5 Lagrangian Relaxation

- Basic idea:
  - Integer problems have frequently a specific structure
  - There is a subproblem within the entire program that can be solved quite efficiently
  - However, there are certain restrictions that make the problem much more complicated
  - Specifically, if we drop these restrictions, we may be able to provide optimal solutions efficiently
- Thus, the basic idea of the Lagrangian Relaxation is to drop the complex restrictions and penalize their violation by Lagrange multipliers in the objective function
- In order to tackle complex problems, the Lagrangian Relaxation has been proven as a very competitive solution technique

5.1 Basic structure

- We obtain the following problem structure

$$\boxed{\begin{array}{l} \text{Maximize } Z = c^T \cdot x \\ \text{s.t.} \\ A^1 \cdot x \le b^1 \quad m_1 \text{ complex restrictions} \\ A^2 \cdot x \le b^2 \quad m_2 \text{ easy restrictions} \\ x \in IR^{n_1} \times \mathbf{Z}^{n_2} \text{ with } n_1 + n_2 = n \end{array}}$$

- The remaining problem (characterized by the restrictions determined in matrix A^2) is easy to solve
- Thus, we transform the problem into a new structure that is depicted on the following slide

Lagrangian Relaxation

- We obtain

$$\boxed{\begin{array}{l} \text{Maximize } Z = c^T \cdot x + \pi^T \cdot \left(b^1 - A^1 \cdot x\right) \\ \text{s.t.} \\ A^2 \cdot x \le b^2 \quad m_2 \text{ easy restrictions} \\ x \in IR^{n_1} \times \mathbf{Z}^{n_2} \text{ with } n_1 + n_2 = n \end{array}}$$

- Attributes
  - The solution space comprises all solutions to the original problem
  - For positive vectors *π*, $z_{LR}(\pi)$ is obviously an upper bound on the objective function value of each feasible solution *x* of the original problem

Upper Bound

- The latter is true since we know that

$$\boxed{\left(b^1 - A^1 \cdot x\right) \geq 0,\ x \text{ feasible}}$$

- Thus, for feasible solutions and positive multipliers, we obtain

$$\boxed{c^T \cdot x + \pi^T \cdot \left(b^1 - A^1 \cdot x\right) \geq c^T \cdot x}$$

- Thus, an optimal solution to the Lagrange Problem provides an upper bound to the original problem

Transformation of objective function

- Thus, we obtain

$$\boxed{c^T \cdot x + \pi^T \cdot \left(b - A^1 \cdot x\right) = \left(c^T - \pi^T \cdot A^1\right) \cdot x + \pi^T \cdot b}$$

- This modified function underlines
  - That the objective function value depends on the chosen Lagrange multipliers *π* in a non-linear way
  - Moreover, these multipliers are multiplied with the variable vector *x*
  - If *π* is kept constant, we just have a simpler problem working with modified *c*-entries
  - Specifically, since the constant adder has no impact on optimality, we have the following modified *c*-entries

$$\boxed{\tilde{c}^T = \left(c^T - \pi^T \cdot A^1\right)}$$

Dual Lagrangian problem

- In what follows, we concentrate on an optimization of the multipliers π
- What values are promising?
- Obviously, since we are able to solve the simpler problem, it is reasonable to apply it to a problem that is the most equivalent to our original instance
- Thus, we obtain the problem

$$\boxed{z_{LD} = \min\left\{\left(c^T - \pi^T \cdot A^1\right) \cdot x + \pi^T \cdot b \,\middle|\, \pi \geq 0\right\}}$$

- In what follows, we tackle this problem by subgradient methods

The Set Packing Problem

- Given is a finite set *U* with *m* elements
- Consider a list of *n* subsets of *U*
- The task is to chose as many subsets as possible …
- … under the condition that the chosen subsets have to be **pairwise disjoint**, i.e. in the final selection, no element is included in multiple subsets
- We use a binary vector *x* to denote our choice
- The subset list is expressed by a binary matrix *A*:

  It holds that $a_{i,j}$=1, if element *i* is included in subset *j*
- In the general case, a weight w_j is assigned for each subset *j*

$$\begin{aligned} &\max z = w^T \cdot x \\ &\text{s.t. } A \cdot x \leq 1 \\ &x \in \{0,1\}^n \end{aligned}$$

$$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \dots & a_{i,j} & \dots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix}$$

Example

- Let us consider the following instance of the **Set Packing Problem**

$$\text{Maximize } Z = 3 \cdot x_1 + 4 \cdot x_2 + 2 \cdot x_3 + 5 \cdot x_4$$

s.t.

$$\underbrace{x_1 + x_2 \leq 1 \wedge x_1 + x_3 \leq 1 \wedge x_1 + x_4 \leq 1 \wedge x_2 + x_4 \leq 1}_{\text{Complex restrictions}} \wedge x_1, x_2, x_3, x_4 \in \{0,1\}$$

$$\Rightarrow A^1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Example – Lagrange Relaxation Problem

- Thus, we obtain

$$\text{Maximize } Z = (3-\pi_1-\pi_2-\pi_3)\cdot x_1 + (4-\pi_1-\pi_4)\cdot x_2$$
$$+(2-\pi_2)\cdot x_3 + (5-\pi_3-\pi_4)\cdot x_4 + \pi_1+\pi_2+\pi_3+\pi_4$$
$$\text{s.t.}$$
$$x_1, x_2, x_3, x_4 \in \{0,1\}$$

- If, for instance $\pi_1=\pi_2=\pi_3=\pi_4=2$, we obtain the following optimal solution

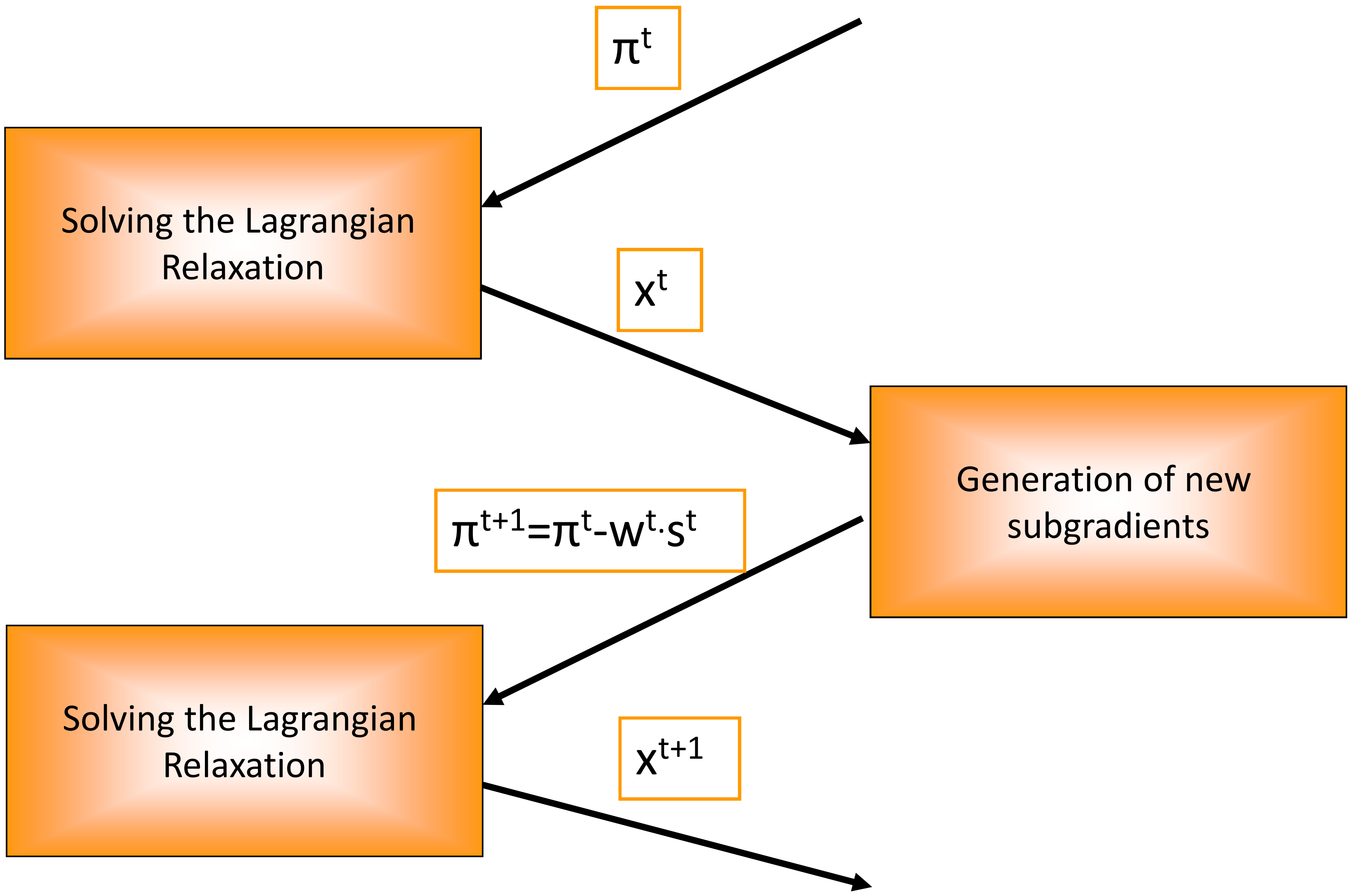
$$\text{Maximize } Z = (3-2-2-2)\cdot x_1 + (4-2-2)\cdot x_2 + (2-2)\cdot x_3$$
$$+(5-2-2)\cdot x_4 + 2+2+2+2$$
$$= -3\cdot x_1 + x_4 + 8$$
$$\text{Optimal solution: } x_1 = 0, x_2 = 0, x_3 = 0, \text{ and } x_4 = 1 \Rightarrow Z = 9$$

Subgradient methods

Basic idea:

1. Generate an optimal solution x^t for the Lagrange Relaxation with predetermined Lagrange multipliers π^t
2. Update the Lagrange multipliers based on the found optimal solution x^t and the found subgradients s_t

Scheme


π^t
Solving the Lagrangian Relaxation
x^t
Generation of new subgradients
$\pi^{t+1}=\pi^t-w^t \cdot s^t$
Solving the Lagrangian Relaxation
x^{t+1}

Subgradient

- For its introduction, we commence with convex functions. Specifically, a function z is denoted as convex if it holds that

$$z : IR^{m_1} \to IR, \text{ with}$$
$$z\left(\alpha \cdot \pi^1 + (1-\alpha) \cdot \pi^2\right) \le \alpha \cdot z\left(\pi^1\right) + (1-\alpha) \cdot z\left(\pi^2\right), \forall \pi^1, \pi^2, 0 \le \alpha \le 1$$

- With other words

$$\exists s \in IR^{m_1} : \forall \pi \in IR^{m_1} : z\left(\bar{\pi}\right) + s^T \cdot \left(\pi - \bar{\pi}\right) \le z\left(\pi\right) \qquad (1)$$

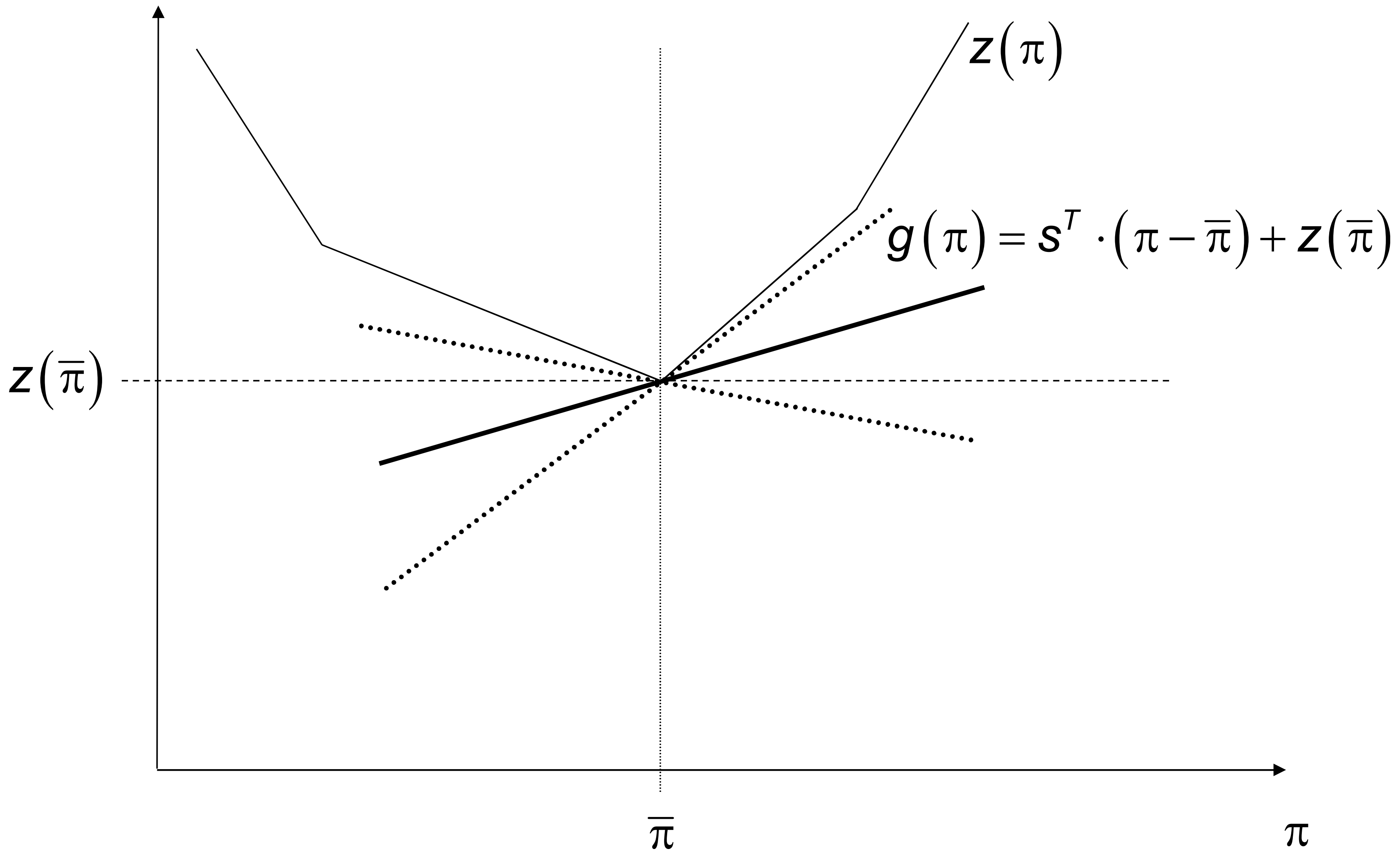
Thus, we may identify a hyperplane with

$$H^{s,\bar{\pi}} = \left\{ \left(\pi, z\right) \middle| s^T \cdot \left(\pi - \bar{\pi}\right) = z - z\left(\bar{\pi}\right) \wedge z \le z\left(\pi\right) \right\} \qquad (2)$$

Subgradient

- Note that **s is denoted as a subgradient** if it fulfills restriction (1)
- Note further that if z is not differentiable at position π, there exists an infinite set of subgradients
- To the contrary, if z is differentiable at position π, the subgradients are unambiguously defined

Illustration



$z(\pi)$
$g(\pi) = s^T \cdot (\pi - \overline{\pi}) + z(\overline{\pi})$
$z(\overline{\pi})$
$\overline{\pi}$
π

5.2 A Lagrangian Relaxation for the KP

- In what follows, we consider the Knapsack Problem
- Its mathematical definition is given by

$$\text{Maximize } Z = \sum_{j=1}^{n} x_j \cdot p_j$$

$$\text{s.t. } \sum_{j=1}^{n} x_j \cdot w_j \leq C \wedge x \in \{0,1\}^n$$

$$\Rightarrow A = \begin{pmatrix} w_1 & w_2 & w_3 & \dots & w_{n-1} & w_n \end{pmatrix}$$

The Lagrangian problem

- Thus, we obtain

$$\text{Maximize } Z = \sum_{j=1}^{n} x_j \cdot p_j + \pi \cdot \left(C - \sum_{j=1}^{n} x_j \cdot w_j \right)$$

$$= \sum_{j=1}^{n} \left(p_j - \pi \cdot w_j \right) \cdot x_j + \pi \cdot C$$

$$\text{s.t. } x \in \{0,1\}^n$$

Conclusions

- Clearly, the Lagrangian Relaxation of the Knapsack Problem is characterized by the solution set $S=\{0,1\}^n$
- It comprises in total 2^n elements
- Let us denote each element by an unambiguous index k, i.e., we obtain the elements x^k out of $\{0,1\}^n$
- Consequently, the *k*th element obtains the objective function value

$$\boxed{z_{x^k}(\pi) = \sum_{j=1}^{n} \left(p_j - \pi \cdot w_j\right) \cdot x_j^k + \pi \cdot C}$$

Further conclusions

- If π is constant, we are looking for the x^k-vector that maximizes the objective function value

$$\boxed{z_{LR}(\pi) = \max_{x^k} z_{x^k}(\pi) = \max_{x^k}\left(\sum_{j=1}^{n}\left(p_j - \pi \cdot w_j\right)\cdot x_j^k + \pi \cdot C\right)}$$

- Clearly, this function is convex since it is a maximum of a finite set of linear functions
- If, however, x is kept constant (set to $\tilde{x}$), we obtain a linear function in π

$$\boxed{z_{\tilde{x}}(\pi) = \sum_{j=1}^{n}\left(p_j - \pi \cdot w_j\right)\cdot \tilde{x}_j + \pi \cdot C = \underbrace{\sum_{j=1}^{n}\left(C - w_j \cdot \tilde{x}_j\right)}_{\text{Constant}}\cdot \pi + \underbrace{\sum_{j=1}^{n} p_j \cdot \tilde{x}_j}_{\text{Constant}}}$$

Example

- We resume with an instance of the Knapsack Problem
- Specifically, we have

$$\boxed{\begin{array}{l}\text{Maximize } Z = 4\cdot x_1 + 7\cdot x_2 + 5\cdot x_3 \\ \text{s.t. } 4\cdot x_1 + 5\cdot x_2 + 3\cdot x_3 \leq 10 \wedge x \in \{0,1\}^n\end{array}}$$

- Clearly, since good 3 obviously outperforms good 1, we obtain an optimal solution by selecting the goods 2 and 3
- This leads to an optimal objective function value of 12

Example – Objective function value

- Let us consider the Lagrange Relaxation

$$\text{Maximize } Z = 4 \cdot x_1 + 7 \cdot x_2 + 5 \cdot x_3 + \pi \cdot \left(10 - 4 \cdot x_1 - 5 \cdot x_2 - 3 \cdot x_3\right)$$
$$= \left(4 - 4 \cdot \pi\right) \cdot x_1 + \left(7 - 5 \cdot \pi\right) \cdot x_2 + \left(5 - 3 \cdot \pi\right) \cdot x_3 + 10 \cdot \pi$$
$$\text{s.t. } x \in \{0,1\}^n$$

- Obviously, there are *2^3=8* possible *x*-vectors

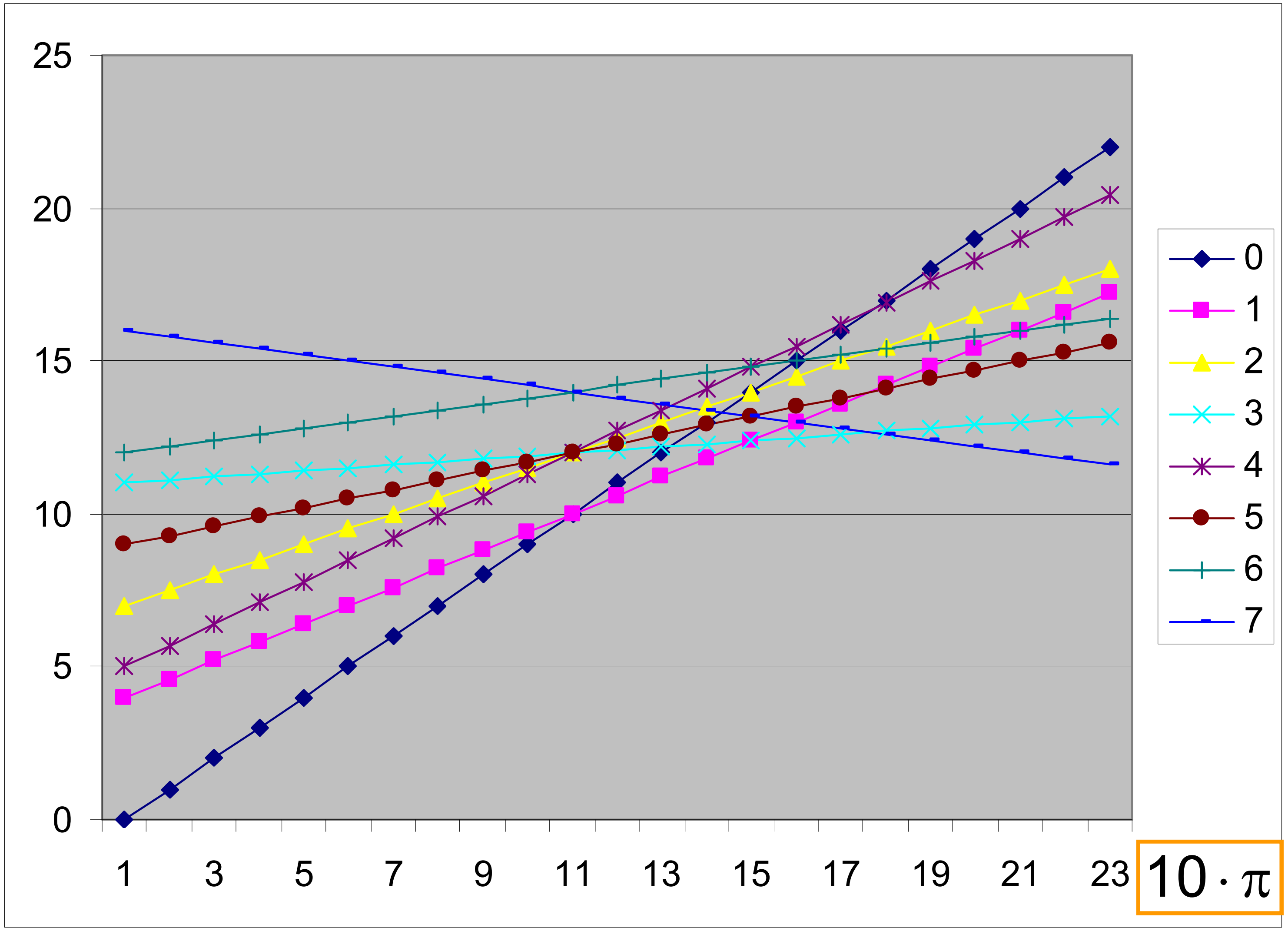
$$x^0 = \begin{pmatrix}0\\0\\0\end{pmatrix}; x^1 = \begin{pmatrix}1\\0\\0\end{pmatrix}; x^2 = \begin{pmatrix}0\\1\\0\end{pmatrix}; x^3 = \begin{pmatrix}1\\1\\0\end{pmatrix}; x^4 = \begin{pmatrix}0\\0\\1\end{pmatrix}; x^5 = \begin{pmatrix}1\\0\\1\end{pmatrix}; x^6 = \begin{pmatrix}0\\1\\1\end{pmatrix}; x^7 = \begin{pmatrix}1\\1\\1\end{pmatrix}$$

… with the objective function values

k	$z_{LR}(\pi)$
0	$10\cdot\pi$
1	$6\cdot\pi+4$
2	$5\cdot\pi+7$
3	$\pi+11$
4	$7\cdot\pi+5$
5	$3\cdot\pi+9$
6	$2\cdot\pi+12$
7	$-2\cdot\pi+16$

Illustration

$z_{LR}\left(\pi, x^i\right)$



$10 \cdot \pi$

Consequence

- Clearly, we obtain the best value 14 at π=1.0
- In order to derive the best multipliers, we may use insights into the structure of the Knapsack Problem
- Note that each good has a quality ratio of price divided by weight, i.e., $q_i=p_i/w_i$
- Clearly, an optimal solution to the continuous relaxation of the original Knapsack Problem may be generated by selecting all goods in non-increasing order until the knapsack is entirely filled
- Consequently, the last good, which we denote as critical, may be assigned only fractionally
- Let us now consider the quality ratio $q_c=p_c/w_c$ of this critical good c

We set $\pi=q_c=p_c/w_c$

- Then, we obtain by $\boxed{\pi \cdot C}$
  the capacity of the knapsack measured in price units of the critical good, i.e., what we can obtain if we would use its capacity entirely for carrying this critical good only
- But, what about the other goods with improved quality ratios?
- Their contributions are determined by

$$\boxed{\sum_{j=1}^{n}\left(p_j - \pi \cdot w_j\right)\cdot x_j^k = \sum_{j=1}^{n}\left(p_j - \left(\frac{p_c}{w_c}\right)\cdot w_j\right)\cdot x_j^k}$$

Resulting objective function value

- Clearly, this formula provides the cognition that we obtain just the price difference for taking a good outperforming *c* instead of *c*
- I.e., with other words, we obtain just the objective function value of the optimal solution of the LP-relaxation
- This is obviously an upper bound of the original problem
- The optimal solution x* to the LP-relaxation may be defined as follows

$$x_j^* = \begin{cases} 1 & \text{if } j < c \\ 0 & \text{if } j > c \\ \dfrac{C - \sum_{i=1}^{c-1} w_i}{w_c} & \text{if } j = c \end{cases}$$

Solving the Lagrangian problem

- By considering the objective function
$$\sum_{j=1}^{n}\left(p_j - \pi \cdot w_j\right) \cdot x_j^k + \pi \cdot C$$
- we may get the following optimal solution
$$x_j^* = \begin{cases} 1 \text{ if } p_j > \pi \cdot w_j \Leftrightarrow \dfrac{p_j}{w_j} > \pi \\ 0 \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$
- Obviously, this is also an optimal solution to the continuous relaxation of the Lagrangian problem
- We obtain the following objective function value for this problem
$$\sum_{j=1}^{n} p_j \cdot x_j^* + \pi \cdot \left(C - \sum_{j=1}^{n} w_j \cdot x_j^* \right) \geq \sum_{j=1}^{n} p_j \cdot x_j^{**},$$
with x_j^{**} optimal solution to the LP-relaxation

Conclusion

- Thus, by setting

$$\boxed{\pi = \frac{p_c}{w_c} \wedge x_j^* = \begin{cases} 1 \text{ if } p_j > \pi \cdot w_j \Leftrightarrow \dfrac{p_j}{w_j} > \pi \\ 0 \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}}$$

- we obtain equal objective function values for the Lagrangian Relaxation and the LP-relaxation
- Consequently, we directly know that there is no strictly positive π-value available that leads to a smaller objective function value

$$\boxed{\sum_{j=1}^{n} \left(p_j - \pi \cdot w_j\right) \cdot x_j^k + \pi \cdot C}$$

Convexity of the Lagrangian dual

- We obtain the objective function of the Lagrangian dual $\boxed{\text{Maximize } Z(x,\pi) = c^T \cdot x + \pi^T \cdot \left(b^1 - A^1 \cdot x\right)}$
- Clearly, if x is kept fixed, this function is linear in π
- Thus, by choosing the maximum of these linear functions, we obviously obtain a convex objective function of the Lagrangian dual
- Note that this function becomes concave if we select the minimum of the linear functions
- In what follows, in case of a maximization problem, we introduce $\boxed{z(\tilde{\pi}) = max_x \left(Z(x,\tilde{\pi})\right)}$

Subgradient method – Description

- Owing to the convexity of the objective function, we are able of applying a subgradient method
- This method commences with initial multipliers
- By selecting a subgradient, we try to improve this multiplier in order to minimize the resulting objective function value
- Specifically, we subtract this subgradient multiplied with an increment rate
- The increment rate is chosen in order to reduce the multipliers as much as possible, but without missing optimal constellations
- Unfortunately, this cannot be guaranteed
- In what follows, we give the formalized definition of the subgradient method

In general: Subgradient method

1. Initialization
   - Generate initial multipliers $\boxed{\pi^{(1)} = \overline{\pi}}$
   - t:=1
2. Determination of subgradients
   - Generate new subgradient $\boxed{s^{(t)} = s^{(t)}\left(\pi^{(t)}\right)}$
3. Stopping criterion
   - If $s^{(t)}=0$ and 0 is feasible subgradient, then terminate
4. Increment update
   - Determine an increment rate
5. Update $\boxed{w^{(t)}}$
   - Set $\boxed{\pi^{(t+1)} = \pi^{(t)} - w^{(t)} \cdot s^{(t)}}$
   - For all i, do: $\boxed{\pi_i^{(t+1)} < 0 \Rightarrow \pi_i^{(t+1)} := 0}$
   - t:=t+1; Go to step 2

Additional information

- Step 3 (Stopping criterion)
  - Clearly, if *s=0* is a valid subgradient, we have found an optimal value for *π*
  - This results directly from the definition of subgradients

$$\exists s \in IR^{m_1} : \forall \pi \in IR^{m_1} : z(\bar{\pi}) + s^T \cdot (\pi - \bar{\pi}) \leq z(\pi) \qquad (1)$$

Thus, we may identify a hyperplane with

$$H^{s,\bar{\pi}} = \left\{ (\pi, z) \middle| s^T \cdot (\pi - \bar{\pi}) = z - z(\bar{\pi}) \wedge z \leq z(\pi) \right\} \qquad (2)$$

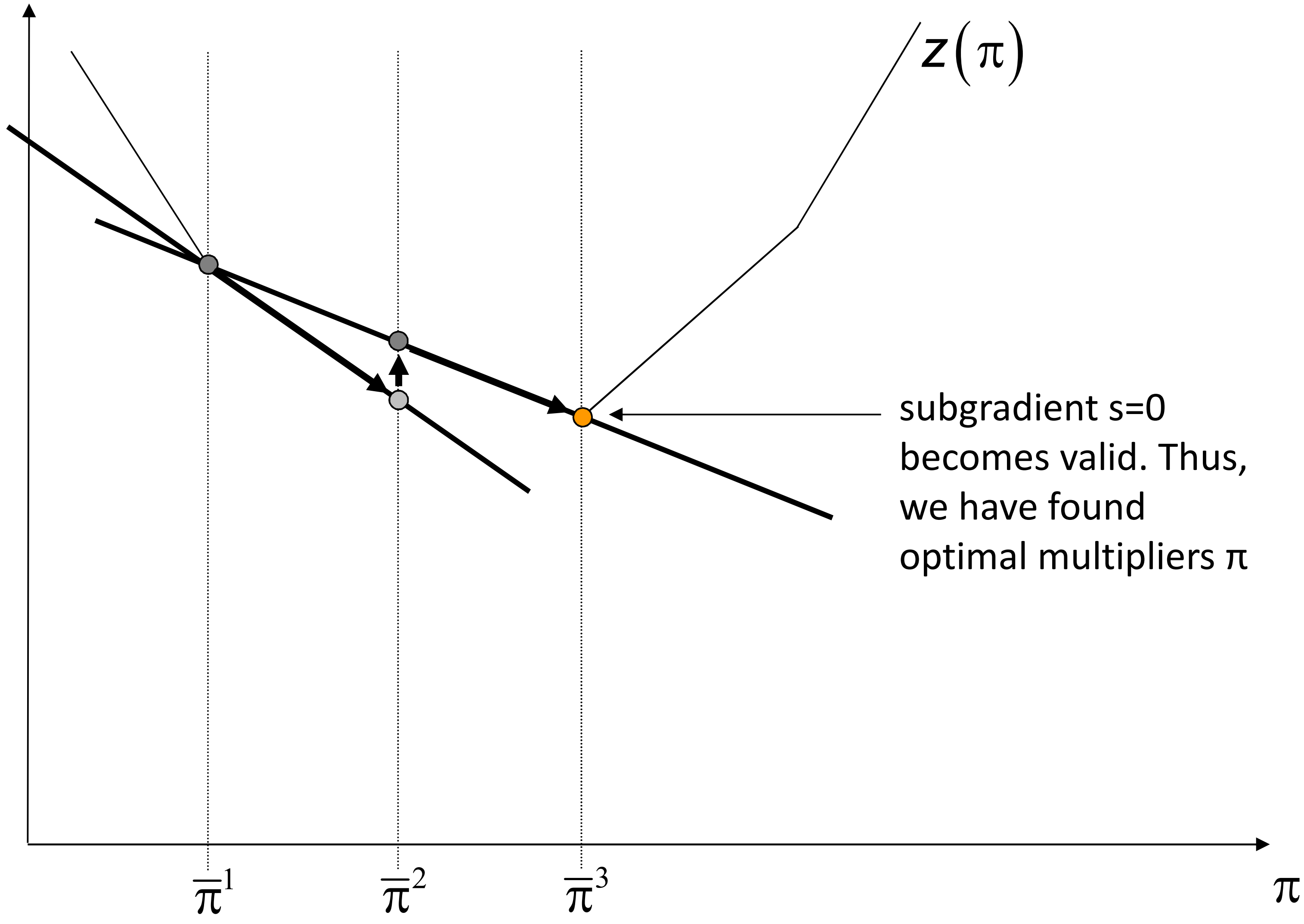
  - Clearly, if *s=0* is a valid subgradient, we obtain

$$\forall \pi \in IR^{m_1} : z(\bar{\pi}) + 0^T \cdot (\pi - \bar{\pi}) = z(\bar{\pi}) \leq z(\pi)$$

$\Rightarrow$

$\bar{\pi}$ is an optimal multiplier

Illustration



subgradient s=0 becomes valid. Thus, we have found optimal multipliers π

Main problems

- Within the subgradient approach, several parameters are difficult to define
- Termination criterion
  - It is frequently impossible to decide whether 0 is a valid subgradient or not
  - Consequently, many methods make use of a predetermined number of iterations
- Finding an appropriate increment rate is a complex task that requires significant insights into the respective problem structure
  - For a successful convergence of the calculation, the increment rate has the limiting value 0
  - However, the sum of all rates used throughout the calculation is unlimited in order to cover the entire solution space

Increment rate – Approach of Polyak

- Increment rate is determined according to the distance to a lower bound/best-known solution value
- Specifically, larger steps are always conducted as long as there is a significant distance to the bound
- Otherwise, as distances become smaller, the increment rate is significantly reduced
- Formula

$$\boxed{w^{(t+1)} = \frac{\rho^{(t)} \cdot \left(z_{LR}\left(\pi^{(t)}\right) - \overline{z}\right)}{\left\| s^{(t)} \right\|^2}, \rho^{(1)} = 2 \wedge \rho^{(t+1)} = \frac{\rho^{(t)}}{2^{\left\lfloor t/T \right\rfloor}}}$$

Generating subgradients

- How to generate subgradients?
- If x is a solution of the Lagrangian problem for $\boxed{\bar{\pi}}$
- Then, s is a valid subgradient, with $\boxed{s = b^1 - A^1 \cdot x}$
- Why? Let us consider

$$\boxed{\forall \pi \in IR^{m_1} : \exists s \in IR^{m_1} : z(\bar{\pi}) + s^T \cdot (\pi - \bar{\pi}) \leq z(\pi)}$$

- We substitute accordingly

$$\forall \pi \in IR^{m_1} : z(\bar{\pi}) + \left(b^1 - A^1 \cdot x\right)^T \cdot (\pi - \bar{\pi})$$

$$= c^T \cdot x + \bar{\pi}^T \cdot \left(b^1 - A^1 \cdot x\right) + \left(b^1 - A^1 \cdot x\right)^T \cdot (\pi - \bar{\pi})$$

$$= c^T \cdot x + \pi^T \cdot \left(b^1 - A^1 \cdot x\right)$$

Due to the definition of z, there may be an improved solution $\tilde{x}$ for π.

Therefore, we conclude

$$= c^T \cdot x + \pi^T \cdot \left(b^1 - A^1 \cdot x\right) \leq z(\pi)$$

Interpretation of subgradients

- Clearly, if a restriction i is not fulfilled by the solution x generated for the current multiplier $\boxed{\overline{\pi}}$,
  we face the following constellation

$$\boxed{\text{Maximize } Z = c^T \cdot x + \pi^T \cdot \left(b^1 - A^1 \cdot x\right)}$$

- Consequently, the subgradient increases these penalties if there is a violation
- Otherwise, these penalties are reduced by the product of increment rate and resulting gap in the respective complex restriction (given by A^1 and b^1)
- Note that all calculations and conclusions are valid only if all multipliers are positive
- Thus, we correct all negative entries $\pi_i<0$ to zero after being calculated
- Consequently, we obtain $\boxed{\pi_i^{(t+1)} = \max\left\{\pi_i^{(t)} - w^{(t)} \cdot s_i^{(t)}, 0\right\}}$

Example

- Let us apply the approach of Polyak to our problem
- We set $\boxed{x_1 = x_2 = x_3 = 1 \wedge \rho^{(t)} = \rho^{(0)} = 2 \wedge \overline{\pi} = 0}$
- Moreover, we have an initial solution x^{ini}=(0,1,1) with $z^{ini}=12$
- Thus, by using $z^{bound}=z^{ini}=12$, we obtain the following results

Iteration	π	x_π	$z(\pi)$	$s^{(t)}$	$w^{(t)}$
1	0	(1,1,1)	16	-2	2
2	4	(0,0,0)	40	10	14/25
3	0	(1,1,1)	16	-2	2
4	4	(0,0,0)	40	10	14/25

Example – Continuation

- We modify $x_1 = x_2 = x_3 = 1 \wedge \rho^{(t)} = \rho^{(0)} = 0{,}25 \wedge \overline{\pi} = 0$

Iteration	π	x_π	$z(\pi)$	$s^{(t)}$	$w^{(t)}$
1	0	(1,1,1)	16	-2	0,25
2	0,5	(1,1,1)	15	-2	0,188
3	0,88	(1,1,1)	14,25	-2	0,141
4	1,16	(0,1,1)	14,31	2	0,145
5	0,87	(1,1,1)	14,27	-2	0,142
6	1,15	(0,1,1)	14,3	2	0,144
7	0,86	(1,1,1)	14,27	-2	0,142
8	1,15	(0,1,1)	14,29	2	0,143
9	0,86	(1,1,1)	14,28	-2	0,142
10	1,15	(0,1,1)	14,29	2	0,143
11	0,86	(1,1,1)	14,28	-2	0,143
12	1,14	(0,1,1)	14,29	2	0,143

Example – Observations

- Unfortunately, it turns out that a convergence of the subgradient method cannot be guaranteed
- However, for the example we may provide a converging calculation by setting

$$\boxed{w^{(t+1)} = \frac{\rho^{(t)} \cdot \left(z_{LR}\left(\pi^{(t)}\right) - \overline{z}\right)}{\left\|s^{(t)}\right\|^2}, \rho^{(0)} = 0{,}5 \wedge \rho^{(t+1)} = \frac{\rho^{(t)}}{2^{\left\lfloor t/10 \right\rfloor}}}$$

- Note that – aside from subgradient methods – the literature provides a large variety of alternative procedures for finding optimal subgradients

5.3 A LR approach for the sTSP

- In what follows, we introduce a second Branch&Bound approach to the **symmetric TSP**
- It bases on a much more sophisticated lower bound
- This bound is obtained by generating a 1-tree and bases on the following cognitions
  - A Traveling Salesman tour defines a spanning tree after erasing one node *s* from the tour and its connecting edges
  - A lower bound therefore can be determined by taking the length of the **minimal** spanning tree plus the weight of two minimally chosen edges that connect the spanning tree with *s*

5.3.1 Deriving a new lower bound

- Bearing these cognitions in mind, we obtain a new lower bound as follows
- We isolate node *s* out of the set of nodes *V* and erase this node from *V*. We define *V'*=*V*-{*s*} and eliminate all edges that connect *s* with the rest of the graph
- Subsequently, we calculate the MST in the remaining graph
- Finally, we add the minimally weighted edges connecting node *s* with the MST
- Clearly, this bound LB(*s*) depends on the choice of node *s*
- Therefore, the node *s* is sought that maximizes this bound
- In literature, this bound is denoted as the **max 1-tree bound**

Symmetric case: Parameters and variables

Parameterss

N: Number of nodes (customers)

$c_{i,j} \left(1 \le i < j \le N\right)$: Costs for using the edge $\left(i, j\right)$

Variables

$x_{i,j} \left(1 \le i < j \le N\right)$: Binary variable that is one if and only if the path of the salesman uses the edge $\left(i, j\right)$

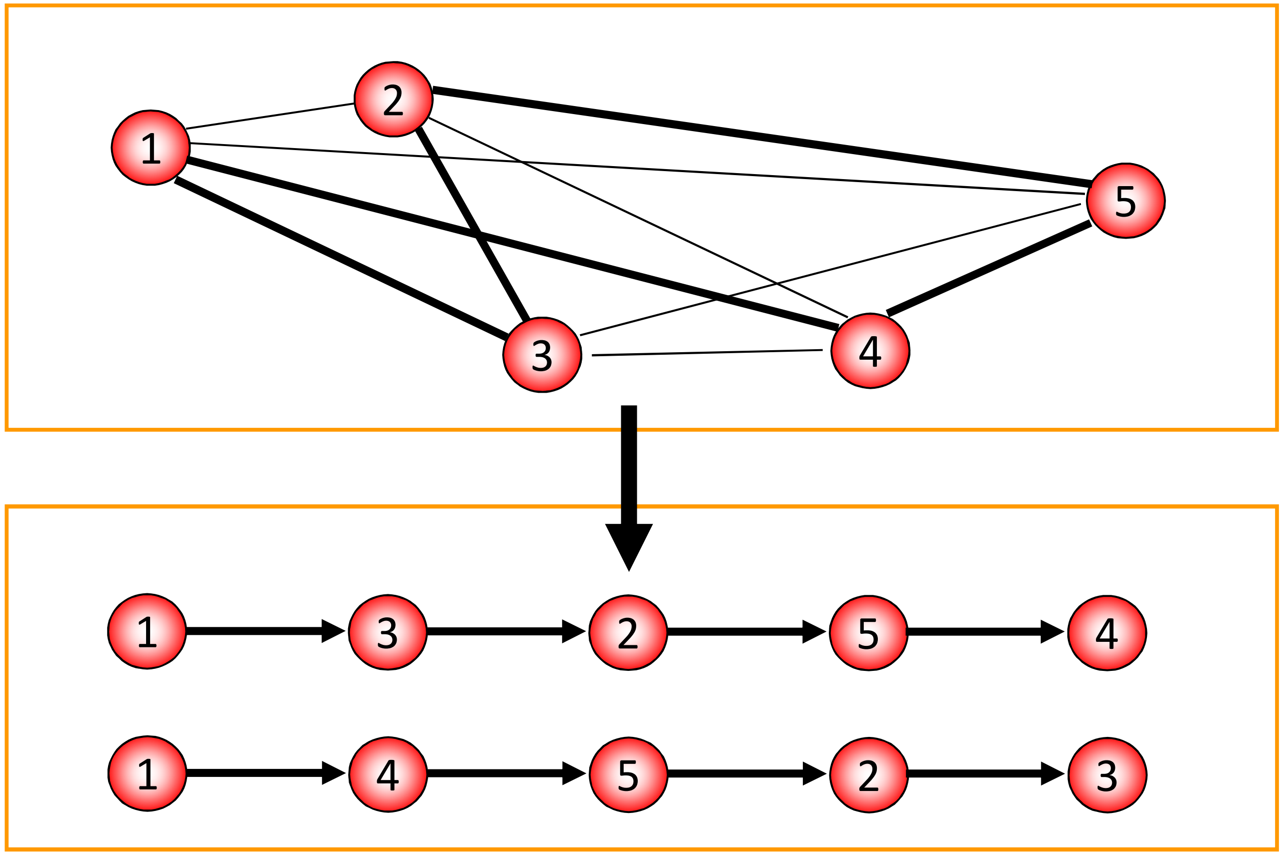
Symmetric case: Restrictions

1. $\forall i \in \{1,\ldots,N\} : \sum_{j>i} x_{i,j} + \sum_{j<i} x_{j,i} = 2$ (Each node has two neighbors)
2. x represents a 1-tree (connected undirected graph with exactly one cycle)

3. $\forall i, j\,(i<j) \in \{1,\ldots,N\} : 0 \le x_{i,j} \le 1$
4. $\forall i, j\,(i<j) \in \{1,\ldots,N\} : x_{i,j}$ is an integer

1-tree tour representation

- A 1-tree is a connected undirected graph with n nodes and n edges comprising a single cycle
- Thus, by ensuring degree 2 for each node, depending on the starting point (and read direction), alternative tours of equal length may result



$$\begin{tikzcd} 1 \arrow[r] & 3 \arrow[r] & 2 \arrow[r] & 5 \arrow[r] & 4 \end{tikzcd}$$

$$\begin{tikzcd} 1 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 2 \arrow[r] & 3 \end{tikzcd}$$

Trees

- In order to connect locations with a minimal number of arcs, trees are used
- A tree comprises
  - No isolated node, i.e., it is connected
  - No cycle
- In a directed graph, each node in a tree has a single unambiguously defined predecessor (i.e., its father node)
- A set of unconnected trees is denoted as a forest

Trees – Some attributes

5.3.1.1 Lemma

We consider an undirected graph N=(V,E) with n nodes. The following attributes are equivalent

(1) N is connected and has no cycle

(2) Each pair of nodes is connected by an unambiguously defined path

(3) N is connected and has n-1 arcs

(4) N has no cycle and has n-1 arcs

Proof of Lemma 5.3.1.1

- (1) $\Rightarrow$ (2)

  Since N is connected, for each pair of nodes there is a path connecting them. Since N is without a cycle, these paths are unambiguously defined. Otherwise, a cycle can be constructed

- (2) $\Rightarrow$ (1)

  Since each pair of nodes is connected, N is connected. In consequence of unambiguous paths, N has no cycle

Proof of Lemma 5.3.1.1

- (4) $\Rightarrow$ (3)

  This proof is given by induction for the number of nodes n

  n=3: trivial

  n>3: we consider the step from n to n+1:

  Let N be a network with n+1 nodes and n arcs. Since N has no cycle, there are nodes with a node degree of less than 2. Otherwise, there would be a cycle.

  At first, we show that these nodes have degree 1. Let us assume that these nodes have degree 0. We identify one of these nodes, let us say node k.

Proof of Lemma 5.3.1.1

- We obtain a network N' with n nodes and n arcs. Moreover, we erase arc (s,t) and obtain a network N'' without cycle, but with n nodes and n-1 arcs. Thus, by induction, we conclude that N'' is connected. By reinserting (s,t), we get a cycle. Thus, we conclude degree(k)=1
- Now, we erase node k and the arc emerging from it
- By induction, we know that the remaining network is connected and has n nodes and n-1 arcs
- Consequently, we obtain the proposition by reinserting k and the connecting arc

Proof of Lemma 5.3.1.1

- $(3) \Rightarrow (4)$

  Trivial since each connected network with n nodes and n-1 arcs is without any cycle

- $(1) \Rightarrow (4)$

  Again, we give this proof by induction for the number of nodes n

  We commence with n=3. Trivial case.

  n+1>3: We assume by induction that the assumption holds for networks with n≥3 nodes. As shown before, since N is without any cycle, there is a node k in N with degree(k)=1. By erasing node k and the arc starting from it, we generate N' out of N. Clearly, N' is still connected and by induction we know that N' has n nodes and n-1 arcs. Consequently, by reinserting k and the arc starting there, the proposition follows immediately

Proof of Lemma 5.3.1.1

- (4) $\Rightarrow$ (1)

  Since N has no cycle and has n-1 arcs connecting n nodes, N must be connected

  Consequently, the propositions of Lemma 5.3.1.1 follow

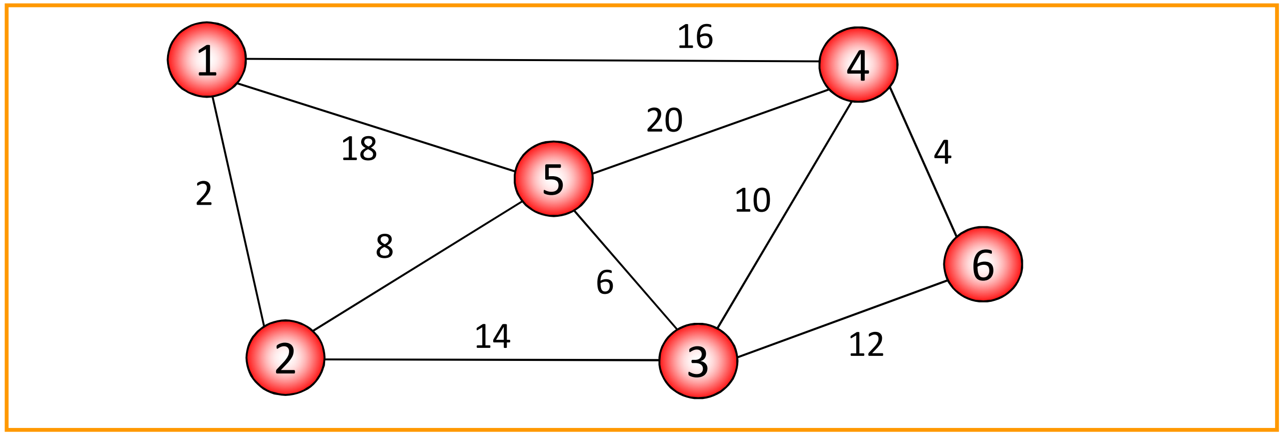
Spanning trees

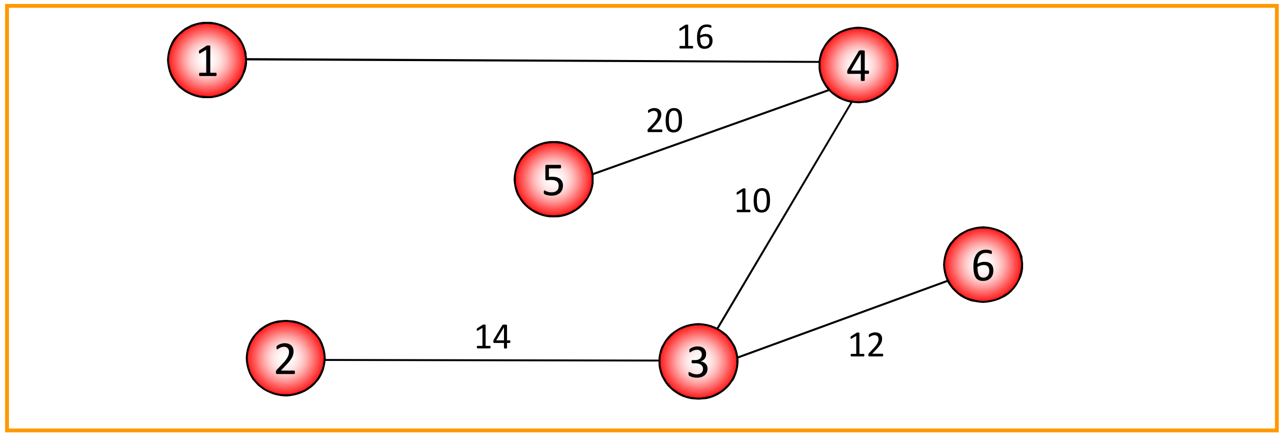
5.3.1.2 Definition

We consider a network N=(V,E) with n nodes. A spanning tree ST(N) of N is a tree that connects all nodes of N. We denote a spanning tree as a minimal spanning tree (ST*(N)) of N if the sum of weights of all used arcs is minimal, i.e., there is no other spanning tree ST(N) of N with a lower total weight.

- In what follows, we denote L(ST*(N)) as the total weight of the minimal spanning tree of network N

Example of a spanning tree





Calculating its weight

- The total weight amounts to 16+20+10+12+14=72
- In what follows, we consider minimum spanning trees and crucial attributes of them
- In order to calculate minimum spanning trees, there is one particular attribute that allows us to generate very efficient construction procedures

An attribute of minimum spanning trees

5.3.1.3 Lemma

Let X be a subset of the nodes of N=(V,E), and let edge e be the smallest edge connecting X to V-X. Then, edge e is part of a minimum spanning tree

Proof of Lemma 5.3.1.3

- Let us suppose there is a minimum spanning tree T not containing edge e
- Furthermore, let e=(u,v), with u in X and v not in X
- Then, since T is a spanning tree, it contains a unique path from u to v, which, together with e, forms a cycle in N
- This path comprises another edge f connecting X to V-X
- T∪{e}-{f} is another spanning tree S
- It has the same number of edges and remains connected since you can replace any path containing f by one going the other way around the cycle
- Since T was optimally chosen, it has identical weight as S, and therefore it holds: w(f)=w(e)
- Consequently, the newly generated spanning tree is also minimal and contains edge e
- This completes the proof

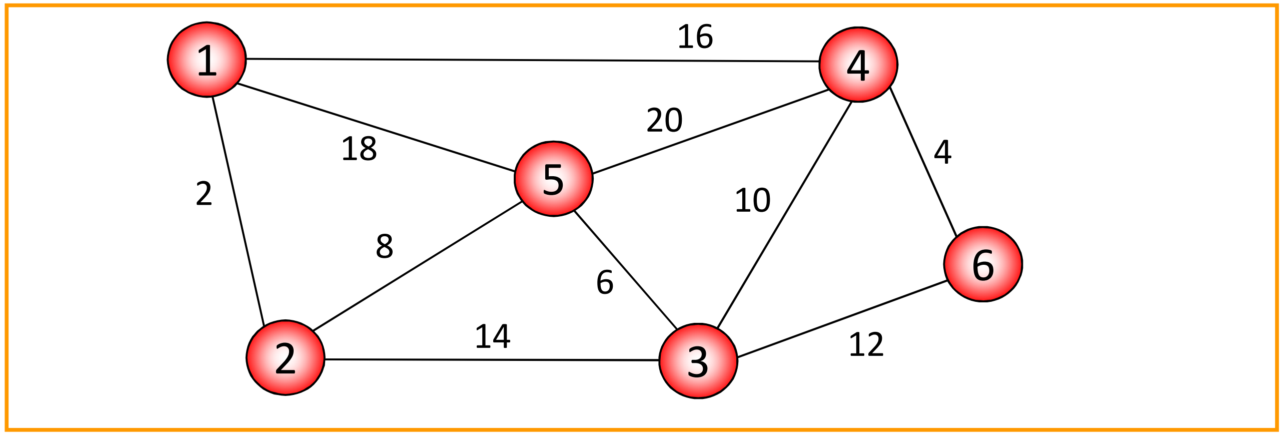
Consequences

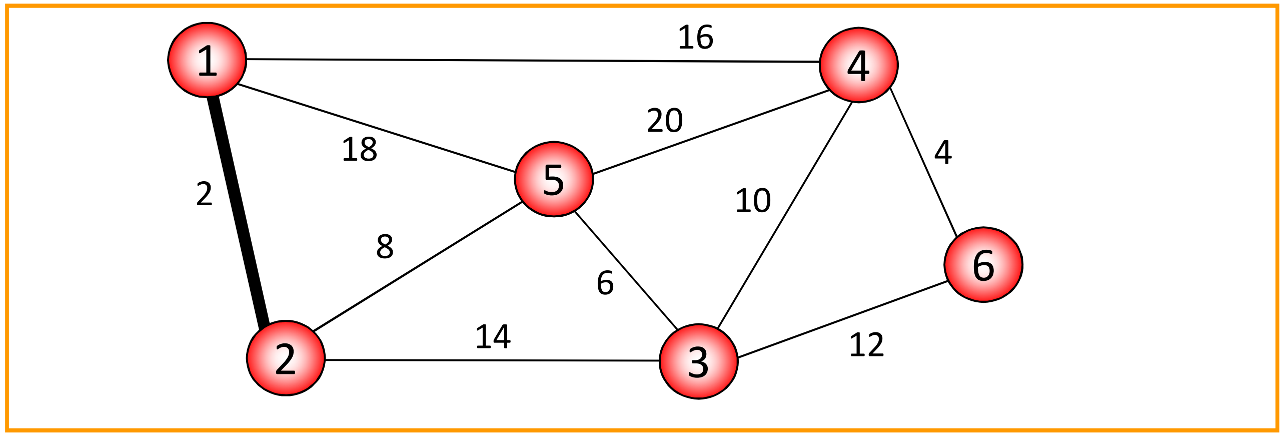
- Lemma 5.3.1.3 provides us with important knowledge for generating minimum spanning trees
- Based on these cognitions, scientific literature introduces two different procedures for calculating minimum spanning trees
  - The procedure of Prim
  - The procedure of Kruskal

The procedure of Prim

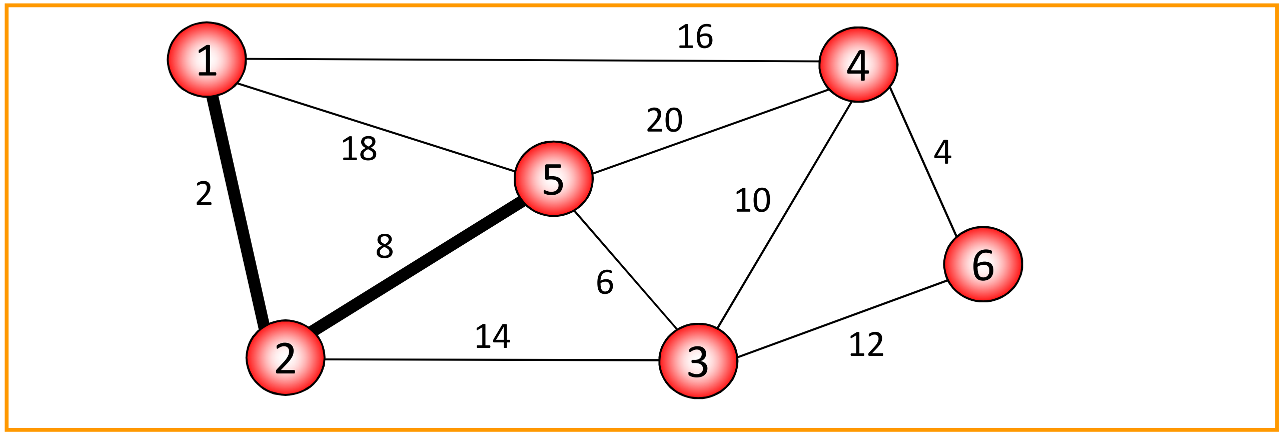
- Input: Network N=(V,E)
- This procedure generates a tree T step by step
- For this purpose, all edges are initially sorted according to their weight in non-increasing order
- The algorithm commences with the edge that has least costs and inserts it into the tree T. Break ties arbitrarily
- Subsequently, a least costs edge is inserted that is connected to a node of the already generated tree. Note that the other node of this edge does not belong to the tree before insertion. Again, break ties arbitrarily
- As soon as all nodes have become members of the generated tree, the algorithm terminates
- Output: Minimal spanning tree ST*(N)

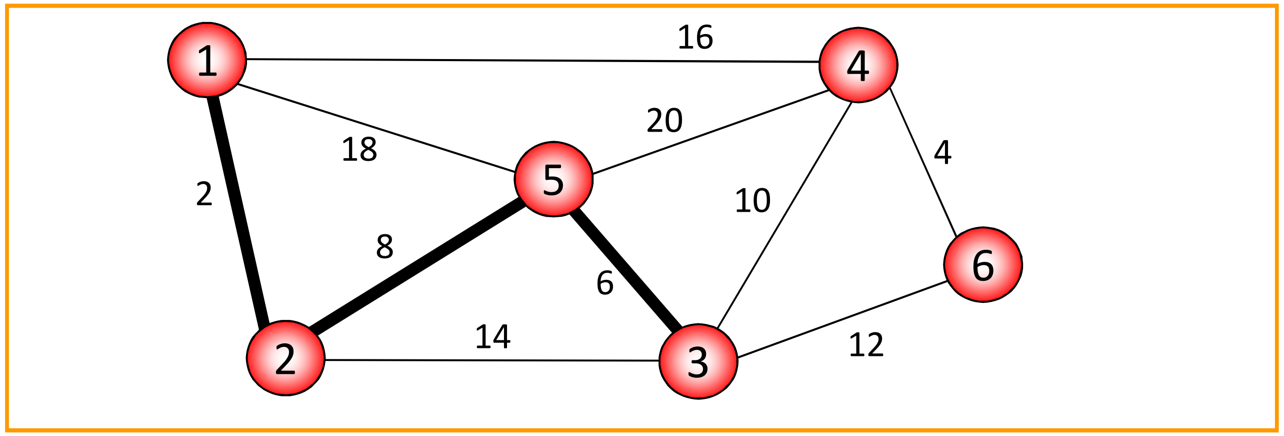
The procedure of Prim – Example



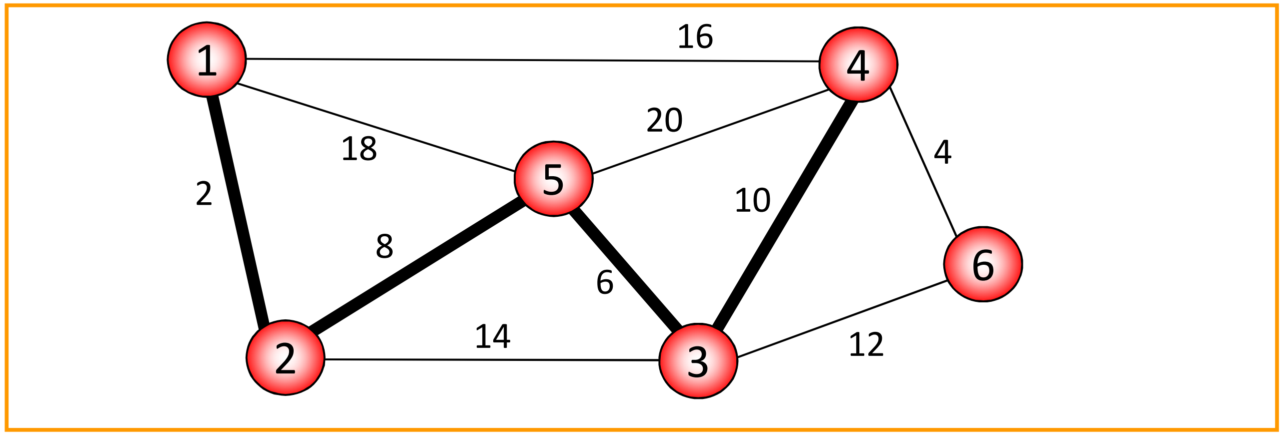


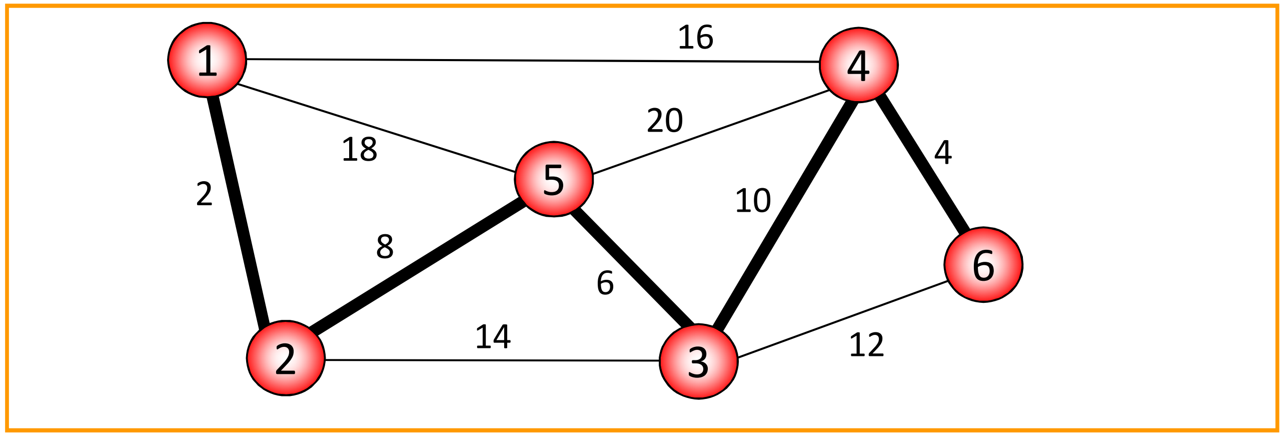
The procedure of Prim – Example





The procedure of Prim – Example





ST*(N) – Calculating its weight

- The total weight amounts to 2+8+6+10+4=30
- This spanning tree is optimal

The procedure of Prim: Correctness

- Clearly, the correctness of this procedure follows immediately by applying Lemma 5.3.1.3

The procedure of Prim: Complexity

- In order to execute the procedure of Prim efficiently, it can be implemented by making use of min-heaps
- This leads to the following program
- Prim with heaps:
  - Make a heap of values (vertex, edge, weight(edge))
    Initially (v,-,infinity) /* Current distance to tree */
  - Insert least cost edge into T (i.e., the connected nodes) and update all weights accordingly
  - While tree T has fewer than n vertices
    - Let (v,e,weight(e)) being the smallest weight in the heap
    - Remove (v,e,weight(e)) from the heap
    - Add v and e to tree T
    - For each edge f=(u,v)
      - If u is not already in T, find value (u,g,weight(g)) in heap
      - If weight(f)<weight(g), replace (u,g,weight(g)) with (u,f,weight(f))

The procedure of Prim: Complexity

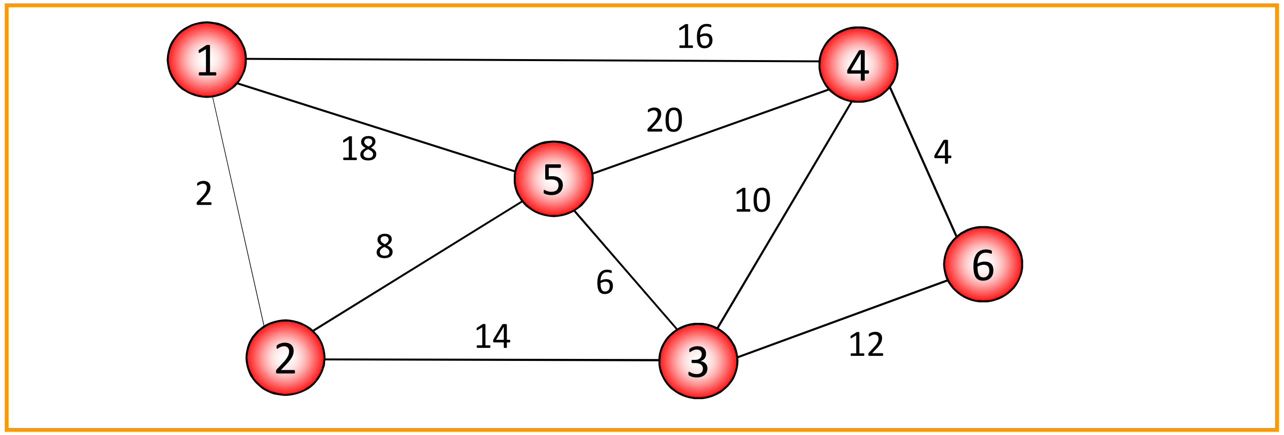
- Since update operations on heaps can be applied in time O(log n), we have O(m log n) steps for building the heap
- However, by using Fibonacci heaps, this is possible even in asymptotic time O(1), and therefore we obtain O(m) as the total running time for performing all updating operations
- Moreover, direct access for each node to its corresponding heap element that represents the closest connection to the tree is maintained
- Therefore, all in all, we have a running time of order O(m+n log n)

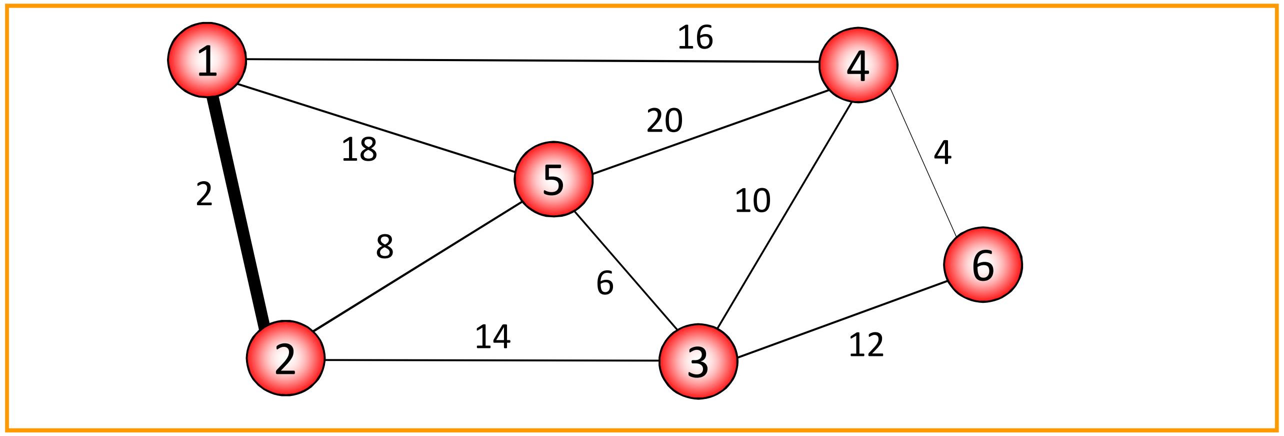
The procedure of Kruskal

- Input: Network N=(V,E)
- This procedure generates a sequence of sets of trees originally unconnected (i.e., forests)
- As soon as the forest becomes a tree, ST*(N) is generated

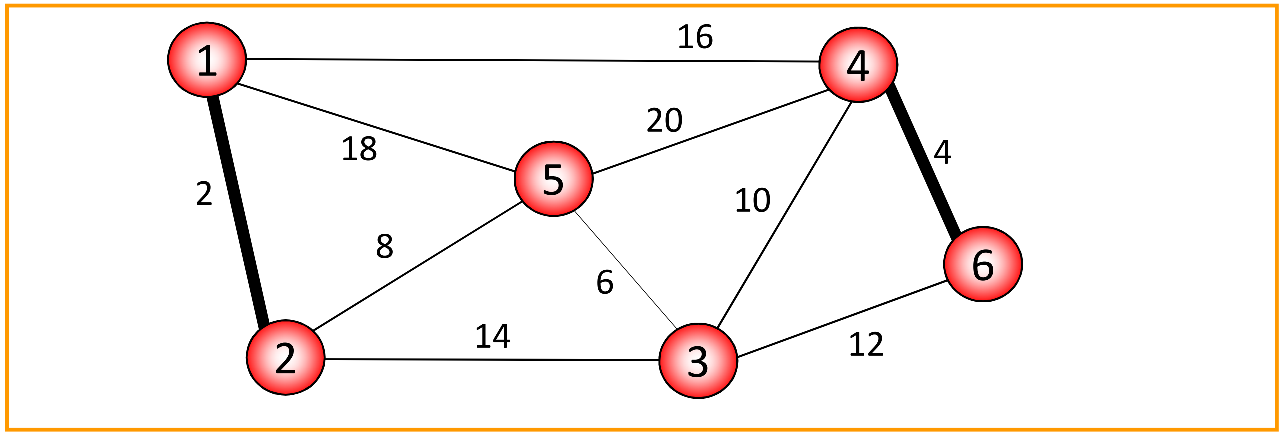
- Sort the edges in set E in increasing order
- Keep a subgraph S of N, initially empty
- For each edge e in sorted order
  - if the endpoints of e are disconnected in S
  - add e to S
- Output: S=ST*(N)

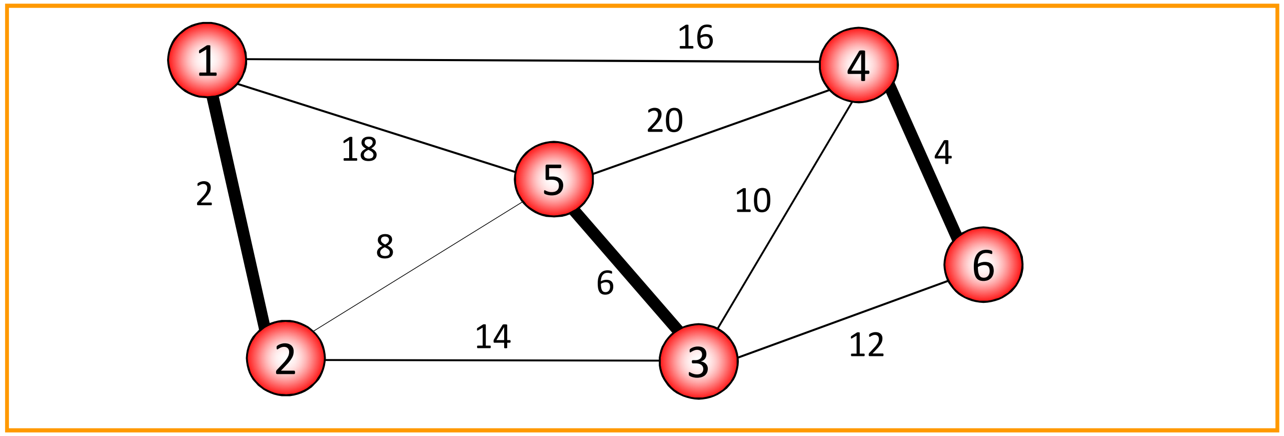
The procedure of Kruskal – Example



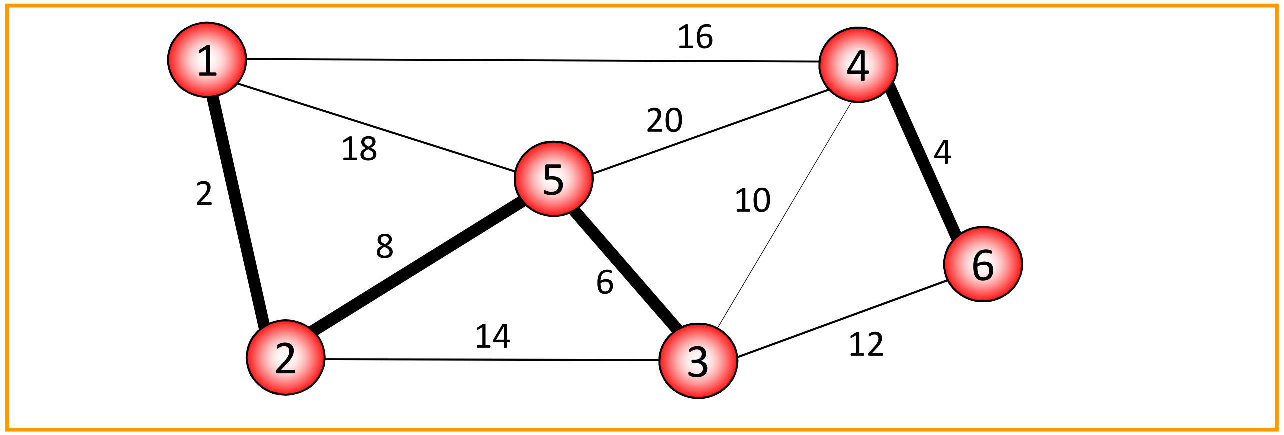


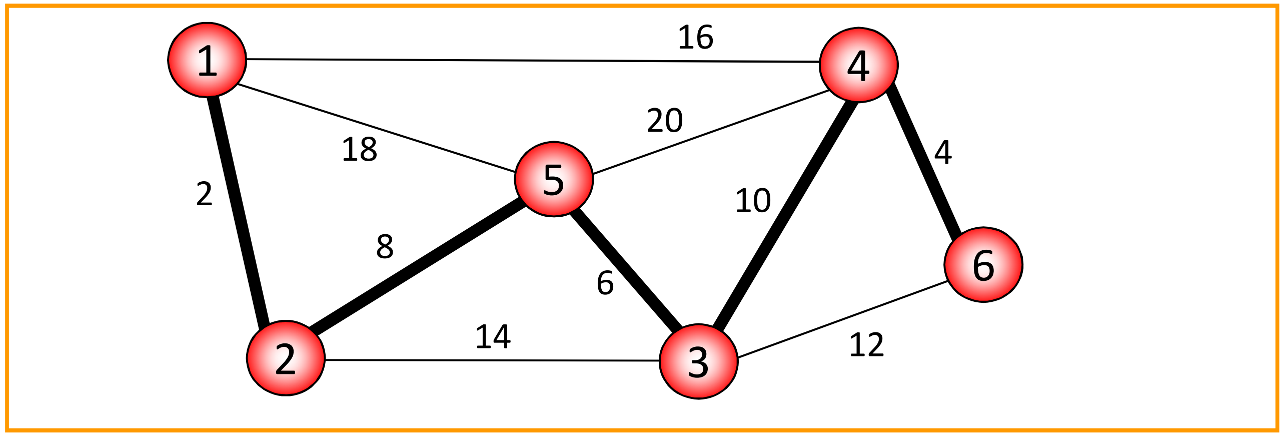
The procedure of Kruskal – Example





The procedure of Kruskal – Example





ST*(N) – Calculating its weight

- The total weight amounts to 2+8+6+10+4=30
- This spanning tree is optimal

The procedure of Kruskal: Correctness

- Clearly, the correctness of this procedure also follows immediately by applying Lemma 5.3.1.3

The procedure of Kruskal: Complexity

- Since all edges are sorted at the beginning, we obtain the running time for this step of order O(m log(m))
- Subsequently, we erase n-1 times the edge with lowest weight bridging two separated sets of nodes. Thus, all in all, we have a running time of order O(m)
- All in all, we have complexity O(m + m log m)
- Since m is of order $O(n^2)$, we obtain O(m log n) as the asymptotic running time

Observation

5.3.1.4 Lemma

Let N be a network with n nodes and symmetric distance matrix. Moreover, T is an optimal tour of the TSP. Additionally, T_0 is an optimal open tour in N between source s and destination t. Then, it holds that:*

$$\boxed{\begin{array}{l} (1)\ \ L\left(ST^{*}(N)\right) \le \left(1-\frac{1}{n}\right)\cdot L\left(T^{*}\right) \\ (2)\ \ L\left(ST^{*}(N)\right) \le L\left(T_0\right) \end{array}}$$

Proof of Lemma 5.3.1.4

- Since we can generate a spanning tree by erasing a single edge from an optimal cyclical TSP tour in a network N, ST*(N) is a lower bound for an open TSP tour
- Therefore, we erase the edge with maximal weight d_{max} from the cyclical TSP tour
- Therefore, it holds:

$$\boxed{L\left(ST^{*}(N)\right) \leq L\left(T^{*}\right) - d_{\max} \leq \left(1 - \frac{1}{n}\right) \cdot L\left(T^{*}\right)}$$

- Moreover, since the corresponding open tour is a spanning tree, the second proposition follows immediately

Generating a minimum 1-tree

5.3.1.5 Definition

A minimum 1-tree of an undirected graph N=(V,E) is a 1-tree with minimum total weight.

5.3.1.6 Algorithm

A minimum 1-tree of an undirected graph N=(V,E) is generated by the following two steps:

1. *Compute the minimum spanning tree S=ST*(N) of network N*
2. *Insert into S an edge with minimum weight of network N that does not belong to ST*(N)*

Output: 1-tree S

Correctness of Algorithm 5.3.1.6

- Clearly, the algorithm generates a 1-tree since *ST*(N)* is a tree while a single edge is added
- Hence, it remains to show that *S* (the outputted 1-tree) is a minimum 1-tree
- We prove this claim by contradiction
- Suppose 1-tree *T* is a minimum 1-tree that is different from *S*.
- Since *S* was built from a minimum spanning tree, we know that the tree directly proceeding adding the last edge *f* (the cycle-inducing edge) was minimal.
- Hence, the total weight of any spanning subtree of *T* is larger or equal to the total weight of *S* minus the weight of *f*

Correctness of Algorithm 5.3.1.6

- Moreover, we know that both 1-trees (*S* and *T*) comprise a single cycle
- *Case 1*: There is no edge in the cycle of *T* that does not belong to *S*. Hence, both cycles are identical. Then, we can erase edge *f* (the edge lastly added by Algorithm 5.3.1.6 (it completed the cycle)) from both 1-trees (results are the trees *S'* and *T'*) and know that the resulting graphs are spanning trees. Hence, it holds that $L(S')\leq L(T')$ and $L(S)=L(S')+L(f)\leq L(T')+L(f)=L(T)$
- *Case 2*: There is, at least, one edge, let say the edge *e*, in the cycle of *T* that does not belong to *S*. We erase *e* from *T* and get the spanning tree *T'*. It holds that $ST^*(N)\leq L(T')$. Moreover, since $ST^*(N)$ is a tree, adding *e* builds a 1-tree. So, the weight of the final edge added by Algorithm 5.3.1.6 is not larger since it was minimally chosen. Hence, *S* is a minimum 1-tree.

5.3.2 Held-Karp bound

- In what follows, we introduce a much tighter bound
- It bases on the cognitions just obtained, but extends this idea considerably
- Specifically, it improves this basic bound (that was depicted above) iteratively by applying a specific Lagrangian Relaxation combined with a subgradient method
- Therefore, in several iterations, obtained bounds are getting tighter
- Besides its technical specifics, focus is set to the basic ideas of the approach
- Basically, it may provide tight bounds to the TSP

Transforming the symmetric TSP

- Again, we commence with the following problem

Parameters

N: Number of nodes (customers)

$c_{i,j}\left(1 \leq i < j \leq N\right)$: Costs for using the edge $\left(i,j\right)$

$s \in \left\{1,\ldots,N\right\}$: Distinguished node in the network

$g_i \geq 0$: Positive cost multiplier for node $i \in \left\{1,\ldots,N\right\}$

Variables

$x_{i,j}\left(1 \leq i < j \leq N\right)$: Binary variable that is one if and only if the edge $\left(i,j\right)$ is used

Symmetric TSP – Restrictions

- Minimize

$$Z = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} x_{i,j} \cdot c_{i,j}$$

- Subject to

(1) $\forall j \in \{1,\ldots,N\} : \sum_{i<j} x_{i,j} + \sum_{k>j} x_{j,k} = 2$

(2) $\sum_{i,j \in \{1,\ldots,N\} \text{ with } i<j} x_{i,j} = n$

(3) The variable set $x_{i,j}$ defines a 1-tree

(4) $\forall i,j \in \{1,\ldots,N\} \wedge i < j : x_{i,j} \in \{0,1\}$

Modifying the problem

By identifying node *s*, we now obtain

- Minimize $$Z = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} x_{i,j} \cdot c_{i,j}$$
- Subject to

(1) $\forall j \in \{1,\ldots,N\}$ with $j \neq s : \sum_{i<j} x_{i,j} + \sum_{k>j} x_{j,k} = 2$

(2) $\sum_{i<s} x_{i,s} + \sum_{k>s} x_{s,k} = 2$

(3) $\sum_{i,j\in\{1,\ldots,N\} \text{ with } i<j} x_{i,j} = n$

(4) The variable set $x_{i,j}$ defines a 1-tree

(5) $\forall i,j \in \{1,\ldots,N\} \wedge i < j : x_{i,j} \in \{0,1\}$

Relaxing the problem

By relaxing hard restriction 1, we obtain

- Minimize $$L_g(x) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} x_{i,j} \cdot c_{i,j} + \sum_{i=1}^{N} g_i \cdot \left(\sum_{h=1}^{i-1} x_{h,i} + \sum_{j=i+1}^{N} x_{i,j} - 2 \right)$$
- Subject to

(1) $\sum_{i<s} x_{i,s} + \sum_{k>s} x_{s,k} = 2$

(2) $\sum_{i,j \in \{1,\ldots,N\} \text{ with } i<j} x_{i,j} = n$

(3) The variable set $x_{i,j}$ defines a 1-tree

(4) $\forall i,j \in \{1,\ldots,N\} \wedge i < j : x_{i,j} \in \{0,1\}$

…and transforming

- Thus, we obtain

$$L_g(x) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(c_{i,j} + g_i + g_j\right) \cdot x_{i,j} - 2 \cdot \sum_{i=1}^{N} g_i$$

(1) $\sum_{i<s} x_{i,s} + \sum_{k>s} x_{s,k} = 2$

(2) $\sum_{i,j \in \{1,\ldots,N\} \text{ with } i<j} x_{i,j} = n$

(3) The variable set $x_{i,j}$ defines a 1-tree

(4) $\forall i,j \in \{1,\ldots,N\} \wedge i < j : x_{i,j} \in \{0,1\}$

Concave objective function

- In contrast to the objective function of the Lagrangian Problem of the Knapsack problem (a convex function), in this case we obtain the following concave objective function

$$z(g) = Min_x L_g(x) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (c_{i,j} + g_i + g_j) \cdot x_{i,j} - 2 \cdot \sum_{i=1}^{N} g_i \ .$$

- By multiplying this objective function with „-1“ we obtain a convex function.
- This implies that the subgradient method works analogously on concave functions.

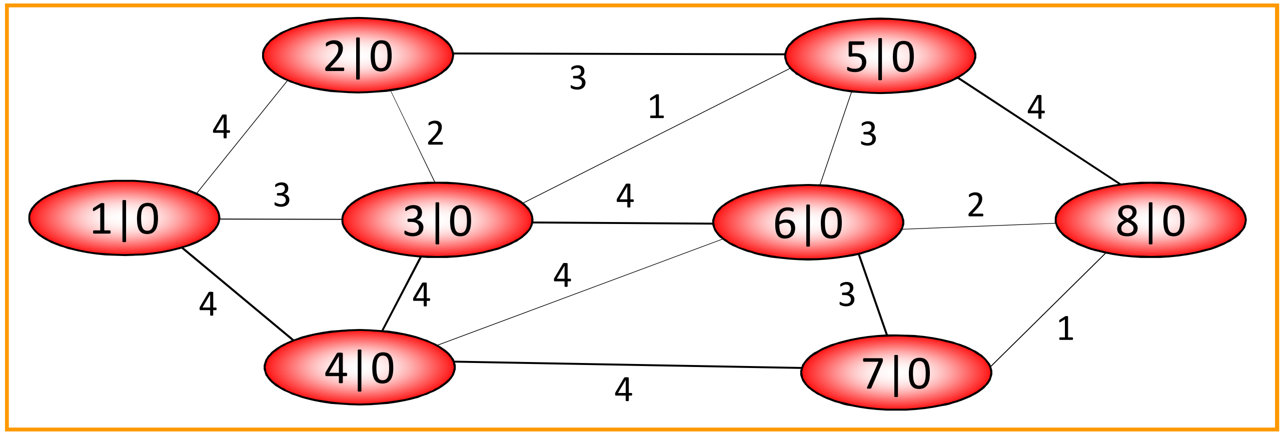
Observations

- By relaxing the hard restriction (1), we have obtained a 1-tree problem which can be easily solved by our known minimum spanning tree problems
- Moreover, node degrees unequal to 2 modify the objective function value
- Consequently, feasible TSP tours are not affected by the relaxed restriction
- Thus, by carefully modifying the multipliers, we pursue moving towards a cyclical tour, i.e., a TSP solution
- Therefore, our intention is to iteratively change the multipliers in order to force the spanning tree generation procedure to result in a TSP solution

Cognitions

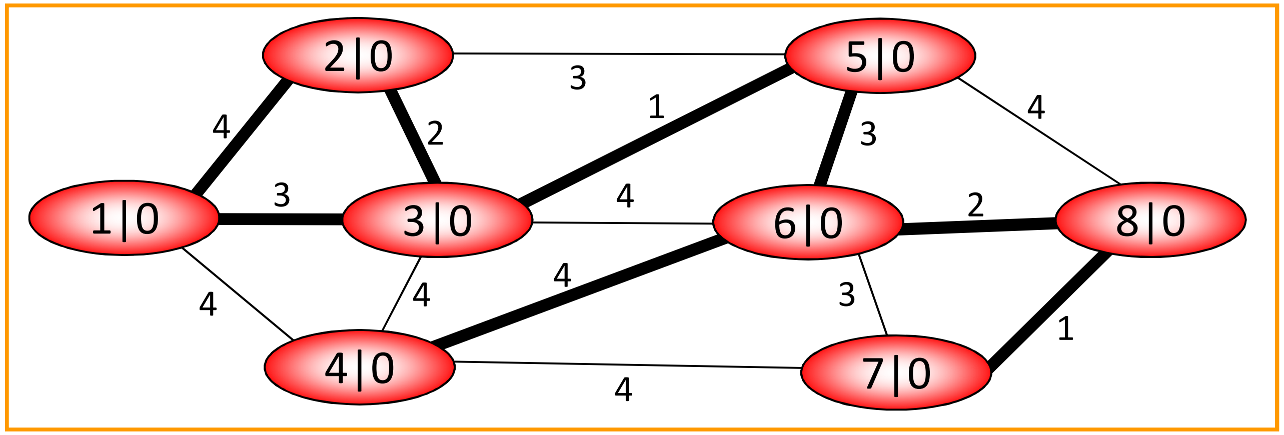
- Basically, we obtain $L_g(y) \leq Z(x)$ for optimal solutions to the relaxed problem
- This can be easily explained by the fact that each feasible TSP tour *t* is also a feasible solution to the relaxed problem and – due to the two neighbor restrictions – leads to identical costs, i.e., $L_g(t)=Z(t)$, for all *t*
- Hence, we have to generate suitable multipliers
- This can be done by adequate subgradient methods

Example



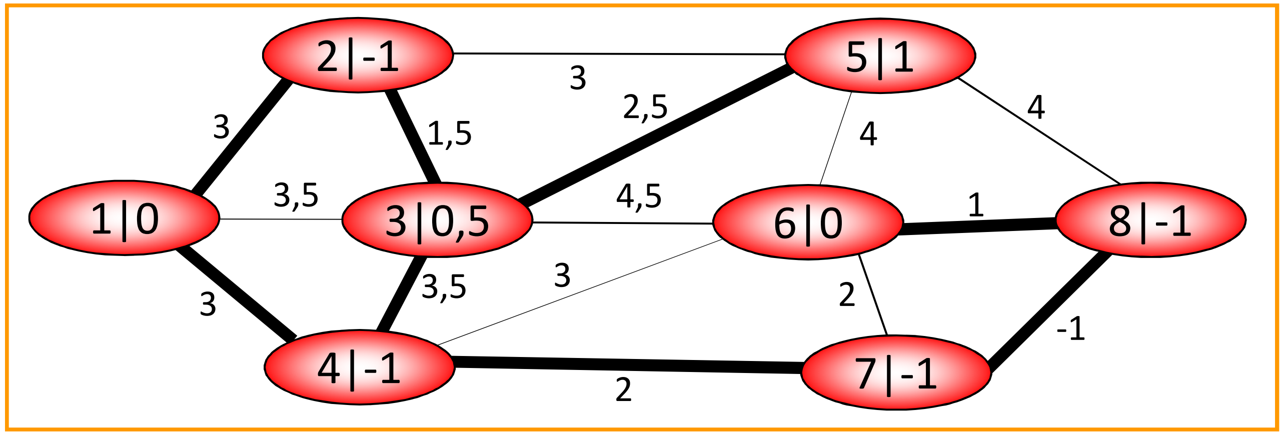
- We consider the network depicted above
- Basically, we can choose eight nodes as the distinguished node *s*
- Clearly, this choice has significant impact on the obtainable 1-tree bound

Example – Min 1-tree



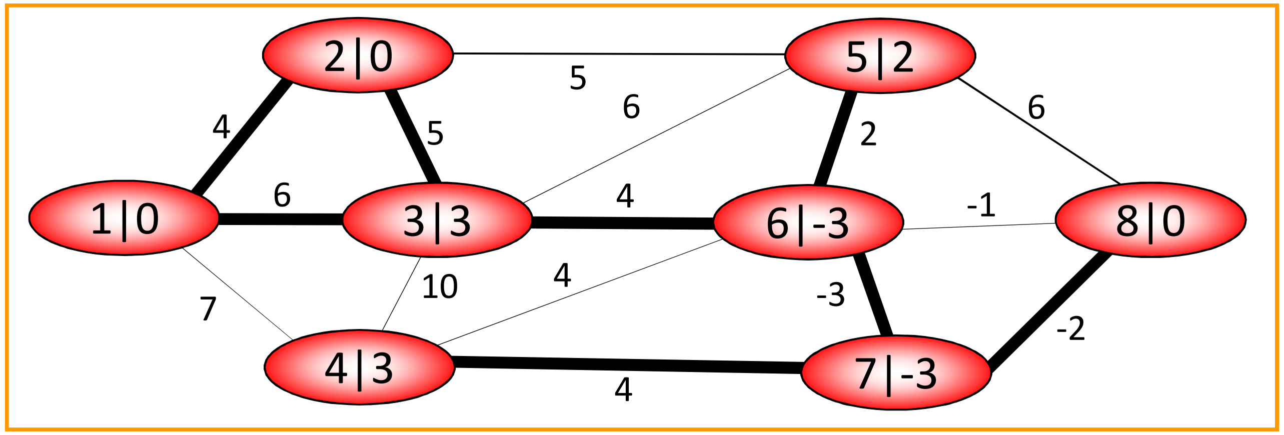
- If we choose node 1 as node *s*, we obtain a minimum 1-tree with total weight 20 (bold edges)
- Identical bounds are obtained by selecting nodes 3, 4, 5, or 6
- However, by selecting node 2, 7, or 8, we only obtain the total weight 19

Influence of multipliers



- By applying the multipliers (summation value is -2,5) that are defined above, we obtain a minimum 1-tree with total weight 15,5 (bold edges)
- Thus, by correcting the values, we obtain 15,5+5=20,5
- Hence, a new lower bound of 21 is obtained
- Please note that this does not necessarily work that smoothly
- The next slide shows a negative example

Multipliers – Negative result



- Here we obtain a minimal 1-tree with total weight 20
- Thus, since the total sum of multipliers sums up to 2, we obtain 20-4=16
- This is not promising since it decreases the bound

A subgradient method (or ascent method)

- Consequently, we have to learn how to modify the g-vector efficiently in order to tighten the lower bound, i.e., we want to find a g*-vector that fulfills

$$\boxed{Max_g Min_x L_g(x) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(c_{i,j} + g_i + g_j\right) \cdot x_{i,j} - 2 \cdot \sum_{i=1}^{N} g_i}$$

- Clearly, when we obtain a TSP tour for the first time, we have generated an optimal solution of our problem
- Such a feasible TSP tour can be identified by the fact that all nodes have a degree of 2
- Consequently, in this special case, multipliers have no impact on the objective function value

Notations

- In what follows, we make use of the following parameters

$T(g)$: minimal 1-tree of the Lagrangian problem defined by the multipliers $(g_1,...,g_n)$

$L_g^*(x)$: objective function value of the optimal solution to the Langrangian problem defined by the multipliers $(g_1,...,g_n)$

$d(T)$: node degree vector of the n nodes. Specifically, we obtain the degree of node i by $d_i(T)$

$d(T)-2$: node degree vector of the n nodes reduced by 2 for all values

δ_j : Step size in iteration j

Update of multipliers

- During the iterative calculation of the lower bound, the following update formula is applied in order to generate new multipliers
- Specifically, g^{j+1} is generated out of g^j

$$\boxed{g^{j+1} = g^j + \delta_j \cdot \left(d\left(T\left(g\right)\right) - 2\right)}$$

- Clearly, nodes with larger degrees are penalized by higher increases of the multipliers
- Node degrees of 2 are not affected at all
- Bonus is given to visit nodes with degree 1

Observation

- Clearly, this update handling keeps an identical sum of multipliers, i.e., $g_1+\ldots+g_n$=constant
- This can be easily explained by

$$\sum_{i=1}^{n} g_i^{j+1} = \sum_{i=1}^{n}\left(g_i^j + \delta_j \cdot \left(d_i\left(T\left(g\right)\right) - 2\right)\right)$$

$$= \sum_{i=1}^{n} g_i^j + \sum_{i=1}^{n} \delta_j \cdot d_i\left(T\left(g\right)\right) - \delta_j \cdot \sum_{i=1}^{n} 2$$

Since a 1-tree comprises just n edges, we obtain a total node degree of $2 \cdot n$

$$= \sum_{i=1}^{n} g_i^j + \delta_j \cdot \sum_{i=1}^{n} d_i\left(T\left(g\right)\right) - \delta_j \cdot 2 \cdot n$$

$$= \sum_{i=1}^{n} g_i^j + \delta_j \cdot 2 \cdot n - \delta_j \cdot 2 \cdot n = \sum_{i=1}^{n} g_i^j$$

Update of step size δ_j

- In order to obtain the maximum lower bound, i.e., in order to find the optimal multipliers, Held et al. propose a step size update that complies with the rules defined by Polyak
- These rules are basically

$$\boxed{\lim_{j\to\infty} \delta_j = 0 \quad \text{and} \quad \sum_{j=0}^{\infty} \delta_j = \infty}$$

Update of step size

- Held et al. (1974) propose the following update formula

$$\boxed{\delta_j = \gamma \cdot \frac{Z^{up} - L_g(x)}{\sum_{i=1}^{n}\left(d_i\left(T(g)\right) - 2\right)^2}}$$

- Starting value is *γ*=2
- This value is kept for *2n* iterations
- Subsequently, *γ* is reduced after *n*, *n/2*, *n/4*,... iterations by applying a reduction factor *λ*

Extended versions of the bound

- Empirical experiments underline that the bound can become very tight (i.e., close to an optimal solution), and therefore should be integrated into enumeration processes
- However, several authors propose specific extensions
- Particularly, parameter settings are modified
- Specifically, it has been observed that specific nodes cycle between a node degree of 1 and >2
- Therefore, better results have been obtained by including the node degree difference of the 1-tree considered before, i.e., by considering the last two 1-trees
- For instance, different approaches are introduced by Smith and Thompson (1977) and Reinelt (1994)

5.3.3 The approach of Reinelt (1994)

- Extends the Held-Karp bound
- It iteratively solves 1-tree problems and terminates
  - if a TSP tour is obtained
  - the generated bound is large enough (depends on the application)
  - a maximum number of iterations T has been executed
- In the approach, the following update formulas are used

$$g_i^{j+1} = \begin{cases} g_i^j + \delta_j \cdot \left(0.7 \cdot \left(d_i\left(T\left(g^j\right)\right) - 2\right) + 0.3 \cdot \left(d_i\left(T\left(g^{j-1}\right)\right) - 2\right)\right) & \text{if } j > 1 \\ \left(1 + \delta_1\right) \cdot d_i\left(T\left(g^1\right)\right) & \text{otherwise} \end{cases}$$

with $g_i^1 = 0, \forall i \in \{1,\ldots,n\}$, $T \in [100,1000]$, $\lambda \in [0.99,\ldots,0.999]$, and

$$\delta_{j+1} = \begin{cases} \in [0.5,\ldots,1.0] & \text{if j=0} \\ \lambda \cdot \delta_j & \text{otherwise} \end{cases}$$

A Branch&Bound approach with LR

- Based on the bound calculation, we introduce a second Branch&Bound approach
- It considers B&B nodes in FIFO manner, i.e., leafs of the B&B tree are considered in sequence of their occurrence
- It directly uses the generated 1-tree in order to select a branching variable
- Basically, we know that a 1-tree has some nodes with higher node degree than 2
- Hence, in the current B&B node, we are looking for a node in the generated 1-tree with minimal node degree larger than 2
- Note that at least one of the incoming edges has to be forbidden in order to result in an optimal TSP tour

Structure of the approach

- At first, we generate an initial solution by applying a simple heuristic

  → We make use of the nearest neighbor heuristic
- Then, after generating a first bound in the root node, we consider the oldest leaf
- Here, we consider the generated 1-tree with maximal bound
- We take a node with minimal degree larger than 2 and branch accordingly
- I.e., we forbid all edges one by one in order to reduce the degree of this node
- In what follows, we consider a simple 10-node example

10 Nodes Euclidean Problem

Node	1	2	3	4	5
X-Coordinate	18	62	71	28	77
Y-Coordinate	84	71	77	72	14
Node	**6**	**7**	**8**	**9**	**10**
X-Coordinate	79	78	4	62	68
Y-Coordinate	6	11	100	63	48

Distance matrix

	1	2	3	4	5	6	7	8	9	10
1		45.88	53.46	15.62	91.55	99.02	94.49	21.26	48.75	61.61
2			10.82	34.01	58.94	67.19	62.1	64.85	8	23.77
3				43.29	63.29	71.45	66.37	70.84	16.64	29.15
4					75.93	83.41	78.87	36.88	35.17	46.65
5						8.25	3.16	112.81	51.24	35.17
6							5.1	120.25	59.48	43.42
7								115.75	54.41	38.33
8									68.8	82.46
9										16.16
10										

Nearest neighbor application

- An initial solution is generated by applying the nearest neighbor heuristic
- We commence this process with node 6
- We obtain a first TSP tour with length **UB=278.83**

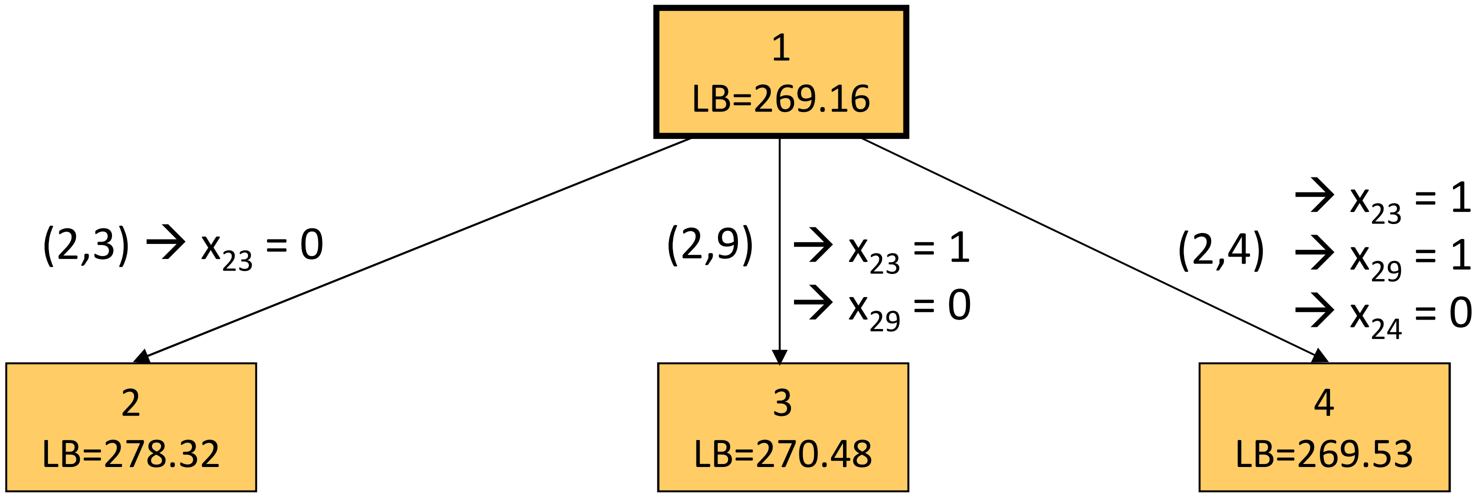
Starting bound: First 1-tree generation

- Depending on λ, we obtain different initial lower bound values
- Moreover, we set δ_1=1.0 and T=300

λ=	0.95	0.96	0.97	0.98	0.99	0.998
LB=	221.34	225.5	232.45	245.18	269.16	269.54

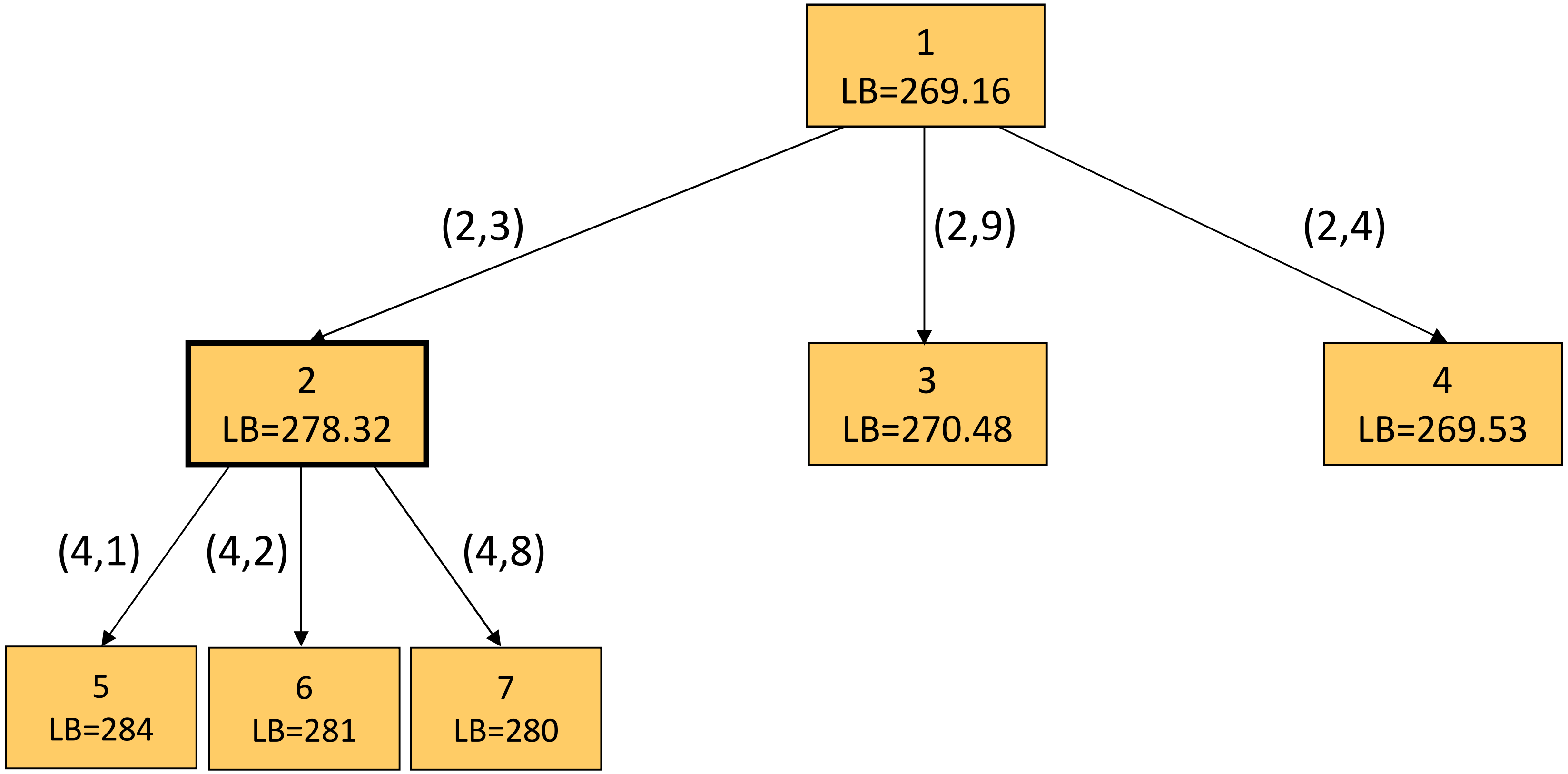
- In what follows, we apply λ=0.99
- After 300 iterations, we obtain a 1-tree with cost 269.16, i.e., LB=269.16. In this 1-tree, node 2 has degree 3. The used edges connecting node 2 are (2,3), (2,9), and (2,4).

First branching step


1
LB=269.16
(2,3) → $x_{23} = 0$
(2,9) → $x_{23} = 1$ → $x_{29} = 0$
(2,4) → $x_{23} = 1$ → $x_{29} = 1$ → $x_{24} = 0$
2
LB=278.32
3
LB=270.48
4
LB=269.53

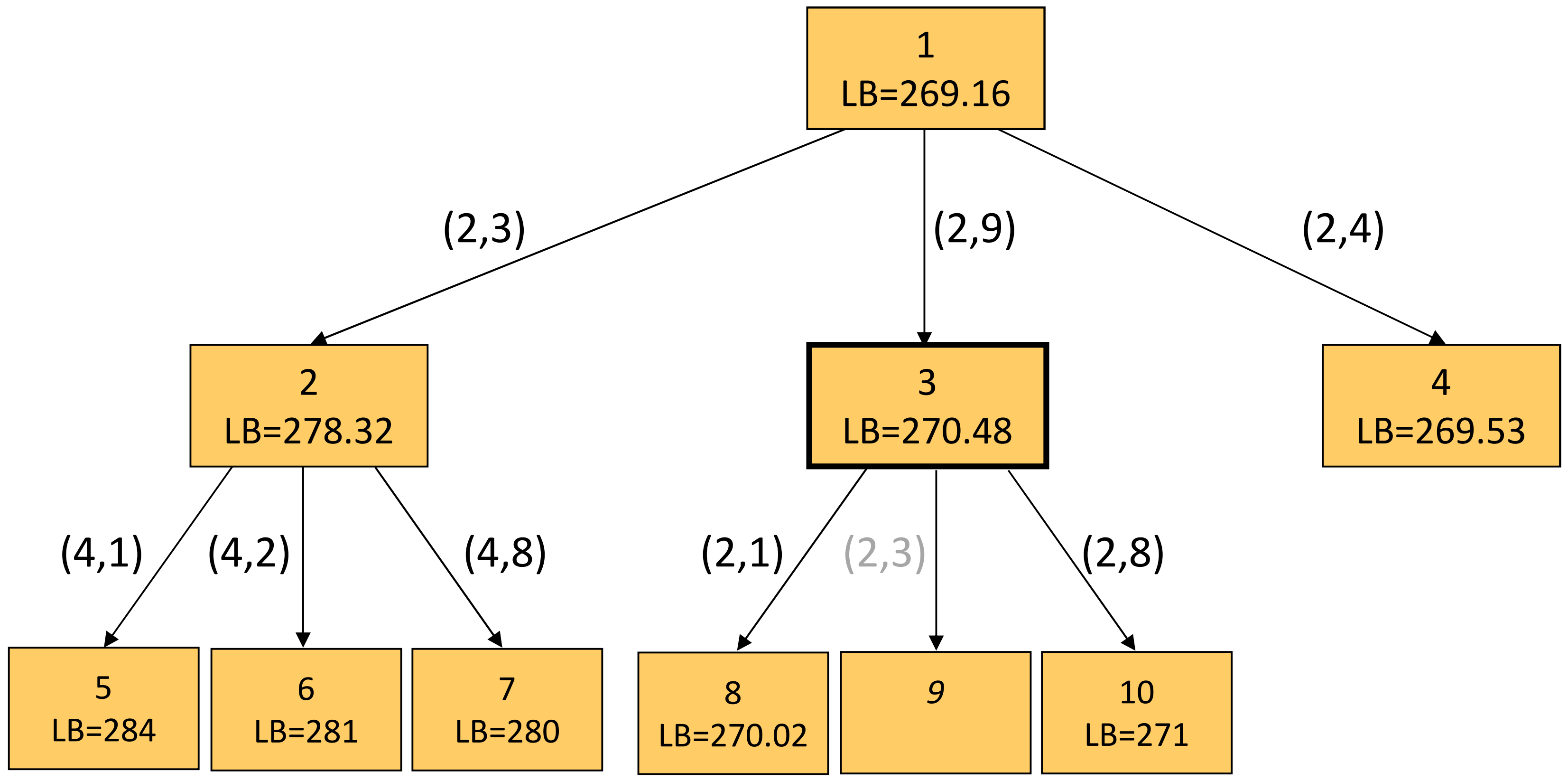
- Unfortunately, we cannot fathom any node since current UB = 278.83 > LB applies for all nodes
- Thus, we proceed with node 2

The next branching step with node 2



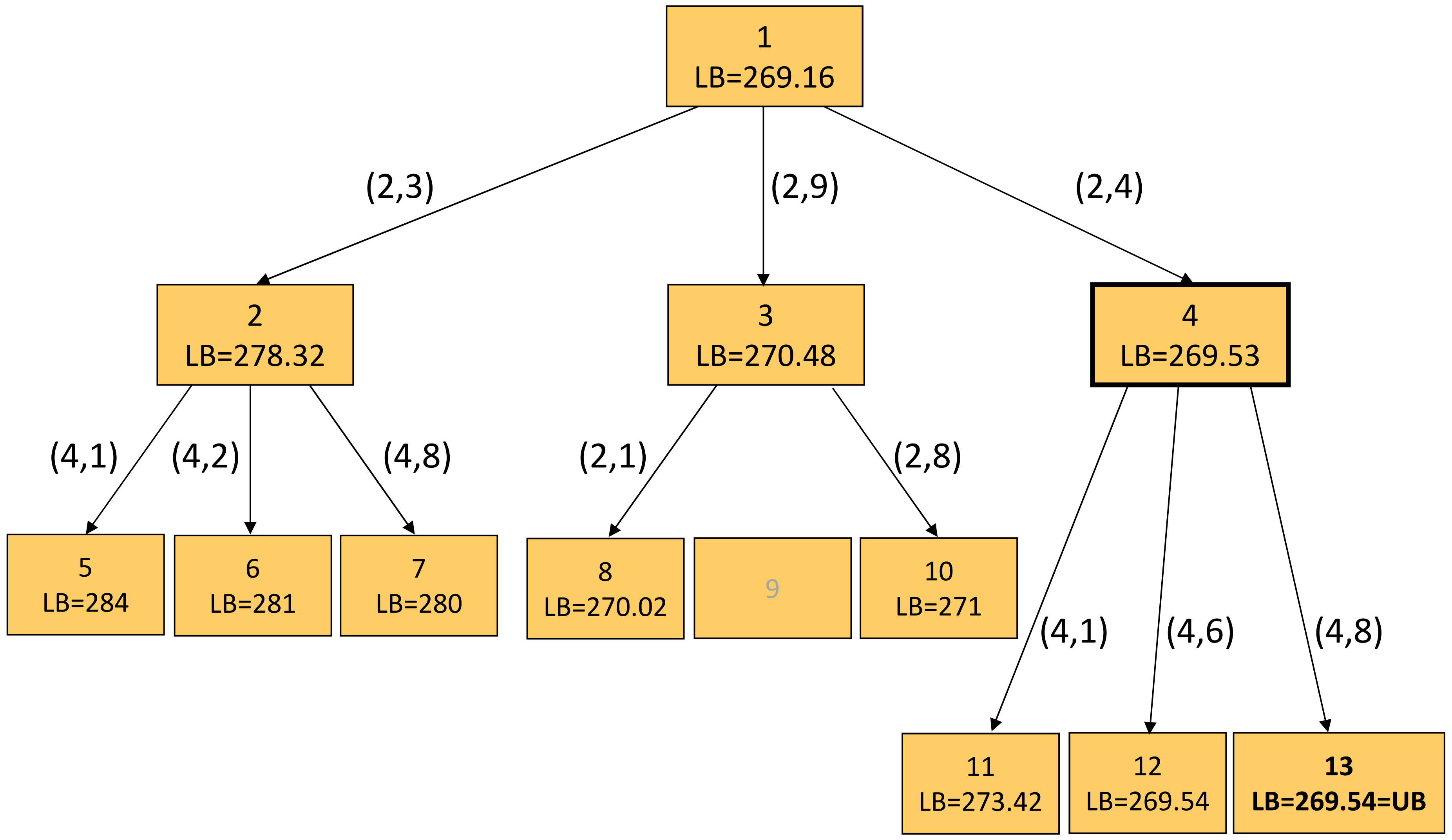
- Now, we can fathom all new nodes since current UB = 278.83 < LB applies

Branching node 3



- Node 9 is NOT generated since we have set $x_{23} = 1$ at the father node and therefore in this subtree

Branching node 4



Branching node 4

- Fortunately, node 13 provides us with a new TSP tour
- Thus, we can improve UB to 269.54
- Consequently, we can fathom the nodes 8, 9, 10, 11, and 12
- Thus, there is no active leaf available anymore
- Node 13 determines an optimal solution
- Optimal TSP tour is 1-4-6-7-5-10-9-2-3-8-1
- Tour length is 269.54

Observations

- Switching to best-first enumeration rule can substantially reduce the computational effort
- For instance, in the example, the nodes 5, 6, 7, 8, 9, and 10 would not have been generated
- Furthermore, due to the subgradient method, nodes in lower levels do not necessarily provide improved bounds (they may be even lower (clearly, in this case they inherit the LB value from their father))

References of Section 5

- Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms and Applications. Prentice Hall. ISBN 0-13-617549-X, 1993.
- Held, M., Karp, R.: The Traveling Salesman Problem and Minimum Spanning Trees. Operations Research 18:1138-1162, 1970.
- Klose, A.: Standortplanung in distributiven Systemen. Modelle, Methoden, Anwendungen. Physica-Verlag, Heidelberg, 2001. ISBN 3-7908-1410-5.
- Leiserson, C., Demaine, E.: Introduction to Algorithms. The Mit Press; Auflage: 3rd edition. Student. ISBN-0262533057, ISBN-13: 978-0262533058, 2009.

References of Section 5

- Polyak, B. T.: A general method of solving extremal problems, Soviet Mathematics Doklady 8:593-597, 1967.
- Polyak, B. T.: Minimization of unsmooth functionals, U.S.S.R. Computational Mathematics and Mathematical Physics 14-29, 1969.
- Reinelt, Gerhard: The Traveling Salesman: Computational Solutions for TSP Applications. Lecture Notes in Computer Science 840, Springer-Verlag, Berlin, 1994.
- Vahrenkamp, R.; Mattfeld, D.C.: Logistiknetzwerke - Modelle für Standortwahl und Tourenplanung. 2nd edition. Springer Gabler, Wiesbaden, 2014.