Effiziente Algorithmen und Datenstrukturen I

Kapitel 12: Approximationsalgorithmen

Christian Scheideler

WS 2008

Übersicht

- P und NP
- Approximationsalgorithmen
- Lastbalancierung
- Zentrumswahl
- Knotenüberdeckung
- Allgemeine Lastbalancierung
- Rucksack-Problem
- Problem des Handlungsreisenden

P und NP

P: Klasse aller Entscheidungsprobleme (Anworten $\in$ {Ja, Nein}), die in polynomieller Zeit (in der Eingabegröße) von einer Turingmaschine / RAM entschieden werden können.

Beispiele: Sortiertheit einer Folge, Auswertung eines Schaltkreises, Lineare Optimierung,…

P und NP

NP: Klasse aller Entscheidungsprobleme (Anworten $\in$ {Ja, Nein}), für die es für Eingaben mit Antwort Ja Zeugen gibt, so dass die Antwort in polynomieller Zeit (in der Eingabegröße) von einer Turingmaschine / RAM verifiziert werden können.

Beispiele: Erfüllbarkeit einer Booleschen Formel in KNF, 3-Färbung, Knotenüberdeckung,…

P und NP

Das 1-Million-Dollar-Problem: Ist P=NP oder nicht????

- Antwort auf diese Frage scheint sehr schwer zu sein.
- Bisher nur Ergebnisse des Typs:
  “Das kann nicht in poly Zeit gelöst werden, es sei denn, P=NP.”
- Klasse der NP-harten Probleme: Hart zu lösen, es sei denn, P=NP.

Approximationsalgorithmen

Frage: Ich will ein NP-hartes Problem lösen. Was muss ich tun?
Antwort: Polynomialzeitalgo dafür wohl nicht möglich.

Eine der drei Eigenschaften muss aufgegeben werden:

- Löse das Problem optimal.
- Löse das Problem in polynomieller Zeit
- Löse beliebige Instanzen des Problems

ρ-Approximationsalgorithmus:

- Läuft in polynomieller Zeit.
- Löst beliebige Instanzen des Problems.
- Findet eine Lösung, die höchstens Faktor ρ weg von Optimum ist.

Herausforderung: Lösung sollte möglichst nah an Optimum sein.

Übersicht

- P und NP
- Approximationsalgorithmen
- Lastbalancierung
- Zentrumswahl
- Knotenüberdeckung
- Allgemeine Lastbalancierung
- Rucksack-Problem
- Problem des Handlungsreisenden

Lastbalancierung

Eingabe: m identische Maschinen, n Jobs. Job i hat Laufzeit t_i.

Einschränkungen:

- Ein einmal ausgeführter Job muss bis zum Ende auf derselben Maschine ausgeführt werden.
- Jede Maschine kann höchstens einen Job gleichzeitig bearbeiten.

Definition 12.1: Sei $J(i)$ die Teilmenge der Jobs, die Maschine i zugewiesen werden. Dann ist $L_i = \sum_{j \in J(i)} t_j$ die Last der Maschine i.

Definition 12.2: Der Makespan L ist die maximale Last einer Maschine, d.h. $L = \max_i L_i$

Lastbalancierung: finde Zuweisung, die Makespan minimiert

Lastbalancierung: List Scheduling

List-Scheduling Algorithmus:

- Betrachte n Jobs in einer festen Reihenfolge
- Weise Job j der Maschine mit z.Zt. geringster Last zu

List-Scheduling(m, n, $(t_1,\ldots,t_n)$):

```
for i:=1 to m do
    L_i := 0; J(i):=∅
for j:=1 to n do
    i:=argmin_k L_k
    J(i):=J(i) ∪ {j}
    L_i:=L_i + t_j
return (J(1),…,J(m))
```

Laufzeit: $O(n \log m)$ mit Priority Queue

Lastbalancierung: List Scheduling

Theorem 12.3 (Graham): Der Greedy Algorithmus ist 2-approximativ.

$\rightarrow$ vergleiche Güte des Algorithmus mit optimalem Makespan L^*

Lemma 12.4: $L^* \geqslant \max_j t_j$

Beweis: Eine Maschine muss den zeitintensivsten Job bearbeiten.

Lemma 12.5: $L^* \geqslant (1/m) \sum_j t_j$

Beweis:

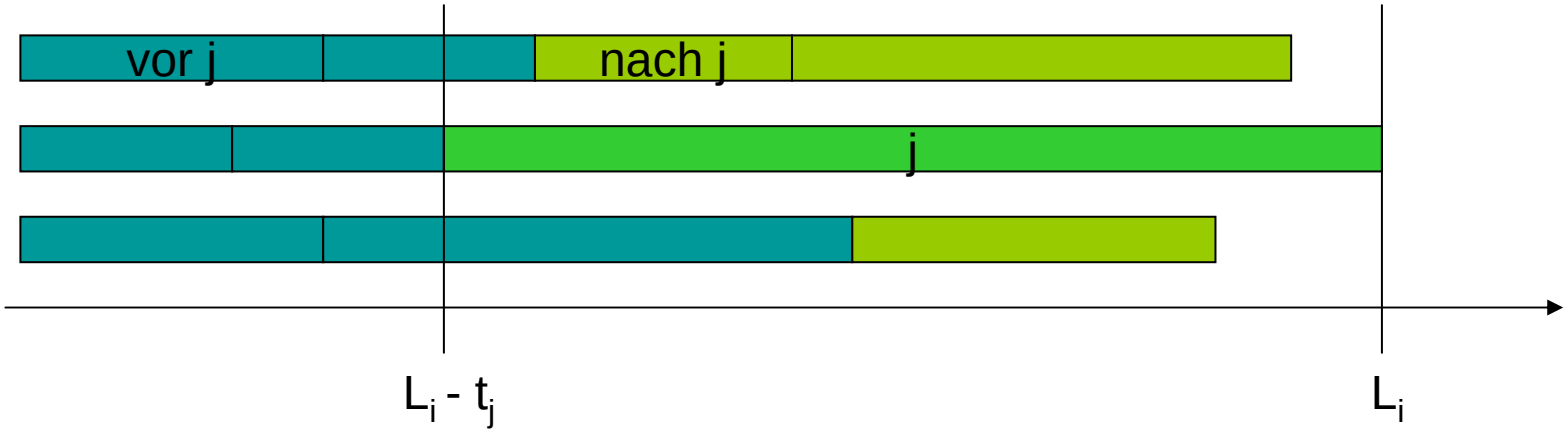
- Die Gesamtlast ist $\sum_j t_j$
- Eine der m Maschinen muss mindestens $1/m$ der Gesamtlast bekommen.

Lastbalancierung: List Scheduling

Theorem 12.3: Der Greedy Algorithmus ist 2-approximativ.

Beweis:

- Betrache Maschine i mit höchster Last L_i.
- Sei j der letzte Job in Maschine i.
- Da Job j Maschine i zugeordnet wurde, hatte i vorher die kleinste Last. Es gilt also $L_i - t_j \leqslant L_k$ für alle k.



Lastbalancierung: List Scheduling

Beweis (Forsetzung):

- Es gilt: $L_i - t_j \leqslant L_k$ für alle k
- Daraus folgt:

$$L_i - t_j \leqslant (1/m) \sum_k L_k$$
$$= (1/m) \sum_k t_k$$
$$\leqslant L^* \qquad \text{(Lemma 12.4)}$$

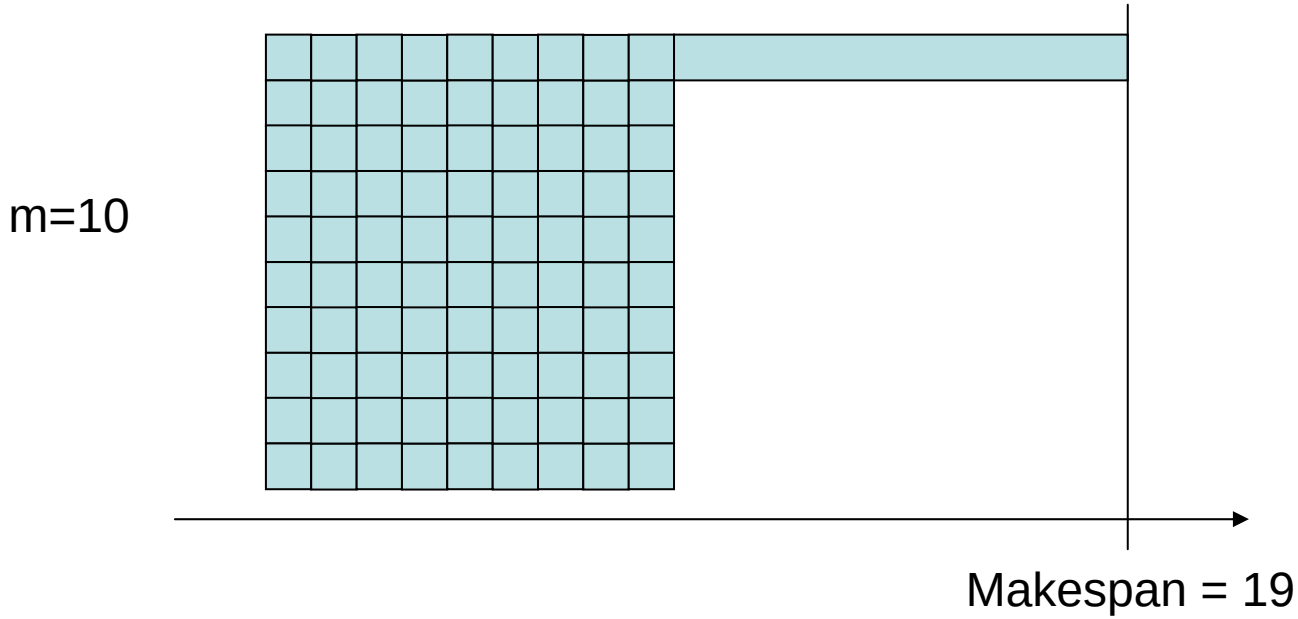
- Also gilt wegen Lemma 12.5:

$$L_i = (L_i - t_j) + t_j \leqslant 2L^*$$

Lastbalancierung: List Scheduling

Ist die Analyse scharf? Ja!

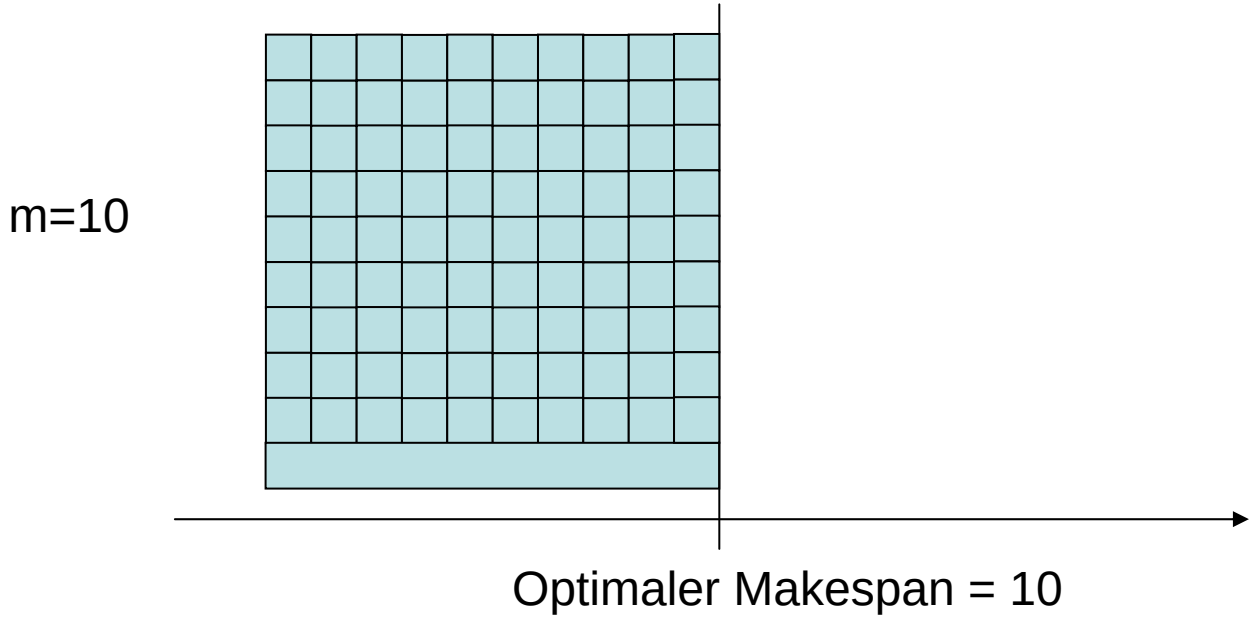
Beispiel: m Maschinen, m(m-1) Jobs der Länge 1,
ein Job der Länge m



Lastbalancierung: List Scheduling

Ist die Analyse scharf? Ja!

Beispiel: m Maschinen, m(m-1) Jobs der Länge 1,
ein Job der Länge m



Lastbalancierung: LPT Regel

Longest Processing Time (LPT): Sortiere die n Jobs in absteigender Reihenfolge und führe dann den List Scheduling Algorithmus aus.

LPT-List-Scheduling(m, n, $(t_1,\ldots,t_n)$):
sortiere Jobs, so dass $t_1 \geqslant t_2 \geqslant \ldots \geqslant t_n$
for i:=1 to m do
$L_i := 0$; $J(i):=\emptyset$
for j:=1 to n do
$i:=\text{argmin}_k\ L_k$
$J(i):=J(i) \cup \{j\}$
$L_i:=L_i + t_j$
return $(J(1),\ldots,J(m))$

Lastbalancierung: LPT Regel

Beobachtung: Wenn es höchstens m Jobs gibt, dann ist List Scheduling optimal.

Beweis: Weise jedem Job eigene Maschine zu.

Lemma 12.6: Falls es mehr als m Jobs gibt, dann ist $L^* \geqslant 2t_{m+1}$.

Beweis:

- Betrachte die ersten $m+1$ Jobs $t_1,\ldots,t_{m+1}$
- Da die t_i's absteigend sortiert sind, benötigt jeder dieser Jobs mindestens t_{m+1} Zeit.
- Bei $m+1$ Jobs muss eine Maschine mindestens zwei Jobs erhalten.

Lastbalancierung: LPT Regel

Theorem 12.7: Die LPT Regel liefert eine 3/2-Approximation.

Beweis:

Falls die Maschine i mit größter Last nur einen Job hat, ist LPT offensichtlich optimal. Sonst gilt für den letzten Job j auf Maschine i, dass $j \geqslant m+1$ und damit nach Lemma 12.6 gilt:

$$\begin{aligned} L_i &= (L_i - t_j) + t_j \\ &\leqslant L^* + (1/2)L^* \\ &\leqslant (3/2)L^* \end{aligned}$$

Ist 3/2 scharf? Nein!

Lastbalancierung: LPT Regel

Theorem 12.8 (Graham): Die LPT Regel ist eine 4/3-Approximation.

Beweis: aufwändig

Theorem 12.8 ist im Wesentlichen scharf.

Beispiel: m Maschinen, $n=2m+1$ Jobs: jeweils 2 Jobs der Länge $m+1, m+2, \ldots, 2m$ und ein Job der Länge m

Vergleich zu OPT: Übung.

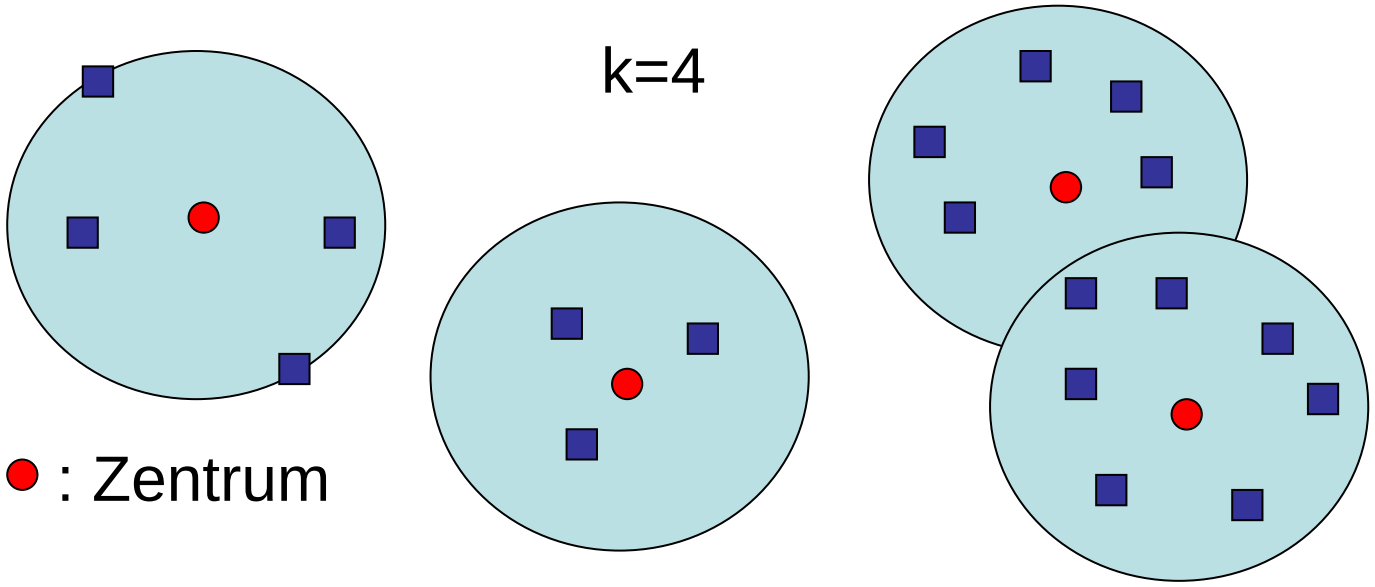
Übersicht

- P und NP
- Approximationsalgorithmen
- Lastbalancierung
- Zentrumswahl
- Knotenüberdeckung
- Allgemeine Lastbalancierung
- Rucksack-Problem
- Problem des Handlungsreisenden

Zentrumswahl-Problem

Eingabe: Menge von n Orten $s_1,\ldots,s_n$ und eine Zahl $k \in \mathbb{N}$.

Zentrumswahl-Problem: Wähle k Zentren C, so dass die maximale Distanz eines Ortes zum nächsten Zentrum minimal ist.



Zentrumswahl-Problem

Eingabe: Menge von n Orten $s_1,\ldots,s_n$ und eine Zahl $k \in \mathbb{N}$.

Zentrumswahl-Problem: Wähle k Zentren C, so dass die maximale Distanz eines Ortes zum nächsten Zentrum minimal ist.

Notation:

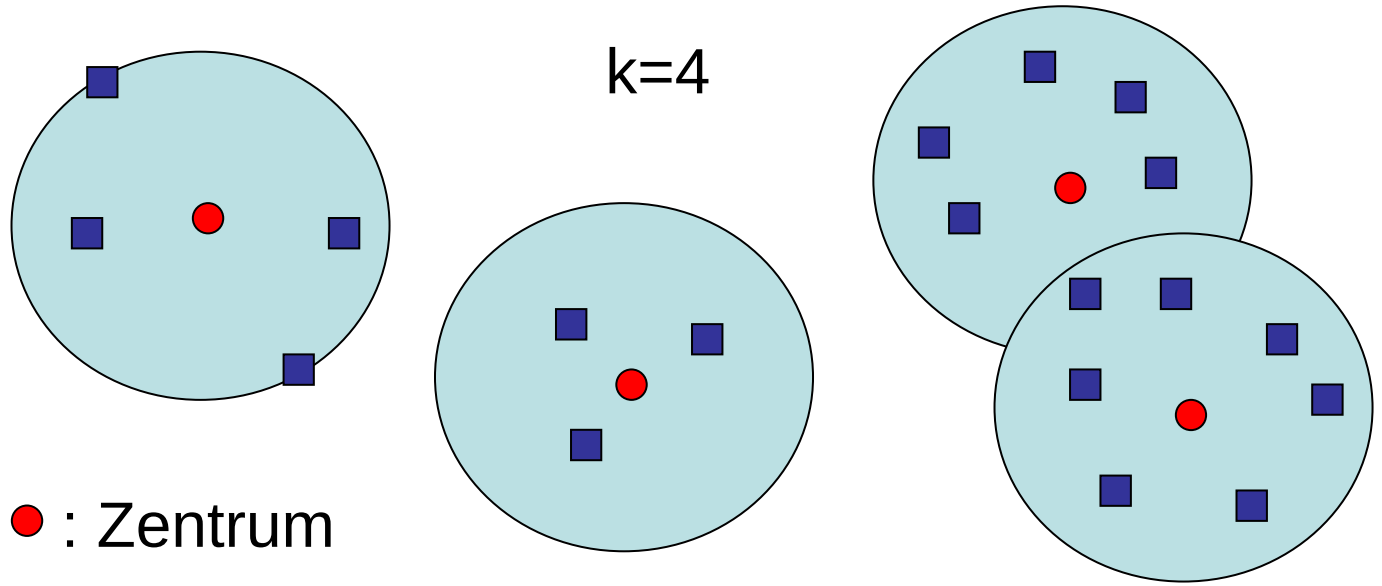
- $dist(x,y)$ = Distanz zwischen x und y
- $dist(s_i, C) = \min_{c \in C} dist(s_i,c)$ = Distanz von s_i zum nächsten Zentrum
- $r(C) = \max_i dist(s_i, C)$ = kleinster Überdeckungsradius

Wir nehmen an, dist ist eine Metrik, d.h.

- $dist(x,x) = 0$ (Identität)
- $dist(x,y) = dist(y,x)$ (Symmetrie)
- $dist(x,y) \leqslant dist(x,z) + dist(z,y)$ (Dreiecksungleichung)

Zentrumswahl-Problem

Beispiel: jeder Ort ist ein Punkt im 2-dimensionalen Euklidischen Raum, dist(x,y) = Euklidische Distanz

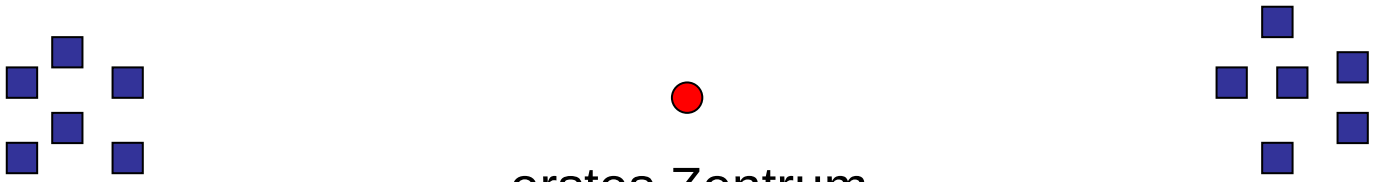


Zentrumswahl: Greedy Algorithmus

Greedy Algorithmus: Setze das erste Zentrum an der bestmöglichen Stelle für ein einzelnes Zentrum, füge dann Zentren hinzu, um den Überdeckungsradius möglichst stark zu verkleinern.

Kann beliebig schlecht werden!!

Beispiel: k=2



Zentrumswahl: Greedy Algorithmus

Greedy Algorithmus: wähle wiederholt als nächstes Zentrum den Ort mit maximaler Distanz zu allen bisherigen Zentren

Greedy-Center-Selection(k, n, (s_1,s_2,…,s_n)):

```
C:=∅
wiederhole k-mal
   wähle Ort s_i mit maximalem dist(s_i,C)
   C:=C ∪ {s_i}
return C
```

Bemerkung: erstes Zentrum ist beliebiger Ort

Zentrumswahl: Greedy Algorithmus

Bemerkung: Zentren in C sind mindestens $r(C)$ entfernt voneinander
Beweis: $r(C)$ sinkt monoton, jeweils minimale paarweise Entfernung

Theorem 12.9: Sei C^* die optimale Wahl der Zentren. Dann ist $r(C) \leqslant 2r(C^*)$.

Beweis (durch Widerspruch):

- Angenommen, $r(C^*) < r(C)/2$.
- Betrachte die Kreise mit Radius $r(C)/2$ um jedes $c_i \in C$.
- Es muss genau ein $c \in C^*$ im Kreis von jedem c_i geben (siehe Bemerkung und Annahme); wir nennen dieses Zentrum c^*_i
- Betrachte einen beliebigen Ort s und sei c^*_i sein nächstes Zentrum in C^*. Es gilt:
  $$dist(s,C) \leqslant dist(s,c_i) \leqslant dist(s,c^*_i) + dist(c^*_i,c_i) \leqslant 2r(C^*)$$
- Also ist $r(C) \leqslant 2r(C^*)$, ein Widerspruch zur Annahme

Zentrumswahl

Wir wissen: Der Greedy Algorithmus ergibt eine 2-Approximation.

Gibt es auch Polynomialzeitalgorithmen mit Approximationsgüte 3/2? Oder 4/3?

Theorem 12.10: Sofern nicht P=NP ist, gibt es keinen Polynomialzeitalgorithmus mit Approximationsgüte < 2 für die Zentrumswahl (für $k>2$).

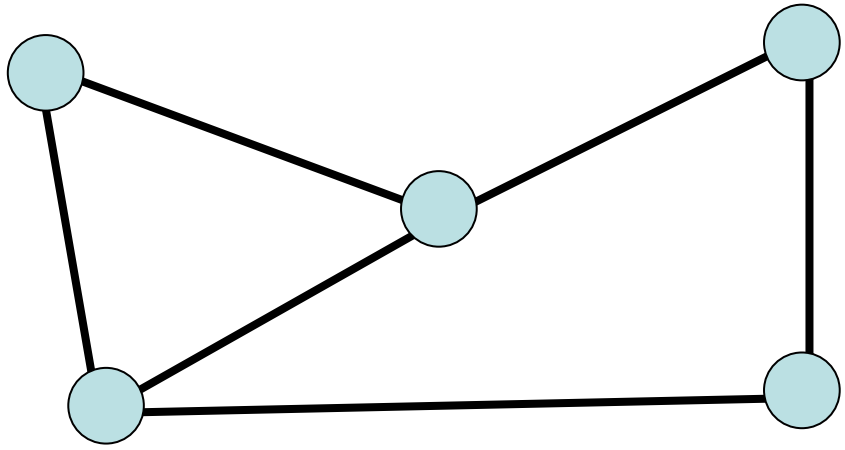
Übersicht

- P und NP
- Approximationsalgorithmen
- Lastbalancierung
- Zentrumswahl
- Knotenüberdeckung
- Allgemeine Lastbalancierung
- Rucksack-Problem
- Problem des Handlungsreisenden

Knotenüberdeckung

(Ungewichtete) Knotenüberdeckung (VC):

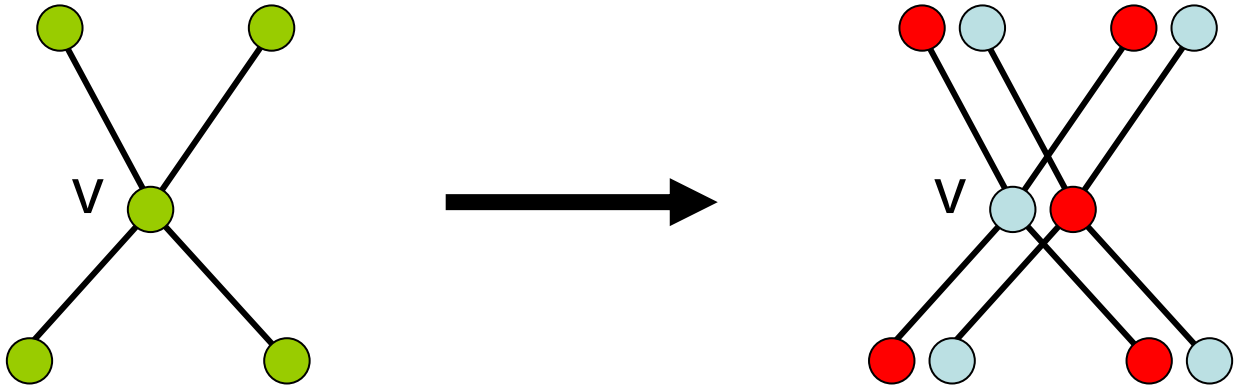
Gegeben ein Graph G, finde eine Knotenüberdeckung mit minimaler Anzahl Knoten.



VC: Lokale Methode

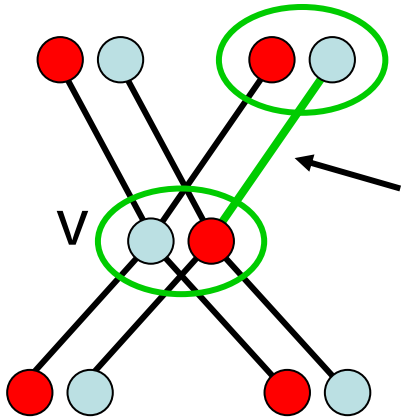
Schritt 1: Wir transformieren Graph G in G' wie folgt:

- Jeder Knoten v erhält eine blaue und eine rote Kopie.
- Die blaue (bzw. rote) Kopie von v verbindet sich mit den roten (bzw. blauen) Kopien der Nachbarn von v.



VC: Lokale Methode

Schritt 2: Finde maximales Matching M in G’ (Methode dazu wird noch beschrieben). Jeder Knoten in G, von dem eine Kopie in M beteiligt ist, wird der Knotenüberdeckung C zugeordnet, die ausgegeben wird.



$v \in C$, da Kante in Matching M

Details zu Schritt 2

Wir nehmen an:

- Nachbarn eines Knotens v durchnummeriert von 1 bis $d(v)$ (Grad von v)
- $i(v)$: Zähler von Knoten v
- $b(v)=i$: Kante {blaues v, rotes i} $\in M$
- $r(v)=j$: Kante {rotes v, blaues j} $\in M$

Anfangs ist $b(v)=\perp$ und $r(v)=\perp$.

Details zu Schritt 2

Für jeden Knoten v parallel zu den anderen:

```
for i(v):=1 to max_w d(w) + 1 do
    // alle Knoten starten Durchlauf zur gleichen Zeit
    if b(v)=⊥ ∧ i(v)⩽d(v)+1 then
       if Botschaft “accept” von i(v)-1 then
          b(v):=i(v)-1
       else
          sende Botschaft “propose” an i(v)
    foreach “propose” von einem Nachbar j do
       if r(v)=⊥ then
          r(v):=j; sende “accept” an j
       else
          sende “reject” an j
```

VC: Lokale Methode

Lemma 12.11: M ist maximales Matching in G'.

Beweis:

- Matchingeigenschaft folgt direkt aus Algorithmus.
- Maximalität folgt aus Beweis von Lemma 12.12.

Lemma 12.12: C ist eine Knotenüberdeckung.

Beweis:

- Betrachte beliebige Kante $e=\{u,v\}$
- $b(u) \neq \perp$: $u \in C$; sonst sendet u "propose" und hat "reject" von v erhalten
- Dann ist aber $r(v) \neq \perp$, also $v \in C$

VC: Lokale Methode

Theorem 12.13: Die lokale Methode ergibt $|C| \leqslant 3 \cdot |C^*|$ für eine optimale Überdeckung C^*.

Beweis:

- Matching M in G' ergibt in G Menge knotendisjunkter Kreise und Pfade, da alle Knoten in G Grad $\leqslant 2$ bzgl. M haben.
- Jeder Knoten in C ist in diesen Kreisen und Pfaden enthalten.
- Lösche aus jedem Kreis eine Kante, so dass wir nur Pfade haben.
- Für jeden Pfad P muss jede Kante in P abgedeckt sein. Falls P k Knoten enthält, müssen mindestens $\lfloor k/2 \rfloor$ Knoten in P in C^* sein. Das ist $\geqslant k/3$ für alle k.

VC: Lokale Methode

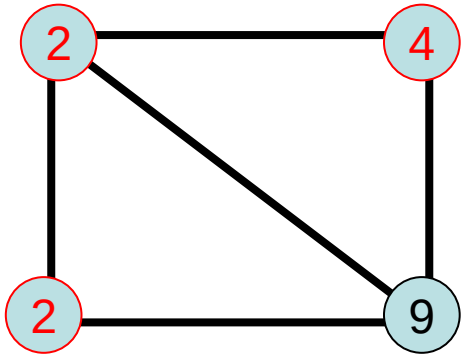
Vorteile der lokalen Methode:

- sehr einfach zu implementieren, auch verteilt
- sehr schnell ($O(n+m)$ Zeit sequentiell, $O(\max_v d(v))$ Zeit parallel)

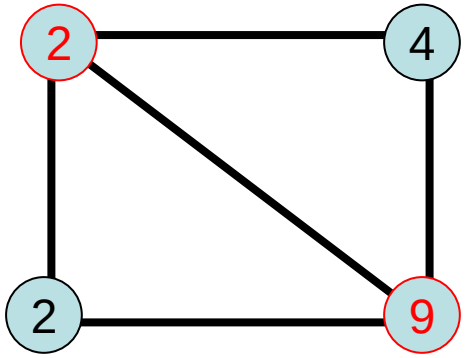
Nachteil: Approximationsgüte 3 nicht optimal

Gewichtete Knotenüberdeckung

Gewichtete Knotenüberdeckung: Gegeben ein Graph G mit Knotengewichten $w \geqslant 0$, finde eine Knotenüberdeckung mit minimalem Gewicht.



Gewicht 2+2+4=8



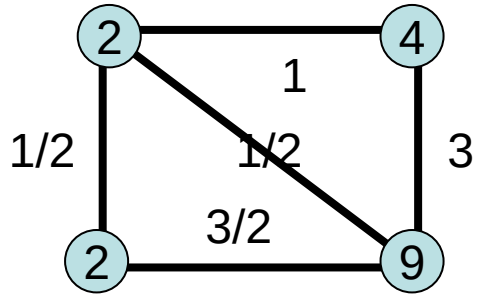
Gewicht 11

VC: Kostenmethode

Jede Kante $e=\{i,j\}$ bezahle Preis $p_e \geqslant 0$, um Knoten i und j zu benutzen.

Fairness: Kanten inzident zu Knoten i sollten ingesamt höchstens w_i bezahlen.

Für jeden Knoten i: $\sum_{e=\{i,j\}} p_e \leqslant w_i$



Lemma 12.14: Für jede Knotenüberdeckung S und jede Wahl von fairen Kosten p_e gilt $\sum_e p_e \leqslant w(S)$.

Beweis:

$$\sum_e p_e \leqslant \sum_{i \in S} \sum_{e=\{i,j\}} p_e \leqslant \sum_{i \in S} w_i = w(S)$$

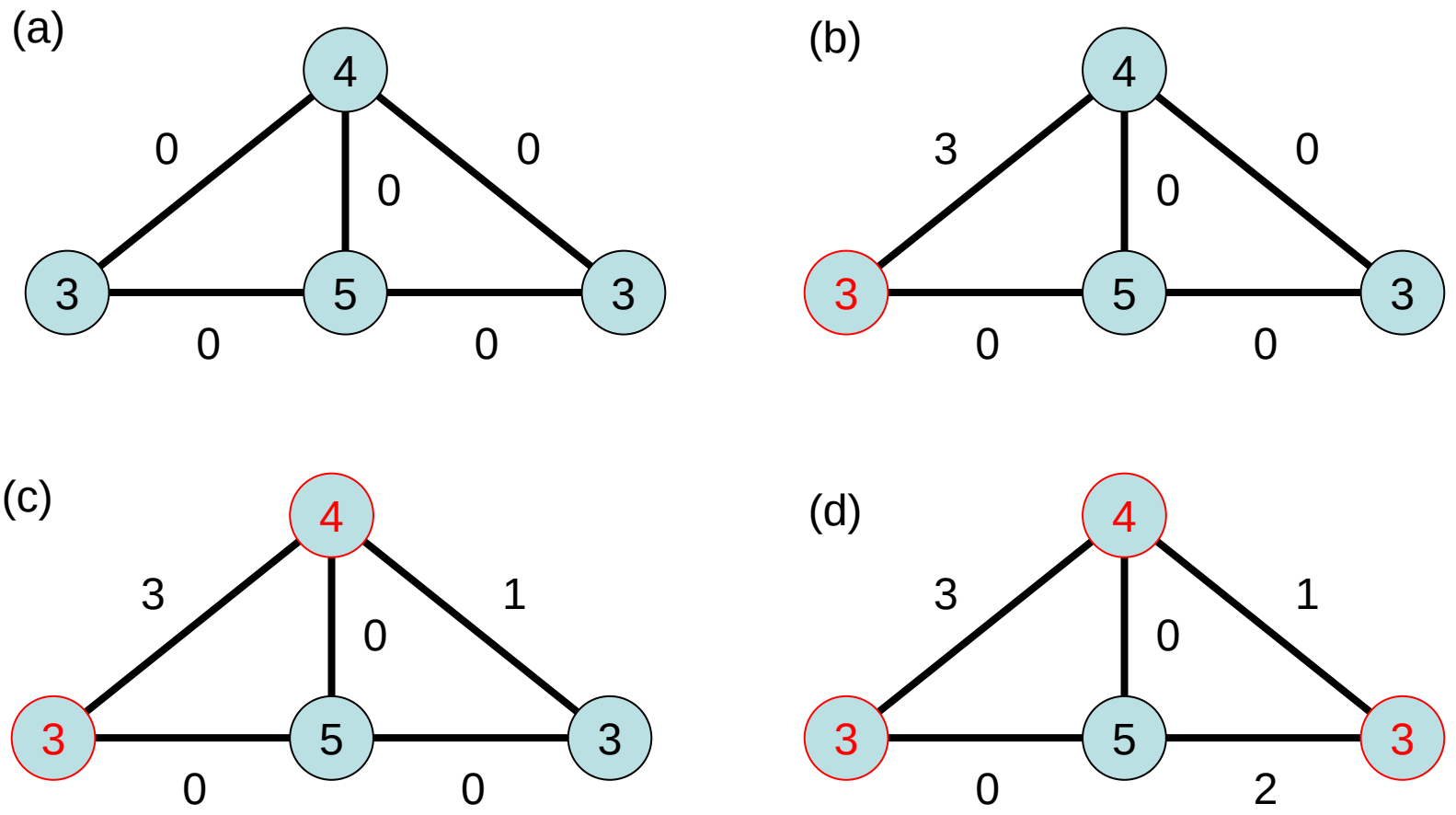
VC: Kostenmethode

Kostenmethode: Wähle gleichzeitig Kosten und eine Knotenüberdeckung.

Knoten i saturiert: $\sum_{e=\{i,j\}} p_e = w_i$

Weighted-Vertex-Cover(G,w):
  foreach $e \in E$: $p_e = 0$
  while ($\exists$ $e=\{i,j\} \in E$ s. d. weder i noch j saturiert)
    erhöhe p_e, bis i oder j saturiert
  S:=Menge der saturierten Knoten
  return S

VC: Kostenmethode



VC: Kostenmethode

Theorem 12.15: Die Kostenmethode ergibt eine 2-Approximation.

Beweis:

- Der Algorithmus terminiert, da in jedem Schleifendurchlauf mindestens ein Knoten saturiert wird.
- S ist eine Knotenüberdeckung, denn wenn eine Kante $\{i,j\}$ nicht abgedeckt wäre, dann wären i und j nicht saturiert. In diesem Fall wäre die while-Schleife aber nicht terminiert.
- Sei S^* eine optimale Knotenüberdeckung. Dann gilt wegen Lemma 12.11:

$$w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=\{i,j\}} p_e$$
$$\leqslant \sum_{i \in V} \sum_{e=\{i,j\}} p_e = 2 \sum_e p_e \leqslant 2w(S^*)$$

VC: ILP Methode

Gewichtete Knotenüberdeckung: Gegeben ein Graph G mit Knotengewichten $w \geqslant 0$, finde eine Knotenüberdeckung mit minimalem Gewicht.

Ganzzahliges Programm:

- Für jeden Knoten i Indikatorvariable x_i

$$x_i = \begin{cases} 0 & \text{falls Knoten } i \text{ nicht in Überdeckung} \\ 1 & \text{falls Knoten } i \text{ in Überdeckung} \end{cases}$$

  Knotenüberdeckung in 1-1-Zusammenhang mit 0/1-Zuweisung: $S=\{i \in V \mid x_i=1\}$.
- Zielfunktion: minimiere $\sum_i w_i x_i$
- Wir müssen für $\{i,j\}$ entweder i oder j nehmen: $x_i+x_j \geqslant 1$

VC: ILP Methode

Gewichtete Knotenüberdeckung als ganzzahliges Programm (ILP):

$$\begin{aligned} & \min \sum_i w_i x_i \\ \text{s. d.} \quad & x_i + x_j \geqslant 1 \text{ für alle } \{i,j\} \in E \\ & x_i \in \{0,1\} \text{ für alle } i \in V \end{aligned}$$

Beobachtung: Für eine optimale Lösung x^* des ILPs ist $S=\{ i \in V \mid x^*_i = 1\}$ eine Knotenüberdeckung minimalen Gewichts

Ganzzahlige Programmierung

Gegeben eine $m \times n$-Matrix $A=(a_{i,j})$, $b=(b_i) \in IR^m$ und $c=(c_j) \in IR^n$, finde ein $x=(x_j) \in IR^n$, so dass

$$\max c^T x \quad \text{s. d.} \quad Ax \leqslant b, \quad x \geqslant 0 \text{ integral}$$

bzw.

$$\max \sum_{j=1}^{n} c_j x_j \quad \sum_{j=1}^{n} a_{i,j} x_j \leqslant b_i \quad 1 \leqslant i \leqslant m, \quad x_i \geqslant 0 \quad 1 \leqslant j \leqslant n, \quad x_i \text{ integral}$$

Beobachtung: Da Knotenüberdeckung NP-hart ist, sind auch ILPs NP-harte Probleme.

Lineare Programmierung

Gegeben eine $m \times n$-Matrix $A=(a_{i,j})$, $b=(b_i) \in \mathbb{R}^m$ und $c=(c_j) \in \mathbb{R}^n$, finde ein $x=(x_j) \in \mathbb{R}^n$, so dass

$$\max c^T x \quad \text{s. d.} \quad Ax \leqslant b, \quad x \geqslant 0$$

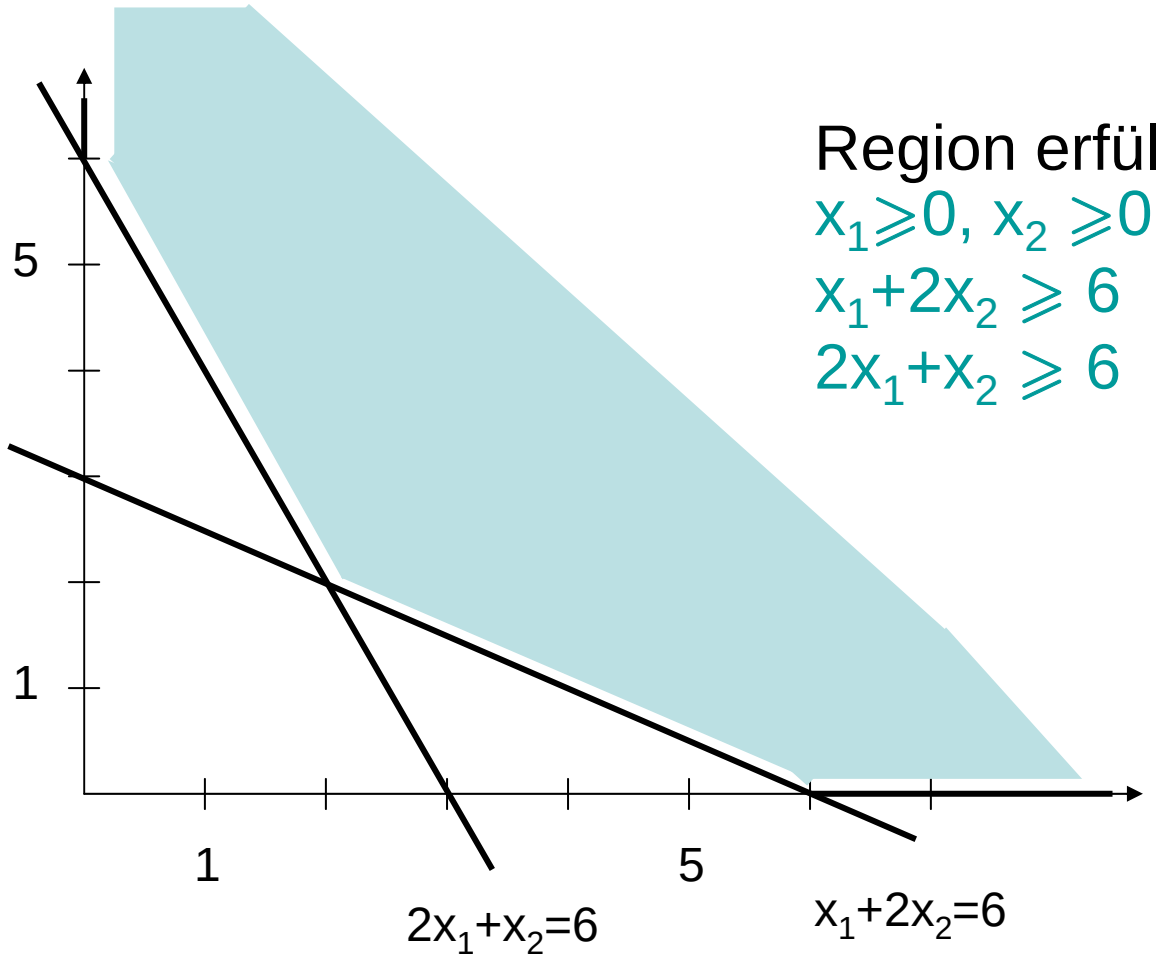
bzw.

$$\max \sum_{j=1}^{n} c_j x_j \quad \sum_{j=1}^{n} a_{i,j} x_j \leqslant b_i \quad 1 \leqslant i \leqslant m, \quad x_i \geqslant 0 \quad 1 \leqslant j \leqslant n$$

Simplex Algo [Dantzig 1947]: Kann LPs in der Praxis lösen.

Ellipsoid Algo [Khachian 1979]: Kann LPs in poly Zeit lösen.

LP - Zulässige Region



Region erfüllt

$x_1 \geqslant 0,\ x_2 \geqslant 0$

$x_1+2x_2 \geqslant 6$

$2x_1+x_2 \geqslant 6$

VC: ILP Methode

ILP zu VC:

$$\min \sum_i w_i x_i$$
$$\text{s. d. } x_i + x_j \geqslant 1 \text{ für alle } \{i,j\} \in E$$
$$x_i \in \{0,1\} \text{ für alle } i \in V$$

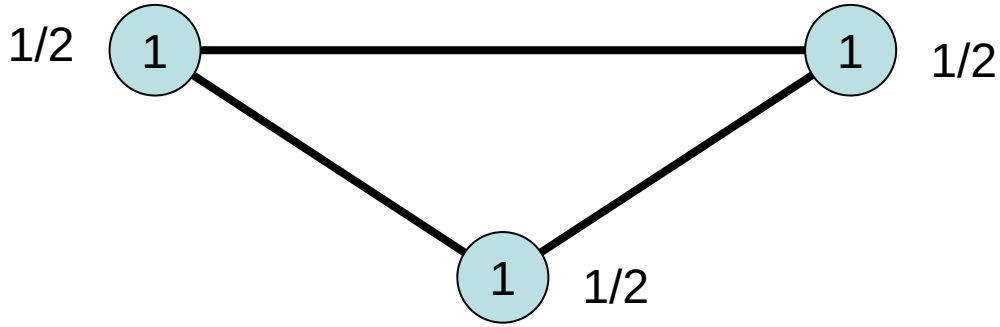
LP Relaxierung zu VC:

$$\min \sum_i w_i x_i$$
$$\text{s. d. } x_i + x_j \geqslant 1 \text{ für alle } \{i,j\} \in E$$
$$x_i \geqslant 0 \text{ für alle } i \in V$$

Beobachtung: $OPT_{LP} \leqslant OPT_{ILP}$

VC: ILP Methode

OPT_{LP} nicht äquivalent zu Knotenüberdeckung:



Wie kann uns ein LP helfen, um eine gute Knotenüberdeckung zu finden?

Idee: Löse LP und runde fraktionale Werte

Gewichtete Knotenüberdeckung

Theorem 12.16: Sei x^* eine optimale Lösung des LPs, dann ist $S=\{ i \in V \mid x^*_i \geqslant ½\}$ eine Knotenüberdeckung, dessen Gewicht höchstens doppelt so groß wie das minimale Gewicht ist.

Beweis:

(1) S ist eine Knotenüberdeckung:

- Betrachte eine Kante $\{i,j\} \in E$
- Da $x^*_i + x^*_j \geqslant 1$ ist, ist entweder $x^*_i \geqslant ½$ oder $x^*_j \geqslant ½$ $\Rightarrow \{i,j\}$ ist überdeckt

(2) $w(S) \leqslant 2w(OPT)$:

- Sei S^* die optimale Knotenüberdeckung. Dann ist

$$\sum_{i \in S^*} w_i \geqslant \sum_{i \in S} w_i x^*_i \geqslant (1/2) \sum_{i \in S} w_i$$

LP-Relaxierung; $x^*_i \geqslant 1/2$

Gewichtete Knotenüberdeckung

Korollar 12.17: Die ILP-Methode ergibt eine 2-Approximation der gewichteten Knotenüberdeckung.

Theorem 12.18 [Dinur-Safra 2001]: Solange nicht P=NP ist, gibt es keine ρ-Approximation mit $\rho<1{,}3607$, selbst wenn alle Knotengewichte 1 sind.

Offenes Forschungsproblem: schließe Lücke.

Übersicht

- P und NP
- Approximationsalgorithmen
- Lastbalancierung
- Zentrumswahl
- Knotenüberdeckung
- Allgemeine Lastbalancierung
- Rucksack-Problem
- Problem des Handlungsreisenden

Allgemeine Lastbalancierung

Eingabe: Menge M von m Maschinen, Menge J von n Jobs

- Job j muss in einem Stück auf einer der autorisierten Maschinen in $M_j \subset M$ ablaufen
- Job j hat Bearbeitungszeit t_j
- Jede Maschine kann höchstens einen Job gleichzeitig bearbeiten

Erinnerung:

- Last von Maschine i: $L_i = \sum_{j \in J(i)} t_j$
- Makespan $L = \max_i L_i$

Aufgabe: finde Jobzuweisung, die Makespan minimiert

Lastbalancierung: ILP-Methode

Sei $x_{I,j}$ Zeit, die Maschine i mit Job j zubringt.

ILP:

$\min L$

s. d. $\sum_i x_{i,j} = t_j$ für alle $j \in J$

$\sum_j x_{i,j} \leqslant L$ für alle $i \in M$

$x_{i,j} \in \{0,t_j\}$ für alle $j \in J$ und $i \in M_j$

$x_{i,j} = 0$ für alle $j \in J$ und $i \notin M_j$

LP:

$\min L$

s. d. $\sum_i x_{i,j} = t_j$ für alle $j \in J$

$\sum_j x_{i,j} \leqslant L$ für alle $i \in M$

$x_{i,j} \geqslant 0$ für alle $j \in J$ und $i \in M_j$

$x_{i,j} = 0$ für alle $j \in J$ und $i \in M_j$

Untere Schranken

Lemma 12.19: Sei L der optimale Wert des LPs. Dann gilt für den optimalen Makespan L^*, dass $L^* \geqslant L$.

Beweis: LP hat weniger Einschränkungen als ILP

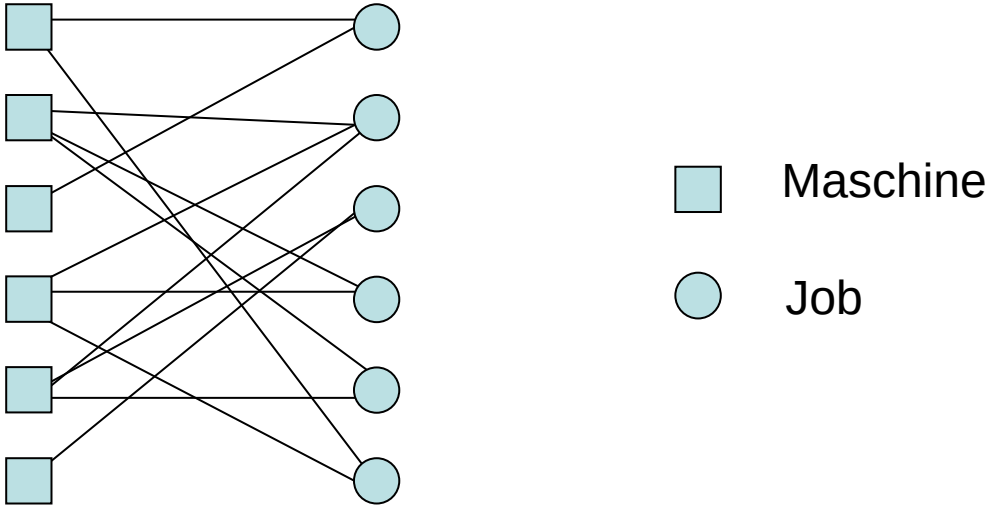
Lemma 12.20: Der optimale Makespan erfüllt $L^* \geqslant \max_j t_j$.

Beweis: klar

ILP-Rundung

Interpretiere LP-Lösung x als bipartiten Graph $G(x)$:

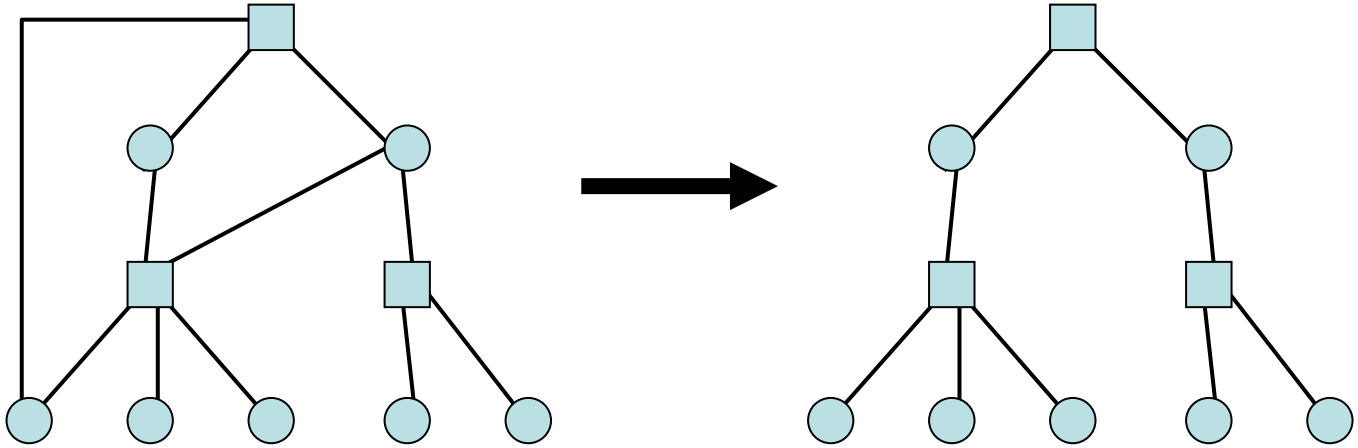
- Jobs und Maschinen sind Knoten in $G(x)$
- Job j und Maschine i besitzen eine Kante $\{i,j\} \Leftrightarrow x_{i,j}>0$



ILP-Rundung

Lemma 12.21: Sei x eine beliebige Lösung des LPs. Dann ist x transformierbar in ein x' gleicher Güte, so dass G(x') azyklisch ist.

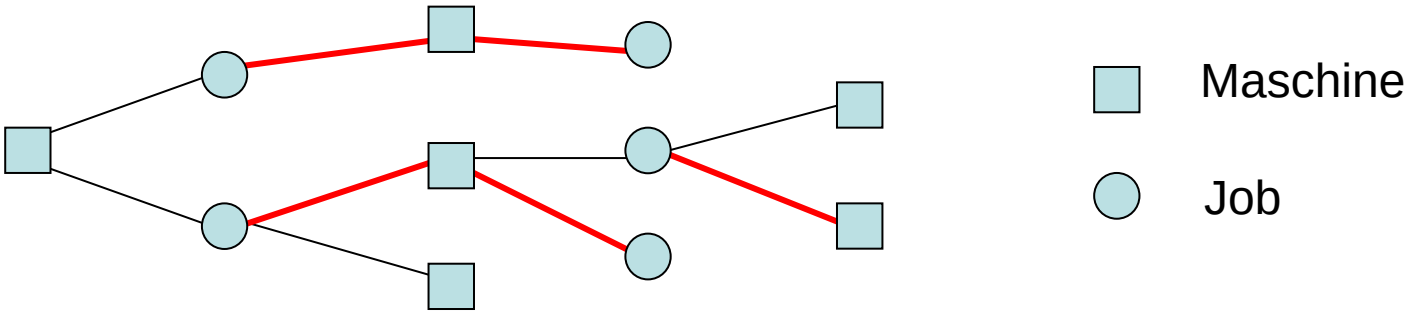
Beweis: Übung



ILP-Rundung

Annahme: G(x) ist azyklisch.

Zeichne G(x) von einer beliebigen Maschine r aus.



- Falls Job j ein Blatt ist, weise j seinem Vater zu.
- Sonse weise Job j einem seiner Kinder zu.

ILP-Rundung

Lemma 12.22: Falls Job j ein Blatt ist und Maschine i der Vater von j, dann $x_{i,j}=t_j$.

Beweis: Da j ein Blatt ist, ist $x_{i,j}=0$ für alle anderen Maschinen

Lemma 12.23: Höchstens ein innerer Knoten j wird einer Maschine zugewiesen.

Beweis: Rundungsregel; $G(x)$ ist azyklisch.

ILP-Rundung

Theorem 12.24: Die ILP-Rundung ergibt eine 2-Approximation.

Beweis:

- Sei J(i) die Menge der Jobs, die Maschine i zugewiesen werden und L die optimale Lösung des LPs.
- Nach Lemma 12.23 besteht die Last L_i von Maschine i aus zwei Komponenten:
  - Blättern:
    $$\sum_{j \in J(i),\ j \text{ Blatt}} t_j = \sum_{j \in J(i),\ j \text{ Blatt}} x_{i,j} \leqslant \sum_{j \in J(i)} x_{i,j} \leqslant L \leqslant L^*$$
  - Vätern:
    $$t_{parent(i)} \leqslant L^*$$
- Also ist die Gesamtlast $L_i \leqslant 2L^*$ für alle i.

Allgemeine Lastbalancierung

Laufzeit der 2-Approximation hängt von der Lösung eines LPs mit $mn+1$ Variablen ab.

Schnellere Lösung des LPs über Kombination von Flusstechniken (siehe EA II) und binäre Suche.

Erweiterungen: unrelated parallel maschines [Lenstra-Shmoys-Tardos 1990]

- Job j benötigt $t_{i,j}$ Zeit auf Maschine i
- 2-Approximation über LP-Rundung
- Keine 3/2-Approximation, es sei denn, P=NP

Übersicht

- P und NP
- Approximationsalgorithmen
- Lastbalancierung
- Zentrumswahl
- Knotenüberdeckung
- Allgemeine Lastbalancierung
- Rucksack-Problem
- Problem des Handlungsreisenden

Rucksack-Problem

Rucksack-Problem:

- Gegeben sind n Objekte und ein Rucksack
- Objekt i hat Wert $v_i>0$ und wiegt $w_i>0$
- Der Rucksack kann max. Gesamtgewicht W tragen.

Ziel: fülle Rucksack mit Objekten mit max. Gesamtwert

Beispiel: $W=11$

{3,4} hat Wert 40

Objekt	Wert	Gewicht
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

RP: Greedy Verfahren

Greedy Strategie:

- Berechne Profitdichten $d_1=v_1/w_1,..,\ d_n=v_n/w_n$
- Sortiere Objekte nach Profitdichten
- Angefangen von dem Objekt mit höchster Profitdichte, füge Objekte zu Rucksack hinzu, bis kein Platz mehr da

Problem: Greedy Strategie kann weit vom Optimum entfernt liegen

RP: Greedy Verfahren

Beispiel: zwei Objekte mit $v_1=1$ und $v_2=W-1$ und $w_1=1$ und $w_2=W$, Rucksackkapazität ist W

Greedy-Methode: berechnet $d_1=1$ und $d_2 = 1-1/W$ und wird nur Objekt 1 in Rucksack packen, da Objekt 2 nicht mehr passt

Optimale Lösung: packe Objekt 2 in Rucksack (viel besser da Wert $W-1$ statt 1)

RP: Greedy Verfahren

Verbesserte Greedy-Methode:

- Seien die Objekte 1 bis n absteigend nach Profitdichte sortiert
- Bestimme maximale Objektmenge {1,…,i} wie bisher mit $\sum_{j \leqslant i} w_j \leqslant W$
- Gib entweder {1,…,i} oder {i+1} aus, je nachdem, welche Menge den maximalen Wert hat

RP: Greedy Verfahren

Theorem 12.25: Die Lösung der verbesserten Greedy-Methode ist höchstens einen Faktor 2 von der optimalen Lösung entfernt.

Beweis:

- Wenn beliebige Bruchteile der Objekte gewählt werden könnten, wäre die optimale Lösung $\{1,\ldots,i+1\}$, wobei von Objekt $i+1$ nur der Bruchteil genommen wird, der noch in den Rucksack passt.
- Für den optimalen Wert OPT gilt demnach: $OPT \leqslant \sum_{j \leqslant i+1} w_j$.
- Also ist $\max\{\sum_{j \leqslant i} w_j, w_{i+1}\} \geqslant OPT/2$

PTAS und FPTAS

PTAS (polynomial time approximation scheme): $(1+\varepsilon)$-Approximationsalgo mit $O(poly(n))$ Laufzeit für jede Konstante $\varepsilon>0$

FPTAS (fully PTAS): PTAS mit Laufzeit $O(poly(1/\varepsilon, n))$

Beispiele:

- Laufzeit $O(n^{1/\varepsilon})$: PTAS, aber kein FPTAS
- Laufzeit $O(n/\varepsilon)$: FPTAS

(F)PTAS kann Lösung beliebig guter Qualität liefern, aber auf Kosten einer zunehmenden Laufzeit.

Hier: FPTAS für das Rucksack-Problem

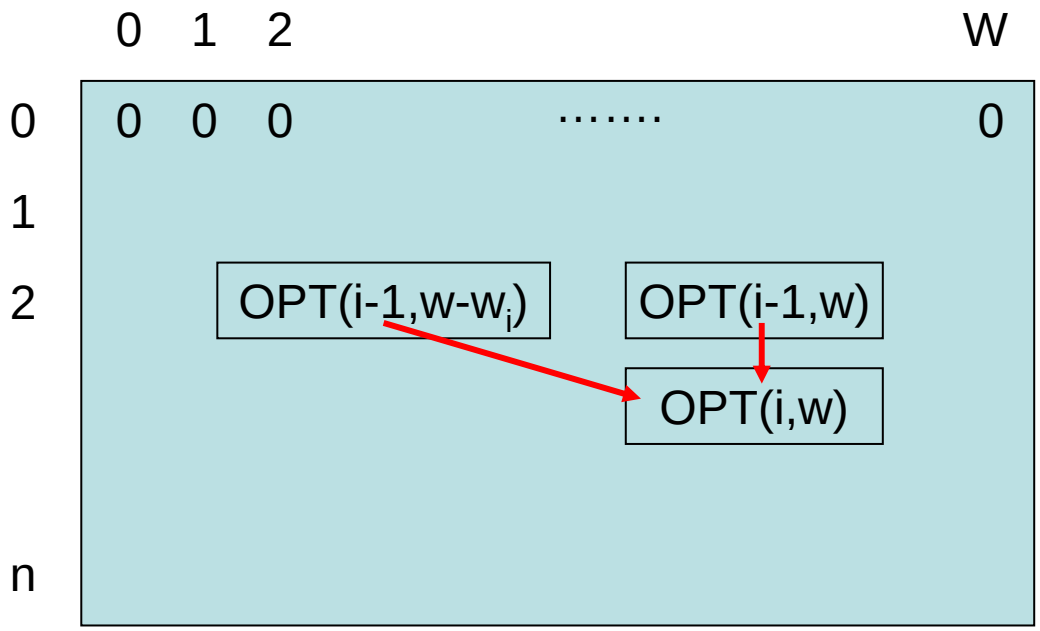
RP: FPTAS

Sei OPT(i,w) = max. Wert einer Teilmenge von Objekten aus {1,…,i} mit max. Gewicht w

- Fall 1: OPT wählt nicht Objekt i
  OPT wählt beste Teilmenge aus {1,…,i-1} mit max. Gewicht w
- Fall 2: OPT wählt Objekt i
  OPT wählt beste Teilmenge aus {1,…,i-1} mit max. Gewicht $w-w_i$

$$OPT(i,w)= \begin{cases} 0 & \text{falls } i=0 \\ OPT(i-1,w) & \text{falls } w_i>w \\ \max\{OPT(i-1,w),\ v_i+OPT(i-1,w-w_i)\} & \text{sonst} \end{cases}$$

RP: FPTAS



Laufzeit: $O(n \cdot W)$, nicht polynomiell in n!

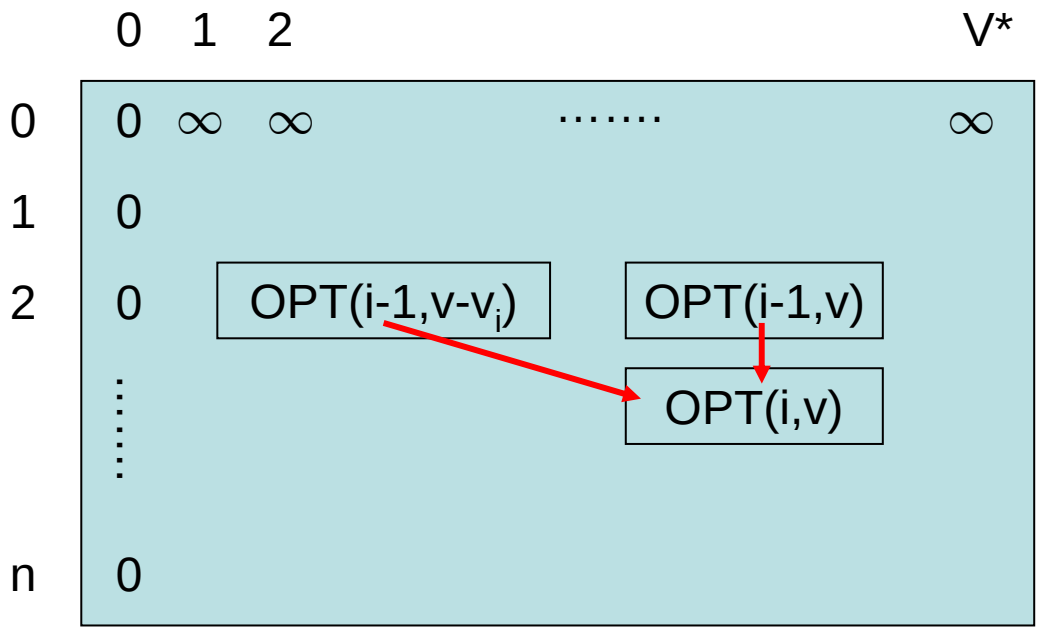
RP: FPTAS

Sei OPT(i,v) = min. Gewicht einer Teilmenge von Objekten aus {1,…,i} mit Wert exakt v

- Fall 1: OPT wählt nicht Objekt i
  OPT wählt beste Teilmenge aus {1,…,i-1} mit Wert v
- Fall 2: OPT wählt Objekt i
  OPT wählt beste Teilmenge aus {1,…,i-1} mit Wert $v-v_i$

$$OPT(i,w)=\begin{cases} 0 & \text{falls } v=0 \\ \infty & \text{falls } i=0,\ v>0 \\ OPT(i-1,v) & \text{falls } v_i>v \\ \min\{OPT(i-1,v),\ w_i+OPT(i-1,v-v_i)\} & \text{sonst} \end{cases}$$

RP: FPTAS



V*: max. Wert v, so dass OPT(n,v) ⩽ W.

Laufzeit: $O(n \cdot V^*) = O(n^2 v_{max})$, zu groß!

RP: FPTAS

Idee für FPTAS:

- Runde alle Werte auf, um kleineren Wertebereich zu erhalten.
- Verwende vorigen Algo auf gerundeten Werten.
- Gib optimale Objekte der gerundeten Werte aus.

Objekt	Wert	Gewicht
1	934,221	1
2	5956,34	2
3	17810,1	5
4	21217,8	6
5	27343,2	7

Objekt	Wert	Gewicht
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

RP: FPTAS

- v_{max}: max. Wert in Originalinstanz
- ε: Präzision
- $\theta = \varepsilon\, v_{max}/n$: Skalierungsfaktor

Betrachte $\bar{v}_i = \lceil v_i/\theta \rceil \theta$ und $\check{v}_i = \lceil v_i/\theta \rceil$.

Beobachtung: Optimale Lösungen zu $\bar{v}$ und $\check{v}$ sind äquivalent.

FPTAS: verwende $\check{v}$-Werte (nur $O(n/\varepsilon)$ viele)

Damit Laufzeit $O(n^3/\varepsilon)$.

RP:FPTAS

Theorem 12.26: Sei S die Lösung unseres Algos und S^* eine beliebige andere zulässige Lösung. Dann ist $(1+\varepsilon)\sum_{i\in S} v_i \geqslant \sum_{i\in S^*} v_i$.

Beweis:

Es gilt:

$\sum_{i\in S^*} v_i \leqslant \sum_{i\in S^*} \overline{v}_i$	runde immer auf
$\leqslant \sum_{i\in S} \overline{v}_i$	finde opt. Lösung
$\leqslant \sum_{i\in S} (v_i+\theta)$	runde nie mehr als θ
$\leqslant \sum_{i\in S} v_i + n\cdot\theta$	$\lvert S\rvert\leqslant n$
$\leqslant (1+\varepsilon)\sum_{i\in S} v_i$	$n\theta = \varepsilon\, v_{max} \leqslant \varepsilon \sum_{i\in S} v_i$

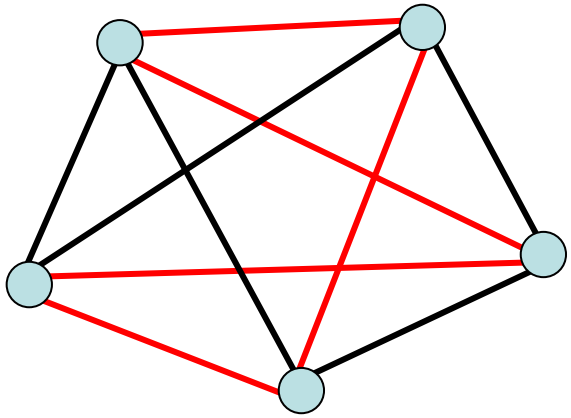
Übersicht

- P und NP
- Approximationsalgorithmen
- Lastbalancierung
- Zentrumswahl
- Knotenüberdeckung
- Allgemeine Lastbalancierung
- Rucksack-Problem
- Problem des Handlungsreisenden

Problem des Handlungsreisenden

Problem des Handlungsreisenden (TSP):

Gegeben eine Knotenmenge V mit Distanzfunktion $d: V \times V \rightarrow \mathbb{R}^+$, finde einen einfachen Kreis durch alle Knoten (Hamilton-Kreis) mit minimaler Länge.



Problem des Handlungsreisenden

Theorem 12.27: TSP hat keine ρ-Approximation für alle Konstanten $\rho>1$, es sei denn, P=NP.

Beweis:

- Betrachte das Hamilton-Kreis (HK) Problem: Gegeben ein ungerichteter Graph G=(V,E), entscheide, ob G einen Hamilton-Kreis besitzt oder nicht.
- Es ist bekannt, dass es NP-hart ist, das HK-Problem zu entscheiden.
- Wir wollen mithilfe des HK-Problems zeigen, dass dann auch Theorem 12.27 gilt.

Problem des Handlungsreisenden

Beweis (Fortsetzung):

- Betrachte eine beliebige Konstante $\rho > 1$.
- Für eine Instanz $G=(V,E)$ des HK-Problems definieren wir eine Distanzfunktion d mit

$$d(v,w) = \begin{cases} 1 & \text{falls } \{v,w\} \in E \\ \rho\, n & \text{sonst} \end{cases}$$

- Falls G einen Hamilton-Kreis hat, dann gibt es eine Lösung des TSPs (V,d) der Länge n.
- Hat G keinen Hamilton-Kreis, muss jede Lösung des TSPs (V,d) mindestens eine Kante $\{v,w\}$ verwenden, die nicht in E ist, kostet damit also mehr als ρn.

Problem des Handlungsreisenden

Beweis (Fortsetzung):

- Angenommen, es gibt eine ρ-Approximation A zum TSP. Dann verwende den folgenden Algorithmus für das HK-Problem:

  Gegeben ein Graph G=(V,E), konstruiere (V,d) wie oben und wende A darauf an. Falls A einen Kreis zurückgibt mit Länge $\leqslant \rho n$, gib “Ja” aus und sonst “Nein”.

- Dieser Algorithmus könnte dann das HK-Problem in polynomieller Zeit entscheiden, ein Widerspruch zur Tatsache, dass das HK-Problem NP-hart ist!
- Also kann es keine ρ-Approximation für das TSP geben, es sei denn, P=NP.

Problem des Handlungsreisenden

Eine natürliche Einschränkung des TSPs ist das folgende Problem:

Δ–TSP: Gegeben eine Knotenmenge V mit Metrik d, bestimme einen Hamilton-Kreis C minimaler Länge.

Dieses Problem ist immer noch NP-hart, kann aber mit konstanter Güte approximiert werden.

Problem des Handlungsreisenden

Definition 12.28: Ein (Multi-)Graph G heißt Eulersch, wenn es in G einen Kreis gibt, der jede Kante von G genau einmal durchläuft (ein sogenannter Euler-Kreis).

Theorem 12.29: G ist Eulersch genau dann, wenn G zusammen-hängend ist und jeder seiner Knoten geraden Grad hat.

Beweis:

$\Rightarrow$: klar

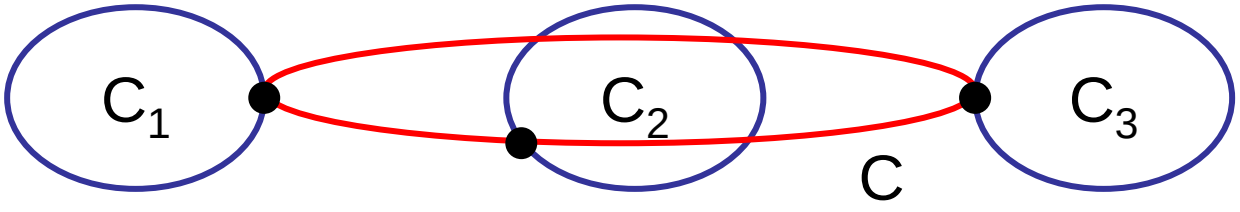
$\Leftarrow$: durch Induktion über die Kantenzahl

- Da alle Knoten Grad $\geqslant 2$ haben, gibt es einen Kreis C in G. (Fange bei beliebigem Knoten an und wandere entlang beliebiger Kanten, bis ein Knoten zum zweitenmal besucht wird.)

Problem des Handlungsreisenden

Beweis (Fortsetzung):

- Seien $G_1,\ldots,G_k$ die ZHKs von G nach Entfernung der Kanten in C. Da C ein Kreis ist, sind die Knotengrade in jedem G_i gerade. Nach der Induktion gibt es also einen Euler-Kreis C_i in jedem G_i
- Diese Euler-Kreise und Kreis C können leicht zu einem Euler-Kreis von G zusammengesetzt werden.



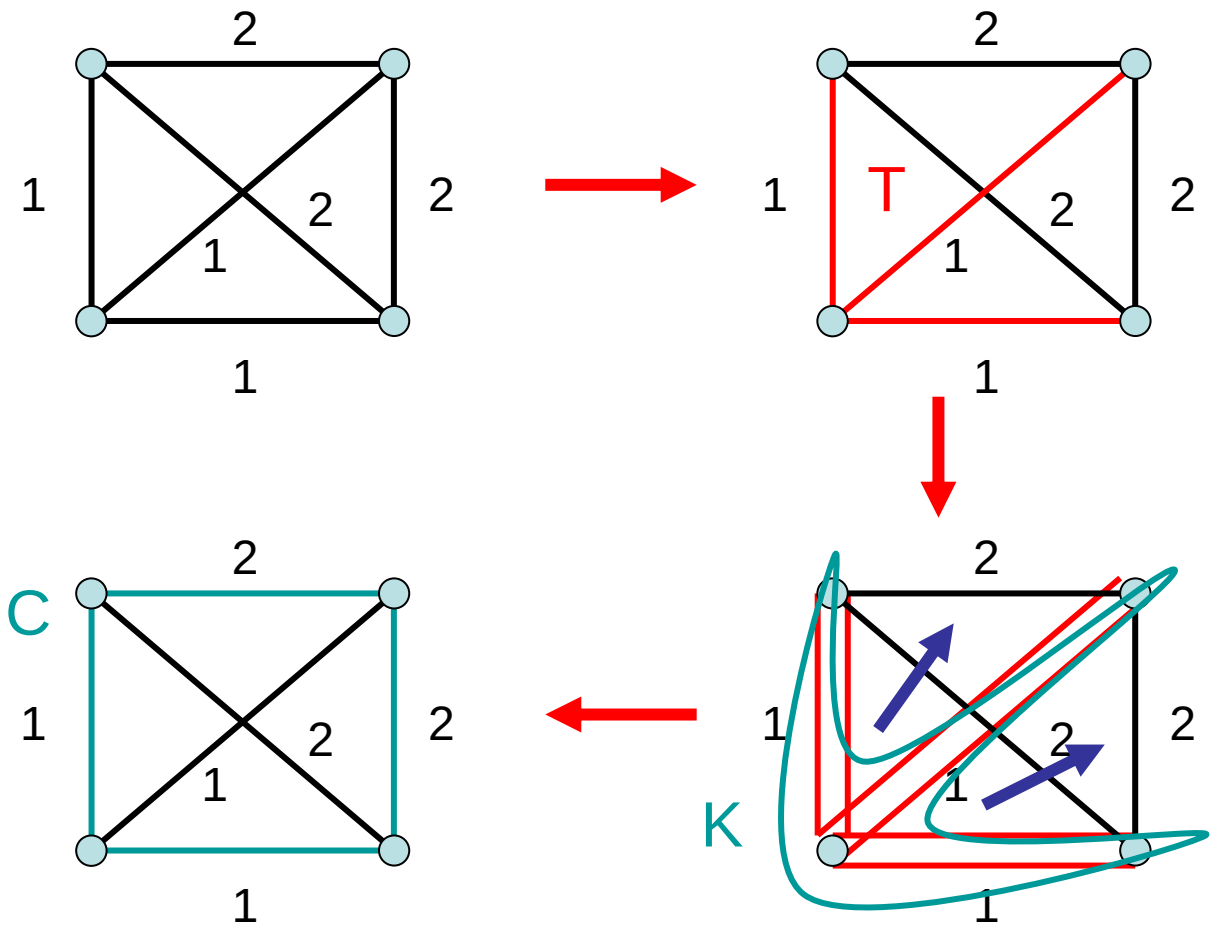
Problem des Handlungsreisenden

Betrachte nun den folgenden Algorithmus für das TSP-Problem:

T-Algorithmus:

1. Berechne einen minimalen Spannbaum T von (V,d).
2. Verdopple alle Kanten in T zum Multigraph T*.
3. Konstruiere einen Euler-Kreis K von T*.
4. Entferne in K Wiederholungen von Knoten, so dass sich ein Hamilton-Kreis C ergibt.
5. Gib C aus.

Problem des Handlungsreisenden



Problem des Handlungsreisenden

Theorem 12.30: Der T-Algorithmus ist eine 2-Approximation des Δ-TSPs.

Beweis:

- Der T-Algorithmus ist in poly. Zeit implementierbar.
- Für eine Kantenmenge E sei $w(E) = \sum_{e \in E} d(e)$.
- Da d eine Metrik ist, gilt $w(C) \leqslant w(K)$
  (C entsteht ja aus K über “Abkürzungen”.)
- Nach der Konstruktion von T^* gilt $w(K) = w(T^*) = 2 \cdot w(T)$
- Sei C^* die optimale Lösung des Δ-TSPs. Löschen wir eine Kante $\{v,w\}$ in C^*, so ist C^* ein Spannbaum in G. Also gilt: $w(T) \leqslant w(C^* \setminus \{v,w\}) \leqslant w(C^*)$
- Damit folgt $w(C) \leqslant 2 \cdot w(C^*)$

Problem des Handlungsreisenden

Ist das Verfahren des T-Algorithmus, einen Euler-Kreis zu erzeugen, das bestmögliche?

Nein. Christofides Algorithmus verwendet ein besseres Verfahren.

Lemma 12.31:

- Für alle Graphen $G=(V,E)$ gilt $\sum_v d(v) = 2|E|$. Insbesondere ist die Anzahl der ungeraden Knoten gerade.
- Für jedes (V,d) mit geradem $|V|$ kann ein perfektes Matching mit minimalen Kosten in $O(n^3)$ Zeit gefunden werden (zeigen wir hier nicht).

Problem des Handlungsreisenden

Christofides Algorithmus:

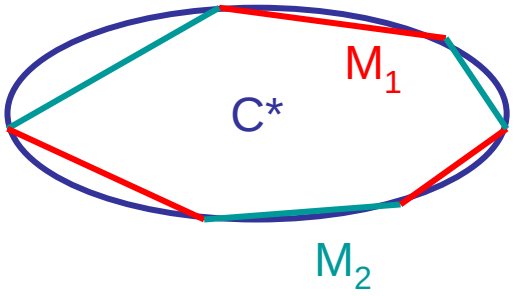
1. Berechne einen MSB T zu (V,d).
2. Bestimme die Menge U der Knoten ungeraden Grades in T.
3. Berechne ein perfektes Matching M minimalen Gewichts für (U,d).
4. Sei $G=(V, E(T) \cup M)$. Konstruiere in G einen Euler-Kreis K.
5. Entferne in K Wiederholungen, so dass Hamilton-Kreis C verbleibt.
6. Gib C aus.

Problem des Handlungsreisenden

Theorem 12.32: Christofides Algorithmus ist eine 3/2-Approximation des Δ-TSP.

Beweis:

- Wie für den Beweis des T-Algorithmus gilt
  $$w(C) \leqslant w(K) = w(G) = w(T)+w(M)$$
- Zu zeigen: $w(M) \leqslant (1/2)w(C^*)$ für einen optimalen Hamilton-Kreis C^*.
- Sei $C^*=(v_1,v_2,\ldots,v_n)$ und $U=(v_{i_1},\ldots,v_{i_m})$. Betrachte die Matchings
  $M_1 = \{ \{v_{i_1},v_{i_2}\}, \{v_{i_3},v_{i_4}\},\ldots,\{v_{i_{m-1}},v_{i_m}\} \}$
  $M_2 = \{ \{v_{i_2},v_{i_3}\}, \{v_{i_4},v_{i_5}\},\ldots,\{v_{i_m},v_{i_1}\} \}$
- Wegen der Dreiecksungleichung gilt
  $$w(M_1)+w(M_2) \leqslant w(C^*)$$
- Aus der Minimalität von M folgt:
  $$2 \cdot w(M) \leqslant w(M_1)+w(M_2) \leqslant w(C^*)$$



- Also ist
  $w(C) \leqslant w(T)+w(M) \leqslant w(C^*) + (1/2)w(C^*) = (3/2)w(C^*)$

Problem des Handlungsreisenden

Das Δ-TSP besitzt kein PTAS (es sei denn, P=NP) und der CH-Algorithmus ist zurzeit der beste bekannte Approximationsalgorithmus für das Δ-TSP.

Es wird jedoch vermutet, dass eine 4/3-Approximation für das Δ-TSP existiert.

Definition 12.33: Das Euklidische TSP ist die Einschränkung des Δ-TSP auf $\mathbb{R}^k$ für ein festes $k \in \mathbb{N}$, wobei d die Euklidische Distanz ist.

Theorem 12.34: Das Euklidische TSP besitzt ein PTAS. Das Euklidische TSP für $\mathbb{R}^2$ sogar ein FPTAS.

Ausblick

Das war’s!