5 Approximationsalgorithmen

Optimierungsproblem

Ein **Optimierungsproblem** Π besteht aus den folgenden Komponenten.

- Menge $\mathcal{I}_\Pi$ von **Instanzen** oder **Eingaben**
- für jedes $I \in \mathcal{I}_\Pi$ Menge $\mathcal{S}_I$ von **Lösungen**
- für jedes $I \in \mathcal{I}_\Pi$ **Zielfunktion** $f_I : \mathcal{S}_I \to \mathbb{R}_{\geq 0}$, die jeder Lösung einen reellen Wert zuweist
- Angabe, ob **minimiert** oder **maximiert** werden soll

Für Eingabe I bezeichne $\mathrm{OPT}(I)$ den **Wert einer optimalen Lösung**.

Beispiel: Spannbaumproblem

- **Eingabe I:** ungerichteter Graph $G = (V, E)$, Kantengewichte $c : E \to \mathbb{N}$
- **Lösungsmenge $\mathcal{S}_I$:** Menge aller Spannbäume von G
- **Zielfunktion f_I:** $f_I(T) = \sum_{e \in T} c(e)$ für Spannbaum $T \in \mathcal{S}_I$
- **Minimiere f_I**

Es gilt $\mathrm{OPT}(I) = \min_{T \in \mathcal{S}_I} f_I(T)$.

5 Approximationsalgorithmen

Ein **Approximationsalgorithmus** A für Π ist ein **Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt**.

Ein **Approximationsalgorithmus** A für Π ist ein **Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt**.

Es sei $A(I)$ die Lösung, die A bei Eingabe I ausgibt, und $w_A(I) = f_I(A(I))$ ihr Wert.

Ein **Approximationsalgorithmus** A für Π ist ein **Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt**.

Es sei $A(I)$ die Lösung, die A bei Eingabe I ausgibt, und $w_A(I) = f_I(A(I))$ ihr Wert.

Definition 5.1 (Approximationsfaktor/Approximationsgüte)

Ein Approximationsalgorithmus A für ein Minimierungs- bzw. Maximierungsproblem Π erreicht einen **Approximationsfaktor** oder eine **Approximationsgüte** von $r \geq 1$ bzw. $r \leq 1$, wenn

$$w_A(I) \leq r \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I) \geq r \cdot \mathrm{OPT}(I)$$

für alle Instanzen $I \in \mathcal{I}_\Pi$ gilt. Wir sagen dann, dass A ein **r-Approximationsalgorithmus** ist.

Ein **Approximationsalgorithmus** A für Π ist ein **Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt**.

Es sei $A(I)$ die Lösung, die A bei Eingabe I ausgibt, und $w_A(I) = f_I(A(I))$ ihr Wert.

Definition 5.1 (Approximationsfaktor/Approximationsgüte)

Ein Approximationsalgorithmus A für ein Minimierungs- bzw. Maximierungsproblem Π erreicht einen **Approximationsfaktor** oder eine **Approximationsgüte** von $r \geq 1$ bzw. $r \leq 1$, wenn

$$w_A(I) \leq r \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I) \geq r \cdot \mathrm{OPT}(I)$$

für alle Instanzen $I \in \mathcal{I}_\Pi$ gilt. Wir sagen dann, dass A ein **r-Approximationsalgorithmus** ist.

Ist Π NP-schwer und gilt P $\neq$ NP, so existiert für Π kein 1-Approximationsalgorithmus.

Traveling Salesman Problem (TSP)

Eingabe:	Menge $V = \{v_1, \ldots, v_n\}$ von Knoten
	symmetrische Distanzfunktion $d : V \times V \to \mathbb{R}_{\geq 0}$
	(d. h. $\forall u, v \in V : d(u, v) = d(v, u) \geq 0$)
Lösungen:	alle Permutationen $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$
	eine solche Permutation nennen wir auch *Tour*
Zielfunktion:	minimiere $\sum_{i=1}^{n-1} d\big(v_{\pi(i)}, v_{\pi(i+1)}\big) + d\big(v_{\pi(n)}, v_{\pi(1)}\big)$

5.2 Traveling Salesman Problem

Traveling Salesman Problem (TSP)

Eingabe: Menge $V = \{v_1, \ldots, v_n\}$ von Knoten
symmetrische Distanzfunktion $d : V \times V \to \mathbb{R}_{\geq 0}$
(d. h. $\forall u, v \in V : d(u, v) = d(v, u) \geq 0$)

Lösungen: alle Permutationen $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$
eine solche Permutation nennen wir auch *Tour*

Zielfunktion: minimiere $\sum_{i=1}^{n-1} d\left(v_{\pi(i)}, v_{\pi(i+1)}\right) + d\left(v_{\pi(n)}, v_{\pi(1)}\right)$

Theorem 5.4

Falls P $\neq$ NP, so existiert kein 2^n-Approximationsalgorithmus für das TSP.

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt:
Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt:
Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Sei $G = (V, E)$ Eingabe für HC. Wir **konstruieren TSP-Instanz** auf V mit:

$$\forall u, v \in V, u \neq v : d(u, v) = d(v, u) = \begin{cases} 1 & \text{falls } \{u, v\} \in E, \\ n2^{n+1} & \text{falls } \{u, v\} \notin E. \end{cases}$$

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt: Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Sei $G = (V, E)$ Eingabe für HC. Wir **konstruieren TSP-Instanz** auf V mit:

$$\forall u, v \in V, u \neq v : d(u, v) = d(v, u) = \begin{cases} 1 & \text{falls } \{u, v\} \in E, \\ n2^{n+1} & \text{falls } \{u, v\} \notin E. \end{cases}$$

G enthält HC. $\Rightarrow$ Es gibt **TSP-Tour C der Länge n**.

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt: Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Sei $G = (V, E)$ Eingabe für HC. Wir **konstruieren TSP-Instanz** auf V mit:

$$\forall u, v \in V, u \neq v : d(u, v) = d(v, u) = \begin{cases} 1 & \text{falls } \{u, v\} \in E, \\ n2^{n+1} & \text{falls } \{u, v\} \notin E. \end{cases}$$

G enthält HC. $\Rightarrow$ Es gibt **TSP-Tour C der Länge n**. $\Rightarrow$ A berechnet Tour C' mit $d(C') \leq 2^n \cdot d(C) \leq n2^n$.

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt: Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Sei $G = (V, E)$ Eingabe für HC. Wir **konstruieren TSP-Instanz** auf V mit:

$$\forall u, v \in V, u \neq v : d(u, v) = d(v, u) = \begin{cases} 1 & \text{falls } \{u, v\} \in E, \\ n2^{n+1} & \text{falls } \{u, v\} \notin E. \end{cases}$$

G enthält HC. $\Rightarrow$ Es gibt **TSP-Tour C der Länge n**. $\Rightarrow$ A berechnet Tour C' mit $d(C') \leq 2^n \cdot d(C) \leq n2^n$. C' **enthält nur Kanten e $\in$ E** $\Rightarrow$ C' ist Hamiltonkreis in G. $\square$

5.2 Traveling Salesman Problem

Beim **metrischen TSP** bilden die Distanzen d eine Metrik auf V.

5.2 Traveling Salesman Problem

Beim **metrischen TSP** bilden die Distanzen d eine Metrik auf V.

Definition 5.5

Sei X eine Menge und $d : X \times X \to \mathbb{R}_{\geq 0}$ eine Funktion. Die Funktion d heißt **Metrik auf X**, wenn die folgenden drei Eigenschaften erfüllt sind.

- $\forall x, y \in X : d(x, y) = 0 \iff x = y$ **(positive Definitheit)**
- $\forall x, y \in X : d(x, y) = d(y, x)$ **(Symmetrie)**
- $\forall x, y, z \in X : d(x, z) \leq d(x, y) + d(y, z)$ **(Dreiecksungleichung)**

Das Paar (X, d) heißt **metrischer Raum**.

5.2 Traveling Salesman Problem

Beim **metrischen TSP** bilden die Distanzen d eine Metrik auf V.

Definition 5.5

Sei X eine Menge und $d : X \times X \to \mathbb{R}_{\geq 0}$ eine Funktion. Die Funktion d heißt **Metrik auf X**, wenn die folgenden drei Eigenschaften erfüllt sind.

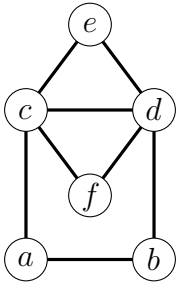
- $\forall x, y \in X : d(x, y) = 0 \iff x = y$ **(positive Definitheit)**
- $\forall x, y \in X : d(x, y) = d(y, x)$ **(Symmetrie)**
- $\forall x, y, z \in X : d(x, z) \leq d(x, y) + d(y, z)$ **(Dreiecksungleichung)**

Das Paar (X, d) heißt **metrischer Raum**.

Das metrische TSP ist ein Spezialfall des TSP.
Es ist noch **NP-schwer** denn das TSP ist bereits dann NP-schwer, wenn alle Distanzen entweder 1 oder 2 sind.

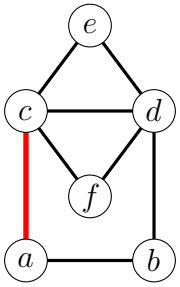
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



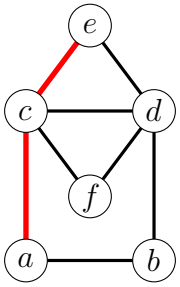
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



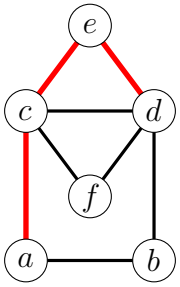
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



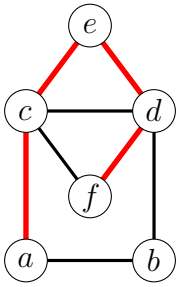
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



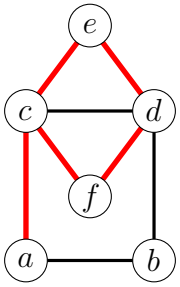
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



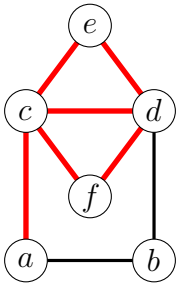
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



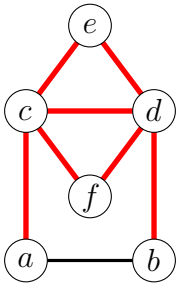
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



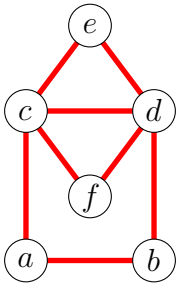
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



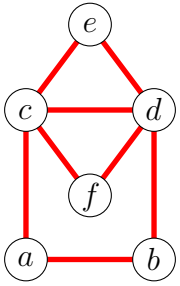
5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



5.2 Traveling Salesman Problem

Eulerkreis: Kreis in einem Graphen, der **jede Kante genau einmal enthält**.



Erweiterung auf Multigraphen: Ein zusammenhängender Multigraph enthält genau dann einen Eulerkreis, wenn **jeder Knoten geraden Grad** besitzt. Ein Eulerkreis kann dann **in polynomieller Zeit** berechnet werden.

Doppelbaum-TSP

Eingabe: Knotenmenge V, Metrik d auf V.

1. Sei $G = (V, E)$ ein vollständiger ungerichteter Graph mit Knotenmenge V. Berechne einen **minimalen Spannbaum T** von G bezüglich der Distanzen d.

Doppelbaum-TSP

Eingabe: Knotenmenge V, Metrik d auf V.

1. Sei $G = (V, E)$ ein vollständiger ungerichteter Graph mit Knotenmenge V. Berechne einen **minimalen Spannbaum T** von G bezüglich der Distanzen d.
2. Erzeuge Multigraph G', der **nur die Kanten aus T enthält und jede davon zweimal**. In G' besitzt jeder Knoten geraden Grad.

Doppelbaum-TSP

Eingabe: Knotenmenge V, Metrik d auf V.

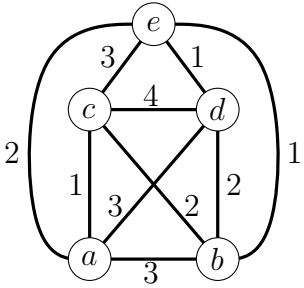
1. Sei $G = (V, E)$ ein vollständiger ungerichteter Graph mit Knotenmenge V. Berechne einen **minimalen Spannbaum T** von G bezüglich der Distanzen d.
2. Erzeuge Multigraph G', der **nur die Kanten aus T enthält und jede davon zweimal**. In G' besitzt jeder Knoten geraden Grad.
3. Finde einen **Eulerkreis A** in G'.

Doppelbaum-TSP

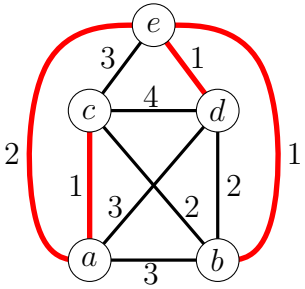
Eingabe: Knotenmenge V, Metrik d auf V.

1. Sei $G = (V, E)$ ein vollständiger ungerichteter Graph mit Knotenmenge V. Berechne einen **minimalen Spannbaum T** von G bezüglich der Distanzen d.
2. Erzeuge Multigraph G', der **nur die Kanten aus T enthält und jede davon zweimal**. In G' besitzt jeder Knoten geraden Grad.
3. Finde einen **Eulerkreis A** in G'.
4. Gib die Knoten in der Reihenfolge ihres ersten Auftretens in A aus. Das Ergebnis sei der **Hamiltonkreis C**.

Doppelbaum-TSP

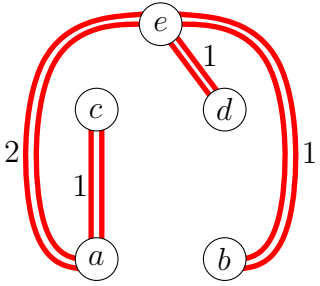


Doppelbaum-TSP



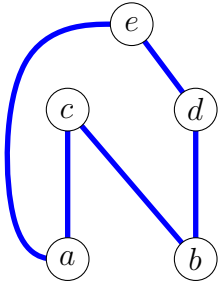
Spannbaum T

Doppelbaum-TSP



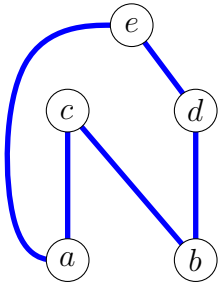
Eulerkreis $A = (c, a, e, d, e, b, e, a, c)$

Doppelbaum-TSP



Tour $C = (c, a, e, d, b, c)$

Doppelbaum-TSP



Tour $C = (c, a, e, d, b, c)$

Theorem 5.6

Der Algorithmus Doppelbaum-TSP ist ein 2-Approximationsalgorithmus für das metrische TSP.

Beweis: Für $X \subseteq E$ sei $d(X) = \sum_{\{u,v\} \in X} d(u, v)$.

Wir fassen T, A und C als ungeordnete Teilmengen der Kanten auf und benutzen die Bezeichnungen $d(T)$, $d(A)$ und $d(C)$.

Beweis: Für $X \subseteq E$ sei $d(X) = \sum_{\{u,v\} \in X} d(u, v)$.

Wir fassen T, A und C als ungeordnete Teilmengen der Kanten auf und benutzen die Bezeichnungen $d(T)$, $d(A)$ und $d(C)$.

Beobachtung: Sei $C^\star \subseteq E$ ein kürzester Hamiltonkreis und T ein MST in G.
Dann gilt $\mathbf{d(T) \leq d(C^\star)}$.

Beweis: Für $X \subseteq E$ sei $d(X) = \sum_{\{u,v\} \in X} d(u,v)$.

Wir fassen T, A und C als ungeordnete Teilmengen der Kanten auf und benutzen die Bezeichnungen $d(T)$, $d(A)$ und $d(C)$.

Beobachtung: Sei $C^\star \subseteq E$ ein kürzester Hamiltonkreis und T ein MST in G.
Dann gilt $\mathbf{d(T) \leq d(C^\star)}$.

Beweis der Beobachtung: Entferne aus $C^\star$ beliebige Kante e. Es ergibt sich ein Weg P, der jeden Knoten genau einmal enthält. Ein solcher Weg ist ein **Spannbaum von G**, also gilt $\mathbf{d(P) \geq d(T)}$**, da T ein minimaler Spannbaum ist**. Insgesamt erhalten wir damit

$$\mathrm{OPT} = d(C^\star) = d(P) + d(e) \geq d(P) \geq d(T). \qquad \square$$

Beweis: Für $X \subseteq E$ sei $d(X) = \sum_{\{u,v\} \in X} d(u,v)$.

Wir fassen T, A und C als ungeordnete Teilmengen der Kanten auf und benutzen die Bezeichnungen $d(T)$, $d(A)$ und $d(C)$.

Beobachtung: Sei $C^\star \subseteq E$ ein kürzester Hamiltonkreis und T ein MST in G.
Dann gilt $\mathbf{d(T) \leq d(C^\star)}$.

Beweis der Beobachtung: Entferne aus $C^\star$ beliebige Kante e. Es ergibt sich ein Weg P, der jeden Knoten genau einmal enthält. Ein solcher Weg ist ein **Spannbaum von G**, also gilt $\mathbf{d(P) \geq d(T)}$**, da T ein minimaler Spannbaum ist**. Insgesamt erhalten wir damit

$$\mathrm{OPT} = d(C^\star) = d(P) + d(e) \geq d(P) \geq d(T). \qquad \square$$

Insgesamt erhalten wir

$$d(C) \leq d(A) = 2d(T) \leq 2 \cdot \mathrm{OPT}. \qquad \square$$

5.2 Traveling Salesman Problem

$M \subseteq E$ heißt **Matching** in $G = (V, E)$, wenn kein Knoten zu mehr als einer Kante aus M inzident ist. **Perfektes Matching** ist ein Matching M mit $|M| = \frac{|V|}{2}$.

5.2 Traveling Salesman Problem

$M \subseteq E$ heißt **Matching** in $G = (V, E)$, wenn kein Knoten zu mehr als einer Kante aus M inzident ist. **Perfektes Matching** ist ein Matching M mit $|M| = \frac{|V|}{2}$.

In einem vollständigen Graphen mit einer geraden Anzahl an Knoten kann ein perfektes Matching mit minimalem Gewicht **in polynomieller Zeit** berechnet werden.

$M \subseteq E$ heißt **Matching** in $G = (V, E)$, wenn kein Knoten zu mehr als einer Kante aus M inzident ist. **Perfektes Matching** ist ein Matching M mit $|M| = \frac{|V|}{2}$.

In einem vollständigen Graphen mit einer geraden Anzahl an Knoten kann ein perfektes Matching mit minimalem Gewicht **in polynomieller Zeit** berechnet werden.

Christofides-Algorithmus

1. Sei $G = (V, E)$ ein vollständiger ungerichteter Graph mit Knotenmenge V. Berechne einen **minimalen Spannbaum T** von G bezüglich der Distanzen d.

5.2 Traveling Salesman Problem

$M \subseteq E$ heißt **Matching** in $G = (V, E)$, wenn kein Knoten zu mehr als einer Kante aus M inzident ist. **Perfektes Matching** ist ein Matching M mit $|M| = \frac{|V|}{2}$.

In einem vollständigen Graphen mit einer geraden Anzahl an Knoten kann ein perfektes Matching mit minimalem Gewicht **in polynomieller Zeit** berechnet werden.

Christofides-Algorithmus

1. Sei $G = (V, E)$ ein vollständiger ungerichteter Graph mit Knotenmenge V. Berechne einen **minimalen Spannbaum T** von G bezüglich der Distanzen d.
2. Sei $V' = \{v \in V \mid v \text{ hat ungeraden Grad in } T\}$. Berechne auf der Menge V' ein **perfektes Matching M mit minimalem Gewicht**.

$M \subseteq E$ heißt **Matching** in $G = (V, E)$, wenn kein Knoten zu mehr als einer Kante aus M inzident ist. **Perfektes Matching** ist ein Matching M mit $|M| = \frac{|V|}{2}$.

In einem vollständigen Graphen mit einer geraden Anzahl an Knoten kann ein perfektes Matching mit minimalem Gewicht **in polynomieller Zeit** berechnet werden.

Christofides-Algorithmus

1. Sei $G = (V, E)$ ein vollständiger ungerichteter Graph mit Knotenmenge V. Berechne einen **minimalen Spannbaum T** von G bezüglich der Distanzen d.
2. Sei $V' = \{v \in V \mid v \text{ hat ungeraden Grad in } T\}$. Berechne auf der Menge V' ein **perfektes Matching M mit minimalem Gewicht**.
3. Sei $\tilde{G} = (V, T \cup M)$ ein Multigraph, der jede Kante $e \in T \cap M$ zweimal enthält. Finde einen **Eulerkreis A** in dem Multigraphen $\tilde{G}$.

5.2 Traveling Salesman Problem

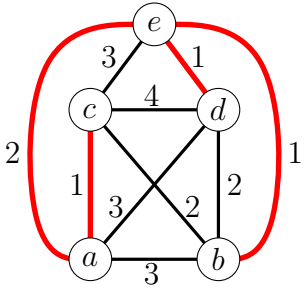
$M \subseteq E$ heißt **Matching** in $G = (V, E)$, wenn kein Knoten zu mehr als einer Kante aus M inzident ist. **Perfektes Matching** ist ein Matching M mit $|M| = \frac{|V|}{2}$.

In einem vollständigen Graphen mit einer geraden Anzahl an Knoten kann ein perfektes Matching mit minimalem Gewicht **in polynomieller Zeit** berechnet werden.

Christofides-Algorithmus

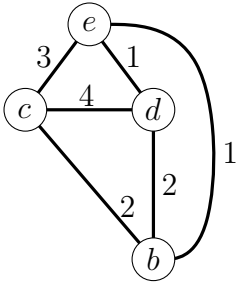
1. Sei $G = (V, E)$ ein vollständiger ungerichteter Graph mit Knotenmenge V. Berechne einen **minimalen Spannbaum T** von G bezüglich der Distanzen d.
2. Sei $V' = \{v \in V \mid v \text{ hat ungeraden Grad in } T\}$. Berechne auf der Menge V' ein **perfektes Matching M mit minimalem Gewicht**.
3. Sei $\tilde{G} = (V, T \cup M)$ ein Multigraph, der jede Kante $e \in T \cap M$ zweimal enthält. Finde einen **Eulerkreis A** in dem Multigraphen $\tilde{G}$.
4. Gib die Knoten in der Reihenfolge ihres ersten Auftretens in A aus. Das Ergebnis sei der **Hamiltonkreis C**.

Christofides-Algorithmus



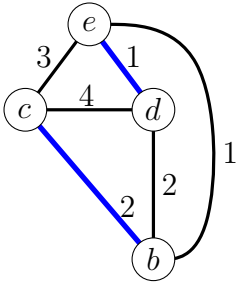
Spannbaum T

Christofides-Algorithmus



$V' \subseteq V$

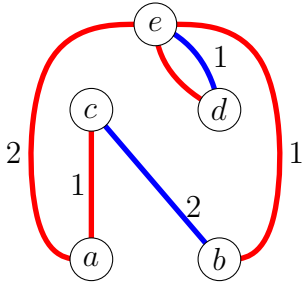
Christofides-Algorithmus



$V' \subseteq V$

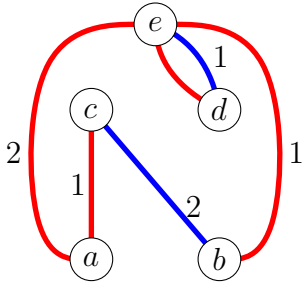
Matching M

Christofides-Algorithmus



$T \cup M$

Christofides-Algorithmus



$T \cup M$

Theorem 5.7

Der Christofides-Algorithmus ist ein $\frac{3}{2}$-Approximationsalgorithmus für das metrische TSP.

Beweis: Der Christofides-Algorithmus **kann ausgeführt werden**:

1. Auf der Menge V' **existiert ein perfektes Matching**.
2. Der Multigraph $\tilde{G} = (V, T \cup M)$ **enthält einen Eulerkreis**.

Beweis: Der Christofides-Algorithmus **kann ausgeführt werden**:

1. Auf der Menge V' **existiert ein perfektes Matching**.
2. Der Multigraph $\tilde{G} = (V, T \cup M)$ **enthält einen Eulerkreis**.

zu 1: G ist vollständig. **Zu zeigen:** $|V'|$ ist gerade.

Für $v \in V$ bezeichne $\delta(v)$ den Grad des Knotens v in dem Graph (V, T). Dann ist

$$\sum_{v \in V} \delta(v) = 2|T|$$

eine gerade Zahl.

Beweis: Der Christofides-Algorithmus **kann ausgeführt werden**:

1. Auf der Menge V' **existiert ein perfektes Matching**.
2. Der Multigraph $\tilde{G} = (V, T \cup M)$ **enthält einen Eulerkreis**.

zu 1: G ist vollständig. **Zu zeigen:** $|V'|$ ist gerade.

Für $v \in V$ bezeichne $\delta(v)$ den Grad des Knotens v in dem Graph (V, T). Dann ist

$$\sum_{v \in V} \delta(v) = 2|T|$$

eine gerade Zahl. Bezeichne q die **Anzahl an Knoten mit ungeradem Grad**. Da die Summe gerade ist, ist q ebenfalls gerade.

Beweis: Der Christofides-Algorithmus **kann ausgeführt werden**:

1. Auf der Menge V' **existiert ein perfektes Matching**.
2. Der Multigraph $\tilde{G} = (V, T \cup M)$ **enthält einen Eulerkreis**.

zu 1: G ist vollständig. **Zu zeigen:** $|V'|$ ist gerade.

Für $v \in V$ bezeichne $\delta(v)$ den Grad des Knotens v in dem Graph (V, T). Dann ist

$$\sum_{v \in V} \delta(v) = 2|T|$$

eine gerade Zahl. Bezeichne q die **Anzahl an Knoten mit ungeradem Grad**. Da die Summe gerade ist, ist q ebenfalls gerade.

zu 2: In $\tilde{G} = (V, T \cup M)$ besitzt **jeder Knoten geraden Grad**.

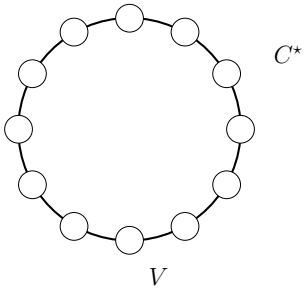
Lemma 5.8

Es sei $V' \subseteq V$ beliebig, sodass $|V'|$ gerade ist. Außerdem sei M ein perfektes Matching auf V' mit minimalem Gewicht $d(M)$. Dann gilt $d(M) \leq \mathrm{OPT}/2$.

Lemma 5.8

Es sei $V' \subseteq V$ beliebig, sodass $|V'|$ gerade ist. Außerdem sei M ein perfektes Matching auf V' mit minimalem Gewicht $d(M)$. Dann gilt $d(M) \leq \mathrm{OPT}/2$.

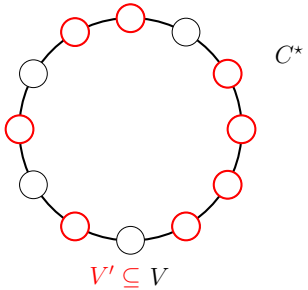
Beweis: $C^\star =$ optimale TSP-Tour auf V $C' =$ durch $C^\star$ induzierte Tour auf V'



Lemma 5.8

Es sei $V' \subseteq V$ beliebig, sodass $|V'|$ gerade ist. Außerdem sei M ein perfektes Matching auf V' mit minimalem Gewicht $d(M)$. Dann gilt $d(M) \leq \mathrm{OPT}/2$.

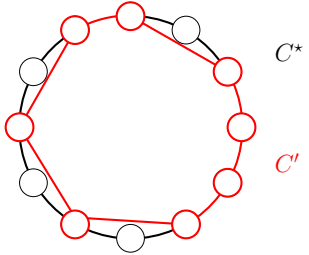
Beweis: $C^\star =$ optimale TSP-Tour auf V $\quad$ $C' =$ durch $C^\star$ induzierte Tour auf V'



Lemma 5.8

Es sei $V' \subseteq V$ beliebig, sodass $|V'|$ gerade ist. Außerdem sei M ein perfektes Matching auf V' mit minimalem Gewicht $d(M)$. Dann gilt $d(M) \leq \mathrm{OPT}/2$.

Beweis: $C^\star$ = optimale TSP-Tour auf V C' = durch $C^\star$ induzierte Tour auf V'

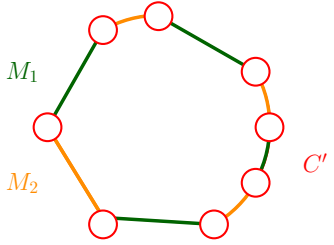


$$d(C') \leq d(C^\star) = \mathrm{opt}$$

Lemma 5.8

Es sei $V' \subseteq V$ beliebig, sodass $|V'|$ gerade ist. Außerdem sei M ein perfektes Matching auf V' mit minimalem Gewicht $d(M)$. Dann gilt $d(M) \leq \mathrm{OPT}/2$.

Beweis: $C^\star$ = optimale TSP-Tour auf V C' = durch $C^\star$ induzierte Tour auf V'



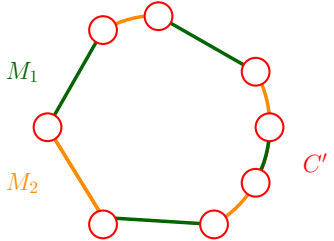
$V' \subseteq V$

$$d(M_1) + d(M_2) = d(C') \leq d(C^\star) = \mathrm{opt}$$

Lemma 5.8

Es sei $V' \subseteq V$ beliebig, sodass $|V'|$ gerade ist. Außerdem sei M ein perfektes Matching auf V' mit minimalem Gewicht $d(M)$. Dann gilt $d(M) \leq \mathrm{OPT}/2$.

Beweis: $C^\star$ = optimale TSP-Tour auf V C' = durch $C^\star$ induzierte Tour auf V'



$V' \subseteq V$

$$d(M_1) + d(M_2) = d(C') \leq d(C^\star) = \mathrm{opt}$$

$\Rightarrow d(M_1) \leq \mathrm{OPT}/2$ oder $d(M_2) \leq \mathrm{OPT}/2$ □

Es gilt $d(T) \leq \mathrm{OPT}$ und $d(M) \leq \mathrm{OPT}/2$.

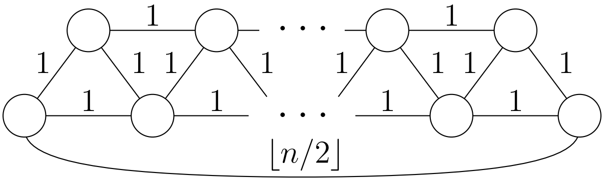
Es gilt $d(T) \leq \mathrm{OPT}$ und $d(M) \leq \mathrm{OPT}/2$.

Zusammen bedeutet das

$$d(C) \leq d(A) = d(T) + d(M) \leq \mathrm{OPT} + \frac{1}{2} \cdot \mathrm{OPT} \leq \frac{3}{2} \cdot \mathrm{OPT}. \qquad \square$$

Untere Schranke für den Approximationsfaktor des Christofides-Algorithmus

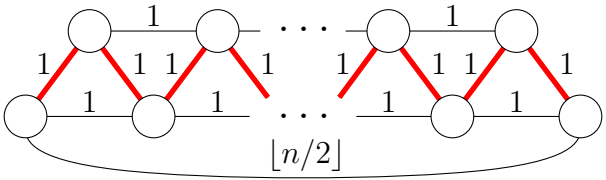
Sei $n \in \mathbb{N}$ ungerade. Wir betrachten die folgende Instanz des metrischen TSP.



Es gilt $\mathrm{OPT} = n$.

Untere Schranke für den Approximationsfaktor des Christofides-Algorithmus

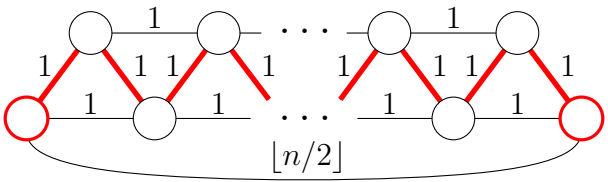
Sei $n \in \mathbb{N}$ ungerade. Wir betrachten die folgende Instanz des metrischen TSP.



Es gilt $\mathrm{OPT} = n$.

Untere Schranke für den Approximationsfaktor des Christofides-Algorithmus

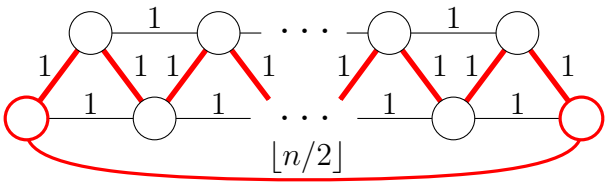
Sei $n \in \mathbb{N}$ ungerade. Wir betrachten die folgende Instanz des metrischen TSP.



Es gilt $\mathrm{OPT} = n$.

Untere Schranke für den Approximationsfaktor des Christofides-Algorithmus

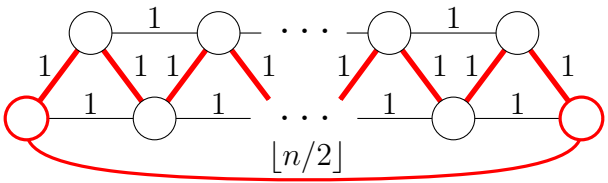
Sei $n \in \mathbb{N}$ ungerade. Wir betrachten die folgende Instanz des metrischen TSP.



Es gilt $\mathrm{OPT} = n$.

Untere Schranke für den Approximationsfaktor des Christofides-Algorithmus

Sei $n \in \mathbb{N}$ ungerade. Wir betrachten die folgende Instanz des metrischen TSP.



Es gilt $\mathrm{OPT} = n$.

Christofides-Algorithmus berechnet Lösung mit Wert

$$(n-1) + \lfloor n/2 \rfloor \approx \frac{3}{2}\mathrm{OPT}.$$

Definition 5.9

Ein **Approximationsschema** A für ein Optimierungsproblem Π ist ein Algorithmus, der zu jeder Eingabe der Form (I, ε) mit $I \in \mathcal{I}_\Pi$ und $\varepsilon > 0$ eine Lösung $A(I, \varepsilon) \in \mathcal{S}_I$ berechnet.

Definition 5.9

Ein **Approximationsschema** A für ein Optimierungsproblem Π ist ein Algorithmus, der zu jeder Eingabe der Form (I, ε) mit $I \in \mathcal{I}_\Pi$ und $\varepsilon > 0$ eine Lösung $A(I, \varepsilon) \in \mathcal{S}_I$ berechnet. Dabei muss der Wert $w_A(I, \varepsilon) = f_I(A(I, \varepsilon))$ dieser Lösung für jede Eingabe (I, ε) bei einem Minimierungs- oder Maximierungsproblem Π folgende Ungleichung erfüllen:

$$w_A(I, \varepsilon) \leq (1 + \varepsilon) \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I, \varepsilon) \geq (1 - \varepsilon) \cdot \mathrm{OPT}(I)$$

Definition 5.9

Ein **Approximationsschema** A für ein Optimierungsproblem Π ist ein Algorithmus, der zu jeder Eingabe der Form (I, ε) mit $I \in \mathcal{I}_\Pi$ und $\varepsilon > 0$ eine Lösung $A(I, \varepsilon) \in \mathcal{S}_I$ berechnet. Dabei muss der Wert $w_A(I, \varepsilon) = f_I(A(I, \varepsilon))$ dieser Lösung für jede Eingabe (I, ε) bei einem Minimierungs- oder Maximierungsproblem Π folgende Ungleichung erfüllen:

$$w_A(I, \varepsilon) \leq (1 + \varepsilon) \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I, \varepsilon) \geq (1 - \varepsilon) \cdot \mathrm{OPT}(I)$$

Ein Approximationsschema A heißt **polynomielles Approximationsschema (PTAS)**, wenn die Laufzeit von A für jede feste Wahl von $\varepsilon > 0$ durch ein Polynom in $|I|$ nach oben beschränkt ist.

Definition 5.9

Ein **Approximationsschema** A für ein Optimierungsproblem Π ist ein Algorithmus, der zu jeder Eingabe der Form (I, ε) mit $I \in \mathcal{I}_\Pi$ und $\varepsilon > 0$ eine Lösung $A(I, \varepsilon) \in \mathcal{S}_I$ berechnet. Dabei muss der Wert $w_A(I, \varepsilon) = f_I(A(I, \varepsilon))$ dieser Lösung für jede Eingabe (I, ε) bei einem Minimierungs- oder Maximierungsproblem Π folgende Ungleichung erfüllen:

$$w_A(I, \varepsilon) \leq (1 + \varepsilon) \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I, \varepsilon) \geq (1 - \varepsilon) \cdot \mathrm{OPT}(I)$$

Ein Approximationsschema A heißt **polynomielles Approximationsschema (PTAS)**, wenn die Laufzeit von A für jede feste Wahl von $\varepsilon > 0$ durch ein Polynom in $|I|$ nach oben beschränkt ist.

Wir nennen ein Approximationsschema A **voll-polynomielles Approximationsschema (FPTAS)**, wenn die Laufzeit von A durch ein bivariates Polynom in $|I|$ und $1/\varepsilon$ nach oben beschränkt ist.

Beispiellaufzeiten:

Beispielhafte Laufzeit eines PTAS: $\Theta(|I|^{1/\varepsilon})$

Beispielhafte Laufzeit eines FPTAS: $\Theta(|I|^3/\varepsilon^2)$.

Beispiellaufzeiten:

Beispielhafte Laufzeit eines PTAS: $\Theta(|I|^{1/\varepsilon})$

Beispielhafte Laufzeit eines FPTAS: $\Theta(|I|^3/\varepsilon^2)$.

Jedes FPTAS ist ein PTAS.

5.3 Rucksackproblem

Rucksackproblem (Knapsack Problem (KP))

Eingabe: **Nutzen** $p_1, \ldots, p_n \in \mathbb{N}$
Kapazität $t \in \mathbb{N}$
Gewichte $w_1, \ldots, w_n \in \{1, \ldots, t\}$

Ausgabe: $x_1, \ldots, x_n \in \{0, 1\}$, sodass Gesamtnutzen $p_1 x_1 + \ldots + p_n x_n$ **maximal unter der Bedingung** $w_1 x_1 + \ldots + w_n x_n \leq t$

Rucksackproblem (Knapsack Problem (KP))

Eingabe: Nutzen $p_1, \ldots, p_n \in \mathbb{N}$
Kapazität $t \in \mathbb{N}$
Gewichte $w_1, \ldots, w_n \in \{1, \ldots, t\}$

Ausgabe: $x_1, \ldots, x_n \in \{0, 1\}$, sodass Gesamtnutzen $p_1 x_1 + \ldots + p_n x_n$ maximal unter der Bedingung $w_1 x_1 + \ldots + w_n x_n \leq t$

Lösung mit dynamischer Programmierung: Wie sehen geeignete Teilprobleme aus?

Sei $P = \max_{i \in \{1,\ldots,n\}} p_i$.

Für jede Kombination aus $i \in \{1, \ldots, n\}$ und $p \in \{0, \ldots, nP\}$ sei

$$W(i, p) = \min\{w_1 x_1 + \ldots + w_i x_i \mid p_1 x_1 + \ldots + p_i x_i \geq p\}.$$

5.3 Rucksackproblem

Rucksackproblem (Knapsack Problem (KP))

Eingabe: Nutzen $p_1, \ldots, p_n \in \mathbb{N}$
Kapazität $t \in \mathbb{N}$
Gewichte $w_1, \ldots, w_n \in \{1, \ldots, t\}$

Ausgabe: $x_1, \ldots, x_n \in \{0, 1\}$, sodass Gesamtnutzen $p_1 x_1 + \ldots + p_n x_n$ maximal unter der Bedingung $w_1 x_1 + \ldots + w_n x_n \leq t$

Lösung mit dynamischer Programmierung: Wie sehen geeignete Teilprobleme aus?

Sei $P = \max_{i \in \{1,\ldots,n\}} p_i$.

Für jede Kombination aus $i \in \{1, \ldots, n\}$ und $p \in \{0, \ldots, nP\}$ sei

$$W(i, p) = \min\{w_1 x_1 + \ldots + w_i x_i \mid p_1 x_1 + \ldots + p_i x_i \geq p\}.$$

D. h. finde **unter allen Teilmengen der Objekte $1, \ldots, i$ mit Nutzen mindestens p die mit dem kleinsten Gewicht**.

Für jede Kombination aus $i \in \{1, \ldots, n\}$ und $p \in \{0, \ldots, nP\}$ sei

$$W(i,p) = \min\{w_1 x_1 + \ldots + w_i x_i \mid p_1 x_1 + \ldots + p_i x_i \geq p\}.$$

Randfälle:

$$W(1,p) = \begin{cases} w_1 & \text{falls } p \leq p_1 \\ \infty & \text{falls } p > p_1 \end{cases}$$

Für jede Kombination aus $i \in \{1, \ldots, n\}$ und $p \in \{0, \ldots, nP\}$ sei

$$W(i,p) = \min\{w_1 x_1 + \ldots + w_i x_i \mid p_1 x_1 + \ldots + p_i x_i \geq p\}.$$

Randfälle:

$$W(1,p) = \begin{cases} w_1 & \text{falls } p \leq p_1 \\ \infty & \text{falls } p > p_1 \end{cases}$$

Konvention:

$W(i,0) = 0$ für alle $i \in \{1, \ldots, n\}$ und $p \leq 0$.

5.3 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $p \in \{0, \ldots, nP\}$ **der Wert $W(i-1, p)$ bekannt**.

Ziel: Berechnung von $W(i, p)$ für $p \in \{0, \ldots, nP\}$.

Sei für ein $i \geq 2$ und für alle $p \in \{0, \ldots, nP\}$ **der Wert $W(i-1, p)$ bekannt**.

Ziel: Berechnung von $W(i, p)$ für $p \in \{0, \ldots, nP\}$.

Sei $I \subseteq \{1, \ldots, i\}$ mit $\sum_{i \in I} p_i \geq p$ und kleinstmöglichen Gewicht, d. h. $\sum_{i \in I} w_i = W(i, p)$.

Sei für ein $i \geq 2$ und für alle $p \in \{0, \ldots, nP\}$ **der Wert $W(i-1, p)$ bekannt**.

Ziel: Berechnung von $W(i, p)$ für $p \in \{0, \ldots, nP\}$.

Sei $I \subseteq \{1, \ldots, i\}$ mit $\sum_{i \in I} p_i \geq p$ und kleinstmöglichen Gewicht, d. h. $\sum_{i \in I} w_i = W(i, p)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I} p_j \geq p$.

5.3 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $p \in \{0, \ldots, nP\}$ **der Wert $W(i-1, p)$ bekannt**.

Ziel: Berechnung von $W(i, p)$ für $p \in \{0, \ldots, nP\}$.

Sei $I \subseteq \{1, \ldots, i\}$ mit $\sum_{i \in I} p_i \geq p$ und kleinstmöglichen Gewicht, d. h. $\sum_{i \in I} w_i = W(i, p)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I} p_j \geq p$.
  $\Rightarrow W(i, p) = W(i-1, p)$

5.3 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $p \in \{0, \ldots, nP\}$ **der Wert $W(i-1, p)$ bekannt**.

Ziel: Berechnung von $W(i, p)$ für $p \in \{0, \ldots, nP\}$.

Sei $I \subseteq \{1, \ldots, i\}$ mit $\sum_{i \in I} p_i \geq p$ und kleinstmöglichen Gewicht, d. h. $\sum_{i \in I} w_i = W(i, p)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I} p_j \geq p$.
  $\Rightarrow W(i, p) = W(i-1, p)$
- Falls $i \in I$, so ist $I \setminus \{i\} \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I \setminus \{i\}} p_j \geq p - p_i$.

5.3 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $p \in \{0, \ldots, nP\}$ **der Wert $W(i-1, p)$ bekannt**.

Ziel: Berechnung von $W(i, p)$ für $p \in \{0, \ldots, nP\}$.

Sei $I \subseteq \{1, \ldots, i\}$ mit $\sum_{i \in I} p_i \geq p$ und kleinstmöglichen Gewicht, d. h. $\sum_{i \in I} w_i = W(i, p)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I} p_j \geq p$.
  $\Rightarrow W(i, p) = W(i-1, p)$
- Falls $i \in I$, so ist $I \setminus \{i\} \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I \setminus \{i\}} p_j \geq p - p_i$.
  $\Rightarrow \sum_{j \in I \setminus \{i\}} w_j = W(i-1, p-p_i)$

5.3 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $p \in \{0, \ldots, nP\}$ **der Wert $W(i-1, p)$ bekannt**.

Ziel: Berechnung von $W(i, p)$ für $p \in \{0, \ldots, nP\}$.

Sei $I \subseteq \{1, \ldots, i\}$ mit $\sum_{i \in I} p_i \geq p$ und kleinstmöglichen Gewicht, d. h. $\sum_{i \in I} w_i = W(i, p)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I} p_j \geq p$.
  $\Rightarrow W(i, p) = W(i-1, p)$
- Falls $i \in I$, so ist $I \setminus \{i\} \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I \setminus \{i\}} p_j \geq p - p_i$.
  $\Rightarrow \sum_{j \in I \setminus \{i\}} w_j = W(i-1, p-p_i)$
  $\Rightarrow W(i, p) = W(i-1, p-p_i) + w_i$

Sei für ein $i \geq 2$ und für alle $p \in \{0, \ldots, nP\}$ **der Wert $W(i-1, p)$ bekannt**.

Ziel: Berechnung von $W(i, p)$ für $p \in \{0, \ldots, nP\}$.

Sei $I \subseteq \{1, \ldots, i\}$ mit $\sum_{i \in I} p_i \geq p$ und kleinstmöglichen Gewicht, d. h. $\sum_{i \in I} w_i = W(i, p)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I} p_j \geq p$.
  $\Rightarrow W(i, p) = W(i-1, p)$
- Falls $i \in I$, so ist $I \setminus \{i\} \subseteq \{1, \ldots, i-1\}$ mit $\sum_{j \in I \setminus \{i\}} p_j \geq p - p_i$.
  $\Rightarrow \sum_{j \in I \setminus \{i\}} w_j = W(i-1, p-p_i)$
  $\Rightarrow W(i, p) = W(i-1, p-p_i) + w_i$

Insgesamt folgt $W(i, p) = \min\{W(i-1, p), W(i-1, p-p_i) + w_i\}$.

DynKP

1 // Sei $W(i,p) = 0$ für $i \in \{1,\ldots,n\}$ und $p \leq 0$.
2 $P := \max_{i\in\{1,\ldots,n\}} p_i$;
3 **for** $(p = 1; p \leq p_1; p{+}{+})$ $W(1,p) := w_1$;
4 **for** $(p = p_1 + 1; p \leq nP; p{+}{+})$ $W(1,p) := \infty$;
5 **for** $(i = 2; i \leq n; i{+}{+})$
6 **for** $(p = 1; p \leq nP; p{+}{+})$
7 $W(i,p) = \min\{W(i-1,p), W(i-1,p-p_i) + w_i\}$;
8 **return** maximales $p \in \{1,\ldots,nP\}$ mit $W(n,p) \leq t$

DynKP

1 // Sei $W(i,p) = 0$ für $i \in \{1,\ldots,n\}$ und $p \leq 0$.
2 $P := \max_{i\in\{1,\ldots,n\}} p_i$;
3 **for** $(p = 1; p \leq p_1; p{+}{+})$ $W(1,p) := w_1$;
4 **for** $(p = p_1 + 1; p \leq nP; p{+}{+})$ $W(1,p) := \infty$;
5 **for** $(i = 2; i \leq n; i{+}{+})$
6 **for** $(p = 1; p \leq nP; p{+}{+})$
7 $W(i,p) = \min\{W(i-1,p), W(i-1,p-p_i) + w_i\}$;
8 **return** maximales $p \in \{1,\ldots,nP\}$ mit $W(n,p) \leq t$

Theorem 2.12

Der Algorithmus DynKP bestimmt in Zeit $\Theta(n^2 P)$ den maximal erreichbaren Nutzen einer gegebenen Instanz des Rucksackproblems.

FPTAS für das Rucksackproblem

FPTAS-KP

Die Eingabe sei $(\mathcal{I}, \varepsilon)$ mit $\mathcal{I} = (p_1, \ldots, p_n, w_1, \ldots, w_n, t)$ und $w_i \leq t$ für alle i.

1 $P := \max_{i \in \{1,\ldots,n\}} p_i$;

2 $K := \frac{\varepsilon P}{n}$; // Skalierungsfaktor

3 **for** $i = 1$ **to** n **do** $p'_i = \lfloor p_i / K \rfloor$; // skaliere und runde die Nutzenwerte

4 Benutze Algorithmus DynKP, um die optimale Lösung für die Instanz $p'_1, \ldots, p'_n, w_1, \ldots, w_n, t$ des Rucksackproblems zu bestimmen.

Theorem 5.11

Der Algorithmus FPTAS-KP ist ein FPTAS für das Rucksackproblem mit einer Laufzeit von $O(n^3/\varepsilon)$.

Theorem 5.11

Der Algorithmus FPTAS-KP ist ein FPTAS für das Rucksackproblem mit einer Laufzeit von $O(n^3/\varepsilon)$.

Beweis:

Laufzeit des Algorithmus: Sei $P' = \max_{i\in\{1,\dots,n\}} p'_i$. Dann beträgt die Laufzeit von DynKP $\Theta(n^2P')$. Es gilt

$$P' = \max_{i\in\{1,\dots,n\}} \left\lfloor \frac{p_i}{K} \right\rfloor = \left\lfloor \frac{P}{K} \right\rfloor = \left\lfloor \frac{n}{\varepsilon} \right\rfloor \leq \frac{n}{\varepsilon}$$

und somit beträgt die Laufzeit von DynKP $\Theta(n^2P') = \Theta(n^3/\varepsilon)$.

Korrektheit:

Sei $I' \subseteq \{1, \ldots, n\}$ opt. Lösung für die Instanz mit den Nutzenwerten $p'_1, \ldots, p'_n$.

Sei $I \subseteq \{1, \ldots, n\}$ opt. Lösung für die Instanz mit den Nutzenwerten $p_1, \ldots, p_n$.

I' ist auch opt. Lösung für die Nutzenwerte $p^*_1, \ldots, p^*_n$ mit $p^*_i = K \cdot p'_i$.

Korrektheit:

Sei $I' \subseteq \{1, \ldots, n\}$ opt. Lösung für die Instanz mit den Nutzenwerten $p'_1, \ldots, p'_n$.

Sei $I \subseteq \{1, \ldots, n\}$ opt. Lösung für die Instanz mit den Nutzenwerten $p_1, \ldots, p_n$.

I' ist auch opt. Lösung für die Nutzenwerte $p^*_1, \ldots, p^*_n$ mit $p^*_i = K \cdot p'_i$.

Beispiel

Sei $n = 4$, $P = 50$ und $\varepsilon = \frac{4}{5}$. Dann ist $K = \frac{\varepsilon P}{n} = 10$.

Die verschiedenen Nutzenwerte könnten zum Beispiel wie folgt aussehen:

$$
\begin{array}{lll}
p_1 = 33 & p'_1 = 3 & p^*_1 = 30 \\
p_2 = 25 & p'_2 = 2 & p^*_2 = 20 \\
p_3 = 50 & p'_3 = 5 & p^*_3 = 50 \\
p_4 = 27 & p'_4 = 2 & p^*_4 = 20
\end{array}
$$

5.3 Rucksackproblem

Für eine Teilmenge $J \subseteq \{1, \ldots, n\}$ sei

$$p(J) = \sum_{i \in J} p_i \qquad \text{und} \qquad p^*(J) = \sum_{i \in J} p_i^*.$$

Für eine Teilmenge $J \subseteq \{1, \ldots, n\}$ sei

$$p(J) = \sum_{i \in J} p_i \qquad \text{und} \qquad p^*(J) = \sum_{i \in J} p_i^*.$$

Dann ist $p(I')$ der Wert der Lösung, die der Algorithmus FPTAS-KP ausgibt, und $p(I)$ ist der Wert OPT der optimalen Lösung.

Für eine Teilmenge $J \subseteq \{1, \ldots, n\}$ sei

$$p(J) = \sum_{i \in J} p_i \qquad \text{und} \qquad p^*(J) = \sum_{i \in J} p_i^*.$$

Dann ist $p(I')$ der Wert der Lösung, die der Algorithmus FPTAS-KP ausgibt, und $p(I)$ ist der Wert OPT der optimalen Lösung.

Es gilt

$$p_i^* = K \cdot \left\lfloor \frac{p_i}{K} \right\rfloor \geq K \left(\frac{p_i}{K} - 1 \right) = p_i - K$$

und

$$p_i^* = K \cdot \left\lfloor \frac{p_i}{K} \right\rfloor \leq p_i.$$

Für eine Teilmenge $J \subseteq \{1, \ldots, n\}$ sei

$$p(J) = \sum_{i \in J} p_i \qquad \text{und} \qquad p^*(J) = \sum_{i \in J} p_i^*.$$

Dann ist $p(I')$ der Wert der Lösung, die der Algorithmus FPTAS-KP ausgibt, und $p(I)$ ist der Wert OPT der optimalen Lösung.

Es gilt

$$p_i^* = K \cdot \left\lfloor \frac{p_i}{K} \right\rfloor \geq K \left(\frac{p_i}{K} - 1 \right) = p_i - K$$

und

$$p_i^* = K \cdot \left\lfloor \frac{p_i}{K} \right\rfloor \leq p_i.$$

Dementsprechend gilt für jede Teilmenge $J \subseteq \{1, \ldots, n\}$

$$p^*(J) \in [p(J) - nK, p(J)] .$$

I' ist optimale Lösung für die Nutzenwerte $p_1^*, \ldots, p_n^*$.
Es gilt also insbesondere $p^*(I') \geq p^*(I)$ und somit

$$p(I') \geq p^*(I') \geq p^*(I) \geq p(I) - nK.$$

I' ist optimale Lösung für die Nutzenwerte $p_1^*, \ldots, p_n^*$.
Es gilt also insbesondere $p^*(I') \geq p^*(I)$ und somit

$$p(I') \geq p^*(I') \geq p^*(I) \geq p(I) - nK.$$

Da jedes Objekt alleine in den Rucksack passt, gilt $p(I) \geq P$ und damit auch

$$\frac{p(I')}{\mathrm{OPT}} = \frac{p(I')}{p(I)} \geq \frac{p(I) - nK}{p(I)} = 1 - \frac{\varepsilon P}{p(I)} \geq 1 - \varepsilon. \qquad \square$$