Operations Research – Zusammenfassung

1 Lineare Programme

Wie kann man dieses Problem mathematisch beschreiben? Wonach wird gefragt?

- Wie könnte eine Antwort aussehen? → dementsprechend **Variablen** definieren
- Welche Informationen sind gegeben? Wie passen sie zu den Variablen? → **Parameter**
- Wie lautet das Optimierungskriterium? Wie kann man die Güte der Lösung beurteilen? → **Zielfunktion** als eine Funktion der Variablen
- Welche Bedingungen muss eine Lösung erfüllen? → **Nebenbedingungen** in Form (Un-)Gleichungen

Beispiel

Firma XY produziert in Basel und Zürich ein bestimmtes Produkt. An einem Tag werden 14 Tonnen in Basel und 16 Tonnen in Zürich produziert. Die Firma hat drei Abnehmer, die sich in Luzern, Chur und Winterthur befinden. Der tägliche Bedarf der Abnehmer ist 9 Tonnen für Luzern, 10 Tonnen für Chur und 11 Tonnen für Winterthur. Firma XY möchte ihre Transportkosten minimieren, deshalb hat sie die Transportpreise pro Tonne auf allen Verbindungen erhoben. Die Ergebnisse sind hier:

	Luzern	Chur	Winterthur
Basel	11	4	5
Zürich	5	4	7

Wie soll die Firma ihre täglichen Transporte planen, damit die täglichen Transportkosten möglichst klein sind?

- **Variablen definieren:** X_{BL} = Basel – Luzern, X_{BC} = Basel – Chur, X_{BW} = Basel – Winti, X_{ZL} = Zürich – Luzern, X_{ZC} = Zürich – Chur, X_{ZW} = Zürich – Winti,
- **Gleichungen aufstellen:** wollen Minimum von: $11X_{BL} + 4X_{BC} + 5X_{BW} + 5X_{ZL} + 4X_{ZC} + 7X_{ZW}$
- Alle $X_{BL}, X_{BC}, X_{BW}, X_{ZL}, X_{ZC}, X_{ZW} \geq 0$
- $X_{BL} + X_{ZL} = 9$, $X_{BC} + X_{ZC} = 10$, $X_{BW} + X_{ZW} = 11$
- $X_{BL} + X_{BC} + X_{BW} \leq 14$, $X_{ZL} + X_{ZC} + X_{ZW} \leq 16$

Allgemeines Optimierungsproblem

- $\prod: \min\{f(\boldsymbol{x}) | \boldsymbol{x} \in S\}$, wobei $S \subseteq \mathbb{R}^n$
- Entscheidungsvariabeln: $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$
- Zulässige Menge: $S \subseteq \mathbb{R}^n$
- Zielfunktion: $f: S \to \mathbb{R}$
- Zulässige Lösung: $\boldsymbol{x} \in S$
- Optimale Lösung: $\boldsymbol{x}^* \in S$ sodass $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ für alle $\boldsymbol{x} \in S$
- Optimum, optimaler Zielfunktionswert: $f(\boldsymbol{x}^*)$

Definition: Lineare Funktion
Eine **Funktion** $f: \mathbb{R}^n \to \mathbb{R}$ heisst **linear**, falls $f(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{x}$, wobei $\boldsymbol{a} \in \mathbb{R}^n$

Definition: Lineare Nebenbedingung
Nebenbedingung ist **linear**, falls sie eine der folgenden Formen hat und f eine **lineare Funktion** ist, $\beta \in \mathbb{R}$:

- $f(\boldsymbol{x}) \leq \beta$
- $f(\boldsymbol{x}) \geq \beta$
- $f(\boldsymbol{x}) = \beta$

Definition: Lineares Programm (LP)
Das Problem der Minimierung/Maximierung einer linearen Funktion, so dass eine **endliche Anzahl** von **linearen Nebenbedingungen** erfüllt ist, heisst **lineares Programm**.

Ausgewählte Formen von LP's

Standardform: $\mathbf{max}\{\boldsymbol{c^T x} | \boldsymbol{Ax = b}, \boldsymbol{x \geq 0}\}$
Kanonische Maximierungsform: $\mathbf{max}\{\boldsymbol{c^T x} | \boldsymbol{Ax \leq b}, \boldsymbol{x \geq 0}\}$
Kanonische Minimierungsform: $\mathbf{min}\{\boldsymbol{c^T x} | \boldsymbol{Ax \geq b}, \boldsymbol{x \geq 0}\}$

Wechsel zwischen verschiedenen Formen

Ungleichung $\leq$ zu Ungleichung $\geq$: $a^T x \leq b \Leftrightarrow -a^T x \geq -b$
Gleichung zu Ungleichungen: $a^T x = b \to a^T x \leq b, a^T x \geq b \to a^T x \leq b, -a^T x \leq -b$
Ungleichungen zu Gleichungen: $a^T x \leq b \to a^T x + s = b, s \geq 0$
$a^T x \geq b \to a^T x - s = b, s \geq 0$
Nichtpositive Variable zur nichtnegativen Variable: $x_j \leq 0 \to \overline{x_j} := -x_j, \overline{x_j} \geq 0$
Freie Variable zu nichtnegativen Variablen: x_j frei $\to x_j = x_j^+ - x_j^-, x_j^+, x_j^- \geq 0$

Besitzt jedes LP eine optimale Lösung?

Betrachten das LP $\max\{c^T x | x \in P\}$ wobei P die **zulässige Menge** bezeichnet ($\emptyset$ = die leere Menge):

- **Zulässig**, falls $P \neq \emptyset$
- **Unzulässig,** falls $P = \emptyset$ (es gibt keine Lösung)
- **Unbeschränkt**, falls es für jede Zahl $\alpha \in \mathbb{R}$ eine zulässige Lösung $x^\alpha \in P$ gibt mit $c^T x^\alpha \geq \alpha$ (im Fall Max.)

Lösbarkeit von LP's

Jedes LP erfüllt genau eine der folgenden Möglichkeiten:

- Das LP ist **unzulässig**
- Das LP ist **unbeschränkt**
- Das LP besitzt genau **eine** optimale Lösung
- Das LP besitzt **unendlich** viele optimale Lösungen

Es sei ein LP mit der zulässigen Menge P gegeben. Zusätzlich gelte folgendes:
i) Das Polyeder P besitzt **mindestens eine Ecke** ii) Das LP besitzt **mindestens eine** optimale **Lösung**
Dann ist **mindestens eine Ecke** des Polyeders P **optimal**.

Lösungsablauf / Schema für kanonische Optimierungsformen und Beispielaufgaben

1. Ist Zielfunktion gemäss Aufgabenstellung zu maximieren/minimieren? Stimmt dies mit Zielfunktion überein? Wenn Werte nicht gleich, ist bei Zielfunktion respektive Vektor c einen Vorzeichenwechsel vorzunehmen!
2. Ungleichungen nach vorgeschriebener Form (bspw. $Ax \leq b$) umstellen (Vorzeichenwechsel).
3. Falls Gleichungen vorhanden, diese durch 2 Ungleichungen mit unterschiedlichen Vorzeichen austauschen!
4. Falls in Aufgabe $x \geq 0$ definiert ist, ist dem via Einfügen von einer Einheitsmatrix in A gerecht zu werden. **Achtung**, falls Form $Ax \leq b$ mit min vorgeschrieben ist, müssen alle Einträge v. Einheitsmatrix negativ sein!

Definieren Sie Variablenvektor x, die Parametervektoren b, c und die Matrix A, sodass die LP's in Form $\max c^T x$, $Ax \leq b$ und $x \geq 0$ geschrieben werden können:

$$\begin{pmatrix} \max 2x_1 + x_2 + x_3 \\ 3x_1 + x_2 + 2x_3 \leq 2 \\ x_1 + x_2 + 3x_3 \leq 5 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}$$

Lösung: Nebenbedingungen nach vorgeschriebener Form $Ax \leq b$ umstellen!

(Form stimmt hier bereits) $x^T = (x_1 \quad x_2 \quad x_3)$, $A = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 1 & 3 \end{pmatrix}$, $b = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$, $c^T = (2 \quad 1 \quad 1)$

Definieren Sie Variablenvektor x, die Parametervektoren b, c und die Matrix A, sodass die LP's in Form $\max c^T x$, $Ax \leq b$ und $x \geq 0$ geschrieben werden können:

$$\begin{pmatrix} \max x_1 - x_2 + 2x_3 + x_4 \\ 3x_1 + x_2 - x_3 + 2x_4 \leq 5 \\ x_1 + 2x_2 + x_3 - 2x_4 \geq 4 \\ -x_2 + 5x_4 \geq -2 \\ x_1, x_2, x_3, x_4 \geq 0 \end{pmatrix}$$

Lösung: Nebenbedingungen nach vorgeschriebener Form $Ax \leq b$ umstellen!

$$\begin{pmatrix} \max x_1 - x_2 + 2x_3 + x_4 \\ 3x_1 + x_2 - x_3 + 2x_4 \leq 5 \\ -x_1 - 2x_2 - x_3 + 2x_4 \leq -4 \\ x_2 - 5x_4 \leq 2 \\ x_1, x_2, x_3, x_4 \geq 0 \end{pmatrix} \rightarrow x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \quad A = \begin{pmatrix} 3 & 1 & -1 & 2 \\ -1 & -2 & -1 & 2 \\ 0 & 1 & 0 & -5 \end{pmatrix}, \quad b = \begin{pmatrix} 5 \\ -4 \\ 2 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ -1 \\ 2 \\ 1 \end{pmatrix}$$

Definieren Sie Variablenvektor x, die Parametervektoren b, c und die Matrix A, sodass die LP's in Form $\min c^T x$, $Ax \geq b$ und $x \geq 0$ geschrieben werden können:

$$\begin{pmatrix} \max 2x_1 + x_2 + x_3 \\ 3x_1 + x_2 + 2x_3 \leq 2 \\ x_1 + x_2 + 3x_3 = 5 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}$$

Lösung: Nebenbedingungen nach vorgeschriebener Form $Ax \geq b$ umstellen!

$$\begin{pmatrix} \min -2x_1 - x_2 - x_3 \\ -3x_1 - x_2 - 2x_3 \geq -2 \\ x_1 + x_2 + 3x_3 \geq 5 \\ -x_1 - x_2 - 3x_3 \geq -5 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}$$

LP in Form min… → Max Funktion mit -1 multiplizieren

$$\rightarrow x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, A = \begin{pmatrix} -3 & -1 & -2 \\ 1 & 1 & 3 \\ -1 & -1 & -3 \end{pmatrix}, b = \begin{pmatrix} -2 \\ 5 \\ -5 \end{pmatrix}, c = \begin{pmatrix} -2 \\ -1 \\ -1 \end{pmatrix}$$

Definieren Sie Variablenvektor x, die Parametervektoren b, c und die Matrix A, sodass die LP's in Form $\max c^T x$, $Ax \geq b$ geschrieben werden können:

$$\begin{pmatrix} \max 2x_1 + x_2 + x_3 \\ 3x_1 + x_2 + 2x_3 \leq 2 \\ x_1 + x_2 + 3x_3 \leq 5 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}$$

Lösung: Nebenbedingungen nach vorgeschriebener Form $Ax \geq b$ umstellen!

Achtung! Keine Angabe wie oben von $x \geq 0$!

$$\begin{pmatrix} -3x_1 - x_2 - 2x_3 \geq -2 \\ -x_1 - x_2 - 3x_3 \geq -5 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \end{pmatrix} \rightarrow x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad A = \begin{pmatrix} -3 & -1 & -2 \\ -1 & -1 & -3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} -2 \\ -5 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad c = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

Definieren Sie Variablenvektor x, die Parametervektoren b, c und die Matrix A, sodass die LP's in Form $\min c^T x$, $Ax \leq b$ geschrieben werden können:

$$\begin{pmatrix} \max x_1 - x_2 + 2x_3 + x_4 \\ 3x_1 + x_2 - x_3 + 2x_4 \leq 5 \\ x_1 + 2x_2 + x_3 - 2x_4 = 4 \\ -x_2 + 5x_4 \geq -2 \\ x_1, x_2, x_3, x_4 \geq 0 \end{pmatrix}$$

Lösung: Nebenbedingungen nach vorgeschriebener Form $Ax \leq b$ umstellen!

LP in Form min… → Max Funktion mit -1 multiplizieren

Achtung! Keine Angabe wie oben von $x \geq 0$!

$$\begin{pmatrix} \min -x_1 + x_2 - 2x_3 - x_4 \\ 3x_1 + x_2 - x_3 + 2x_4 \leq 5 \\ x_1 + 2x_2 + x_3 - 2x_4 \leq 4 \\ -x_1 - 2x_2 - x_3 + 2x_4 \leq -4 \\ x_2 - 5x_4 \leq 2 \\ -x_1 \leq 0 \\ -x_2 \leq 0 \\ -x_3 \leq 0 \\ -x_4 \leq 0 \end{pmatrix} \rightarrow x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, A = \begin{pmatrix} 3 & 1 & -1 & 2 \\ 1 & 2 & 1 & -2 \\ -1 & -2 & -1 & 2 \\ 0 & 1 & 0 & -5 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, b = \begin{pmatrix} 5 \\ 4 \\ -4 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, c = \begin{pmatrix} -1 \\ 1 \\ -2 \\ -1 \end{pmatrix}$$

LP mit reellen Zahlen

Sollte ein LP auftreten, indem Variablen der Zielfunktion alle reellen Zahlen (also auch negative) umfassen, muss je nachdem, ob sie in eine kanonische oder in die Standardform gebracht werden muss, unterschiedlich vorgegangen werden. Dies wird im nachfolgenden Abschnitt behandelt:

Kanonische Optimierungsform

Wenn bei Variabel x_j alle (freie) Werte ($x_j \in \mathbb{R}$) zugelassen sind, ist diese in die Form zu bringen: $x_j = x_j^+ - x_i^-$

Sind nur negative Werte $x_j \leq 0$ zugelassen, dann gilt $x_j \leq 0 \rightarrow \overline{x_j} \coloneqq -x_j, \overline{x_j} \geq 0$, Vorzeichenwechsel bei x_j

Beispiele: Gegeben sei das folgende LP:
Schreiben Sie das LP in **kanonischer Form** als **Min**imierungsproblem.

$$\begin{pmatrix} \min 3x_1 + 4x_2 - 2x_3 \\ -x_1 + x_2 \leq 4 \\ 2x_2 + x_3 \leq 8 \\ x_2 + 3x_3 \geq -2 \\ x_1 \geq 0, x_2 \in \mathbb{R}, x_3 \geq 0 \end{pmatrix}$$

Daraus folgt $\mathbf{min}\{\boldsymbol{c^T x} | \boldsymbol{Ax} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ und $x_2 \coloneqq x_2 - x_4$, müssen alle x_2 ersetzen und die Ungleichungen nach $\geq$ umstellen

$$\rightarrow \begin{pmatrix} \min 3x_1 + 4x_2 - 2x_3 - 4x_4 \\ x_1 - x_2 + x_4 \geq -4 \\ -2x_2 - x_3 + 2x_4 \geq -8 \\ x_2 + 3x_3 - x_4 \geq -2 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0 \end{pmatrix}$$

Gegeben sei das folgende LP:
Schreiben Sie das LP in **kanonischer Form** als **Max**imierungsproblem.

$$\begin{pmatrix} \min 2x_1 + x_2 + x_3 \\ 2x_1 + x_2 + 3x_3 \geq 4 \\ -x_1 + 2x_2 + x_3 \leq -6 \\ 2x_2 + 3x_3 \geq 2 \\ x_1 \in \mathbb{R}, x_2 \leq 0 \in, x_3 \in \mathbb{R} \end{pmatrix}$$

Daraus folgt $\mathbf{max}\{\boldsymbol{c^T x} | \boldsymbol{Ax} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ und $x_1 \coloneqq x_1 - x_4,\ x_3 \coloneqq x_3 - x_5$ müssen alle x_1, x_3 ersetzen und die Ungleichungen nach $\leq$ umstellen
Zudem $x_2 \leq 0 \rightarrow \overline{x_2} \coloneqq -x_2, \overline{x_2} \geq 0$

$$\rightarrow \begin{pmatrix} \max -2x_1 + x_2 - x_3 + 2x_4 + x_5 \\ -2x_1 + x_2 - 3x_3 + 2x_4 + 3x_5 \leq -4 \\ -x_1 - 2x_2 + x_3 + x_4 - x_5 \leq -6 \\ 2x_2 - 3x_3 + 3x_5 \leq -2 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0 \end{pmatrix}$$

Standardform als Maximierungsform

- Wenn bei Variabel x_j alle (freie) Werte ($\boldsymbol{x_j} \in \mathbb{R}$) zugelassen sind, ist diese in Form zu bringen: $x_j = x_j^+ - x_i^-$
- Sind nur negative Werte $x_j \leq 0$ zugelassen, dann gilt $x_j \leq 0 \rightarrow \overline{x_j} \coloneqq -x_j, \overline{x_j} \geq 0$, Vorzeichenwechsel bei x_j
- Wechsel Ungleichung auf Gleichung (Schlupfvar.): $a^T x \leq b \rightarrow a^T x + s = b,\ a^T x \geq b \rightarrow a^T x - s = b, s \geq 0$

Beispiele: Gegeben sei das folgende LP:
Schreiben Sie das LP in der **Standardform** als **Max**imierungsproblem.

$$\begin{pmatrix} \min 3x_1 + 4x_2 - 2x_3 \\ -x_1 + x_2 \leq 4 \\ 2x_2 + x_3 \leq 8 \\ 2x_2 + 3x_3 \geq 12 \\ x_1 \geq 0, x_2 \in \mathbb{R}, x_3 \geq 0 \end{pmatrix}$$

Es folgt $\mathbf{max}\{\boldsymbol{c^T x} | \boldsymbol{Ax} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ und $x_2 \coloneqq x_2 - x_4$, müssen alle x_2 ersetzen und die **Un**gleichungen mit Schlupfvariabeln ausstatten

$$\rightarrow \begin{pmatrix} \max -3x_1 - 4x_2 + 2x_3 + 4x_4 \\ -x_1 + x_2 - x_4 + x_5 = 4 \\ 2x_2 + x_3 - 2x_4 + x_6 = 8 \\ 2x_2 + 3x_3 - 2x_4 - x_7 = 12 \\ x_1, \dots, x_7 \geq 0 \end{pmatrix}$$

Konvexes Polyeder

Die Schnittmenge **endlich** vieler **Halbräume** heisst **konvexes Polyeder**. Die Definition konvexer Polyeder **schliesst nicht** aus, dass sie **leer** oder **unbeschränkt** sind. Ein **nichtleeres** und **beschränktes konvexes Polyeder** heisst **konvexes Polytop**.

Nicht lineare Optimierungsproblem in ein Lineares umwandeln

Dieses Optimierungsproblem ist kein lineares Programm, da durch die zwei Zielfunktionen in der Minimierung ein Knick entsteht und damit die zwingende Stetigkeit nicht gewährt ist.

$$\begin{pmatrix} \min\{2x_1 + x_2 + x_3, x_1 - x_3\} \rightarrow max \\ 3x_1 + x_2 + 2x_3 \leq 2 \\ x_1 + x_2 + 3x_3 \leq 5 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{pmatrix}$$

Umformulierung in LP:

$$\begin{pmatrix} \max\{a\} \\ 3x_1 + x_2 + 2x_3 \leq 2 \\ x_1 + x_2 + 3x_3 \leq 5 \\ 2x_1 + x_2 + x_3 \geq a \\ x_1 - x_3 \geq a \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{pmatrix}$$

2 Simplex-Algorithmus

Die **Auswahl** einer **Zeile** des Simplex-Tableaus **bestimmt** die **Basisvariable**, die die **Basis verlässt**, und die **Wahl** einer **Spalte** bestimmt die **Nichtbasisvariable**, die in die **Basis aufgenommen** wird → Pivotzeile/-spalte

Primaler Simplex

Folgende **Bedingungen** müssen **erfüllt** sein für die Anwendung des primalen Simplex:

- **Standardform** • **Rechte Seiten nichtnegativ** • **Einheitsmatrix unter Schlupfvariablen**
- Der Punkt $(0_n, b)$ (Nullpunkt im Raum der Variablen $x_1, \dots, x_n$) ist zulässige Basislösung und dient als Start.

Ablauf, wenn die Zielfunktion maximiert wird!

1. Ungleichungen müssen ≤ sein. Wenn dies nicht der Fall ist, muss Vorzeichenwechsel durchgeführt werden.
2. **Spaltentitel** $x_1, \dots, x_n, s_1, \dots, s_m$, ($s_i$ = Schlupfvariable pro Nebenbedingung/Zeile), **Zeilentitel** $z, s_1, \dots, s_m$ $s_1, \dots, s_m$ sind dabei die **Basisvariablen** und $x_1, \dots, x_n$ die **Nichtbasisvariablen**
3. Tableau aufstellen, Zielfunktion wird **maximiert** → Vorzeichenwechsel durchführen, da $z = c_1x_1 + \dots + c_ix_i \rightarrow z - c_1x_1 - \dots - c_ix_i = 0$ (Bei Minimierung Zielfunktion ohne Vorzeichenwechsel direkt übernehmen)
4. Bei Variablen der Zielfunktion (**kleinste) negative Element** suchen. Diese Spalte markieren → **Pivotspalte**
5. Letzte Spalte durch Pivotspalte teilen (ohne Zielfunktion!), danach kleinste Zahl (**> 0**) nehmen → **Pivotzeile** Element im Fadenkreuz → **Pivotelement**, es darf **nicht Null/negativ** sein! (**nicht** $x/0$, $x/-y$!!!). Da letzte Spalte die Werte v. Basisvariablen sind, sollten sie für zulässige Lösung Nichtnegativitätsbedingung erfüllen
6. Pivotelement muss anschliessend auf 1 gebracht werden und alle Elemente in der Spalte (oben/unten) des Pivotelements müssen gleich 0 sein. (zeilenweise Addition/Subtraktion gemäss G(r)auss-Algorithmus)
7. Wenn Werte der Zielfunktion noch **negativ** sind, muss das Vorgehen ab Punkt 4. wieder wiederholt werden.
8. **Stoppregel 1:** Sobald alle Zielfunktionswerte **≥ 0** (grösser gleich 0) sind (erste Zeile/Zeile z), kann der Zielfunktionswert (Optimum) und die Nichtbasisvariablen $x_1, \dots, x_n$ abgelesen werden. Dabei nimmt man die Elemente, die beim entsprechenden $x_i = 1$ sind und die restlichen Spaltenelemente (oben/unten) $= 0$ sind. Alle anderen x_i Werte, die diese Kriterien nicht erfüllen, sind automatisch $x_i = 0$
   Stoppregel 2: Nach Auswahl der Pivotspalte gibt es beim Teilen von b_i/x_i keine positive Zahl **> 0**. Die Zielfunktion ist **unbeschränkt** und kennt **keine Lösung** des Problems.

Beispielaufgabe

Finden Sie mit Hilfe der Simplex-Methode alle optimalen Lösungen von folgenden linearen Programmen:

$$\begin{pmatrix} \max 2x_1 + x_2 + x_3 \\ 3x_1 + x_2 + 2x_3 \leq 2 \\ x_1 + x_2 + 3x_3 \leq 5 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}$$

Lösungsweg

	x_1	x_2	x_3	s_1	s_2	b_i	b_i/x_1
z	-2	-1	-1	0	0	0	
s_1	3	1	2	1	0	2	2/3
s_2	1	1	3	0	1	5	5/1

→

	x_1	x_2	x_3	s_1	s_2	b_i	/
z	0	$-1/3$	1/3	2/3	0	4/3	
x_1	1	1/3	2/3	1/3	0	2/3	.
s_2	0	2/3	7/3	$-1/3$	1	13/3	.

	x_1	x_2	x_3	s_1	s_2	b_i	b_i/x_2
z	0	$-1/3$	1/3	2/3	0	4/3	
x_2	1	1/3	2/3	1/3	0	2/3	2/3 : 1/3
s_2	0	2/3	7/3	$-1/3$	1	13/3	13/3 : 2/3

→

	x_1	x_2	x_3	s_1	s_2	b_i
z	1	0	1	1	0	2
x_2	3	1	2	1	0	2
s_2	-2	0	1	-1	1	3

Der maximale Zielwert beträgt 2 und eine optimale Lösung ist $(x_1, x_2, x_3) = (0,2,0)$

Beispielaufgabe

Finden Sie mit Hilfe der Simplex-Methode alle optimalen Lösungen von folgenden linearen Programmen:

$$\begin{pmatrix} \max x_1 - x_2 + 2x_3 + x_4 \\ 3x_1 + x_2 - x_3 + 2x_4 \leq 5 \\ x_1 + 2x_2 + x_3 - 2x_4 \leq 4 \\ x_1, x_2, x_3, x_4 \geq 0 \end{pmatrix}$$

Lösungsweg

	x_1	x_2	x_3	x_4	s_1	s_2	b_i	b_i/x_3
z	-1	1	-2	-1	0	0	0	
s_1	3	1	-1	2	1	0	5	$5/-1$
s_2	1	2	1	-2	0	1	4	4/1

→

	x_1	x_2	x_3	x_4	s_1	s_2	b_i	b_i/x_i
z	1	5	0	-5	0	2	8	
s_1	4	3	0	0	1	1	9	.
x_3	1	2	1	-2	0	1	4	.

	x_1	x_2	x_3	x_4	s_1	s_2	b_i	b_i/x_4
z	1	5	0	-5	0	2	8	
s_1	4	3	0	0	1	1	9	9/0
x_3	1	2	1	-2	0	1	4	$4/-2$

→ Stoppregel Nr. 2 wurde soeben ausgelöst!
Die Werte der Spalte b_i/x_4 sind alle nicht positiv!
Das System ist daher **unbeschränkt**!

Bemerkungen zur Simplex-Methode

- In jedem gültigen Simplex-Tableau gibt es alle Spalten der Einheitsmatrix. Die entsprechenden Variablen heissen Basisvariablen; alle anderen heissen Nichtbasisvariablen. Nichtbasisvariablen haben den Wert Null; Basisvariablen haben den Wert der rechten Seiten.
- **Was können für Fälle bei der Simplexmethode vorkommen/auftreten:**
  i) Falls in der $z-$Zeile **alle Einträge** unterhalb von **Nichtbasisvariablen (NBV) positiv** sind, liegt eine **einzige optimale Lösung** vor.
  ii) Falls es in der $z-$Zeile unterhalb von **mindestens einer Nichtbasisvariablen (NBV)** eine **Null** gibt, **existieren unendlich** viele optimale **Lösungen**.
  iii) Falls in einem Simplex-Tableau eine potenzielle **Pivotspalte** (**negativer** Koeffizient in der z−Zeile) existiert, in der kein Pivotelement ausgewählt werden kann (weil alle **potenziellen Pivotelemente Null** oder **negativ** sind), kann man **abbrechen**; das LP ist **unbeschränkt**.

Dualer Simplex

Normalform, Rechte Seiten negativ, Einheitsmatrix unter Schlupfvariablen

Zwei-Phasen-Methode (ZPHM)

Falls Polyeder nicht im Ursprung beginnt/beinhaltet → $(0_n, b)$ keine zulässige Basislösung. Deshalb ZPHM!

1. Rechte Seite (RS) muss grösser gleich Null **≥ 0** sein. Entsprechende Zeile multiplizieren mit (-1).
2. Neben Schlupfvariablen, benötigt es nun auch noch künstliche Variablen $y_1, y_2, \ldots$ (können auch x_i heissen). Diese müssen die Bedingung $y_i \geq 0$ (Nichtnegativ) erfüllen. Dabei gilt die Regel je nach Un-/gleichung:
   $\leq \rightarrow +s$ (Schlupfvar.), $= \rightarrow +y$ (künstliche Var.), $\geq \rightarrow -s+y$ (- Schlupfvar. + künstliche Var.)
3. **Phase 1:** Tableau wie gewohnt aufstellen. Wollen nun wie immer Zielfunktion maximieren. Brauchen jedoch noch künstliche Zielfunktion für die y Variablen, welche wir zuerst minimieren → $W = \min y_1 + y_2 + \cdots + y_i$. $W = -\sum y_i, y_i \geq 0$. Führen nun Zeile W im Tableau ein. Dort (Zeile W) sollen die $y_1 = 1, y_2 = 1, \ldots$ Werte mit Hilfe des G(r)auss und der Zeilen der Nebenbedingungen zuerst auf null gebracht werden. Danach müssen die negativen Variablen optimiert werden und damit der Zielfunktionswert auf 0 gebracht werden.
   - **Möglichkeit 1:** Künstliche Zielfunktion hat Wert Null (bei b_i) → Originalproblem = **zulässig**.
     i) Spalten y_i und Zielfunktionszeile W fallen weg, wenn alle $y_i = NBV$ (keine Nebenbedingung mit y_i)
     ii) Wenn ein y_i in Basis, muss dies ohne Beeinflussung des Zielfunktionswerts von w behoben werden.
   - **Möglichkeit 2:** Künstliche Zielfunktion hat Wert kleiner Null. Originalproblem ist **unzulässig**. → Stopp
4. **Phase 2:** Wenn Möglichkeiten i) und ii) abgeschlossen, dann gewohnter Ablauf des Simplex-Verfahrens.
5. Bei Variablen der Zielfunktion (**kleinste) negative Element** suchen. Diese Spalte markieren → **Pivotspalte**
6. Letzte Spalte durch Pivotspalte teilen (ohne Zielfunktion!), danach kleinste Zahl (**> 0**) nehmen → **Pivotzeile** Element im Fadenkreuz → **Pivotelement**. Pivotelement darf **nicht Null/negativ** sein! (**nicht** $x/0$, $x/-y$!!!)
7. Pivotelement muss anschliessend auf 1 gebracht werden und alle Elemente in der Spalte (oben/unten) des Pivotelements müssen gleich 0 sein. (zeilenweise Addition/Subtraktion gemäss G(r)auss-Algorithmus)
8. Wenn Werte der Zielfunktion noch **negativ** sind, muss das Vorgehen ab Punkt 4. wieder wiederholt werden.
9. **Stoppregel 1:** Sobald alle Zielfunktionswerte **≥ 0** (grösser gleich 0) sind (erste Zeile/Zeile z), kann der Zielfunktionswert (Optimum) und die Nichtbasisvariablen $x_1, \ldots, x_n$ abgelesen werden. Dabei nimmt man die Elemente, die beim entsprechenden $x_i = 1$ sind und die restlichen Spaltenelemente (oben/unten) $= 0$ sind. Alle anderen x_i Werte, die diese Kriterien nicht erfüllen, sind automatisch $x_i = 0$
   Stoppregel 2: Nach Auswahl der Pivotspalte gibt es beim Teilen von b_i/x_i keine positive Zahl **> 0**. Die Zielfunktion ist **unbeschränkt** und kennt **keine Lösung** des Problems.

Aufgabe (ZPHM)

Finden Sie mit Hilfe der Zwei-Phasen-Methode alle optimalen Lösungen vom linearen Programm:

$$\begin{pmatrix} \max 2x_1 + x_2 \\ -x_1 - 2x_2 \geq -3 \\ 3x_1 + x_2 = 6 \\ 4x_1 + 3x_2 \geq 3 \\ x_1, x_2 \geq 0 \end{pmatrix} \rightarrow 1.) \begin{pmatrix} \max 2x_1 + x_2 \\ x_1 + 2x_2 \leq 3 \\ 3x_1 + x_2 = 6 \\ 4x_1 + 3x_2 \geq 3 \\ x_1, x_2 \geq 0 \end{pmatrix} \rightarrow 2./3.) \begin{pmatrix} \max 2x_1 + x_2 \\ x_1 + 2x_2 + s_1 = 3 \\ 3x_1 + x_2 \quad + y_1 = 6 \\ 4x_1 + 3x_2 - s_2 + y_2 = 3 \\ x_1, x_2, s_1, s_2, y_1, y_2 \geq 0 \end{pmatrix}$$

	x_1	x_2	s_1	s_2	y_1	y_2	b_i	b_i/x_i
z	-2	-1	0	0	0	0	0	
w	0	0	0	0	1	1	0	
s_1	1	2	1	0	0	0	3	/
y_1	3	1	0	0	1	0	6	/
y_2	4	3	0	-1	0	1	3	/

→ 4.)

	x_1	x_2	s_1	s_2	y_1	y_2	b_i	b_i/x_i	
z	-2	-1	0	0	0	0	0		
w	-7	-4	0	1	0	0	-9		$w - y_1$
s_1	1	2	1	0	0	0	3	/	$w - y_2$
y_1	3	1	0	0	1	0	6	/	
y_2	4	3	0	-1	0	1	3	/	

	x_1	x_2	s_1	s_2	y_1	y_2	b_i	b_i/x_1
z	-2	-1	0	0	0	0	0	
w	-7	-4	0	1	0	0	-9	
s_1	1	2	1	0	0	0	3	3/1
y_1	3	1	0	0	1	0	6	6/3
y_2	4	3	0	-1	0	1	3	3/4

→

	x_1	x_2	s_1	s_2	y_1	y_2	b_i	b_i/x_i	
z	0	1/2	0	$-1/2$	0	1/2	3/2		$z + 2III$
w	0	5/4	0	$-3/4$	0	7/4	$-15/4$		$w + 7III$
s_1	0	5/4	1	1/4	0	$-1/4$	9/4	/	$I - 1III$
y_1	0	$-5/4$	0	3/4	1	$-3/4$	15/4	/	$II - 3III$
x_1	1	3/4	0	$-1/4$	0	1/4	3/4	/	$III/4$

	x_1	x_2	s_1	s_2	y_1	y_2	b_i	b_i/x_i
z	0	1/2	0	$-1/2$	0	1/2	3/2	
w	0	5/4	0	$-3/4$	0	7/4	$-15/4$	
s_1	0	5/4	1	1/4	0	$-1/4$	9/4	9/4: 1/4
s_2	0	$-5/4$	0	3/4	1	$-3/4$	15/4	15/4: 3/4
x_1	1	3/4	0	$-1/4$	0	1/4	3/4	–

→

	x_1	x_2	s_1	s_2	y_1	y_2	b_i	b_i/x_i	
z	0	$-1/3$	0	0	2/3	0	4		$z + 1/2II$
w	0	0	0	0	1	1	0		$w + 3/4II$
s_1	0	5/3	1	0	$-1/3$	0	1	/	$I - 1/4II$
s_2	0	$-5/3$	0	1	4/3	-1	5	/	$II: 3/4$
x_1	1	1/3	0	0	1/3	0	2	/	$III + 1/4II$

Nun weiter mit Phase 2

	x_1	x_2	s_1	s_2	b_i	b_i/x_i
z	0	$-1/3$	0	0	4	
x_2	0	5/3	1	0	1	1: 5/3
s_2	0	$-5/3$	0	1	5	–
x_1	1	1/3	0	0	2	2: 1/3

→

	x_1	x_2	s_1	s_2	b_i	b_i/x_i	
z	0	0	1/5	0	21/5		$z + 1/3I$
x_2	0	1	3/5	0	3/5	/	$I: 5/3$
s_2	0	0	1	1	6	–	$II + 5/3I$
x_1	1	0	$-1/5$	0	9/5	/	$III - 1/3I$

Stoppregel Nr. 1 wurde ausgelöst!
Die Werte der Zeile z sind alle nicht negativ!
$x_1 = \frac{9}{5}, x_2 = \frac{3}{5}, o = \frac{21}{5}$

3 Dualität linearer Programme

Beispiel: Produktionsproblem und Schattenpreise

Firma XY stellt aus 2 Arten Rohmaterial 3 verschiedene Produkte her. Wir haben folgende Informationen: benötigte Menge Rohmaterial pro Produkt, Profit pro Produkt und zur Verfügung stehende Menge an Rohmaterial.

- Wie sieht der bezüglich des Profits optimale Produktionsplan aus? → Lösung mittels LP
- Angenommen, eine andere Firma möchte das Rohmaterial abkaufen. Welche Preise sollte sie offerieren?
- Wie hoch sollten Preise sein, damit Firma XY lieber das Rohmaterial verkauft, als dass sie selbst produziert?
- Wie viel ist der Firma XY das Rohmaterial wert? → **Schattenpreise**, führt uns zu **Dualproblem**.

Vom Primalproblem zum Dualproblem, wenn kanonische Form vorliegt

Primal-duales Paar

$(\boldsymbol{P})$	$(\boldsymbol{D})$
$\boldsymbol{c^T x \rightarrow max}$	$\boldsymbol{b^T y \rightarrow min}$
$\boldsymbol{Ax \leq b}$	$\boldsymbol{A^T y \geq c}$
$\boldsymbol{x \geq 0}$	$\boldsymbol{y \geq 0}$

- max → min
- Zielfunktionsvektor c → Vektor der rechten Seite b
- Vektor der rechten Seite b → Zielfunktionsvektor c
- Koeffizientsmatrix A → transponierte Koeffizientsmatrix A^T
- Nebenbedingung $\leq$ → Variable ≥ 0
- Variable ≥ 0 → Nebenbedingung $\geq$

Betrachten primal-duales Paar. Welches der beiden LP's ist das primale und welches das duale Problem?
Gegeben sei das primal-duale Paar $(P) - (D)$ in kanonischer Form. Das duale Problem zu (D) ist (P).
Das Problem D heisst **Dualproblem** von P, und die Variablen y heissten **Dualvariablen**. In Anwesenheit eines Dualproblems nennt man das Originalproblem **Primalproblem** und seine Variablen **Primalvariablen**. Da das Dualproblem eines Dualproblem wieder das Primalproblem ist, spielt es keine Rolle, welches der beiden Probleme man als Primal- oder Dualproblem auffasst. Man sagt deswegen einfach, P und D seinen zueinander **dual**.
→ Deshalb ist obige **Tabelle** nicht nur von **links nach rechts**, sondern **auch** von **rechts nach links** anwendbar.

Beispielaufgaben: Transformieren Sie das LP in kanonische Maximierungsform. Formuliere entsprechendes duale Problem.

$$\begin{pmatrix} \min 2x_1 + x_2 - 3x_3 \\ x_1 + 2x_2 + x_3 = 8 \\ x_1 + x_2 + 2x_3 \leq 12 \\ 2x_1 + 2x_2 \geq 5 \\ x_2 \geq 2 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}$$

Kanonische Max.-form:

$$\begin{pmatrix} \max -2x_1 - x_2 + 3x_3 \\ x_1 + 2x_2 + x_3 \leq 8 \\ -x_1 - 2x_2 - x_3 \leq -8 \\ x_1 + x_2 + 2x_3 \leq 12 \\ -2x_1 - 2x_2 \leq -5 \\ -x_2 \leq -2 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}, \quad b^T = (8, -8, 12, -5, -2), \quad A = \begin{pmatrix} 1 & 2 & 1 \\ -1 & -2 & -1 \\ 1 & 1 & 2 \\ -2 & -2 & 0 \\ 0 & -1 & 0 \end{pmatrix}, \quad c^T = (-2, -1, 3), \quad A^T = \begin{pmatrix} 1 & -1 & 1 & -2 & 0 \\ 2 & -2 & 1 & -2 & -1 \\ 1 & -1 & 2 & 0 & 0 \end{pmatrix}$$

$$(D): \begin{pmatrix} \min 8y_1 - 8y_2 + 12y_3 - 5y_4 - 2y_5 \\ y_1 - y_2 + y_3 - 2y_4 \geq -2 \\ 2y_1 - 2y_2 + y_3 - 2y_4 - y_5 \geq -1 \\ y_1 - y_2 + 2y_3 \geq 3 \\ y_1, y_2, y_3, y_4, y_5 \geq 0 \end{pmatrix} = \begin{pmatrix} \min 8y_1 + 12y_3 - 5y_4 - 2y_5 \\ y_1 + y_3 - 2y_4 \geq -2 \\ 2y_1 + y_3 - 2y_4 - y_5 \geq -1 \\ y_1 + 2y_3 \geq 3 \\ y_1 \in \mathbb{R}; y_3, y_4, y_5 \geq 0 \end{pmatrix}$$

Beispielaufgaben: Transformieren Sie das LP in kanonische Minimierungsform. Formuliere entsprechendes duale Problem.

$$\begin{pmatrix} \min 2x_1 + x_2 - 3x_3 \\ x_1 + 2x_2 + x_3 = 8 \\ x_1 + x_2 + 2x_3 \leq 12 \\ 2x_1 + 2x_2 \geq 5 \\ x_2 \geq 2 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}$$

Kanonische Min.-form:

$$\begin{pmatrix} \min 2x_1 + x_2 - 3x_3 \\ x_1 + 2x_2 + x_3 \geq 8 \\ -x_1 - 2x_2 - x_3 \geq -8 \\ -x_1 - x_2 - 2x_3 \geq -12 \\ 2x_1 + 2x_2 \geq 5 \\ x_2 \geq 2 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}, \quad b^T = (8, -8, -12, 5, 2), \quad A = \begin{pmatrix} 1 & 2 & 1 \\ -1 & -2 & -1 \\ -1 & -1 & -2 \\ 2 & 2 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad c^T = (2, 1, -3), \quad A^T = \begin{pmatrix} 1 & -1 & -1 & 2 & 0 \\ 2 & -2 & -1 & 2 & 1 \\ 1 & -1 & -2 & 0 & 0 \end{pmatrix}$$

$$(D): \begin{pmatrix} \max 8y_1 - 8y_2 - 12y_3 + 5y_4 + 2y_5 \\ y_1 - y_2 - y_3 + 2y_4 \leq 2 \\ 2y_1 - 2y_2 - y_3 + 2y_4 + y_5 \leq 1 \\ y_1 - y_2 - 2y_3 \leq -3 \\ y_1, y_2, y_3, y_4, y_5 \geq 0 \end{pmatrix} = \begin{pmatrix} \max 8y_1 - 12y_3 + 5y_4 + 2y_5 \\ y_1 - y_3 + 2y_4 \leq 2 \\ 2y_1 - y_3 + 2y_4 + y_5 \leq 1 \\ y_1 - 2y_3 \leq -3 \\ y_1 \in \mathbb{R}; y_3, y_4, y_5 \geq 0 \end{pmatrix}$$

Vom Primalproblem zum Dualproblem, wenn Standardform vorliegt

Primal-duales Paar

- $(\boldsymbol{P1})$
  - $\boldsymbol{c^T x \to max}$
  - $\boldsymbol{Ax = b}$
  - $\boldsymbol{x \geq 0}$
- $(\boldsymbol{D1})$
  - $\boldsymbol{b^T y \to min}$
  - $\boldsymbol{A^T y \geq c}$

Beispielaufgaben: Transformieren Sie das LP in die Standardform.
Formuliere entsprechendes duale Problem.

$$\begin{pmatrix} \min 2x_1 + x_2 - 3x_3 \\ x_1 + 2x_2 + x_3 = 8 \\ x_1 + x_2 + 2x_3 \leq 12 \\ 2x_1 + 2x_2 \geq 5 \\ x_2 \geq 2 \\ x_1, x_2, x_3 \geq 0 \end{pmatrix}$$

Kanonische Max.-form:

$$\begin{pmatrix} \max -2x_1 - x_2 + 3x_3 \\ x_1 + 2x_2 + x_3 = 8 \\ x_1 + x_2 + 2x_3 + x_4 = 12 \\ 2x_1 + 2x_2 - x_5 = 5 \\ x_2 - x_6 = 2 \\ x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{pmatrix}, \quad b^T = (8,\ 12,\ 5,\ 2), \quad c^T = (-2, -1,\ 3,\ 0,\ 0,\ 0), \quad A = \begin{pmatrix} 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 0 \\ 2 & 2 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix},$$

$$A^T = \begin{pmatrix} 1 & 1 & 2 & 0 \\ 2 & 1 & 2 & 1 \\ 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad (D): \begin{pmatrix} 8y_1 & +12y_2 & +5y_3 & +2y_4 & \to & min \\ y_1 & +y_2 & 2y_3 & & \geq & -2 \\ 2y_1 & +y_2 & +2y_3 & +y_4 & \geq & -1 \\ y_1 & +2y_2 & & & \geq & 3 \\ & y_2 & & & \geq & 0 \\ & & -y_3 & & \geq & 0 \\ & & & -y_4 & \geq & 0 \end{pmatrix}$$

alle Variable werden freie Variablen!

Bemerkung: Alle drei dualen LP's von oben die wir gelöst haben beschreiben dasselbe Optimierungsproblem, obwohl sie formal unterschiedlich aussehen.

Allgemein: Vom Primalproblem zum Dualproblem

Damit das Originalproblem nicht zuerst in Standardform überführt werden muss (wenn bspw. Gleichungen oder freie Variable/$x \in \mathbb{R}$ vorliegen), gibt es folgende direkte Transformationsregeln:

• $\boldsymbol{max}$	↔	$\boldsymbol{min}$
• Zielfunktionsvektor c	↔	Vektor der rechten Seite b
• Vektor der rechten Seite b	↔	Zielfunktionsvektor c
• Koeffizientenmatrix A	↔	transponierte Koeffizientenmatrix A^T
• Nebenbedingung $i: \leq$	↔	Variable $y_i \geq 0$
• Nebenbedingung $i: =$	↔	Variable $y_i \in \mathbb{R}$ (freie Variable)
• Nebenbedingung $i: \geq$	↔	Variable $y_i \leq 0$
• Variable $x_j \geq 0$	↔	Nebenbedingung $j: \geq$
• Variable $x_j \in \mathbb{R}$ (freie Variable)	↔	Nebenbedingung $j: =$
• Variable $x_j \leq 0$	↔	Nebenbedingung $j: \leq$

Achtung: Bei Nebenbedingungen zu Variable und umgekehrt gilt nicht das gleiche (6 letzten Zeilen!)

Beispielaufgabe: Dualisieren Sie folgendes LP:

$$\begin{pmatrix} \max x_1 + 2x_2 + 3x_3 \\ 3x_1 + x_2 \leq 5 \\ x_1 - x_2 = 7 \\ x_1 \geq 0, x_2 \in \mathbb{R}, x_3 \geq 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 3 & 1 & 0 \\ 1 & -1 & 0 \end{pmatrix}, A^T = \begin{pmatrix} 3 & 1 \\ 1 & -1 \\ 0 & 0 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, c = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, b = \begin{pmatrix} 5 \\ 7 \end{pmatrix}$$

Duales LP lautet damit:
→ Regeln gemäss Tabelle von oben anwenden!

$$\begin{pmatrix} \min 5y_1 + 7y_2 \\ 3y_1 + y_2 \geq 1\ (\geq da\ Variable\ x_1 \geq 0) \\ y_1 - y_2 = 2\ (= da\ Variable\ x_2 \in \mathbb{R}) \\ 0 \leq 3\ (\geq da\ Variable\ x_3 \geq 0) \\ y_1 \geq 0\ (\geq da\ NB_1 \to \leq), y_2 \in \mathbb{R}(\mathbb{R}\ da\ NB_2 \to =) \end{pmatrix} = \begin{pmatrix} \min 5y_1 + 7y_2 \\ 3y_1 + y_2 \geq 1 \\ y_1 - y_2 = 2 \\ y_1 \geq 0, y_2 \in \mathbb{R} \end{pmatrix}$$

Beispielaufgabe: Dualisieren Sie folgendes LP:

$$\begin{pmatrix} \min x_1 - x_2 + 3x_3 \\ x_1 - 4x_2 + 3x_3 = 5 \\ 2x_1 + 3x_2 + 5x_4 \geq 12 \\ x_1 + x_2 + 2x_3 \leq 4 \\ x_1 \geq 0, x_2 \in \mathbb{R}, x_3 \geq 0, x_4 \in \mathbb{R} \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & -4 & 3 & 0 \\ 2 & 3 & 0 & 5 \\ 1 & 1 & 2 & 0 \end{pmatrix}, A^T = \begin{pmatrix} 1 & 2 & 1 \\ -4 & 3 & 1 \\ 3 & 0 & 2 \\ 0 & 5 & 0 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, c = \begin{pmatrix} 1 \\ -1 \\ 3 \\ 0 \end{pmatrix}, b = \begin{pmatrix} 5 \\ 12 \\ 4 \end{pmatrix}$$

Achtung: Tabelle muss nun wegen min von rechts nach links gelesen werden!

Duales LP lautet damit:

$$\begin{pmatrix} \max 5y_1 + 12y_2 + 4y_3 \\ y_1 + 2y_2 + y_3 \leq 1\ (\leq da\ Variable\ x_1 \geq 1) \\ -4y_1 + 3y_2 + y_3 = -1\ (= da\ Variable\ x_2 \in \mathbb{R}) \\ 3y_1 + 2y_3 \leq 3\ (\leq da\ Variable\ x_3 \geq 1) \\ 5y_2 = 0\ (= da\ Variable\ x_4 \in \mathbb{R}) \\ y_1 \in \mathbb{R}(\mathbb{R}\ da\ NB_1 \to =), y_2 \geq 0(\geq da\ NB_2 \to \geq), y_3 \leq 0(\leq da\ NB_3 \to \leq) \end{pmatrix}$$

Schwache Dualität

- Bis auf einfache Ausnahmefälle stimmen die optimalen Werte von P und D überein. Über die optimalen Punkte kann man das nicht sagen, da sie in Räumen verschiedener Dimensionen ($\mathbb{R}^n$ bzw. $\mathbb{R}^m$) liegen.
- Gegeben sei das primal-duale Paar $(P) - (D)$ in kanonischer Form.
- Sei x^* ein primal zulässiger Punkt in (P) und sei y^* ein dual zulässiger Punkt in (D), dann gilt: $\boldsymbol{c^T x^* \leq b^T y^*}$
- Der schwache Dualitätssatz besagt, dass der Zielfunktionswert $c^T x$ jedes **primal** zulässigen **Punktes** x eine **Unterschranke** für alle **Zielfunktionswerte** $b^T y$ **dual** zulässiger **Punkte** y liefert.
- Der Optimalwert von P sei z^* und der Optimalwert von D sei v^*, dann gilt: $\boldsymbol{z^* = c^T x^* \leq b^T y^* = v^*}$
- Aus der schwachen Dualität folgt, dass falls P unbeschränkt ist, ist D unzulässig. Ist D unbeschränkt ist, ist P unzulässig. D.h. ist das eine Problem unbeschränkt, ist das andere automatisch unzulässig.

Starke Dualität

Wenn das primale Problem eine optimale Lösung besitzt, so hat auch das duale Problem eine optimale Lösung; die optimalen Zielfunktionswerte stimmen überein.

Der Optimalwert von P sei z^* und der Optimalwert von D sei v^*, dann gilt: $\boldsymbol{z^* = c^T x^* = b^T y^* = v^*}$

- $\boldsymbol{(P2)}$
  - $\boldsymbol{c^T x \to max}$
  - $\boldsymbol{Ax + s = b}$
  - $\boldsymbol{x, s \geq 0}$
- $\boldsymbol{(D2)}$
  - $\boldsymbol{b^T y \to min}$
  - $\boldsymbol{A^T y - t = c}$
  - $\boldsymbol{y, t \geq 0}$

Satz über komplementären Schlupf

Sei (x^*, s^*) eine zulässige Lösung von $(P2)$ und (y^*, t^*) eine zulässige Lösung von $(D2)$. Dann sind diese zulässigen Lösungen optimal genau dann, wenn

- $x_j^* t_j^* = 0$ für alle j **und**
- $y_i^* s_i^* = 0$ für alle i

Beispielaufgabe: Entscheide mit Hilfe der Dualitätstheorie, ob $x^* = \left(x_1 = \frac{400}{3}, x_2 = 0, x_3 = 0, x_4 = \frac{20}{3}\right)$ eine optimale Lösung ist von: →

$$\begin{pmatrix} \max 6x_1 + 10x_2 + 9x_3 + 10x_4 \\ 4x_1 + 9x_2 + 7x_3 + 10x_4 \leq 600 \\ x_1 + x_2 + 3x_3 + 40x_4 \leq 400 \\ 3x_1 + 4x_2 + 2x_3 + x_4 \leq 500 \\ x_1, x_2, x_3, x_4 \geq 0 \end{pmatrix}$$

Überführen Problem in **Standardform** und setzen dann Kandidat für optimale Lösung x^* in Standardform ein

$$P = \begin{pmatrix} \max 6x_1 + 10x_2 + 9x_3 + 10x_4 \\ 4x_1 + 9x_2 + 7x_3 + 10x_4 + s_1 = 600 \\ x_1 + x_2 + 3x_3 + 40x_4 + s_2 = 400 \\ 3x_1 + 4x_2 + 2x_3 + x_4 + s_3 = 500 \\ x_1, x_2, x_3, x_4, s_1, s_2, s_3 \geq 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 \cdot \frac{400}{3} + 9 \cdot 0 + 7 \cdot 0 + 10 \cdot \frac{20}{3} + s_1 = 600 \\ \frac{400}{3} + 0 + 3 \cdot 0 + 40 \cdot \frac{20}{3} + s_2 = 400 \\ 3 \cdot \frac{400}{3} + 4 \cdot 0 + 2 \cdot 0 + \frac{20}{3} + s_3 = 500 \end{pmatrix} = \begin{pmatrix} \frac{1600}{3} + \frac{200}{3} + s_1 = 600 \\ \frac{400}{3} + \frac{800}{3} + s_2 = 400 \\ \frac{1200}{3} + \frac{20}{3} + s_3 = 500 \end{pmatrix} = \begin{pmatrix} 600 + s_1 = 600 \\ 400 + s_2 = 400 \\ \frac{1220}{3} + s_3 = 500 \end{pmatrix} = \begin{pmatrix} s_1 = 0 \\ s_2 = 0 \\ s_3 = \frac{280}{3} \end{pmatrix}$$

Da alle s_i die Bedingung $s_i \geq 0$ erfüllen, haben wir die Zulässigkeit nachgewiesen.

Nun brauchen wir das duale Problem D zum obenstehenden primalen Problem P.

$$D = \begin{pmatrix} \min 600y_1 + 400y_2 + 500y_3 \\ 4y_1 + y_2 + 3y_3 - t_1 = 6 \\ 9y_1 + y_2 + 4y_3 - t_2 = 10 \\ 7y_1 + 3y_2 + 2y_3 - t_3 = 9 \\ 10y_1 + 40y_2 + y_3 - t_4 = 10 \\ y_1, y_2, y_3, t_1, t_2, t_3, t_4 \geq 0 \end{pmatrix}$$

Die primale zulässige Lösung ist optimal in P genau dann, wenn es eine dual zulässige Lösung gibt, die zusammen mit der primalen Lösung die Komplementaritärsbedingungen erfüllt. Deshalb formulieren wir jetzt die Komplementaritätsbedingungen (Satz über komplementären Schlupf):

$$x_1 t_1 = 0, \quad x_2 t_2 = 0, \quad x_3 t_3 = 0, \quad x_4 t_4 = 0$$
$$y_1 s_1 = 0, \quad y_2 s_2 = 0, \quad y_3 s_3 = 0$$

Müssen nun die x^* → $x_1 = \frac{400}{3}, x_2 = 0, x_3 = 0, x_4 = \frac{20}{3}$ und $s_1 = 0, s_2 = 0, s_3 = \frac{280}{3}$ überprüfen, auf Elemente, die nicht Null sind, den dann muss der zweite Teil der Multiplikation zwingend eine Null sein, um die Bedingung von oben erfüllen zu können. → $x_1 > 0 \to t_1 = 0$, $x_4 > 0 \to t_4 = 0, s_3 > 0 \to y_3 = 0$

Nun setzen wir die neu gewonnen Informationen in D ein und erhalten:

$$D = \begin{pmatrix} \min 600y_1 + 400y_2 + 500y_3 \\ 4y_1 + y_2 + 3y_3 - t_1 = 6 \\ 9y_1 + y_2 + 4y_3 - t_2 = 10 \\ 7y_1 + 3y_2 + 2y_3 - t_3 = 9 \\ 10y_1 + 40y_2 + y_3 - t_4 = 10 \\ y_1, y_2, y_3, t_1, t_2, t_3, t_4 \geq 0 \end{pmatrix} = \begin{pmatrix} \min 600y_1 + 400y_2 \\ 4y_1 + y_2 = 6 \\ 9y_1 + y_2 - t_2 = 10 \\ 7y_1 + 3y_2 - t_3 = 9 \\ 10y_1 + 40y_2 = 10 \\ y_1, y_2, t_2, t_3 \geq 0 \end{pmatrix}$$

Müssen nur noch eine zulässige Lösung für dieses vorliegende Programm finden, dann ist $x^*\left(x_1 = \frac{400}{3}, x_2 = 0, x_3 = 0, x_4 = \frac{20}{3}\right)$ Primal optimal. Vierte Gleichung in erste substituiert ergibt $y_1 = \frac{23}{15}$, $y_2 = -\frac{2}{15}$

$$10y_1 + 40y_2 = 10 \Leftrightarrow y_1 = 1 - 4y_2 \quad \to \quad 4y_1 + y_2 = 6 \to 4(1 - 4y_2) + y_2 = 6 \Leftrightarrow 4 - 16y_2 + y_2 = 6 \Leftrightarrow y_2 = -\frac{2}{15}$$

Da alle $y_i$nicht negativ sein müssen, sehen wir aufgrund von $y_2 = -\frac{2}{15}$, dass das System keine zulässige Lösung hat, woraus folgt, dass dass $x_1 = \frac{400}{3}, x_2 = 0, x_3 = 0, x_4 = \frac{20}{3}$ keine optimale Lösung für P ist.

Kombinationen des Primal-Dualen Paares
Die Tabelle zeigt, welche Kombinationen theoretisch möglich sind (ja) und welche sicher nicht vorkommen können (nein).

P \ D	Optimal	Unbeschränkt	Unzulässig
Optimal	Ja	Nein	Nein
Unbeschränkt	Nein	Nein	Ja
Unzulässig	Nein	Ja	Ja

4 Ganzzahlige lineare Programme (Integer Linear Program) – ILP

Arten von linearen Programmen:

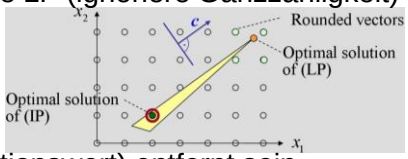
- Lineares Programm: $\max\{c^T x | x \in P\}$ mit $P = \{x \in \mathbb{R}^n | Ax \leq b\}$
- Ganzzahliges lineares Programm: $\max\{c^T x | x \in P \cap \mathbb{Z}^n\}$ mit $P = \{x \in \mathbb{R}^n | Ax \leq b\}$
- Binäres lineares Programm $\max\{c^T x | x \in P \cap \{0,1\}^n\}$ mit $P = \{x \in \mathbb{R}^n | Ax \leq b\}$
- Gemischt-ganzzahliges lineares Programm mindestens eine Variable darf nur ganzzahlige Werte haben

Ganzzahlige lineare Programme:

- **Ganzzahlige Variablen** modellieren typischerweise **Anzahlen** von **unteilbaren Objekten**
- **Ja/nein** Entscheidungen durch **binäre Variablen** modelliert
- **Komplexität: Ganzzahlige** lineare Programme gehören zur am **schwierigsten lösbaren Problemklasse**
  - **NP**-vollständig: nur **exponentielle Algorithmen** bekannt
  - im Gegensatz zu «**klassischen**» LPs, die **polynomiell lösbar** sind
- Heutzutage stehen mächtige ILP Solver zur Verfügung
  - kommerziell: Gurobi, CPLEX, ...
  - nicht-kommerziell: GLPK, lpsolve, ...
- Kombiniert mit algebraischen Modellierungssprachen für die Problemformulierung GAMS, AMPL, LPL, ...

Auf-/abrunden funktioniert nicht

- Naive Idee zur Lösung von ILPs (nicht anwendbar!): Löse das entsprechende LP (ignoriere Ganzzahligkeit)
- Falls Lösung gebrochene Komponenten enthält, runde auf/ab...
- Warum funktioniert dieser Ansatz nicht?
  - Runden optimalen Lösung von LPs muss keine zulässige Lösung liefern
  - Optimale Lösung des LPs (und dessen Zielfunktionswert) können «beliebig weit» von der optimalen Lösung des ILPs (und dessen Zielfunktionswert) entfernt sein
  - Runden bei binären Problemen kann alle 0-1-Kombinationen produzieren; somit ist die Information bzgl. der optimalen Lösung vom LP verloren



Relaxierungen von Optimierungsproblemen

- **Relaxierung** des Optimierungsproblems $\max\{f(x)|x \in S\}$ durch Vergrösserung des Lösungsraums: ersetze S durch S', sodass $S \subseteq S'$ gilt.
- **Proposition:** Sei $S \subseteq S'$, dann gilt: $\max\{f(x)|x \in S\} \leq \max\{f(x)|x \in S'\}$
- **Definition Relaxierung eines Optimierungsproblems:** Es sei das Optimierungsproble $\Pi: \max\{f(x)|x \in S\}$ gegeben. Das Optimierungsproblem $\Pi': \max\{f(x)|x \in S'\}$ ist eine Relaxierung von Π falls gilt $S \subseteq S'$.

LP-Relaxierungen

- ILP ist $\Pi: \max\{c^T x | Ax \leq b, x \in \mathbb{Z}^n\}$, die LP-Relaxierung von Π ist dann das LP $\Pi^{LP}: \max\{c^T x | Ax \leq b\}$
- Binäre LP ist $\Pi: \max\{c^T x | Ax \leq b, x \in \{0,1\}^n\}$, die LP-Relaxierung von Π ist dann das LP $\Pi^{LP}: \max\{c^T x | Ax \leq b, 0 \leq x \leq 1\}$
- Gemäss Proposition liefert der optimale Zielfunktionswert der LP-Relaxierung eine **obere Schranke** für den optimalen Zielfunktionswert des ILPs (im **Falle** einer **Maximierung**).

Beispiel: Knapsack-Problem

Eine Firma möchte b Franken in eine Auswahl von n Projekten investieren. Jedes Projekt i $(i = 1, \ldots, n)$ benötigt eine Investition in der Höhe a_i Franken und verspricht einen Gewinn von c_i Franken. Wie soll investiert werden, damit der Gewinn maximal wird?

Parameter:
n Anzahl Projekte
a_i benötigte Investition für Projekt i
c_i Gewinn für Projekt i
b Total Geld zur Verfügung

Variablen:
$x_i,\ i = 1, \ldots, n$ und $x \in \{0,1\}$

Zielfunktion:
$\max \sum_{i=1}^{n} c_i \cdot x_i$

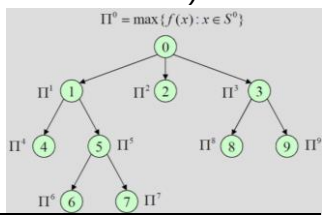
Nebenbedingung:
$\sum_{i=1}^{n} a_i \cdot x_i \leq b$

Alternative
$Q_i = \frac{c_i}{a_i}$ bilden und Q_i absteigend sortieren und in Projekte investieren, solange das Geld reicht. → Greedy Ansatz

Branch-and-Bound (B&B) Methode

B&B ist eine allgemeine Methode zum Lösen von Optimierungsproblemen (nicht auf ILPs beschränkt).

Grober Ablauf von B&B (bei **Maximierung**):



- Lösungsraum in kleinere Mengen iterativ aufteilen (**Branch**)
- In jedem Subproblem:
  - Berechne obere Schranke (**Bound**), z.B. mit Relaxierung
  - Bestimme zulässige Lösung → untere Schranke (**Bound**)
- Nutze diese Schrankeninfo zum «Wegschneiden» von gewissen Subproblemen

Detaillierter Ablauf: In jedem Subproblem (Knoten): Suche eine optimale Lösung der LP-Relaxierung

- LP-Relaxierung unzulässig ⇒ Subproblem unzulässig → Knoten fertig
- LP-Relaxierung besitzt eine **ganzzahlige optimale Lösung** ⇒ haben **optimale Lösung** im **Subproblem gefunden** → merke **Lösungskandidaten**, falls besser als letzter gefundener Kandidat → Knoten fertig
- LP-Relaxierung besitzt eine **nicht-ganzzahlige optimale Lösung** → obere Schranke im Knoten ≤ Wert des letzten gemerkten Kandidaten → Knoten fertig
- Suche eine zulässige Lösung des **ganzzahligen Subproblems** (falls möglich) → untere Schranke im Knoten = obere Schranke im Knoten ⇒ Lösungskandidat gefunden → merke Kandidaten, falls besser als letzter gefundener Kandidat → Knoten fertig
- Haben gebrochene optimale Lösung der LP-Relaxierung im Knoten. Suche eine Variable x_j, die einen nicht ganzzahligen Wert $k_j + f_j$ besitzt, wobei $k_j \in \mathbb{Z}$ und $0 < f_j < 1$. Produziere zwei neue Knoten:
  - im ersten wird die Nebenbedingung $x_j \leq k_j$ dazu genommen;
  - im zweiten wird die Nebenbedingung $x_j \geq k_j + 1$ dazu genommen.
- Gehe zum nächsten Knoten und wiederhole Verfahren, bis alle Knoten fertig sind.
- Der zuletzt gemerkte Lösungskandidat liefert eine optimale Lösung des ganzzahligen LP's.

Beispiel anhand Knapsack Problem

- Quotienten von Wert geteilt durch Volumen/Gewicht (Limitierung) teilen. Einzelne Elemente (Quotienten) absteigend sortieren, beginnend mit dem höchsten Wert.
- Bei Relaxierung (Vergrösserung Lösungsraum) ist es explizit erlaubt, nicht ganzzahlige Zahlen zu nehmen.

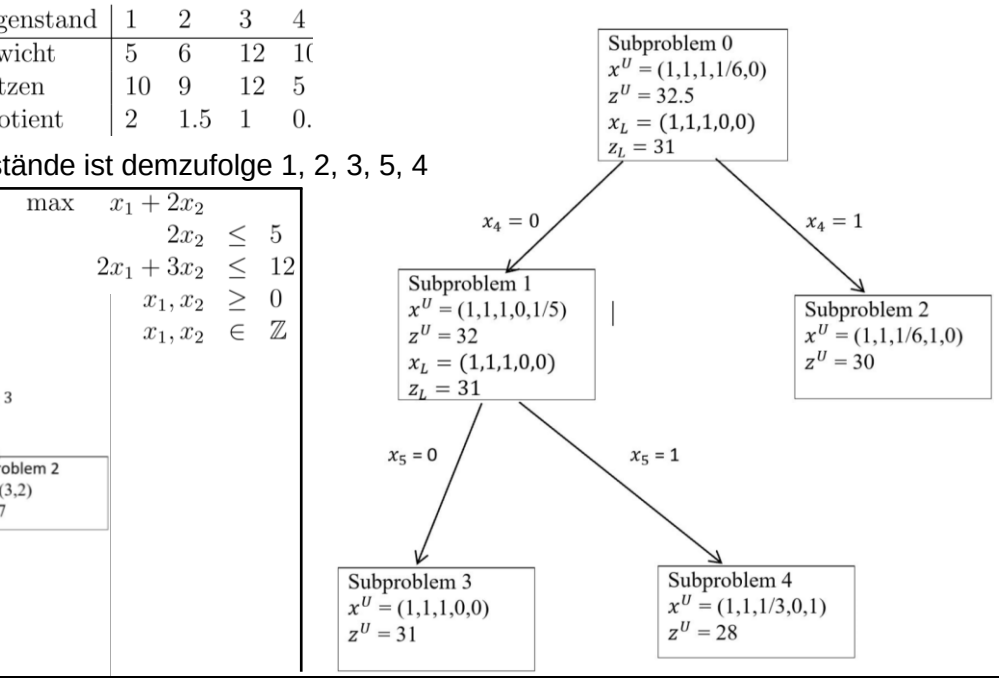
Ablauf B&B-Methode

- Werte durch Gewichte teilen, danach die erhaltenen Werte absteigend sortieren (höchster Wert zuerst, …, tiefster zuletzt)
- Danach absteigend das maximal mögliche aus den einzelnen Werte herausholen, bis ein Wert nur noch gebrochen dargestellt werden kann und die weiteren Werte (falls noch welche vorhanden) nur noch mit Null eingesetzt werden können (da Nebenbedingung sonst nicht mehr erfüllt werden kann). Dieser Wert wird dann als obere Schranke genommen.
- Der gebrochene Wert wird abgerundet und daraus den Wert als untere Schranke gewählt.
- Im ersten Schritt teilt man sich auf. Ein Weg wird den gebrochnen Wert aufrunden und der andere Abrunden. Bei beiden muss dann der Wert davor im Umzug angepasst werden, sodass die Nebenbedingungen weiter eingehalten werden.

Beispiel B&B-Methode
Maximales Gewicht 25.
Finden Sie das Maximum.

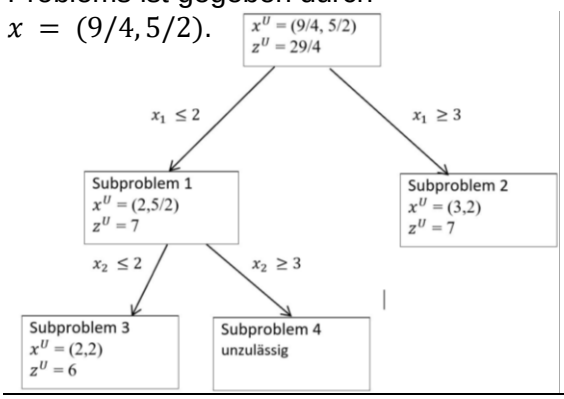
Gegenstand	1	2	3	4
Gewicht	5	6	12	1(
Nutzen	10	9	12	5
Quotient	2	1.5	1	0.

Die neue Reihenfolge der Gegenstände ist demzufolge 1, 2, 3, 5, 4



Weiteres Beispiel
Optimale Lösung des relaxierten Problems ist gegeben durch $x = (9/4, 5/2)$.

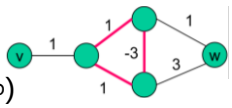
$$\begin{aligned} \max \quad & x_1 + 2x_2 & & \\ & 2x_2 & \leq & 5 \\ & 2x_1 + 3x_2 & \leq & 12 \\ & x_1, x_2 & \geq & 0 \\ & x_1, x_2 & \in & \mathbb{Z} \end{aligned}$$



Teil II: Optimierung in Graphen

5 Optimale Wege

- Gegeben ist ungerichteter oder gerichteter Graph $G = (V, E)$ mit **Knotenmenge** V und **Kantenmenge** E.
- Der Graph ist gewichtet, d.h. die Kanten $e \in E$ sind mit einem Gewicht (bzw. Länge) d_e versehen.
- Betrachtet wird das Optimierungsproblem, kürzeste oder längste Wege in diesem Graphen zu berechnen.
- Die Länge eines Weges ist definiert als die Summe der Längen seiner Kanten.
- Viele praktische Anwendungen: Routenplanung, Projektplanung, Maschinen-Einplanung, etc.
- Verschiedene Problemvarianten:
  - Bestimme den optimalen Weg zwischen zwei bestimmten Knoten ("one-to-one").
  - Bestimme den optimalen Weg von einem bestimmten Knoten aus zu allen andern Knoten ("one-to-all").
  - Bestimme den optimalen Weg zwischen allen Knotenpaaren ("all-to-all").
- Bemerkung: Zusammenhang kürzeste/längste Wege: Ein längster Weg bzgl. Gewichten d_e ist ein kürzester Weg bzgl. invertierter Gewichte $-d_e$ (und umgekehrt).
- Beachte: Voraussetzung, damit ein optimaler Weg immer definiert ist:
  - **Kürzeste** Wege: **keine negativen Zyklen** im Graphen (Bsp: Zyklus Länge -1 (rot), kürzester Weg $v \to w = -\infty$)
  - **Längste** Wege: **keine positiven Zyklen** im Graphen



- Denn: Falls ein negativer (bzw. positiver) Zyklus auf einem Weg von v nach w liegt, kann der Weg beliebig verkleinert (bzw. vergrössert) werden, indem der Zyklus beliebig oft durchschritten wird.

Bemerkung zum Lösungsaufwand eines Algorithmus:

- Lösungsaufwand ist abhängig von Problemgrösse n. (Graphenproblem: Anz. Knoten n, Anz. Kanten m)
- Aus Komplexitätstheorie: O-Notation. Ein Algorithmus hat Aufwand $O(f(n))$, wenn Anzahl elementarer Rechenschritte $\leq c \cdot f(n)$ ist für $n > n_0$ (c ist beliebige Konstante). Aufwand ist in Grössenordnung von $f(n)$.
- **Beispiel für Graphenalgorithmen:** Problemgrösse: Anzahl Knoten $n = 100$, Anzahl Kanten $m = 1000$
  Polynomialer Aufwand (d.h. effizient lösbar):
  - $O(m)$: Anzahl Rechenschritte in Grössenordnung $m = 1'000$
  - $O(n^2)$: Anzahl Rechenschritte in Grössenordnung $n^2 = 10'000$
  - $O(n \cdot m)$: Anzahl Rechenschritte in Grössenordnung $n \cdot m = 100'000$
  - $O(n^3)$: Anzahl Rechenschritte in Grössenordnung $n^3 = 1'000'000$

  Exponentieller Aufwand (d.h. „nicht effizient lösbar"):
  - $O(10^n)$: Anzahl Rechenschritte in Grössenordnung $10^n = 10^{100}$!!!

Übersicht über die wichtigsten Weg-Algorithmen:

One-to-All	Kürzeste Wege	Längste Wege
a) Keine Zyklen Gewichte beliebig	**Algorithmus (Top. Sortierung)** Aufwand: $O(m)$	**Algorithmus (Top. Sortierung)** Aufwand: $O(m)$
b1) Zyklen erlaubt Gewichte $\geq$ **0**	**Algorithmus von Dijkstra** Aufwand: $O(n^2)$	(Problem nicht definiert, falls positive Zyklen vorhanden)
b2) Zyklen erlaubt Gewichte $\leq$ **0**	(Problem nicht definiert, falls negative Zyklen vorhanden)	**Algorithmus von Dijkstra** Aufwand: $O(n^2)$ Bem: Gewichte invertieren und kürzesten Weg suchen.
c) Zyklen erlaubt	**Algorithmus von Bellman-Ford** Aufwand: $O(n \cdot m)$ Bem: Erkennt negative Zyklen.	**Algorithmus von Bellman-Ford** Aufwand: $O(n \cdot m)$ Bem: Erkennt positive Zyklen.
All-to-All		
d) Zyklen erlaubt Gewichte beliebig	**Algorithmus von Floyd-Warshall** Aufwand: $O(n^3)$ Bem: Erkennt negative Zyklen.	**Algorithmus von Floyd-Warshall** Aufwand: $O(n^3)$ Bem: Erkennt positive Zyklen.

Optimale Wege in allgemeinen Graphen

Algorithmus von Dijkstra

- **Problemstellung:** Berechne **kürzeste Wege** von **einem Knoten** s zu **allen anderen Knoten** im ungerichtetem Graphen $G = (V, E)$ („one-to-all“).
- **Voraussetzungen:** Gewichte $d_e \geq 0$
- Zyklen sind erlaubt, jedoch sind keine negativen Zyklen möglich.

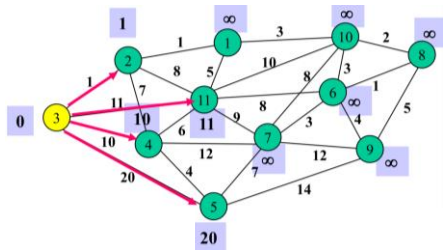
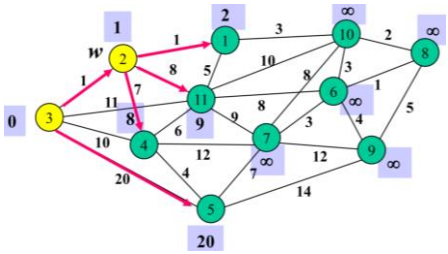
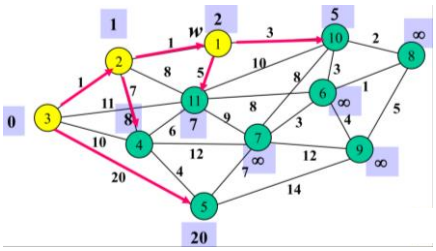
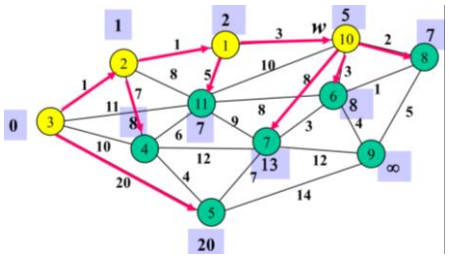
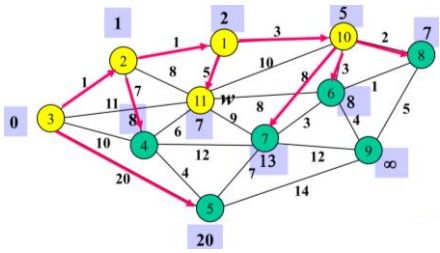
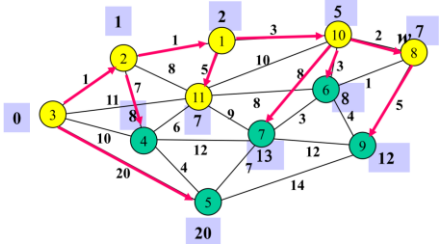
Grundidee:

- Sei S(●) die Menge der k "nächsten" Knoten von s. Nehmen an, dass ein kürzester Weg (-) zu diesen Knoten schon berechnet ist.
- Betrachten von s aus alle kürzesten Wege, die aus Menge S hinausführen direkt zu einem Knoten ausserhalb S(− − −). (die kürzesten Wege zu Knoten ausserhalb S, die nur Zwischenknoten aus S benützen)
- Sei W(−) der kürzeste unter diesen Wegen (und nehme an, er führt zu w), dann gilt:
  - w ist der $k+1$ „nächste“ Knoten von s.
  - W ist ein kürzester Weg zum Knoten w.
  - Jeder andere Weg U von s nach w muss zuerst „direkt“ aus S hinaus zu einem Knoten v (Wegstück $U_{s,v}$) und von dort weiter nach w (Wegstück $U_{v,w}$).
  - Da W der kürzeste Weg aus S hinaus ist, gilt $c(U_{s,v}) \geq c(W)$. Da Gewichte $c \geq 0$ sind, gilt $c(U_{v,w}) \geq 0$.
  - Somit ist $c(U) = c(U_{s,v}) + c(U_{v,w}) \geq c(W) + 0$, d.h. U **ist mind**. **so lang wie** W.

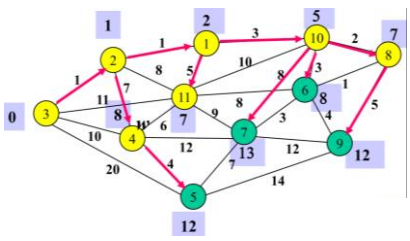
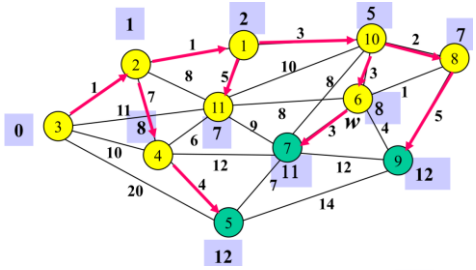
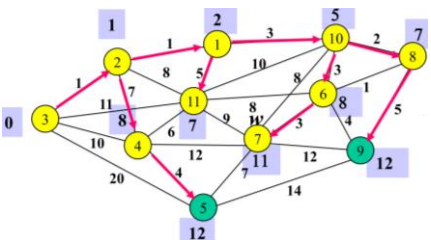
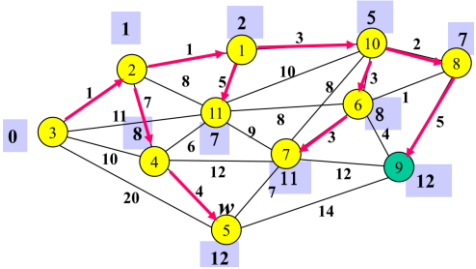
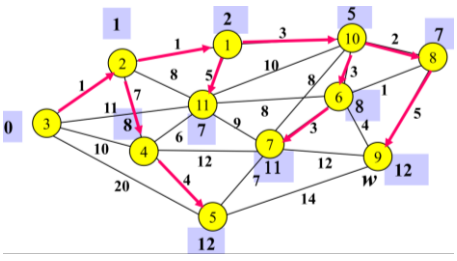
Vorgehen:

1. Beginne mit $S = \{s\}$. Berechne „die kürzesten Wege von s aus S hinaus“ (d.h. kürzeste Wege mit Zwischenknoten aus S, Längen l_v).
2. Wähle den „kürzesten Weg hinaus“ (d.h. den „nächste“ Knoten w ausserhalb S)
3. Füge w zu S hinzu: $S := S \cup \{w\}$.
4. Update von w aus die „kürzesten Wege von s aus S hinaus“: Für alle $v \in V - S$ mit $(w, v) \in E$: Wenn $l_v > l_w + d_{wv}$ dann $l_v := l_w + d_{wv}$

Beispiel: Startknoten $s = 3$

hier spielt es keine Rolle welche sieben zuerst gewählt wird für w, ob Punkt 8 oder 11 ↗ ↗ ↗

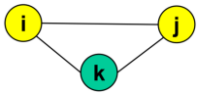






Algorithmus von Floyd-Warshall

- **Problemstellung:** Berechne **kürzeste Wege** von **jedem Knoten** i zu **jedem Knoten** j im ungerichtetem Graphen $G = (V, E)$ («all-to-all»).
- **Voraussetzungen:** keine
- **Bemerkungen:** Zyklen sind erlaubt. Gewichtung beliebig.
  - Negative Zyklen werden entdeckt.
  - Für Berechnung längster **Wege**: Vertausche «≥, >» und «≤, <», «min» und «max» (sowie ∞ und $-\infty$)

Grundidee:

- Sei n die Anzahl Knoten, im Beispiel hier $n = 11$, und die Knoten seien numeriert von 1 bis n.
- In n sukzessiven Iterationen wird für jedes Knotenpaar (i, j) die Länge eines «bedingten» kürzesten Weges berechnet, wobei die Bedingungen lauten:
  - Iteration 1: Knoten 1 ist als Zwischenknoten erlaubt
  - Iteration 2: Knoten 1,2 sind als Zwischenknoten erlaubt…
  - Iteration n: Knoten $1,2,\ldots,n$ sind als Zwischenknoten erlaubt
- Sind die Längen l_{ij}^{k-1} der kürzesten Wege KW mit Zwischenknoten $1,\ldots,k-1$ schon berechnet. Dann ist es einfach, die Längen l_{ij}^{k} der KW mit Zwischenknoten $1,\ldots,k$ zu berechnen (Dreiecksoperation): Ein KW von i nach j mit Zwischenknoten $1,\ldots,k$ ist gegeben als der kürzere der folgenden beiden Wege:
  - $P1$
  - $P1$: KW von i nach j mit Zwischenknoten $1,..,k-1$
  - $P2$: KW von i nach k mit Zwischenknoten $1,..,k-1$
  - $P3$: KW von k nach j mit Zwischenknoten $1,..,k-1$
  - $P2$ zusammengesetzt mit $P3$



Anwendung Floyd-Warshall am Beispiel mit 11 Knoten:

- Zuerst Erfassung im Excel aller Kanten
- Danach den Algorithmus laufen lassen
- Wir erhalten folgende Matrix → → →
- Matrix enthält kürzeste Wege mit der Information der «Vorgängerknoten v_{ij}^{k}» zur Bestimmung eines KW.
- Die erste Zeile bei einer Zahl zeigt dabei den kürzesten Weg auf und die zweite Zeile den Vorgängerknoten.
- **Beispiel:** Ein KW von 1 nach 7 hat die Länge 9.
- Rekonstruktion des KW:
  - Der Vorgänger von 7 ist 6 (KW von 1 nach 7).
  - Der Vorgänger von 6 ist 10 (KW von 1 nach 6).
  - Der Vorgänger von 10 ist 1. (KW von 1 nach 10).
  - KW von 1 nach 7: 1,10,6,7

11	1	2	3	4	5	6	7	8	9	10	11
1	0	1	2	8	12	6	9	5	10	3	5
	1	*1*	*2*	*2*	*4*	***10***	***6***	*10*	*6*	***1***	*1*
2	1	0	1	7	11	7	10	6	11	4	6
	2	*2*	*2*	*2*	*4*	*10*	*6*	*10*	*6*	*1*	*1*
3	2	1	0	8	12	8	11	7	12	5	7
	2	*3*	*3*	*2*	*4*	*10*	*6*	*10*	*6*	*1*	*1*
4	8	7	8	0	4	14	11	13	18	11	6
	2	*4*	*2*	*4*	*4*	*7*	*5*	*10*	*5*	*1*	*4*
5	12	11	12	4	0	10	7	11	14	13	10
	2	*4*	*2*	*5*	*5*	*7*	*5*	*6*	*5*	*6*	*4*
6	6	7	8	14	10	0	3	1	4	3	8
	10	*1*	*2*	*5*	*7*	*6*	*6*	*6*	*6*	*6*	*6*
7	9	10	11	11	7	3	0	4	7	6	9
	10	*1*	*2*	*5*	*7*	*7*	*7*	*6*	*6*	*6*	*7*
8	5	6	7	13	11	1	4	0	5	2	9
	10	*1*	*2*	*2*	*7*	*8*	*6*	*8*	*8*	*8*	*6*
9	10	11	12	18	14	4	7	5	0	7	12
	10	*1*	*2*	*5*	*9*	*9*	*6*	*9*	*9*	*6*	*6*
10	3	4	5	11	13	3	6	2	7	0	8
	10	*1*	*2*	*2*	*7*	*10*	*6*	*10*	*6*	*10*	*1*
11	5	6	7	6	10	8	9	9	12	8	0
	11	*1*	*2*	*11*	*4*	*11*	*11*	*6*	*6*	*1*	*11*

Initialisierung Algorithmus:

- Nummeriere die Knoten von 1 bis n. Für jedes Paar $(i, j), 1 \leq i, j \leq n$, setze:
- $l_{ij}^{0} := \begin{cases} d_{ij} & falls\ (i,j) \in E \\ \infty & falls\ (i,j) \notin E\ und\ i \neq j, \\ 0 & falls\ i = j \end{cases}$ $\quad v_{ij}^{0} := \begin{cases} i & falls\ (i,j) \in E \\ \emptyset & falls\ (i,j) \notin E\ und\ i \neq j \\ i & falls\ i = j \end{cases}$

Iterationen:

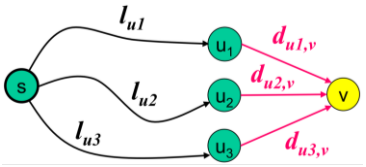
- Für alle Zwischenknoten $k = 1,2,\ldots,n$ führe aus: Für jedes Paar $(i, j), 1 \leq i, j \leq n$, mit $i \neq k, j \neq k$, setze:
- $l_{ij}^{k} := \begin{cases} l_{ij}^{k-1} & falls\ l_{ik}^{k-1} + l_{kj}^{k-1} \geq l_{ij}^{k-1} \\ l_{ik}^{k-1} + l_{kj}^{k-1} & sonst \end{cases}$, $v_{ij}^{k} := \begin{cases} v_{ij}^{k-1} & falls\ l_{ik}^{k-1} + l_{kj}^{k-1} \geq l_{ij}^{k-1} \\ v_{kj}^{k-1} & sonst \end{cases}$

Optimale Wege in azyklischen Graphen

- Falls Graph **keine Zyklen** enthält, ist Problem des optimalen Weges (one-to-all) einfach zu lösen (mit beliebigen Kantengewichten).
- Aufwand ist $O(m)$. Anz. Kanten m, **minimaler Aufwand**, da jede Kante **nur einmal** betrachtet werden muss.
- Das Problem längster Weges in azyklischen (gerichteten) Graphen tritt beispielsweise auf bei der Einplanung von Aktivitäten im Projektmanagement sowie als Teilproblem bei Maschinenbelegungsproblemen.

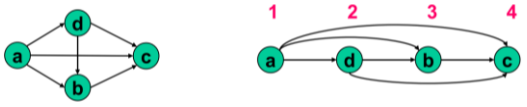
Algorithmus für azyklische Graphen

- Problemstellung: Berechne längste Wege von Quelle s zu allen anderen Knoten im gerichtetem Graphen $G = (V, E)$ («one-to-all»).
- Voraussetzungen: **keine Zyklen**
- Bemerkungen: Gewichtung beliebig.
- Für Berechnung kürzester Wege: Vertausche «≥, >» und «≤, <», «max» und «min» (sowie ∞ und −∞)
- **Grundidee:** Betrachte einen Knoten v und nehme an, dass die längsten Wege l_u von s zu allen Vorgängern u von v bekannt sind. Dann ist der längste Weg von s nach v gegeben durch: $l_v = max\{l_u + d_{uv}: u\ ist\ Vorgänger\ von\ v\}$
- Wähle Knotenreihenfolge zur Berechnung der längsten Wege LW so, dass stets alle Vorgänger schon berechnet sind (→ **Top. Sortierung**)

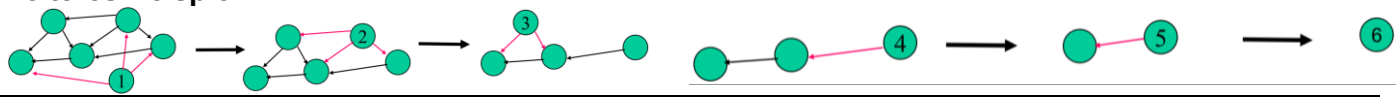


Topologische Sortierung:

- Jeder azyklische Graph hat eine Topologische Sortierung: Die Knoten können so von 1 bis n numeriert werden, dass gilt: Wenn $(i, j) \in E$, dann ist $i < j$.
- **Vorgehen** zur Bestimmung einer Topologischen Sortierung:
  - 1. Wähle einen Knoten v, der **keine Vorgänger** hat und gebe ihm die nächste Nummer. (Beachte: v **existiert immer**, **wenn keine Zyklen** vorhanden!)
  - 2. Entferne Knoten v (mit seinen Bogen) und gehe zu Schritt 1.
- **Beispiel:** Topologische Sortierung ist in Grafik rot eingezeichnet



- **Weiteres Beispiel:**



Algorithmus: Längste Wege in azyklischen Graphen

Initialisierung (Annahme: s hat keine Vorgänger):

- $S \coloneqq \{s\}, l_s \coloneqq 0, (pred_s \coloneqq \emptyset)$
- FOR ALL $v \in V - s\ DO\ l_v \coloneqq -\infty\ (pred_v \coloneqq \emptyset)$

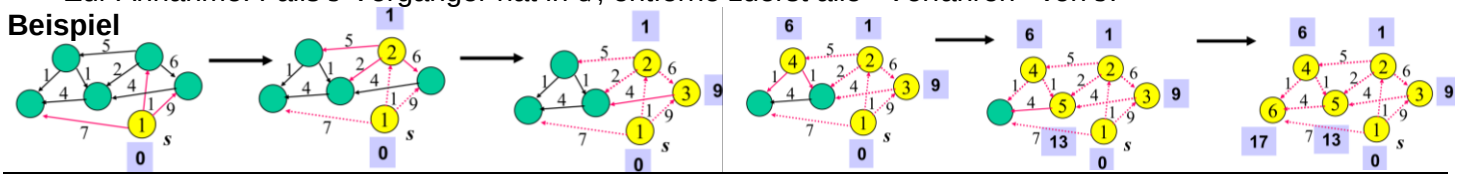
Iterationen:

- $WHILE\ S \neq V\ DO$
  Bestimme $v \in V - S$ so dass v keine Vorgänger hat in $G[V - S]$.
  $S \coloneqq S \cup \{v\}$
  Setze $l_v = max\{l_u + d_{uv}: u\ ist\ Vorgänger\ von\ v\}(pred_v \coloneqq u^*)$

Bemerkungen:

- $G[V - S]$ ist der resultierende Graph, wenn die Knoten von S entfernt werden.
- u^* ist derjenige Vorgänger, welcher das Maximum ergibt.
- Beachte: Wegen Topol. Reihenfolge sind alle l_u schon berechnet für $u \in S$.
- Zur Annahme: Falls s Vorgänger hat in G, entferne zuerst alle «Vorfahren» von s.

Beispiel



Eine LP-Formulierung für das Wegproblem

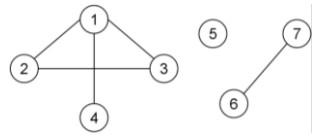
- **Problem:** Kürzester Weg von Quelle s nach Senke t in gerichtetem s,t-Graphen $G = (V, E)$ mit positiven Bogengewichten $c_{vw} > 0, (v, w) \in E$
- $\sum_{(v,w)\in E} c_{vw} x_{vw}$ (0)
- $\sum_{w\in V:(v,w)\in E} x_{vw} - \sum_{u\in V:(u,v)\in E} x_{uv} \begin{cases} 1 & v = s \\ 0 & v \in V - \{s, t\} \\ -1 & v = t \end{cases}$ «Flow Balance» (1)
- $0 \leq x_{vw} \leq 1, (v, w) \in E$ und x_{vw} integer → Binary/Integer Conditions, Variable ist binär $(0,1)$ (2, 3)
- **Es gilt:** Jede Lösung $x \in R|E|$ der Nebenbedingungen (1-3) ist Inzidenzvektor einer Bogenmenge $S^x \in E$, welche in einen s,t-Pfad und eine Menge von Zyklen zerlegt werden kann. Die Umkehrung gilt ebenfalls.
- Da alle Gewichte positiv, entspricht optimale Lösung des ILP (0-3) somit einem (minimalen) s,t-Pfad.
- **Es kann gezeigt werden:** Das durch (1, 2) definierte Polyeder P ist ganzzahlig, d.h. alle seine Eckpunkte sind ganzzahlig: $P = \{x \in R^{|E|}: x\ erfüllt\ (1, 2)\}$
- Somit kann die Ganzzahligkeitsforderung (3) weggelassen und das Problem als LP (0-2) gelöst werden.
- Das Polyeder P ist also die konvexe Hülle aller Inzidenzvektoren $x \in \{0,1\}^{|E|}$, die einer Kantenmenge $S^x \in E$ entsprechen, welche in einen s,t-Pfad und eine Menge von Zyklen zerlegt werden kann.
- Das LP (0-2) gehört zur Klasse der sogenannten Flussprobleme.
- Flussprobleme bilden wichtigen Bereich kombinatorischen Optimierung → haben viele prakt Anwendungen.

Ungerichtete Graphen

- Gegeben ist ein **ungerichteter Graph** $G = (V, E)$ mit endliche **Knotenmenge** V und **Kanten** E.
- Ein Bogen $e \in E$ ist ein ungerichtetes Paar $e = (v, w)$ von Knoten $v, w \in V$, hier gilt $e = (v, w) = (w, v)$
- Für eine Kante $e \in E$ mit $e = (v, w)$, v, w sind Endknoten.
- v, w werden als **benachbart** oder **adjazent** bezeichnet. e verbindet die Knoten v, w, e ist **inzident** mit v, w
- Ein Bogen der Form $e = (v, v)$ heisst **Schlinge** (loop). Ein **schlichter Graph** hat **keine Schlingen**.
- **Adjazenzmatrix** von Graph $G = (V, E)$ ist $A \in \{0,1\}^{V \times V}$, deren Elemente gegeben sind $a_{vw} = \begin{cases} 1, & (v,w) \in E \\ 0, & sonst \end{cases}$
  Diese Matrix ist eine **symmetrische** Matrix! Gilt **nicht** für gerichtete Graphen.
- **Inzidenzmatrix** ist $A \in \{0,1\}^{V \times E}$, deren Elemente gegeben sind $a_{ve} = \begin{cases} 1, & v\ ist\ inzident\ mit\ e \\ 0, & sonst \end{cases}, v \in V, e \in E$
- Der **Grad** (Valenz) $deg(v)$ eines Knotens $v \in V$ ist die Anzahl Kanten, welche mit v inzident sind.
- Ein Knoten $v \in V$ heisst **isoliert**, falls $deg(v) = 0$ und **hängend**, falls $deg(v) = 1$.
- Für **Knotenmenge** $S \subseteq V$ bezeichnet $\delta(S)$ die Menge der aus S hinausführenden Kanten → die Kanten, welche einen Endknoten in S und den andern ausserhalb von S haben: $\delta(S) = \{(v, w) \in E : v \in S, w \in V - S\}$
- $\gamma(S)$ bezeichnet die Menge der in S **liegenden Kanten** → die Kanten, welche beide Endknoten in S haben, → $\gamma(S) = \{(v, w) \in E : v, w \in S\}$. Für $\delta(\{v\}), v \in V$, wird kurz $\delta(v)$ geschrieben, und es gilt $deg(v) = |\delta(v)|$
- Ein Graph $G' = (V', E')$ ist ein **Subgraph** von $G = (V, E)$, falls $V' \subseteq V$ und $E' \subseteq E$.
- Falls $V' = V$, heisst G' ein **aufspannender Subgraph** (spanning subgraph) von G.
- G' ist der durch die Knotenmenge V' **induzierte** Subgraph $G[V']$ von G, falls $E' = \gamma(V')$.
- Ein Kantenzug (walk) in $G = (V, E)$ ist eine **alternierende Folge** von Knoten/Kanten der Form $P = (v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k)$ wobei $k \geq 0$, $v_i \in V$ für $i = 0, \ldots, k$ und $e_i = (v_{i-1}, v_i) \in E$ für $i = 1, \ldots, k$.
- Falls keine Missverständnisse möglich sind, wird ein **Kantenzug** identifiziert durch den zugehörigen Subgraphen $G_P = (V_P, E_P)$ mit $V_P = \{v_0, v_1, \ldots, v_k\}$ und $E_P = \{e_1, e_2, \ldots, e_k\}$, oder auch nur durch die Knotenmenge V_P oder die Kantenmenge E_P. Es wird gesagt, dass P die Knoten v_0, v_k verbindet und dass P von v_0 (bzw. v_k) nach v_k (bzw. v_0) führt. Die Kantenanzahl k wird auch als die Länge von P bezeichnet.
- Der **Kantenzug** P heisst **geschlossen**, falls $v_0 = v_k$, und andernfalls **offen**.
- P ist ein **einfacher Kantenzug** (simple walk), falls abgesehen von den Endknoten v_0 und v_k alle Knoten **paarweise verschieden** sind, d.h. falls $v_i = v_j$ für alle Knotenpaare $\{(i, j) : 0 \leq i < j \leq k\} - \{(v_0, v_k)\}$.
- Ein **offener**, **einfacher Kantenzug** ist ein **Weg** (path), und ein **geschlossener**, **einfacher Kantenzug** ein **Kreis** (circuit). Ein **Graph ohne Kreise** heisst **azyklisch.**
- Beachte, dass ein Graph, welcher einen offenen (bzw. geschlossenen) Kantenzug P von v_0 nach v_k enthält, stets auch einen Weg (bzw. einen Kreis) P' von v_0 nach v_k enthält mit $V_{P'} \subseteq V_P$ und $E_{P'} \subseteq E_P$.
- Ein Graph $G = (V, E)$ heisst **zusammenhängend** (connected), falls **jedes Knotenpaar** $v, w \in V$ durch einen **Weg verbunden** ist. Eine Zusammenhangskomponente (connected component) von G ist ein zusammenhängender, induzierter Subgraph $G[V'], V' \neq \emptyset$, welcher bezüglich der Knotenmenge maximal ist, d.h. $G[V']$ ist zusammenhängend und es existiert kein zusammenhängender induzierter Subgraph $G[V'']$ mit $V' \subset V''$.
- Ein **bipartiter Graph** (bipartite graph) ist ein Graph $G = (V, E)$, dessen **Knotenmenge** sich **aufteilen** lässt in zwei **disjunkte Mengen** $S, T \subseteq V, S \cap T = \emptyset$, so dass **jede Kante** einen **Endknoten** in S und den **andern Endknoten** in T hat, d.h. $E \subseteq \{(v, w) : v \in S, w \in T\}$.
- Ein Graph $G = (V, E)$ heisst **vollständig** (complete), falls **jedes Knotenpaar mit** einer **Kante verbunden** ist, d.h. falls $E = \{(i, j) : i, j \in V, i < j\}$. Folglich gilt $|E| = |V|(|V| - 1)/2$.
- Eine Knotenmenge $V' \subseteq V$ in einem Graphen $G = (V, E)$ ist eine **Clique**, falls $G[V']$ vollständig ist, und eine **unabhängige Menge** (stable set), falls $G[V']$ eine **leere Kantenmenge** hat. In einer **Clique** ist somit jedes Knotenpaar **adjazent**, und in einer **unabhängigen Menge** ist jedes Knotenpaar **nicht adjazent**. Eine Kantenmenge $E' \subseteq E$ ist ein **Matching**, falls keine **zwei Kanten** aus E' einen **gemeinsamen Endknoten** haben.

Beispiel: Ungerichtete Graph $G = (V, E)$

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



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

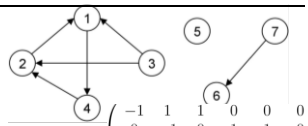
- Menge $V = \{1, 2, \ldots, 7\}$,
- Kantenmenge $E = \{(1, 2), (1, 3), (1, 4), (2, 3), (6, 7)\}$
- Adjazenzmatrix (obere)
- Inzidenzmatrix (untere, weiter rechts), Kanten lexikographisch
- Knotengrade in G gilt bspw. $deg(1) = 3$, $deg(2) = 2$, $deg(6) = 1$ und $deg(5) = 0$
- Menge der hängenden Knoten von G ist gegeben durch $\{4, 6, 7\}$
- Menge der isolierten Knoten durch $\{5\}$.
- Für Knotenmenge $S = \{1, 3, 4\}$ gilt $\delta(S) = \{(1, 2), (2, 3)\}$ und $\gamma(S) = \{(1, 3), (1, 4)\}$, für $S = \{6, 7\}$ gilt $\delta(S) = \emptyset$ und $\gamma(S) = \{(6, 7)\}$, und für den Knoten $v = 1$ gilt $\delta(v) = \{(1, 2), (1, 3), (1, 4)\}$
- Ein Subgraph von G ist bspw. gegeben $G' = (V', E')$ mit $V' = \{1, 2, 3\}$ und $E' = \{(1, 3), (2, 3)\}$. Beachte, dass bspw. $G' = (V', E')$ mit $V' = \{1, 2\}$ und $E' = \{(1, 3), (2, 3)\}$ kein Graph, somit auch kein Subgraph von G ist.

Gerichtete Graphen

- Gegeben ist ein **gerichteter Graph** $G = (V, E)$ mit **Knotenmenge** V und **Bogen** E.
- Ein Bogen $e \in E$ ist ein gerichtetes Paar $e = (v, w)$ von Knoten $v, w \in V$
- Die Notationen sind im **Gegensatz** zum ungerichteten Graph hier **nicht identisch** $e = (v, w) \neq e = (w, v)$
- Ein Bogen $e \in E$ mit $e = (v, w)$, v, w sind Endknoten, wobei $v = tail(e)$ Start und $w = head(e)$ Ende sind.
- v, w werden als **benachbart** oder **adjazent** bezeichnet. e verbindet die Knoten v, w, e ist **inzident** mit v, w
- Ein Bogen der Form $e = (v, v)$ heisst **Schlinge** (loop). Ein **schlichter Graph** hat **keine Schlingen**.
- **Adjazenzmatrix** von Graph $G = (V, E)$ ist $A \in \{0,1\}^{V \times V}$, deren Elemente gegeben sind $a_{vw} = \begin{cases} 1, & (v,w) \in E \\ 0, & sonst \end{cases}$
- **Inzidenzmatrix** ist $A \in \{-1,0,1\}^{V \times E}$, deren Elemente gegeben sind $a_{ve} = \begin{cases} -1, & v = tail(e) = Start \\ 1, & v = head(e) = Ende \\ 0, & sonst \to v \in V, e \in E \end{cases}$
- Der **Ausgangsgrad** $\deg^+(v)$ (outdegree) eines Knotens $v \in V$ ist die Anzahl Bogen, welche aus v hinausgehen und der **Eingangsgrad** $\deg^-(v)$ (indegree) ist die Anzahl Bogen, welche in v hineingehen.
- Ein **Knoten** $v \in V$ heisst **isoliert**, falls $\deg^+(v) = \deg^-(v) = 0$.
- Ein **Knoten** $v \in V$ ist eine **Quelle**, falls $\deg^-(v) = 0$ und eine **Senke**, falls $\deg^+(v)$.
- Für **Knotenmenge** $S \subseteq V$ ist $\delta^+(S)$ die Menge aus S **hinausführenden** Bogen: $\delta^+(S) = \{(v, w) \in E : v \in S, w \in V - S\}$ und $\delta^-(S)$ die Menge in S **hineinführenden** Bogen: $\delta^-(S) = \{(v, w) \in E : v \in V - S, w \in S\}$.
- $\gamma(S)$ ist Menge in S **liegenden Bogen** (haben **beide Endknoten** in S), d.h. $\gamma(S) = \{(v, w) \in E : v, w \in S\}$
- Für $\delta^+(\{v\})$, $\delta^-(\{v\})$, $v \in V$, wird $\delta^+(v), \delta^-(v)$ geschrieben, es gilt $\deg^+(v) = |\delta^+(v)|$, $\deg^-(v) = |\delta^-(v)|$.
- Ein Graph $G' = (V', E')$ ist ein **Subgraph** von $G = (V, E)$, falls $V' \subseteq V$ und $E' \subseteq E$.
- Falls $V' = V$, heisst G' ein **aufspannender Subgraph** von G. G' ist der durch die Knotenmenge V' **induzierte Subgraph** von G, falls $E' = \gamma(V')$. Der von V' induzierte Subgraph in G wird auch als $G[V']$ dargestellt.
- Ein Bogenzug (directed walk) in $G = (V, E)$ ist eine **alternierende Folge** von Knoten und Bogen der Form $P = (v_0, e_1, v_1, e_2, v_2, \dots, e_k, v_k)$ wobei $k \geq 0$, $v_i \in V$ für $i = 0, \dots, k$ und $e_i = (v_{i-1}, v_i) \in E$ für $i = 1, \dots, k$.
- Falls keine Missverständnisse möglich sind, wird ein **Bogenzug** identifiziert durch den zugehörigen Subgraphen $G_P = (V_P, E_P)$ mit $V_P = \{v_0, v_1, \dots, v_k\}$ und $E_P = \{e_1, e_2, \dots, e_k\}$, oder auch nur durch die Knotenmenge V_P oder die Bogenmenge E_P. Es wird gesagt, dass P die Knoten v_0, v_k verbindet und dass P von v_0 (bzw. v_k) nach v_k (bzw. v_0) führt. Die **Bogenanzahl** k wird auch als die **Länge** von P bezeichnet.
- Der Bogenzug P heisst **geschlossen**, falls $v_0 = v_k$, und andernfalls **offen**. P ist ein **einfacher Bogenzug** (simple (directed) walk), falls abgesehen von den Endknoten v_0 und v_k **alle Knoten paarweise verschieden** sind, d.h. falls $v_i \neq v_j$ für alle Knotenpaare $\{(i, j) : 0 \leq i < j \leq k\} - \{(v_0, v_k)\}$.
- **Offener**, **einfacher Bogenzug** ist **Pfad**, und **geschlossener**, **einfacher** Bogenzug ein **Zyklus**.
- Beachte, dass ein Graph, welcher einen **offenen** (bzw. geschlossenen) **Bogenzug** P von v_0 nach v_k enthält, stets auch einen **Pfad** (bzw. Zyklus) P' von v_0 nach v_k **enthält** mit $V_{P'} \subseteq V_P$ und $E_{P'} \subseteq E_P$.
- Ein Graph $G = (V, E)$ heisst **stark zusammenhängen**, falls für jedes Knotenpaar $v, w \in V$ ein **Pfad** von v nach w **existiert**. G heisst **schwach zusammenhängend**, falls der entsprechende ungerichtete Graph zusammenhängend ist, d.h. falls in G ein Weg existiert, wenn man von den Bogenrichtungen absieht.
- Eine **starke** (bzw. schwache) **Zusammenhangskomponente** (strongly resp. weakly connected component) von G ist ein stark (bzw. schwach) zusammenhängender, induzierter Subgraph $G[V']$, $V' \neq \emptyset$, welcher bezüglich der Knotenmenge **maximal** ist, d.h. $G[V']$ ist **stark** (bzw. schwach) zusammenhängend und es existiert **kein stark** (bzw. schwach) **zusammenhängender induzierter Subgraph** $G[V'']$ mit $V' \subset V''$.
- Ein **Arboreszenz** ist ein **schwach zusammenhängender**, **azyklischer** Graph $G = (V, E)$ mit $\deg^-(v) \leq 1$ für alle $v \in V$. Eine Arboreszenz $G = (V, E)$ hat **nur** einen Knoten $r \in V$ mit $\deg^-(r) = 0$, welcher die **Wurzel** von G genannt wird. Ein **Branching** ist **azyklischer** Graph $G = (V, E)$, so dass $\deg^-(v) \leq 1$ für alle $v \in V$, d.h. ein Graph, dessen schwach **zusammenhängenden** Komponenten Arboreszenzen sind. Eine aufspannende **Arboreszenz** in einem Graphen G ist ein aufspannender Subgraph von G, welcher Arboreszenz ist.
- Ein **Graph** $G = (V, E)$ heisst **vollständig** (complete), falls **jedes geordnete Knotenpaar** mit **einem Bogen verbunden** ist, d.h. falls $E = \{(i, j) : i, j \in V, i \neq j\}$. Folglich gilt $|E| = |V|(|V| - 1)$.

Beispiel: gerichtete Graph $G = (V, E)$

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



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

- Knotenmenge $V = \{1, 2, \dots, 7\}$,
- Bogenmenge $E = \{(1, 4), (2, 1), (3, 1), (3, 2), (4, 2), (7, 6)\}$
- Adjazenzmatrix (obere) und Inzidenzmatrix (untere, weiter rechts)
- Knotengrade: $\deg^+(1) = 1$, $\deg^-(1) = 2$, $\deg^+(3) = 2$, $\deg^-(3) = 0$, $\deg^+(5) = \deg^-(5) = 0$
- Menge der Quellen bzw. Senken ist $\{3, 5, 7\}$ bzw. $\{5, 6\}$, und Menge isolierten Knoten $\{5\}$
- Für Knotenmenge $S = \{1, 3, 4\}$ gilt $\delta^+(S) = \{(3, 2), (4, 2)\}, \delta^-(S) = \{(2, 1)\}$, $\gamma(S) = \{(1, 4), (3, 1)\}$, für $S = \{6, 7\}$ gilt $\delta^+(S) = \delta^-(S) = \emptyset$ und $\gamma(S) = \{(7, 6)\}$, und für $v = 1$ gilt $\delta^+(v) = \{(1, 4)\}$, $\delta^-(v) = \{(2, 1), (3, 1)\}$.
- Ein Subgraph von G ist bspw. gegeben durch $G' = (V', E')$ mit $V = \{1, 2, 3\}$ und $E' = \{(3, 1), (3, 2)\}$

6 Optimale Zyklen: Das Traveling-Salesman-Problem (TSP)

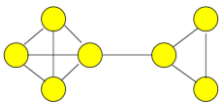
- **Problem:** Ein Handelsreisender („Traveling Salesman“) muss die Städte $1, 2, 3, .., n$ in beliebiger Reihenfolge besuchen. (Wir nehmen an, dass er z.B. in Stadt 1 startet und am Ende wieder dorthin zurückkehrt.)
- Kurz ausgedrückt: das TSP entspricht einer Suche nach einem **kürzesten hamiltonschen Zyklus** (oder kürzesten Hamiltonkreis) in einem vollständigen Graphen.

Hamiltonsche Wege und Zyklen, TSP

- In diesem Kapitel werden ausschliesslich **un**gerichtete Graphen $G = (V, E)$ betrachtet.
- Seien $v, w \in V$ zwei unterschiedliche Knoten: Ein Hamiltonscher Weg von v nach w ist ein Weg von v nach w, welcher alle Knoten von G genau einmal besucht.
- Ein Hamiltonscher Zyklus ist ein Zyklus, welcher alle Knoten von G genau einmal besucht.
- Seien $d_e, e \in E$, die Kantengewichte in G.
- Zwei fundamentale Probleme im Graphen $G = (V, E)$:
  - Bestimme einen Hamiltonschen Weg minimaler Länge von v nach w.
  - Bestimme einen Hamiltonschen Zyklus minimaler Länge.
- Beispiel mit Stadt von oben: Sei $D = (d_{ij})$ die Matrix der Distanzen zwischen allen Städten (i, j). Dann lautet das TSP: Bestimme eine Reihenfolge $i(1), i(2), i(3), \ldots, i(n)$ der Städte, so dass die gesamte zurückgelegte Distanz minimal ist: $d_{i(1),i(2)} + d_{i(2),i(3)} + \cdots + d_{i(n-1),i(n)} + d_{i(n),i(1)} \rightarrow min$
- Das TSP entspricht dem Problem des minimalen Hamiltonschen Zyklus (in vollständigen Graphen G).
- Symmetrisches (bzw. ungerichtetes) TSP: Distanzmatrix ist symmetrisch, d.h. $d_{ij} = d_{ji}$ für alle (i, j).
- Asymmetrisches (bzw. gerichtetes) TSP: Distanzmatrix ist asymmetrisch.

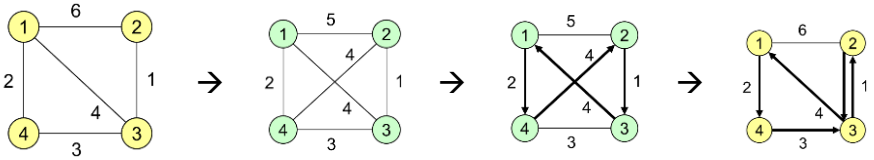
Formulierung des (symmetrischen) TSP im ungerichteten Graphen:

- Distanzmatrix $D = (d_{ij})$, symmetrisch
- Graph $G = (V, E)$, ungerichtet, mit:
  - V: Menge der Städte, $V = \{1, 2, \ldots, n\}$
  - E: Menge der Kanten, $E = \{(i, j): i, j \in V, i \neq j\}$ (G ist vollständig, d.h. alle Knotenpaare sind mit einer Kante verbunden.)
  - Kantenlängen d_{ij} für $(i, j) \in E$
- Bestimme einen Hamiltonschen Zyklus („Tour“) minimaler Länge in G
- Das TSP kann auch in einem beliebigen (**unvollständigen**) Graphen definiert werden.
- In diesem Fall ist es möglich, dass kein Hamiltonscher Zyklus (Tour) existiert. Beispiel:



- Ein TSP im unvollständigen Graph lässt sich immer in vollständigen Graphen formulieren:
  - Füge die fehlenden Kanten mit einem genügend grossen Gewicht M ein, z.B. $M > \sum\{d_e: e \in E\}$.
  - Eine optimale Tour enthält genau dann solche zusätzlichen Kanten, wenn keine Tour im ursprünglichen Graphen existiert!
- Eine häufige Version des TSP in der Praxis: «Graphisches TSP»: Jede Stadt (Knoten) muss **mindestens einmal** besucht werden. Für Tourenplanung in der Praxis meist diese Version relevant!
- Graphisches TSP kann als «normales» TSP formuliert werden: Konstruiere einen vollständigen Graphen G', dessen Kantenlängen d'_{ij} der Länge eines kürzesten Weges von i nach j entsprechen und bestimme eine Tour in G'.

Beispiel: Graphisches TSP



Optimale «Tour»: $(1, 4, 3, 2, 3, 1)$
Länge 11.

Gelb: Graph $G = (V, E')$, Gewichte d_{ij} Grün: Graph $G' = (V, E')$ vollständig, Gewichte d'_{ij}: kürzeste Wege

Einige Heuristiken für das TSP

- TSP (d.h. Bestimmung Hamiltonschen Zyklus min. Länge) ist – ausser in Spezialfällen ein «schwieriges» (NP-vollständiges) Problem, d.h. es sind **keine Algorithmen** mit polynomialem Lösungsaufwand **bekannt**.
- Sogar das Problem, festzustellen, ob Graph einen Hamiltonschen Zyklus besitzt, ist schon NP-vollständig!
- Aus diesem Grunde wurden schon früh zahlreiche **Heuristiken** für das TSP **entwickelt**.

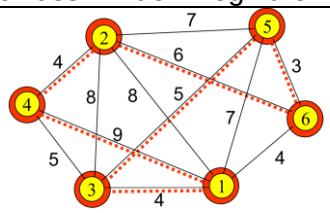
Ziel: Möglichst gute (nicht unbedingt optimale) Tour zu konstruieren.

Konstruktive Heuristiken

A «Nearest Neighbor» Heuristik

- Fixieren bei Initialisierung den Startknoten und setzen ihn in Menge S (beinhaltet bereits besuchten Knoten).
- In jedem Iterationsschritt wird der nächste zu besuchende Knoten gesucht. Dazu sucht man, ausgehend aus dem Knoten $i(k)$, wo man sich gerade befindet, den am nächsten (also denjenigen mit der kleinsten Distanz zu $i(k)$) unbesuchten Knoten j. Dieser neue Knoten j wird besucht und als solches markiert (geht in Menge S) und damit ist der Iterationsschritt fertig. Es wird so lange iteriert, bis alle Knoten besucht sind. Zum Schluss wir der Weg zu einer Tour geschlossen (die Kante $(i(n), i(1))$ wird dem Weg zugefügt).

Beispiel:



(Startknoten ist 4.) Reihenfolge, in welcher die Knoten besucht werden: 4,2,6,5,3,1, Tour der Länge 31 (liefert hier nicht die optimale Lösung!)

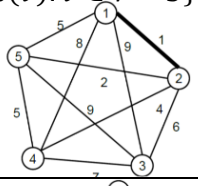
B) «Insertion» Heuristiken

- Starten mit Teiltour (nur 2 Knoten), in die sukzessive ein geeigneter Knoten eingefügt wird, bis Tour entsteht.
- Initialisierung werden zwei Knoten v und w gewählt und Teiltour (v, w, v) konstruiert.
- Man **könnte**, zum Beispiel, diejenigen Knoten v, w wählen, die zueinander am nächsten liegen.
- Die Menge S bezeichnet die Knoten, die bereits in der Teiltour auftreten.
- Der Iterationsschritt besteht aus zwei Teilen:
  - (a) Ein geeigneter Knoten wird gesucht, der in die aktuelle Teiltour eingefügt werden soll.
  - (b) Es wird geeignete Stelle in der aktuellen Teiltour gesucht, wo dieser Knoten eingefügt werden soll.
- Der eingefügte Knoten wird Menge S beigefügt. Falls alle Knoten eingefügt worden sind, endet Verfahren.
- Müssen bestimmen, was «geeigneter» Knoten und was eine geeignete Stelle ist. Haben einen Graphen mit einer Teiltour. Zu jedem Knoten $j \in V - S$, der nicht in der Teiltour $\big(i(1), i(2), \dots, i(k)\big)$ liegt, können wir seine Distanz zur Teiltour $dist(j)$ wie folgt definieren: $dist(j) \coloneqq \min\{d_{i(r),j} : r = 1, \dots, k\}$
- D.h.: vergleichen Distanzen vom Knoten j zu allen Knoten der Teiltour, und wählen die kleinste davon.

B1) «Nearest Insertion»

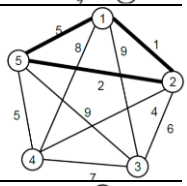
- Nearest und Farthest Insertion unterscheiden sich durch die **verschiedene Knoten**, die sie zum **Einfügen auswählen**. Die **Einfügestelle** wird auf die **gleiche Weise** für beide Heuristiken bestimmt.
- Bestimmung Distanzen zwischen potenziellen Einfügeknoten v und der aktuellen Teiltour. Die kleinste von den Distanzen bestimmt den Knoten $j \in V - S$, der eingefügt werden soll: $dist(j) = \min\{dist(v) : v \in V - S\}$

Beispiel
Start mit vorzugebender Teiltour $T = (1, 2, 1)$, i.d.R. kürzester Weg in Graph suchen

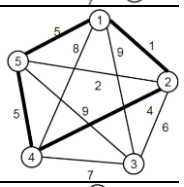


Iteration 1: $S = \{1,2\}, V - S = \{3,4,5\}$ alle möglichen nächsten Knoten
$dist(3) = \min\{9, 6\} = 6$, $dist(4) = \min\{8, 4\} = 4$, $dist(5) = \min\{5, 2\} = 2$ ← tiefster Wert
Wo soll Wert eingefügt werden: $\Delta(5, 1, 2) = 5 + 2 - 1 = 6$ oder $\Delta(5, 2, 1) = 2 + 5 - 1 = 6$

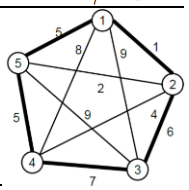
→ Neue Teiltour $T = (1, 5, 2, 1)$, Länge 8



Iteration 2: $S = \{1,2,5\}, V - S = \{3,4\}$ alle möglichen nächsten Knoten
$dist(3) = \min\{9, 6, 9\} = 6$, $dist(4) = \min\{8, 5, 4\} = 4$ ← tiefster Wert
Wo Wert einfügen: $\Delta(4, 1, 5) = 8 + 5 - 5 = 8$ oder $\Delta(4, 5, 2) = 5 + 4 - 2 = 7$
oder $\Delta(4, 2, 1) = 4 + 8 - 1 = 11$
→ Neue Teiltour $T = (1, 5, 4, 2, 1)$, Länge 15



Iteration 3: $S = \{1, 2, 4, 5\}, V - S = \{3\}$ alle möglichen nächsten Knoten
$dist(3) = \min\{9, 6, 7, 9\} = 6$ ← tiefster Wert
Wo Wert einfügen: $\Delta(3, 1, 5) = 9 + 9 - 5 = 13$ oder $\Delta(3, 5, 4) = 9 + 7 - 5 = 11$
oder $\Delta(3, 4, 2) = 7 + 6 - 4 = 9$ oder $\Delta(3, 2, 1) = 6 + 9 - 1 = 14$
→ Neue und nun letzte Teiltour $T = (1, 5, 4, 3, 2, 1)$, Länge 24

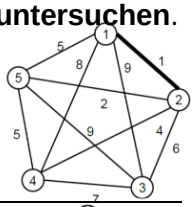


B2) «Farthest Insertion»

- Knoten $j \in V - S$ einfügen, der max. Distanz zur aktuellen Teiltour hat: $dist(j) = \max\{dist(v): v \in V - S\}$
- **Wahl Einfügestelle für Nearest und Farthest Insertion:** Es wird diejenige Einfügestelle ausgewählt, die zur kleinsten Verlängerung der Teiltour führt. Wenn wir den Knoten j zwischen Knoten p und Knoten q einfügen, führt das zu Verlängerung der Teiltour um die Grösse $\Delta(j,p,q) = d_{pj} + d_{jq} - d_{pq}$. Diese Stelle wird durch Knoten p^*, q^* definiert: $\Delta(j,p^*,q^*) = \min\{\Delta(j,p,q): p = i(r), q = i(r+1)\ für\ r\ mit\ 1 \leq r \leq k\}$
- Hier bezeichnet i Reihenfolge der besuchten Knoten, d.h. p ist r-te besuchte Knoten und q ist der $(r+1)$-te besuchte Knoten für geeignetes r, was bedeutet, dass **nur benachbarte Knoten** p und q **untersuchen**.

Beispiel

Start mit vorzugebender Teiltour $T = (1,2,1)$, i.d.R. kürzester Weg in Graph suchen

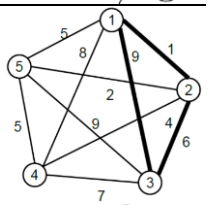


Iteration 1: $S = \{1,2\}, V - S = \{3,4,5\}$ alle möglichen nächsten Knoten

$dist(3) = \min\{9,6\} = 6$, $dist(4) = \min\{8,4\} = 4$,

$dist(5) = \min\{5,2\} = 2$ ↖ **höchster** Wert von Minimums nehmen!!!

Wo soll Wert eingefügt werden: $\Delta(3,1,2) = 9 + 6 - 1 = 14$ oder $\Delta(3,2,1) = 9 + 6 - 1 = 14$



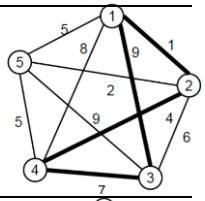
→ Neue Teiltour $T = (1,3,2,1)$, Länge 16

Iteration 2: $S = \{1,2,3\}, V - S = \{4,5\}$ alle möglichen nächsten Knoten

$dist(4) = \min\{8,7,4\} = 4$, $dist(5) = \min\{5,9,2\} = 2$ **höchster** Wert von Minimums nehmen!!!

Wo Wert einfügen: $\Delta(4,1,3) = 8 + 7 - 9 = 6$ oder $\Delta(4,3,2) = 7 + 4 - 6 = 5$

oder $\Delta(4,2,1) = 4 + 8 - 1 = 11$



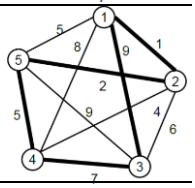
→ Neue Teiltour $T = (1,3,4,2,1)$, Länge 21

Iteration 3: $S = \{1,2,3,4\}, V - S = \{5\}$ alle möglichen nächsten Knoten

$dist(5) = \min\{5,5,2,9\} = 2$ ← **höchster** Wert von Minimums nehmen!!!

Wo Wert einfügen: $\Delta(5,1,3) = 5 + 9 - 9 = 5$ oder $\Delta(5,3,4) = 9 + 5 - 7 = 7$

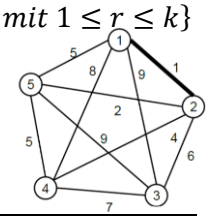
oder $\Delta(5,4,2) = 5 + 2 - 4 = 3$ oder $\Delta(5,2,1) = 2 + 5 - 1 = 6$



→ Neue und nun letzte Teiltour $T = (1,3,4,5,2,1)$, Länge 24

B3) «Cheapest Insertion»

- Cheapest Insertion bestimmt den **geeigneten Knoten** und die **geeignete Stelle** zum Einfügen **gleichzeitig** (im Gegensatz zu Nearest und Farthest Insertion, wo diese Suche in zwei Schritten abläuft).
- Zu jedem potenziellen Einfügeknoten $v \in V - S$ wird optimale Einfügestelle in Teiltour bestimmt (Minimierung Verlängerung der Teiltour): $\Delta(v,p^*,q^*) = min\{\Delta(v,p,q): p = i(r), q = i(r+1)\ für\ ein\ r\ mit\ 1 \leq r \leq k\}$
- Länge der Teiltour ändert sich um folgende Grösse: $d_v := \Delta(v,p^*,q^*) = d_{p^*,v} + d_{v,q^*} - d_{p^*,q^*}$
- Knoten mit Einfügestelle mit kleinsten Verlängerung wird gewählt: $d_j = min\{d_v: v \in V - S\}$



Beispiel

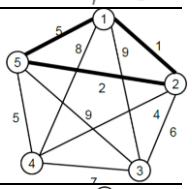
Start mit vorzugebender Teiltour $T = (1,2,1)$, i.d.R. kürzester Weg in Graph suchen

---

Iteration 1: $S = \{1,2\}, V - S = \{3,4,5\}$ alle möglichen nächsten Knoten

$\Delta(3,1,2) = \Delta(3,2,1) = 9 + 6 - 1 = 14$, $\Delta(4,1,2) = \Delta(4,2,1) = 8 + 4 - 1 = 11$

$\Delta(5,1,2) = \Delta(5,2,1) = 5 + 2 - 1 = 6$ ← tiefster Wert



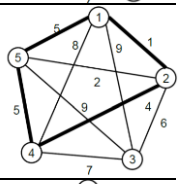
→ Neue Teiltour $T = (1,5,2,1)$, Länge 8

Iteration 2: $S = \{1,2,5\}, V - S = \{3,4\}$ alle möglichen nächsten Knoten

$\Delta(3,1,5) = 9 + 9 - 5 = 13$, $\Delta(3,5,2) = 9 + 6 - 2 = 13$, $\Delta(3,2,1) = 6 + 9 - 1 = 14$

$\Delta(4,1,5) = 8 + 5 - 5 = 8$, $\Delta(4,5,2) = 5 + 4 - 2 = 7$, $\Delta(4,2,1) = 4 + 8 - 1 = 11$

tiefster Wert wiederum nehmen

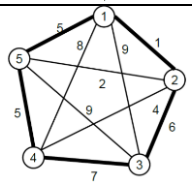


→ Neue Teiltour $T = (1,5,4,2,1)$, Länge 15

Iteration 3: $S = \{1,2,4,5\}, V - S = \{3\}$ alle möglichen nächsten Knoten

$\Delta(3,1,5) = 9 + 9 - 5 = 13$, $\Delta(3,5,4) = 9 + 7 - 5 = 11$,

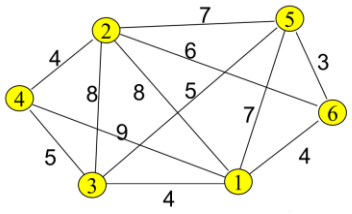
$\Delta(3,4,2) = 7 + 6 - 4 = 9$, $\Delta(3,2,1) = 6 + 9 - 1 = 14$

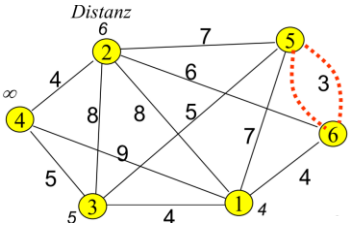


→ Neue und nun letzte Teiltour $T = (1,5,4,3,2,1)$, Länge 24

Vorsicht bei nicht vollständigen Graphen

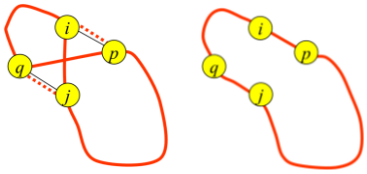
- Bei **nicht vollständigen** Graphen ist **Vorsicht** bei der **Anwendung** der **Heuristiken** geboten. Es könnte nämlich passieren, dass man den Knoten mit kleinster Distanz zur Teiltour gar nicht einfügen könnte, weil er nur **eine** Verbindung zur Teiltour hat.
- **Beispiel:** Man beginnt mit Teiltour $(5, 6, 5)$ mit Länge 6. In diesem Beispiel wäre das der Fall vom Knoten 3, wenn das Gewicht der Kante (3,5) kleiner als 4 wäre, da es nur eine Verbindung zu Punkt 3 gibt von der Teiltour $(5, 6, 5)$.



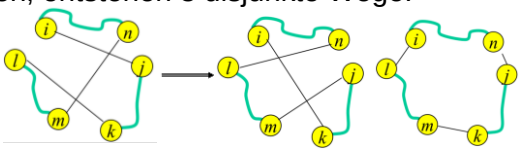


Verbesserungs-Heuristiken – «k-opt»-Heuristiken

- Neben konstruktiven Heuristiken gibt es auch Verbesserungs-Heuristiken. Diese haben zum Ziel, eine **bereits vorhandene Tour weiter** zu **verbessern** und auf diesem Weg möglichst nahe an optimale Tour heranzukommen. Wenn keine Verbesserung mehr möglich ist, stoppt das Verfahren und die gefundene Lösung ist dann «lokal optimal» (weil sie durch eine «lokale Suche» nicht weiter verbessert werden kann).
- Bekanntestes «**Local Search**» (Meta-Heuristik) Verfahren für TSP sind die sogenannten k-opt Heuristiken.
- Sie basieren auf dem Austausch von k **Kanten** in **einer Tour** durch k Kanten **ausserhalb** der **Tour**.
- Der Rechenaufwand steigt sehr schnell, wenn k vergrössert wird (und alle Austausche versucht werden).
- Meistens wird $k \leq 3$ gewählt.
- Erklärung Verfahren an einem 2-opt-Beispiel:
  - In einer gegebenen Tour suchen wir Kanten (i,j) und (p,q) (alle Knoten voneinander verschieden), für welche der Austausch noch nicht versucht wurde. Versuchsweise ersetzt man die Kanten (i,j) und (p,q) durch die Kanten (i,p) und (j,q). Dadurch vergrössert sich die Tourenlänge um $\Delta = d_{ip} + d_{jq} - d_{ij} - d_{pq}$
  - Falls diese Verlängerung negativ ist $\Delta < 0$ (die Tour wird also verkürzt), akzeptiert man den Austausch, sonst bleibt die Tour ohne Veränderung. Nun versucht man einen weiteren Austausch durchzuführen.



- Erklärung Verfahren an einem 3-opt-Beispiel:
  - nun $k = 3$ Kanten in der Tour vertauscht mit 3 Kanten ausserhalb der Tour.
  - Wenn drei Kanten $(i,j),(k,l),(m,n)$ aus der Tour entfernt werden, entstehen 3 disjunkte Wege.
  - Es gibt verschiedene Möglichkeiten, diese 3 Wege durch drei andere Kanten zu einer neuen Tour zusammenzufügen (für $k = 2$ nur eine!) Beispiel (2 Möglichkeiten sind aufgeführt):



Eine ILP-Formulierung für das symmetrische TSP

- **Problem:** Bestimme minimale Tour in **un**gerichtetem Graphen $G = (V, E)$
- **Zielfunktion:** $\min \sum_{(v,w)\in E}(c_{vw}x_{vw})$, wobei c_{vw} das Gewicht der Kante und x_{vw} die binäre Variable für die Reise von v nach w ist. Es gilt jeweils $x_{vw} = x_{wv}$, da symmetrisch
- **Degree Constraints:** $\sum_{w\in V:(v,w)\in E}(x_{vw} = 2)$, $v \in V$ was bedeutet, dass **ein einziges Mal** an diesem Knoten angekommen wird und dieser Knoten dann auch wieder **ein einziges Mal** verlassen wird. D.h. hier werden alle Möglichkeiten von einem anderen Knoten in diesen hineinzugehen (und damit auch hinaus) summiert. Die Anzahl dieser Nebenbedingungen ist gleich der Anzahl Knoten. Für die erste NB, sucht man alle x_{vw} wo $v = 1$ oder $w = 1$ ist und addiert diese zusammen. Siehe Beispiel unten
  Subtour Constraints: $\sum_{v,w\in S:(v,w)\in E}(x_{vw} \leq |S| - 1)$, $\emptyset \subset S \subset V$, was bedeutet zwischen leeren und voller Menge, diese Nebenbedingung verhindert, dass es eine Abspaltung in Subtouren (bspw. zwei Graphen ohne Verbindung zueinander) gibt. Diese Nebenbedingungen wachsen mit zunehmenden V exponentiell an! Diese NB braucht es erst ab $\left\lfloor \frac{|V|}{2} \right\rfloor \geq 3$, somit erst wenn es 6 und mehr Knoten gibt.
- $0 \leq x_{vw} \leq 1, (v,w) \in E$ und x_{vw} integer → Binary/Integer Conditions, Variable ist binär $(0,1)$

Beispielaufgabe: Betrachtet wird ein symmetrisches TSP, welches durch folgende Gewichtsmatrix definiert ist: Geben Sie eine ILP-Formulierung für dieses TSP. Schreiben Sie dabei alle Restriktionen explizit (d.h. ohne Verwendung des Summenzeichens) auf.

d_{ij}	1	2	3	4	5	6
1	–	72	47	7	14	58
2		–	17	59	26	69
3			–	51	66	45
4				–	37	46
5					–	39
6						–

Zielfunktion: $\min(72x_{12} + 47x_{13} + 7x_{14} + 14x_{15} + 58x_{16} + 17x_{23} + 59x_{24} + 26x_{25} + 69x_{26} + 51x_{34} + 66x_{35} + 45x_{36} + 37x_{45} + 46x_{46} + 39x_{56})$

Degree Constraints:

$x_{12} + x_{13} + x_{14} + x_{15} + x_{16} = 2$ (1), $x_{13} + x_{23} + x_{34} + x_{35} + x_{36} = 2$ (3), $x_{15} + x_{25} + x_{35} + x_{45} + x_{56} = 2$ (5),
$x_{12} + x_{23} + x_{24} + x_{25} + x_{26} = 2$ (2), $x_{14} + x_{24} + x_{34} + x_{45} + x_{46} = 2$ (4), $x_{16} + x_{26} + x_{36} + x_{46} + x_{56} = 2$ (6)

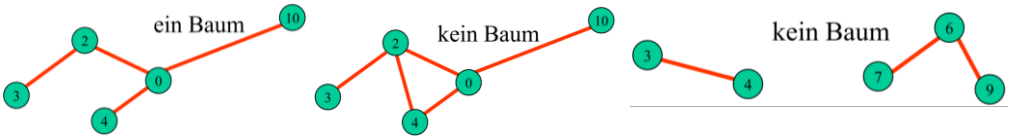
Subtour Constraints:

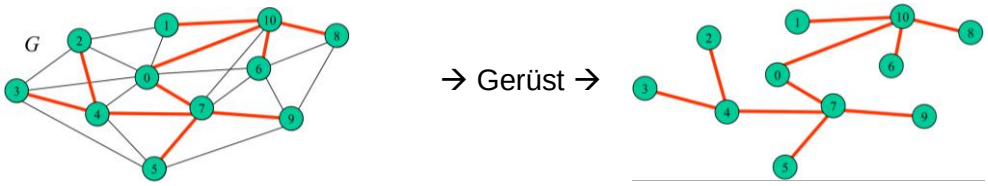
$x_{12} + x_{23} + x_{13} \leq 2$ $x_{13} + x_{35} + x_{15} \leq 2$ $x_{23} + x_{34} + x_{24} \leq 2$ $x_{25} + x_{56} + x_{26} \leq 2$
$x_{12} + x_{24} + x_{14} \leq 2$ $x_{13} + x_{36} + x_{16} \leq 2$ $x_{23} + x_{35} + x_{25} \leq 2$ $x_{34} + x_{45} + x_{35} \leq 2$
$x_{12} + x_{25} + x_{15} \leq 2$ $x_{14} + x_{45} + x_{15} \leq 2$ $x_{23} + x_{36} + x_{26} \leq 2$ $x_{34} + x_{46} + x_{36} \leq 2$
$x_{12} + x_{26} + x_{16} \leq 2$ $x_{14} + x_{46} + x_{16} \leq 2$ $x_{24} + x_{45} + x_{25} \leq 2$ $x_{35} + x_{56} + x_{36} \leq 2$
$x_{13} + x_{34} + x_{14} \leq 2$ $x_{15} + x_{56} + x_{16} \leq 2$ $x_{24} + x_{46} + x_{26} \leq 2$ $x_{45} + x_{56} + x_{46} \leq 2$
$0 \leq x_{12} \leq 1, \ldots, 0 \leq x_{56} \leq 1$ und integer

7 Optimale Bäume

Bäume in Graphen

- Sei $G = (V, E)$ ein ungerichteter Graph. Ein **Baum** in G ist ein Subgraph von G, der **zusammenhängend** ist **und keine Zyklen** enthält.
- Ein **Gerüst** (spanning tree) in G ist ein Baum in G, welcher **alle Knoten** von G umfasst.

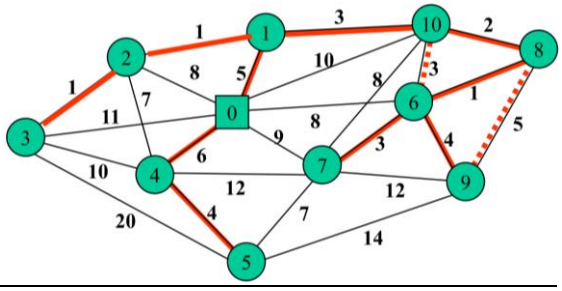




- Ein **Wald** (forest) ist **azyklischer Graph**, d.h. ein Graph, dessen **Komponenten Bäume** sind.
- Sei $G = (V, E)$ ein gewichteter Graph mit Kantengewichten(-längen) $d_e, e \in E$. Das **Gewicht** (Länge) eines **Baumes** in G ist definiert als die **Summe** der **Gewichte** seiner **Kanten**.
- **Problem des minimalen Gerüsts:** Bestimme ein Gerüst minimalen Gewichts in G, d.h. bestimme ein Gerüst von G, dessen Länge minimal ist unter allen möglichen Gerüsten von G.

Minimales Gerüst und Kruskal-Algorithmus

- Das Problem des minimalen Gerüsts ist eines der **einfachsten Probleme** der kombinatorischen Optimierung! Es kann mit dem Kruskal-Algorithmus (Typ Greedy) auf einfache Weise exakt gelöst werden.
- **Algorithmus von Kruskal:**
  - Sei n die Anzahl Knoten und m die Anzahl Kanten des Graphen. Ordne die Kanten nach aufsteigenden Gewichten: $e_1, e_2, e_3, \dots e_m$ d.h. $d_{e_1} \leq d_{e_2} \leq d_{e_3} \leq \cdots \leq d_{e_m}$
  - **Initialisierung:** Menge der gewählten Kanten $J := \emptyset$; $i := 0$;
  - **Iterationen:** WHILE $|J| < n - 1$ DO
    $i := i + 1$
    Falls $\{e_i\} \cup J$ keinen Zyklus enthält, wähle e_i, d.h. $J := \{e_i\} \cup J$
- **Lösung:** Ein minimales Gerüst ist:
- **Minimale Kosten:** $1 + 1 + 3 + 2 + 1 + 3 + 4 + 5 + 6 + 4 = 30$



Formulierungen als ILP und LP

- Die Probleme der Kombinatorischen Optimierung mit linearer Zielfunktion können als **ganzzahlig-lineare Programme** (ILP: Integer Linear Program) formuliert werden.
- Bei gewissen Problemformulierungen kann die Ganzzahligkeit-Forderung weggelassen werden, und das Problem kann als Lineares Programm (LP) gelöst werden. (Eigenschaft der «**natürlichen Ganzzahligkeit**»: Alle Eckpunkte des den Lösungsraum beschreibenden Polyeders sind ganzzahlig)
- Auch wenn ein kombinatorisches Problem leicht lösbar ist, kann es sein, dass eine Formulierung als ILP nicht evident (offenkundig) ist, und dass es «**bessere**» und «**schlechtere**» **ILP-Formulierungen** gibt.
- Zur Illustration im Folgenden zwei verschiedene ILP-Formulierungen für das Problem des minimalen Gerüsts (wobei die zweite die Eigenschaft der natürlichen Ganzzahligkeit hat).

A) Eine korrekte (aber «schlechte») ILP-Formulierung

- Für jede Kante (i, j) des Graphen binäre $(0, 1)$ Variable x_{ij}: $\begin{cases} 1 & wenn\ die\ Kante\ (i,j)\ gewählt\ wird \\ 0 & sonst \end{cases}$
- Minimiere $\sum_{alle\ Kanten\ (i,j)} d_{ij} x_{ij}$ unter den Bedingungen:
  - $\sum_{alle\ Kanten\ (i,j)\ mit\ i \in S, j \notin S} x_{ij} \geq 1$ für jede nicht-leere Knotenmenge S, welche den Knoten 0 (d.h. den Verzweigungsknoten nicht enthält)
  - $x_{ij} = 0\ oder\ 1$ für jede Kante $(i, j) \in E$
- Bedeutung der Restriktionen: Garantieren, dass Auswahl der Kanten einen zusammenhängenden Subgraphen bildet: Es muss ein Weg existieren von jedem Knoten von S zum Knoten 0 ⇒ Mindestens eine ausgewählte Kante (i, j) (mit $x_{ij} = 1$) muss aus der Menge S hinausführen.
- Wie viele solche Restriktionen? Die **Anzahl Knotenteilmengen** S **wächst exponentiell** mit der Anzahl Knoten. Folglich gibt es eine expone**ntielle Anzahl Restriktionen**!

B) Eine «gute» ILP- (bzw. LP-) Formulierung

- Minimiere $\sum_{alle\ Kanten\ (i,j)} d_{ij}x_{ij}$ unter den Bedingungen:
  - $\sum_{alle\ Kanten\ (i,j)\ mit\ i \in S_p, q \neq pS} x_{ij} \geq k-1$ für jede Partition* von V in Mengen $S_1,\ldots,S_k, 2 \leq k \leq n$
  - $x_{ij} = 0\ oder\ 1$ für alle Kante $(i,j) \in E$ (**überflüssig**) → $x_{ij} \geq 0$ für alle Kante $(i,j) \in E$ (**genügt**)
    *) Eine Partition von V in Teilmengen $S_1,\ldots,S_k \subset V$ ist gegeben, falls $V = S_1 \cup \ldots \cup S_k$ ist und die Mengen $S_1,\ldots,S_k$ paarweise disjunkt sind, d.h. $S_p \cap S_q = \emptyset, 1 \leq p < q \leq k$.
- Bedeutung der Restriktionen: Garantieren den Zusammenhang zwischen den Teilmengen jeder Partition: Mindestens $k-1$ Kanten müssen ausgewählt werden (mit $x_{ij} = 1$) «zwischen» den Mengen $S_1,..,S_k$

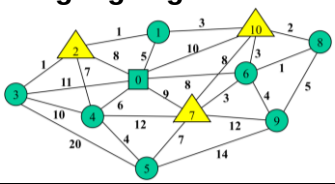
Steiner-Bäume

- Es geht darum, einen Baum zu bestimmen, welcher alle Knoten N enthält.
- Zusätzlich kann (muss aber nicht) der Baum auch Zwischenknoten aus der Menge $V-N$ benützen.
- Ein solcher Baum bildet einen Steiner-Baum im Graphen $G=(V,E)$ mit «obligatorischen» Knoten N und «fakultativen» Knoten $V-N$ (Steiner-Knoten genannt).
- Das Problem des minimalen Steiner-Baums besteht darin, unter allen möglichen Steiner-Bäumen von G einen mit minimalem Gewicht zu finden.

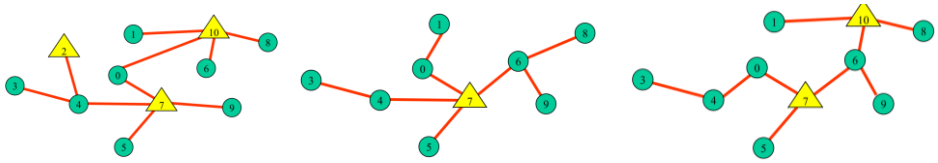
Beispiel

Grün = Knoten, gelb = Zwischenknoten,

Ausgangslage:



Einige Möglichkeiten



Ein einfacher Lösungsansatz (exakt, «naiv»)

- **Alle** möglichen **Kombinationen** (Auswahlen) von Steiner-Knoten **prüfen**.
- Für jede Auswahl von Steiner-Knoten wird das entsprechende Problem des minimalen Gerüsts gelöst (die Knotenmenge ist festgelegt).
- Unter diesen minimalen Gerüsten, wähle dasjenige mit **kleinstem Gewicht**.
- Anzahl Kombinationen von Steiner-Knoten **wächst exponentiell** mit Anzahl Knoten $|V-N|$: $2^{|V-N|}$
- Folglich ist Ansatz nur möglich, wenn Anzahl Steiner-Knoten klein ist!
- Illustration im Beispiel mit Steiner-Knoten $V-N=\{2,7,10\}$. Mögliche Kombinationen von Steiner-Knoten:
  - 1) ∅
  - 2) {2}
  - 3) {7}
  - 4) {10}
  - 5) {2,7}
  - 6) {2,10}
  - 7) {7,10}
  - 8) {2,7,10}

Eine Transformation des Problems

- Falls die Anzahl obligatorischer Knoten $|N|$ «klein» ist, kann der obige Ansatz mittels einer Transformation verbessert werden.
- **Lemma:** Sei $H=(V,U)$ ein vollständiger Graph, dessen Kantengewichte die Dreiecksungleichung erfüllen, d.h. $h_{ij}+h_{jk} \geq h_{ik}$. Dann existiert ein minimaler Steiner-Baum in H, welcher höchstens $|N|-2$ Steiner-Knoten enthält.
- **Anwendung:** Zur Bestimmung eines minimalen Steiner-Baums in einem beliebigen Graphen $G=(V,E)$ mit Gewichten d_{ij} und obligatorischen Knoten N, wobei $|N|$ «klein ist»: Bestimme einen minimalen Steiner-Baum im vollständigen Graphen $H=(V,U)$ mit Kantengewichten h_{ij}, h_{ij} die Länge eines kürzesten Weges von i nach j in G ist.

Stand der Arbeit

SW	Vorlesung / Kapitel	Übungen (Abgaben)	Gelöste Aufgaben	ZF
1	☒ LP's	☒	☒	☒
2	☒ Simplex-Methode	☒	☒	☒
3	☒ Dualität von LP's	☒	☒	☒
4	☒ Ganzzahlige LP's	☒	☒	☒
5	☒ Optimale Wege	☒	☒	☒
6	☒ Optimale Zyklen	☒	☒	☒
7	☒ Optimale Bäume	☒	☒	☒
8	☐	☒	☒	☐
9	☐	☒	☒	☐
10	☐	☒	☒	☐
11	☐	☒	☒	☐
12	☐	☐	☐	☐
13	☐	☐	☐	☐
14	☐	☐	☐	☐

Modulauflagen

Obligatorische Leistungsnachweise für die Modulbewertung:

- Obligatorisch: mündliche Semesterendprüfung, Dauer 30 Min, 15 Minuten Vorbereitung: 5/6 der Endnote
- Obligatorisch: Case Study, 2. Teil (Fortsetzung aus Factory Physics): 1/6 der Endnote
- Freiwillig: Spätestens vor Beginn der Vorlesung notieren alle Studierende in Moodle, welche Aufgaben sie gelöst haben. Die gelösten Aufgaben müssen auf Moodle hochgeladen werden. Der Dozierende wählt Studierende aus, die ihre Lösungen im Klassenzimmer vorführen. Studierende, die mindestens 75% der Aufgaben erfolgreich gelöst haben und ausgewählte Aufgaben präsentiert haben, erhalten einen Bonus in der Endnote von 0.3.