

# Lineare und kombinatorische Optimierung Algorithmen

Dieter Weninger  
FAU Erlangen-Nürnberg, Lehrstuhl für Wirtschaftsmathematik  
Wintersemester 2017/2018

# Breitensuche

**Eingabe:** Graph  $G = (V, E)$

**Ausgabe:** Anzahl der Zusammenhangskomponenten und die Menge der Knoten jeder Zusammenhangskomponente.

```

1: Setze  $\text{mark}(v) = -1$  für alle  $v \in V$ ,  $\text{next} = 0$  und  $L = \emptyset$ .
2: for  $v \in V$  do
3:     if  $\text{mark}(v) < 0$  then
4:          $\text{next} \leftarrow \text{next} + 1$ 
5:         Füge  $v$  ans Ende der Liste  $L$  an.
6:         while  $L \neq \emptyset$  do
7:             Wähle  $v$  vom Anfang der Liste und entferne  $v$  aus  $L$ .
8:             if  $\text{mark}(v) < 0$  then
9:                  $\text{mark}(v) \leftarrow \text{next}$ 
10:            for alle zu  $v$  adjazenten  $w$  do
11:                if  $\text{mark}(w) < 0$  then
12:                     $\text{mark}(w) \leftarrow \text{next}$ 
13:                    Füge  $w$  am Ende der Liste an.
14: return  $\text{next}$ ,  $\text{mark}(\cdot)$ 

```

# Tiefensuche

**Eingabe:** Graph  $G = (V, E)$

**Ausgabe:** Anzahl der Zusammenhangskomponenten und die Menge der Knoten jeder Zusammenhangskomponente  
Setze  $\text{mark}(v) = -1$  für alle  $v \in V$  und  $\text{next} = 0$ .

```

1: for  $v \in V$  do
2:   if  $\text{mark}(v) < 0$  then
3:      $\text{next} \leftarrow \text{next} + 1$ 
4:      $\text{mark}(v) \leftarrow \text{next}$ 
5:     for  $w$  adjacent zu  $v$  do
6:        $\text{checkNeighbor}(w, \text{next})$ 
7: return  $\text{next}, \text{mark}(\cdot)$ 

```

$\text{checkNeighbor}(w, \text{next})$ :

**Eingabe:**  $G = (V, E), w \in V, \text{next}, \text{mark}(\cdot)$

```

1: if  $\text{mark}(w) < 0$  then
2:    $\text{mark}(w) = \text{next}$ 
3:   for alle zu  $w$  adjazenten  $u$  do
4:      $\text{checkNeighbor}(u, \text{next})$ 

```

# Selectionsort

**Eingabe:** Ein Array  $a$  der Länge  $n$  mit  $a[i] \in \mathbb{Z}$ .

**Ausgabe:** Das sortierte Array  $a$  mit  $a[1] \leq a[2] \leq \dots \leq a[n]$ .

```

1: for  $i = 1 : n$  do
2:    $\text{min} \leftarrow i$ 
3:   for  $j = i + 1 : n$  do
4:     if  $a[j] < a[\text{min}]$  then
5:        $\text{min} \leftarrow j$ 
6:   Tausche  $a[\text{min}]$  und  $a[i]$ .
7: return  $a$ 

```

# Bubblesort

**Eingabe:** Ein Array  $a$  der Länge  $n$  mit  $a[i] \in \mathbb{Z}$ .

**Ausgabe:** Das sortierte Array  $a$  mit  $a[1] \leq a[2] \leq \dots \leq a[n]$ .

```

1: for  $i = 1 : n - 1$  do
2:   for  $j = 1 : n - i$  do
3:     if  $a[j] > a[j + 1]$  then
4:       Tausche  $a[j]$  und  $a[j + 1]$ .
5: return  $a$ 

```

# Quicksort

**Eingabe:** Ein Array  $a$  der Länge  $n$  mit  $a[i] \in \mathbb{Z}$ , eine untere und obere Grenze  $l, r$  mit  $1 \leq l \leq r \leq n$ .

**Ausgabe:** Das sortierte Array  $a$  mit  $a[l] \leq a[l+1] \leq \dots \leq a[r]$ .

```

1: Setze  $i \leftarrow l$  und  $j \leftarrow r$ .
2: while  $i < j$  do
3:   while  $a[i] \leq a[r]$  und  $i < j$  do
4:     Setze  $i \leftarrow i + 1$ 
5:   while  $a[j] \geq a[r]$  und  $i < j$  do
6:     Setze  $j \leftarrow j - 1$ 
7:   Tausche  $a[i]$  und  $a[j]$ .
8: Tausche  $a[i]$  und  $a[r]$ .
9: if  $l < i - 1$  then
10:   Quicksort( $a, l, i - 1$ )
11: if  $i + 1 < r$  then
12:   Quicksort( $a, i + 1, r$ )
13: return  $a$ 
  
```

# Heapsort

**Eingabe:** Ein Array  $a$  der Länge  $n$  mit  $a[i] \in \mathbb{Z}$ .

**Ausgabe:** Das sortierte Array  $a$  mit  $a[1] \leq a[2] \leq \dots \leq a[n]$ .

```

1: for  $i = \lfloor n/2 \rfloor : 1$  do
2:   HEAPIFY( $a, i, n$ )
3: for  $i = n : 2$  do
4:   Tausche  $a[1]$  und  $a[i]$ .
5:   HEAPIFY( $a, 1, i - 1$ )
6: return  $a$ 

1: procedure HEAPIFY( $a, l, r$ )
2:   Setze  $f \leftarrow l$  und  $s \leftarrow 2f$ .
3:   while  $s \leq r$  do
4:     if  $s + 1 \leq r$  und  $a[s] \leq a[s + 1]$  then
5:       Setze  $s \leftarrow s + 1$ .
6:     if  $a[f] \leq a[s]$  then
7:       Tausche  $a[f]$  und  $a[s]$ .
8:       Setze  $f \leftarrow s$  und  $s \leftarrow 2f$ .
9:     else
10:      break
  
```

# Bucketsort

**Eingabe:** Ein Array  $a$  der Länge  $n$  mit  $a[i] \in \mathbb{Z}$ .

**Ausgabe:** Das sortierte Array  $a$  mit  $a[1] \leq a[2] \leq \dots \leq a[n]$ .

1: Bestimme die Anzahl der Buckets  $B$  und eine Funktion

$$f : \{a[1], \dots, a[n]\} \rightarrow \{1, \dots, B\}.$$

2: **for**  $i = 1 : n$  **do**

3:     Weise  $a[i]$  Bucket  $f(a[i])$  zu.

4: **for**  $b = 1 : B$  **do**

5:     Sortiere Bucket  $b$ .

6: Kopiere Elemente aus dem Bucket in  $a$  zurück.

7: **return**  $a$



# Algorithmus von Kruskal

**Eingabe:** Graph  $G = (V, E)$ , Gewichte  $c_e$  für  $e \in E$

**Ausgabe:** Maximaler Wald (bzgl. Kantenanzahl)  $W \subseteq E$  minimalen Gewichts.

1: Sortiere die Kantengewichte in nicht-absteigender Reihenfolge

$$c_{e_1} \leq c_{e_2} \leq \dots \leq c_{e_m}.$$

2: Setze  $W \leftarrow \emptyset$ .

3: **for**  $i = 1 : m$  **do**

4:     **if**  $W \cup \{e_i\}$  enthält keinen Kreis **then**

5:         Setze  $W \leftarrow W \cup \{e_i\}$ .

6: **return**  $W$

# Bestimmung minimaler Spannbäume

**Eingabe:** Ein zusammenhängender Graph  $G = (V, E)$  mit Gewichten  $c_e$  für alle Kanten  $e \in E$ .

**Ausgabe:** Aufspannender Baum  $T \subseteq E$  minimalen Gewichts.

- 1: Für alle  $i \in V$  setze  $V_i \leftarrow \{i\}$  und  $T_i \leftarrow \emptyset$ .
- 2: **for**  $|V| - 1$  mal **do**
- 3:     Wähle nichtleeres  $V_i$ .
- 4:     Wähle  $uv \in \delta(V_i)$  mit  $u \in V_i$  und  $c_{uv} \leq c_{pq}$  für alle  $pq \in \delta(V_i)$ .
- 5:     Bestimme  $j$ , so dass  $v \in V_j$ .
- 6:     Setze  $V_i \leftarrow V_i \cup V_j$ ,  $V_j \leftarrow \emptyset$  und  $T_i \leftarrow T_i \cup T_j \cup \{uv\}$ ,  $T_j \leftarrow \emptyset$ .
- 7: **return**  $T_i$  mit  $T_i \neq \emptyset$ .

# Algorithmus von Prim

**Eingabe:** Ein zusammenhängender Graph  $G = (V, E)$  mit Gewichten  $c_e$  für alle Kanten  $e \in E$ .

**Ausgabe:** Ein aufspannender Baum  $T \subseteq E$  minimalen Gewichts.

- 1: Wähle  $w \in V$  beliebig und setze  $T \leftarrow \emptyset$ ,  $W \leftarrow \{w\}$  und  $V' \leftarrow V \setminus \{w\}$ .
- 2: **while**  $V' \neq \emptyset$  **do**
- 3:     Wähle  $c_{uv} = \min\{c_e: e \in \delta(W)\}$  mit  $u \in W$  und  $v \in V'$ .
- 4:     Setze  $T \leftarrow T \cup \{uv\}$ ,  $W \leftarrow W \cup \{v\}$  und  $V' = V' \setminus \{v\}$ .
- 5: **return**  $T$ .

# Topologische Sortierung

**Eingabe:** Ein Digraph  $D = (V, A)$ .

**Ausgabe:** Eine topologische Sortierung  $f : V \rightarrow \{1, \dots, |V|\}$  oder die Meldung, dass  $D$  einen gerichteten Kreis enthält.

1: Setze

$$\begin{aligned} \text{indeg}(v) &= |\delta^{\text{in}}(v)| \quad \text{für alle } v \in V, \\ L &= \{v \in V : \text{indeg}(v) = 0\}, \\ \text{next} &= 0. \end{aligned}$$

2: **while**  $L \neq \emptyset$  **do**

3:     Wähle ein  $u \in L$ .

4:     Setze  $L \leftarrow L \setminus \{u\}$ .

5:     Setze  $\text{next} \leftarrow \text{next} + 1$ .

6:     Setze  $f(u) \leftarrow \text{next}$ .

7:     **for all**  $(u, v) \in \delta^{\text{out}}(u)$  **do**

8:         Setze  $\text{indeg}(v) \leftarrow \text{indeg}(v) - 1$ .

9:         **if**  $\text{indeg}(v) = 0$  **then**

10:             Setze  $L \leftarrow L \cup \{v\}$ .

11: **if**  $\text{next} < |V|$  **then**

12:     **return** “ $D$  enthält einen Kreis.”

13: **else**

14:     **return** topologische Sortierung  $f$ .

# Moore–Bellmann für azyklische Graphen

**Eingabe:** Ein azyklischer Digraph  $D = (V, A)$  mit Bogengewichten  $c_a, a \in A$  (möglicherweise auch negativ). Ein Startknoten  $s \in V$ .

**Ausgabe:** Kürzeste gerichtete Wege von  $s$  nach  $v$  für alle  $v \in V$  und deren Längen.

1: Sortiere die Knoten in  $V$  topologisch o.B.d.A.  $1, \dots, n$  mit  $i < j$  für alle  $(i, j) \in A$ .

2: Setze

$$d(v) = \begin{cases} 0, & \text{falls } v = s, \\ +\infty, & \text{sonst,} \end{cases}$$

$$\text{pred}(v) = \begin{cases} s, & \text{falls } v = s, \\ \text{ungesetzt,} & \text{sonst.} \end{cases}$$

3: **for**  $u = s : n$  **do**

4:     **for all**  $(u, v) \in \delta^{\text{out}}(u)$  **do**

5:         **if**  $d(v) > d(u) + c_{uv}$  **then**

6:             Setze  $d(v) \leftarrow d(u) + c_{uv}$ .

7:             Setze  $\text{pred}(v) \leftarrow u$ .

8: **if**  $d(v) < \infty$  **then**

9:      $d(v)$  enthält die Länge eines kürzesten Weges von  $s$  nach  $v$ .

10:     $\text{pred}(\cdot)$  die Vorgänger auf den kürzesten Wegen.

11: **else if**  $d(v) = \infty$  **then**

12:    Es existiert kein Weg von  $s$  nach  $v$ .

# Dijkstra-Algorithmus

**Eingabe:** Ein Digraph  $D = (V, A)$ ,  $c_a \geq 0$ ,  $a \in A$ , ein Startknoten  $s \in V$ .

**Ausgabe:** Kürzeste gerichtete Wege von  $s$  nach  $v$  für alle  $v \in V$  in folgender Form:

$d(v)$  = Distanz von  $s$  nach  $v$ ,  
 $\text{pred}(v)$  = Vorgänger von  $v$  auf dem Weg von  $s$  nach  $v$ .

- 1: Setze  $S \leftarrow \emptyset$ ,  $\bar{S} \leftarrow V$ ,  $d(j) \leftarrow +\infty$  für alle Knoten  $j \in V$ .
- 2: Setze  $d(s) \leftarrow 0$  und  $\text{pred}(s) \leftarrow s$ .
- 3: **while**  $|S| < |V|$  **do**
- 4:     Wähle einen Knoten  $i$  mit  $d(i) = \min\{d(j) : j \in \bar{S}\}$ .
- 5:     Setze  $S \leftarrow S \cup \{i\}$  und  $\bar{S} \leftarrow \bar{S} \setminus \{i\}$ .
- 6:     **for all**  $(i, j) \in \delta^{\text{out}}(i)$  **do**
- 7:         **if**  $d(j) > d(i) + c_{ij}$  **then**
- 8:             Setze  $d(j) \leftarrow d(i) + c_{ij}$ .
- 9:             Setze  $\text{pred}(j) \leftarrow i$ .
- 10: **return**  $d(\cdot)$  und  $\text{pred}(\cdot)$ .

# Grundversion des Moore–Bellmann-Algorithmus

**Eingabe:** Ein Digraph  $D = (V, A)$  mit Gewichten  $c_a, a \in A$ , und ohne negative Kreise, ein Startknoten  $s \in V$ .

**Ausgabe:** Kürzeste Wege von  $s$  nach  $v$ , für alle  $v \in V$  und deren Längen.

1: Setze

$$d(v) = \begin{cases} 0, & \text{falls } v = s, \\ +\infty, & \text{sonst,} \end{cases}$$

$$\text{pred}(v) = \begin{cases} s, & \text{falls } v = s, \\ \text{ungesetzt,} & \text{sonst.} \end{cases}$$

2: **while**  $\exists(i, j) \in A$  mit  $d(j) > d(i) + c_{ij}$  **do**

3:     Setze  $d(j) \leftarrow d(i) + c_{ij}$ .

4:     Setze  $\text{pred}(j) \leftarrow i$ .

5: **return**  $d(\cdot)$  und  $\text{pred}(\cdot)$ .

# Bellmann-Algorithmus

**Eingabe:** Ein Digraph  $D = (V, A)$  mit Gewichten  $c_a \in \mathbb{Z}$  (oder  $\mathbb{Q}, \mathbb{R}$ ) für  $a \in A$  und ohne negative Kreise, ein Startknoten  $s \in V$ .

**Ausgabe:** Die kürzesten  $s$ - $v$ -Wege mit Längen  $d(\cdot)$ .

1: Setze

$$d(v) = \begin{cases} 0, & \text{falls } v = s, \\ +\infty, & \text{sonst,} \end{cases}$$

$$\text{pred}(v) = \begin{cases} s, & \text{falls } v = s, \\ \text{ungesetzt,} & \text{sonst.} \end{cases}$$

2: Nummeriere die Bögen in  $A = \{a_1, \dots, a_m\}$  in beliebiger Reihenfolge.

3: **for**  $k = 1 : n$  **do**

4:     **for**  $l = 1 : m$  **do**

5:         Wähle  $a_l = (i, j)$ .

6:         **if**  $d(j) > d(i) + c_{ij}$  **then**

7:             Setze  $d(j) \leftarrow d(i) + c_{ij}$ .

8:             Setze  $\text{pred}(j) \leftarrow i$ .

9: **return**  $d(\cdot)$  und  $\text{pred}(\cdot)$ .



# Floyd–Warshall-Algorithmus

**Eingabe:** Ein Digraph  $D = (V, A)$  mit Knoten  $V = \{1, \dots, n\}$  und Gewichten  $c_a$  für alle  $a \in A$ .

**Ausgabe:** Die  $n \times n$  Kürzeste-Weglängen-Matrix  $W$  und eine  $n \times n$ -Matrix  $P$  mit folgenden Eigenschaften:

$w_{ij}$  = Länge eines kürzesten  $(i, j)$ -Weges,

$w_{ii}$  = Länge eines kürzesten  $(i, i)$ -Kreises,

$p_{ij}$  = vorletzter Knoten auf einem kürzesten  $(i, j)$ -Weg,

$p_{ii}$  = vorletzter Knoten auf einem kürzesten  $(i, i)$ -Kreis.

1: **for**  $i = 1 : n$  **do**

2:     **for**  $j = 1 : n$  **do**

3:         Setze

$$w_{ij} = \begin{cases} c_{ij}, & \text{falls } (i, j) \in A, \\ +\infty, & \text{sonst,} \end{cases} \quad \text{und} \quad p_{ij} = \begin{cases} i, & \text{falls } (i, j) \in A, \\ 0, & \text{sonst.} \end{cases}$$

4: **for**  $l = 1 : n$  **do**

5:     **for**  $i = 1 : n$  **do**

6:         **for**  $j = 1 : n$  **do**

7:             **if**  $w_{ij} > w_{il} + w_{lj}$  **then**

8:                 Setze  $w_{ij} \leftarrow w_{il} + w_{lj}$  und  $p_{ij} = p_{lj}$ .

9:             **if**  $i = j$  und  $w_{ii} < 0$  **then**

10:                 **go to** Schritt 11.

11: **return**  $W$  und  $P$ .

# Augmentierende-Wege-Algorithmus

**Eingabe:** Ein Digraph  $D = (V, A)$  mit Bogenkapazitäten  $c_a \geq 0, a \in A$ , und zwei Knoten  $s, t \in V$  mit  $s \neq t$ .

**Ausgabe:** Ein zulässiger  $(s, t)$ -Fluss  $x$  mit maximalem Wert  $\text{val}(x)$  und ein kapazitätsminimaler  $(s, t)$ -Schnitt  $\delta^{\text{out}}(W)$ .

- 1: Bestimme einen zulässigen Fluss, z. B.  $x = 0$ .
- 2: **while** Es existiert ein augmentierender Weg von  $s$  nach  $t$  **do**
- 3:     Identifiziere einen augmentierenden  $[s, t]$ -Weg  $P$ .
- 4:     Setze

$$\varepsilon = \min_{(i,j) \in P} \begin{cases} c_{ij} - x_{ij}, & \text{falls } (i,j) \in P \text{ Vorwärtsbogen,} \\ x_{ij}, & \text{falls } (i,j) \in P \text{ Rückwärtsbogen.} \end{cases}$$

- 5:     Erhöhe  $x$  entlang  $P$  um  $\varepsilon$ , d. h., setze

$$x_{ij} \leftarrow \begin{cases} x_{ij} + \varepsilon, & \text{falls } (i,j) \in P \text{ Vorwärtsbogen,} \\ x_{ij} - \varepsilon, & \text{falls } (i,j) \in P \text{ Rückwärtsbogen,} \\ x_{ij}, & \text{sonst.} \end{cases}$$

- 6: Bestimme  $W$  wie im Beweis des Optimalitätskriteriums.
- 7: **return**  $x, \text{val}(x)$  und  $W$ .

# Push-Relabel-Algorithmus (Goldberg & Tarjan, 1985)

**Eingabe:** Ein Digraph  $D = (V, A)$  mit Bogenkapazitäten  $c_a \geq 0, a \in A$ , und zwei Knoten  $s, t \in V$  mit  $s \neq t$ .

**Ausgabe:** Ein maximaler  $(s, t)$ -Fluss  $x$ .

- 1: Setze  $x$  und  $d$  gemäß Gleichung (6.4).
- 2: **while** es gibt einen aktiven Knoten **do**
- 3:     Wähle einen aktiven Knoten  $v$ .
- 4:     **if** es gibt einen zulässigen Bogen  $(v, w)$  **then**
- 5:         Push: Schiebe  $\delta = \min\{e_x(v), c_{vw} - x_{vw} + x_{wv}\}$  von  $v$  nach  $w$ .
- 6:     **else**
- 7:         Relabel: Setze  $d(v) \leftarrow 1 + \min\{d(w) : (v, w) \in A(x)\}$ .
- 8: **return**  $x$ .

# Kreis-Löschungs-Algorithmus

**Eingabe:** Ein Digraph  $D = (V, A)$  mit Bogenkapazitäten  $c_a \in \mathbb{R}_{\geq 0}$  und Kosten  $w_a$  für alle  $a \in A$ , Knotenbedarfe  $b_v$  für alle  $v \in V$ .

**Ausgabe:** Ein Minimalkostenfluss  $x$  oder die Meldung, dass es keinen zulässigen Fluss gibt.

- 1: Bestimme einen zulässigen Fluss  $x$ .
- 2: Falls es keinen zulässigen Fluss gibt: **return** "Problem ist unzulässig".
- 3: **while**  $G(x)$  enthält einen negativen Kreis **do**
- 4:     Bestimme negativen Kreis  $C$ .
- 5:     Setze  $\varepsilon = \min\{r_{ij} : ij \in C\}$ .
- 6:     Augmentiere  $\varepsilon$  Flusseinheiten entlang  $C$  und aktualisiere  $G(x)$ .
- 7: **return**  $x$

# Greedy-Max für Unabhängigkeitssysteme

**Eingabe:** Ein Problem der Form (8.2), eine Grundmenge  $E$  und eine Funktion  $c : 2^E \rightarrow \mathbb{R}_{\geq 0}$ .

**Ausgabe:** Eine zulässige Lösung  $I_G \in \mathcal{I}$ .

1: Sortiere die Elemente in  $E$  derart, dass

$$c(I_1) \geq c(I_2) \geq \dots \geq c(I_m)$$

gilt, wobei  $|E| = m$  gelte.

2: Setze  $I_G \leftarrow \emptyset$ .

3: **for**  $k = 1 : m$  **do**

4:     **if**  $I_G \cup \{I_k\} \in \mathcal{I}$  **then**

5:         Setze  $I_G \leftarrow I_G \cup \{I_k\}$ .

6: **return**  $I_G$ .

# Das Simplex-Verfahren für LPs in Standardform

**Eingabe:**  $A, b, c$  und eine primal zulässige Basis  $B = (B_1, \dots, B_m)$ .

**Ausgabe:** Eine Optimallösung  $\bar{x}$  oder die Meldung, dass das LP unbeschränkt ist.

- 1: BTRAN (Backward Transformation): Löse  $\bar{y}^\top A_B = c_B^\top$ .
- 2: Pricing: Berechne  $z_N = c_N - A_N^\top \bar{y}$ .
- 3: **if**  $z_N \geq 0$  **then**
- 4:     **return** optimale Basis  $B$  und Optimallösung  $\bar{x} = (\bar{x}_B, \bar{x}_N)^\top$ .
- 5: Wähle ein  $j \in N$  mit  $z_j < 0$ .
- 6: FTRAN (Forward Transformation): Löse  $A_B w = A_j$ .
- 7: **if**  $w \leq 0$  **then**
- 8:     **return** "Das LP ist unbeschränkt."
- 9: Ratio-Test (Quotiententest): Berechne

$$\gamma = \frac{\bar{x}_{B_i}}{w_i} = \min \left\{ \frac{\bar{x}_{B_k}}{w_k} : w_k > 0 \text{ und } k \in \{1, \dots, m\} \right\}.$$

- 10: Update: Setze

$$\bar{x}_B = \bar{x}_B - \gamma w, \quad N = N \setminus \{j\} \cup \{B_i\}, \quad B_i = j, \quad \bar{x}_j = \gamma$$

und gehe zu Schritt 1.

# Der duale Simplex-Algorithmus

**Eingabe:** Eine dual zulässige Basis  $B$  und  $z_N = c_N - A_N^T A_B^{-T} c_B \geq 0$ .

**Ausgabe:** Eine Optimallösung  $\bar{x}$  für (P) bzw. eine Optimallösung  $\bar{y}$  für das zu (P) duale Problem (aber nicht in Standardform) bzw. eine Optimallösung  $\bar{y}$ ,  $z_N, z_B = 0$  für (D) oder die Meldung das primale Problem (P) unzulässig bzw. das duale Problem (D) unbeschränkt ist.

1: BTRAN: Löse  $A_B \bar{x}_B = b$ .

2: Pricing:

3: **if**  $\bar{x}_B \geq 0$  **then**

4:     **return** dual optimale Basis  $B$ , duale Optimallösung  $\bar{y} = A_B^{-T} c_B$ .

5: Wähle ein  $i \in \{1, \dots, m\}$  mit  $x_{B_i} < 0$ , d. h.,  $B_i$  verlässt die Basis  $B$ .

6: FTRAN: Löse  $A_B^T w = e_i$  und berechne  $\alpha_N = -A_N^T w$ .

7: **if**  $\alpha_N \leq 0$  **then**

8:     **return** "(D) ist unbeschränkt und (P) ist unzulässig."

9: Ratio-Test: Berechne

$$\gamma = \frac{z_j}{\alpha_j} = \min \left\{ \frac{z_k}{\alpha_k} : \alpha_k > 0, k \in N \right\}$$

mit  $j \in N$  und  $\alpha_j > 0$ . Die Variable  $j$  kommt in die Basis.

10: Update: Setze

$$z_N = z_N - \gamma \alpha_N, \quad N = N \setminus \{j\} \cup \{B_i\}, \quad B_i = j, \quad z_{B_i} = \gamma.$$

11: Gehe zu Schritt 1.

# Branch-and-Bound für 0/1-IPs

- 1:  $\ell \leftarrow -\infty$  und  $Q \leftarrow \{(\emptyset, \emptyset)\}$ .
- 2: **while**  $Q \neq \emptyset$  **do**
- 3:     Wähle ein  $(Z, O) \in Q$  und setze  $Q \leftarrow Q \setminus \{(Z, O)\}$ .
- 4:     Löse die LP-Relaxierung mit  $Z$  und  $O$ .
- 5:     **if** LP-Relaxierung mit  $Z$  und  $O$  ist unzulässig **then**
- 6:         Gehe zu Schritt 2.
- 7:     Setze  $\bar{x}$  auf die Optimallösung der LP-Relaxierung.
- 8:     **if**  $c^\top \bar{x} \leq \ell$  **then**
- 9:         Gehe zu Schritt 2.
- 10:    **if**  $\bar{x}$  ist ganzzahlig **then**
- 11:       Setze  $x^* \leftarrow \bar{x}$ ,  $\ell \leftarrow c^\top x^*$  und gehe zu Schritt 2.
- 12:    Wähle  $i$  mit  $\bar{x}_i \notin \{0, 1\}$ .
- 13:    Setze  $Q \leftarrow Q \cup \{(Z \cup \{i\}, O), (Z, O \cup \{i\})\}$ .
- 14: **if**  $\ell > -\infty$  **then**
- 15:     **return** Optimallösung  $x^*$ .
- 16: **else**
- 17:     **return** "Das Problem ist unzulässig."