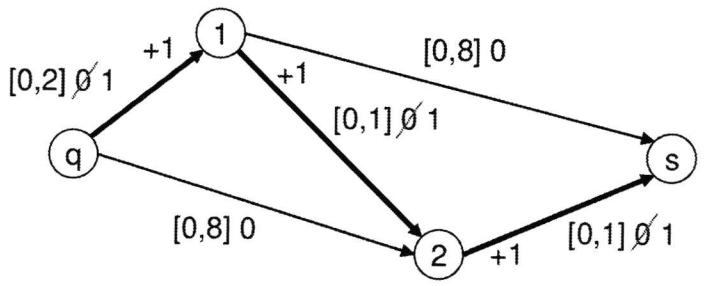


Figure 5.23: Solution after Iteration 1

Iteration 2 The result of Iteration 2 is given in Table 5.13 and Figure 5.24.

Node	Start solution	Iteration 1	2
q	$(+, 0)$	$(+, \infty)$	$(+, \infty)$
1	$(q^+, 0)$	$(q^+, 2)$	$(q^+, 1)$
2	$(q^+, 0), (1^+, 0)$	$(1^+, 1)$	
s	$(1^+, 0), (2^+, 0)$	$(2^+, 1)$	$(1^+, 1)$
Δ Flow	0	1	1
$\sum$ Flows	0	1	2

Table 5.13: Solution after Iteration 2

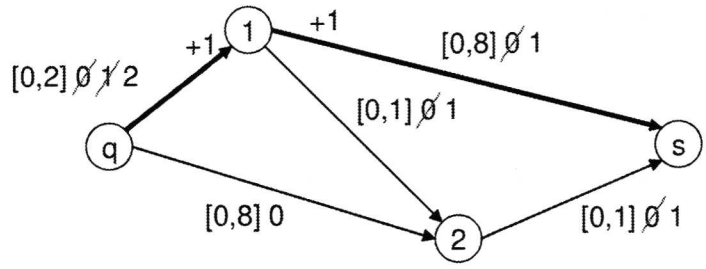


Figure 5.24: Solution after Iteration 2

Iteration 3 We try to increase the flow further. Starting at q we can ship 8 additional flow units to node 2 which is depicted by entry $(q^+, 8)$ in the row of node 2. From node 2 we cannot send any more flows directly to the sink s since the maximum capacity $(2, s)$ is used. However, we can ship additional flow in the opposite direction of an arc (i, j) if $x_{i,j} > \lambda_{i,j}$ holds. The maximum additional flow which can be shipped in the opposite direction of the arc (i, j) is $x_{i,j} - \lambda_{i,j}$. For arc $(1, 2)$ we have $x_{1,2} - \lambda_{1,2} = 1 - 0 = 1$. In the table we depict a flow against the direction of the arc $(1, 2)$ as $(2^-, 1)$ in the row of node 1. The "$-$" in the superscript of 2 indicates that the additional flow is in the opposite direction of the arc. After we have shipped an additional flow into node 1 we can ship one additional flow unit on arc $(1, s)$ into sink s. This is denoted by the entry $(1^+, 1)$ in the row of node s. Table 5.14 and Figure 5.25 provide the result of the third iteration which yields a total flow of 3.

	Start	Iteration		
Node	solution	1	2	3
q	$(+, 0)$	$(+, \infty)$	$(+, \infty)$	$(+, \infty)$
1	$(q^+, 0)$	$(q^+, 2)$	$(q^+, 1)$	$(2^-, 1)$
2	$(q^+, 0), (1^+, 0)$	$(1^+, 1)$		$(q^+, 8)$
s	$(1^+, 0), (2^+, 0)$	$(2^+, 1)$	$(1^+, 1)$	$(1^+, 1)$
Δ Flow	0	1	1	1
$\sum$ Flows	0	1	2	3

Table 5.14: Solution after Iteration 3

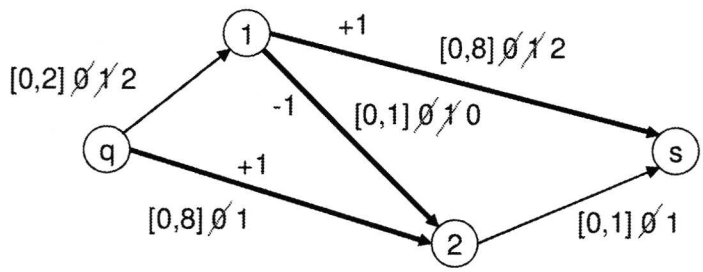


Figure 5.25: Solution after Iteration 3

Iteration 4 We try to further improve the flow. Afer sending 7 addtional flow units into node 2 we cannot ship any additional flow units out of node 2. Neither in the direction of the arc $(2, s)$ nor in the opposite direction of arc $(1, 2)$. As soon as we cannot further increase the flow, the algorithm stops. Table 5.15 gives the protocol of the last iteration. The solution obtained is optimal. In order to prove this we have to introduce the definition of a cut.

	Start	Iteration			
Node	solution	1	2	3	4
q	$(+, 0)$	$(+, \infty)$	$(+, \infty)$	$(+, \infty)$	$(+, \infty)$
1	$(q^+, 0)$	$(q^+, 2)$	$(q^+, 1)$	$(2^-, 1)$	
2	$(q^+, 0), (1^+, 0)$	$(1^+, 1)$		$(q^+, 8)$	$(q^+, 7)$
s	$(1^+, 0), (2^+, 0)$	$(2^+, 1)$	$(1^+, 1)$	$(1^+, 1)$	
Δ Flow	0	1	1	1	–
$\sum$ Flows	0	1	2	3	–

Table 5.15: Stop of the algorithm in Iteration 4

Definition 25 *A* **cut** *separates the node set V in two disjoint node sets V^q and V^s. V^q has to contain the source q and V^s has to contain the sink s. The intersection of V^q and V^s has to be empty (i.e., $V^q \cap V^s = \emptyset$) and the union of V^q and V^s has to be V (i.e., $V^q \cup V^s = V$).*

Definition 26 *The* **capacity K of a cut** *is the sum of the $\kappa_{i,j}$ of the arcs (i, j) with $i \in V^q$ and $j \in V^s$, i.e. the arcs leading from set V^q into set V^s minus the sum of the $\lambda_{i,j}$ of the arcs (i, j) with $i \in V^s$ and $j \in V^q$, i.e., the arcs leading from the set V^s into the set V^q.*

Figure 5.26 shows all possible cuts and the capaciy of the cuts for the example problem. The nodes in V^q are encircled.

The problem to find a cut of the graph with minmal capacity is the dual of the maximum flow problem. Due to the duality properties the following holds:

- The objective function value of a feasible but not optimal solution of the maximum flow problem is strictly smaller than the objective function value of a feasible but not optimal solution of the minimum cut problem. This follows immediately from the weak duality property given in Property 4 of Section 3.6.4.
- The objective function value of the optimal solution of the maximum flow problem equals the objective function value of the optimal solution of the minimum cut problem. This follows immediately from the strong duality property given in Property 5 of Section 3.6.4.

To illustrate the weak duality property let us consider the solution of the maximum flow problem which has been obtained after the first iteration. Obviously, this solution is feasible

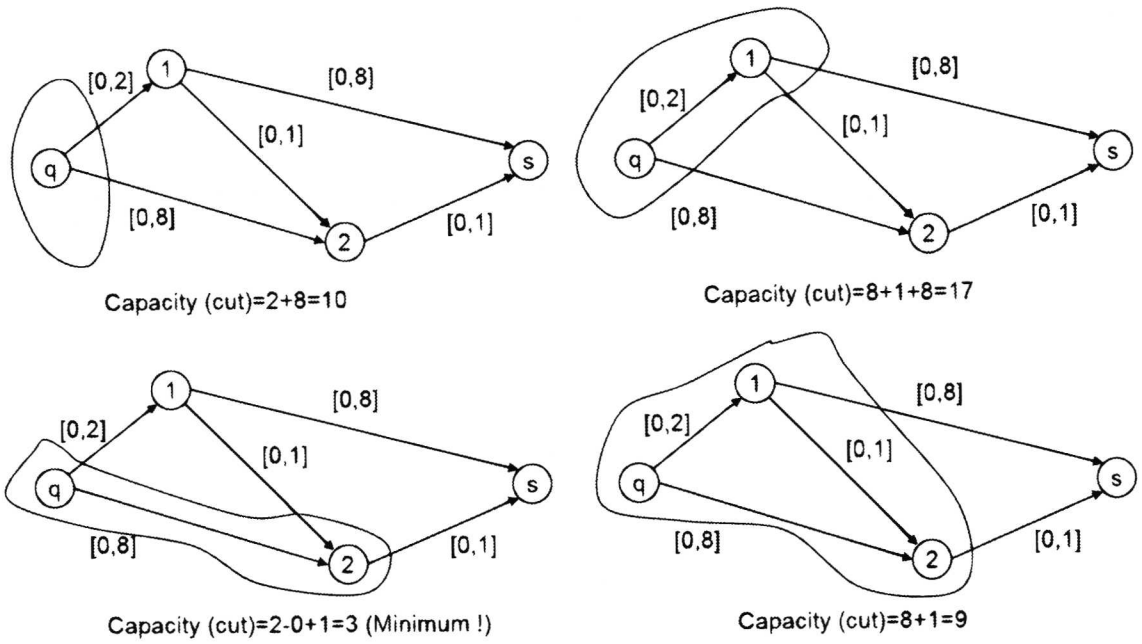


Figure 5.26: All cuts of the example problem

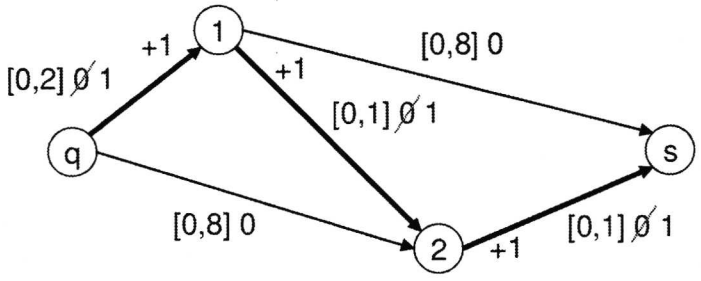


Figure 5.27: Solution of the maximum flow problem after the first iteration

but not optimal. The objective function value is 1 while the objective function values of the solutions of the minimum cut problem are 9, 3, 17 and 10.

In order to illustrate the strong duality property we consider the optimal solutions of the maximum flow and the minimum cut problem. The objective function value of the optimal solution of the maximum flow problem is 3 and the smallest objective function value of all solutions of the minimum cut problem is also 3.

We now show that when stopping the Ford–Fulkerson algorithm has found an optimal solution. Remember that the Ford–Fulkerson algorithm stops when he fails to increase the flow from q to s. Let all nodes which have been visited when increasing the flow be in set V^q and let all other nodes be in V^s. Now we can observe the following:

- For each arc (i, j) which leads from a node $i \in V^q$ to a node $j \in V^s$ it holds that

$x_{i,j} = \kappa_{i,j}$ because otherwise we could have increased the flow.

- For each arc (i, j) which leads from a node $i \in V^s$ to a node $j \in V^q$ it holds that $x_{i,j} = \lambda_{i,j}$ because otherwise we could have generated an additional flow in the opposite direction of the arc.

If we cut all arcs (i, j) with $i \in V^q$ and $j \in V^s$ then the capacity of the cut equals the flow which has been obtained in the last successful iteration before the Ford–Fulkerson algorithm stops. Hence, we have found the maximal flow and the minimum cut and are thus optimal.

Figure 5.28 shows the graph of the example problem when the Ford–Fulkerson algorithm stops. As can be seen, the optimality conditions hold because $x_{q,1} = \kappa_{q,1} = 2$, $x_{1,2} = \lambda_{1,2} = 0$ and $x_{2,s} = \kappa_{2,s} = 1$.

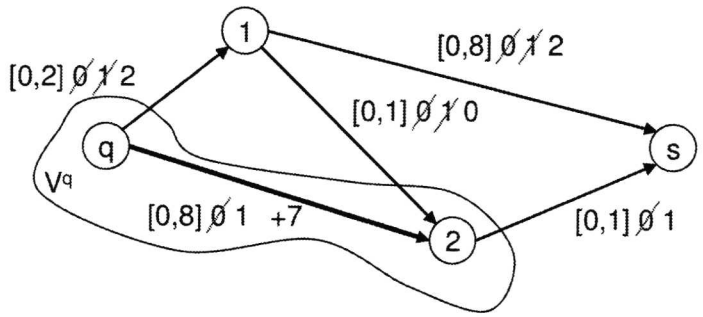


Figure 5.28: State of the Ford–Fulkerson algorithm when stopping in Iteration 4

Chapter 6

Dynamic Programming

Reference: Bradley et al. [3] Chapter 11, Winston [11] Chapter 20. Domschke et al. [4] Chapter 7, Hillier and Lieberman [7] Chapter 10.

6.1 An Introductory Example: The Commuter's Problem

Let us illustrate dynamic programming with the commuter's problem (see Bradely et al. [3] pp. 453). The graph given in Figure 6.1 depicts a street network with arcs representing streets and nodes representing intersections. Commuters from different residential areas have to pass a sequence of 6 intersections before they arrive at a parking lot. The first intersection is located immediately next to the residential area and there is no choice for the commuter. Afterwards, at the second until the fifth intersection, commuters can chose in most cases between two roads. The driving time for the roads are identical and each commuter has to take 5 roads between the first and the fifth intersection. The waiting times at the intersections vary and are depicted above the intersection nodes. The planning task is to determine for each residential area the route with the shortest commute time. Since the driving time is constant this translates into minimizing the total waiting time.

A commuter from the residential area connected to intersection 3 has to make a decision at each of the five intersections, which he has to pass, on the next street to take. This leads to $2^5 = 32$ possible routes. We call each of this route a "policy". Basically, it is a plan on which route to take or a sequence of decisions. Commuters from the other residential areas have a smaller number of possible paths, due to the structure of the network. Thus, an upper bound on the number of paths, which have to be investigated in order to determine the optimal policy for each residential area, is 6 (residential areas) times 32 possible paths, which gives 172 paths to investigate. We will now show how we can solve the problem with a substantial smaller effort by using dyamic programming.

Dynamic programming has been developed in 1953 by Richard Bellmann ($\star$1920, †1984), an

applied mathematician who worked for the RAND corporation, an US american think tank, and who was a professor at the University of Southern California. The optimality principle of Bellmann essentially says that for solving a global problem to optimality it suffices to solve a series of local problems to optimality. In Definition 27 below we will give a more precise definition of the optimality principle.

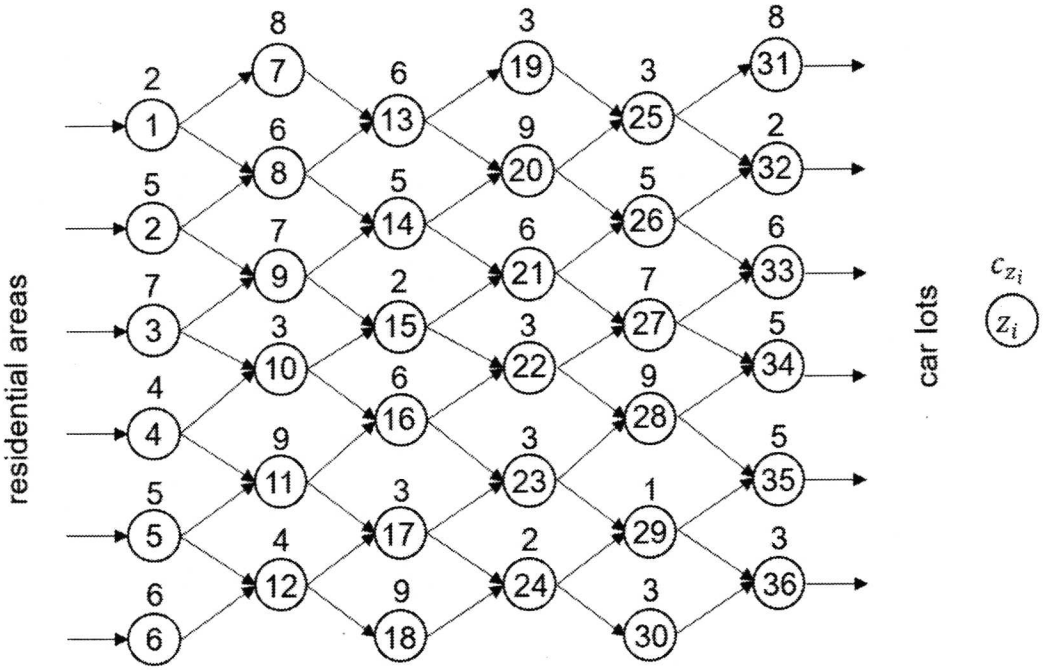


Figure 6.1: Commuter's street network

For modelling a problem as dynamic program we have to define stages, states, decisions, state-dependent cost functions, and the recursion function. We will do so in the following. Depending on the problem, recursions functions can be formulated backward or forward oriented. In the commuter example we will employ a backward oriented recursion function.

Stages: Travelling from his residential area to one of the city parking lots a commuter has to pass 6 intersections. At each of the intersections except the sixth one the commuter has to decide, which street to take. We divide the journey of the commuter into stages. At stage 1 he is at intersection 1, at stage 2 at intersection 2 etc. With i we denote the stage and with n the total number of stages.

States: In each stage $i = 1, \ldots, n$ the commuter will pass exactly one intersection. The intersections, which can be passed in stage i are defined as states. $Z_1 = \{1, \ldots, 6\}$ are the states of the first stage, $Z_2 = \{7, \ldots, 12\}$ the states of the second stage etc.

Decisions: In each state $z_i \in Z_i$ of the stage $i = 1, \ldots, 5$ the commuter has to decide, which street to take. With (z_i, z_{i+1}) we denote the street leading from intersection (state) z_i at stage i to the intersection (state) z_{i+1} at stage $i + 1$. We denote with $X_i(z_i)$ the set of decisions in state z_i. E.g., $X_3(z_3 = 13) = \{(13, 19), (13, 20)\}$ and $X_4(z_4 = 30) = \{(30, 36)\}$.

State-dependent cost function: If the commuter arrives at intersection z_i, the commuter has a waiting time of c_{z_i}. Hence, we have the state-dependent cost function $f_i(z_i) = c_{z_i}$. "Cost" is used in dynamic programming as a generic term for the goal considered in the objective function, which is in our example waiting time.

Backward recursion function: The backward recursive function determines for each state z_i in each stage i the total cost from state z_i to the final state if decision x_i is taken. In our example the total cost equals the waiting time of the driver from the point in time when he arrives at z_i until he reaches the parking lot.

$$F_i(z_i, x_i) = f_i(z_i) + F^*_{i+1}(z_{i+1}) \quad (i = 5, \ldots, 1;\ z_i \in Z_i;\ x_i \in X_i(z_i)) \tag{6.1}$$

E.g., being in state 25 and choosing street $(25, 31)$ gives total cost of $F_5(25, (25, 31)) = 3 + 8 = 11$ while being in the same state but choosing street $(25, 32)$ results in total cost of $F_5(25, (25, 32)) = 3 + 2 = 5$.

The function is termed "backward" because we are solving the problem in a backward fashion starting with the states in stage 5 and then moving backwards until we have reached the states in stage 1. The function is termed "recursion" because Equation (6.1) is used in a recursive manner to determine the total cost for each decision in each state.

Minimum total cost function: The minimum total cost function for state z_i determines amongst all possible decisions the one with the minimum cost. Thus, the function does not depend on the decision any more, which is indicated by the notation $F^*_i(z_i)$. The value of the minimum cost function is also termed as "cost-to-go". For the commuter's problem the cost-to-go for an intersection z_i is the total waiting of the commuter at intersection z_i and all following intersections on his drive to a parking lot. For each state of each stage we can now determine the minimum cost-to-go as follows:

$$F^*_i(z_i) = \min_{x_i \in X_i(z_i)} \left\{ f_i(z_i) + F^*_{i+1}(z_{i+1}) \right\} \quad (i = 5, \ldots, 1;\ z_i \in Z_i) \tag{6.2}$$

Definition 27 *The* **optimality principle of dynamic programming** *proposed by Richard Bellman states that for optimally solving a dynamic program it suffices to sequentially determine for each state z_i of each stage i the optimal decision x^*_i out of the set of decisions X_i in order to solve the overall problem to optimality.*

In order to calculate the cost-to-go for the states in stage 5 we need to know the total cost for the states in stage 6. However, these are known upfront because once arriving at an intersection in stage 6 there are no more choices for the commuter and thus he knows the wating time until reaching the parking lot. Hence, we can initialize the cost-to-go for all states in stage 6 as follows:

z_6	31	32	33	34	35	36
$F^*_6(z_6) = c_{z_6}$	8	2	6	5	5	3

We now apply the backward recursion function (6.1) and the minimum total cost function (6.2) in order to calculate the cost-go-go of all states in stages 5 to 1.

Stage $i = 5$ with intersection states $Z_5 = \{25, \ldots, 30\}$:

z_5	x_5	z_6	$f_5(z_5)$	$F_6^*(z_6)$	$F_5(z_5, x_5)$	$F_5^*(z_5)$
25	(25,31)	31	3	8	11	
25	(25,32)	32	3	2	5	5
26	(26,32)	32	5	2	7	7
26	(26,33)	33	5	6	11	
27	(27,33)	33	7	6	13	
27	(27,34)	34	7	5	12	12
28	(28,34)	34	9	5	14	14
28	(28,35)	35	9	5	14	
29	(29,35)	35	1	5	6	
29	(29,36)	36	1	3	4	4
30	(30,36)	36	3	3	6	6

For state $z = 28$ both decisions give the minimum cost. In such a case we choose one of the decisions. In this case we choose the decision, which leads into the state with the smaller number. This is state 34 and the selected decision is thus $(28, 34)$.

Stage $i = 4$:

z_4	x_4	z_5	$f_4(z_4)$	$F_5^*(z_5)$	$F_4(z_4, x_4)$	$F_4^*(z_4)$
19	(19,25)	25	3	5	8	8
20	(20,25)	25	9	5	14	14
20	(20,26)	26	9	7	16	
21	(21,26)	26	6	7	13	13
21	(21,27)	27	6	12	18	
22	(22,27)	27	3	12	15	15
22	(22,28)	28	3	14	17	
23	(23,28)	28	3	14	17	
23	(23,29)	29	3	4	7	7
24	(24,29)	29	2	4	6	6
24	(24,30)	30	2	6	8	

Stage $i = 3$:

z_3	x_3	z_4	$f_3(z_3)$	$F_4^*(z_4)$	$F_3(z_3, x_3)$	$F_3^*(z_3)$
13	(13,19)	19	6	8	14	14
13	(13,20)	20	6	14	20	
14	(14,20)	20	5	14	19	
14	(14,21)	21	5	13	18	18
15	(15,21)	21	2	13	15	15
15	(15,22)	22	2	15	17	
16	(16,22)	22	6	15	21	
16	(16,23)	23	6	7	13	13
17	(17,23)	23	3	7	10	
17	(17,24)	24	3	6	9	9
18	(18,24)	24	9	6	15	15

Stage $i = 2$:

z_2	x_2	z_3	$f_2(z_2)$	$F_3^*(z_3)$	$F_2(z_2, x_2)$	$F_2^*(z_2)$
7	(7,13)	13	8	14	22	22
8	(8,13)	13	6	14	20	20
8	(8,14)	14	6	18	24	
9	(9,14)	14	7	18	25	
9	(9,15)	15	7	15	22	22
10	(10,15)	15	3	15	18	
10	(10,16)	16	3	13	16	16
11	(11,16)	16	9	13	21	
11	(11,17)	17	9	9	18	18
12	(12,17)	17	4	9	13	13
12	(12,18)	18	4	15	19	

Stage $i = 1$:

z_1	x_1	z_2	$f_1(z_1)$	$F_2^*(z_2)$	$F_1(z_1, x_1)$	$F_1^*(z_1)$
1	(1,7)	7	2	22	24	
1	(1,8)	8	2	20	22	22
2	(2,8)	8	5	20	25	25
2	(2,9)	9	5	22	27	
3	(3,9)	9	7	22	29	
3	(3,10)	10	7	16	23	23
4	(4,10)	10	4	16	20	20
4	(4,11)	11	4	18	22	
5	(5,11)	11	5	18	23	
5	(5,12)	12	5	13	18	18
6	(6,12)	12	6	13	19	19

The table for stage 1 gives the shortest total waiting time for each residential area. The policy for obtaining the shortest waiting time, i.e., the route, which the commuter has to take, can be obtained by going through the tables for stages 1 to 5. Let us consider the residential area connected to the intersection 3. From the table for stage 1 we see that the optimal decision is $(3, 10)$ leading into intersection 10. From the table for stage 2 we see that the optimal decision at intersection 10 is $(10, 16)$ leading into intersection 16. Continuing this way for stages 3 to 5 gives the optimal policy $3 \rightarrow 10 \rightarrow 16 \rightarrow 23 \rightarrow 29 \rightarrow 36$ with a waiting time of 23 minutes. Solving the problem with dynamic programming we have applied the backward recursion function (6.1) for 55 times instead of enumerating 172 paths

6.2 Dynamic Ordering Problem

As a second example for applying dynamic programming let us consider the dynamic ordering problem (see Winston [11] pp. 1046–1050). A company needs to order material for the next five periods where the demand is as follows:

Period (i)	1	2	3	4	5
Demand (d_i)	220	280	360	140	270

For each period an order is placed and material is delivered there are fixed costs of $c = 250$. Keeping a unit of inventory for one period costs $h = 1$. There is no limit on the amount which can be ordered or kept in inventory. At the beginning of period 1 the inventory is empty. Hence, at least 220 units have to be ordered and delivered at the beginning of period 1. At the end of period 5 the inventory is required to be empty. The planning problem is how much should be ordered and delivered at the beginning of each period such that the demand is met and the costs are minimized.

Stages: The stages are the number of periods $n = 5$ for which a decision on the amount to be ordered and delivered has to be undertaken.

States: States relate to the inventory position at the end of each period. Let z_i denote the inventory at the end of period i. Since we have assumed zero inventory at the beginning of period 1 and at the end of period 5 we have $z_0 = 0$ and $z_5 = 0$. The set of possible states at the end of period $i = 1, \ldots, 4$ is

$$Z_i = \left\{ 0, \sum_{\tau=i+1}^{u} d_\tau \mid u = i+1, \ldots, n \right\} \qquad (i = 1, \ldots, n-1) \qquad (6.3)$$

For our example we obtain: $Z_1 = \{0, 280, 640, 780, 1.050\}$, $Z_2 = \{0, 360, 500, 770\}$, $Z_3 = \{0, 140, 410\}$, and $Z_4 = \{0, 270\}$.

Policy: In the case of the dynamic ordering problem, a policy is a sequence of decisions where we decide in each stage (which equals a period) how much is ordered and delivered at

the beginning of the period. The policy thus transforms the initial state "zero inventory at the beginning of period 1" into the final state "zero inventory at the end of period 5" while fulfilling the demand in periods 1 to 5.

Decisions: Depending on state z_i on stage i we have a set of decisions which we can make in stage $i+1$. For all states with nonzero inventory it can be shown that an order in the next period will never be optimal and hence the only decision to take is not to order in the next period. Since the inventory level at the initial state z_0 is zero it can be shown that the set of decisions in the next period is to order

$$X_1(z_0 = 0) = \left\{ \sum_{\tau=i}^{u} d_\tau \mid u = 1, \ldots, n \right\} \tag{6.4}$$

which gives for our example $X_1(z_0 = 0) = \{220, 500, 860, 1.000, 1.270\}$. Note that we have not considered an order decision such as $X_1 = \{640\}$ where we would order for periods 1, 2 and 4. In this case, we would have to place a new order in period 3 to cater the demand of period 3. However, when ordering in period 3 we can include the demand of period 4 in that order. By this we save on the holding costs for the demand of period 4 in periods 1 and 2. Thus ordering in period 1 for periods 1 and 2 and ordering in period 3 for periods 3 and 4 is (from the cost perspective) always better than ordering in period 1 for periods 1, 2 and 4.

The set of possible order decisions at the beginning of period $i = 2, \ldots, n$ in the case of zero inventory are:

$$X_i(z_{i-1} = 0) = \left\{ 0, \sum_{\tau=i}^{u} d_\tau \mid u = i, \ldots, n \right\} \qquad (i = 2, \ldots, n) \tag{6.5}$$

Note, that for periods $i = 2, \ldots, 5$ we have the additional order quantity of 0 in case the demand of period i has already been ordered in a preceding period. This option is not available for period 1 because we assume the inventory to be empty at the beginning of period 1. Applying Formula (6.5) to every period we obtain $X_2 = \{0, 280, 640, 780, 1.050\}$, $X_3 = \{0, 360, 500, 770\}$, $X_4 = \{0, 140, 410\}$, and $X_5 = \{0, 270\}$.

State transition function: The state transition function gives the state z_{i-1} on stage $i-1$ depending on the state z_i and the decision x_i on stage i.

$$z_i = z_{i-1} + x_i - d_i \tag{6.6}$$

Obviously, the function does simple bookkeeping by taking the inventory at the end of the preceding period adding the amount received by the beginning of the period and subtracting the demand of the period in order to come up with the inventory at the end of the period. Note that equation (6.6) is also a flow constraint. In production management equation (6.6) is termed dynamic inventory balance constraint.

State–dependent cost function

$$f_i(z_i, x_i) = h \cdot z_i + c \cdot \begin{cases} 1 \text{ , if } x_i > 0 \\ 0 \text{ , else} \end{cases} \tag{6.7}$$

The amount of inventory a the end of period i incurs cost of h per unit and in case of a delivery of x_i at the beginning of period i there are ordering costs of c.

Backward recursion function:

$$F_i(z_i, x_{i+1}) = f_{i+1}(z_{i+1}, x_{i+1}) + F^*_{i+1}(z_{i+1}) \quad (i = 4, \ldots, 0;\ z_i \in Z_i;\ x_{i+1} \in X_{i+1}(z_i)) \tag{6.8}$$

Minimum total cost function:

$$F^*_i(z_i) = \min_{x_{i+1} \in X_{i+1}(z_i)} \{F_i(z_i, x_{i+1})\} \qquad (i = 4, \ldots, 0;\ z_i \in Z_i) \tag{6.9}$$

Let us now solve the problem by dynamic programming.

Stage 5: We start with the final state $z_5 = 0$ and associated cost-to-go of $F^*_5(0) = 0$.

Stage 4: We have two possible states $z_4 = 0$ and $z_4 = 270$. In each case there is only a single decision leading into each state. We use a table for the dynamic programming. The $*$ in the last column indicates for all rows of a state $z_4 \in Z_4$ the optimal decision x_5 leading into that state as well as the optimal cost of the state $F_4(z_4)$.

z_4	x_5	z_5	$f_5(z_5, x_5)$	$F^*_5(z_5)$	$F_4(z_4, x_5)$	$F^*_4(z_4)$
0	270	0	250	0	$250 + 0 = 250$	250
270	0	0	0	0	$0 + 0 = 0$	0

Stage 3: In stage 3 we have the state set $Z_3 = \{0, 140, 410\}$. The table gives the calculations:

z_3	x_4	z_4	$f_4(z_4, x_4)$	$F^*_4(z_4)$	$F_3(z_3, x_4)$	$F^*_3(z_3)$
0	140	0	250	250	$250 + 250 = 500$	500
0	270+140=410	270	$250 + 270 = 520$	0	$520 + 0 = 520$	
140	0	0	0	250	$0 + 250 = 250$	250
410	0	270	270	0	$270 + 0 = 270$	270

Stage 2: In stage 2 we have the states $Z_2 = \{0, 360, 500, 770\}$ and the following calculations:

z_2	x_3	z_3	$f_3(z_3, x_3)$	$F_3^*(z_3)$	$F_2(z_2, x_3)$	$F_2^*(z_2)$
0	360	0	250	500	250 + 500 = 750	
0	140+360=500	140	250+140=390	250	390+250=640	640
0	410+360=770	410	410+250=660	270	660+270=930	
0	410+360=770	410	410+250=660	270	660+270=930	
360	0	0	0	500	0 + 500 = 500	500
500	0	140	140	250	140+250=390	390
770	0	410	410	270	410+270=680	680

Stage 1: In stage 1 we have the states $Z_1 = \{0, 280, 640, 780, 1.050\}$ and calculation gives:

z_1	x_2	z_2	$f_2(z_2, x_2)$	$F_2^*(z_2)$	$F_1(z_1, x_2)$	$F_1^*(z_1)$
0	280	0	250	640	250 + 640 = 890	890
0	360+280=640	360	250+360=610	500	610+500=1.110	
0	500+280=780	500	250+500=750	390	750+390=1.140	
0	770+280=1.050	770	250+770=1.020	680	1.020+680=1.700	
280	0	0	0	640	0 + 640 = 640	640
640	0	360	360	500	360+500=860	860
780	0	500	500	390	500+390=890	890
1.050	0	770	770	680	770+680=1.450	1.450

Stage 0: In stage 0 the set of states is made of the single state, i.e. $Z_0 = \{0\}$ and we obtain:

z_0	x_1	z_1	$f_1(z_1, x_1)$	$F_1^*(z_1)$	$F_0(z_0, x_1)$	$F_0^*(z_0)$
0	220	0	250	890	250 + 890 = 1.140	1.140
0	280+220=500	280	250+280=530	640	530+640=1.170	
0	640+220=860	640	250+640=890	860	890+860=1.750	
0	780+220=1.000	780	250+780=1.030	890	1.030+890=1.920	
0	1.050+220=1.270	1.050	250+1.050=1.300	1.450	1.300+1.450=2.750	

The optimal policy for the dynamic ordering problem is $x^* = (220, 280, 500, 0, 270)$. The total cost are $F_0^*(0) = 1.140$ and are made of ordering cost of 1.000 and holding cost of 140. The optimal states are $z^* = (0, 0, 140, 0, 0)$.

Solving the dynamic program to optimality also provides for non–optimal state the optimal policy to the final state $z_5 = 0$. E.g., let us consider the non–optimal state $z_2 = 500$ on stage 2. The optimal policy to reach the final state is $x^* = (0, 0, 270)$ (note that the elements of x are for stages 3, 4 and 5). The cost from the beginning of period 3 onwards are $F_2^*(500) = 390$.

Figure 6.3 gives the state space of the decision problem. Nodes represent states and arcs represent decisions. Solid arcs present optimal decisions. The optimal policy from state $z_0 = 0$ to $z_5 = 0$ fulfilling demand d_t for $t = 1, \ldots, 5$ is depicted by the sequence of fat solid

z_4	x_5	z_5	$f_5(z_5, x_5)$	$F_5^*(z_5)$	$F_4(z_4)$	
0	270	0	250	0	250 + 0 = 250	*
270	0	0	0	0	0 + 0 = 0	*
z_3	x_4	z_4	$f_4(z_4, x_4)$	$F_4^*(z_4)$	$F_3(z_3)$	
0	140	0	250	250	250 + 250 = 500	*
0	270+140=410	270	250 + 270 = 520	0	520 + 0 = 520	
140	0	0	0	250	0 + 250 = 250	*
410	0	270	270	0	270 + 0 = 270	*
z_2	x_3	z_3	$f_3(z_3, x_3)$	$F_3^*(z_3)$	$F_2(z_2)$	
0	360	0	250	500	250 + 500 = 750	
0	140+360=500	140	250+140=390	250	390+250=640	*
0	410+360=770	410	410+250=660	270	660+270=930	
360	0	0	0	500	0 + 500 = 500	*
500	0	140	140	250	140+250=390	*
770	0	410	410	270	410+270=680	*
z_1	x_2	z_2	$f_2(z_2, x_2)$	$F_2^*(z_2)$	$F_1(z_1)$	
0	280	0	250	640	250 + 640 = 890	*
0	360+280=640	360	250+360=610	500	610+500=1.110	
0	500+280=780	500	250+500=750	390	750+390=1.140	
0	770+280=1.050	770	250+770=1.020	680	1.020+680=1.700	
280	0	280	0	640	0 + 640 = 640	*
640	0	360	0+360=360	500	360+500=860	*
780	0	500	500	390	500+390=890	*
1.050	0	770	770	680	770+680=1.450	*
z_0	x_1	z_1	$f_1(z_1, x_1)$	$F_1^*(z_1)$	$F_0(z_0)$	
0	220	0	250	890	250 + 890 = 1.140	*
0	280+220=500	280	250+280=530	640	530+640=1.170	
0	640+220=860	640	250+640=890	860	890+860=1.750	
0	780+220=1.000	780	250+780=1.030	890	1.030+890=1.920	
0	1.050+220=1.270	1.050	250+1.050=1.300	1.450	1.300+1.450=2.750	

Figure 6.2: Calculating the optimal policy for the dynamic ordering problem

arcs. Optimal policies from non–optimal states z_i to the final state $z_5 = 0$ are depicted by the unique sequence of solid arcs from z_i to the final state $z_5 = 0$. E.g., from state $z_2 = 500$ to $z_5 = 0$ the optimal policy is $x^* = (0, 0, 270)$ with costs $F_2^*(500) = 390$.

6.3 Capital Budgeting Problem

As the next example for dynamic programming let us use the capital budgeting problem which we have introduced in Chapter 4 Integer and Mixed–Integer Programming. There, we have solved the problem with branch–and–bound. Table 6.1 recapitulates the problem data.

Modeling and solving the capital budegting problem with dynamic programming we will be using forward recursion. Let:

Stages: The problem has $n = 4$ stages.

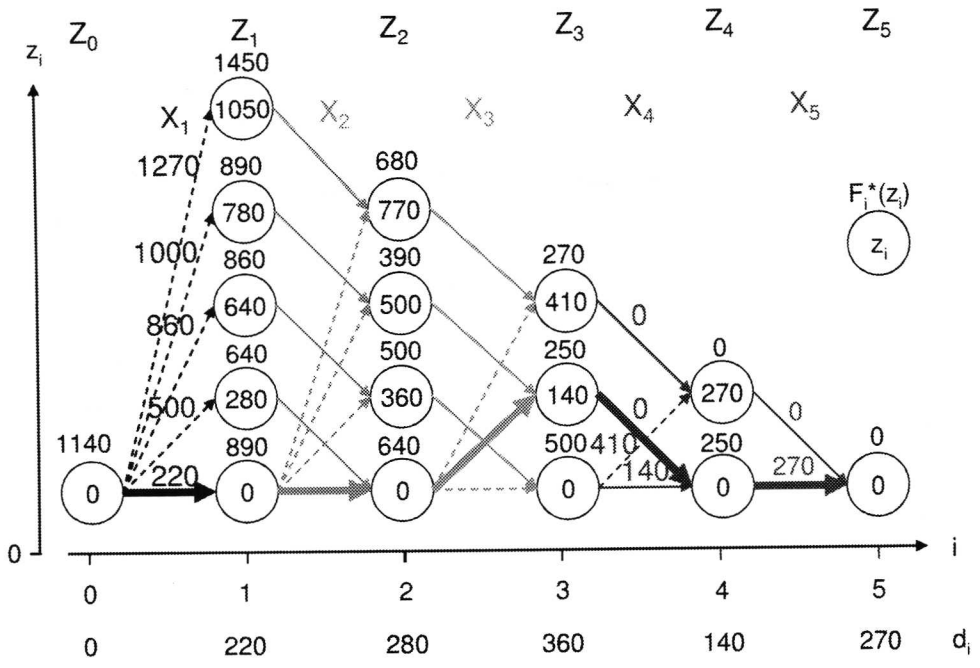


Figure 6.3: State space for the dynamic ordering problem

Project i	Value c_i	Budget demand a_i
1	16	5
2	22	7
3	12	4
4	8	3
Available budget $b = 14$		

Table 6.1: Data of the capital budgeting problem

States: Let z_i be the allocated budget by making decisions on stages 0 until i. The starting state is $z_0 = 0$ and for the final state we have $z_4 \leq 14$. This final state is untypical and also detrimental as we will see.

Decisions: In stage $i = 1, \ldots, 4$ we decide upon the project i. We thus have the decision

$$x_i = \begin{cases} 1, \text{ if project } i \text{ is selected} \\ 0, \text{ otherwise} \end{cases} \tag{6.10}$$

and the set of possible decisions:

$$X_i(z_{i-1}) = \{x_i \in \{0, 1\} \mid z_{i-1} + a_i \cdot x_i \leq 14\} \tag{6.11}$$

State transition function: Selecting project i in stage i, i.e., $x_i = 1$, increases the allocated

budget by a_i. Otherwise, the state does not change. That is we obtain

$$z_i = z_{i-1} + a_i \cdot x_i \tag{6.12}$$

State–dependent cost function:

$$f_i(x_i, z_i) = c_i \cdot x_i + 0 \cdot (14 - z_i) \tag{6.13}$$

Selecting project i in stage i increases the objective function by $f_i(x_i, z_i)$. Note that the cost function is generally defined based on the selection decision x_i and the remaining budget $(14 - z_i)$. However, for the particular problem we don't value any left–over budget, hence $(14 - z_i)$ is multiplied by 0. Note also, that in our case we maximize value but speak of a "cost" function. In dynamic programming we speak of cost function and total cost, regardless if we are considering cost or value.

Forward recursion function:

$$F_i(z_i, x_i) = f_i(x_i, z_i) + F^*_{i-1}(z_{i-1}) \quad (i = 1, \ldots, 4; z_i \in Z_i; x_i \in X_i(z_{i-1})) \tag{6.14}$$

Maximum total cost function:

$$F^*_i(z_i) = \max_{x_i \in X_i(z_{i-1})} \{F_i(z_i, x_i)\} \quad (i = 1, \ldots, 4;\ z_i \in Z_i) \tag{6.15}$$

Let us now do the calculations for our example problem. We sort the rows according to increasing state z_i.

Stage $i = 1$:

z_1	x_1	z_0	$f_1(x_1)$	$F^*_0(z_0)$	$F_1(z_1, x_1)$	$F^*_1(z_1)$
0	0	0	0	0	0	0
5	1	0	16	0	16	16

Stage $i = 2$:

z_2	x_2	z_1	$f_2(x_2)$	$F^*_1(z_1)$	$F_2(z_2, x_2)$	$F^*_2(z_2)$
0	0	0	0	0	0	0
5	0	5	0	16	16	16
7	1	0	22	0	22	22
12	1	5	22	16	38	38

Stage $i = 3$:

z_3	x_3	z_2	$f_3(x_3)$	$F_2^*(z_2)$	$F_3(z_3, x_3)$	$F_3^*(z_3)$
0	0	0	0	0	0	0
4	1	0	12	0	12	12
5	0	5	0	16	16	16
7	0	7	0	22	22	22
9	1	5	12	16	28	28
11	1	7	12	22	34	34
12	0	12	0	38	38	38

Stage $i = 4$:

z_4	x_4	z_3	$f_4(x_4)$	$F_3^*(z_3)$	$F_4(z_4, x_4)$	$F_4^*(z_4)$
0	0	0	0	0	0	0
3	1	0	8	0	8	8
4	0	4	0	12	12	12
5	0	5	0	16	16	16
7	0	7	0	22	22	22
7	1	4	8	12	20	
8	1	5	8	16	24	24
9	0	9	0	28	28	28
10	1	7	8	22	30	30
11	0	11	0	34	34	34
12	0	12	0	38	38	38
12	1	9	8	28	36	
14	1	11	8	34	42	42

In stage $i = 4$ we obtain 11 non–dominated states z_i which respect the budget constraint $z_i \leq 14$. The optimal state is $z_4 = 14$ which has the largest total cost $F_4^*(14)$. The optimal policy for the capital budgeting problem is $x^* = (0, 1, 1, 1)$. That is projects 2, 3, and 4 are selected. Figure 6.4 gives the state space of the problem. The optimal policy is indicated by the solid bold arcs, dominated decisions are given by dashed arcs.

6.4 Shortest Path Problem

Let us solve the shortest path problem addressed in Section 5.2.3 with dynamic programming with backward recursion. Figure 6.5 gives the digraph $G = (V, A, c)$ of the example.

Stages: Stages model the maximum number of nodes which have to be traversed in order to go from node 1 to any node. As can be seen in Figure 6.5 there are $n = 4$ stages. Note that in the case where the graph would not have node 3 and there would be an arc (1,5), then node 5, jointly with node 4, would still be on stage 2 as the maximum number of arcs to go from node 1 to node 5 is 2.

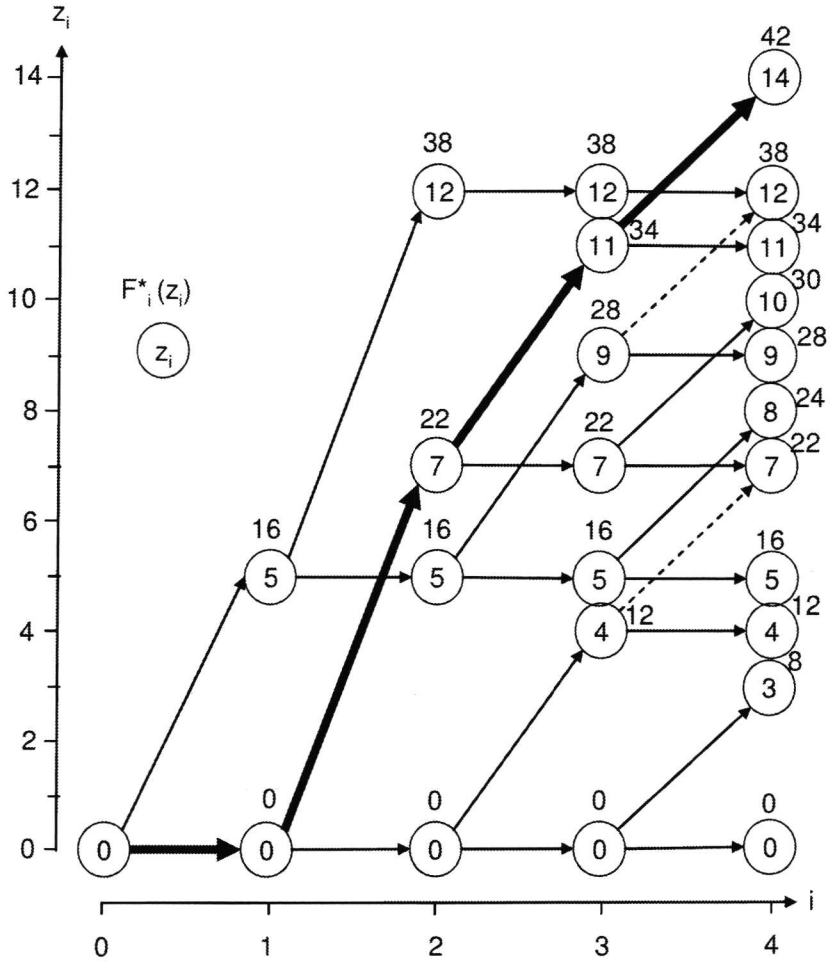


Figure 6.4: State space for the capital budgeting problem

States: Let z_i be a node on stage i. Let V_i denote the set of nodes on stage i then we obtain the set of all possible states on stage i as

$$Z_i = V_i \tag{6.16}$$

The starting state is $z_1 = 1$ and the terminal state is $z_4 = 6$

Decisions: On each stage $i = 3, \ldots, 1$ we have to determine for each state z_i from which state z_{i+1} on stage $i+1$ we are going backwards to z_i. By this we also select an arc between the two nodes z_i and z_{i+1}. The set of all possible decisions on stage i going into state z_i is

$$X_i(z_i) = \{(z_i, z_{i+1}) : z_{i+1} \in Z_{i+1} \text{ and } (z_i, z_{i+1}) \in A\}. \tag{6.17}$$

The set of all possible decisions on stage i going into any stage $z_i \in Z_i$ is

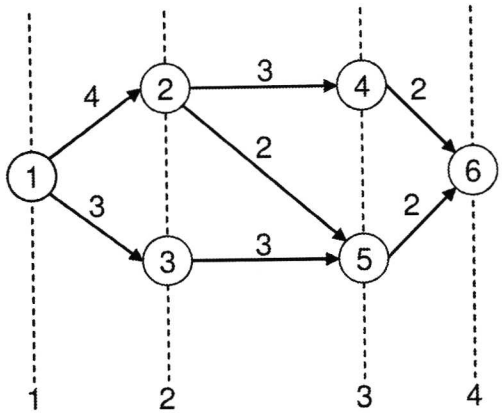


Figure 6.5: Shortest path problem

$$X_i = \{(z_i, z_{i+1}) : z_i \in Z_i \text{ and } z_{i+1} \in Z_{i+1} \text{ and } (z_i, z_{i+1}) \in A\}. \tag{6.18}$$

State–dependent cost function: $f_i(z_i, z_{i+1})$ is the length of the arc connecting node z_i and node z_{i+1}.

Backward recursion function:

$$F_i(z_i, x_i) = f_i(z_i, z_{i+1}) + F^*_{i+1}(z_{i+1}) \quad (i = 3, \ldots, 1;\ z_i \in Z_i;\ x_i \in X_i(z_i)) \tag{6.19}$$

with $F^*_4(6) = 0$. $F^*_4(6) = 0$ is the minimum cost-to-go for the final state 6 on stage 4.

Minimum total cost function

$$F^*_i(z_i) = \min_{x_i \in X_i(z_i)} \{F_i(z_i, x_i)\} \quad (i = 3, \ldots, 1;\ z_i \in Z_i) \tag{6.20}$$

We can now undertake the calcuations for the stages 3 to 1.

Stage $i = 3$:

z_3	x_3	z_4	$f_3(z_3, z_4)$	$F^*_4(z_4)$	$F_3(z_3, x_3)$	$F^*_3(z_3)$
4	(4,6)	6	2	0	2	2
5	(5,6)	6	2	0	2	2

Stage $i = 2$:

z_2	x_2	z_3	$f_2(z_2, z_3)$	$F^*_3(z_3)$	$F_2(z_2, x_2)$	$F^*_2(z_2)$
2	(2,4)	4	3	2	5	
2	(2,5)	5	2	2	4	4
3	(3,5)	5	3	2	5	5

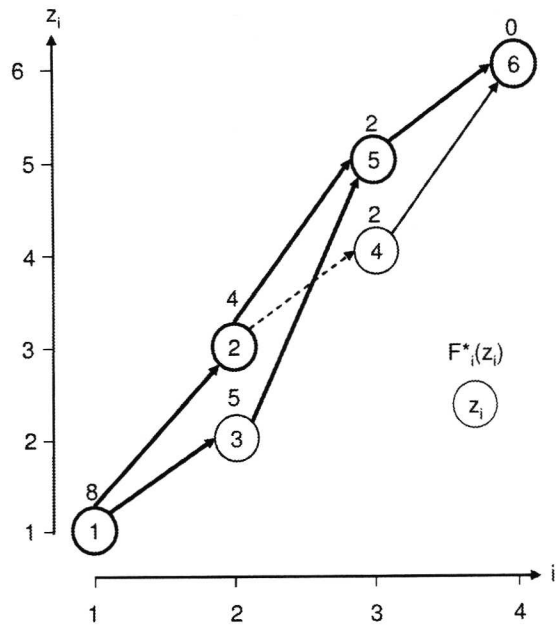


Figure 6.6: State space for the shortest path problem

Stage $i = 1$:

z_1	x_1	z_2	$f_1(z_1, z_2)$	$F_2^*(z_2)$	$F_1(z_1, x_1)$	$F_1^*(z_1)$
1	(1,2)	2	4	4	8	8
1	(1,3)	3	3	5	8	

The three tables provide for each stage $i = 4, \ldots, 1$, each state z_i, and each decision x_i the resulting total cost $F_i(z_i, x_i)$ as well as the minimum cost-to-go $F_i^*(z_i)$ for each state z_i. The optimal policy is $x^* = ((1,2),(2,5),(5,6))$ with cost $F_1^*(1) = 8$. Figure 6.6 presents the state space.

6.5 Allocation of Sales Workers to Markets

As final example we want to consider the allocation of sales staff to sub–markets (see Winston [11] pp. 1043–1044). Again, we will be using backward recursion for solving the problem. In contrast to the other problems considered so far, the objective function is non–linear. Contrary to linear programming, dynamic programming does not require linear objective functions.

The problem can be described as follows: A company wants to introduce a new product on a market which is separated into three sub–markets. The probability p_i to be successful on sub–market i depends on the number of assigned sales staff and is given in Table 6.2.

	Success probabilities (p_i)		
# Sales staff	Region 1	Region 2	Region 3
0	0	0	0
1	0.6	0.5	0.3
2	0.8	0.7	0.55
3	0.85	0.85	0.7

Table 6.2: Success probabilities for the sub–markets

The company wants to assign 5 available staff members to the three markets such that the probability to be successful on all three markets

$$P(\text{Success on all three sub–markets}) = \prod_{i=1}^{3} p_i.$$

is maximized. E.g., if we assign 3 staff members to sub–market 1 and 1 staff member to each of the sub–markets 2 and 3 then we obtain

$$P(\text{Success on all three sub–markets}) = 0.85 \cdot 0.5 \cdot 0.3 = 0.1275.$$

After these preliminary thoughts, let us model the problem as dynamic program.

Stages: We have $n = 3$ stages where each stage depicts a sub–market.

Decisions: At each stage $i = 1, \ldots, 3$ we decide how many staff member x_i we assign to sub–market i. Taking into account that we have 5 sales worker in total and that we have to assign at least 1 worker to each sub–market we have the decision set $X_i = \{1, 2, 3\}$ of possible decisions for each stage i.

States: Going backward, in stage i we depict with state z_i the number of workers which are still available for decisions in stages 0 through i. The initial state at stage 0 is $z_0 = 0$ and the final state at stage 3 is $z_3 = 5$. Thus we have $Z_1 = \{1, 2, 3\}$, $Z_2 = \{2, 3, 4\}$, and $Z_3 = \{3, 4, 5\}$.

State–dependent cost function: Assigning x_i worker to sub–market i results in a success probability $f_i(x_i) = p_i(x_i)$ on this sub–market where the probabilities are given in Table 6.2. Note that the state–dependent cost function does not depend on the state z_i. Hence, we could consider any sequence of the three sub–markets.

State transition function:

$$z_{i-1} = z_i - x_i \qquad (i = 3, \ldots, i) \tag{6.21}$$

Backward recursion function:

$$F_i(z_i, x_{i+1}) = p_{i+1}(x_{i+1}) \cdot F^*_{i+1}(z_{i+1}) \quad (i = 2, \ldots, 0;\ z_i \in Z_i;\ x_{i+1} \in X_{i+1}(z_i)) \tag{6.22}$$

with $F^*_3(5) = 1$, i.e. the probability to be successful on the non–existing sub–market 4 by assigning 0 workers to this market and leaving 5 workers for sub–markets 1 to 3. $X_{i+1}(z_i)$ denotes the set of decisions which can be undertaken on stage $i+1$ leading into state z_i on stage i. For example, the decisions leading to state $z_1 = 2$ on stage $i = 1$ are $X_2(z_1 = 2) = \{1, 2\}$. That is, we can assign in stage 2 (and thus to market 2) either 1 or 2 workers. We cannot assign 3 workers because then we would have assigned 5 workers for markets 1 and 2 and no workers would be left for market 3.

Total maximum cost function: We denote with $F^*_i(z_i)$ the probability to be successful on sub–markets $i+1, \ldots, 3$, if z_i sales workers are available for the sub–markets $0, \ldots, i$ and $5 - z_i$ sales workers have been assigned to sub–markets $i+1, ..., 3$.

$$F^*_i(z_i) = \max_{x_{i+1} \in X_{i+1}(z_i)} \{F_i(z_i, x_{i+1})\} \quad (i = 2, \ldots, 0;\ z_i \in Z_i) \tag{6.23}$$

We now solve the problem by backward recursion starting on stage 2:

Stage $i = 2$

z_2	x_3	z_3	$f_3(x_3)$	$F^*_3(z_3)$	$F_2(z_2, x_3)$	$F^*_2(z_3 2$
4	1	5	0.3	1	$0.3 \cdot 1 = 0.3$	0.3
3	2	5	0.55	1	$0.55 \cdot 1 = 0.55$	0.55
2	3	5	0.7	1	$0.7 \cdot 1 = 0.7$	0.7

Stage $i = 1$

z_1	x_2	z_2	$f_2(x_2)$	$F^*_2(z_2)$	$F_1(z_1, x_2)$	$F^*_1(z_1)$
3	1	4	0.5	0.3	$0.5 \cdot 0.3 = 0.15$	0.15
2	2	4	0.7	0.3	$0.7 \cdot 0.3 = 0.21$	
2	1	3	0.5	0.55	$0.5 \cdot 0.55 = 0.275$	0.275
1	3	4	0.85	0.3	$0.85 \cdot 0.3 = 0.255$	
1	2	3	0.7	0.55	$0.7 \cdot 0.55 = 0.385$	0.385
1	1	2	0.5	0.7	$0.5 \cdot 0.7 = 0.35$	

Stage $i = 0$

z_0	x_1	z_1	$f_1(x_1)$	$F^*_1(z_1)$	$F_0(z_0, x_1)$	$F^*_0(z_0)$
0	3	3	0.85	0.15	$0.85 \cdot 0.15 = 0.1275$	
0	2	2	0.8	0.275	$0.8 \cdot 0.275 = 0.22$	
0	1	1	0.6	0.385	$0.6 \cdot 0.385 = 0.231$	0.231

We obtain the optimal policy $x^* = (1, 2, 2)$ with total maximum cost $F_0^*(0) = 0.231$. The state space is given in Fig. 6.7.

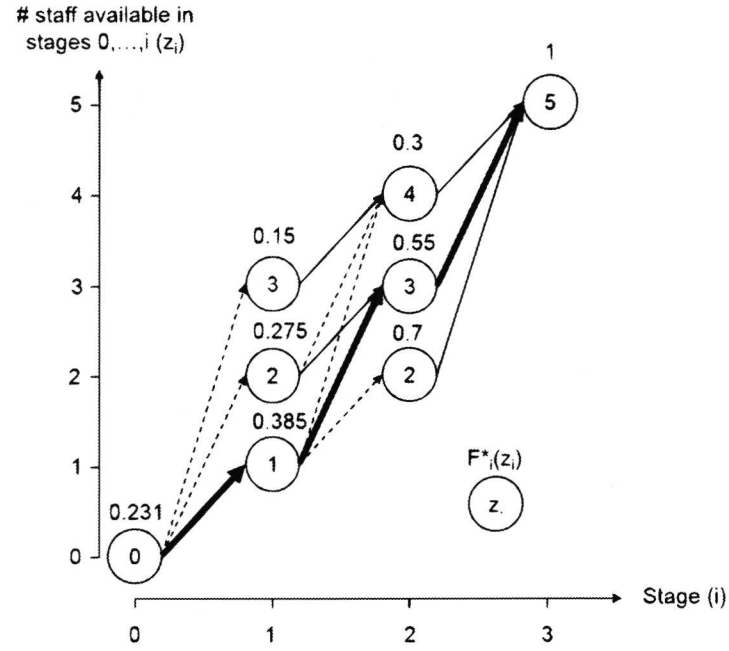


Figure 6.7: State space for the resource allocation problem

Chapter 7

Operations Research Software

7.1 Gurobi

Gurobi is one of the most powerful commercial solvers available today. Gurobi is widely used in industry and academia. In this class, we use Gurobi with Google Colab. Google Colab operates as a digital notebook, enabling you to write and execute computer code without requiring any software installations on your personal computer. It is important to recognize that, when tackling larger and more intricate problems above the scope of this course, you may eventually need to install the software on your computer and acquire the necessary licenses.

Figure 7.1 gives the Gurobi implementation of the beer glass production planning problem. The problem is solved with the "m.optimize()"-command within the notebook. Gurobi not only finds the optimal solution but also provides additional information like shadow prices and reduced costs.

```
[ ]  # Declare and initialize model
     m = gp.Model('Beer Mug Production Planning')

     # Create decision variables
     x = m.addVars(2, name=["Wheat", "Lager"])

     # Objective function
     m.setObjective(5*x[0] + 4.5*x[1], GRB.MAXIMIZE)

     # Create capacity constraints
     production = m.addConstr(6*x[0] + 5*x[1] <= 60, name='Production')
     inventory = m.addConstr(10*x[0] + 20*x[1] <= 150, name='Inventory')
     demand = m.addConstr(1*x[0] <= 8, name='Demand')
```

Figure 7.1: Gurobi model of the beer glas production problem

7.2 LINDO

LINDO is a commercial software for solving linear programs which is offered by LINDO Systems Inc (see www.lindo.com). A free version of the program ("Classic LINDO") can be downloaded at `https://www.lindo.com/index.php/ls-downloads`. A user manual can be downloaded at `https://www.lindo.com/index.php/ls-downloads/user-manuals`.

7.2.1 Modelling and Solving Linear Programs with LINDO

Figure 7.2 gives the LINDO model formulation for the beer production example. As can bee seen, the model is straight forward and very similar to the way we are modeling linear programs in this manuscript.

```
C:\USERS\KOLISCH\DOCUMENTS\Beer glass.ltx
Max 500 x1 + 450 x2
st
6 x1 + 5 x2 <= 60
10 x1 + 20 x2 <= 150
x2 <= 8
end
```

Figure 7.2: LP of the beer glass production planning problem in LINDO

Using the `Solve`–command, the linear program is solved and the solution given in Figure 7.3 is obtained. As can be seen, LINDO is providing the information about the slack or surplus of the constraints as well as dual prices and reduced costs.

```
Reports Window

LP OPTIMUM FOUND AT STEP      2

        OBJECTIVE FUNCTION VALUE

        1)       5142.857

 VARIABLE        VALUE          REDUCED COST
       X1         6.428571          0.000000
       X2         4.285714          0.000000

      ROW   SLACK OR SURPLUS     DUAL PRICES
        2)         0.000000         78.571426
        3)         0.000000          2.857143
        4)         3.714286          0.000000

NO. ITERATIONS=       2
```

Figure 7.3: Optimal solution of the beer glass production planning problem in LINDO

With the command `Reports Tableu` the tableau can be visualized. The tableau will be shown in the "Report Window". Figure 7.4 shows the initial LINDO tableau of the beer mug production planning problem. The tableau is casted slightly different from the way

done in this manuscript. The z–variable for the objective function is called `ART` (for artifical variable), the objective function coefficients are multiplied by -1 and slack variables are called `SLK` and numbered according to the row. Furthermore, the objective function is represented in the first row. Note, that the start tableau (and only the start tabelau) has a copy of the objective function in the last row.

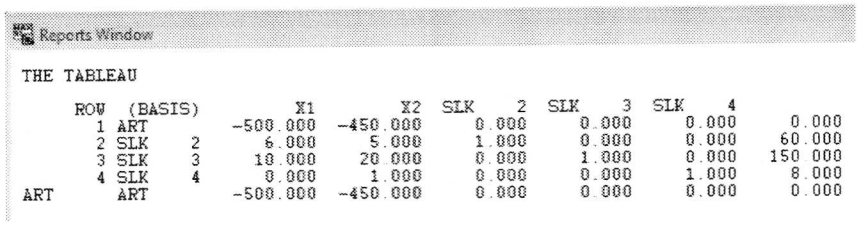


Figure 7.4: Initial tableau of the beer glass production planning problem in LINDO

7.2.2 Performing a Pivot-Step with LINDO

When solving the linear program, LINDO provides the option to manually select the pivot element defined by pivot column and pivot row. For this one has to activiate the problem window (which includes the formulation of the LP) and choose the command sequence "`Solve Pivot...`". Figure 7.5 shows that the pivot element (X2, Row 3) is selected. Figure 7.6 shows the first pivot step and the resulting second tableau.

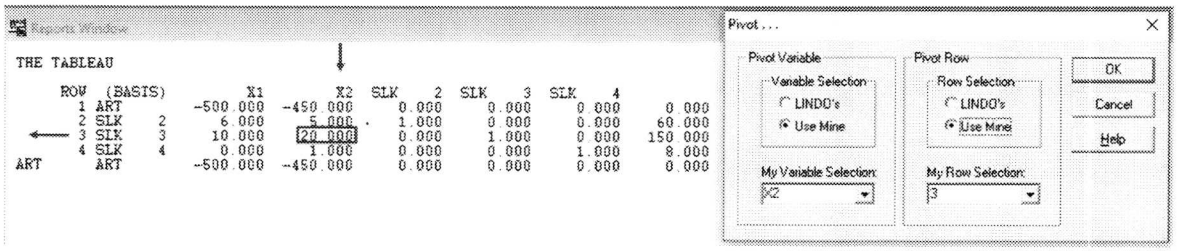


Figure 7.5: Manually selection of the pivot element in LINDO

```
      X2 ENTERS AT VALUE    7.5000     IN ROW     3 OBJ. VALUE=  3375.0

THE TABLEAU

     ROW  (BASIS)          X1         X2  SLK    2  SLK    3  SLK    4
       1 ART         -275.000      0.000    0.000   22.500    0.000  3375.000
       2 SLK    2       3.500      0.000    1.000   -0.250    0.000    22.500
       3          X2    0.500      1.000    0.000    0.050    0.000     7.500
       4 SLK    4      -0.500      0.000    0.000   -0.050    1.000     0.500
```

Figure 7.6: Second tableau in LINDO

7.2.3 Modelling and Solving Integer Programs with LINDO

Integer variables are defined in LINDO by using the GIN–command. Figure 7.7 provides the LINDO–formulation of the integer program (4.5) – (4.8) and Figure 7.8 gives the protocol when solving the problem with branch–and–bound.

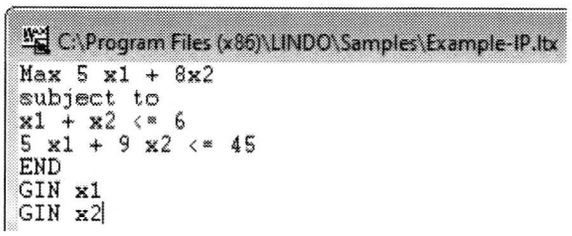


Figure 7.7: LINDO–model of the integer problem

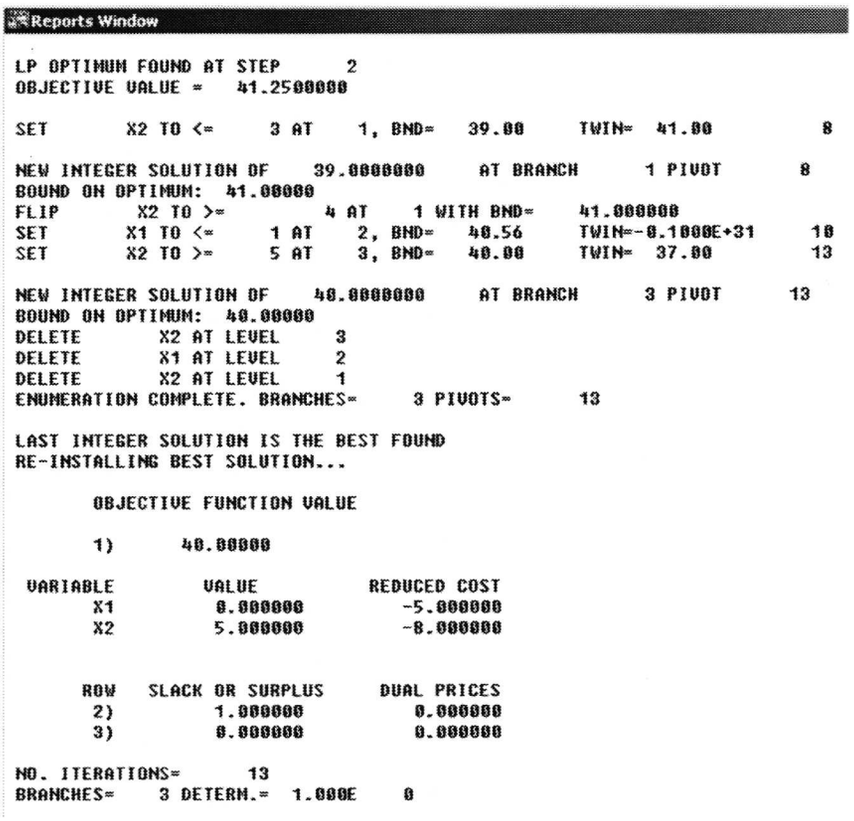


Figure 7.8: LINDO–protocol when solving the integer problem

Binary variables are defined in LINDO by using the INT–command. There are two ways using the INT– (and the GIN–) command. We can either define each variable separately as it has been done for the integer variables in Figure 7.7 or we can define INT 4 which states that the first 4 variables are binary. This option is used in the LINDO–model given in Figure

7.9 which depicts the capital budgeting problem. The protocol of the branch–and–bound procedure is given in 7.10.

C:\Program Files (x86)\LINDO\Samples\CBP.ltx

```
! Capital Budgeting Problem
!
Max 16 x1 + 22 x2 + 12 x3 + 8 x4
subject to
5 x1 + 7 x2 + 4 x3+ 3 x4 <=14
END
INT 4
```

Figure 7.9: LINDO–model of the capital budgeting problem

Reports Window

```
LP OPTIMUM FOUND AT STEP      3
OBJECTIVE VALUE =   44.0000000


NEW INTEGER SOLUTION OF    42.0000000     AT BRANCH      0 PIVOT       3
RE-INSTALLING BEST SOLUTION...

       OBJECTIVE FUNCTION VALUE

       1)      42.00000

 VARIABLE        VALUE          REDUCED COST
       X1         0.000000        -16.000000
       X2         1.000000        -22.000000
       X3         1.000000        -12.000000
       X4         1.000000         -8.000000


      ROW   SLACK OR SURPLUS     DUAL PRICES
       2)         0.000000          0.000000

NO. ITERATIONS=       3
BRANCHES=     0 DETERM.=  1.000E    0
```

Figure 7.10: LINDO–protocol when solving the capital budgeting problem

Bibliography

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice Hall, Englewood Cliffs, 1993.

[2] D.P. Bertsekas. *Network Optimization: Continuous and Discrete Models.* Athena Scientific, Belmont, Mass., 1998.

[3] S.P. Bradley, A.C. Hax, and T.L. Magnanti. *Applied Mathematical Programming.* Addison-Wesley, Reading, Massachusetts, 1977. A pdf is available at *http://web.mit.edu/15.053/www/AMP.htm.*

[4] W. Domschke, A. Drexl, R. Klein, and A. Scholl. *Einführung in Operations Research.* Springer, Berlin, 9 edition, 2015.

[5] F. Eisenführ, M. Weber, and T. Langer. *Rational Decision Making.* Springer, Heidelberg, 1 edition, 2010.

[6] M Friedman and L.J. Savage. The utility analysis of choices involving risk. *Journal of Political Economy,* 56(4):279–304, 1848.

[7] F.S. Hillier and G.J. Lieberman. *Introduction to Operations Research.* McGraw–Hill, New York, 8 edition, 2005.

[8] P.A. Jensen and J.F. Bard. *Operations Research Models and Methods.* Wiley, Hoboken, NJ, 2002.

[9] A.R. Ravindran. *Operations Research and Management Science Handbook.* Taylor & Francis, Boca Raton, 2008.

[10] G. Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, 6 edition, 2023.

[11] W.L. Winston. *Operations Research — Applications and Algorithms.* Brooks/Cole—Thompson Learning, Belmont, 4 edition, 2004.