lineare Programmierung

- Viele Probleme sind durch **lineare Gleichungssysteme** charakterisiert
  $\Rightarrow$ lineare Programmiermethoden
- Der **Lösungsraum** ist häufig auf **ganze Zahlen** oder gar **natürliche Zahlen** eingeschränkt!
- Das Auffinden einer Lösung wird schwieriger!
- **Konsequenz: Betrachtung von linearer Programmierung mit Ganzzahllösungen.**
- **Problem: Effizientes Auffinden der gesuchten Lösung(en).**

Beispiel: Investitionsplanung

Problem: Der Betrag von € 14000 soll möglichst gewinnbringend investiert werden.

#	Investitionssumme	Wert
(1)	€ 5000	€ 8000
(2)	€ 7000	€ 11000
(3)	€ 4000	€ 6000
(4)	€ 3000	€ 4000

Maximiere $8x_1 + 11x_2 + 6x_3 + 4x_4$ mit den **Einschränkungen** $5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$ und $x_i \in \{0, 1\}$ für alle $i \in \{1, 2, 3, 4\}$.

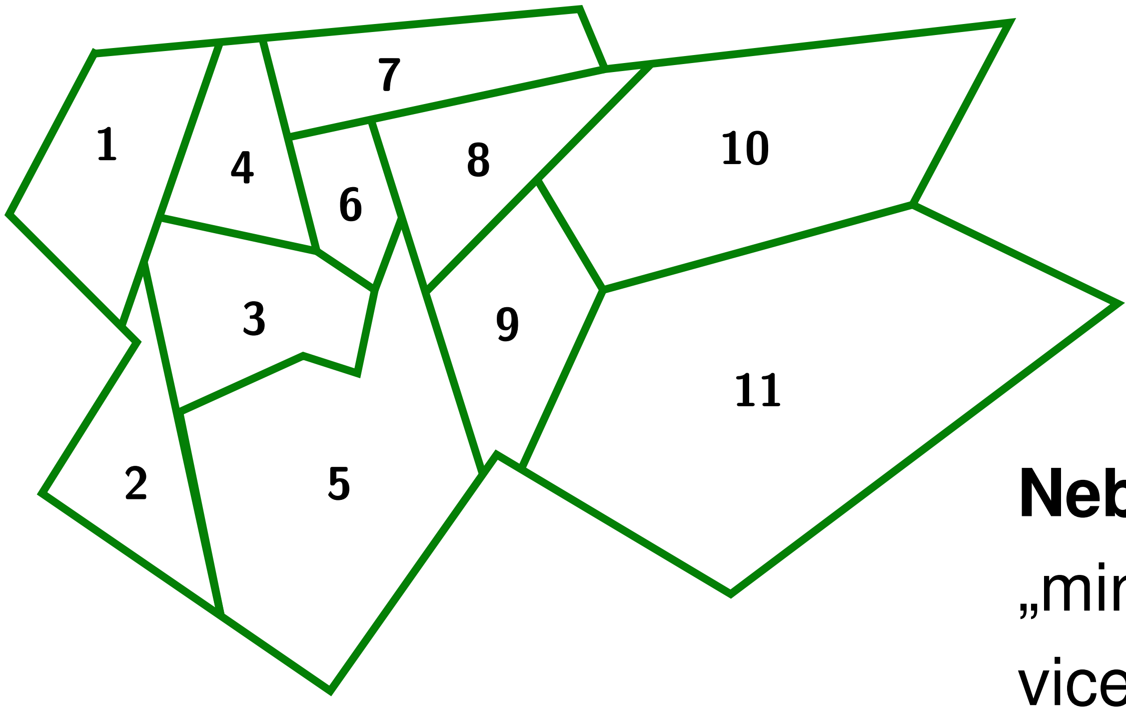
Lösung

- Standardverfahren liefern als Lösung: $x_1 = x_2 = 1$, $x_3 = 0,5$ und $x_4 = 0$ mit dem erreichten Wert € 22000.
- Keine ganzzahlige Lösung!
- Abrunden ergibt nur einen Wert von € 19000.
- Optimale ganzzahlige Lösung ist € 21000. ($x_2 = x_3 = x_4 = 1$ und $x_1 = 0$)
- Weitere Einschränkungen formalisierbar:
  - höchstens zwei Investitionen: $\sum x_i \leq 2$
  - wird in (2) investiert, so auch in (4): $x_2 - x_4 \leq 0$
  - wird in (1) investiert, dann nicht in (3): $x_1 - x_3 \leq 1$

allgemeines Rucksackproblem

Gegeben:	Menge von Paaren $(l_i, d_i) \in I\!N \times I\!N, 1 \leq i \leq m$ sowie **Kapazität** $L \in I\!N$ des Rucksacks. Hierbei ist l_i die **Größe** und d_i der **Wert** des i-ten Objekts.
Gesucht:	Lösung des Optimierungsproblems 1. $w = \sum_{i=1}^{m} d_i x_i$ ist maximal. 2. Nebenbedingung: $\sum_{i=1}^{m} l_i x_i \leq L$.
Antwort:	Vektor $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in I\!N^m$

Servicepoints



Minimiere

$x_1+x_2+x_3+x_4+$
$x_5+x_6+x_7+x_8+$
$x_9+x_{10}+x_{11}.$

Nebenbedingungen:
„mindestens ein Servicepoint in der Nachbarschaft“

$x_1+x_2+x_3+x_4 \geq 1 \qquad x_1+x_2+x_3+x_5 \geq 1$

$x_1+x_2+x_3+x_4+x_5+x_6 \geq 1$ usw.

Instanz des **set coverability problem**

Travelling Salesperson Problem (TSP)

Gegeben:	Menge von Orten (oder Knoten) $V = \{1, 2, \ldots, n\}$und eine $(n \times n)$-Entfernungsmatrix $D = (d_{i,k}) \in I\!N^{n \times n}$
Gesucht:	Eine Rundreise (Hamiltonkreis) K durch mindestens (exakt) alle n Orte von V mit minimaler Gesamtlänge.
Antwort:	Wegbeschreibung (Folge von Knoten)

Minimiere $\sum_{i=1}^{n} \sum_{j=1}^{i-1} d_{i,j} x_{i,j}$ mit

Nebenbedingung $\sum_{i \in S} \sum_{j \notin S} x_{i,j} \geq 2$ für alle $S \subset V$

TSP als lin. Programmierproblem

- sehr große Anzahl von Nebenbedingungen

# Städte	# Nebenbedingungen
20	524288
300	1018517988167243043134 2228442046890805257341 9683296812531807022467 7190649881668353091698 688

- Dennoch ist dies derzeit für Instanzen zwischen 100 und 1000 Knoten der beste bekannte Ansatz!!!

Lösbarkeit des TSP

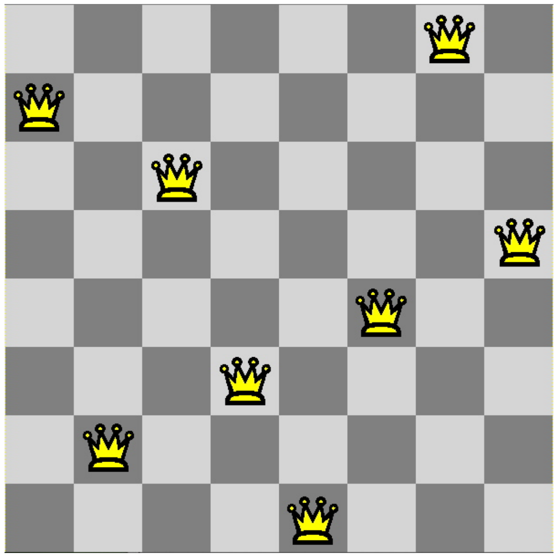
- mit dynamischer Programmierung:
  - Teilwege berechnen,
  - Zwischenergebnisse in Tabelle ablegen,
  - Kombination mit *branch and bound*
- konkretes Beispiel im Skript

Exkurs: reines **Backtracking** z.B. zur Lösung des 8-Königinnen-Problems.

8-Königinnen-Problem

Gegeben:	Ein Schachbrett (8×8 Felder), eine beliebige Anzahl von Königinnen
Gesucht:	Stellungen, so daß sich keine zwei Königinnen bedrohen.
Antwort:	Anzahl solcher Stellungen

Es können niemals mehr als 8 sein $\Rightarrow$ 8 Königinnen-Problem



IP vs. LR

Integer-Programmierung (IP)

`Minimiere`	cx
`Nebenbedingungen`	$Ax = b$
	$x \geq 0$ `und` x `ist ganzzahlig`

Lineare-Programmierung (LR) *engl. linear relaxation*

`Minimiere`	cx
`Nebenbedingungen`	$Ax = b$
	$x \geq 0$

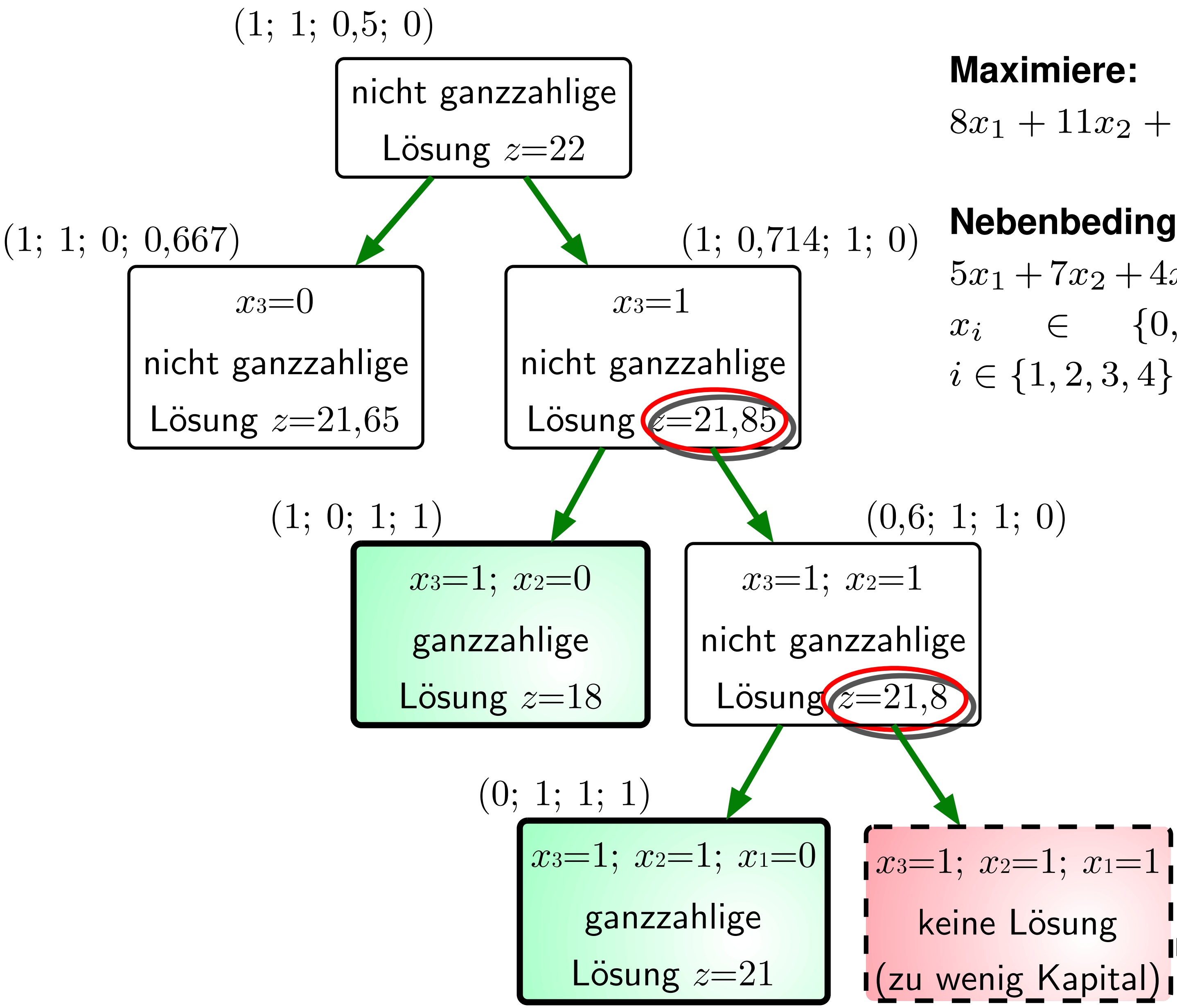
(analog für Maximierungsprobleme)

IP vs. LR (2)

Da (LR) weniger Nebenbedingungen hat als (IP) gilt:

- Ist (IP) eine Minimierung/Maximierung, so ist der optimale Wert für (LR) $\leq/\geq$ dem für (IP).
- Gibt es für (LR) keine optimale Lösung, so gibt es auch für (IP) keine.
- Ist die Lösung von (LR) ganzzahlig, so ist (IP) lösbar mit genau dieser optimalen Lösung.
- Auf-/abrunden der optimalen Lösung für (LR) liefert einen Wert $\leq/\geq$ dem Optimum für (IP) bei einem Minimierungs-/Maximierungsproblem.

Investitionen (branch and bound)



Maximiere:

$8x_1 + 11x_2 + 6x_3 + 4x_4$

Nebenbedingungen:

$5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$

$x_i \in \{0,1\}$ für alle $i \in \{1,2,3,4\}$.

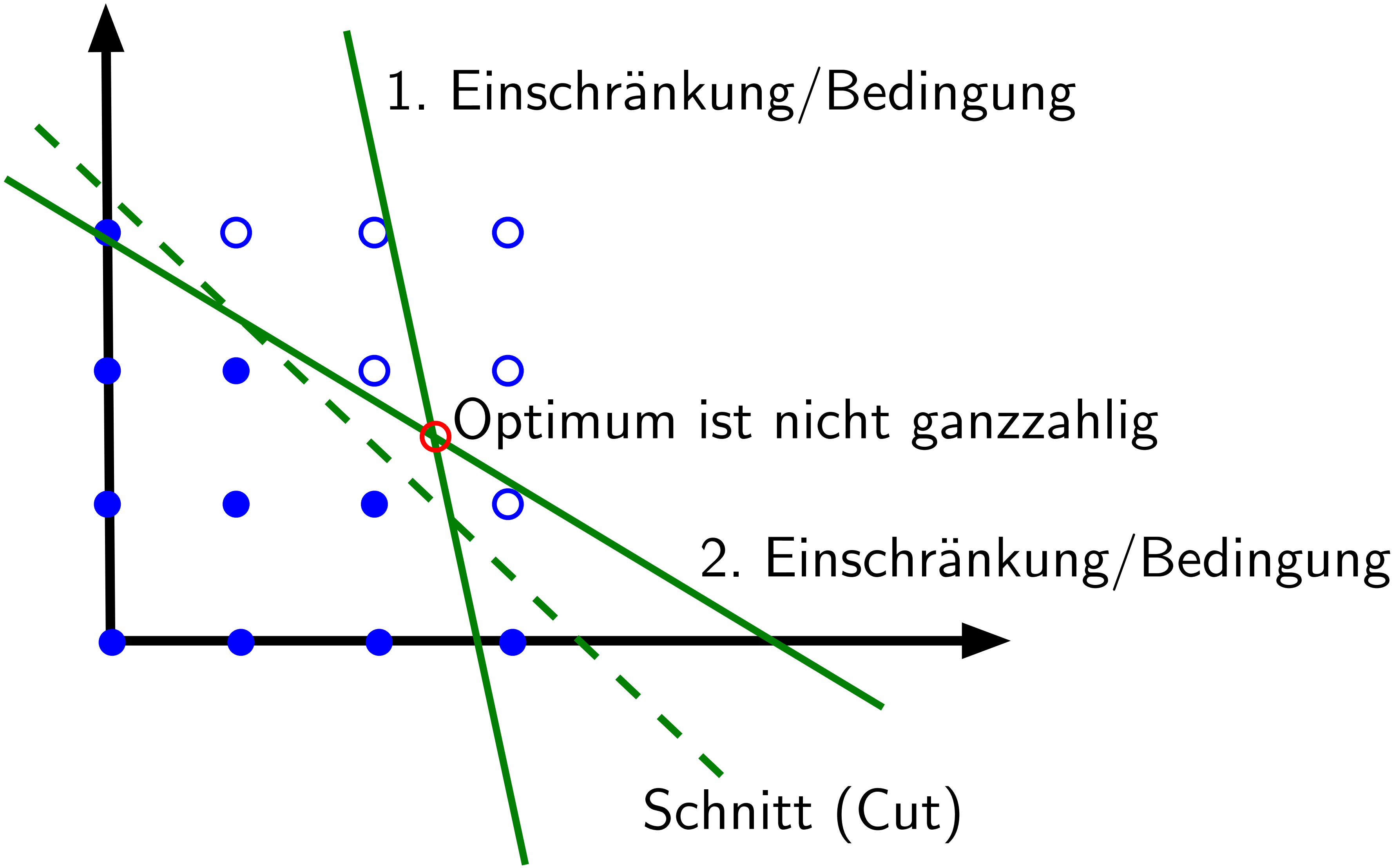
branch and bound (Zusammenf.)

1. **Löse (LR)-Variante des Problems.** Falls Lösung ganzzahlig → fertig, ansonsten erzeuge Subprobleme bzgl. der Belegung einer kritischen Variablen.
2. Ein **Subproblem ist nicht aktiv** falls
   (a) es bereits für eine Verzweigung verwandt wurde,
   (b) alle Variablen der Lösung ganzzahlig sind,
   (c) das Subproblem nicht ganzzahlig lösbar ist,
   (d) ein *bounding*-Argument es ausschließt.
3. **Wähle ein aktives Subproblem und verzweige** bzgl. einer kritischen Variable. **Wiederhole** dies bis keine aktiven Subprobleme mehr vorhanden sind.

Fazit: branch and bound

- Branch-and-bound Verfahren sind im schlechtesten Falle exponentiell.
- Es ist möglich, daß die optimale Lösung erst gefunden wird, wenn alle möglichen Verzweigungen abgesucht worden sind.
  - Ein vollständiger binärer Verzweigungsbaum der Tiefe n hat 2^n Blätter.
- Auch bei der branch-and-bound-Lösung des TSP bringt die detaillierte Analyse nicht viel, denn wir wissen aus der NP-Vollständigkeit dieses Problems, daß es zur Zeit kein deterministisches polynomiales Verfahren zu dessen Lösung gibt.

Alternativen: Ebenen



Alternativen: Heuristiken

- Der Begriff **Heuristik** stammt von dem griechischen Mathematiker *Archimedes* ($\approx$ 285 - 212 v.Z.) und erinnert an seinen berühmten Ausspruch:

  $E\nu\rho\eta\kappa\alpha$ - Heureka! ("Ich hab's gefunden")
  (heurisko (griech.): ich finde).

- Eine **Heuristik** (ein **heuristisches Verfahren**) ist ein Algorithmus, der (im allgemeinen recht gute) Lösungen für Problem-Beispiele erzeugt, aber einer exakten Analyse nicht (oder nur sehr schwer) zugänglich ist. („Die Verfahren funktionieren, aber man weiß nicht genau warum!").

Heuristik vs. Approximation

- Viele lokale Suchverfahren sind Heuristiken. Weitere Heuristiken sind:
  - Simulated Annealing,
  - Tabu Search,
  - Genetische Algorithmen.
- **Approximationsverfahren** sind Algorithmen zur Erzeugung von Lösungen, die i.a. nur suboptimal sind, aber der optimalen Lösung (nachweislich) nahekommen.

diskrete Optimierung

- Ein ein DOP (diskretes Optimierungsproblem) P ist gegeben durch:

  $\mathcal{I}$: Menge der **Instanzen** (Beispiele),

  $\mathcal{L}$: Menge der **Lösungen**,

  w: **Wertfunktion (Zielfunktion)** $w : \mathcal{L} \to I\!R$.

- $s^*(I)$ heißt **optimale Lösung** für I : $\Leftrightarrow$
  - Min-Problem: $w(s^*(I)) \leq w(s(I))$ für alle Lösungen $s(I)$ von $I \in \mathcal{I}$,
  - Max-Problem: $w(s^*(I)) \geq w(s(I))$ für alle Lösungen $s(I)$ von $I \in \mathcal{I}$.

- *Kurz:* $OPT(I) := w(s^*(I))$.

PZ-Algorithmus

- Ein Algorithmus $\mathcal{A}$ heißt **Polynomial-Zeit-Approximationsalgorithmus** (PZ-Algorithmus) für P genau dann, wenn $\mathcal{A}$ für jedes $I \in \mathcal{I}$ in polynomialer Zeit eine Lösung $s_A(I)$ von I erzeugt:

$$\mathcal{A}(I) := w(s_A(I)).$$

Gütemaß für Approx.alg.

P **ist Min-Problem:** $R_{\mathcal{A}}(I) = \frac{\mathcal{A}(I)}{OPT(I)}$

P **ist Max-Problem:** $R_{\mathcal{A}}(I) = \frac{OPT(I)}{\mathcal{A}(I)}$

- Es gilt stets: $1 \leq R_{\mathcal{A}}(I) \leq +\infty$ für alle $I \in \mathcal{I}$ und alle Algorithmen für P.
- $\mathcal{A}$ heißt ein **Optimierungsalgorithmus** für P genau dann, wenn gilt: $\mathcal{A}(I) = OPT(I)$ für alle $I \in \mathcal{I}$.
- Wir betrachten: $R_{\mathcal{A}} := \inf\{c \mid R_{\mathcal{A}}(I) \leq c$ für alle $I \in \mathcal{I}\}$ $\rightarrow 1 \leq R_{\mathcal{A}} \leq +\infty$.
- $R_{\mathcal{A}}$ kann nur selten exakt bestimmt werden. Einfacher kann es sein, $R_{\mathcal{A}}$ abzuschätzen.

Gut- vs. schlecht-approximierbar

Man teilt diskrete Optimierungsprobleme (DOP) ein in:

- **gut-approximierbare DOP** P: es gibt für P Algorithmen $\mathcal{A}$ mit $R_{\mathcal{A}}$ "nahe" bei 1,
- **schlecht-approximierbare DOP** sind Probleme, für die es *nachweislich* keine guten Approximationsalgorithmen gibt.

Beispiel: ΔTSP (D symmetrisch)

ΔTSP ist ein Spezialfall des *TSP*, und zwar soll für D die Dreiecksungleichung gelten: $D = (d_{i,k})$, und für alle i, k, l mit $1 \leq i, k, l \leq n$ sei: $d_{ik} + d_{kl} \geq d_{il}$.

Algorithmus für $\Delta\,TSP$

Gegeben: $I \in \Delta\,TSP$

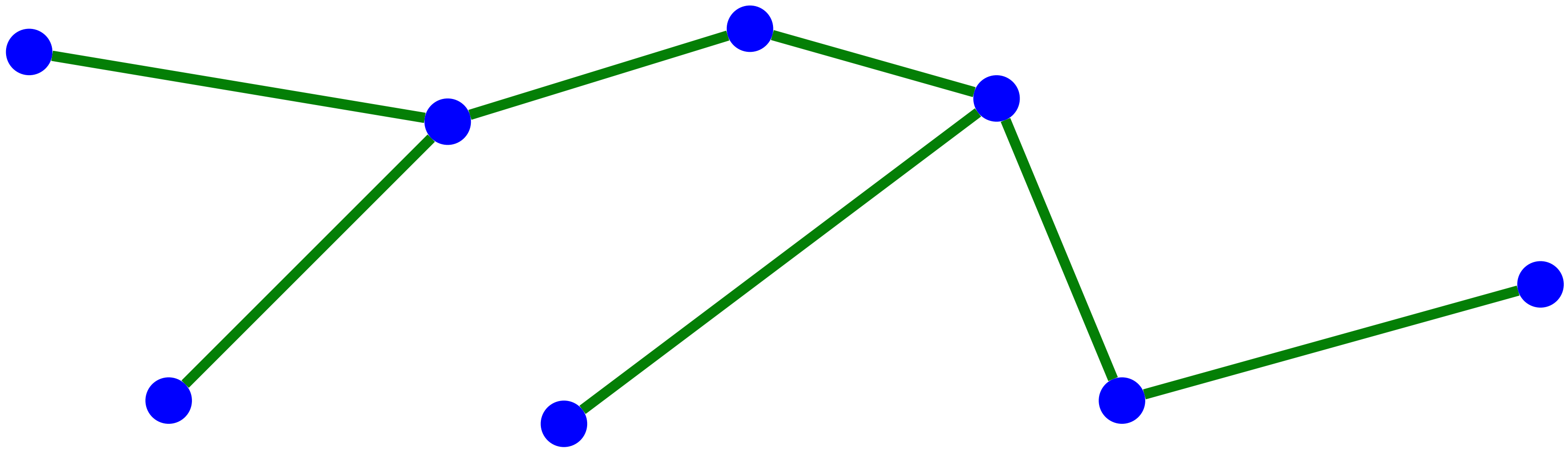
$G = (V, D)$, $E = V \times V$, $D : E \to I\!R^+ \cup \{\infty\}$

Entfernungsmatrix D erfülle die Δ-Ungleichung.

1. Bestimme ein **Minimalgerüst** H von G.
2. Verdopple die Kanten von H
   $\Rightarrow$ **Multigraph** H' ist **Eulergraph** (gerader Knotengrad).
   $\Rightarrow$ H' besitzt **Eulerkreis** C'. Bestimme C' (Zeit: $0(m)$)
3. Erzeuge aus C' durch Anwendung der Δ-Ungleichung einen Hamiltonkreis C^* von G, indem bereits erfaßte Knotenpunkte übersprungen werden.
4. STOP: Output: C^* mit $l(C^*) = \mathcal{A}(I)$.

Beispiel

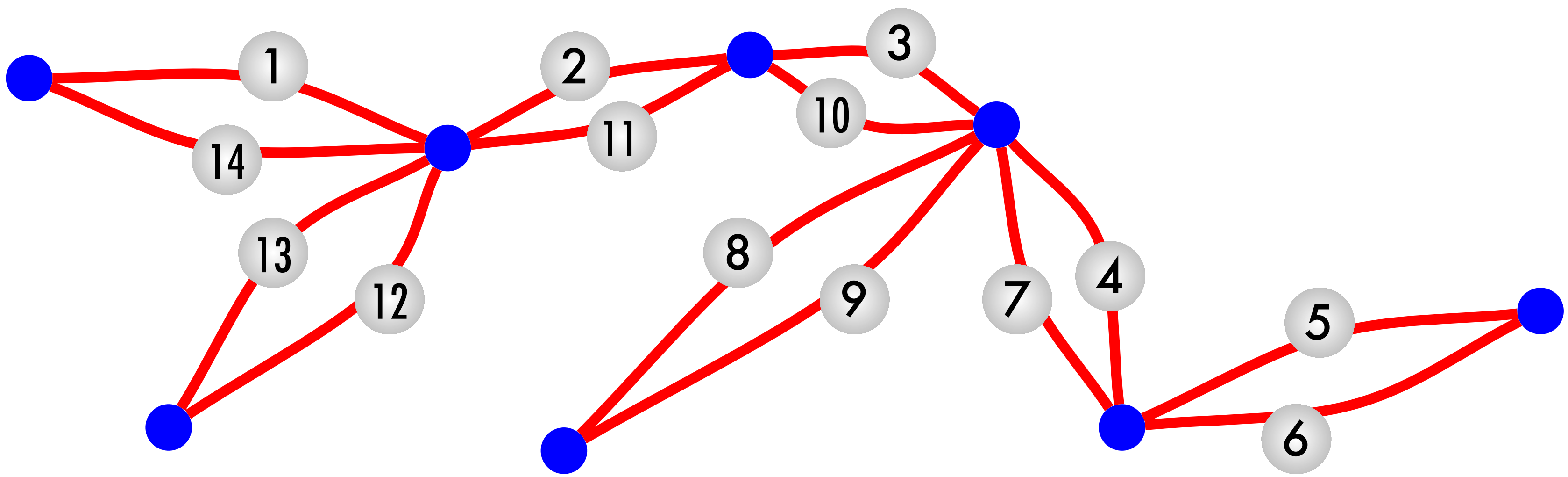
Sei dies ein Minimalgerüst H eines Graphen G:



Beispiel (2)

Wir verdopplen die Kanten von H und erhalten einen Eulergraphen H' mit dem Eulerkreis C', für den die Reihenfolge der Kanten hier angegeben ist:

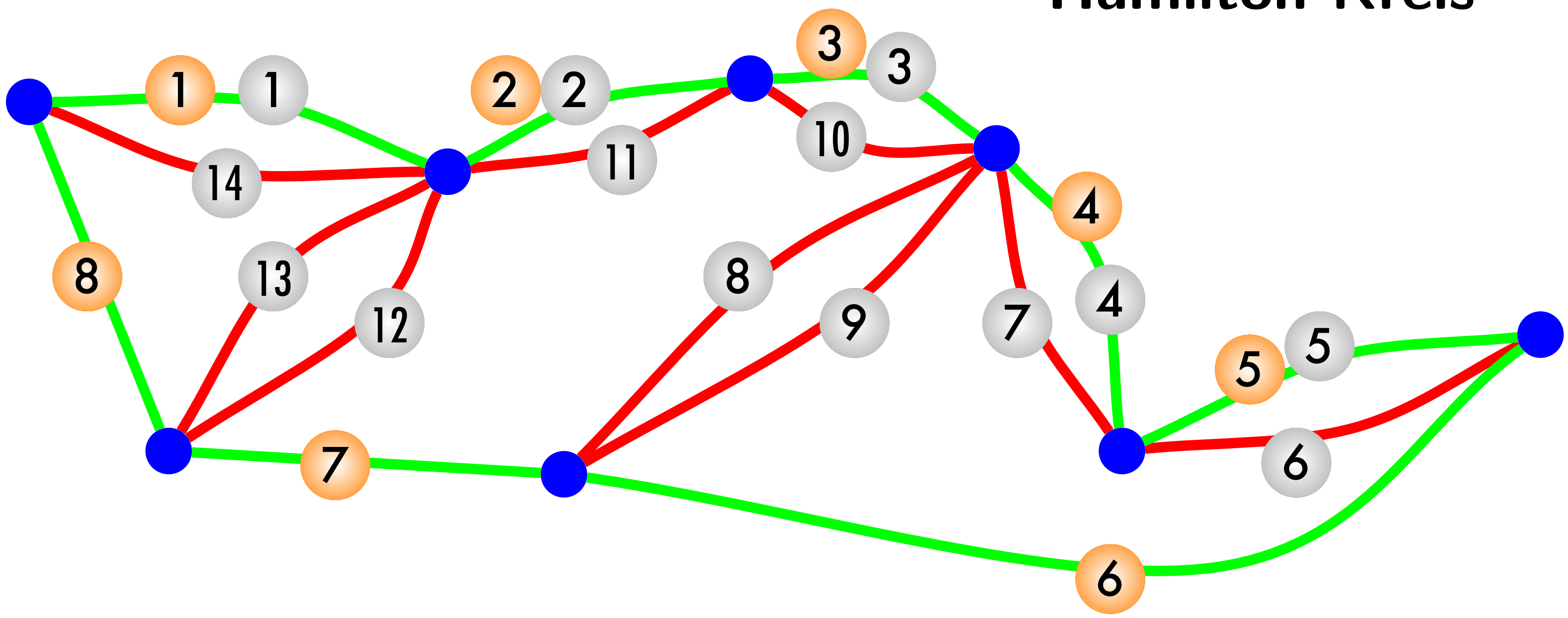
Euler-Kreis



Beispiel

Aus C' konstruieren wir den Hamiltonkreis C^*:

Hamilton-Kreis



Gut- vs. schlecht-approximierbar (2)

- Es gilt: $l(H) \leq OPT(I) \leq l(C^*) \leq l(C') = 2 \cdot l(H)$.
- Folglich ist $OPT(I) \leq \mathcal{A}(I) \leq 2 \cdot OPT(I)$, also $R_A(I) = \frac{\mathcal{A}(I)}{OPT(I)} \leq 2$ und somit $R_A \leq 2$.
- Dieses Ergebnis kann kaum verbessert werden, denn es gelten die folgenden Ergebnisse.
  - **Theorem:** Das ΔTSP-Problem ist NP-vollständig. (d.h. es ist ein sehr schwieriges Problem, für das bislang bestenfalls Exponentialzeitalgorithmen existieren!)

Gut- vs. schlecht-approximierbar (3)

- **Theorem:** Gäbe es für jedes $\epsilon > 0$ einen PZ-Approximationsalgorithmus $\mathcal{A}$ für TSP, der für jede symmetrische Entfernungsmatrix $D \in \mathbb{N}^{n \times n}$ in polynomialer Zeit eine Tour T_A liefert, mit der Länge $\mathcal{A}_{T_A}(I)$, für die $R_{\mathcal{A}} \leq 1 + \epsilon$ gilt, dann ist HAMILTONKREIS $\in \mathcal{P}$, d.h. dann ist $\mathcal{P} = \mathcal{NP}$.

- Also ist TSP vermutlich (d.h. unter der Annahme, daß $\mathcal{P} \neq \mathcal{NP}$ gilt) ein schlecht-approximierbares Problem.

- **Beispiel für ein gut-approximierbares Problem:** Für das Rucksackproblem gibt es einen PZ-Algorithmus $\mathcal{A}$ mit $R_{\mathcal{A}} \leq 1 + \epsilon$ für beliebige $\epsilon \geq 0$.

Konstruktion von Wegen

Nearest neighbour: Start-Ort wird zufällig gewählt und es wird immer die nächste noch nicht besuchte Stadt besucht bis die letzte erreicht ist. Dann zurück zum Start-Ort.

Nearest insertion: Beginn mit einer möglichst *kurzen* Strecke zwischen zwei Orten. Dann: Entfernen jeweils einer Kante mit Einfügen einer Stadt als Zwischenstop. Wähle die Stadt so, daß *die Strecke wenig erhöht* wird!

Furthest insertion: Beginn mit möglichst *langer* Strecke. Dann: wie oben, aber wähle die Stadt und Kante so, daß *die Strecke größtmöglich erhöht* wird!

Sweep: Markiere den Mittelpunkt der Landkarte. Auswahl der nächsten Stadt, wie ein Radarstrahl.

Optimierung von Wegen

Two-opt: Systematischer Vergleich jeweils zweier Kanten und ggf. Austausch gegen günstigere Variante.

Three-opt: analog Two-opt, aber mit je drei Kanten.

Lin-Kernighan: Entfernen einer Kante aus Rundtour liefert einen Pfad. Ein Ende mit innerem Knoten verbinden und andere Kante löschen, so daß wieder ein Pfad vorliegt. Wiederholung dieser Prozedur solange die *gain sum* (gelöschte minus eingefügte Kantengewichte) positiv ist und noch nicht behandelte Kanten existieren. Merke jeweilige Kosten für wiederhergestellte Rundtour. Wähle am Ende die Beste!