Karlsruher Institut für Technologie

Algorithmen II
Vorlesung am 07.01.2014

Approximierende Algorithmen: Grundlagen, Knapsack, TSP

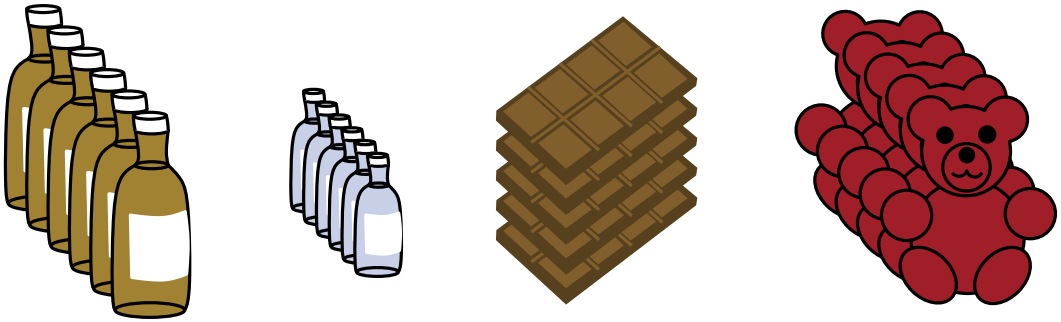
INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

Einführung

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

KNAPSACK

Menge $M = \{1, \ldots, n\}$ mit n Gegenständen



Rucksack mit Maximalgewicht W

Jeder Gegenstand i

- hat Kosten (eigentlich Nutzen) $c_i \in \mathbb{N}_0$.
- hat ein Gewicht $w_i \in \mathbb{N}$.
- kann beliebig oft eingepackt werden.

Problem: KNAPSACK

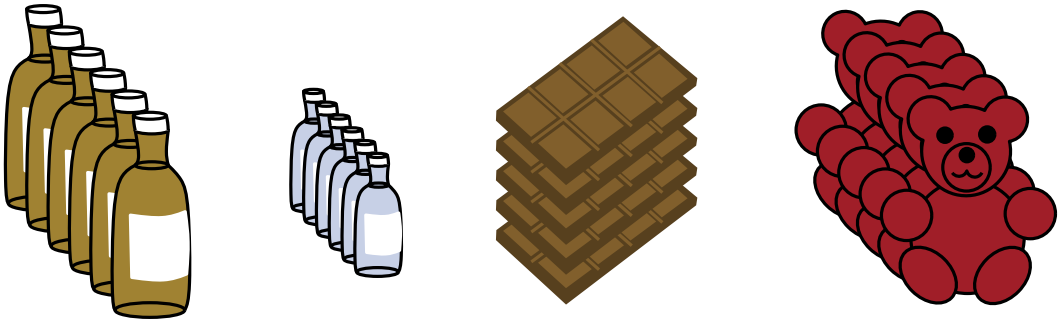
Finde Anzahlen $x_1, \ldots, x_n \in \mathbb{N}_0$, sodass $\sum_{i=0}^{n} x_i w_i \leq W$ und $\sum_{i=1}^{n} x_i c_i$ maximal ist.

Gesamtgewicht Gesamtkosten

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

KNAPSACK

Menge $M = \{1, \ldots, n\}$ mit n Gegenständen



Rucksack mit Maximalgewicht W

Jeder Gegenstand i

- hat Kosten (eigentlich Nutzen) $c_i \in \mathbb{N}_0$.
- hat ein Gewicht $w_i \in \mathbb{N}$.
- kann beliebig oft eingepackt werden.

Problem: KNAPSACK

Finde Anzahlen $x_1, \ldots, x_n \in \mathbb{N}_0$, sodass $\sum_{i=0}^{n} x_i w_i \leq W$ und $\sum_{i=1}^{n} x_i c_i$ maximal ist.

Gesamtgewicht Gesamtkosten

- KNAPSACK ist NP-schwer.
- Damit kann KNAPSACK (wie viele andere Probleme) vermutlich nicht effizient und optimal gelöst werden.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Approximation

Lösungsansatz für NP-schwere Probleme

- Finde Algorithmus, der effizient ist, aber ggf. nicht die optimale Lösung liefert.
- Der Algorithmus soll trotzdem eine beweisbar gute Lösung liefern.

Wie misst man das?

Approximation

Lösungsansatz für NP-schwere Probleme

- Finde Algorithmus, der effizient ist, aber ggf. nicht die optimale Lösung liefert.
- Der Algorithmus soll trotzdem eine beweisbar gute Lösung liefern.

Wie misst man das?

Definition: Absoluter Approximationsalgorithmus **(Definition 4.36)**

Sei Π ein Optimierungsproblem. Ein polynomieller Algorithmus $\mathcal{A}$, der für jede Instanz I einen Wert $\mathcal{A}(I)$ liefert, mit

$$|\textsc{Opt}(I) - \mathcal{A}(I)| \leq K$$

und $K \in \mathbb{N}_0$ konstant, heißt *Approximationsalgorithmus mit Differenzengarantie* oder *absoluter Approximationsalgorithmus*.

Approximation

Lösungsansatz für NP-schwere Probleme

- Finde Algorithmus, der effizient ist, aber ggf. nicht die optimale Lösung liefert.
- Der Algorithmus soll trotzdem eine beweisbar gute Lösung liefern.

Wie misst man das?

Definition: Absoluter Approximationsalgorithmus **(Definition 4.36)**

Sei Π ein Optimierungsproblem. Ein polynomieller Algorithmus $\mathcal{A}$, der für jede Instanz I einen Wert $\mathcal{A}(I)$ liefert, mit

$$|\textsc{Opt}(I) - \mathcal{A}(I)| \leq K$$

und $K \in \mathbb{N}_0$ konstant, heißt *Approximationsalgorithmus mit Differenzengarantie* oder *absoluter Approximationsalgorithmus*.

Satz: Negatives Ergebnis *(ohne Beweis)* **(Satz 4.37)**

Falls $\mathcal{P} \neq \mathcal{NP}$, so gibt es keinen absoluten Approximationsalgo für KNAPSACK.

Für die meisten NP-schweren Probleme gibt es keinen Approximationsalgorithmus mit absoluter Güte.

Relative Approximation – KNAPSACK

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Approximationsalgorithmen

Definition: Approximationsalgorithmus **(Definition 4.39)**

Sei Π ein Optimierungsproblem, sowie $\mathcal{A}$ ein polynomieller Algorithmus, der für jede Instanz I von Π eine Lösung mit Wert $\mathcal{A}(I)$ liefert.

$$\text{Sei } \mathcal{R}_{\mathcal{A}}(I) = \begin{cases} \frac{\mathcal{A}(I)}{\text{OPT}(I)} & \text{falls } \Pi \text{ Minimierungsproblem} \\ \frac{\text{OPT}(I)}{\mathcal{A}(I)} & \text{falls } \Pi \text{ Maximierungsproblem} \end{cases}$$

Wenn es einen Konstante K gibt, sodass $\mathcal{R}_{\mathcal{A}}(I) \leq K$ für alle Instanzen I, dann ist $\mathcal{A}$ ein *Approximationsalgorithmus mit relativer Gütegarantie*.

Definition: Gütegarantie **(Definition 4.39)**

Die *Approximationsgüte* $\mathcal{R}_{\mathcal{A}}$ eines Approximationsalgorithmus ist definiert als

$$\mathcal{R}_{\mathcal{A}} = \inf\{r \geq 1 \mid \mathcal{R}_{\mathcal{A}}(I) \leq r \text{ für alle Instanzen } I \text{ von } \Pi\}.$$

$\mathcal{A}$ heißt ε*-approximierend*, falls $\mathcal{R}_{\mathcal{A}} \leq 1 + \varepsilon$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Greedy-Algorithmus für KNAPSACK

Idee: Wähle Elemente mit möglichst hoher Gewichtsdichte.

GREEDY KNAPSACK(M, $\{c_1, \ldots, c_n\}$, $\{w_1, \ldots, w_n\}$, W) $O(n \log n)$

for $i \in M$ **do**
  $p_i \leftarrow \frac{c_i}{w_i}$

sortiere nach Gewichtsdichte

Sortiere Gegenstände, sodass $p_1 \geq p_2 \geq \cdots \geq p_n$.

for $i \in M$ **do**
  $x_i \leftarrow \left\lfloor \frac{W}{w_i} \right\rfloor$
  $W \leftarrow W - x_i \cdot w_i$

wähle bevorzugt dichte Elemente

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Greedy-Algorithmus für KNAPSACK

Idee: Wähle Elemente mit möglichst hoher Gewichtsdichte.

GREEDY KNAPSACK(M, $\{c_1, \ldots, c_n\}$, $\{w_1, \ldots, w_n\}$, W) $O(n \log n)$

for $i \in M$ **do**
  $p_i \leftarrow \frac{c_i}{w_i}$

sortiere nach Gewichtsdichte

Sortiere Gegenstände, sodass $p_1 \geq p_2 \geq \cdots \geq p_n$.

for $i \in M$ **do**
  $x_i \leftarrow \left\lfloor \frac{W}{w_i} \right\rfloor$
  $W \leftarrow W - x_i \cdot w_i$

wähle bevorzugt dichte Elemente

Satz: Approximation **(Satz 4.40)**

Sei $\mathcal{A}$ der Algorithmus GREEDY KNAPSACK. $\mathcal{A}$ hat Approximationsgüte $\mathcal{R}_{\mathcal{A}} \leq 2$.

Greedy-Algorithmus für KNAPSACK

Idee: Wähle Elemente mit möglichst hoher Gewichtsdichte.

GREEDY KNAPSACK(M, $\{c_1, \ldots, c_n\}$, $\{w_1, \ldots, w_n\}$, W) $O(n \log n)$

for $i \in M$ **do**
  $p_i \leftarrow \frac{c_i}{w_i}$

sortiere nach Gewichtsdichte

Sortiere Gegenstände, sodass $p_1 \geq p_2 \geq \cdots \geq p_n$.

for $i \in M$ **do**
  $x_i \leftarrow \left\lfloor \frac{W}{w_i} \right\rfloor$
  $W \leftarrow W - x_i \cdot w_i$

wähle bevorzugt dichte Elemente

Satz: Approximation **(Satz 4.40)**

Sei $\mathcal{A}$ der Algorithmus GREEDY KNAPSACK. $\mathcal{A}$ hat Approximationsgüte $\mathcal{R}_{\mathcal{A}} \leq 2$.

Beweis: O.B.d.A. sei $w_1 \leq W$.

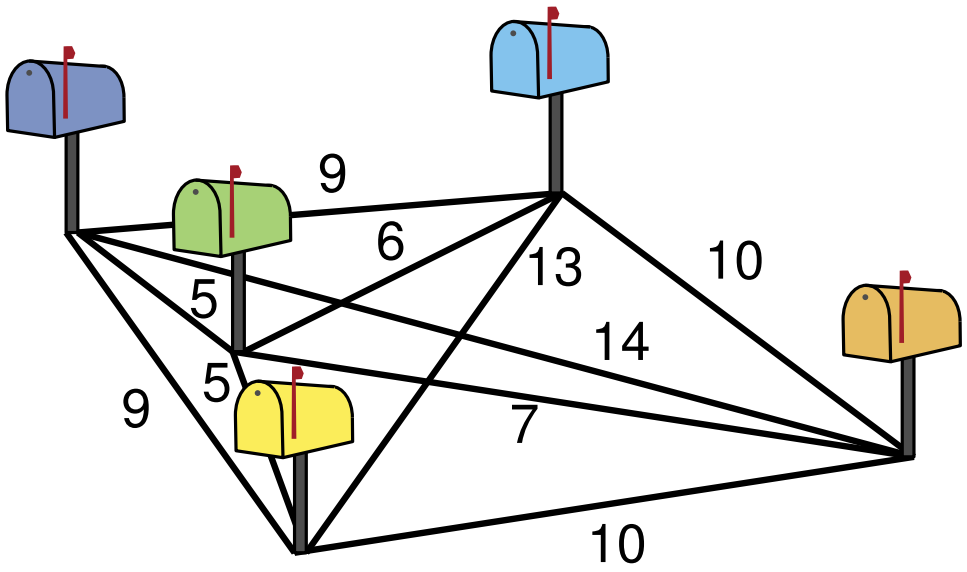
- Für alle Instanzen I gilt: $\mathcal{A}(I) \geq c_1 \cdot x_1 = c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor$
- Außerdem gilt: $\text{OPT}(I) \leq c_1 \cdot \frac{W}{w_1} \leq c_1 \cdot \left(\left\lfloor \frac{W}{w_1} \right\rfloor + 1 \right) \leq 2 \cdot c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor \leq 2 \cdot \mathcal{A}(I)$
- $\Rightarrow \mathcal{R}_{\mathcal{A}}(I) = \frac{\text{OPT}(I)}{\mathcal{A}(I)} \leq 2.$

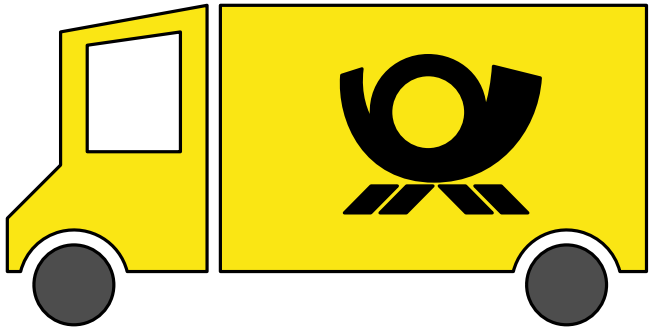
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Relative Approximation – TSP

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

TSP

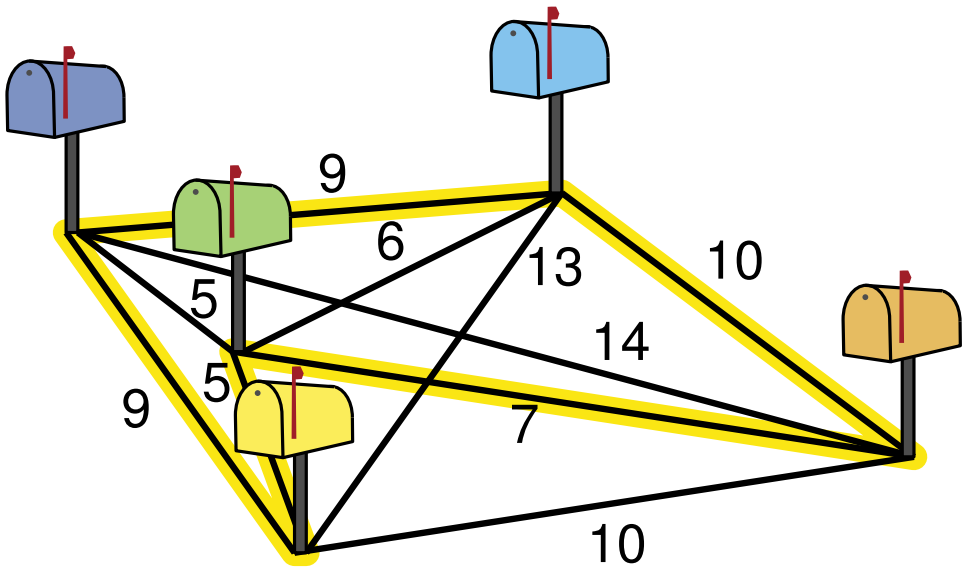


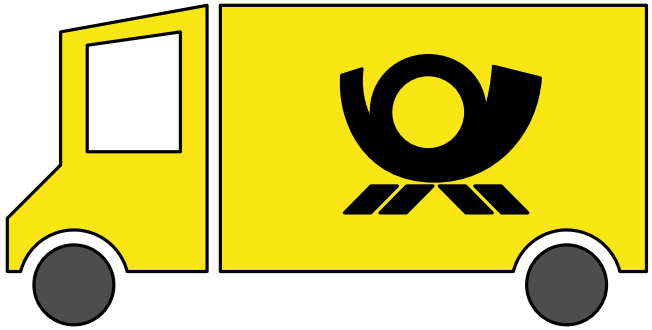


Vollständiger Graph $G = (V, E)$ mit Gewichtsfunktion $c\colon E \to \mathbb{N}$

Problem: TSP (TRAVELLING SALESMAN PROBLEM)
Finde eine kürzeste *Rundreise* (ein kürzester Kreis, der alle Knoten besucht).

TSP



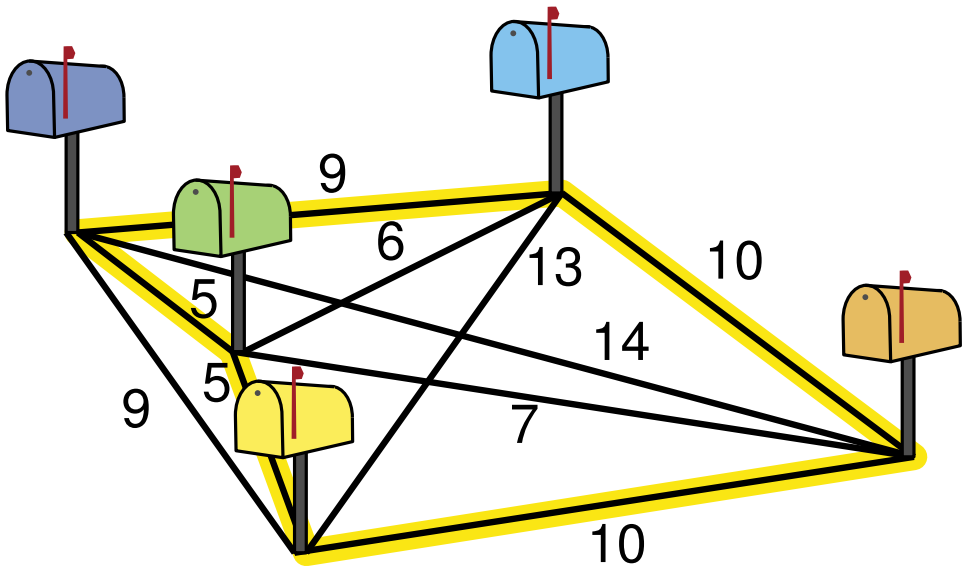


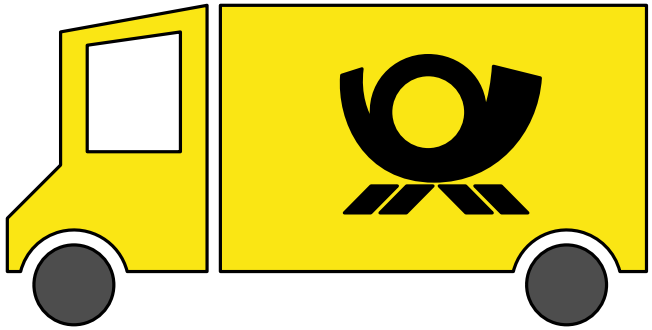
Gewicht: 40

Vollständiger Graph $G = (V, E)$ mit Gewichtsfunktion $c\colon E \to \mathbb{N}$

Problem: TSP (TRAVELLING SALESMAN PROBLEM)
Finde eine kürzeste *Rundreise* (ein kürzester Kreis, der alle Knoten besucht).

TSP



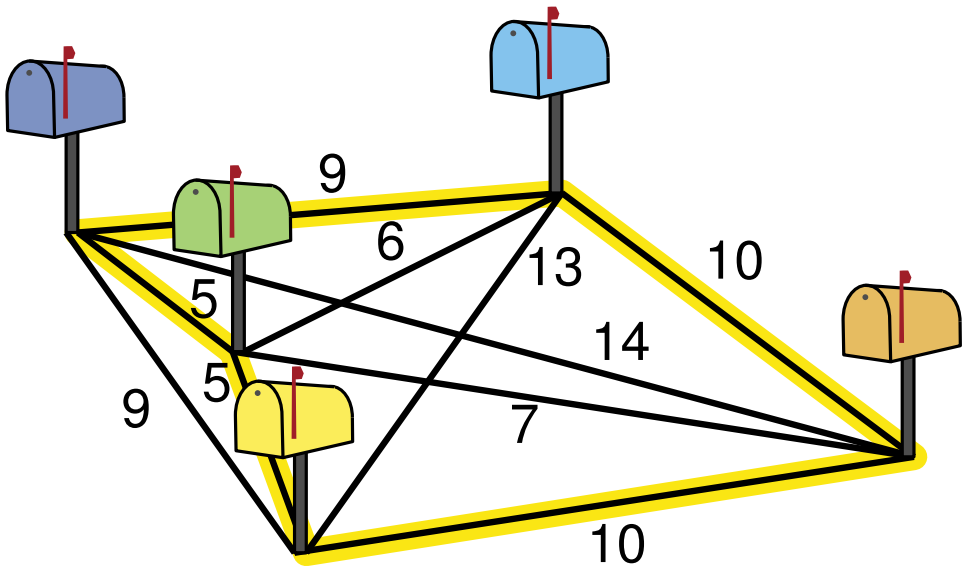


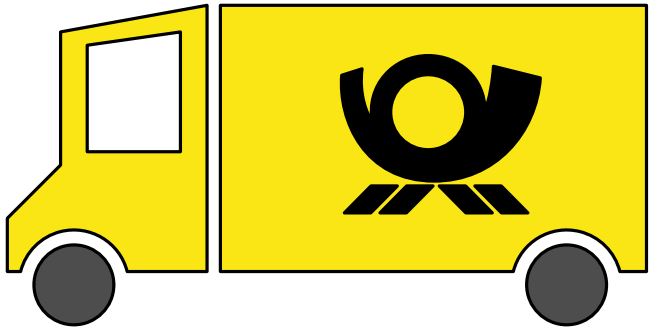
Gewicht: 39

Vollständiger Graph $G = (V, E)$ mit Gewichtsfunktion $c\colon E \to \mathbb{N}$

Problem: TSP (TRAVELLING SALESMAN PROBLEM)
Finde eine kürzeste *Rundreise* (ein kürzester Kreis, der alle Knoten besucht).

TSP



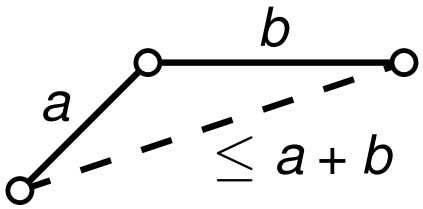


Gewicht: 39

Vollständiger Graph $G = (V, E)$ mit Gewichtsfunktion $c\colon E \to \mathbb{N}$

Problem: TSP (TRAVELLING SALESMAN PROBLEM)
Finde eine kürzeste *Rundreise* (ein kürzester Kreis, der alle Knoten besucht).

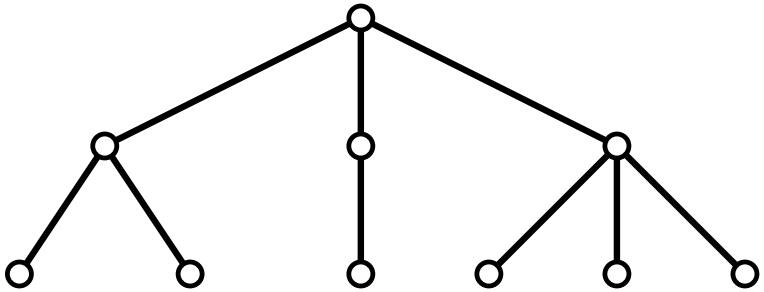
Einschränkung: Wir nehmen an, dass die Dreiecksungleichung gilt.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

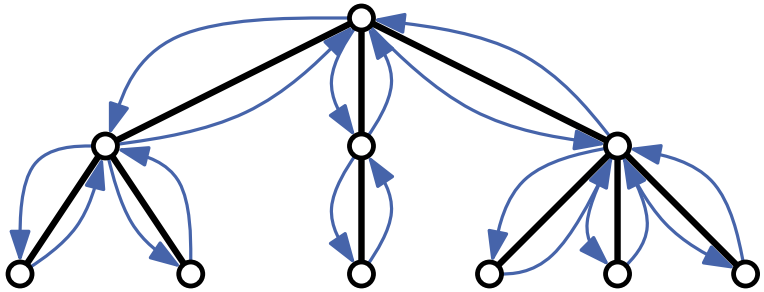
2-Approximation

Schritt 1



Berechne minimalen Spannbaum des Eingabegraphen

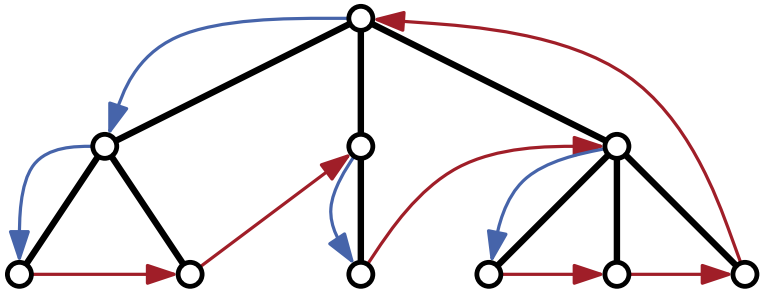
Schritt 2



Tiefensuch-Tour T durch den MST

Unschön: Tour ist kein einfacher Kreis

Schritt 3

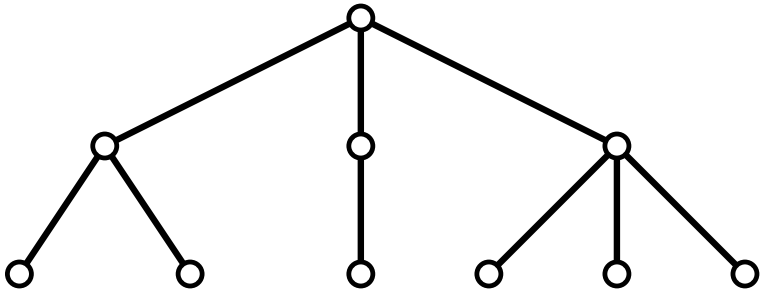


TSP-Tour als abgekürzte Tiefensuch-Tour T'

(Schon besuchte Knoten werden einfach übersprungen.)

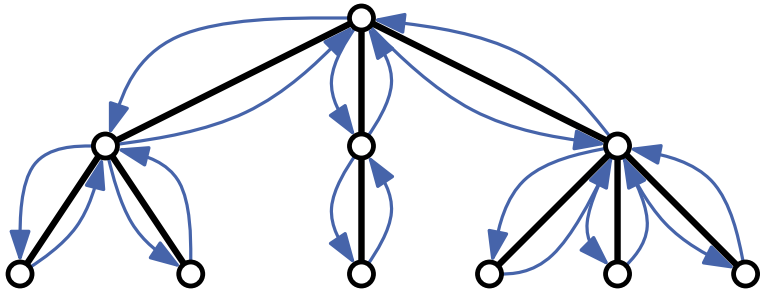
2-Approximation

Schritt 1



Berechne minimalen Spannbaum des Eingabegraphen

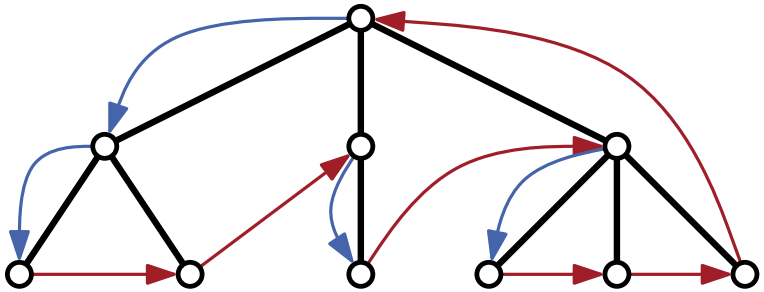
Schritt 2



Tiefensuch-Tour T durch den MST

Unschön: Tour ist kein einfacher Kreis

Schritt 3



TSP-Tour als abgekürzte Tiefensuch-Tour T'

(Schon besuchte Knoten werden einfach übersprungen.)

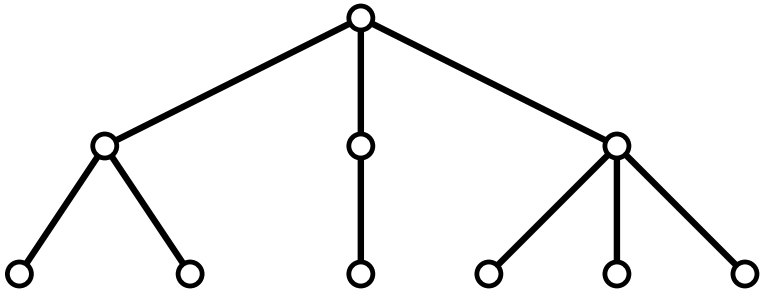
Satz: Approximation **(Satz 4.43)**

Für TSP existiert ein Approximationsalgorithmus $\mathcal{A}$ mit $\mathcal{R}_{\mathcal{A}} \leq 2$.

Beweis:

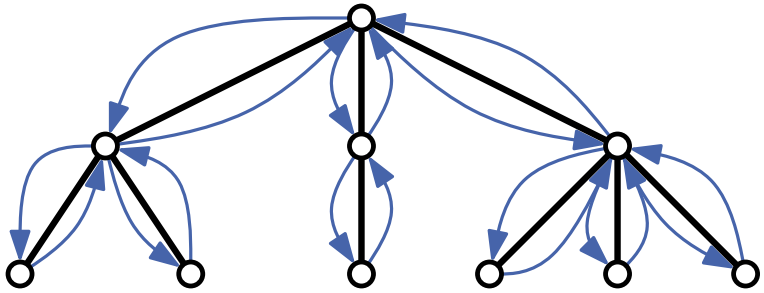
2-Approximation

Schritt 1



Berechne minimalen Spannbaum des Eingabegraphen

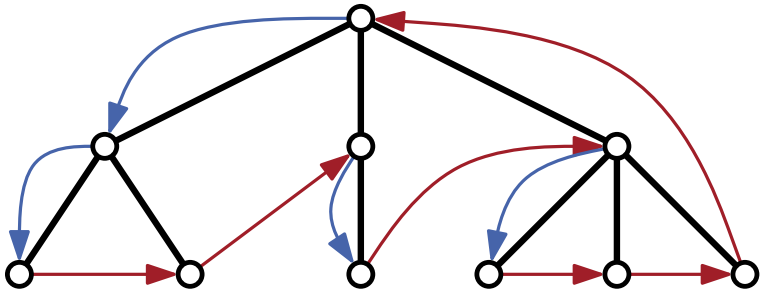
Schritt 2



Tiefensuch-Tour T durch den MST

Unschön: Tour ist kein einfacher Kreis

Schritt 3



TSP-Tour als abgekürzte Tiefensuch-Tour T'

(Schon besuchte Knoten werden einfach übersprungen.)

Satz: Approximation **(Satz 4.43)**

Für TSP existiert ein Approximationsalgorithmus $\mathcal{A}$ mit $\mathcal{R}_{\mathcal{A}} \leq 2$.

Beweis:

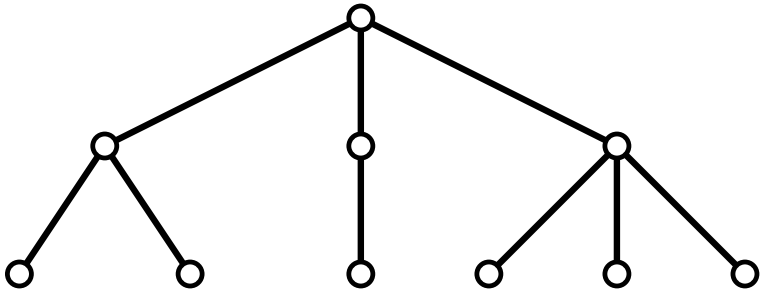
- Wegen der Dreiecksungleichung gilt:

  $c(T') \leq c(T) = 2 \cdot c(\text{MST})$

  ($c(X)$ ist Summe aller Kantengewichte in X)

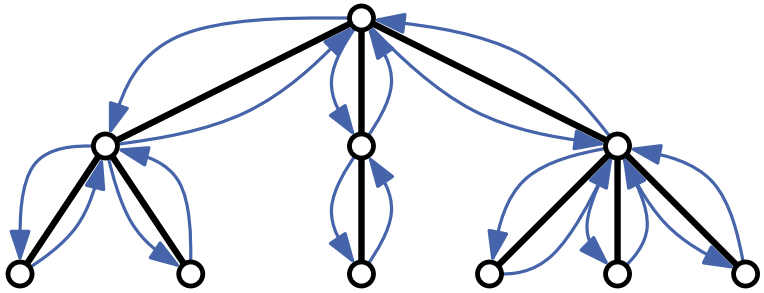
2-Approximation

Schritt 1



Berechne minimalen Spannbaum des Eingabegraphen

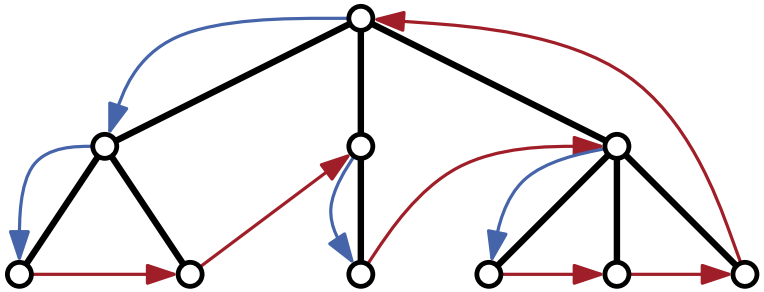
Schritt 2



Tiefensuch-Tour T durch den MST

Unschön: Tour ist kein einfacher Kreis

Schritt 3



TSP-Tour als abgekürzte Tiefensuch-Tour T'

(Schon besuchte Knoten werden einfach übersprungen.)

Satz: Approximation **(Satz 4.43)**

Für TSP existiert ein Approximationsalgorithmus $\mathcal{A}$ mit $\mathcal{R}_{\mathcal{A}} \leq 2$.

Beweis:

- Wegen der Dreiecksungleichung gilt:
  $c(T') \leq c(T) = 2 \cdot c(\text{MST})$
  ($c(X)$ ist Summe aller Kantengewichte in X)
- TSP-Tour ist aufspannender Baum mit einer zusätzlichen Kante.
  $\Rightarrow c(\text{MST}) \leq c(\text{OPT})$

Approximationsschemata – KNAPSACK

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Approximationsschemata

Definition: PAS **(Definition 4.44)**

Ein *(polynomielles) Approximationsschema (PAS)* für ein Optimierungsproblem Π ist eine Familie von Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$, sodass $\mathcal{A}_\varepsilon$ ein ε-approximierender Algorithmus ist.

(häufig auch PTAS genannt)

Beachte: Die Laufzeit jedes Algorithmus $\mathcal{A}_\varepsilon$ ist nur polynomiell in der Eingabegröße und kann stark von ε abhängen.

Definition: FPAS **(Definition 4.44)**

Ein Approximationsschema $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ heißt *vollpolynomiell (FPAS)*, falls die Laufzeit jedes $\mathcal{A}_\varepsilon$ zusätzlich polynomiell in $\frac{1}{\varepsilon}$ ist.

(häufig auch FPTAS genannt)

Beispiel:

- Laufzeit $O\left(n^2 + n^{\frac{1}{\varepsilon}}\right)$ für $A_\varepsilon : \rightarrow$ PAS aber kein FPAS
- Laufzeit $O\left(\sqrt{n} \cdot 3^{\frac{1}{\varepsilon}}\right)$ für $A_\varepsilon : \rightarrow$ PAS aber kein FPAS
- Laufzeit $O\left(n^4 \cdot \left(\frac{1}{\varepsilon}\right)^2\right)$ für $A_\varepsilon : \rightarrow$ FPAS

KNAPSACK

Menge $M = \{1, \ldots, n\}$ mit n Gegenständen

Rucksack mit Maximalgewicht W

Jeder Gegenstand i

- hat Kosten (eigentlich Nutzen) $c_i \in \mathbb{N}_0$.
- hat ein Gewicht $w_i \in \mathbb{N}$.
- kann nur einmal eingepackt werden.

Problem: KNAPSACK

Finde Teilmenge $M' \subseteq M$, sodass $\sum_{i \in M'} w_i \leq W$ und $\sum_{i \in M'} c_i$ maximal ist.

Gesamtgewicht Gesamtkosten

- **Achtung:** Vorhin wurde eine andere Variante von KNAPSACK betrachtet.
  $\rightarrow$ Jetzt kann jedes Element nur einmal eingepackt werden.
- Diese Variante von KNAPSACK ist ebenfalls NP-schwer.

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

minimiere Gewicht — fixe Gesamtkosten (Nutzen)

$$w_r^j = \min_{M' \subseteq \{1,\ldots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

Einschränkung der möglichen Elemente

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

Einschränkung der möglichen Elemente

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

minimiere Gewicht — fixe Gesamtkosten (Nutzen)

$$w_r^j = \min_{M' \subseteq \{1,\ldots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

Einschränkung der möglichen Elemente

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt — Element j wird nicht gewählt

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞				
2	∞				
3	∞				
4	∞				
5	∞				
6	∞				
7	∞				

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞			
2	∞				
3	∞				
4	∞				
5	∞				
6	∞				
7	∞				

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j,\ w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞			
2	∞	3			
3	∞				
4	∞				
5	∞				
6	∞				
7	∞				

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞			
2	∞	3			
3	∞	∞			
4	∞				
5	∞				
6	∞				
7	∞				

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞			
2	∞	3			
3	∞	∞			
4	∞	∞			
5	∞				
6	∞				
7	∞				

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞			
2	∞	3			
3	∞	∞			
4	∞	∞			
5	∞	∞			
6	∞				
7	∞				

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞			
2	∞	3			
3	∞	∞			
4	∞	∞			
5	∞	∞			
6	∞	∞			
7	∞				

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞			
2	∞	3			
3	∞	∞			
4	∞	∞			
5	∞	∞			
6	∞	∞			
7	∞	∞			

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3		
2	∞	3			
3	∞	∞			
4	∞	∞			
5	∞	∞			
6	∞	∞			
7	∞	∞			

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

leeres Kästchen = ∞

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3		
2	∞	3	3		
3	∞	∞			
4	∞	∞			
5	∞	∞			
6	∞	∞			
7	∞	∞			

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3		
2	∞	3	3		
3	∞	∞	6		
4	∞	∞			
5	∞	∞			
6	∞	∞			
7	∞	∞			

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

minimiere Gewicht — fixe Gesamtkosten (Nutzen)

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt — Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3		
2	∞	3	3		
3	∞	∞	6		
4	∞	∞	∞		
5	∞	∞	∞		
6	∞	∞	∞		
7	∞	∞	∞		

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	
2	∞	3	3		
3	∞	∞	6		
4	∞	∞	∞		
5	∞	∞	∞		
6	∞	∞	∞		
7	∞	∞	∞		

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	
2	∞	3	3	3	
3	∞	∞	6		
4	∞	∞	∞		
5	∞	∞	∞		
6	∞	∞	∞		
7	∞	∞	∞		

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	
2	∞	3	3	3	
3	∞	∞	6	5	
4	∞	∞	∞		
5	∞	∞	∞		
6	∞	∞	∞		
7	∞	∞	∞		

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	
2	∞	3	3	3	
3	∞	∞	6	5	
4	∞	∞	∞	8	
5	∞	∞	∞		
6	∞	∞	∞		
7	∞	∞	∞		

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	
2	∞	3	3	3	
3	∞	∞	6	5	
4	∞	∞	∞	8	
5	∞	∞	∞	8	
6	∞	∞	∞		
7	∞	∞	∞		

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	
2	∞	3	3	3	
3	∞	∞	6	5	
4	∞	∞	∞	8	
5	∞	∞	∞	8	
6	∞	∞	∞	11	
7	∞	∞	∞		

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	
2	∞	3	3	3	
3	∞	∞	6	5	
4	∞	∞	∞	8	
5	∞	∞	∞	8	
6	∞	∞	∞	11	
7	∞	∞	∞	∞	

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

leeres Kästchen = ∞

$r \downarrow$ $j \rightarrow$	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	2
2	∞	3	3	3	3
3	∞	∞	6	5	5
4	∞	∞	∞	8	7
5	∞	∞	∞	8	8
6	∞	∞	∞	11	10
7	∞	∞	∞	∞	13

$c = \sum_{i \in M} c_i$

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	2
2	∞	3	3	3	3
3	∞	∞	6	5	5
4	∞	∞	∞	8	7
5	∞	∞	∞	8	8
6	∞	∞	∞	11	10
7	∞	∞	∞	∞	13

Optimale Lösung: $c^\star = \max_{1 \le j \le n} \{ r \mid w_r^j \le W \} = 10$

mit $M' = \{1, 3, 4\}$

$c = \sum_{i \in M} c_i$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Idee: Berechne Gewicht w_r^j für alle „sinnvollen“ r und j, mit:

$$w_r^j = \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$$

minimiere Gewicht

fixe Gesamtkosten (Nutzen)

Einschränkung der möglichen Elemente

Beispiel: $W = 10$

i	1	2	3	4
c_i	2	1	3	1
w_i	3	3	5	2

Initialisierung: $w_0^j = 0$ und $w_r^j = \infty$ für $r \neq 0$

wähle kein Element um Nutzen 0 zu erhalten

Berechne: $w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w_j, w_r^{j-1} \right\}$

Element j wird gewählt

Element j wird nicht gewählt

leeres Kästchen = ∞

r \ j	0	1	2	3	4
−2	∞	∞	∞	∞	∞
−1	∞	∞	∞	∞	∞
0	0	0	0	0	0
1	∞	∞	3	3	2
2	∞	3	3	3	3
3	∞	∞	6	5	5
4	∞	∞	∞	8	7
5	∞	∞	∞	8	8
6	∞	∞	∞	11	10
7	∞	∞	∞	∞	13

Optimale Lösung: $c^\star = \max_{1 \le j \le n} \{ r \mid w_r^j \le W \} = 10$

mit $M' = \{1, 3, 4\}$

$c = \sum_{i \in M} c_i$

Laufzeit: $O(c \cdot n)$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Dynamisches Programm für KNAPSACK

Satz: Dynamisches Programm
Das dyn. Programm berechnet eine optimale Lösung für KNAPSACK in $O(c \cdot n)$.

Problem:

- $c = \sum_{i \in M} c_i$ ist ggf. exponentiell in der Eingabegröße (die Zahlen c_i sind binär kodiert).
- Ein solcher Algorithmus wird *pseudopolynomiell* genannt.

Dynamisches Programm für KNAPSACK

Satz: Dynamisches Programm
Das dyn. Programm berechnet eine optimale Lösung für KNAPSACK in $O(c \cdot n)$.

Problem:

- $c = \sum_{i \in M} c_i$ ist ggf. exponentiell in der Eingabegröße (die Zahlen c_i sind binär kodiert).
- Ein solcher Algorithmus wird *pseudopolynomiell* genannt.

Idee für einen Approximationsschema:

- Skaliere Gewichte c_i mit einer Konstanten k; benutze also $c'_i = \left\lfloor \frac{c_i}{k} \right\rfloor$.
- Je größer k, desto kleiner wird c (und damit die Laufzeit).
- Je größer k, desto größer ist der Fehler, den man durchs Runden macht.

Dynamisches Programm für KNAPSACK

Satz: Dynamisches Programm
Das dyn. Programm berechnet eine optimale Lösung für KNAPSACK in $O(c \cdot n)$.

Problem:

- $c = \sum_{i \in M} c_i$ ist ggf. exponentiell in der Eingabegröße (die Zahlen c_i sind binär kodiert).
- Ein solcher Algorithmus wird *pseudopolynomiell* genannt.

Idee für einen Approximationsschema:

- Skaliere Gewichte c_i mit einer Konstanten k; benutze also $c'_i = \left\lfloor \frac{c_i}{k} \right\rfloor$.
- Je größer k, desto kleiner wird c (und damit die Laufzeit).
- Je größer k, desto größer ist der Fehler, den man durchs Runden macht.

Algorithmus $\mathcal{A}_\varepsilon$:

- Skaliere mit $k = \dfrac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$ ($c_{\max} = \max_{i \in M} c_i$ ist maximales Gewicht in gegebener Instanz)
- Berechne optimale Lösung der skalierten Instanz mit dynamischem Programm.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

Beweis – Laufzeit:

Erinnerung: $k = \frac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$

- $\mathcal{A}_\varepsilon$ hat Laufzeit $O(n \cdot \sum_{i \in M} c'_i)$
- Es gilt:

$$\sum_{i \in M} c'_i = \sum_{i \in M} \left\lfloor \frac{c_i}{k} \right\rfloor \leq \sum_{i \in M} \frac{c_i}{k} \leq n \cdot \frac{c_{\max}}{k} = \left(\frac{1}{\varepsilon} + 1\right) n^2$$

- $\Rightarrow \mathcal{A}_\varepsilon$ hat Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

Beweis – Approximation:

Erinnerung: $k = \frac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$

- Sie I eine beliebige Instanz und sei I_k die skalierte Instanz.
- Sei $M' \subseteq M$ eine optimale Lösung für I, also $\text{OPT}(I) = \sum_{i \in M'} c_i$. Es gilt:

$$\text{OPT}(I_k) \geq \sum_{i \in M'} \left\lfloor \frac{c_i}{k} \right\rfloor \geq \sum_{i \in M'} \left(\frac{c_i}{k} - 1 \right) \geq \sum_{i \in M'} \frac{c_i}{k} - n = \frac{\text{OPT}(I)}{k} - n$$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

Beweis – Approximation:

Erinnerung: $k = \frac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$

- Sie I eine beliebige Instanz und sei I_k die skalierte Instanz.
- Sei $M' \subseteq M$ eine optimale Lösung für I, also $\text{OPT}(I) = \sum_{i \in M'} c_i$. Es gilt:

$$\text{OPT}(I_k) \geq \sum_{i \in M'} \left\lfloor \frac{c_i}{k} \right\rfloor \geq \sum_{i \in M'} \left(\frac{c_i}{k} - 1 \right) \geq \sum_{i \in M'} \frac{c_i}{k} - n = \frac{\text{OPT}(I)}{k} - n$$

- Umstellen ergibt: $\text{OPT}(I) - k \cdot \text{OPT}(I_k) \leq k \cdot n$
- Da $\mathcal{A}_\varepsilon(I) \geq k \cdot \text{OPT}(I_k)$ ist, folgt $\text{OPT}(I) - \mathcal{A}_\varepsilon(I) \leq k \cdot n$
- Außerdem gilt: $\text{OPT}(I) \geq c_{\max}$ (da o.B.d.A. $W \geq w_i$ für alle $i \in M$)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

Beweis – Approximation:

Erinnerung: $k = \frac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$

- Sie I eine beliebige Instanz und sei I_k die skalierte Instanz.

Bisher gezeigt: $\text{OPT}(I) - \mathcal{A}_\varepsilon(I) \leq k \cdot n$ und $\text{OPT}(I) \geq c_{\max}$

$$\mathcal{R}_{\mathcal{A}_\varepsilon}(I) = \frac{\text{OPT}(I)}{\mathcal{A}_\varepsilon(I)} \leq \frac{\mathcal{A}_\varepsilon(I) + kn}{\mathcal{A}_\varepsilon(I)} = 1 + \frac{kn}{\mathcal{A}_\varepsilon(I)}$$

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

Beweis – Approximation:

Erinnerung: $k = \dfrac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$

- Sie I eine beliebige Instanz und sei I_k die skalierte Instanz.

Bisher gezeigt: $\text{OPT}(I) - \mathcal{A}_\varepsilon(I) \leq k \cdot n$ und $\text{OPT}(I) \geq c_{\max}$

$$\mathcal{R}_{\mathcal{A}_\varepsilon}(I) = \frac{\text{OPT}(I)}{\mathcal{A}_\varepsilon(I)} \leq \frac{\mathcal{A}_\varepsilon(I) + kn}{\mathcal{A}_\varepsilon(I)} = 1 + \frac{kn}{\mathcal{A}_\varepsilon(I)}$$

$$\leq 1 + \frac{kn}{\text{OPT}(I) - kn}$$

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

Beweis – Approximation:

Erinnerung: $k = \dfrac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$

- Sie I eine beliebige Instanz und sei I_k die skalierte Instanz.

Bisher gezeigt: $\text{OPT}(I) - \mathcal{A}_\varepsilon(I) \leq k \cdot n$ und $\text{OPT}(I) \geq c_{\max}$

$$\mathcal{R}_{\mathcal{A}_\varepsilon}(I) = \frac{\text{OPT}(I)}{\mathcal{A}_\varepsilon(I)} \leq \frac{\mathcal{A}_\varepsilon(I) + kn}{\mathcal{A}_\varepsilon(I)} = 1 + \frac{kn}{\mathcal{A}_\varepsilon(I)}$$

$$\leq 1 + \frac{kn}{\text{OPT}(I) - kn} \leq 1 + \frac{kn}{c_{\max} - kn}$$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

Beweis – Approximation:

Erinnerung: $k = \frac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$

- Sie I eine beliebige Instanz und sei I_k die skalierte Instanz.

Bisher gezeigt: $\text{OPT}(I) - \mathcal{A}_\varepsilon(I) \leq k \cdot n$ und $\text{OPT}(I) \geq c_{\max}$

$$\mathcal{R}_{\mathcal{A}_\varepsilon}(I) = \frac{\text{OPT}(I)}{\mathcal{A}_\varepsilon(I)} \leq \frac{\mathcal{A}_\varepsilon(I) + kn}{\mathcal{A}_\varepsilon(I)} = 1 + \frac{kn}{\mathcal{A}_\varepsilon(I)}$$

$$\leq 1 + \frac{kn}{\text{OPT}(I) - kn} \leq 1 + \frac{kn}{c_{\max} - kn}$$

kürzen mit kn

$$= 1 + \frac{1}{\frac{c_{\max}}{kn} - 1}$$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

FPAS für KNAPSACK

Satz: FPAS **(Satz 4.46)**

$\mathcal{A}_\varepsilon$ hat die Laufzeit $O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$ und die Approximationsrate $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$. Die Menge der Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist also ein FPAS für KNAPSACK.

Beweis – Approximation:

Erinnerung: $k = \frac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$

- Sie I eine beliebige Instanz und sei I_k die skalierte Instanz.

Bisher gezeigt: $\text{OPT}(I) - \mathcal{A}_\varepsilon(I) \leq k \cdot n$ und $\text{OPT}(I) \geq c_{\max}$

$$\mathcal{R}_{\mathcal{A}_\varepsilon}(I) = \frac{\text{OPT}(I)}{\mathcal{A}_\varepsilon(I)} \leq \frac{\mathcal{A}_\varepsilon(I) + kn}{\mathcal{A}_\varepsilon(I)} = 1 + \frac{kn}{\mathcal{A}_\varepsilon(I)}$$

$$\leq 1 + \frac{kn}{\text{OPT}(I) - kn} \leq 1 + \frac{kn}{c_{\max} - kn}$$

kürzen mit kn

$$= 1 + \frac{1}{\frac{c_{\max}}{kn} - 1} = 1 + \frac{1}{\frac{1}{\varepsilon} + 1 - 1}$$

$$= 1 + \varepsilon$$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner