Algorithmentheorie

Daniel Neuen (Universität Bremen)

WiSe 2023/24

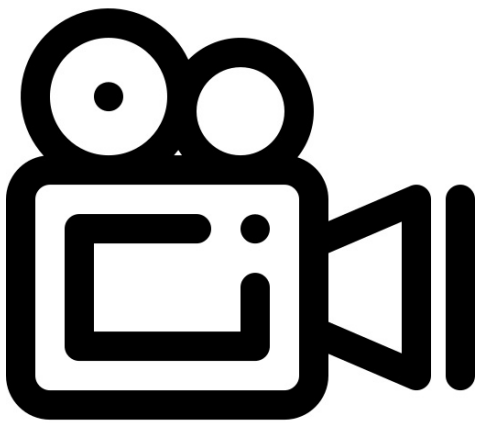
Greedy Algorithmen

4. Vorlesung

Aufzeichnung der Vorlesung

Diese Vorlesung wird aufgezeichnet und live gestreamt.

- Aufzeichnungen nur der Lehrenden durch sich selbst.
- Bei Rückfragen aus dem Auditorium und Diskussion bitte deutlich anzeigen, falls das Mikro stumm geschaltet werden soll.



Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

- Trifft in jedem Schritt eine lokal optimale Entscheidung.

Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

- Trifft in jedem Schritt eine lokal optimale Entscheidung.
  $\rightarrow$ ist optimal bezüglich eines „kurzsichtigen“ Kriteriums

Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

- Trifft in jedem Schritt eine lokal optimale Entscheidung.
  $\rightarrow$ ist optimal bezüglich eines „kurzsichtigen“ Kriteriums
- Das Kriterium ist „einfach“ und leicht auszuwerten.

Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

- Trifft in jedem Schritt eine lokal optimale Entscheidung.
  $\rightarrow$ ist optimal bezüglich eines „kurzsichtigen“ Kriteriums
- Das Kriterium ist „einfach“ und leicht auszuwerten.
- Getroffene Entscheidungen werden nicht mehr zurück genommen.

Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

- Trifft in jedem Schritt eine lokal optimale Entscheidung.
  $\rightarrow$ ist optimal bezüglich eines „kurzsichtigen“ Kriteriums
- Das Kriterium ist „einfach“ und leicht auszuwerten.
- Getroffene Entscheidungen werden nicht mehr zurück genommen.

- Greedy Algorithmen sind oft schnell,

Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

- Trifft in jedem Schritt eine lokal optimale Entscheidung.
  $\rightarrow$ ist optimal bezüglich eines „kurzsichtigen“ Kriteriums
- Das Kriterium ist „einfach“ und leicht auszuwerten.
- Getroffene Entscheidungen werden nicht mehr zurück genommen.

- Greedy Algorithmen sind oft schnell,
- finden manchmal eine optimale Lösung,

Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

- Trifft in jedem Schritt eine lokal optimale Entscheidung.
  $\rightarrow$ ist optimal bezüglich eines „kurzsichtigen“ Kriteriums
- Das Kriterium ist „einfach“ und leicht auszuwerten.
- Getroffene Entscheidungen werden nicht mehr zurück genommen.

- Greedy Algorithmen sind oft schnell,
- finden manchmal eine optimale Lösung,
- aber nicht immer! Sie sind manchmal beliebig schlecht.

Greedy Algorithmen

Greedy Algorithmus (greedy = gierig)

- Trifft in jedem Schritt eine lokal optimale Entscheidung.
  $\rightarrow$ ist optimal bezüglich eines „kurzsichtigen“ Kriteriums
- Das Kriterium ist „einfach“ und leicht auszuwerten.
- Getroffene Entscheidungen werden nicht mehr zurück genommen.

- Greedy Algorithmen sind oft schnell,
- finden manchmal eine optimale Lösung,
- aber nicht immer! Sie sind manchmal beliebig schlecht.
- Achtung: wiederholte Wahl eines lokalen Optimums führt nicht unbedingt zu einem globalen Optimum.

Optimierungsprobleme

Optimierungsproblem

Gegeben: eine Menge X von zulässigen Lösungen und
eine Zielfunktion $f : X \to \mathbb{R}$

Finde: eine Lösung $x^* \in X$ mit maximalem (min) Zielfunktionswert,
d.h. für alle $x \in X$ gilt: $f(x^*) \geq f(x)$ (bzw. $f(x^*) \leq f(x)$).

Optimierungsprobleme

Optimierungsproblem

Gegeben: eine Menge X von zulässigen Lösungen und
eine Zielfunktion $f : X \to \mathbb{R}$

Finde: eine Lösung $x^* \in X$ mit maximalem (min) Zielfunktionswert,
d.h. für alle $x \in X$ gilt: $f(x^*) \geq f(x)$ (bzw. $f(x^*) \leq f(x)$).

- Finde den kürzesten Weg.

Optimierungsprobleme

Optimierungsproblem

Gegeben: eine Menge X von zulässigen Lösungen und
eine Zielfunktion $f : X \to \mathbb{R}$

Finde: eine Lösung $x^* \in X$ mit maximalem (min) Zielfunktionswert,
d.h. für alle $x \in X$ gilt: $f(x^*) \geq f(x)$ (bzw. $f(x^*) \leq f(x)$).

- Finde den kürzesten Weg.
- Finde eine wertvollste Packung eines Containers.

Optimierungsprobleme

Optimierungsproblem

Gegeben: eine Menge X von zulässigen Lösungen und
eine Zielfunktion $f : X \to \mathbb{R}$

Finde: eine Lösung $x^* \in X$ mit maximalem (min) Zielfunktionswert,
d.h. für alle $x \in X$ gilt: $f(x^*) \geq f(x)$ (bzw. $f(x^*) \leq f(x)$).

- Finde den kürzesten Weg.
- Finde eine wertvollste Packung eines Containers.
- Finde einen „besten“ Zeitplan....

Optimierungsprobleme

Optimierungsproblem

Gegeben: eine Menge X von zulässigen Lösungen und
eine Zielfunktion $f : X \to \mathbb{R}$

Finde: eine Lösung $x^* \in X$ mit maximalem (min) Zielfunktionswert,
d.h. für alle $x \in X$ gilt: $f(x^*) \geq f(x)$ (bzw. $f(x^*) \leq f(x)$).

- Finde den kürzesten Weg.
- Finde eine wertvollste Packung eines Containers.
- Finde einen „besten“ Zeitplan....
- Zielfunktion bewertet die Qualität zulässiger Lösungen

Einfaches Beispiel: Geld auszahlen

Gegeben:

Quelle: Bundesministerium der Finanzen

Einfaches Beispiel: Geld auszahlen

Gegeben:

Quelle: Bundesministerium der Finanzen

Gesucht: Kleinste Anzahl an Münzen um Betrag S auszugeben.

Einfaches Beispiel: Geld auszahlen

Gegeben:

Quelle: Bundesministerium der Finanzen

Gesucht: Kleinste Anzahl an Münzen um Betrag S auszugeben.

Greedy Algorithmus:

```
1 Multiset M = ∅
2 while S ≠ 0 do
3 |   Finde Münze mit größtem Wert z, so dass z ≤ S
4 |   M := M ∪ {z}
5 |   S := S − z
6 return M
```

Einfaches Beispiel: Geld auszahlen

Quelle: Bundesministerium der Finanzen

Beispiel: 20 Euro + 34 Cent

Einfaches Beispiel: Geld auszahlen

Quelle: Bundesministerium der Finanzen

Beispiel: 20 Euro + 34 Cent

- Greedy Lösung: $10 \times 2€$, 1×20ct, 1×10ct, 2×2ct.

Einfaches Beispiel: Geld auszahlen

Quelle: Bundesministerium der Finanzen

Beispiel: 20 Euro + 34 Cent

- Greedy Lösung: $10 \times 2€$, 1×20ct, 1×10ct, 2×2ct.

Quiz:

a) Greedy ist optimal für unser Münzsystem.

b) Greedy ist optimal für alle Münzsysteme $m_1 < m_2 < \ldots < m_n$ mit $m_1 = 1$ct.

c) Greedy ist selbst für unser Münzsystem nicht optimal.

Einfaches Beispiel: Geld auszahlen

Quelle: Bundesministerium der Finanzen

Beispiel: 20 Euro + 34 Cent

- Greedy Lösung: $10 \times 2€$, 1×20ct, 1×10ct, 2×2ct.

Quiz:

a) Greedy ist optimal für unser Münzsystem. Ja. (Nicht so leicht.)

b) Greedy ist optimal für alle Münzsysteme $m_1 < m_2 < \ldots < m_n$ mit $m_1 = 1$ct.

c) Greedy ist selbst für unser Münzsystem nicht optimal. Nein, s. a).

Einfaches Beispiel: Geld auszahlen

Quelle: Bundesministerium der Finanzen

Beispiel: 20 Euro + 34 Cent

- Greedy Lösung: $10 \times 2€$, 1×20ct, 1×10ct, 2×2ct.

Quiz:

a) Greedy ist optimal für unser Münzsystem. Ja. (Nicht so leicht.)

b) Greedy ist optimal für alle Münzsysteme $m_1 < m_2 < \ldots < m_n$ mit $m_1 = 1$ct. Nein, betrachte $\{8, 7, 1\}$ mit $S = 14$.

c) Greedy ist selbst für unser Münzsystem nicht optimal. Nein, s. a).

Interval Scheduling

Beispiel: Interval Scheduling (Stundenplanung)

- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.

Beispiel: Interval Scheduling (Stundenplanung)

- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.
- Beispiele:

Beispiel: Interval Scheduling (Stundenplanung)

- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.
- Beispiele:
  - Buchung von Räumen für Vorlesungen.

Beispiel: Interval Scheduling (Stundenplanung)

- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.
- Beispiele:
  - Buchung von Räumen für Vorlesungen.
  - Produktionsprozesse auf Maschinen.

Beispiel: Interval Scheduling (Stundenplanung)

- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.
- Beispiele:
  - Buchung von Räumen für Vorlesungen.
  - Produktionsprozesse auf Maschinen.
- Die Prozesse $1, \ldots, n$ haben Startzeiten s_i und Endzeiten (deadlines) f_i für $1 \leq i \leq n$.

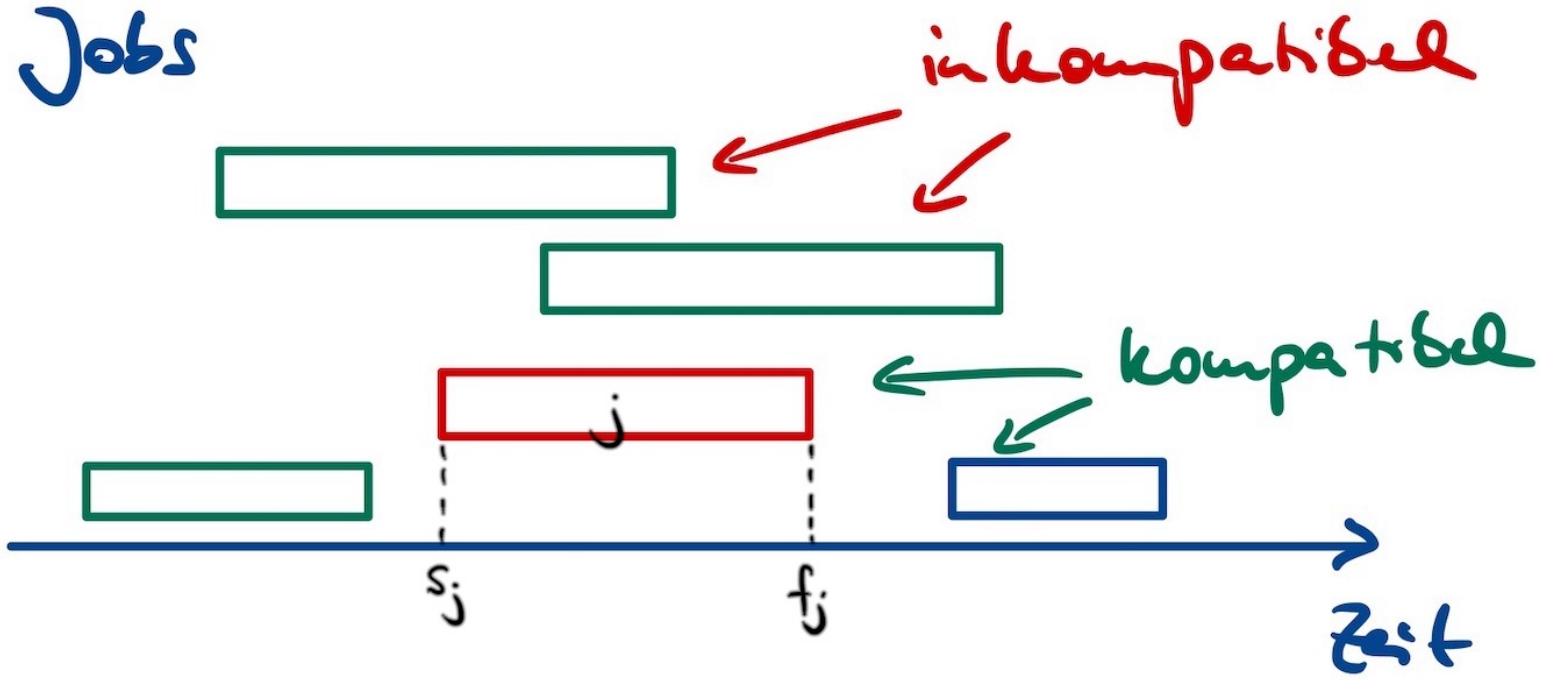
Beispiel: Interval Scheduling (Stundenplanung)

- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.
- Beispiele:
  - Buchung von Räumen für Vorlesungen.
  - Produktionsprozesse auf Maschinen.
- Die Prozesse $1, \ldots, n$ haben Startzeiten s_i und Endzeiten (deadlines) f_i für $1 \leq i \leq n$.
- Prozess i und j sind kompatibel, wenn $[s_i, f_i) \cap [s_j, f_j) = \emptyset$.

Beispiel: Interval Scheduling (Stundenplanung)

- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.
- Beispiele:
  - Buchung von Räumen für Vorlesungen.
  - Produktionsprozesse auf Maschinen.
- Die Prozesse $1, \ldots, n$ haben Startzeiten s_i und Endzeiten (deadlines) f_i für $1 \leq i \leq n$.
- Prozess i und j sind kompatibel, wenn $[s_i, f_i) \cap [s_j, f_j) = \emptyset$.
- **Ziel**: Finde eine Teilmenge maximaler Größe (Kardinalität) von kompatiblen Prozessen.

Beispiel: Interval Scheduling



Interval Scheduling: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job, der mit den bisher gewählten Jobs kompatibel ist.

Interval Scheduling: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job, der mit den bisher gewählten Jobs kompatibel ist.

Mögliche Auswahlkriterien:

Interval Scheduling: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job, der mit den bisher gewählten Jobs kompatibel ist.

Mögliche Auswahlkriterien:

- Früheste Startzeit: Berücksichtige Jobs in aufsteigender Reihenfolge von s_i.

Interval Scheduling: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job, der mit den bisher gewählten Jobs kompatibel ist.

Mögliche Auswahlkriterien:

- Früheste Startzeit: Berücksichtige Jobs in aufsteigender Reihenfolge von s_i.
- Früheste Beendigungszeit: Berücksichtige Jobs in aufsteigender Reihenfolge von f_i.

Interval Scheduling: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job, der mit den bisher gewählten Jobs kompatibel ist.

Mögliche Auswahlkriterien:

- Früheste Startzeit: Berücksichtige Jobs in aufsteigender Reihenfolge von s_i.
- Früheste Beendigungszeit: Berücksichtige Jobs in aufsteigender Reihenfolge von f_i.
- Kleinstes Intervall: Berücksichtige Jobs in aufsteigender Reihenfolge von $f_i - s_i$.

Interval Scheduling: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job, der mit den bisher gewählten Jobs kompatibel ist.

Mögliche Auswahlkriterien:

- Früheste Startzeit: Berücksichtige Jobs in aufsteigender Reihenfolge von s_i.
- Früheste Beendigungszeit: Berücksichtige Jobs in aufsteigender Reihenfolge von f_i.
- Kleinstes Intervall: Berücksichtige Jobs in aufsteigender Reihenfolge von $f_i - s_i$.
- Wenigste Konflikte: Zähle für jeden Job die Anzahl der nichtkompatiblen Jobs. Berücksichtige Jobs in aufsteigender Reihenfolge dieser Konflikte.

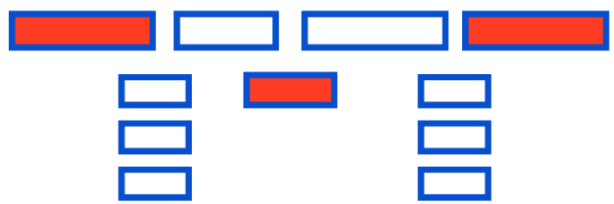
Beispiele Auswahlstrategien

Gegenbeispiele für Optimalität für Greedy

- nach frühester Startzeit

- nach kleinstem Intervall

- nach wenigsten Konflikten



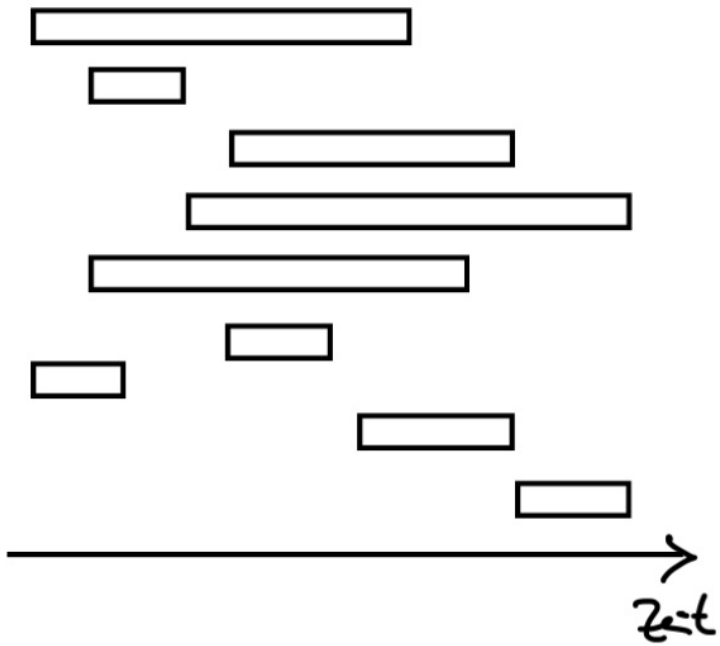
Auswahlstrategie: früheste Beendigungszeit

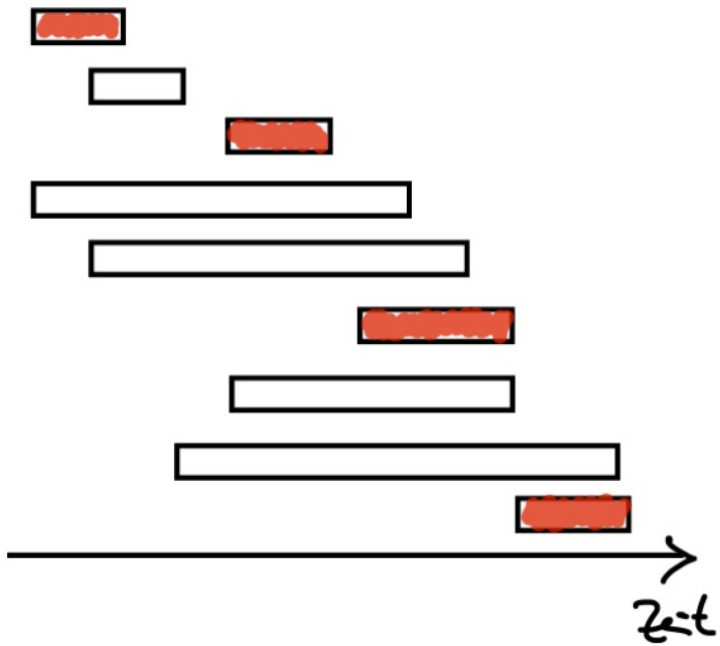
Satz

Der Greedy Algorithmus mit Auswahlstrategie „früheste Beendigungszeit“ liefert einen optimalen Plan für Interval Scheduling.

```
1 Sortiere Jobs, so dass f_1 ≤ f_2 ≤ ... ≤ f_n
2 A := ∅
3 t := 0
4 for i := 1 to n do
5     if t ≤ s_i then
6         A := A ∪ {i}
7         t := f_i
8 return A
```

Beispiel





Analyse

Korrektheit: Der Algorithmus produziert einen gültigen Plan.

- **Schleifeninvariante**: Beim Start der Schleife sind alle Jobs in A konfliktfrei und alle Jobs in A sind zur Zeit t beendet.

Analyse

Korrektheit: Der Algorithmus produziert einen gültigen Plan.

- **Schleifeninvariante**: Beim Start der Schleife sind alle Jobs in A konfliktfrei und alle Jobs in A sind zur Zeit t beendet.
- IA: $A = \emptyset$, $t = 0$ ✓.

Analyse

Korrektheit: Der Algorithmus produziert einen gültigen Plan.

- **Schleifeninvariante**: Beim Start der Schleife sind alle Jobs in A konfliktfrei und alle Jobs in A sind zur Zeit t beendet.
- IA: $A = \emptyset$, $t = 0$ ✓.
- IS:
  - Nach IV ist t die Beendigungszeit aller Jobs in A.

Analyse

Korrektheit: Der Algorithmus produziert einen gültigen Plan.

- **Schleifeninvariante**: Beim Start der Schleife sind alle Jobs in A konfliktfrei und alle Jobs in A sind zur Zeit t beendet.
- IA: $A = \emptyset$, $t = 0$ ✓.
- IS:
  - Nach IV ist t die Beendigungszeit aller Jobs in A.
  - Es wird Job i zu A nur dann hinzugefügt, wenn $t \leq s_i$. Also ist A auch nach dem Schleifendurchlauf konfliktfrei.

Analyse

Korrektheit: Der Algorithmus produziert einen gültigen Plan.

- **Schleifeninvariante**: Beim Start der Schleife sind alle Jobs in A konfliktfrei und alle Jobs in A sind zur Zeit t beendet.
- IA: $A = \emptyset$, $t = 0$ ✓.
- IS:
  - Nach IV ist t die Beendigungszeit aller Jobs in A.
  - Es wird Job i zu A nur dann hinzugefügt, wenn $t \leq s_i$. Also ist A auch nach dem Schleifendurchlauf konfliktfrei.
  - Da die Jobs aufsteigend nach Beendigungszeit sortiert sind, ist f_i nach dem Hinzufügen von Job i die größte Beendigungszeit aller Jobs in A.

Analyse

Korrektheit: Der Algorithmus produziert einen gültigen Plan.

- **Schleifeninvariante**: Beim Start der Schleife sind alle Jobs in A konfliktfrei und alle Jobs in A sind zur Zeit t beendet.
- IA: $A = \emptyset$, $t = 0$ ✓.
- IS:
  - Nach IV ist t die Beendigungszeit aller Jobs in A.
  - Es wird Job i zu A nur dann hinzugefügt, wenn $t \leq s_i$. Also ist A auch nach dem Schleifendurchlauf konfliktfrei.
  - Da die Jobs aufsteigend nach Beendigungszeit sortiert sind, ist f_i nach dem Hinzufügen von Job i die größte Beendigungszeit aller Jobs in A.
  - Damit ist die Invariante wieder hergestellt.

Analyse

Korrektheit: Der Algorithmus produziert einen gültigen Plan.

- **Schleifeninvariante**: Beim Start der Schleife sind alle Jobs in A konfliktfrei und alle Jobs in A sind zur Zeit t beendet.
- IA: $A = \emptyset$, $t = 0$ ✓.
- IS:
  - Nach IV ist t die Beendigungszeit aller Jobs in A.
  - Es wird Job i zu A nur dann hinzugefügt, wenn $t \leq s_i$. Also ist A auch nach dem Schleifendurchlauf konfliktfrei.
  - Da die Jobs aufsteigend nach Beendigungszeit sortiert sind, ist f_i nach dem Hinzufügen von Job i die größte Beendigungszeit aller Jobs in A.
  - Damit ist die Invariante wieder hergestellt.

Laufzeit: Sortieren + n Schritte in der Schleife:

Analyse

Korrektheit: Der Algorithmus produziert einen gültigen Plan.

- **Schleifeninvariante**: Beim Start der Schleife sind alle Jobs in A konfliktfrei und alle Jobs in A sind zur Zeit t beendet.
- IA: $A = \emptyset$, $t = 0$ ✓.
- IS:
  - Nach IV ist t die Beendigungszeit aller Jobs in A.
  - Es wird Job i zu A nur dann hinzugefügt, wenn $t \leq s_i$. Also ist A auch nach dem Schleifendurchlauf konfliktfrei.
  - Da die Jobs aufsteigend nach Beendigungszeit sortiert sind, ist f_i nach dem Hinzufügen von Job i die größte Beendigungszeit aller Jobs in A.
  - Damit ist die Invariante wieder hergestellt.

Laufzeit: Sortieren + n Schritte in der Schleife:

$$\mathcal{O}(n \log n) + \mathcal{O}(n) = \mathcal{O}(n \log n).$$

Analyse II

Optimalität: Der Algorithmus liefert eine optimale Lösung.

Beweis.

- Angenommen, Greedy ist nicht optimal.

Analyse II

Optimalität: Der Algorithmus liefert eine optimale Lösung.

Beweis.

- Angenommen, Greedy ist nicht optimal.
- Betrachte die Jobs $j_1, j_2, \ldots$ einer optimalen Lösung und $i_1, i_2, \ldots$ der Greedy Lösung.

Analyse II

Optimalität: Der Algorithmus liefert eine optimale Lösung.

Beweis.

- Angenommen, Greedy ist nicht optimal.
- Betrachte die Jobs $j_1, j_2, \ldots$ einer optimalen Lösung und $i_1, i_2, \ldots$ der Greedy Lösung.
- Beide Jobmengen aufsteigend nach $f_\star$ indiziert.

Analyse II

Optimalität: Der Algorithmus liefert eine optimale Lösung.

Beweis.

- Angenommen, Greedy ist nicht optimal.
- Betrachte die Jobs $j_1, j_2, \ldots$ einer optimalen Lösung und $i_1, i_2, \ldots$ der Greedy Lösung.
- Beide Jobmengen aufsteigend nach $f_\star$ indiziert.
- Unter den optimalen Lösungen, haben wir diejenige gewählt, die bis zu einem maximalen Index $(\ell - 1)$ mit Greedy übereinstimmt.

Analyse II

Optimalität: Der Algorithmus liefert eine optimale Lösung.

Beweis.

- Angenommen, Greedy ist nicht optimal.
- Betrachte die Jobs $j_1, j_2, \ldots$ einer optimalen Lösung und $i_1, i_2, \ldots$ der Greedy Lösung.
- Beide Jobmengen aufsteigend nach $f_\star$ indiziert.
- Unter den optimalen Lösungen, haben wir diejenige gewählt, die bis zu einem maximalen Index $(\ell - 1)$ mit Greedy übereinstimmt.
- Betrachte den kleinsten Index ℓ, bei dem die Jobs voneinander abweichen.

Optimalität: Der Algorithmus liefert eine optimale Lösung.

Beweis.

- Angenommen, Greedy ist nicht optimal.
- Betrachte die Jobs $j_1, j_2, \ldots$ einer optimalen Lösung und $i_1, i_2, \ldots$ der Greedy Lösung.
- Beide Jobmengen aufsteigend nach $f_\star$ indiziert.
- Unter den optimalen Lösungen, haben wir diejenige gewählt, die bis zu einem maximalen Index $(\ell - 1)$ mit Greedy übereinstimmt.
- Betrachte den kleinsten Index ℓ, bei dem die Jobs voneinander abweichen. Also gilt $f_{i_{\ell-1}} = f_{j_{\ell-1}} \leq s_{i_\ell}, s_{j_\ell}$ und $f_{i_\ell} \leq f_{j_\ell}$.

Analyse II

Optimalität: Der Algorithmus liefert eine optimale Lösung.

Beweis.

- Angenommen, Greedy ist nicht optimal.
- Betrachte die Jobs $j_1, j_2, \ldots$ einer optimalen Lösung und $i_1, i_2, \ldots$ der Greedy Lösung.
- Beide Jobmengen aufsteigend nach $f_\star$ indiziert.
- Unter den optimalen Lösungen, haben wir diejenige gewählt, die bis zu einem maximalen Index $(\ell - 1)$ mit Greedy übereinstimmt.
- Betrachte den kleinsten Index ℓ, bei dem die Jobs voneinander abweichen. Also gilt $f_{i_{\ell-1}} = f_{j_{\ell-1}} \leq s_{i_\ell}, s_{j_\ell}$ und $f_{i_\ell} \leq f_{j_\ell}$.
- Dann ist $j_1, j_2, \ldots, j_{\ell-1}, i_\ell, j_{\ell+1} \ldots, j_n$ auch optimale Lösung.

Optimalität: Der Algorithmus liefert eine optimale Lösung.

Beweis.

- Angenommen, Greedy ist nicht optimal.
- Betrachte die Jobs $j_1, j_2, \ldots$ einer optimalen Lösung und $i_1, i_2, \ldots$ der Greedy Lösung.
- Beide Jobmengen aufsteigend nach $f_\star$ indiziert.
- Unter den optimalen Lösungen, haben wir diejenige gewählt, die bis zu einem maximalen Index $(\ell - 1)$ mit Greedy übereinstimmt.
- Betrachte den kleinsten Index ℓ, bei dem die Jobs voneinander abweichen. Also gilt $f_{i_{\ell-1}} = f_{j_{\ell-1}} \leq s_{i_\ell}, s_{j_\ell}$ und $f_{i_\ell} \leq f_{j_\ell}$.
- Dann ist $j_1, j_2, \ldots, j_{\ell-1}, i_\ell, j_{\ell+1} \ldots, j_n$ auch optimale Lösung.
- Es gilt $i_1 = j_1, i_2 = j_2, \ldots, i_\ell = j_\ell$, im Widerspruch zur Annahme, dass optimale Lösung bis max. Index mit Greedy übereinstimmte.

□

Interval partitioning – Coloring

Beispiel: Interval partitioning/coloring

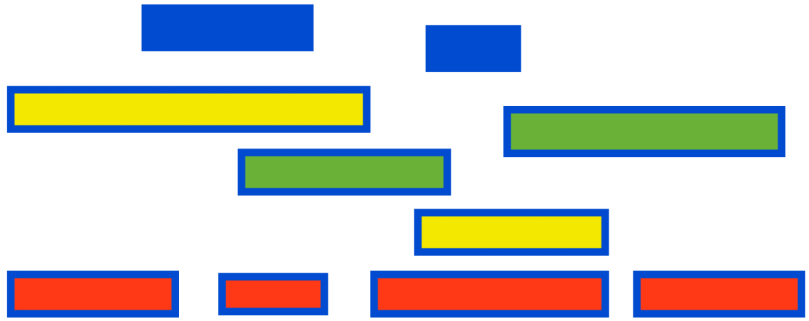
- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.
- Beispiele:
  - Buchung von Räumen für Vorlesungen.
  - Produktionsprozesse auf Maschinen.
- Die Prozesse $1, \ldots, n$ haben Startzeiten s_i und Endzeiten (deadlines) f_i für $1 \leq i \leq n$.
- Prozess i und j sind kompatibel, wenn $[s_i, f_i) \cap [s_j, f_j) = \emptyset$.

Beispiel: Interval partitioning/coloring

- Prozesse/Aktivitäten/Aufgaben brauchen exklusiven Zugriff auf Ressourcen/Maschinen.
- Beispiele:
  - Buchung von Räumen für Vorlesungen.
  - Produktionsprozesse auf Maschinen.
- Die Prozesse $1, \ldots, n$ haben Startzeiten s_i und Endzeiten (deadlines) f_i für $1 \leq i \leq n$.
- Prozess i und j sind kompatibel, wenn $[s_i, f_i) \cap [s_j, f_j) = \emptyset$.
- **Ziel**: Setze alle Prozesse um mit der minimalen Anzahl an Ressourcen.

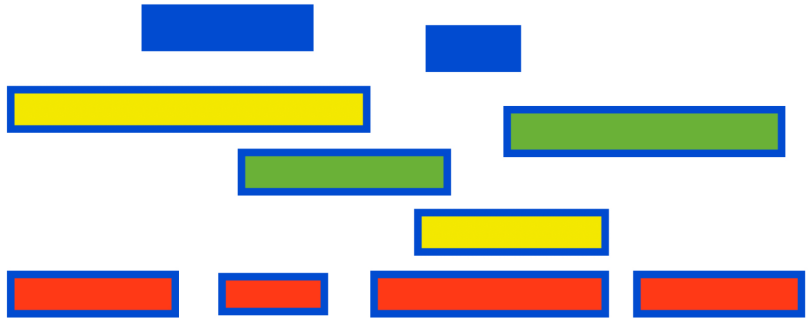
Beispiel: Interval partitioning/coloring

Beispiel: Jobs auf gleicher Maschine haben gleiche Farbe



Beispiel: Interval partitioning/coloring

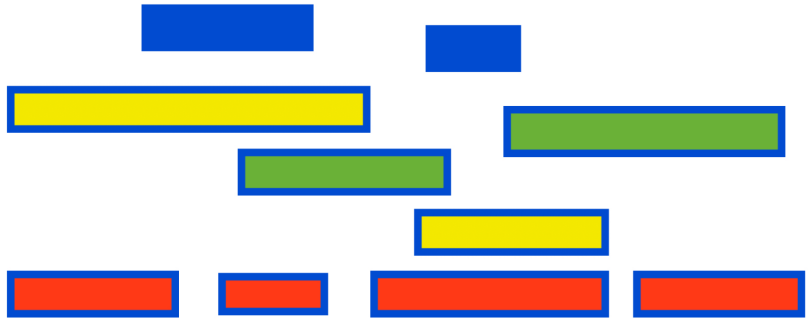
Beispiel: Jobs auf gleicher Maschine haben gleiche Farbe



Greedy Idee. Finde maximale Anzahl von Intervallen für eine Maschine (= Interval Scheduling, Greedy optimal).

Beispiel: Interval partitioning/coloring

Beispiel: Jobs auf gleicher Maschine haben gleiche Farbe



Greedy Idee. Finde maximale Anzahl von Intervallen für eine Maschine (= Interval Scheduling, Greedy optimal). → nicht optimal

Interval partitioning: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job und ordne ihn einer beliebigen freien Maschine zu; starte eine neue Maschine, falls keine Maschine frei ist.

Interval partitioning: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job und ordne ihn einer beliebigen freien Maschine zu; starte eine neue Maschine, falls keine Maschine frei ist.

Mögliche Auswahlkriterien:

Interval partitioning: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job und ordne ihn einer beliebigen freien Maschine zu; starte eine neue Maschine, falls keine Maschine frei ist.

Mögliche Auswahlkriterien:

- Früheste Startzeit: Berücksichtige Jobs in aufsteigender Reihenfolge von s_i.

Interval partitioning: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job und ordne ihn einer beliebigen freien Maschine zu; starte eine neue Maschine, falls keine Maschine frei ist.

Mögliche Auswahlkriterien:

- Früheste Startzeit: Berücksichtige Jobs in aufsteigender Reihenfolge von s_i.
- Früheste Beendigungszeit: Berücksichtige Jobs in aufsteigender Reihenfolge von f_i.

Interval partitioning: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job und ordne ihn einer beliebigen freien Maschine zu; starte eine neue Maschine, falls keine Maschine frei ist.

Mögliche Auswahlkriterien:

- Früheste Startzeit: Berücksichtige Jobs in aufsteigender Reihenfolge von s_i.
- Früheste Beendigungszeit: Berücksichtige Jobs in aufsteigender Reihenfolge von f_i.
- Kleinstes Intervall: Berücksichtige Jobs in aufsteigender Reihenfolge von $f_i - s_i$.

Interval partitioning: Greedy Algorithmus

Greedy-Ansatz: Wähle einen Job und ordne ihn einer beliebigen freien Maschine zu; starte eine neue Maschine, falls keine Maschine frei ist.

Mögliche Auswahlkriterien:

- Früheste Startzeit: Berücksichtige Jobs in aufsteigender Reihenfolge von s_i.
- Früheste Beendigungszeit: Berücksichtige Jobs in aufsteigender Reihenfolge von f_i.
- Kleinstes Intervall: Berücksichtige Jobs in aufsteigender Reihenfolge von $f_i - s_i$.
- Wenigste Konflikte: Zähle für jeden Job die Anzahl der nichtkompatiblen Jobs. Berücksichtige Jobs in aufsteigender Reihenfolge dieser Konflikte.

Beispiele Auswahlstrategien

Gegenbeispiele für Optimalität für Greedy

- nach frühester Fertigstellungszeit sowie nach kleinstem Intervall

- nach wenigsten Konflikten

Auswahlstrategie: früheste Startzeit

Satz

Der Greedy Algorithmus mit Auswahlstrategie "früheste Startzeit" liefert einen optimalen Plan für Interval partition/coloring.

```
1  Sortiere Jobs, so dass s_1 ≤ s_2 ≤ ... ≤ s_n
2  m := 0                          ← Zahl der Maschinen.
3  for i := 1 to n do
4      if Job i mit existierender Maschine kompatibel then
5          Plane Job i auf beliebiger solcher Maschine
6      else
7          Starte neue Maschine m + 1
8          Plane Job i auf Maschine m + 1
9          m := m + 1
10 return Planung
```

Analyse

- Korrektheit: Der Algorithmus produziert einen gültigen Plan.

Analyse

- Korrektheit: Der Algorithmus produziert einen gültigen Plan.
- Laufzeit: $\mathcal{O}(n \log n)$.

Analyse

- Korrektheit: Der Algorithmus produziert einen gültigen Plan.
- Laufzeit: $\mathcal{O}(n \log n)$.
- Optimalität:

Analyse

- Korrektheit: Der Algorithmus produziert einen gültigen Plan.
- Laufzeit: $\mathcal{O}(n \log n)$.
- Optimalität:
  - Die Tiefe einer Menge von Intervallen ist die maximale Zahl an Intervallen, die sich an einem Punkt schneiden.

Analyse

- Korrektheit: Der Algorithmus produziert einen gültigen Plan.
- Laufzeit: $\mathcal{O}(n \log n)$.
- Optimalität:
  - Die Tiefe einer Menge von Intervallen ist die maximale Zahl an Intervallen, die sich an einem Punkt schneiden.
  - Die Zahl der Maschinen muss größer oder gleich der Tiefe der Jobintervalle sein.

Analyse

- Korrektheit: Der Algorithmus produziert einen gültigen Plan.
- Laufzeit: $\mathcal{O}(n \log n)$.
- Optimalität:
  - Die Tiefe einer Menge von Intervallen ist die maximale Zahl an Intervallen, die sich an einem Punkt schneiden.
  - Die Zahl der Maschinen muss größer oder gleich der Tiefe der Jobintervalle sein.
  - Behauptung: Die Greedy Heuristik mit frühester Startzeit findet einen Plan mit maximal Tiefe vielen Maschinen, ist also optimal.

Analyse

Lemma

Die Greedy Heuristik mit frühester Startzeit findet einen Plan mit höchstens Tiefe vielen Maschinen.

Analyse

Lemma

Die Greedy Heuristik mit frühester Startzeit findet einen Plan mit höchstens Tiefe vielen Maschinen.

Beweis.

- Sei m die Anzahl der Maschinen, die der Greedy Algorithmus startet.

Analyse

Lemma

Die Greedy Heuristik mit frühester Startzeit findet einen Plan mit höchstens Tiefe vielen Maschinen.

Beweis.

- Sei m die Anzahl der Maschinen, die der Greedy Algorithmus startet.
- Maschine m wurde gestartet, weil es einen Job j gab, der auf jeder anderen Maschine mit mindestens einem Job inkompatibel war.

Analyse

Lemma

Die Greedy Heuristik mit frühester Startzeit findet einen Plan mit höchstens Tiefe vielen Maschinen.

Beweis.

- Sei m die Anzahl der Maschinen, die der Greedy Algorithmus startet.
- Maschine m wurde gestartet, weil es einen Job j gab, der auf jeder anderen Maschine mit mindestens einem Job inkompatibel war.
- Da die Jobs nach Startzeiten sortiert sind, schneiden sich die Intervalle dieser $m-1$ Jobs auf den anderen Maschinen sowie j alle in s_j.

Analyse

Lemma

Die Greedy Heuristik mit frühester Startzeit findet einen Plan mit höchstens Tiefe vielen Maschinen.

Beweis.

- Sei m die Anzahl der Maschinen, die der Greedy Algorithmus startet.
- Maschine m wurde gestartet, weil es einen Job j gab, der auf jeder anderen Maschine mit mindestens einem Job inkompatibel war.
- Da die Jobs nach Startzeiten sortiert sind, schneiden sich die Intervalle dieser $m-1$ Jobs auf den anderen Maschinen sowie j alle in s_j.
- Also ist die Tiefe der Intervalle m, also gleich der Anzahl von Maschinen. □

Das Rucksackproblem

(mit teilbaren Gegenständen)

Fractional Knapsack Problem

Beispiel Rucksackproblem

Gegeben: Menge von n Objekten U mit

- Gewichten: $w : U \to \mathbb{N}$
- Nutzen: $p : U \to \mathbb{N}$
- Gewichtsschranke $B \in \mathbb{N}$

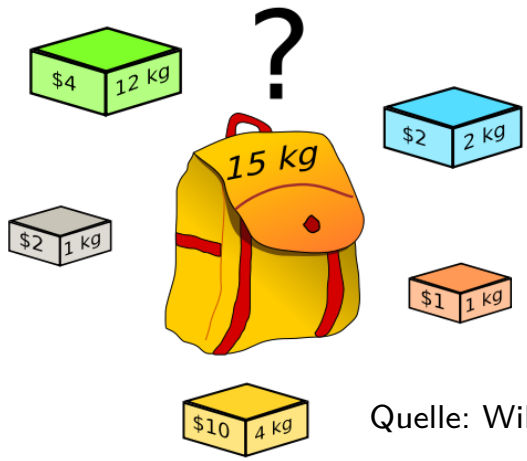
Beispiel Rucksackproblem

Gegeben: Menge von n Objekten U mit

- Gewichten: $w : U \to \mathbb{N}$
- Nutzen: $p : U \to \mathbb{N}$
- Gewichtsschranke $B \in \mathbb{N}$

Gesucht: Teilmenge $K \subseteq U$ mit

- $\sum_{u \in K} w(u) \leq B$
- $\sum_{u \in K} p(u)$ maximal



Quelle: Wikipedia

Greedy Algorithmus

Greedy Ansatz

Wähle nacheinander Objekte mit maximalem Nutzen-pro-Gewicht Verhältnis, d.h. $\arg\max_{u \in U} \frac{p(u)}{w(u)}$.

Greedy Algorithmus

Greedy Ansatz

Wähle nacheinander Objekte mit maximalem Nutzen-pro-Gewicht Verhältnis, d.h. $\arg\max_{u \in U} \frac{p(u)}{w(u)}$.

1 Sortiere Objekte, so dass $\frac{p(1)}{w(1)} \geq \frac{p(2)}{w(2)} \geq \cdots \geq \frac{p(n)}{w(n)}$.
2 $W := 0$ $\leftarrow$ aktueller Füllstand (Gewicht)
3 $A := \emptyset$ $\leftarrow$ gepackte Objekte
4 **for** $u := 1$ **to** n **do**
5 **if** $W + w(u) \leq B$ **then**
6 $A \leftarrow A \cup \{u\}$
7 $W \leftarrow W + w(u)$
8 **return** A

Greedy Algorithmus

Greedy Ansatz

Wähle nacheinander Objekte mit maximalem Nutzen-pro-Gewicht Verhältnis, d.h. $\arg\max_{u \in U} \frac{p(u)}{w(u)}$.

1 Sortiere Objekte, so dass $\frac{p(1)}{w(1)} \geq \frac{p(2)}{w(2)} \geq \cdots \geq \frac{p(n)}{w(n)}$.
2 $W := 0$ $\leftarrow$ aktueller Füllstand (Gewicht)
3 $A := \emptyset$ $\leftarrow$ gepackte Objekte
4 **for** $u := 1$ **to** n **do**
5 **if** $W + w(u) \leq B$ **then**
6 $A \leftarrow A \cup \{u\}$
7 $W \leftarrow W + w(u)$
8 **return** A

Offensichtlich: Greedy gibt zulässige Lösung in $\mathcal{O}(n \log n)$.

Greedy Algorithmus

Greedy Ansatz

Wähle nacheinander Objekte mit maximalem Nutzen-pro-Gewicht Verhältnis, d.h. $\arg\max_{u \in U} \frac{p(u)}{w(u)}$.

1 Sortiere Objekte, so dass $\frac{p(1)}{w(1)} \geq \frac{p(2)}{w(2)} \geq \cdots \geq \frac{p(n)}{w(n)}$.
2 $W := 0$ ← aktueller Füllstand (Gewicht)
3 $A := \emptyset$ ← gepackte Objekte
4 **for** $u := 1$ **to** n **do**
5 **if** $W + w(u) \leq B$ **then**
6 $A \leftarrow A \cup \{u\}$
7 $W \leftarrow W + w(u)$
8 **return** A

Offensichtlich: Greedy gibt zulässige Lösung in $\mathcal{O}(n \log n)$.

Ist das optimal?

Greedy Algorithmus

Greedy Ansatz

Wähle nacheinander Objekte mit maximalem Nutzen-pro-Gewicht Verhältnis, d.h. $\arg\max_{u \in U} \frac{p(u)}{w(u)}$.

1 Sortiere Objekte, so dass $\frac{p(1)}{w(1)} \geq \frac{p(2)}{w(2)} \geq \cdots \geq \frac{p(n)}{w(n)}$.
2 $W := 0$ ← aktueller Füllstand (Gewicht)
3 $A := \emptyset$ ← gepackte Objekte
4 **for** $u := 1$ **to** n **do**
5 **if** $W + w(u) \leq B$ **then**
6 $A \leftarrow A \cup \{u\}$
7 $W \leftarrow W + w(u)$
8 **return** A

Offensichtlich: Greedy gibt zulässige Lösung in $\mathcal{O}(n \log n)$.

Ist das optimal? → Nein!

Greedy für Fraktionales Rucksackproblem

Fraktionales Rucksackproblem: wie bisher, aber Objekte dürfen zu einem beliebigen Anteil gepackt werden und man erhält den anteiligen Nutzen (“fractional profit”, Sand packen).

Greedy für Fraktionales Rucksackproblem

Fraktionales Rucksackproblem: wie bisher, aber Objekte dürfen zu einem beliebigen Anteil gepackt werden und man erhält den anteiligen Nutzen ("fractional profit", Sand packen).

Satz

Der Greedy Algorithmus, der das letzte Objekt evt. nur anteilig bis zur vollen Kapazität packt, löst das fraktionale Rucksackproblem optimal.

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.
- Betrachte ersten Index k bei dem $a_k \neq b_k$;

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.
- Betrachte ersten Index k bei dem $a_k \neq b_k$;

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.
- Betrachte ersten Index k bei dem $a_k \neq b_k$; nach Greedy: $a_k > b_k$.
- Erhöhe b_k um kleines $\varepsilon > 0$ und senke b_{k+1} entsprechend ab:

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.
- Betrachte ersten Index k bei dem $a_k \neq b_k$; nach Greedy: $a_k > b_k$.
- Erhöhe b_k um kleines $\varepsilon > 0$ und senke b_{k+1} entsprechend ab:

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.
- Betrachte ersten Index k bei dem $a_k \neq b_k$; nach Greedy: $a_k > b_k$.
- Erhöhe b_k um kleines $\varepsilon > 0$ und senke b_{k+1} entsprechend ab:

  OPT$' = \{b'_1, \ldots, b'_n\}$ mit $b'_i = b_i$ für alle $i \notin \{k, k+1\}$ und $b'_k = b_k + \varepsilon$ und $b'_{k+1} = b_{k+1} - \varepsilon \cdot \frac{w(k)}{w(k+1)}$
- Das Gesamtgewicht ist gleich $\sum_{i=1}^{n} b'_i \cdot w(i) = \sum_{i=1}^{n} b_i \cdot w(i)$.

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.
- Betrachte ersten Index k bei dem $a_k \neq b_k$; nach Greedy: $a_k > b_k$.
- Erhöhe b_k um kleines $\varepsilon > 0$ und senke b_{k+1} entsprechend ab:

  OPT$' = \{b'_1, \ldots, b'_n\}$ mit $b'_i = b_i$ für alle $i \notin \{k, k+1\}$ und $b'_k = b_k + \varepsilon$ und $b'_{k+1} = b_{k+1} - \varepsilon \cdot \frac{w(k)}{w(k+1)}$
- Das Gesamtgewicht ist gleich $\sum_{i=1}^{n} b'_i \cdot w(i) = \sum_{i=1}^{n} b_i \cdot w(i)$.
- Aber der Nutzen erhöht sich um $+\varepsilon \cdot p(k) - \varepsilon \cdot \frac{w(k)}{w(k+1)} \cdot p(k+1)$

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.
- Betrachte ersten Index k bei dem $a_k \neq b_k$; nach Greedy: $a_k > b_k$.
- Erhöhe b_k um kleines $\varepsilon > 0$ und senke b_{k+1} entsprechend ab:

  OPT$' = \{b'_1, \ldots, b'_n\}$ mit $b'_i = b_i$ für alle $i \notin \{k, k+1\}$ und $b'_k = b_k + \varepsilon$ und $b'_{k+1} = b_{k+1} - \varepsilon \cdot \frac{w(k)}{w(k+1)}$
- Das Gesamtgewicht ist gleich $\sum_{i=1}^{n} b'_i \cdot w(i) = \sum_{i=1}^{n} b_i \cdot w(i)$.
- Aber der Nutzen erhöht sich um $+\varepsilon \cdot p(k) - \varepsilon \cdot \frac{w(k)}{w(k+1)} \cdot p(k+1)$

Greedy ist fraktional optimal

Beweis.

- ALG $= \{a_1, a_2, \ldots, a_n\}$ mit a_i Anteil von Objekt i in ALG Lösung
- OPT $= \{b_1, b_2, \ldots, b_n\}$ mit b_i Anteil von Objekt i in OPT Lösung
- Indizes nach $\frac{p(i)}{w(i)}$ absteigend; alle Quotienten seien verschieden.
- Betrachte ersten Index k bei dem $a_k \neq b_k$; nach Greedy: $a_k > b_k$.
- Erhöhe b_k um kleines $\varepsilon > 0$ und senke b_{k+1} entsprechend ab:

  OPT$' = \{b'_1, \ldots, b'_n\}$ mit $b'_i = b_i$ für alle $i \notin \{k, k+1\}$ und $b'_k = b_k + \varepsilon$ und $b'_{k+1} = b_{k+1} - \varepsilon \cdot \frac{w(k)}{w(k+1)}$
- Das Gesamtgewicht ist gleich $\sum_{i=1}^{n} b'_i \cdot w(i) = \sum_{i=1}^{n} b_i \cdot w(i)$.
- Aber der Nutzen erhöht sich um $+\varepsilon \cdot p(k) - \varepsilon \cdot \frac{w(k)}{w(k+1)} \cdot p(k+1)$ > 0, wegen $\frac{p(k)}{w(k)} > \frac{p(k+1)}{w(k+1)}$.
- Widerspruch zur Optimalität von OPT. □

Zusammenfassung

Greedy Algorithmen für ein Optimierungsproblem

Zusammenfassung

Greedy Algorithmen für ein Optimierungsproblem

- Greedy Prinzip: in jedem Schritt wird ein lokales Optimum gewählt. (Oft gibt es mehrere Möglichkeiten für die Auswahl.)

Zusammenfassung

Greedy Algorithmen für ein Optimierungsproblem

- Greedy Prinzip: in jedem Schritt wird ein lokales Optimum gewählt. (Oft gibt es mehrere Möglichkeiten für die Auswahl.)
- Getroffene Entscheidungen werden nicht mehr zurückgenommen.

Zusammenfassung

Greedy Algorithmen für ein Optimierungsproblem

- Greedy Prinzip: in jedem Schritt wird ein lokales Optimum gewählt. (Oft gibt es mehrere Möglichkeiten für die Auswahl.)
- Getroffene Entscheidungen werden nicht mehr zurückgenommen.

Eigenschaften von Greedy Algorithmen:

- Greedy Algorithmen sind oft schnell,

Zusammenfassung

Greedy Algorithmen für ein Optimierungsproblem

- Greedy Prinzip: in jedem Schritt wird ein lokales Optimum gewählt. (Oft gibt es mehrere Möglichkeiten für die Auswahl.)
- Getroffene Entscheidungen werden nicht mehr zurückgenommen.

Eigenschaften von Greedy Algorithmen:

- Greedy Algorithmen sind oft schnell,
- finden manchmal aber nicht immer eine optimale Lösung,

Zusammenfassung

Greedy Algorithmen für ein Optimierungsproblem

- Greedy Prinzip: in jedem Schritt wird ein lokales Optimum gewählt. (Oft gibt es mehrere Möglichkeiten für die Auswahl.)
- Getroffene Entscheidungen werden nicht mehr zurückgenommen.

Eigenschaften von Greedy Algorithmen:

- Greedy Algorithmen sind oft schnell,
- finden manchmal aber nicht immer eine optimale Lösung,
- problemspezifische Analyse zeigt die Qualität des Greedy Ansatzes.