Technische Universität Braunschweig

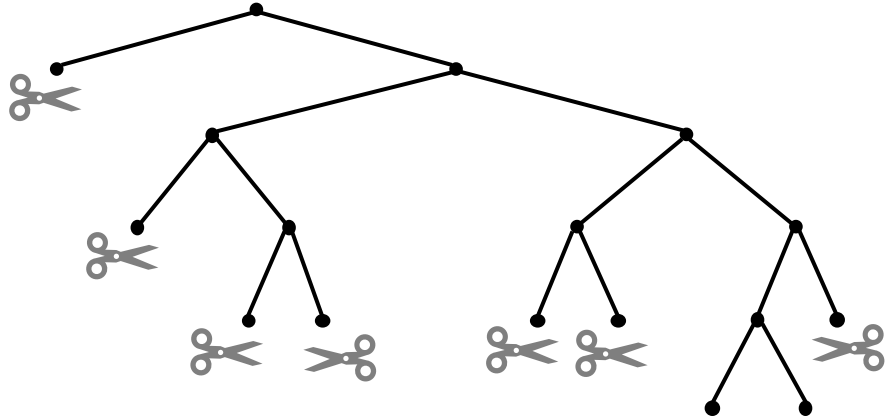
# Algorithmen und Datenstrukturen 2 – Übung #3

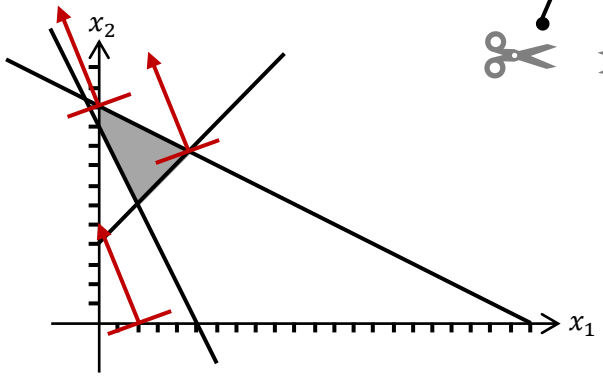
Ramin Kosfeld und Chek-Manh Loi

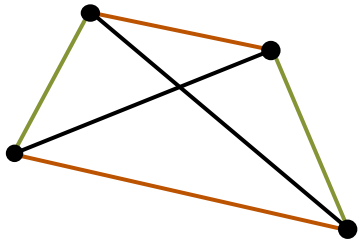
20.05.2026

# Tagesordnung

- **Branch-and-Bound für Knapsack**
- **Euklidisches TSP**





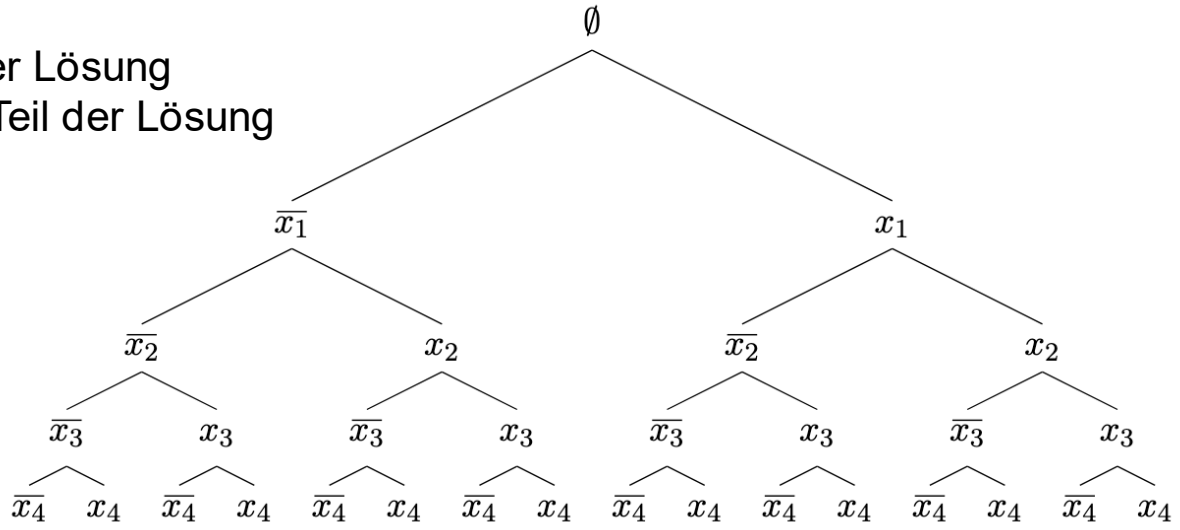
# Branch & Bound – wie war das doch gleich?

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $z_i$ | 11 | 5 | 13 | 18 |
| $p_i$ | 14 | 6 | 13 | 16 |

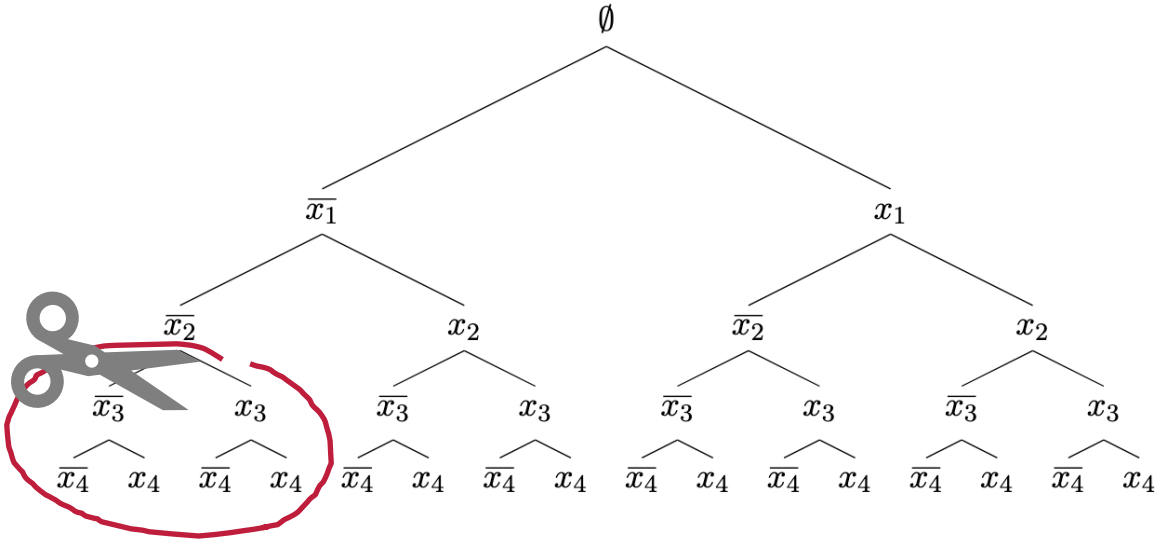
$Z = 28$

$x_i = 1$: Das $i$-te Element ist Teil der Lösung
$x_i = 0$: Das $i$-te Element ist *nicht* Teil der Lösung

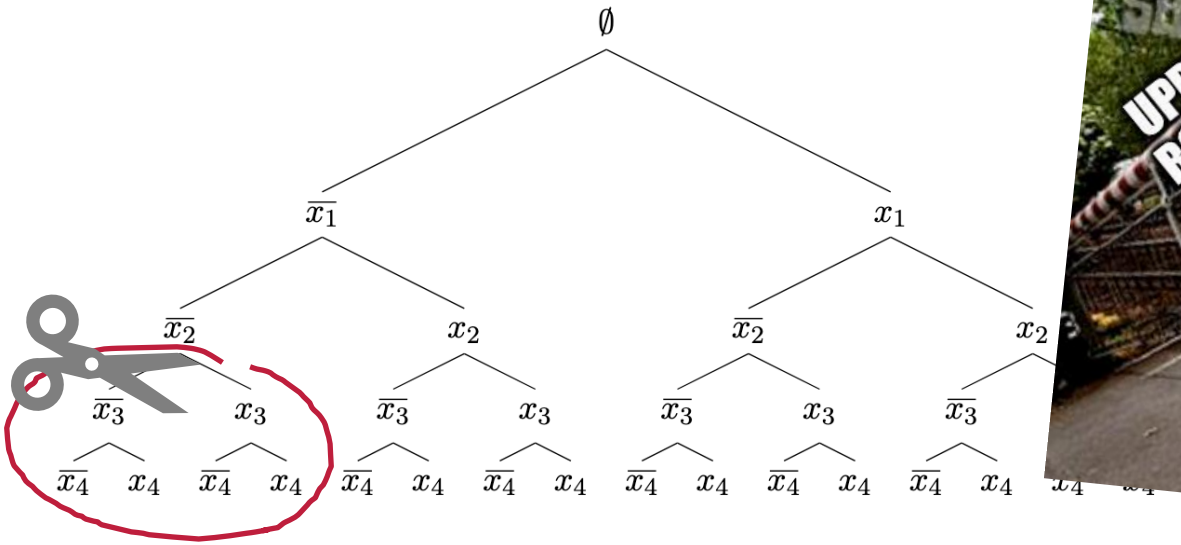
Enumerationsbaum:



# Branch & Bound – wie war das doch gleich?

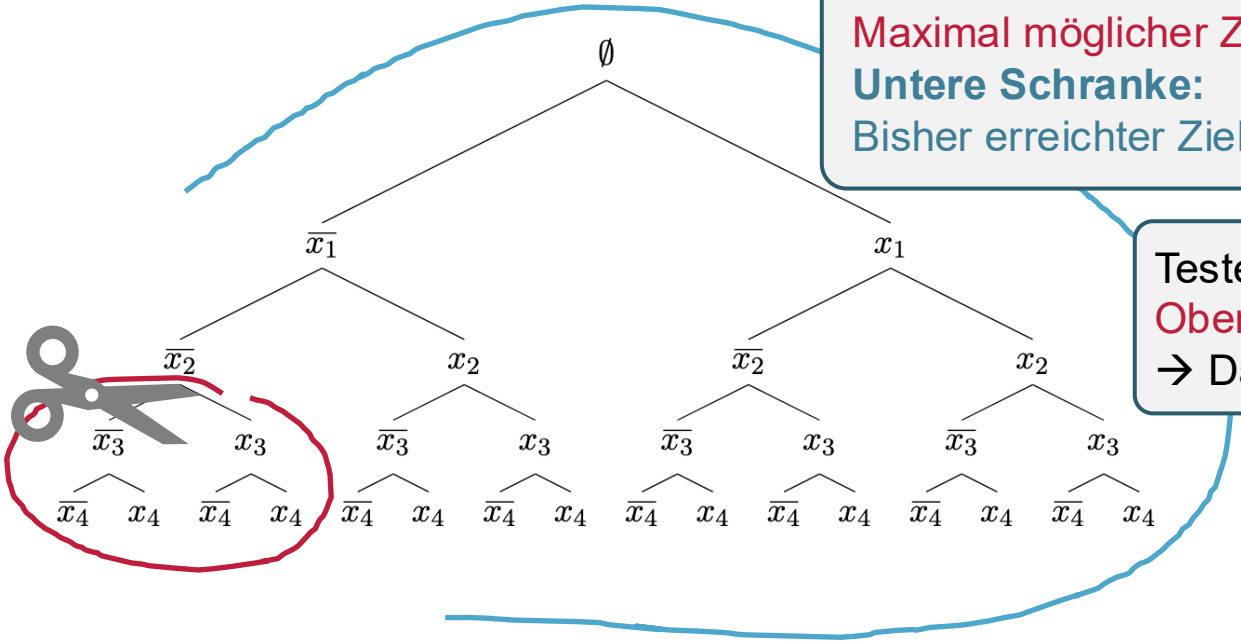


# Branch & Bound – wie war das doch gleich?



Schranken!

# Branch & Bound – wie war das doch gleich?




**Obere Schranke:**
Maximal möglicher Zielfunktionswert im *Teilbaum*
**Untere Schranke:**
Bisher erreichter Zielfunktionswert im *ganzen* Baum

Teste für jeden Teilbaum:
Obere Schranke ≤ Untere Schranke
→ Dann abschneiden!

# Branch

Allgem

1. Tes
2. Sch
3. Upd
4. Prü
5. Ver

**Algorithmus 3.1** Branch-And-Bound als Unterroutine

**Eingabe:** $z_1, ..., z_n, Z, p_1, ...p_n$ (global: Kosten/Nutzenwerte, Kostenschranke)
$P$ (global: bester bekannter Lösungswert)
$\ell$ (nächster Index, über den verzweigt werden soll)
$x_j = b_j$ für $j = 1, ..., \ell - 1$ mit $b_j \in \{0, 1\}$ (bislang fixierte Binärvariable)

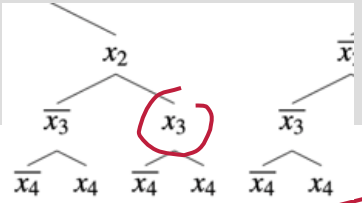
**Ausgabe:** $\max \left\{ \sum_{j=1}^{\ell-1} b_j p_j + \sum_{j=\ell}^{n} x_j p_j \mid \sum_{j=1}^{\ell-1} b_j z_j + \sum_{j=\ell}^{n} x_j z_j \leq Z, x_j \in \{0,1\} \right\}$

Also: Lösung des Knapsackproblems mit den ersten $\ell - 1$ Variablen fixiert

1: **procedure** BRANCH-AND-BOUND($\ell$)
2: **if** $(\sum_{j=1}^{\ell-1} b_j z_j > Z)$ **then return** ▷ unzulässig
3: Compute $L := LB(b_1, \ldots, b_{\ell-1})$
4: **if** $L > P$ **then** $P := L$ ▷ Lösungswert verbessert
5: **if** $(\ell > n)$ **then return** ▷ Blatt im Baum erreicht
6: $U := UB(b_1, \ldots, b_{\ell-1})$ ▷ (Obere Schranke berechnen)
7: **if** $(U > P)$ **then**
8: $b_\ell := 0$; BRANCH-AND-BOUND($\ell + 1$);
9: $b_\ell := 1$; BRANCH-AND-BOUND($\ell + 1$);
10: **return**

Der B&B Pseudocode aus dem Skript

# Branch & Bound




MAXIMUM KNAPSACK

**Allgemein (an Baumknoten):**

1. Test auf Zulässigkeit
2. Schranken berechnen
3. Update globale Schranke
4. Prüfe auf Pruning
5. Verzweigen falls nötig

$$\sum_{i=1}^{l-1} b_i z_i \leq Z$$

$$U = UB(I, b_1, \dots, b_{\ell-1})$$
$$L = LB(I, b_1, \dots, b_{\ell-1})$$

$$P = \max(P, L)$$

$U \leq P \Rightarrow$ ignoriere Teilbaum

1. Betrachte Fall $b_l := 0$
2. Betrachte Fall $b_l := 1$

**Herausforderungen**

Finde gute Schranken, d.h. finde **kleine obere** und **große untere** Schranken.

# Branch & Bound – Schranken für Maximum Knapsack

Obere Schranke:

$$U = UB(I, b_1, \dots, b_{\ell-1})$$
$$L = LB(I, b_1, \dots, b_{\ell-1})$$

Greedy für Fractional Knapsack

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $z_i$ | 11 | 5 | 18 | 16 | 9 |
| $p_i$ | 14 | 6 | 18 | 15 | 7 |
| $x_i$ | 1 | 1 | 0.5 | 0 | 0 |

$Z = 25$

**Herausforderungen**

Finde gute Schranken, d.h. finde **kleine obere** und **große untere** Schranken.

# Branch & Bound – Schranken für Maximum Knapsack

$$U = UB(I, b_1, \dots, b_{\ell-1})$$
$$L = LB(I, b_1, \dots, b_{\ell-1})$$

Obere Schranke:

Greedy für Fractional Knapsack
→ Ergebnis abrunden
(denn wir arbeiten nur mit ganzen Zahlen)

Untere Schranke:

GREEDY0 (für Maximum Knapsack)

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $z_i$ | 11 | 5 | 18 | 16 | 9 |
| $p_i$ | 14 | 6 | 18 | 15 | 7 |
| | 1 | 1 | 0 | 0 | 1 |

$Z = 25$

**Herausforderungen**

Finde gute Schranken, d.h. finde **kleine obere** und **große untere** Schranken.

# Wie Schranken berechnen mit Fixierungen?

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $z_i$ | 11 | 5 | 18 | 16 | 9 |
| $p_i$ | 14 | 6 | 18 | 15 | 7 |
| $x_i$ | ? | ? | ? | ? | ? |

$$Z = 25$$

# Wie Schranken berechnen mit Fixierungen?

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $z_i$ | 11 | 5 | 18 | 16 | 9 |
| $p_i$ | 14 | 6 | 18 | 15 | 7 |
| $x_i$ | 0 | 1 | ? | ? | ? |

fixiert

$Z = 25$

Greedy mit Fixierungen?

→ Kriegen wir hin :)

# Wie Schranken berechnen mit Fixierungen?

| $i$ | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| $z_i$ | 5 | 18 | 16 | 9 |
| $p_i$ | 6 | 18 | 15 | 7 |
| $x_i$ | 1 | ? | ? | ? |

fixiert

$$Z = 25$$

# Wie Schranken berechnen mit Fixierungen?

| $i$ | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| $z_i$ | 5 | 18 | 16 | 9 |
| $p_i$ | 6 | 18 | 15 | 7 |
| $x_i$ | 1 | ? | ? | ? |

fixiert

$Z = 25 \quad -5$

$Solution + 6$

# Wie Schranken berechnen mit Fixierungen?

| $i$ | 3 | 4 | 5 |
|---|---|---|---|
| $z_i$ | 18 | 16 | 9 |
| $p_i$ | 18 | 15 | 7 |
| $x_i$ | ? | ? | ? |

$Z = 25 \quad -5$

$Solution + 6$

# Rolle der jeweiligen Schranken

**Bei Maximierungsproblemen (bei Minimierung genau umgekehrt):**

- Untere Schranke: Global, entspricht bester bisher gefundener Lösung
- Obere Schranke: Lokal, beschränkt Wert bester möglicher Lösung im aktuellen Teilbaum
- Untere Schranke: Gibt uns am Ende die optimale Lösung
- Obere Schranke: Hilft, den Baum kleiner zu halten

**Wofür berechnen wir Schranken an Baumknoten?**

- Obere Schranke: Prüfen, ob wir diesen Knoten weiter betrachten müssen
- Untere Schranke: Frühzeitiges Erkennen von neuen, möglicherweise besseren Lösungen (streng genommen optional)

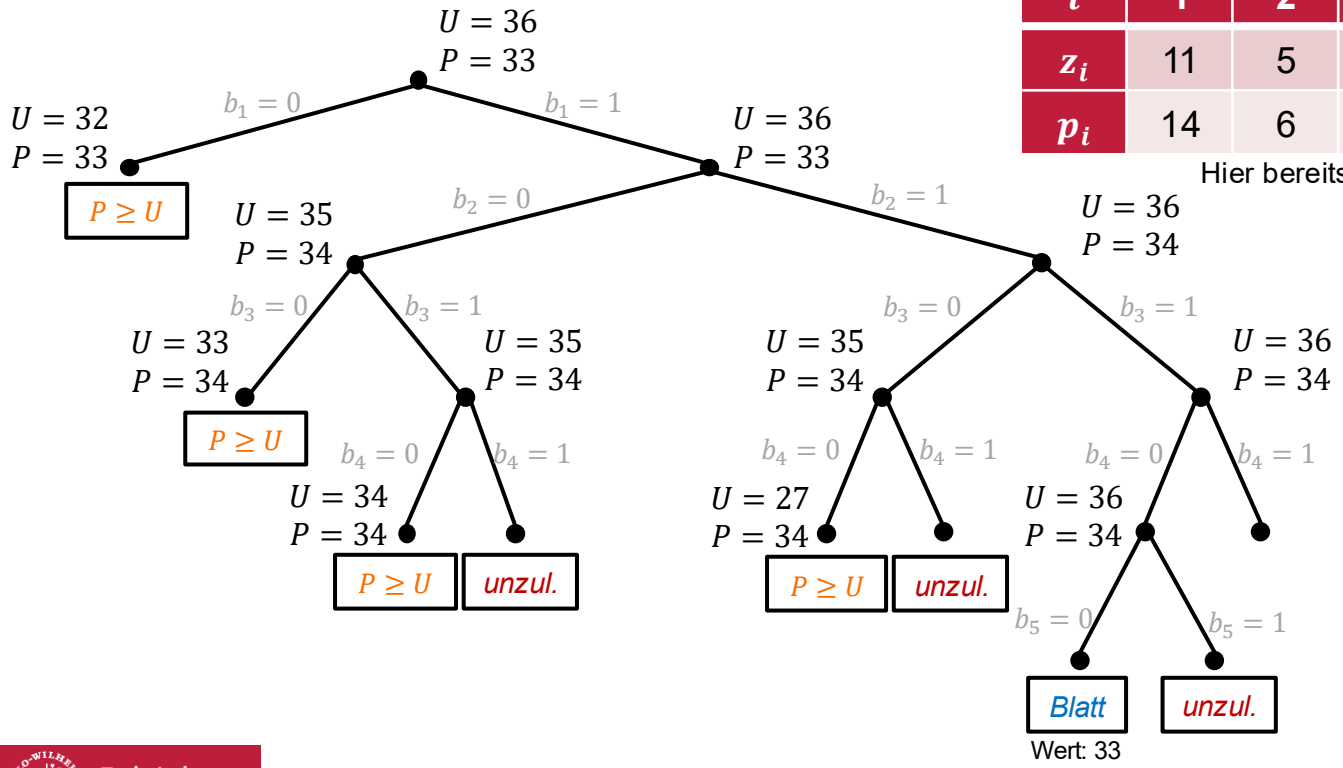
- Innere Knoten des Suchbaums: Partielle Zuweisung der Variablen
- Blätter: Vollständige Zuweisungen der Variablen (also potentielle Lösungen)

# Branch & Bound – Beispiel

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $z_i$ | 11 | 5 | 13 | 18 | 9 |
| $p_i$ | 14 | 6 | 13 | 16 | 7 |

Hier bereits sortiert, voll praktisch!*

$Z = 33$




| Menge | Wert |
|---|---|
| | |
| | |



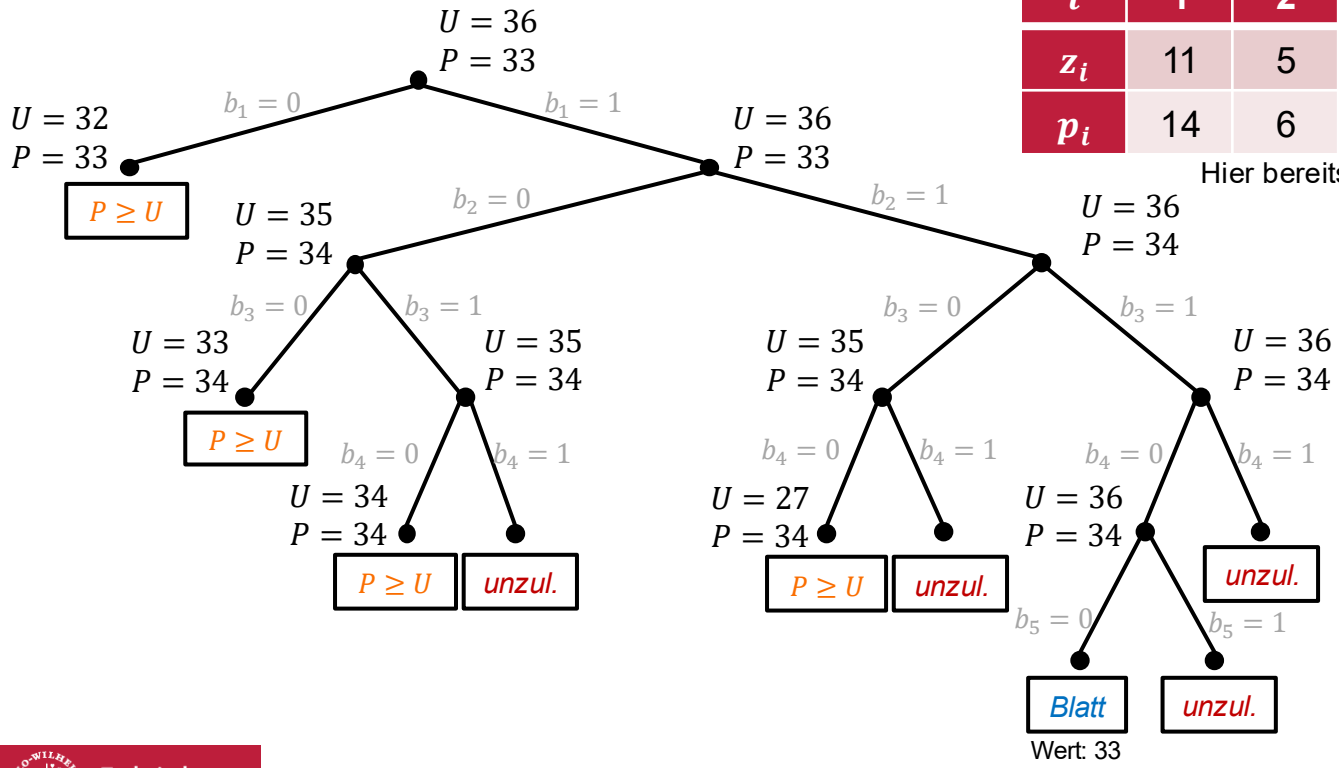
*das ist in Übungsaufgaben oft *nicht* so!

# Branch & Bound – Beispiel

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $z_i$ | 11 | 5 | 13 | 18 | 9 |
| $p_i$ | 14 | 6 | 13 | 16 | 7 |

Hier bereits sortiert, voll praktisch!*

$Z = 33$




| Menge | Wert |
|---|---|
| {1,2,3} | 33 |
| {1,3,5} | 34 |



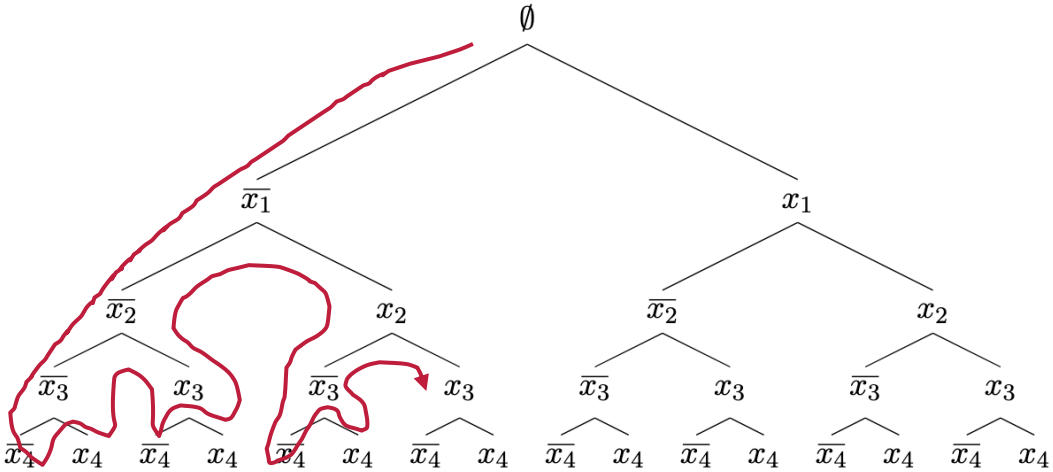
*das ist in Übungsaufgaben oft *nicht* so!

# Fragen?

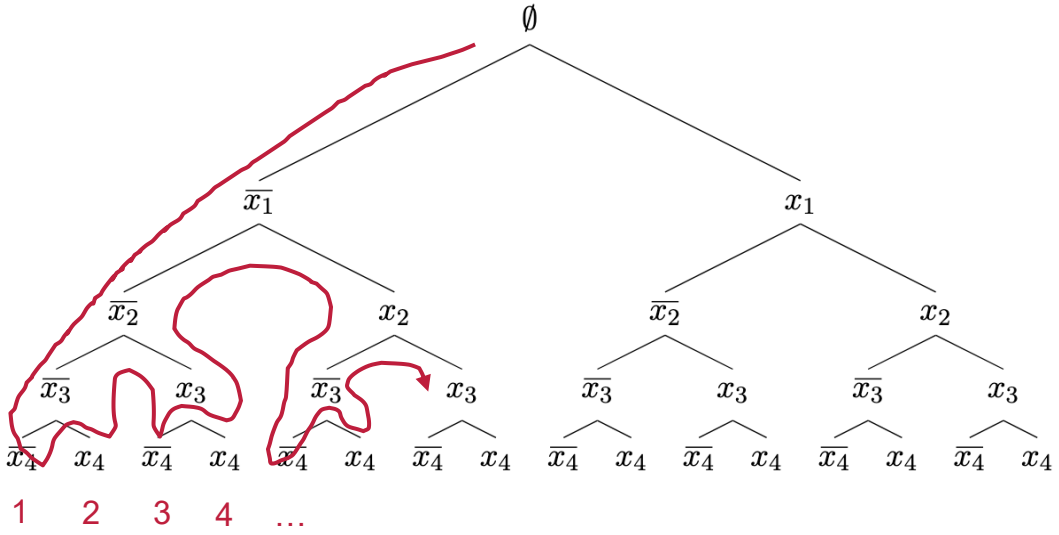
# Wie durchläuft man den B&B-Baum am schlausten?

Die eben vorgestellte Variante ist das, was wir in Hausaufgaben und Klausur sehen wollen.

*Aber das sind nur die Grundlagen! Beim Design von B&B-Algorithmen hat man viele Freiheiten…*



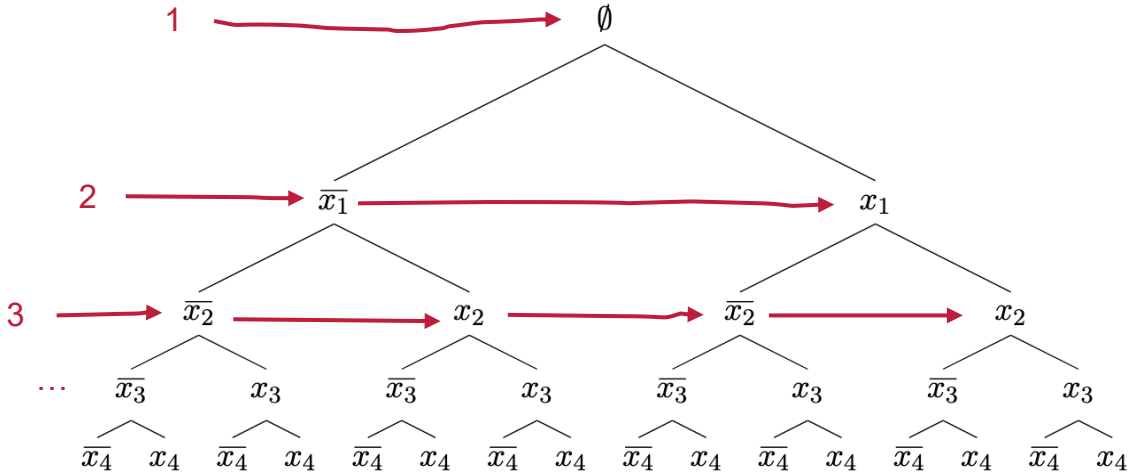
# Wie durchläuft man den B&B-Baum am schlausten?




**Tiefensuche**

- Erreichen schnell Blätter
- (Finden schnell Lösungen)
- Was, wenn die erste Entscheidung generell schlecht ist und wir auf ewig an Details feilen?

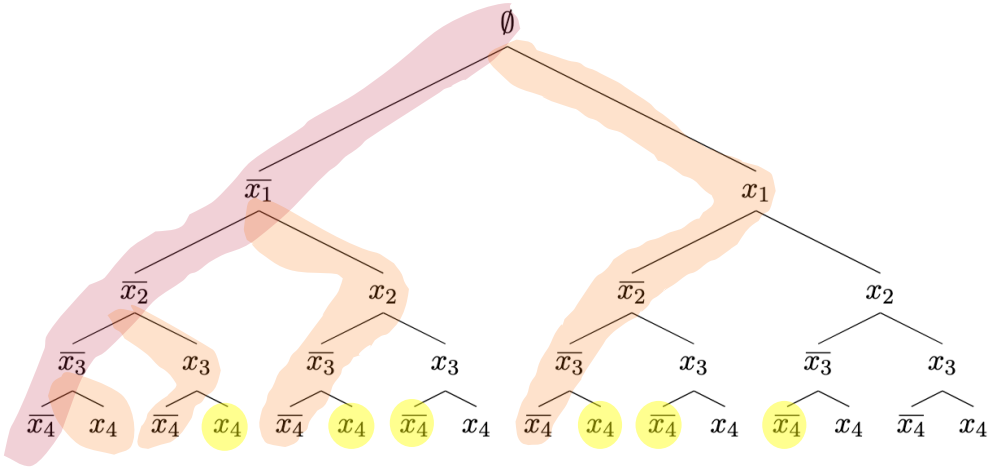
# Wie durchläuft man den B&B-Baum am schlausten?




**Breitensuche**

- Schneller, breiter Überblick
- Problematisch bzgl. benötigtem Speicher

# Wie durchläuft man den B&B-Baum am schlausten?




- Starte mit einer Lösung, die vermutlich schon viele richtige Entscheidungen getroffen hat (z.B. Greedy)
- Prüfe diesen Branch
- Prüfe Branches, wo 1 Variable anders ist
- Prüfe Branches, wo 2 Vars anders sind
- …

→ Vielleicht ein guter Kompromiss?

# Wie durchläuft man den B&B-Baum am schlausten?

Weitere Ideen für smartes B&B?

Über welche Variable branched man als nächstes?
→ Wenn es LP-Relaxierung gibt, sind oft die gebrochenen Variablen die spannendsten

Was sind Upper und Lower Bounds?
→ Müssen individuell fürs Problem gefunden werden!

Was, wenn Lower Bound schlechte Schranken liefert? Oder wir keine finden? Und die Berechnung sehr rechenintensiv ist?
→ Theoretisch brauchen wir überhaupt keine zusätzlichen Lower Bound zu berechnen (bei Max. Problem):
Jedes Blatt entspricht einer Lösung.
→ Aber: Baum wird halt größer, Lower Bound schon sinnvoll!
→ Upper Bound brauchen wir.

Propagation: Beim Durchlaufen des Baums logische Zusammenhänge ableiten, mit denen noch mehr Äste abgeschnitten werden können!

# Wie durchläuft man den B&B-Baum am schlausten?

… und was ist jetzt am schlausten?

- Joar, das hängt ganz von dem Problem *und* den Instanzen ab!

→ Ausprobieren!

# Branch & Bound – Laufzeit

**Satz 3.2.** *Algorithmus 3.1 (als rekursiv arbeitende Unterroutine) berechnet eine Optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der Schranken ist.*

# Branch & Bound – Laufzeit

**Satz 3.2.** *Algorithmus 3.1 (als rekursiv arbeitende Unterroutine) berechnet eine Optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit* $O(2^n f(n))$*, wobei* $f(n)$ *die Zeit für die Berechnung der Schranken ist.*

In unserem Fall: $f(n) \in O(n \log n)$; mit geschickter Vorsortierung $O(n)$

$2^n$ ist hierbei der schlimmste Teil!

(Das ist aber ”nur” der Worse-Case.
In der Praxis perform B&B üblicherweise viel besser)

AND NOW FOR SOMETHING
COMPLETELY DIFFERENT.

# Euclidean Traveling Salesman Problem

# Euclidean Traveling Salesman Problem

Keine beliebig ausgdachten Werte für die Distanzen, sondern basierend auf Punkten im euklidischen (2D) Raum.

Kürzeste Rundreise

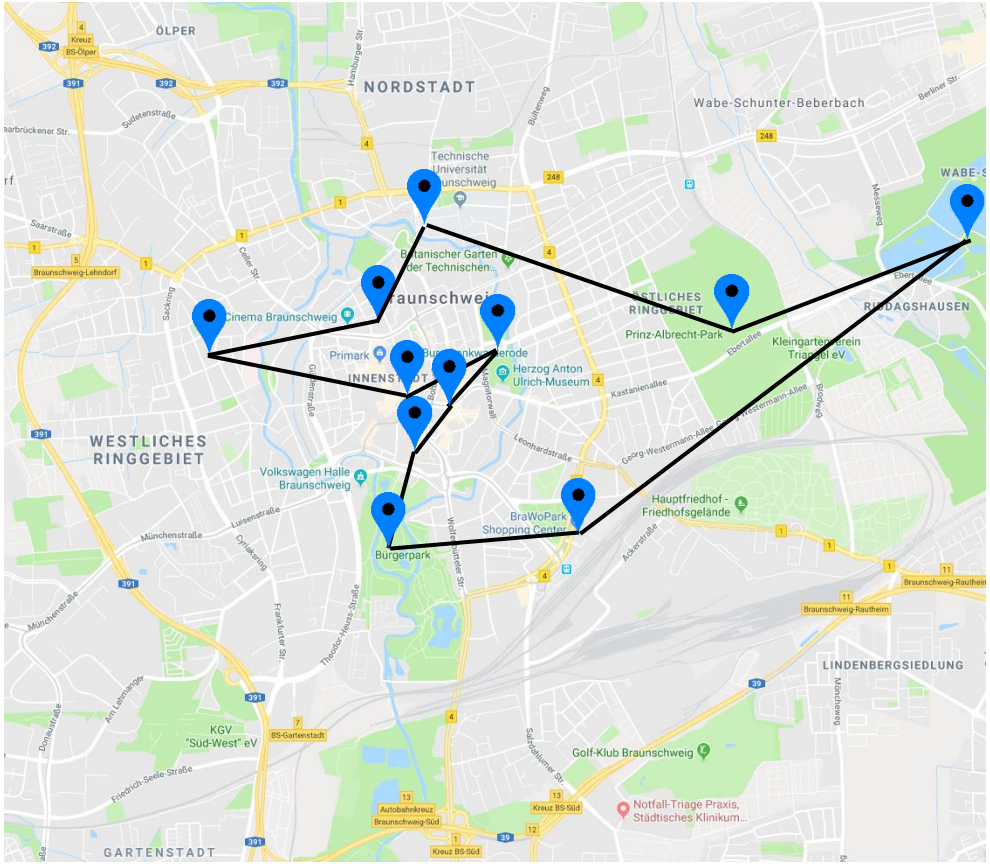
# Euclidean Traveling Salesman Problem

**Gegeben:**
Punkte $p_1, \dots, p_n \in \mathbb{R}^2$

**Gesucht:**
Permutation $\pi$ mit

$$\sum_{i=1}^{n} \left\| p_{\pi(i)} - p_{\pi(i+1)} \right\|_2$$

minimal. Dabei ist $p_{\pi(n+1)} = p_{\pi(1)}$.
Finde also eine **kürzeste** Rundreise, die alle Punkte besucht.



# Euclidean Traveling Salesman Problem

**Gegeben:**
Punkte $p_1, \dots, p_n \in \mathbb{R}^2$

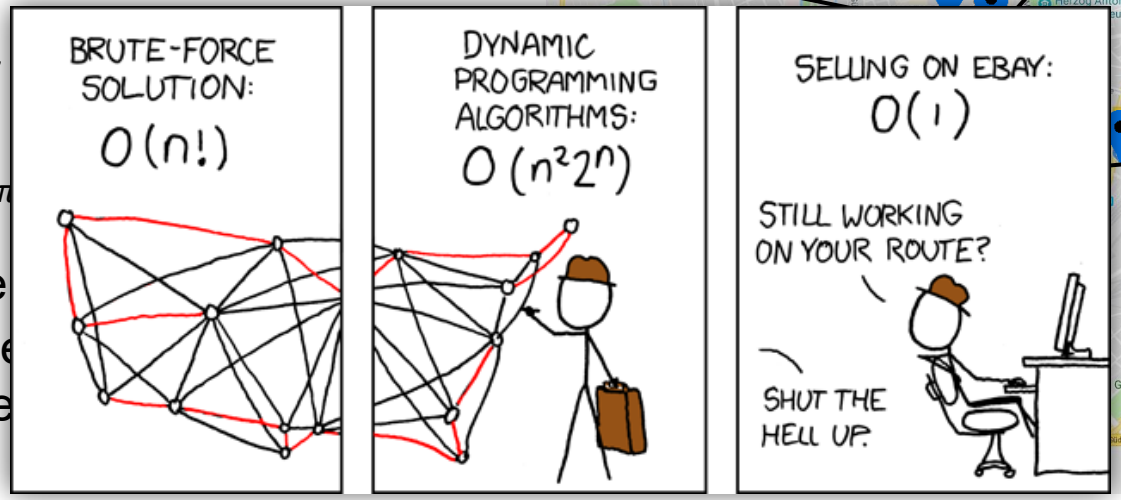
**Gesucht:**
Permutation $\pi$

$$\sum_{i=1}^{n} \|p_{\pi}$$

minimal. Dabe

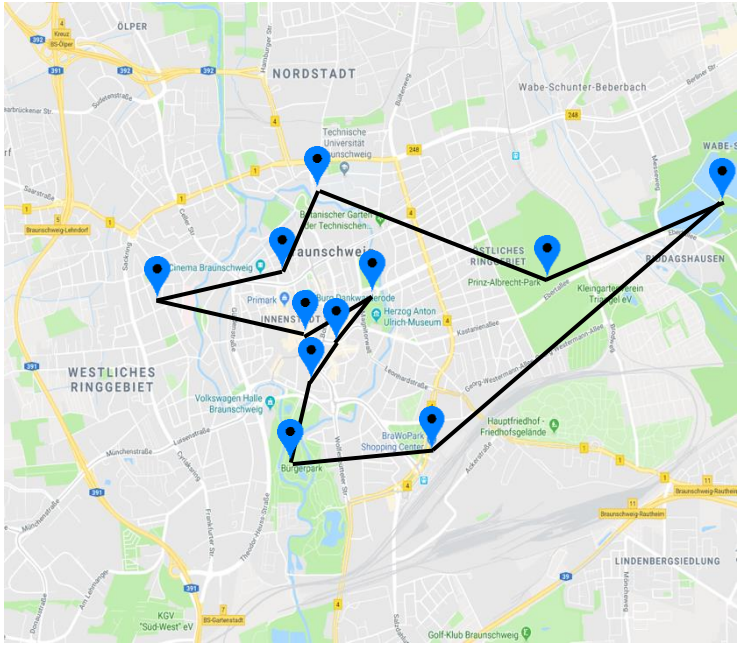
Finde also eine

die alle Punkte

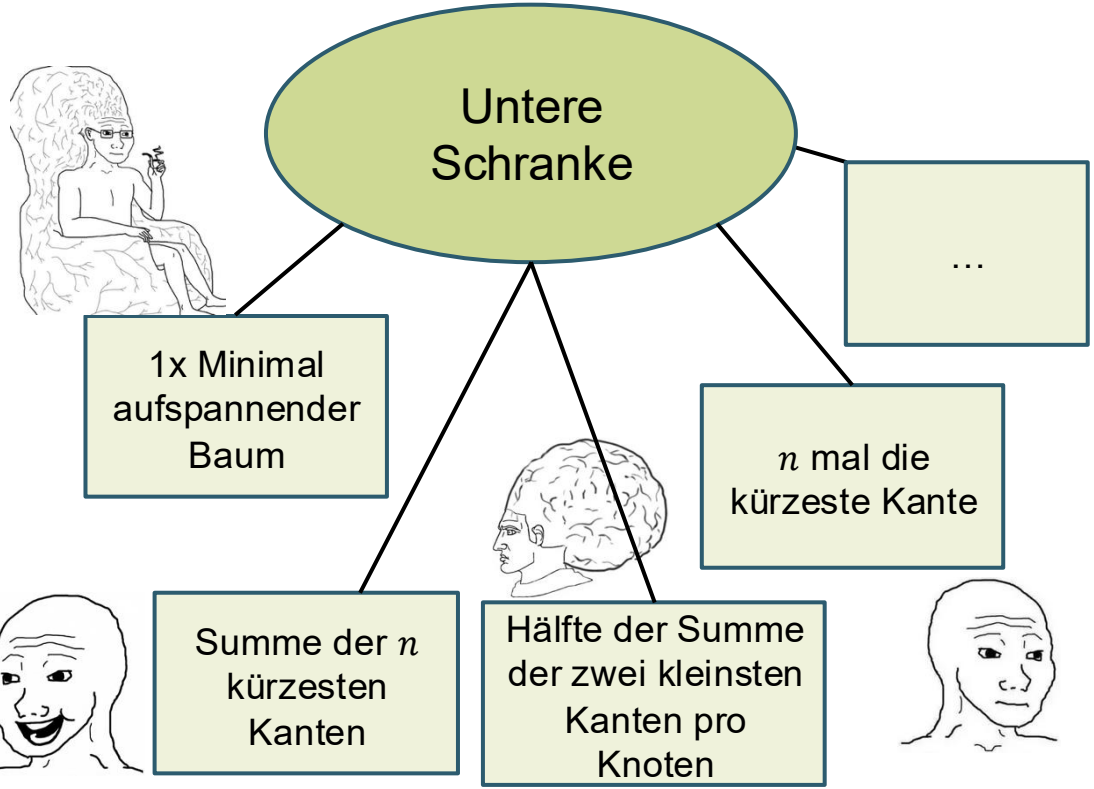



https://imgs.xkcd.com/comics/travelling_salesman_problem.png

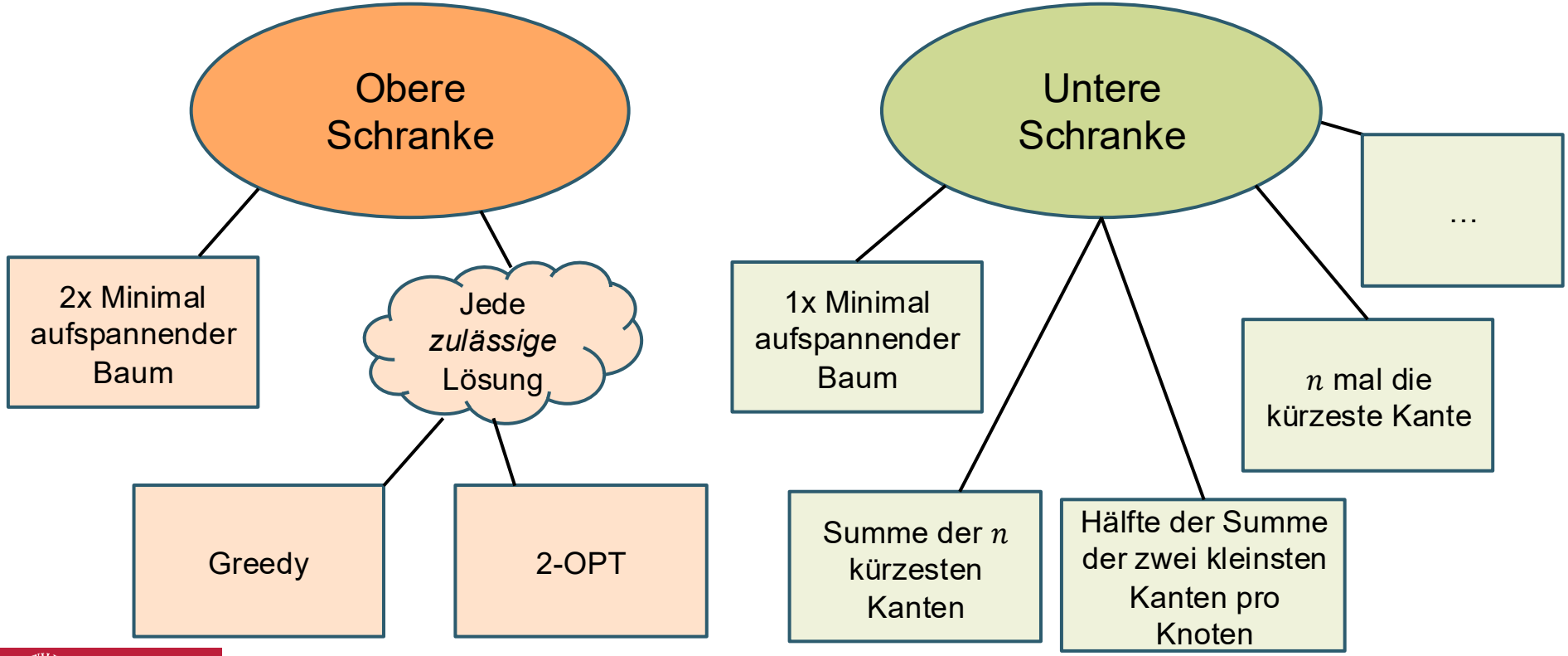
# ETSP und B&B: Schranken




Jede Rundreise hat $n$ Knoten und Kanten.

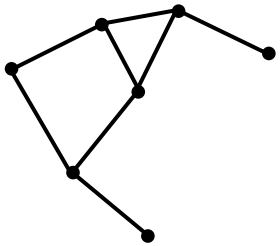


# ETSP und B&B: Schranken



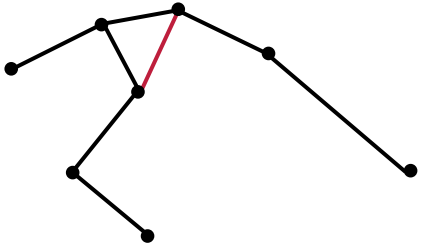
# ETSP – Untere Schranken

Summe der $n$ kürzesten Kanten





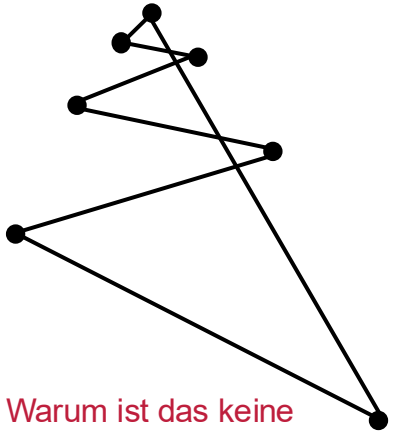
1x Minimal aufspannender Baum
(+ kürzeste Kante)



# ETSP – Obere Schranken

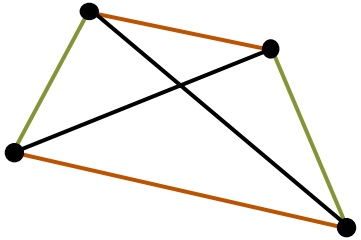
Greedy
Nearest-Neighbor

Starte bei $p_1$ und gehe iterativ zum nächstgelegenen Punkt.



Warum ist das keine optimale Lösung?

**Grundsätzliche Feststellung im ETSP:**
Sobald sich zwei Kanten kreuzen, kann ich sie tauschen und bekomme eine Lösung, die (gleich oder) kürzer ist



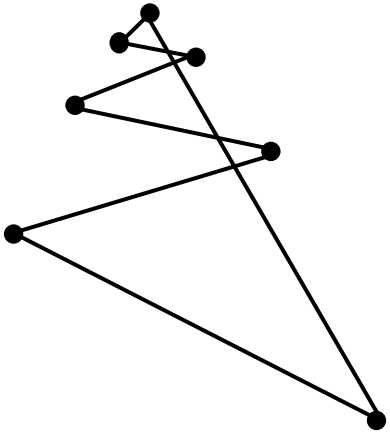
Die grünen bzw. braunen Kanten sind höchstens so lang wie die schwarzen.
Grund: Dreiecksungleichung

$$d(a,b) + d(b,c) \geq d(a,c)$$

# ETSP – Obere Schranken

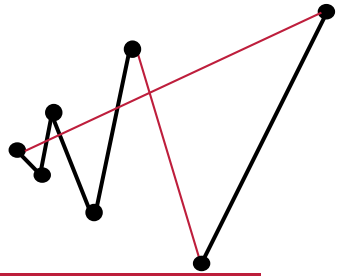
Greedy
Nearest-Neighbor

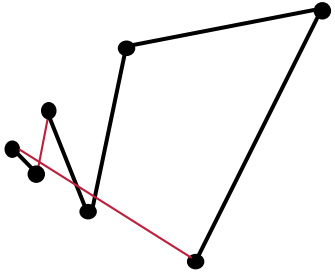
Starte bei $p_1$ und gehe iterativ zum nächstgelegenen Punkt.

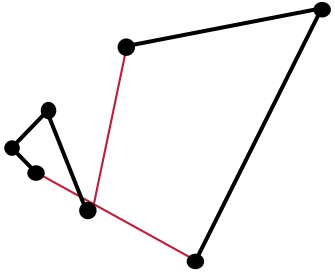


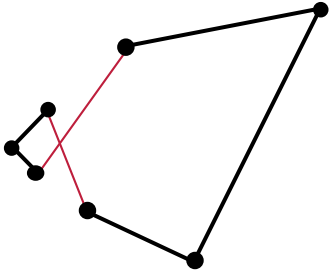
2-OPT
(Das ist nur der Name, das hat nichts mit optimalen Lösungen zu tun)

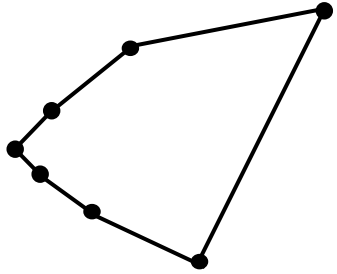
In einer beliebigen Tour, tausche zwei existierende Kanten mit zwei nicht-existierenden Kanten, um eine bessere Tour zu erhalten. (Bis keine Verbesserung mehr erreicht wird)











# ETSP – Branch-and-Bound

Zunächst: Worüber verzweigen? Wo können wir Entscheidungen treffen?

**Knoten**

- Entscheide für jeden Knoten, wann er besucht wird.
- Verschiedene Knoten dürfen nicht gleichzeitig besucht werden.

**Kanten**

- Kante ist entweder da, oder nicht.
- Entscheidungen sind unabhängig.
- Einfach Schranken zu finden.

**(Teil-)Kreise/Pfade**

- Also alle Permutationen testen?
- Vorgänger von Knoten raten?
- Wie kann ich darüber Schranken finden?

# ETSP – zulässige Teillösungen

Betrachte eine beliebige Reihenfolge der Kanten. Sei $x_i \in \{0,1\}$ die Entscheidungsvariable für die $i$-te Kante.

Wann ist eine Auswahl von Kanten nicht zulässig?

1. Maximal $n$ Kanten ausgewählt, also
   $\sum_i x_i \leq n$
2. An jedem Punkt $p$ dürfen nur zwei Kanten liegen, also
   $\sum_{i \in \delta(p)} x_i \leq 2$
3. Falls $< n$ Kanten ausgewählt sind, darf kein Kreis existieren. (Erkennbar mit BFS/DFS aus AuD)
4. Falls $= n$ Kanten ausgewählt sind, muss exakt ein Kreis existieren. (Wieder mit BFS/DFS)
5. Es dürfen sich keine zwei Kanten schneiden.

# ETSP – Lösen mit linearen Ungleichungssystemen

Etwas anderer Ansatz:

$$\min \sum_{e \in E} c_e x_e$$

s.t.

$$\sum_{e \in \delta(p)} x_e = 2 \qquad \text{für alle } p \in \mathrm{P}$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \text{für alle } \emptyset \neq S \subsetneq P$$

$$x_e \in \{0,1\} \qquad \text{für alle } e \in E$$

$E$: Menge aller Knotenpaare
$c_e$: Distanz des Knotenpaares $e$

Das ist ein
*Integer Linear Program*

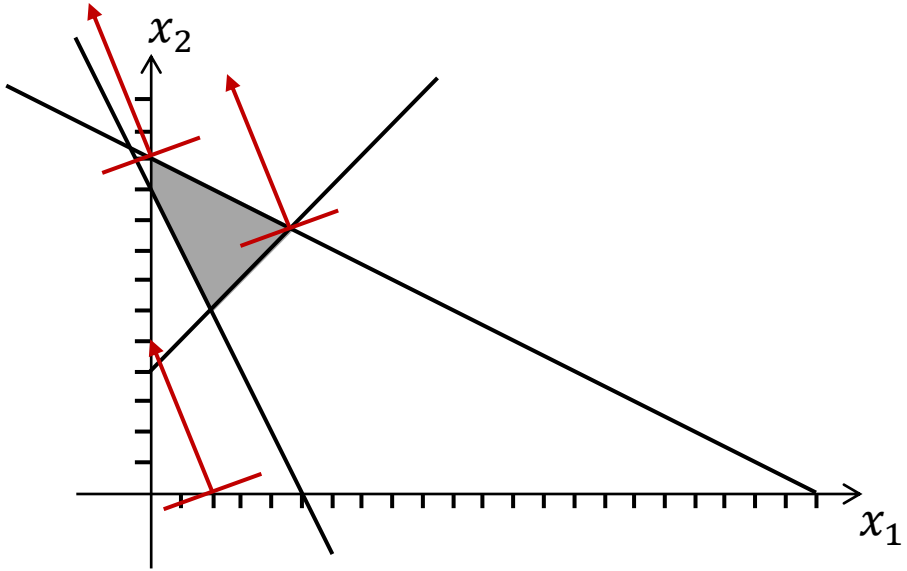
# Linear/Integer Programming

Man stellt ein (Un-)Gleichungssystem mit $m$ Variablen in der folgenden Form auf:

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t. } Ax \leq b \\ &\qquad x \in \mathbb{R}^m \end{aligned}$$

Beispiel:

$$\begin{aligned} &\min\ 2x_1 - 5x_2 \\ &\text{s.t.}\quad x_1 + 2x_2 \leq 22 \\ &\qquad 2x_1 + x_2 \geq 10 \\ &\qquad -x_1 + x_2 \geq 4 \\ &\qquad x_1, x_2 \geq 0 \end{aligned}$$



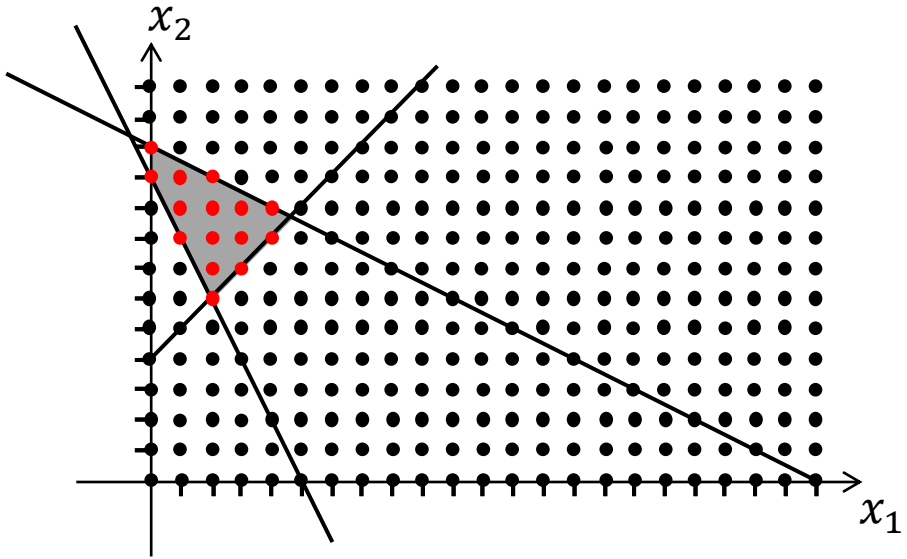
# Linear/Integer Programming

Man stellt ein (Un-)Gleichungssystem mit $m$ Variablen in der folgenden Form auf:

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t. } Ax \leq b \\ &\qquad x \in \mathbb{R}^m \end{aligned}$$

Beispiel:

$$\begin{aligned} &\min\ 2x_1 - 5x_2 \\ &\text{s.t.}\quad x_1 + 2x_2 \leq 22 \\ &\qquad 2x_1 + x_2 \geq 10 \\ &\qquad -x_1 + x_2 \geq 4 \\ &\qquad x_1, x_2 \in \mathbb{N} \end{aligned}$$



# Linear/Integer Programming

## LP

$$\begin{aligned} \min\ & 2x_1 - 5x_2 \\ \text{s.t.}\ & x_1 + 2x_2 \leq 22 \\ & 2x_1 + x_2 \geq 10 \\ & -x_1 + x_2 \geq 4 \\ & x_1, x_2 \geq 0 \end{aligned}$$

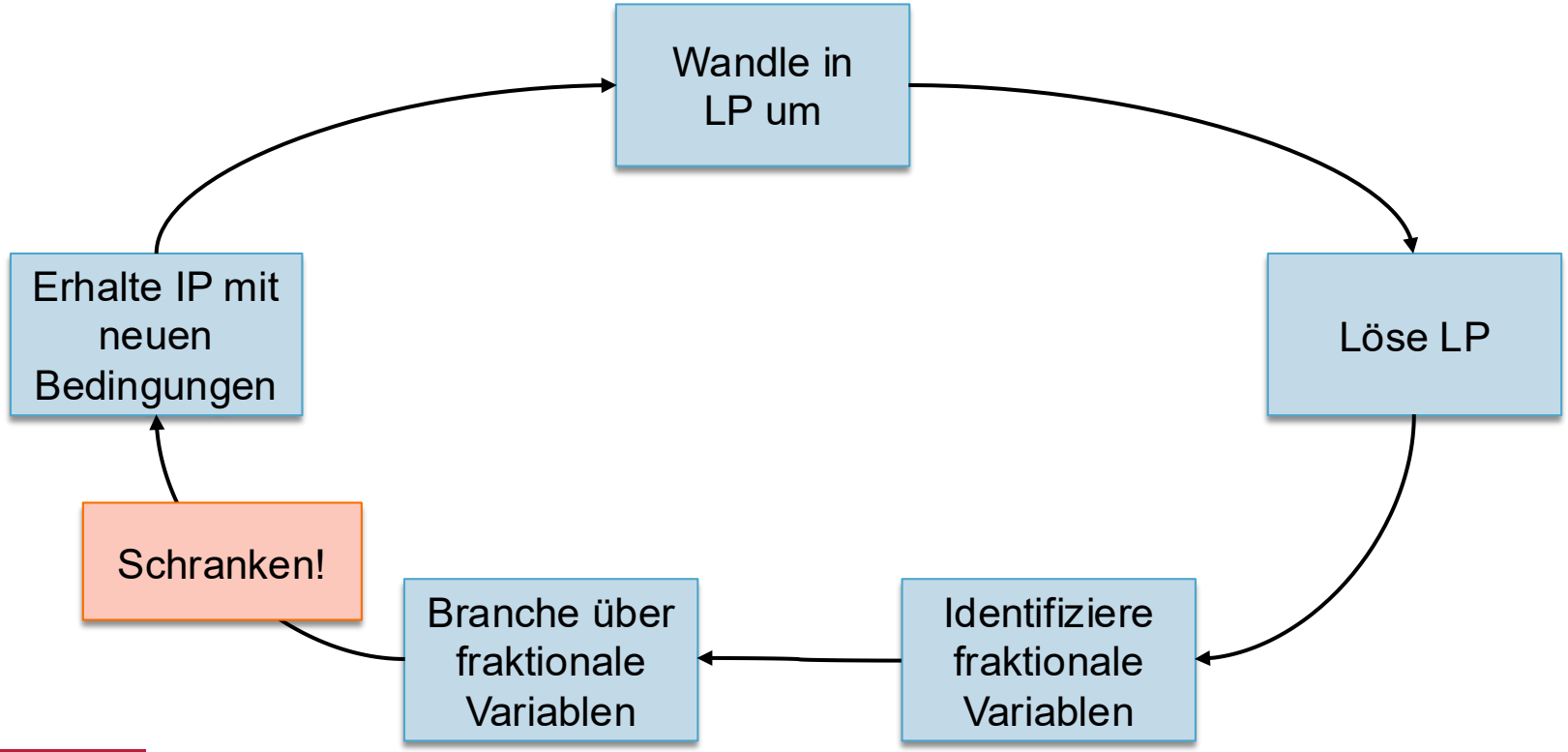
Effizient lösbar

## IP

$$\begin{aligned} \min\ & 2x_1 - 5x_2 \\ \text{s.t.}\ & x_1 + 2x_2 \leq 22 \\ & 2x_1 + x_2 \geq 10 \\ & -x_1 + x_2 \geq 4 \\ & x_1, x_2 \in \mathbb{N} \end{aligned}$$

Vermutlich nicht effizient lösbar

# Lösen von IPs (Vereinfachte Darstellung)



# Solver für Linear/Integer Programming

Microsoft Excel

*CP-SAT ist nur begrenzt für MIPs allgemein nutzbar – aber z.B. TSP geht

# ETSP

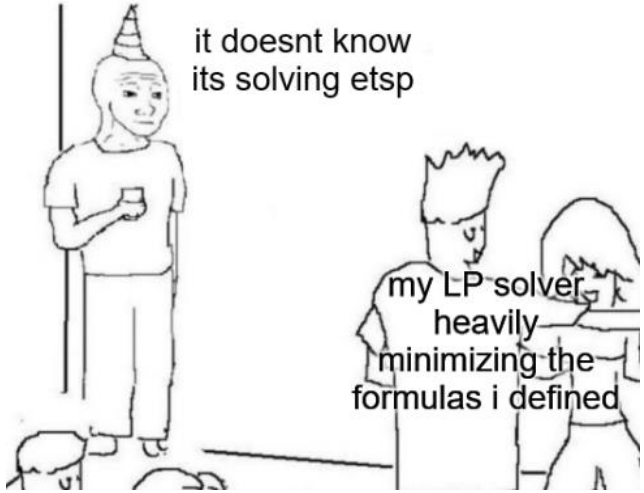
$$\min \sum_{e \in E} c_e x_e$$

s.t.

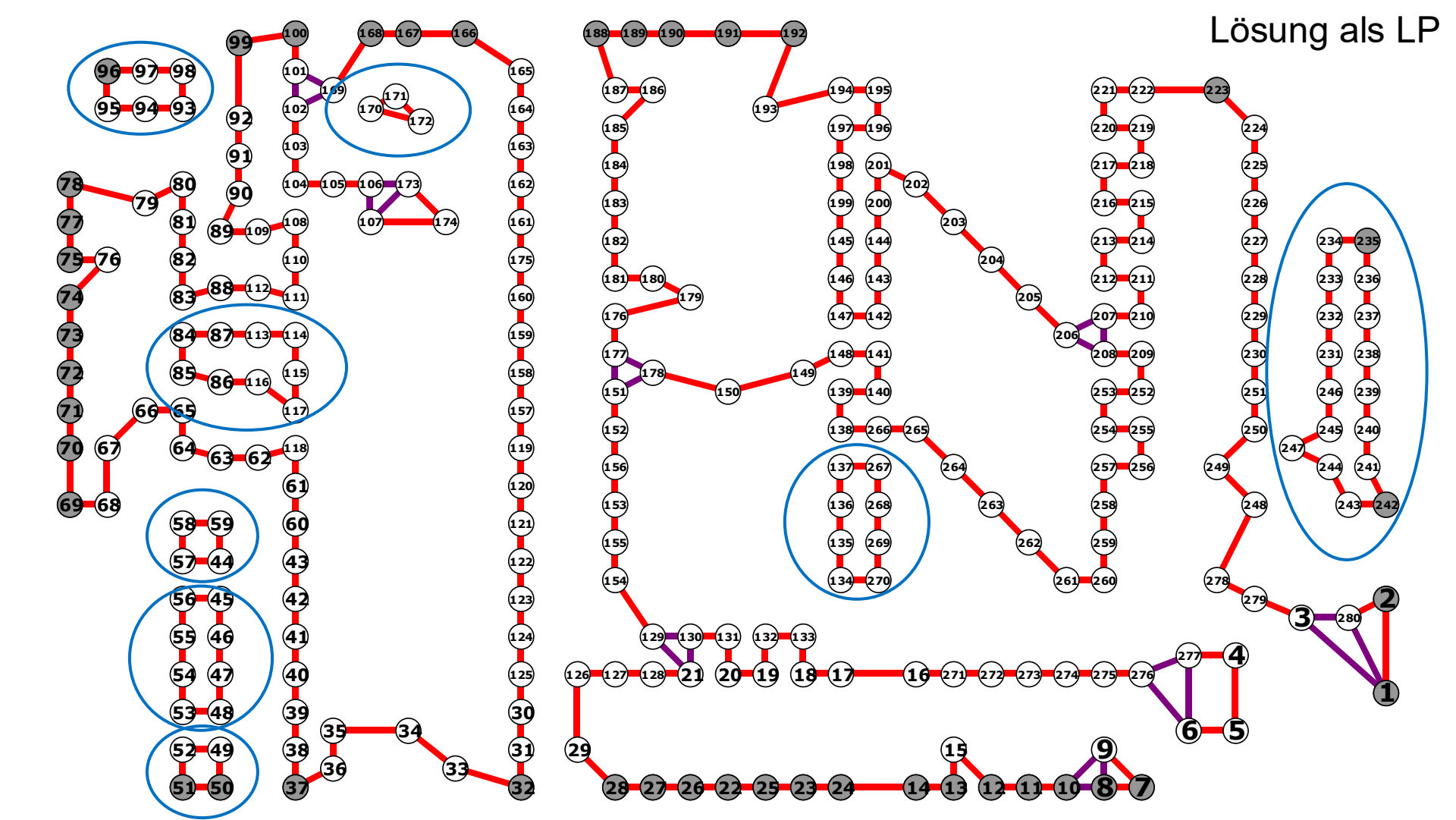
$$\sum_{e \in \delta(p)} x_e = 2 \qquad \text{für alle } p \in \mathrm{P}$$

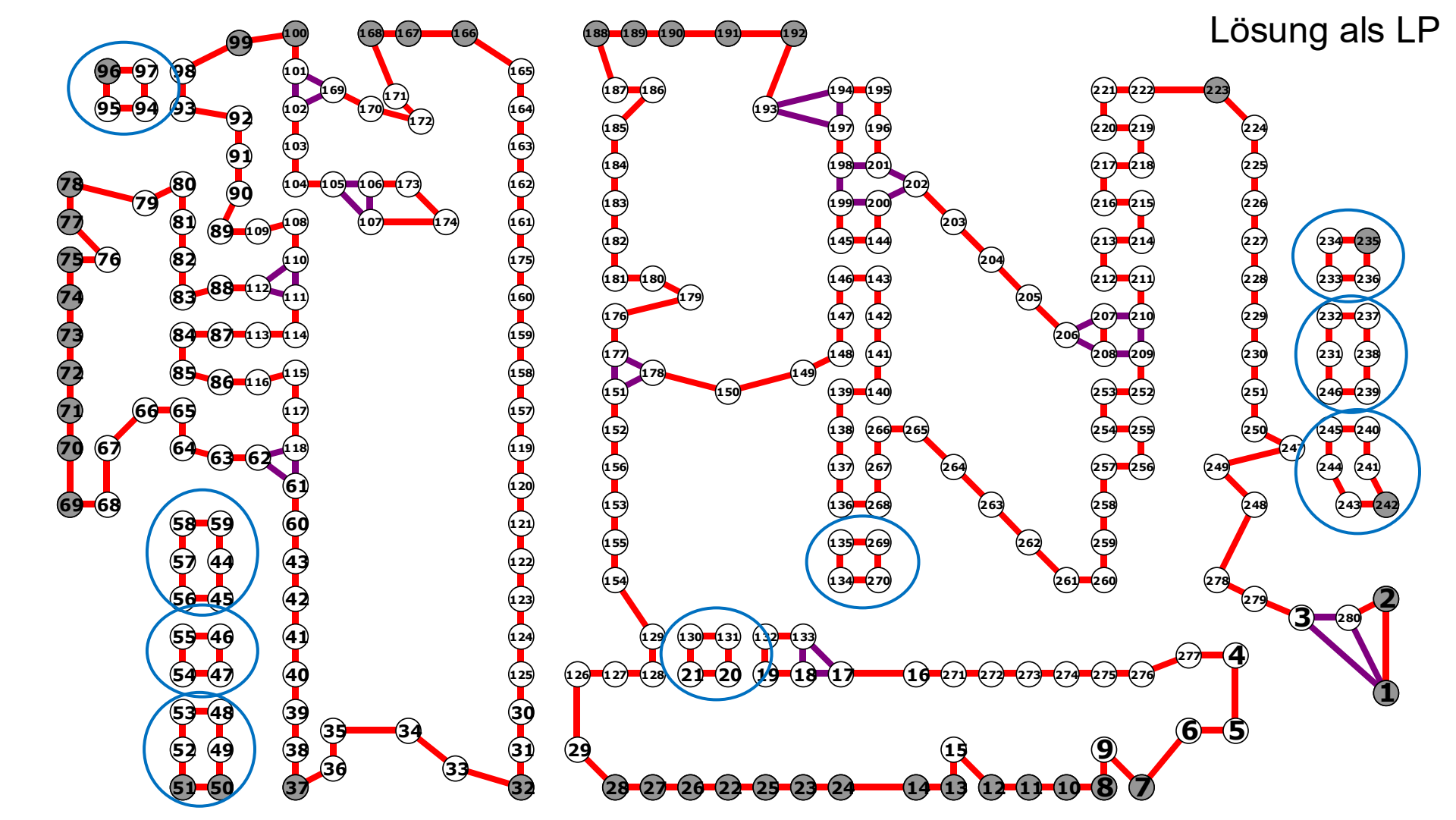
$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \text{für alle } \emptyset \neq S \subsetneq P$$

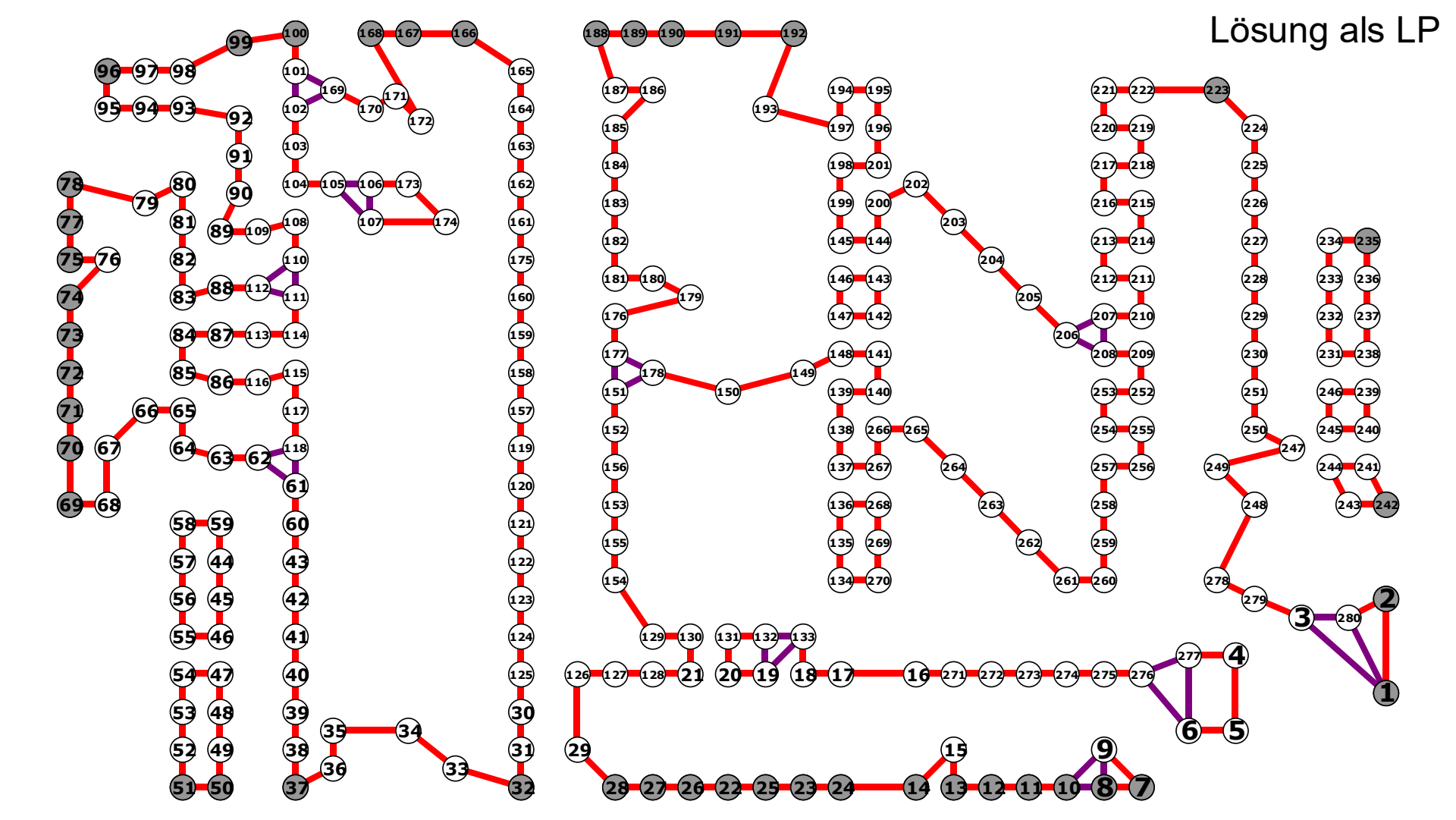
$$x_e \in \{0,1\} \qquad \text{für alle } e \in E$$

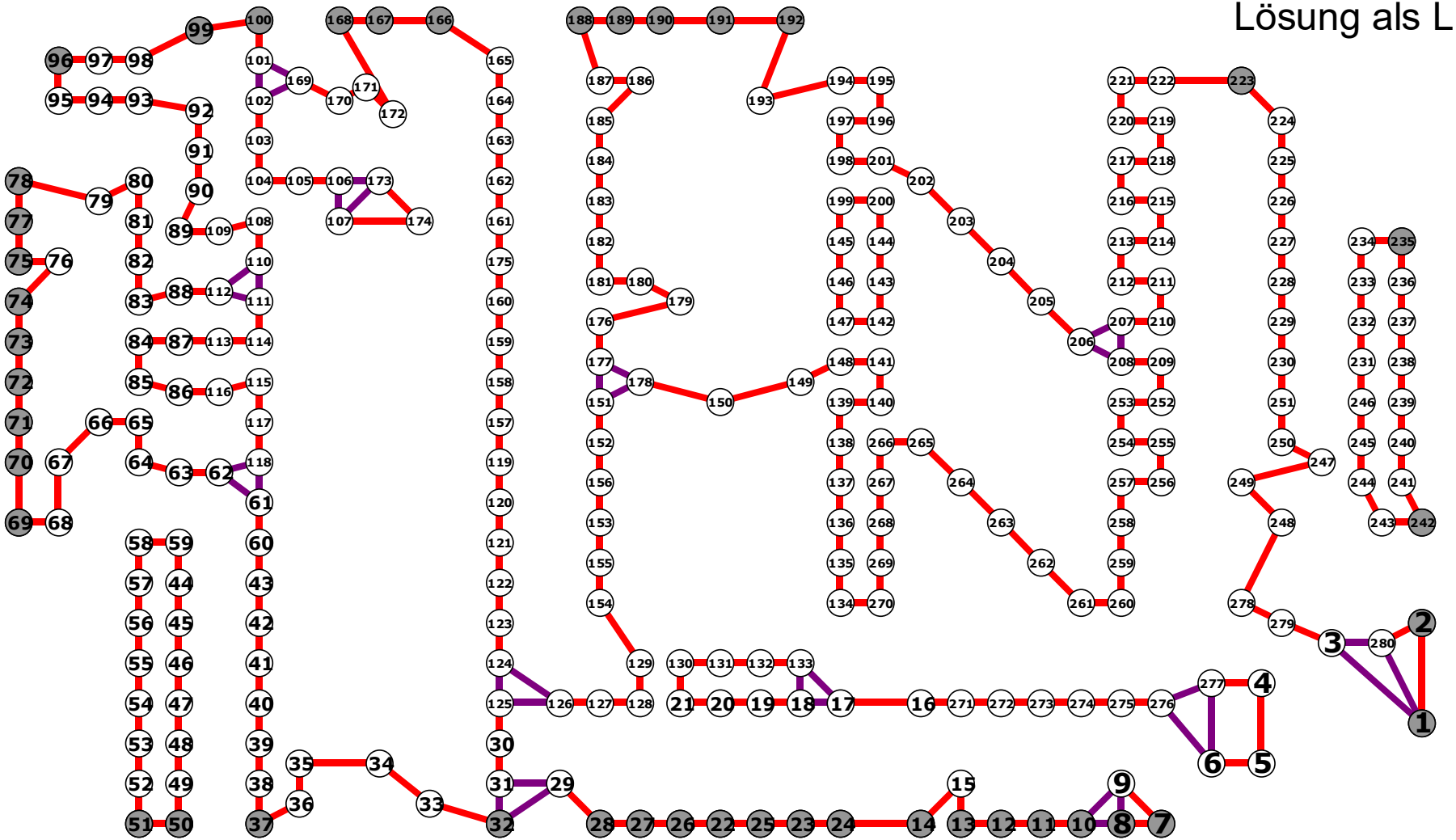


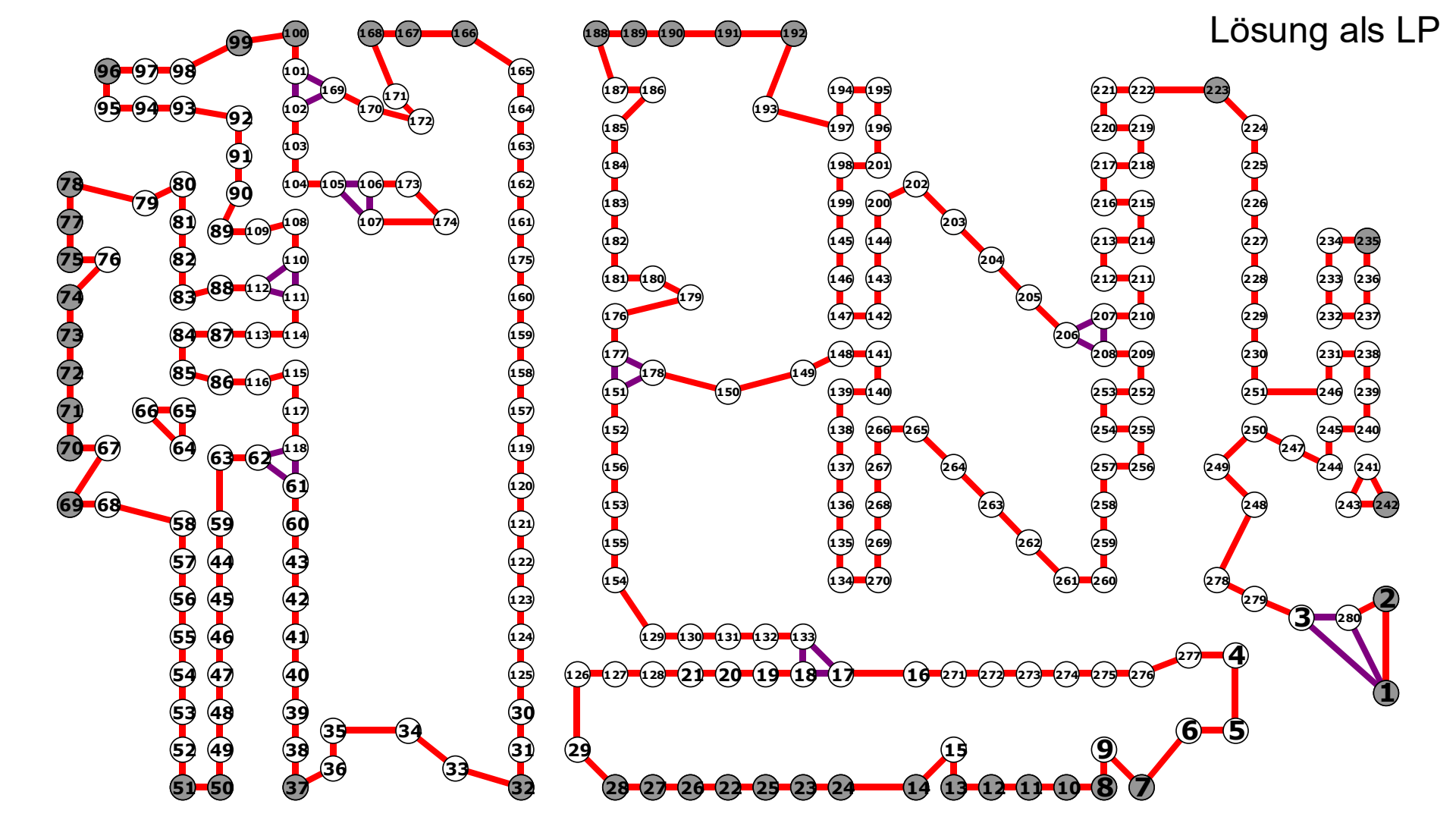

Exponentiell viele…
Füge sie während des Lösungsprozesses hinzu!

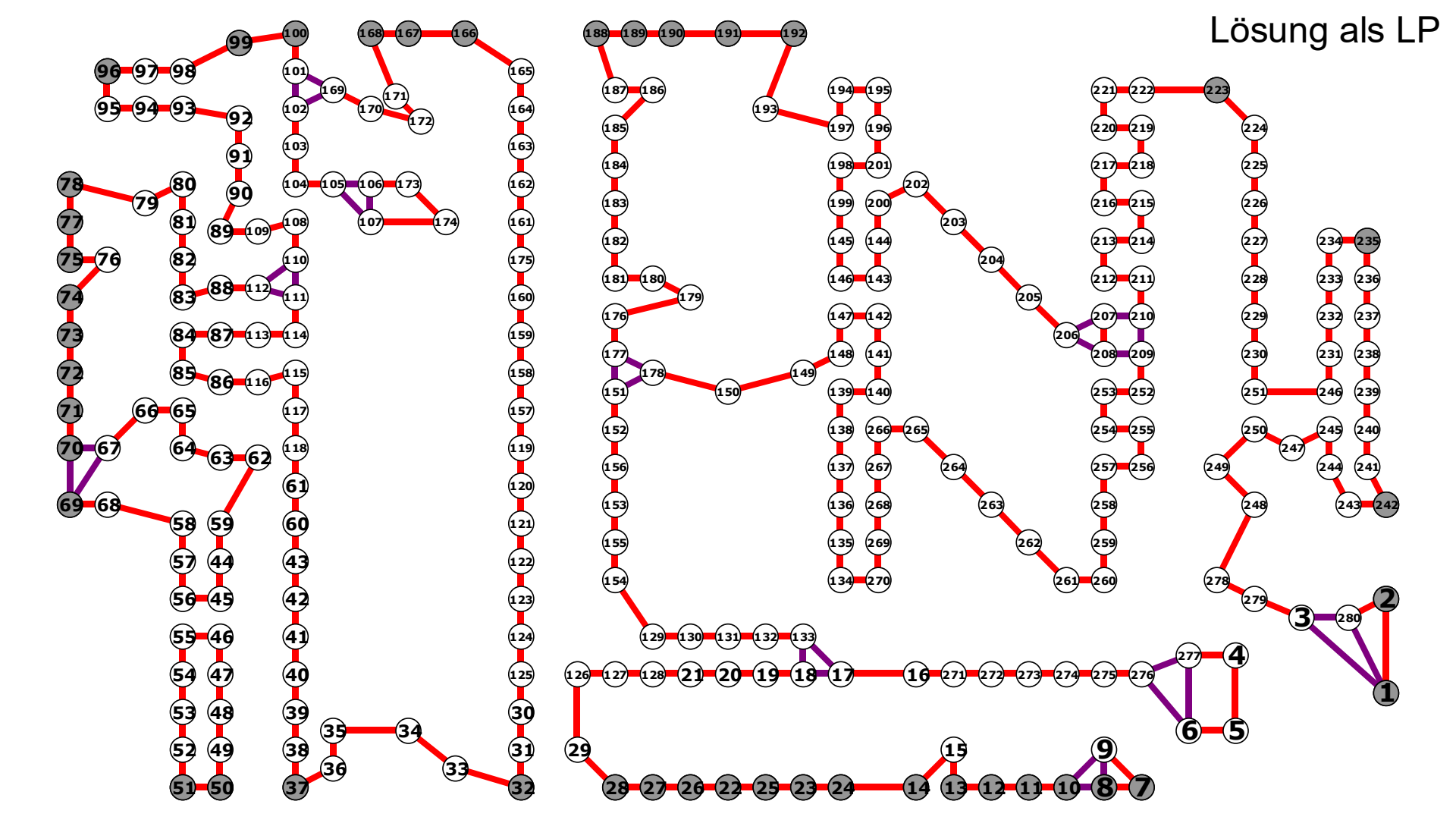

Lösung als LP

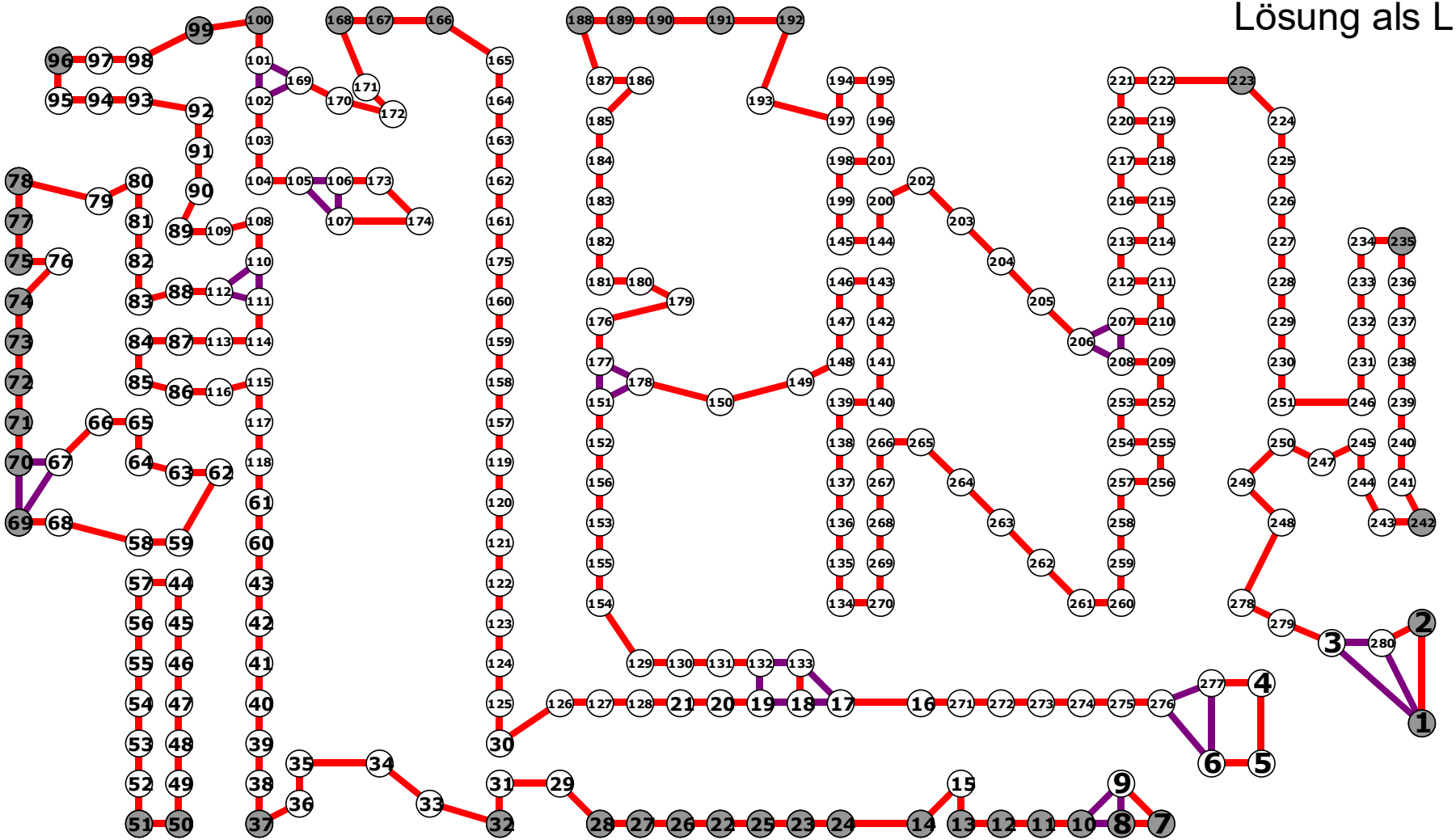

Lösung als LP

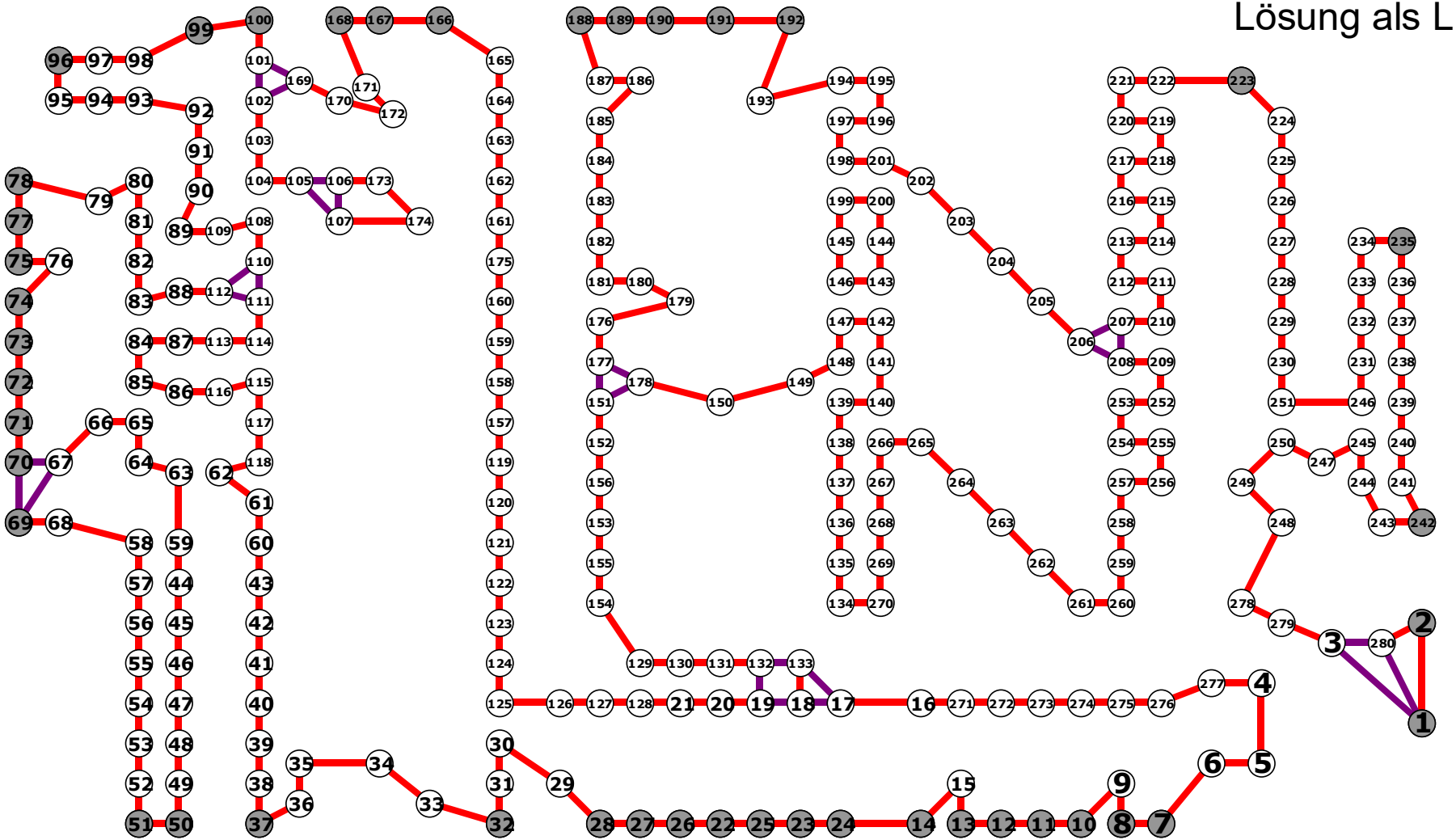

Lösung als LP

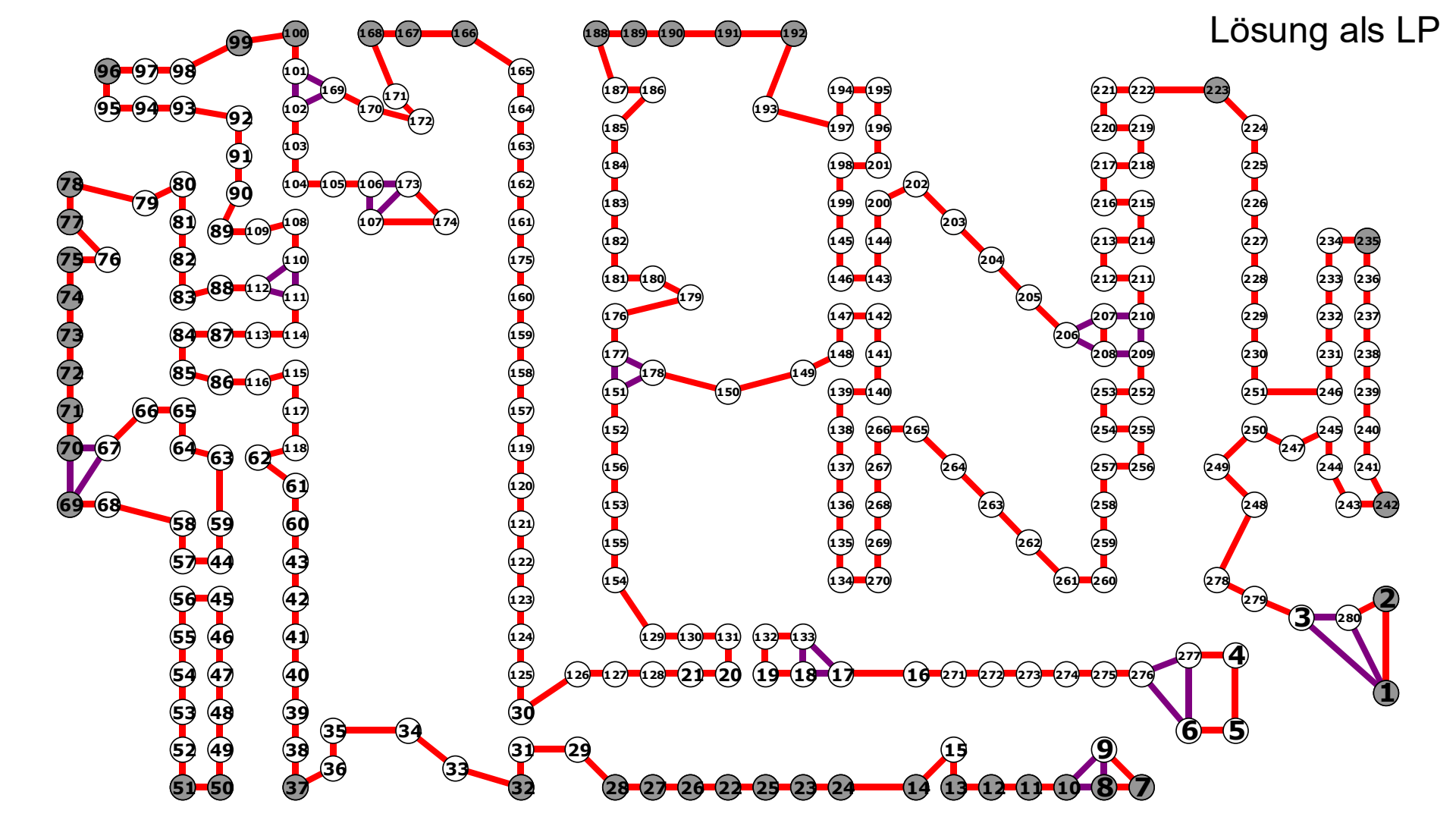

Lösung als LP

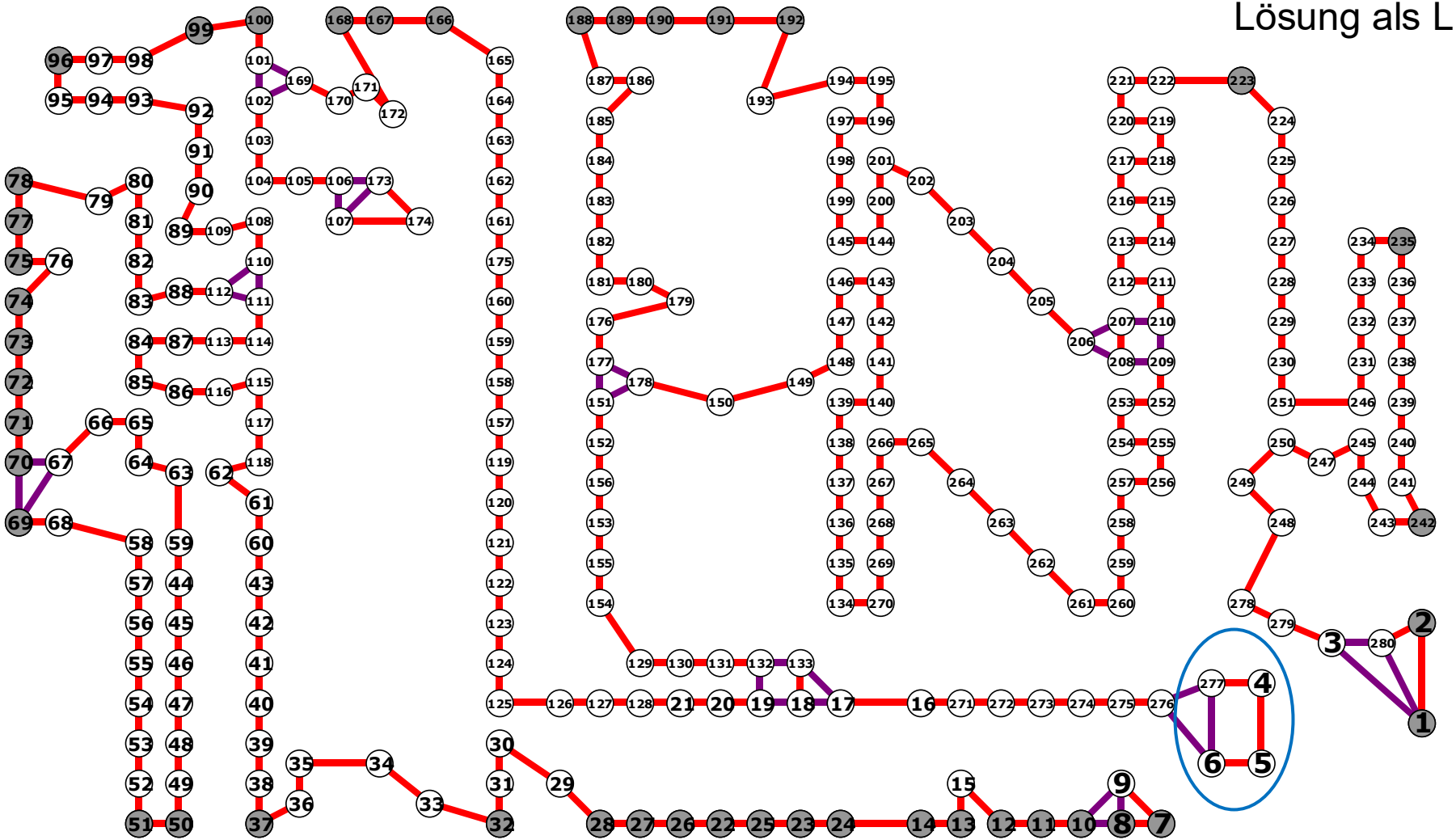

Lösung als LP

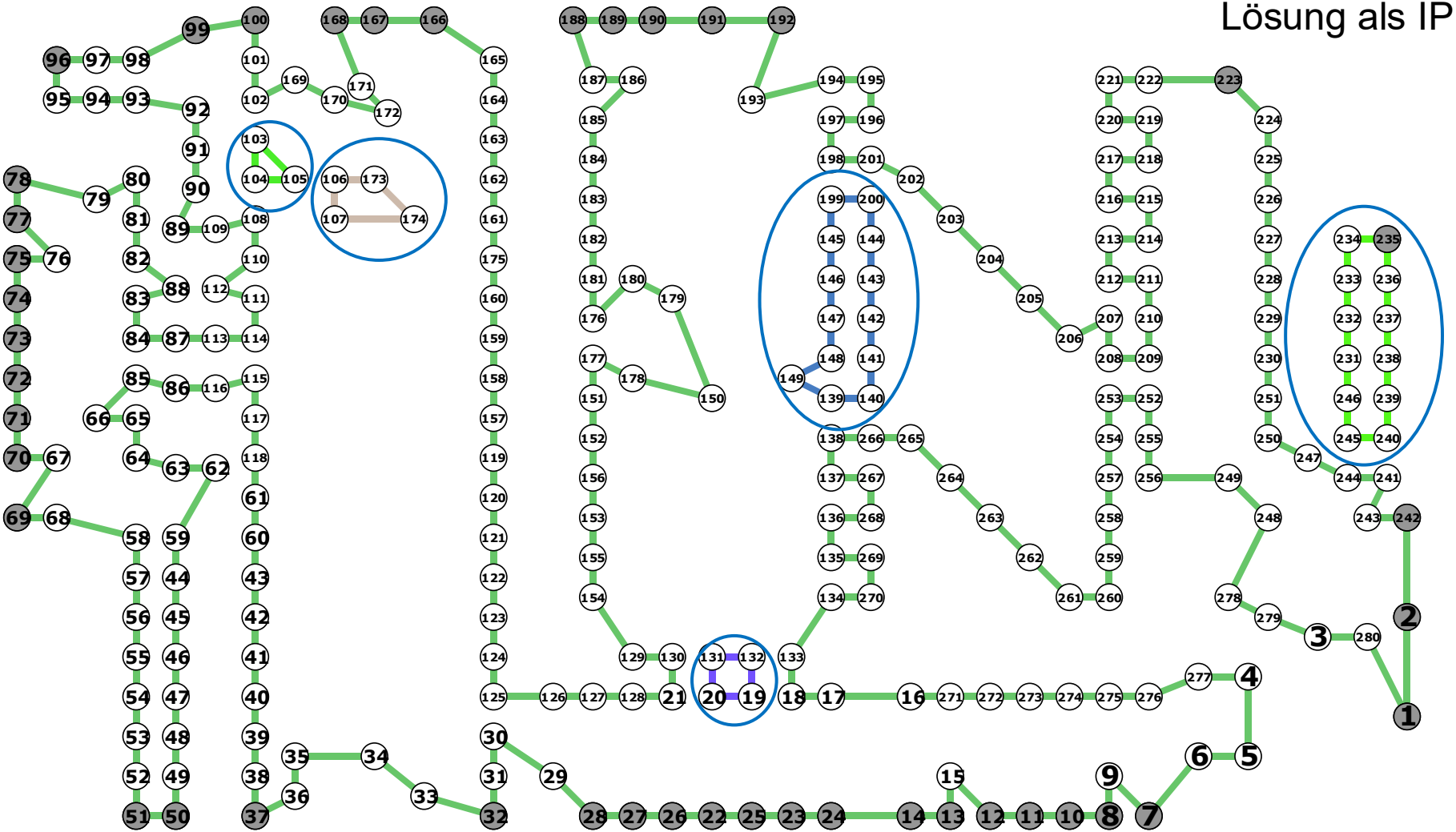

Lösung als LP

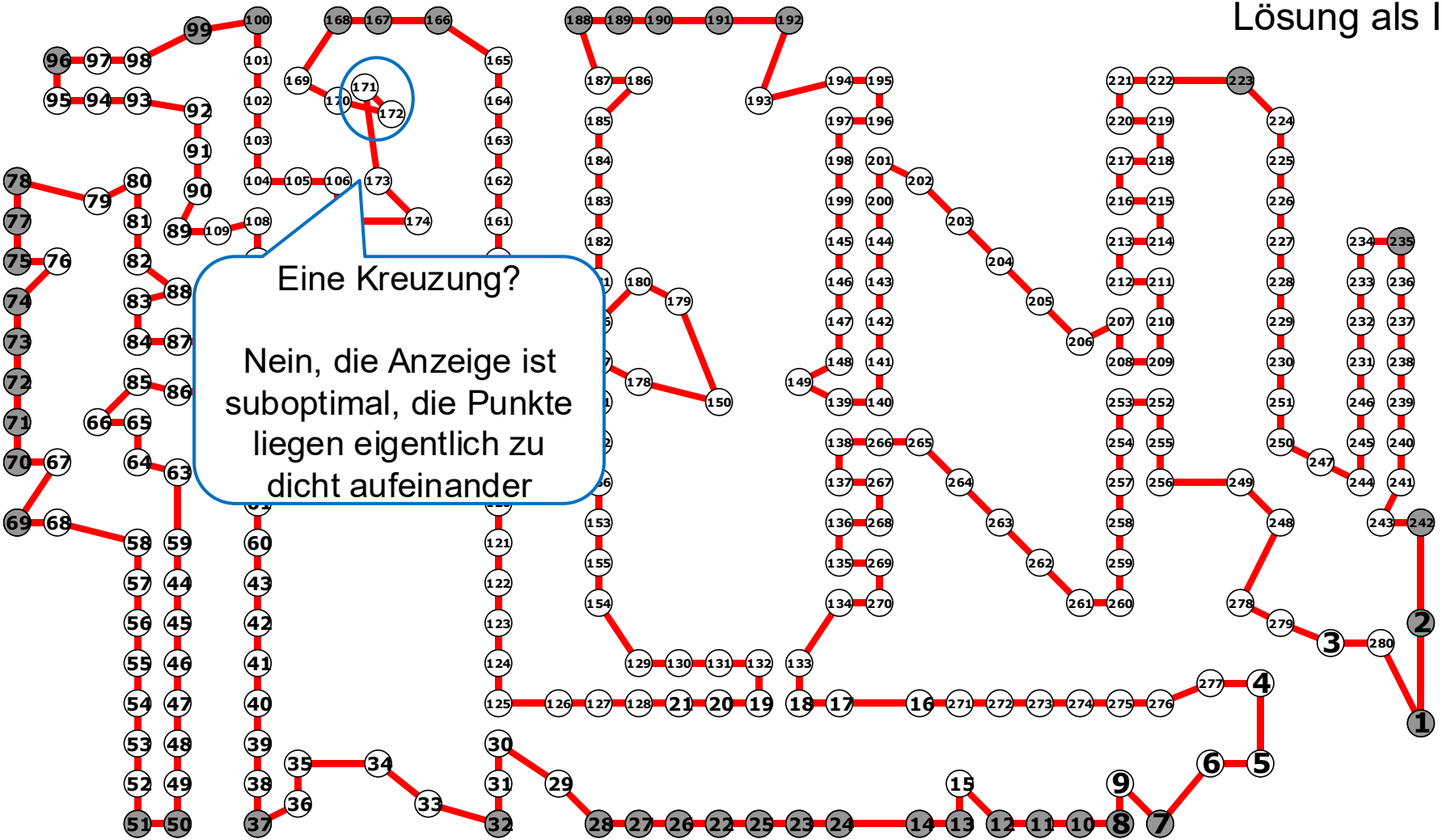

Lösung als LP


Lösung als LP


Lösung als LP

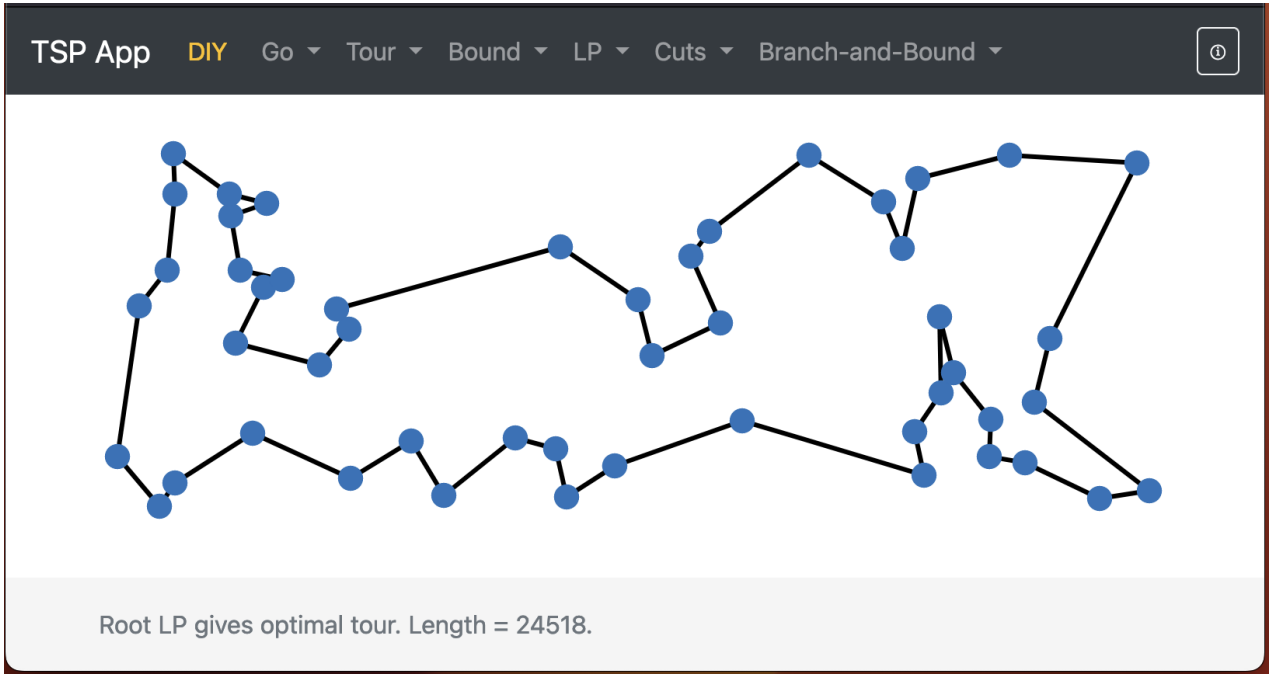

Lösung als LP


Lösung als IP


Lösung als IP
Eine Kreuzung?
Nein, die Anzeige ist suboptimal, die Punkte liegen eigentlich zu dicht aufeinander

# Interaktives Tool zum TSP

https://www.math.uwaterloo.ca/tsp/app/diy.html



# … und nächstes Mal …

## … geht es um Approximationen!

### Greedy$_k$ – Beispiel

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $z_i = p_i$ | 13 | 11 | 10 | 8 |

$Z = 30, k = 2$

$\bar{S}$ : Fixierte Menge

$\sum_{i \in \bar{S}} z_i$ : Gewicht der fixierten Menge

$Z - \sum_{i \in \bar{S}} z_i$ : Restkapazität

$G + \sum_{i \in \bar{S}} p_i$ : Wert der fixierten Menge + auffüllen

$G_k, S$ : Bisher bester Lösungswert und Menge

| $\bar{S}$ | $\sum_{i \in \bar{S}} z_i$ | $Z - \sum_{i \in \bar{S}} z_i$ | $G + \sum_{i \in \bar{S}} p_i$ | $G_k$ | $S$ |
|---|---|---|---|---|---|
| $\emptyset$ | 0 | 30 | 24 | 24 | {1,2} |
| {1} | 13 | 17 | 24 | 24 | {1,2} |
| {2} | 11 | 19 | 24 | 24 | {1,2} |



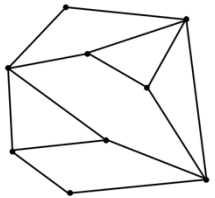
### Vertex Cover

Geht das noch besser?

**Gegeben**
- Graph $G = (V, E)$
- Zahl $k \in \mathbb{N}$

Das ist unser Vertex-Cover

**Frage**
- Existiert eine Menge $VC \subseteq V$ mit $|VC| \leq k$, sodass für jede Kante $\{u, v\} \in E$ gilt: $u \in VC$ oder $v \in VC$?



$|VC| = 6$

Als Optimierungsproblem: Suche die kleinste Zahl $k$.

(Kleinste) Menge an Knoten, die jede Kante abdeckt!

… aber erst am 17.06.2026!