



Technische  
Universität  
Braunschweig



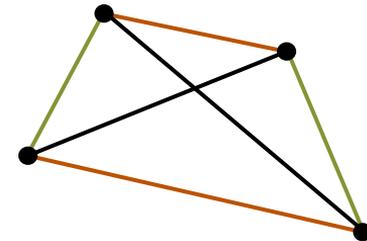
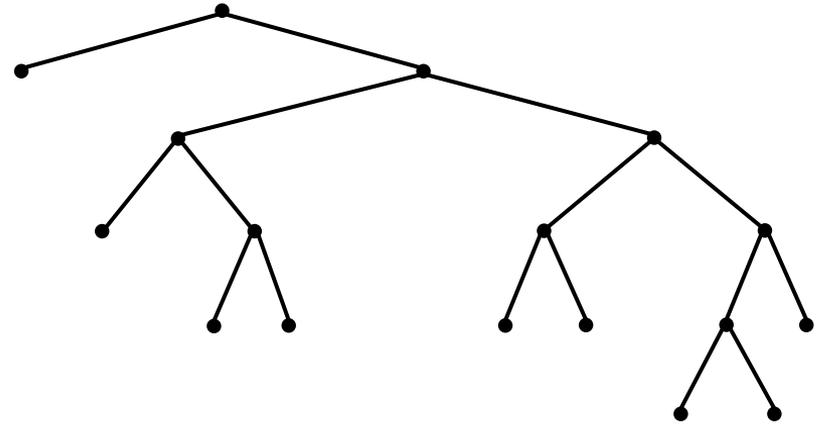
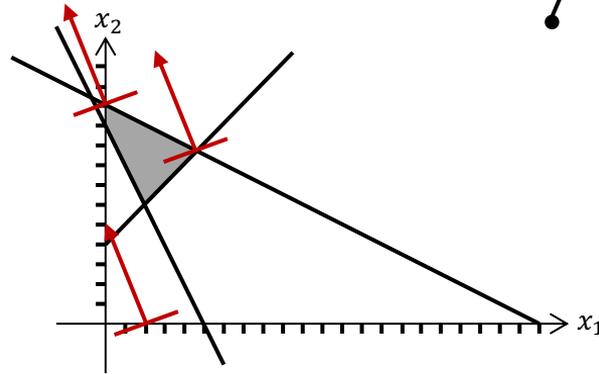
# Algorithmen und Datenstrukturen 2 – Übung #3

Ramin Kosfeld und Chek-Manh Loi

15.05.2024

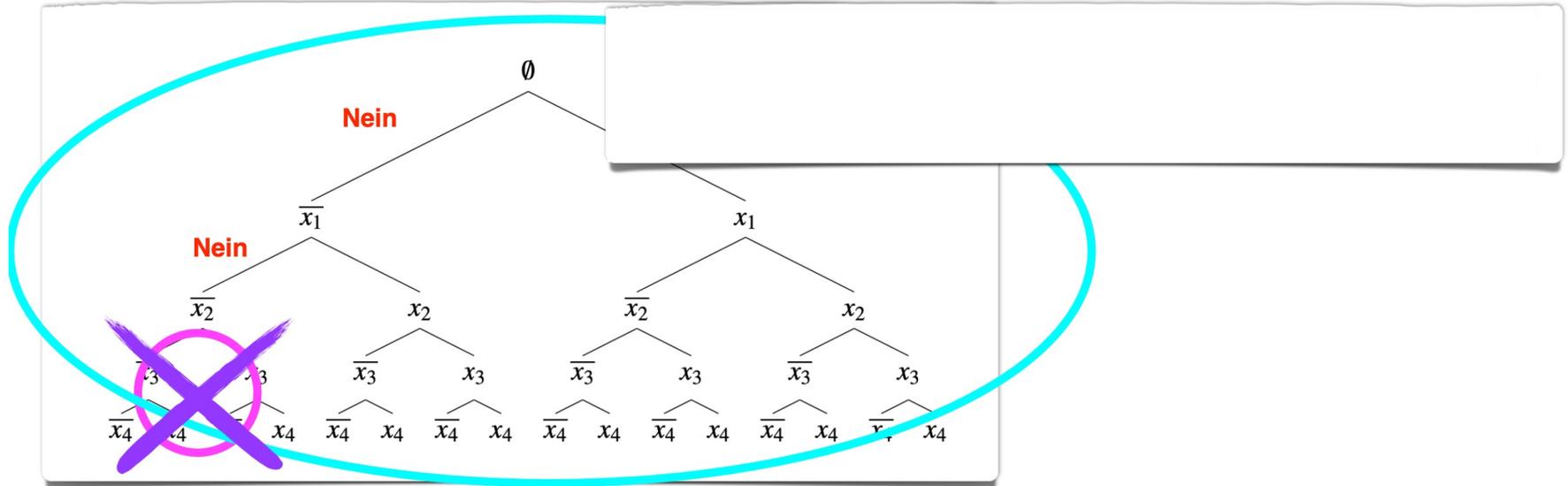
# Tagesordnung

- Recap: Branch-and-Bound für Knapsack
- Euklidisches TSP
  - Definition
  - Schranken
  - Branch-and-Bound
  - Linear Programming



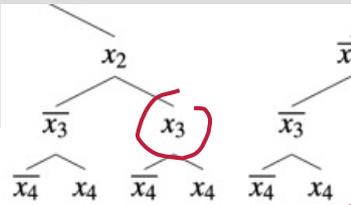
# Branch & Bound: Schematisches Bild aus der Vorlesung

Für Maximierungsproblem:





# Branch & Bound



Allgemein (an Baumknoten):

1. Test auf Zulässigkeit

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

2. Schranken berechnen

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

3. Update globale Schranke

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

4. Prüfe auf Pruning

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

5. Verzweigen falls nötig

$U \leq P \Rightarrow$  ignoriere Teilbaum

MAXIMUM  
KNAPSACK

## Herausforderungen

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

1. Betrachte Fall  $b_l := 0$

2. Betrachte Fall  $b_l := 1$

# 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

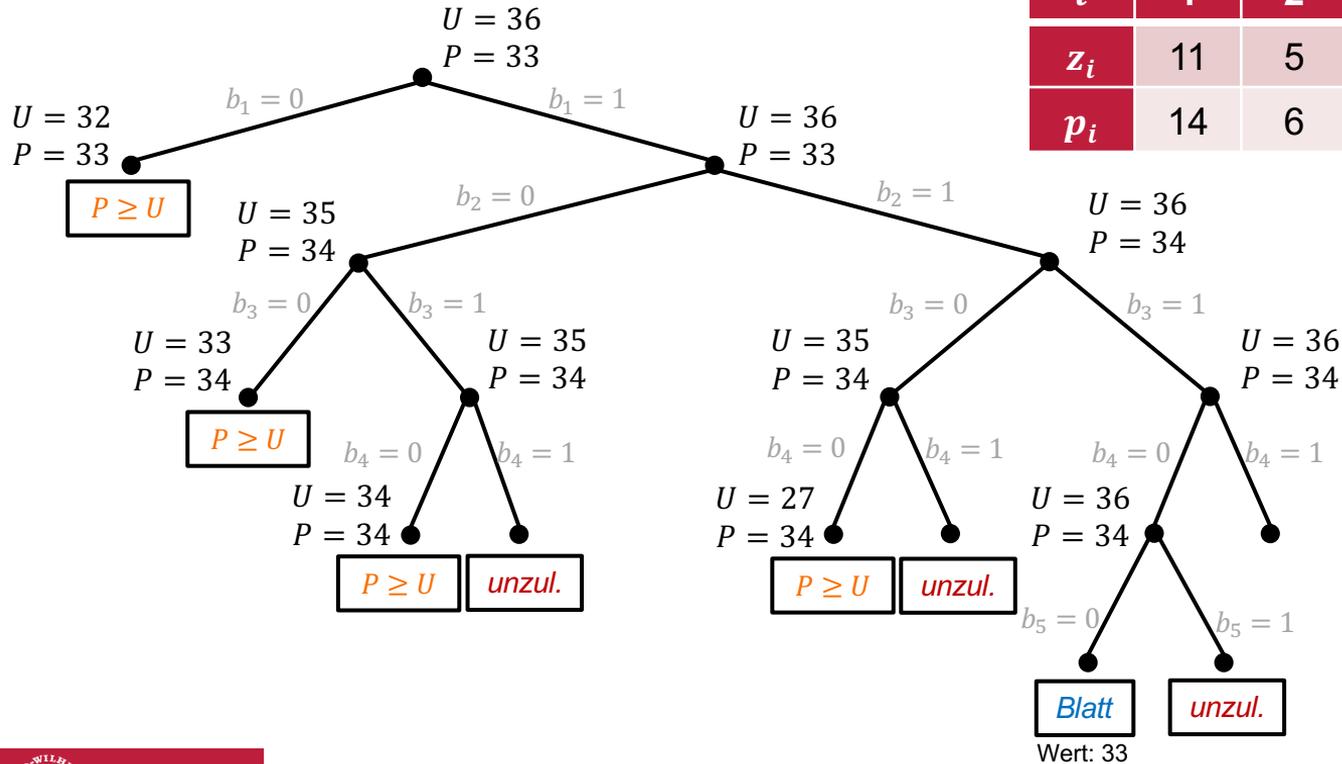
## Wofür berechnen wir Schranken an Baumknoten?

- Obere Schranke: Prüfen, ob wir diesen Knoten weiter betrachten müssen
- Untere Schranke: Erkennen von neuen, möglicherweise besseren Lösungen
- Innere Knoten des Suchbaums: Partielle Zuweisung der Variablen
- Blätter: Vollständige Zuweisungen der Variablen (also potentielle Lösungen)
- Theoretisch reicht es, untere Schranken an Blättern auszuwerten
- Das verzögert möglicherweise das Finden besserer Lösungen  $\Rightarrow$  größerer Baum

# Branch & Bound – Beispiel

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

$Z = 33$

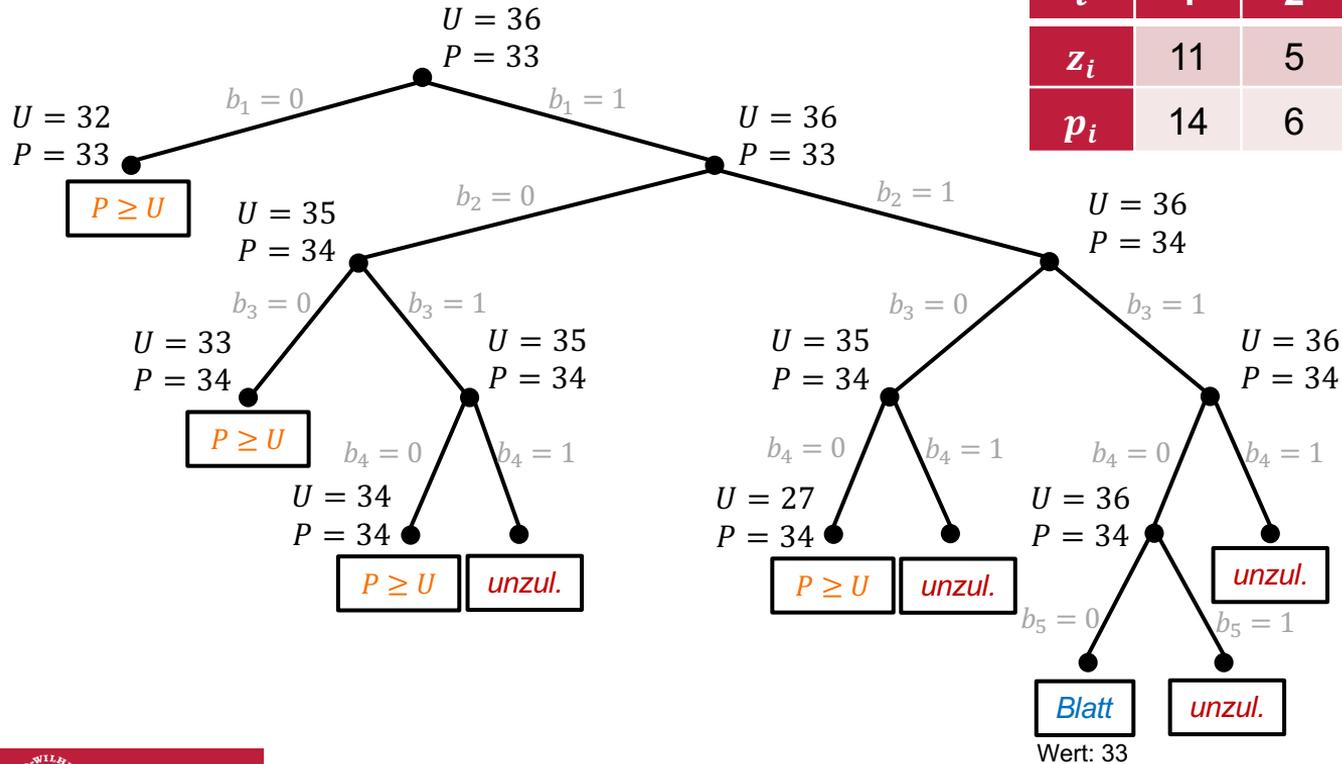


Menge	Wert
-------	------

# Branch & Bound – Beispiel

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

$Z = 33$



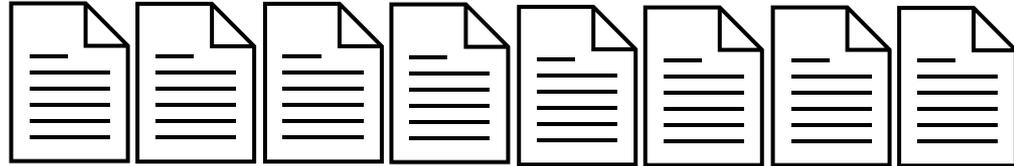
Menge	Wert
{1,2,3}	33
{1,3,5}	34

# 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!

**Achtung:**  
Jetzt ist euer  
Bauchgefühl gefragt!



$2^{100}$ : 100 mal verdoppeln und stapeln. Wie hoch ist der Papierstapel?

Dicke von Papier:  $\sim 0.1\text{mm}$

# $2^{100}$ Blätter gestapelt



6.371 km



384.400 km

Distanz Sonne-Erde: 150 Mio. km

Distanz Sonne-Pluto: 6 Mrd. km

Distanz zu Alpha Centauri: 4,2 Lichtjahre

Distanz zum Andromeda-Nebel: 2,5 Mio. Lichtjahre

Distanz zum Virgohaufen: 60 Mio. Lichtjahre

Wie hoch ist nun der Stapel mit  $2^{100}$  Blättern?

Die Antwort: Etwa 14 Mrd. Lichtjahre  $\approx 10^{23}$  km

# $2^{100}$ Blätter gestapelt



6.371 km

384

Achtung: Das ist der Worst Case!  
Branch & Bound erreicht in der Praxis oft eine deutlich  
bessere Laufzeit, da durch Pruning viel weniger Knoten  
betrachtet werden müssen!

Mio. km  
d. km  
Lichtjahre  
Mio. Lichtjahre  
io. Lichtjahre

# Euclidean Traveling Salesman Problem

**Gegeben:**

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

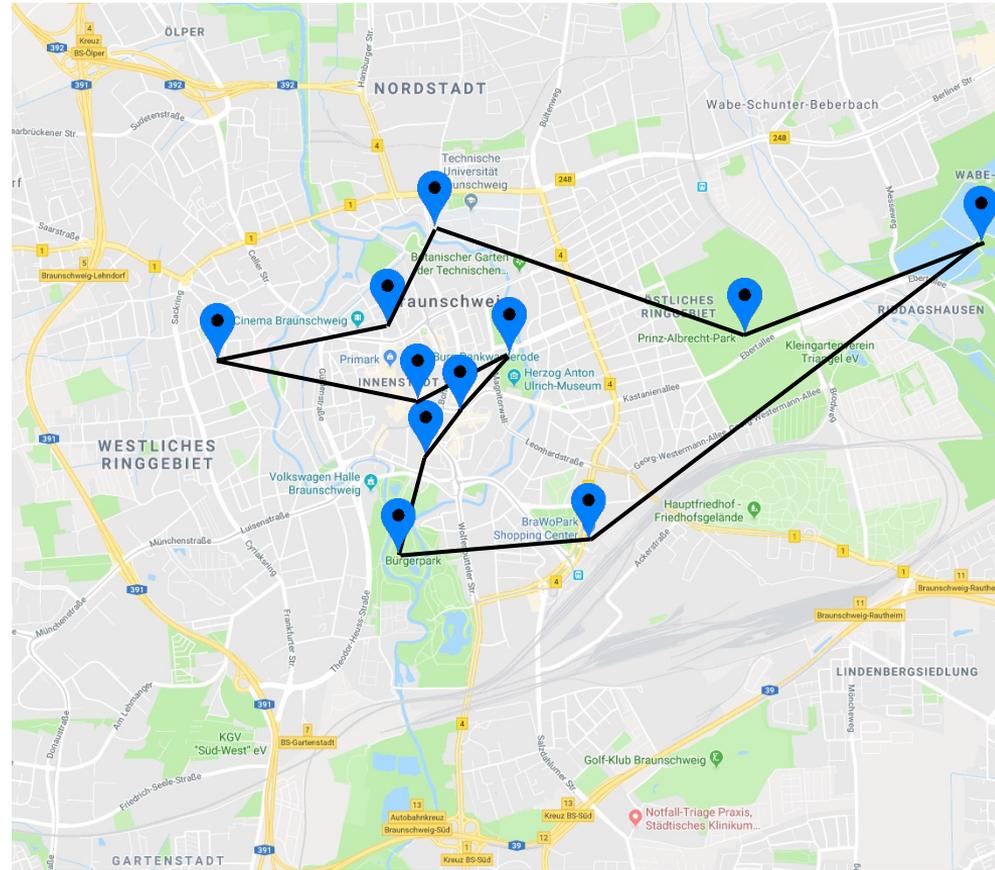
**Gesucht:**

Permutation  $\pi$  mit

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

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:**

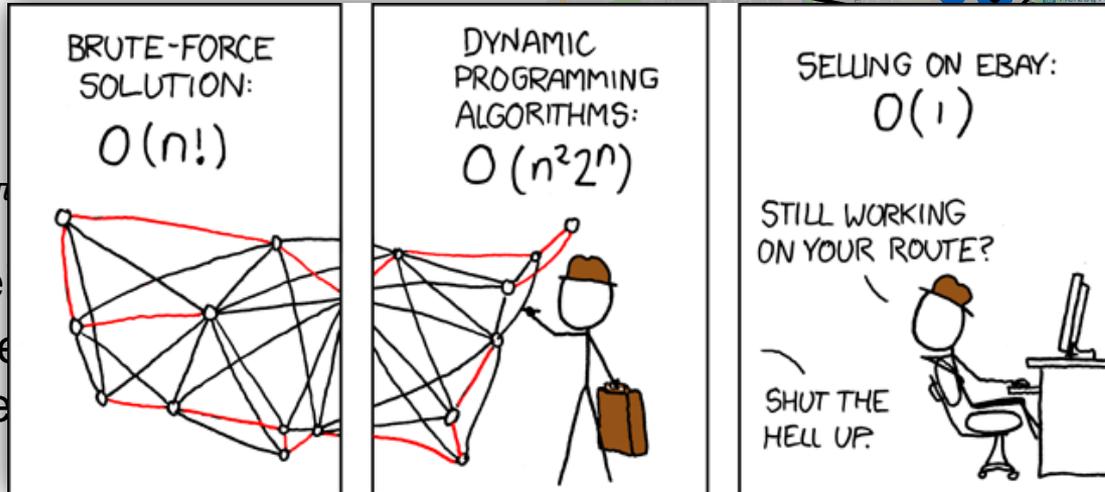
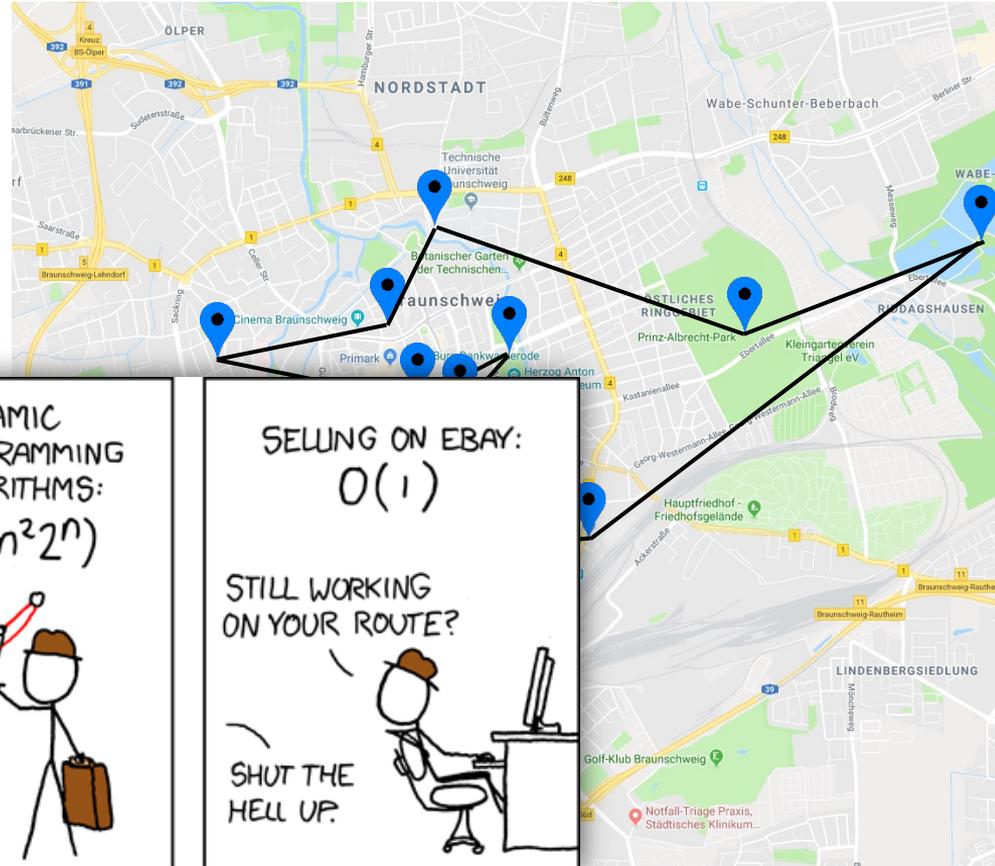
Permutation  $\pi$

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

minimal. Dabei

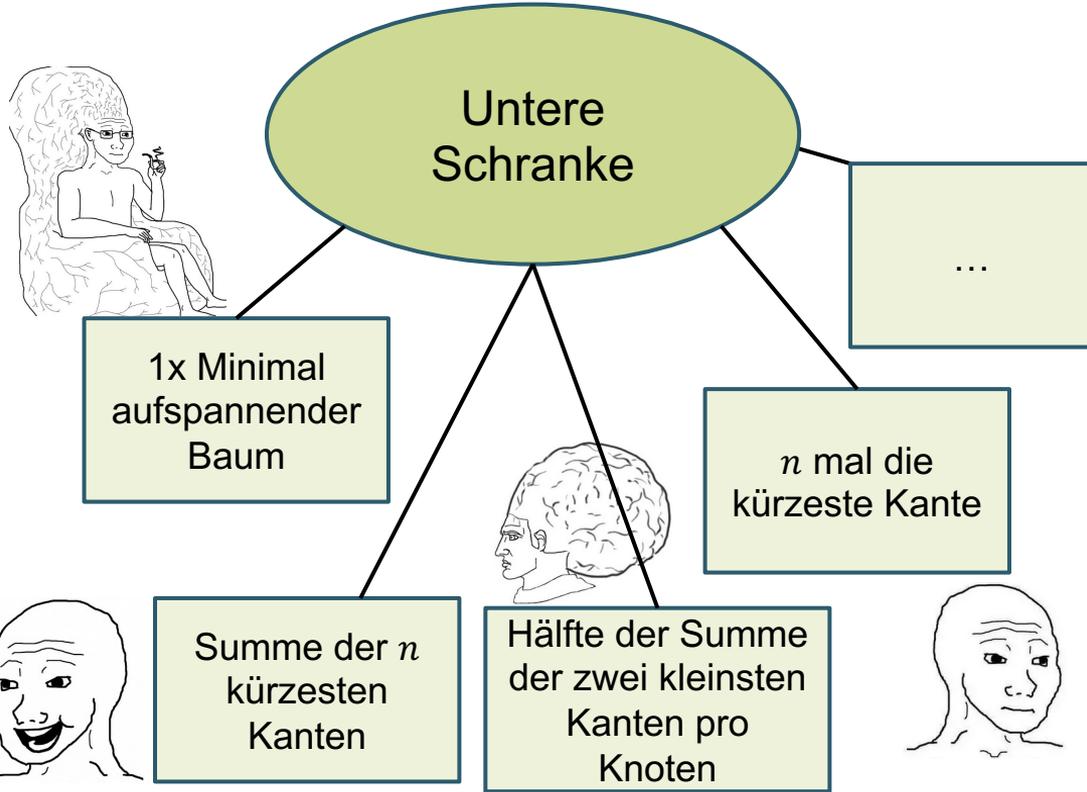
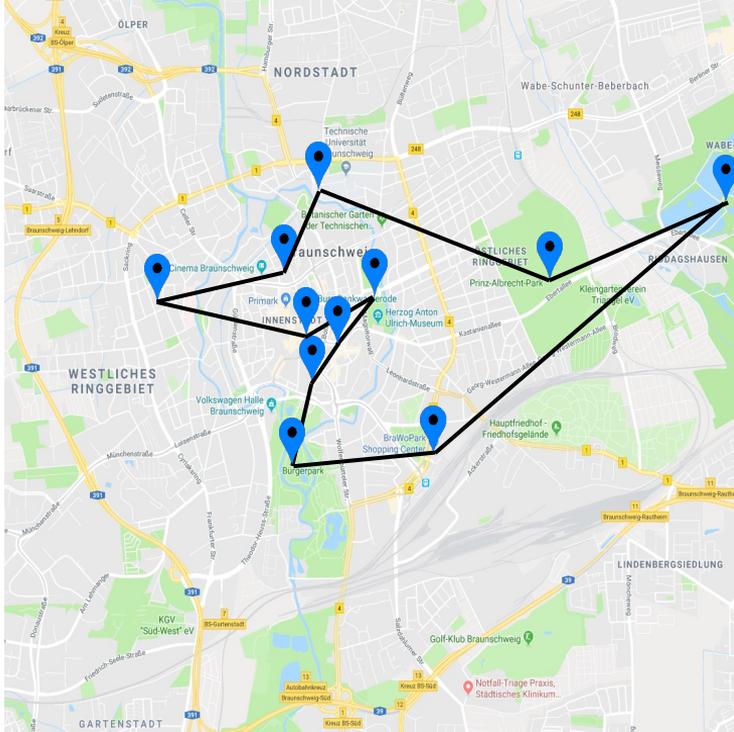
Finde also eine

die alle Punkte



[https://imgs.xkcd.com/comics/travelling\\_salesman\\_problem.png](https://imgs.xkcd.com/comics/travelling_salesman_problem.png)

# ETSP – Schranken



# ETSP – Schranken



2x Minimal  
aufspannender  
Baum

Jede *zulässige*  
Lösung

Greedy

2-OPT



1x Minimal  
aufspannender  
Baum

...

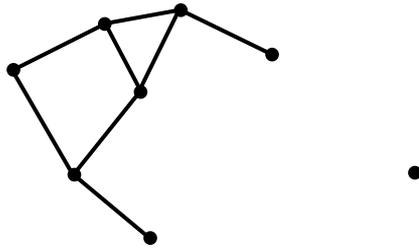
$n$  mal die  
kürzeste Kante

Summe der  $n$   
kürzesten  
Kanten

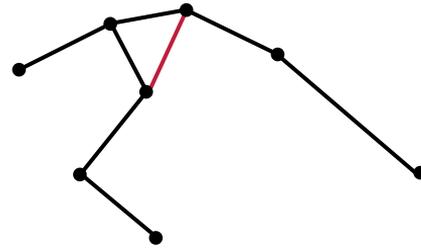
Hälfte der Summe  
der zwei kleinsten  
Kanten pro  
Knoten

# ETSP – Untere Schranken

Summe der  $n$   
kürzesten Kanten



1x Minimal  
aufspannender  
Baum  
(+ kürzeste Kante)



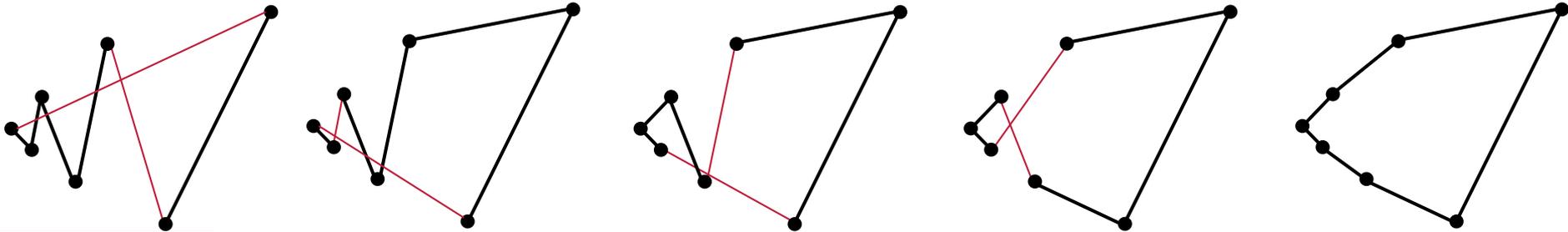
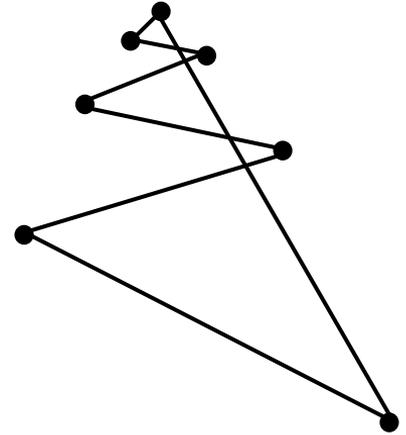
# ETSP – Obere Schranken

Greedy  
Nearest-Neighbor

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

2-OPT

In einer beliebigen Tour, tausche zwei existierende Kanten mit zwei nicht-existierenden Kanten, um eine bessere Tour zu erhalten.



# 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

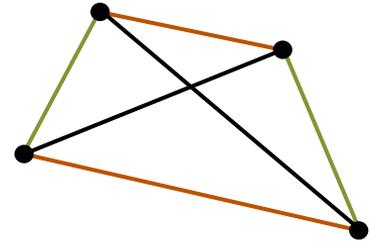
- Also alle Permutationen testen?
- Vorgänger von Knoten raten?
- Wie kann ich darüber Schranken finden?

# 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.



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 – Lineares Ungleichungssystem

Etwas anderer Ansatz:

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

s.t.

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

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

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

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

Das ist ein  
*Integer Linear Program*

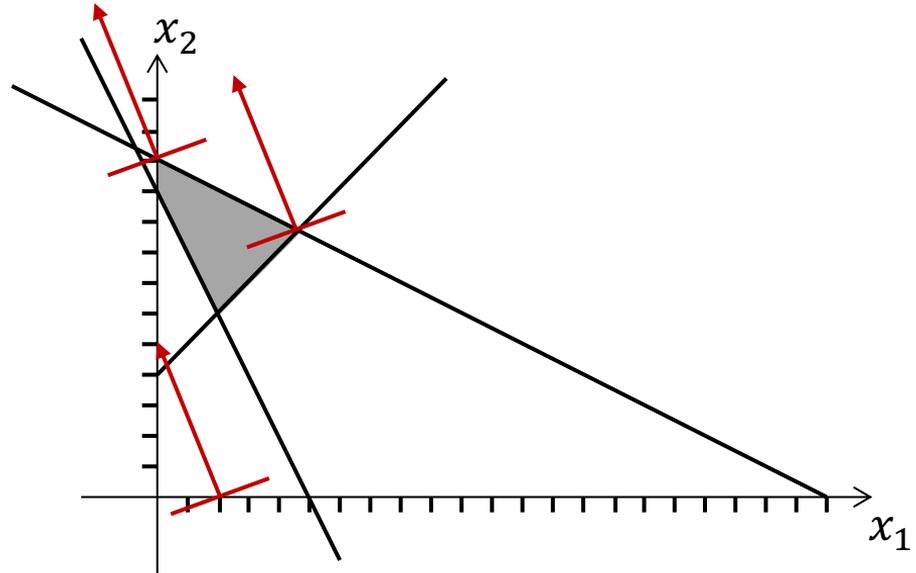
# Linear/Integer Programming

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

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

Beispiel:

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



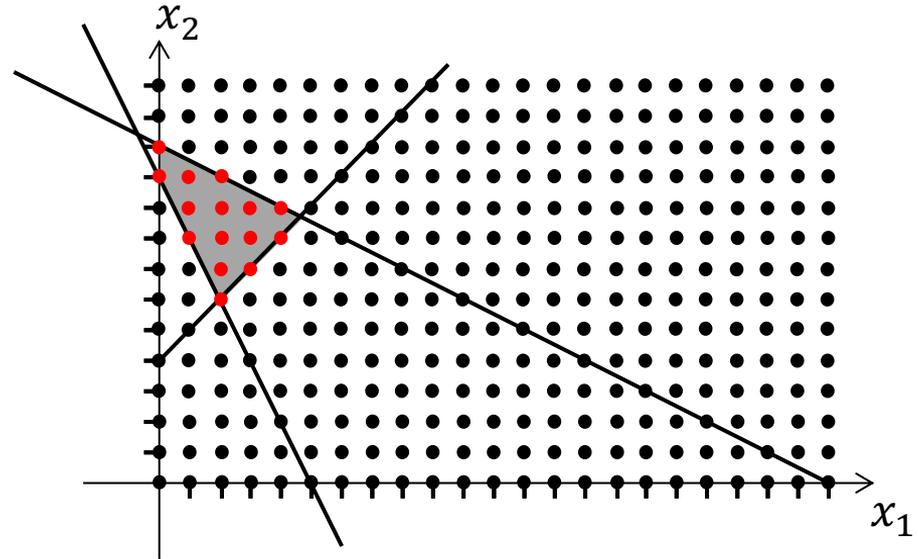
# Linear/Integer Programming

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

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

Beispiel:

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

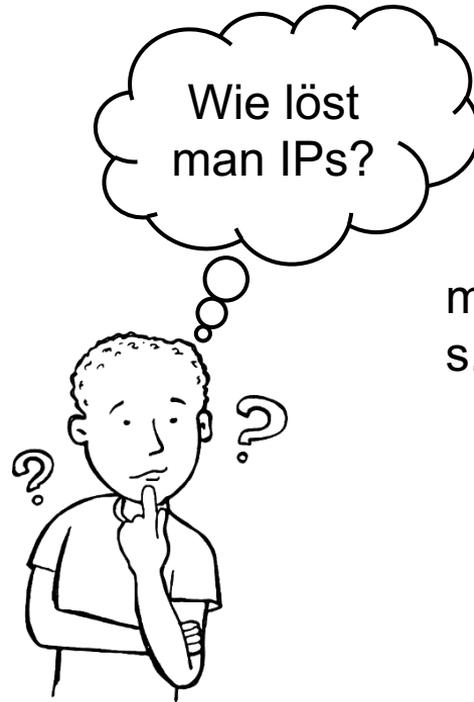


# Linear/Integer Programming

**LP**

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

Effizient lösbar

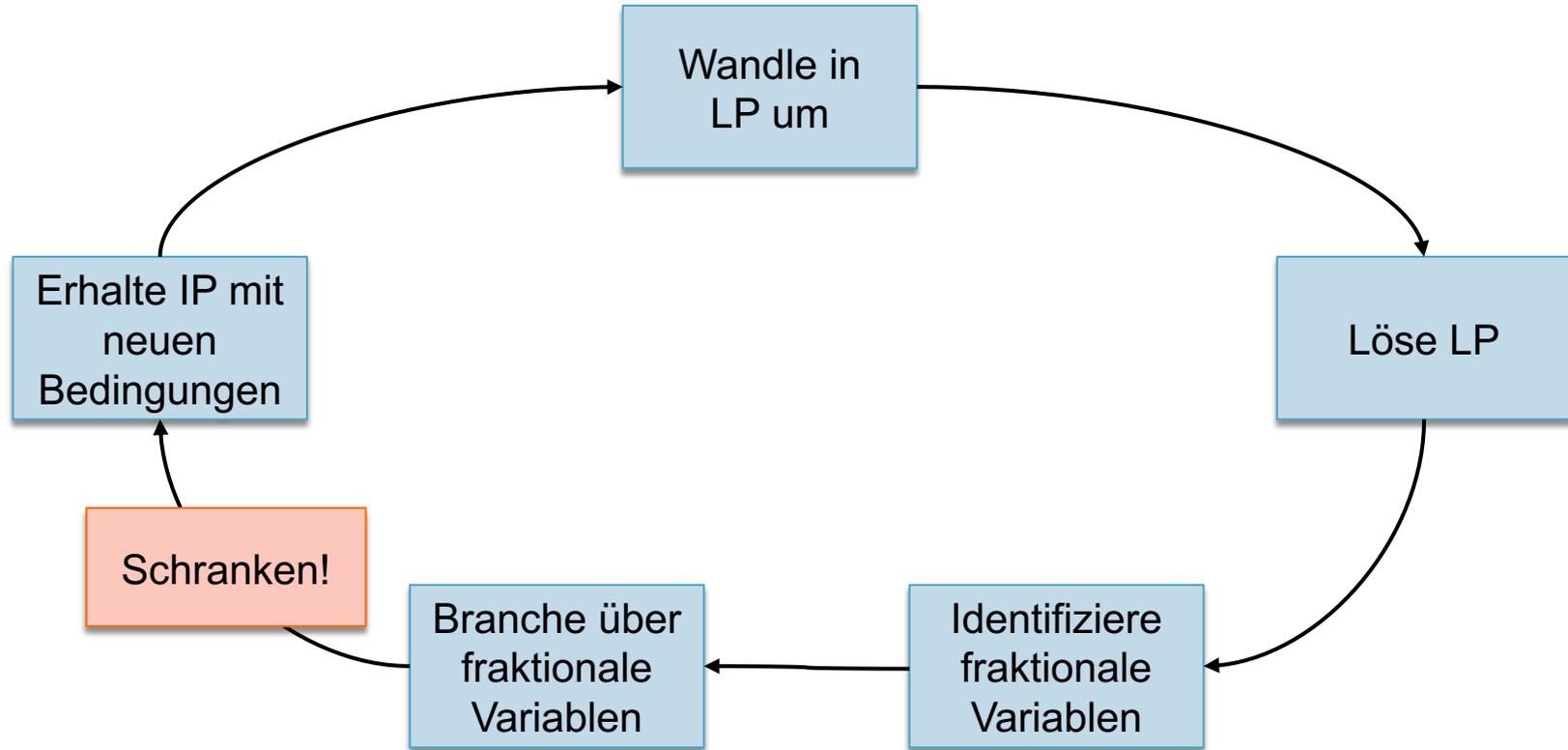


**IP**

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

Vermutlich nicht  
effizient lösbar

# Lösen von IPs (Vereinfachte Darstellung)



# Solver für Linear/Integer Programming



Microsoft Excel



# ETSP

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

s.t.

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

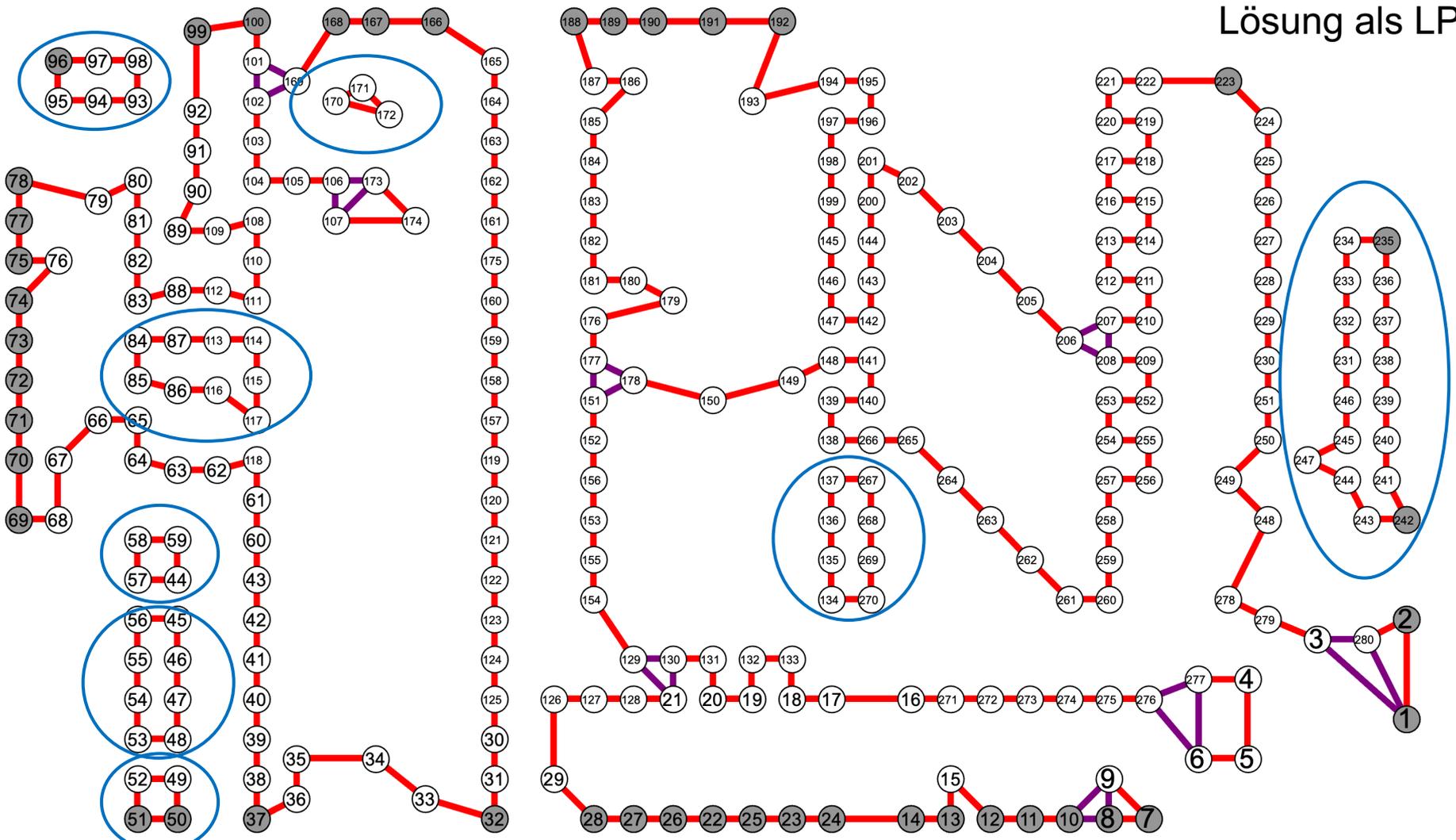
$$\sum_{e \in \delta(S)} x_e \geq 2 \quad \text{für alle } \emptyset \neq S \subsetneq P$$

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



Exponentiell viele...  
Füge sie während des  
Lösungsprozesses hinzu!

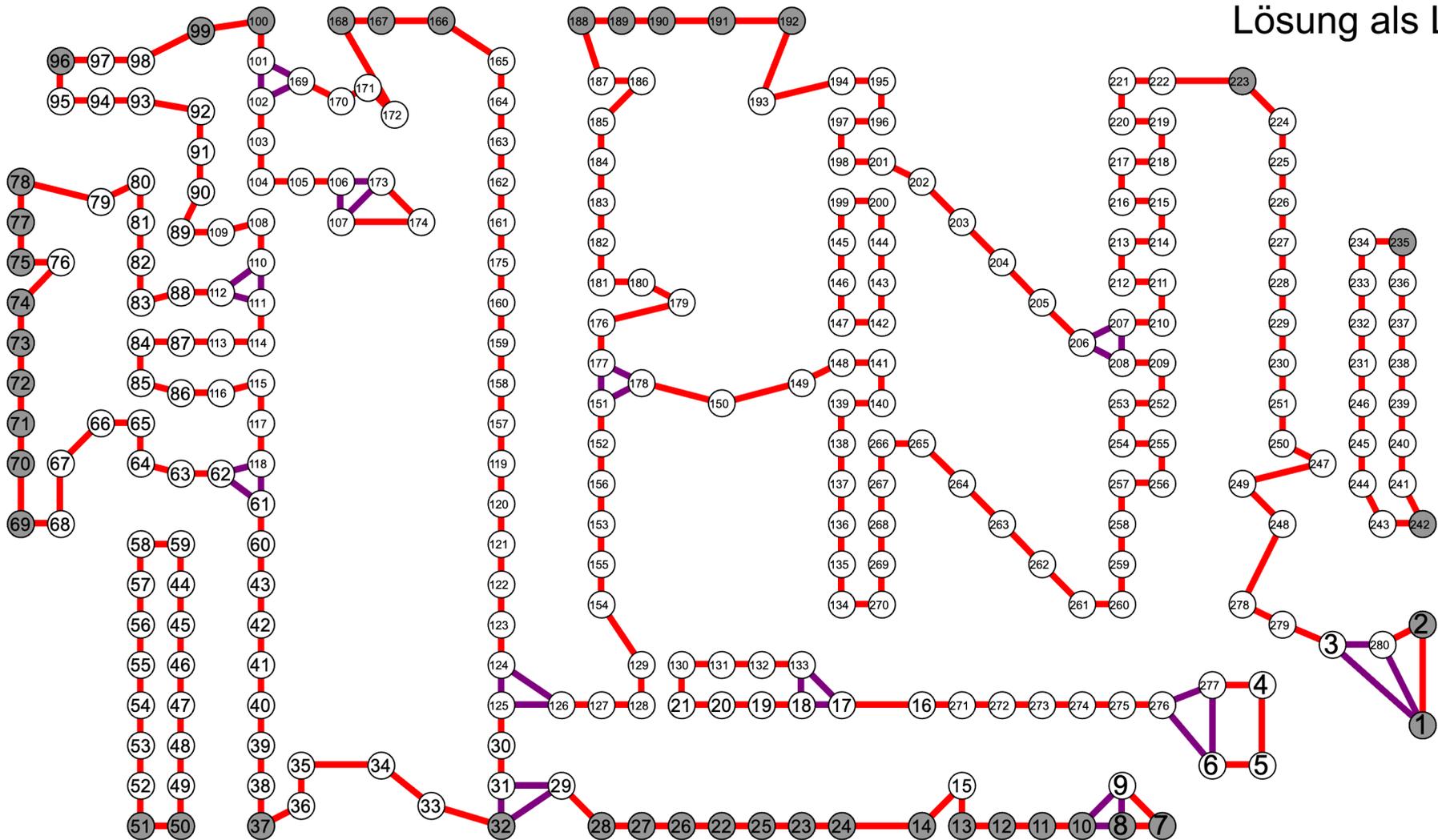
# Lösung als LP





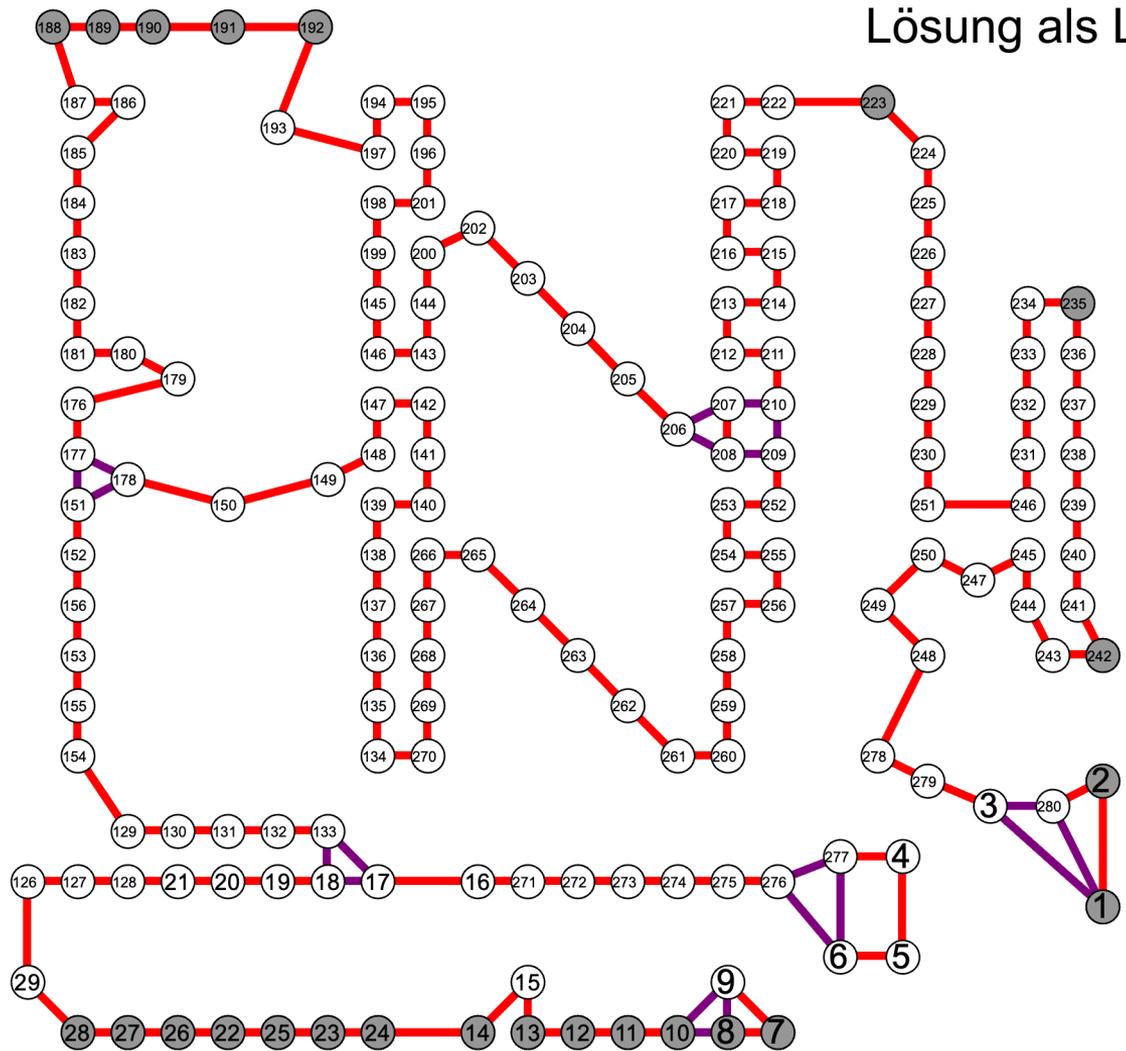
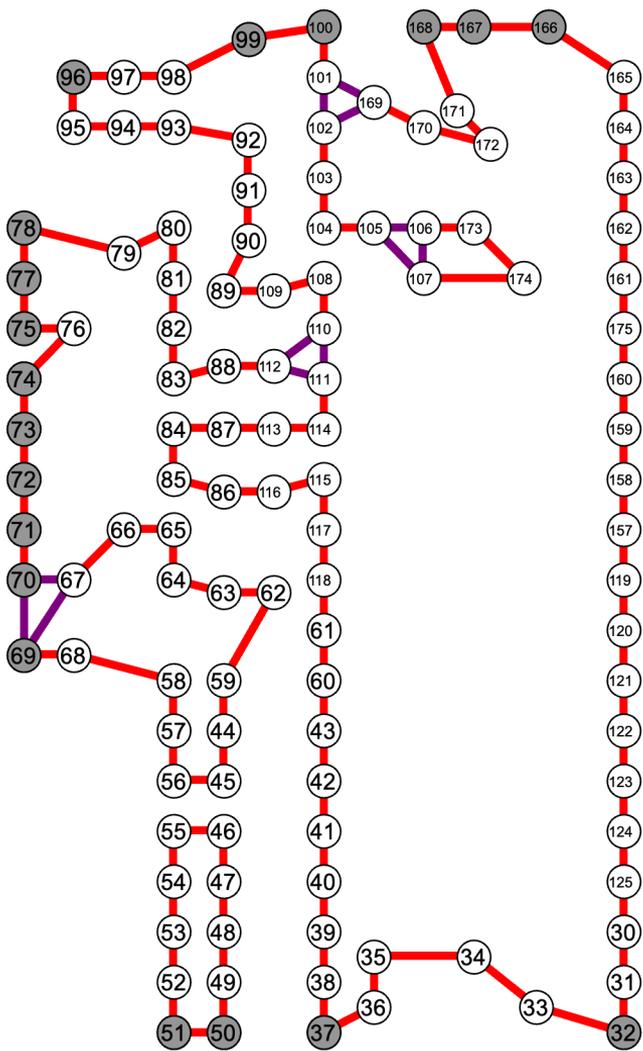


# Lösung als LP





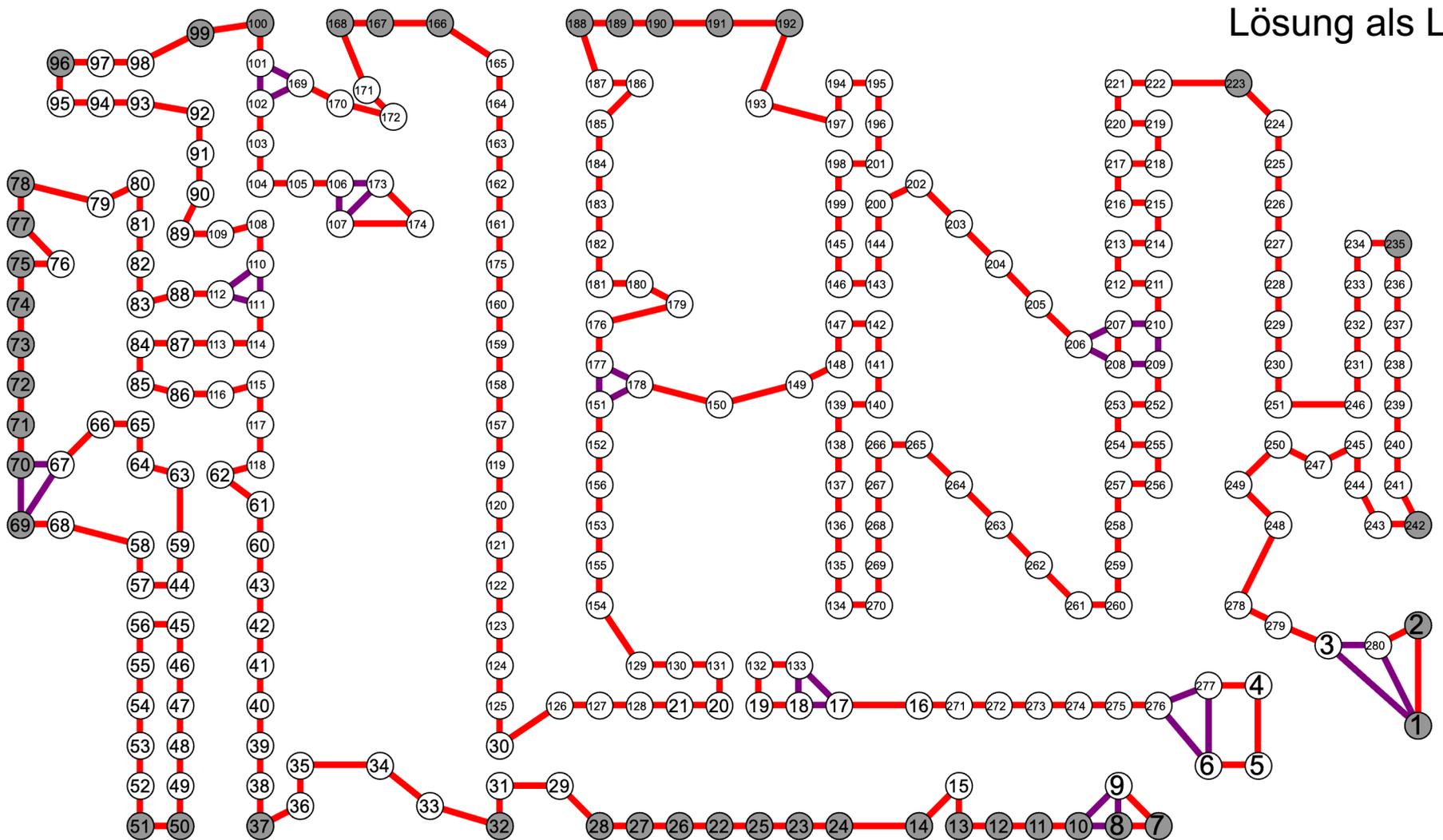
# Lösung als LP





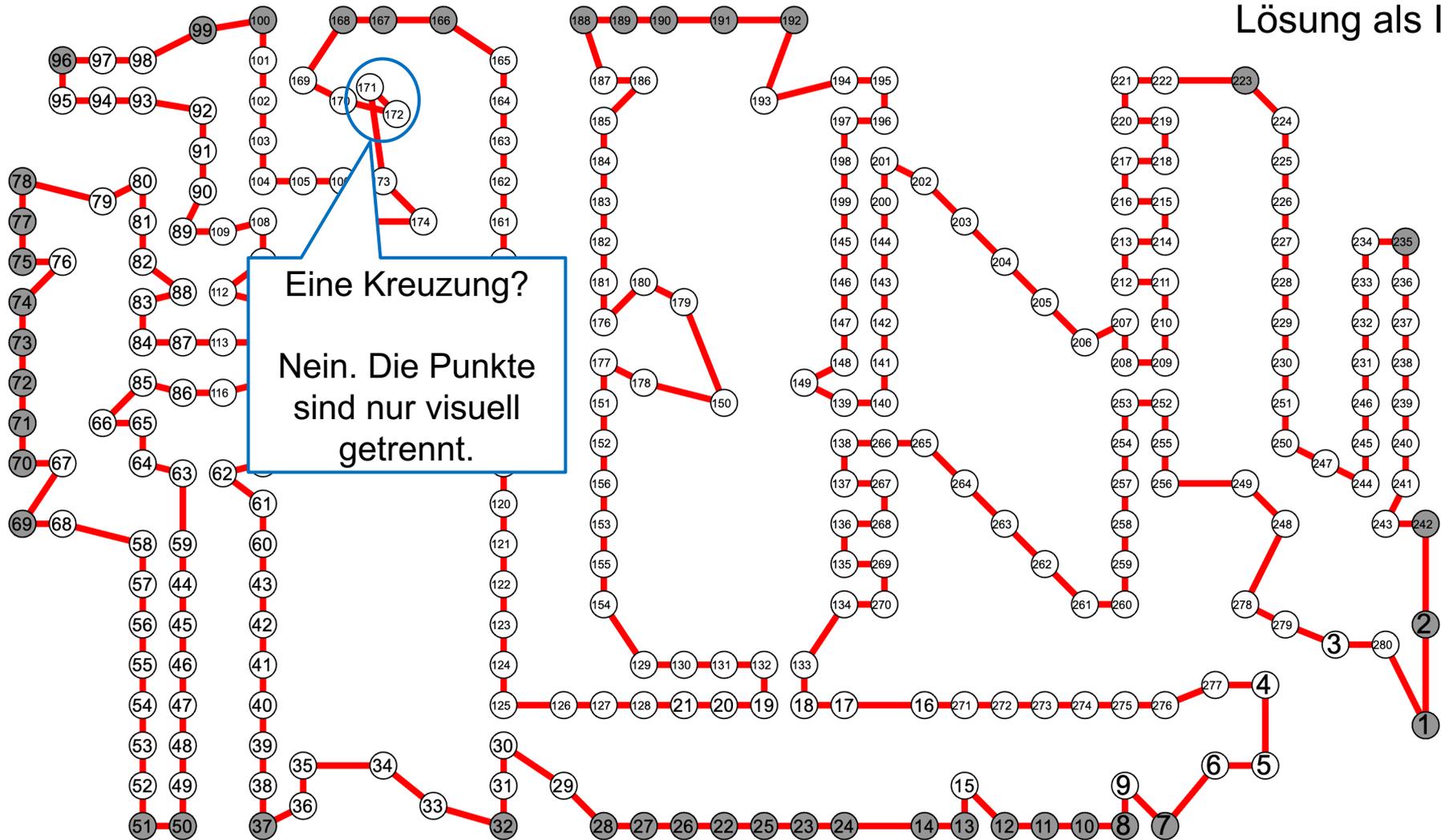


# Lösung als LP



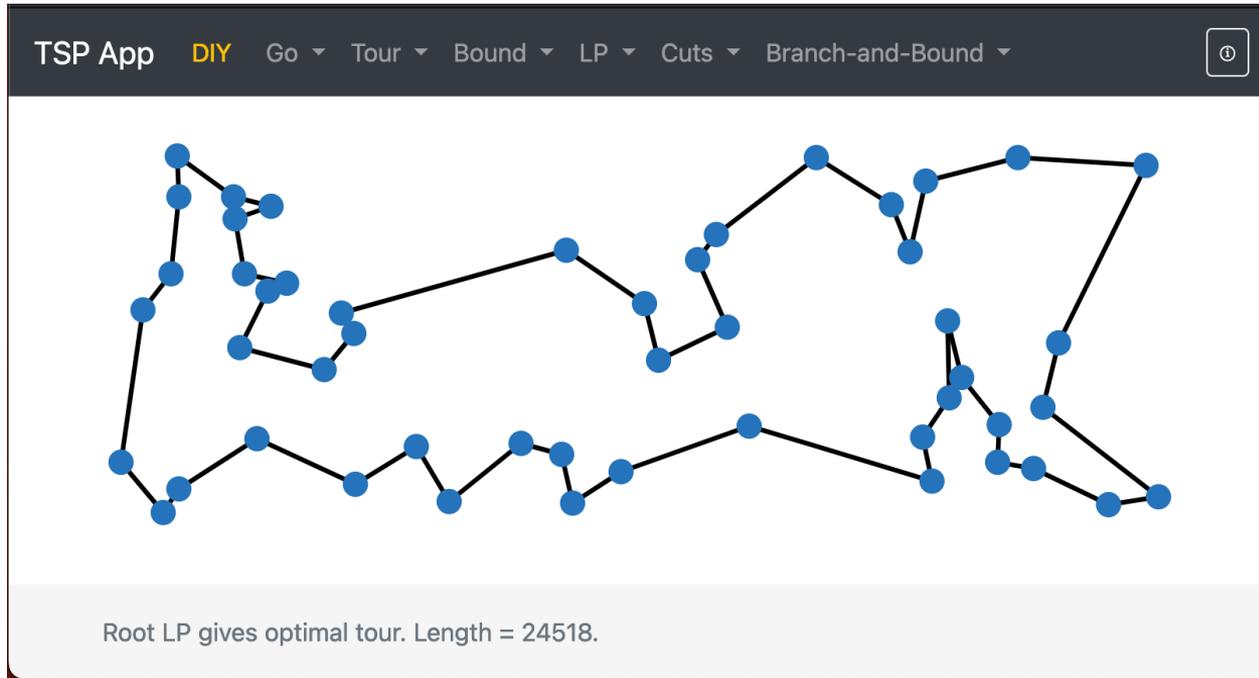






# Interaktives Tool zum TSP

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



# Interesse an mehr?

## Mathematische Methoden der Algorithmik

Semester Winter 2022/2023 ▾

Programmes Business Information Systems Master, Computer and Communication Systems Engineering Master, Computer Science Master

IBR Group ALG (Prof. Fekete)

Type Lecture & Exercise

Lecturers



**Dr. Phillip Keldenich**  
Wissenschaftlicher Mitarbeiter  
✉ keldenich@ibr.cs.tu-bs.de  
☎ +49 531 3913112  
📍 Room 318



**Dr. Ahmad Moradi**  
Wissenschaftlicher Mitarbeiter  
✉ moradi@ibr.cs.tu-bs.de  
☎ +49 531 3913180  
📍 Room 315

Master-Veranstaltung

## Algorithmikpraktikum: Solving NP-hard Problems in Practice

Semester Summer 2023 ▾

Programme Computer Science Bachelor

IBR Group ALG (Prof. Fekete)

Type Lecture & Exercise

Lecturer



**Prof. Dr. Sándor P. Fekete**  
Abteilungsleiter  
✉ s.fekete@tu-bs.de  
☎ +49 531 3913111  
📍 Room 335

Assistant



**Dr. Dominik Krupke**  
Wissenschaftlicher Mitarbeiter  
✉ krupke@ibr.cs.tu-bs.de  
☎ +49 531 3913116  
📍 Room 332

Bachelor/Master-Veranstaltung

# ... und nächstes Mal ...

## ... lösen wir Sudoku (vielleicht)

### Propagation: Beispiele bei Sudoku

#### Beispiele für Regeln (wir haben zu jeder Zelle eine Menge zulässiger Werte):

- Wenn eine Zeile, eine Spalte oder ein Block nur eine Zelle mit möglichem Wert  $x$  hat, kann man alle anderen Werte aus dieser Zelle entfernen.

8	1,2,5	1,2,5,9
4	2,5	2,5,6,9
7	1,5	1,3,5,9

→

8	1,2,5	1,2,5,9
4	2,5	6
7	1,5	3

→

8	1,2,5	9
4	2,5	6
7	1,5	3

### Resolution

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge x_4$$

Sieht man der Formel irgendwie anders an, dass auch  $x_1 \vee x_3$  gelten muss?

- Der Konflikt entstand durch die zwei Klauseln  $(x_1 \vee x_2 \vee x_3)$  und  $(x_1 \vee \bar{x}_2 \vee x_3)$
- Zwei Fälle:  $x_2 \rightarrow x_1 \vee x_3$ ,  $\bar{x}_2 \rightarrow x_1 \vee x_3$ ; insgesamt also  $x_1 \vee x_3$ !
- Allgemein: Resolution
- Zwei Klauseln  $(x_i \vee \ell_1 \vee \dots \vee \ell_p)$ ,  $(\bar{x}_i \vee \mu_1 \vee \dots \vee \mu_k)$  implizieren immer, dass auch  $(x_i \vee \ell_1 \vee \dots \vee \ell_p) \circ (\bar{x}_i \vee \mu_1 \vee \dots \vee \mu_k) = (\ell_1 \vee \dots \vee \ell_p \vee \mu_1 \vee \dots \vee \mu_k)$  gelten muss
- Dies nennt man (binäre) Resolution.
- Weitere Beispiele:

$$(x_1 \vee x_2 \vee x_3) \circ (\bar{x}_1 \vee x_2 \vee x_3) = (x_2 \vee x_3)$$
$$(x_1 \vee x_2 \vee x_3) \circ (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) = (x_2 \vee x_3 \vee \bar{x}_2 \vee \bar{x}_3)$$

Wie kann uns das helfen?



Am 05.06.2024!